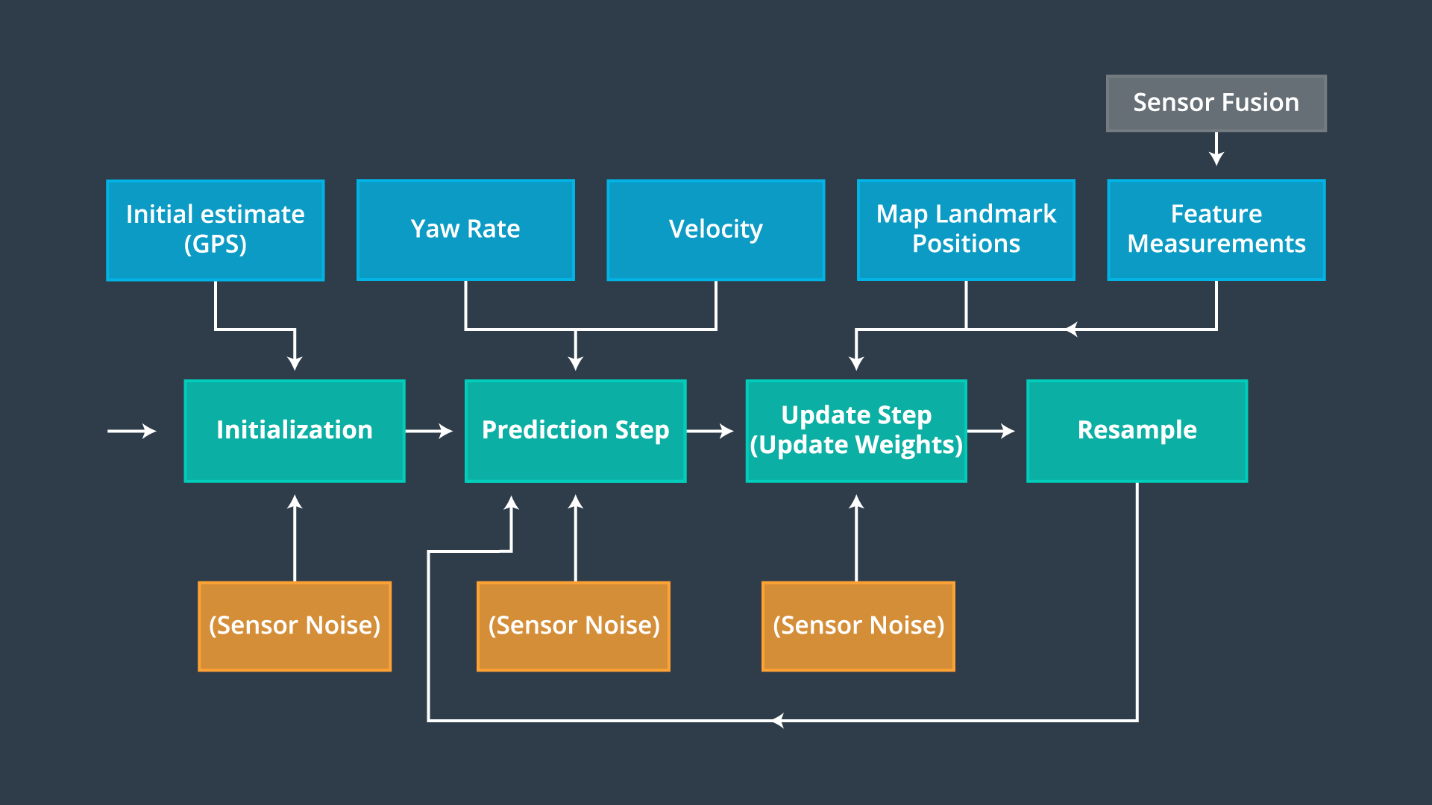
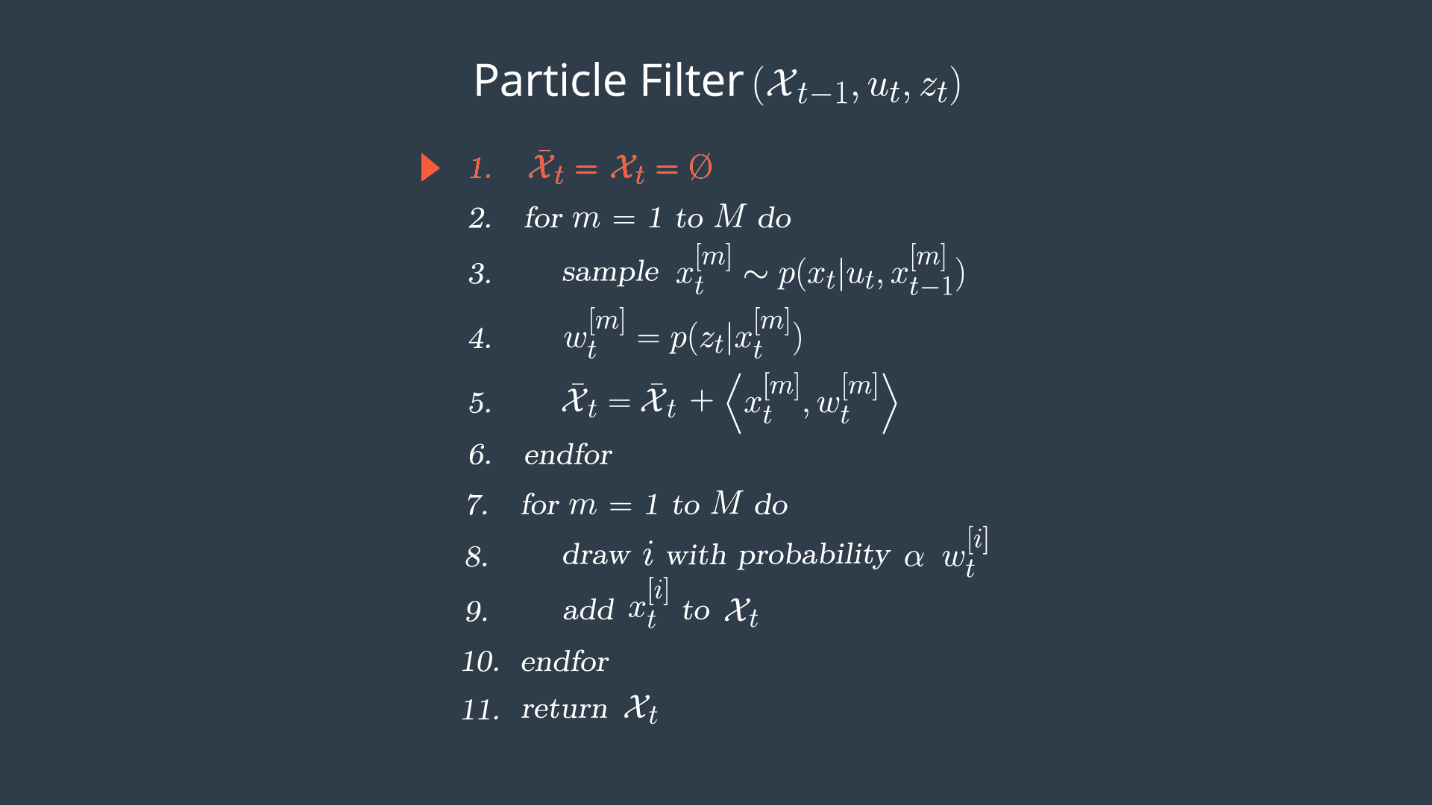
**Implementation of a Particle Filter**





**Step 1: Initialization**

The most practical way to initialize our particles and generate real time output, is to make an initial estimate using GPS input. We also need to account for the GPS sensors noise as well.

We will implement particles by sampling a Gaussian distribution taking into account Gaussian sensor noise around the initial GPS position and heading estimates.

How many particles shall we use?

Theoretically, the particle filter will exactly represent the Bayesian posterior distribution as the number of particles approaches infinity. We also don’t want to have too few particles. If we do, we won’t have enough to cover all the high likelihood positions.

Too many particles will result in a slower algorithm.

**Two ways to initialize particles:**

1. Sample the state space uniformly. We could always separate our state space into a grid, and have 1 particle per cell. This approach isn’t practical when the state space is large, like it is for self-driving cars. The state space we have is essentially the entire landmass of the earth.
2. The more practical way is to sample around an initial estimate, like GPS. We can’t use GPS for the entire localization process due to its inaccuracy, and unavailability in certain environments.

GPS makes a great initialization.

So what we will do is take the GPS position, and the sensors noise values, and sample around the Gaussian to make our particles.

We will use the C++ standard libraries normal distribution and random engine functions to sample our particles around the GPS value.

**Step 2: Prediction Step**

Now that we have particles, we need to predict the vehicles position. We will use the motion model of the vehicle, a bicycle model, to predict where the vehicle will be at the next time step by updating based on yaw rate and velocity while accounting for sensor noise.

We will update the location of every particle.

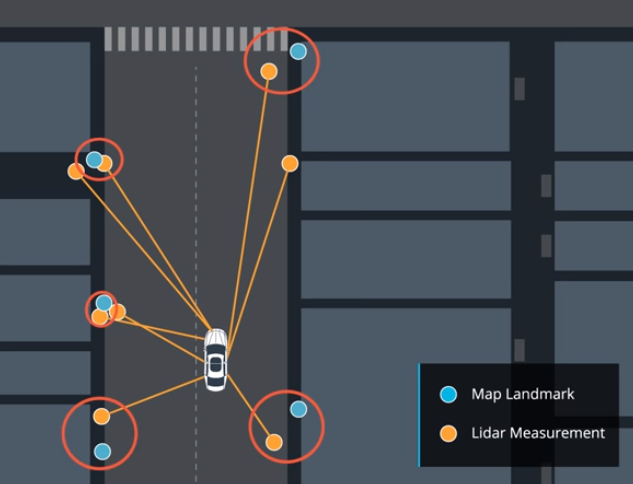
|  |
| --- |
|  |
|  |

Before we can move on to the Update Step, we need to take care of the Data Association Problem.

Data association is the problem of matching landmark measurements to objects in the real world like map landmarks.

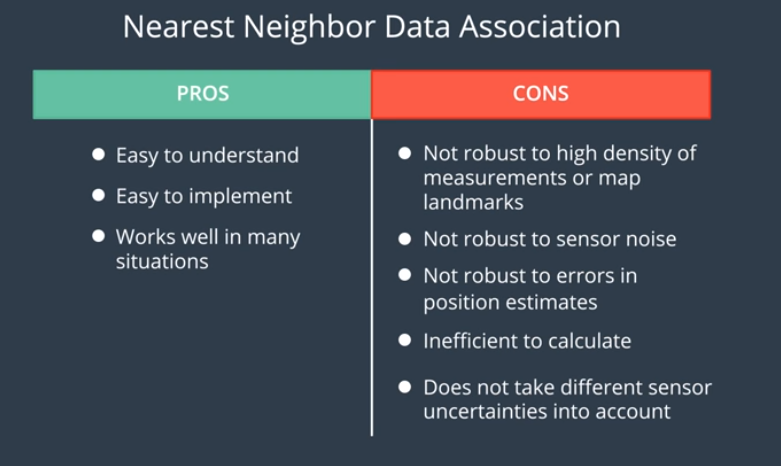
Sometimes we will have multiple LIDAR measurements representing the same Landmark. How do we know which detection to choose? We will solve this issue in the easiest way we can with **Nearest Neighbour**.

We will just choose the closest detection as the actual landmark.

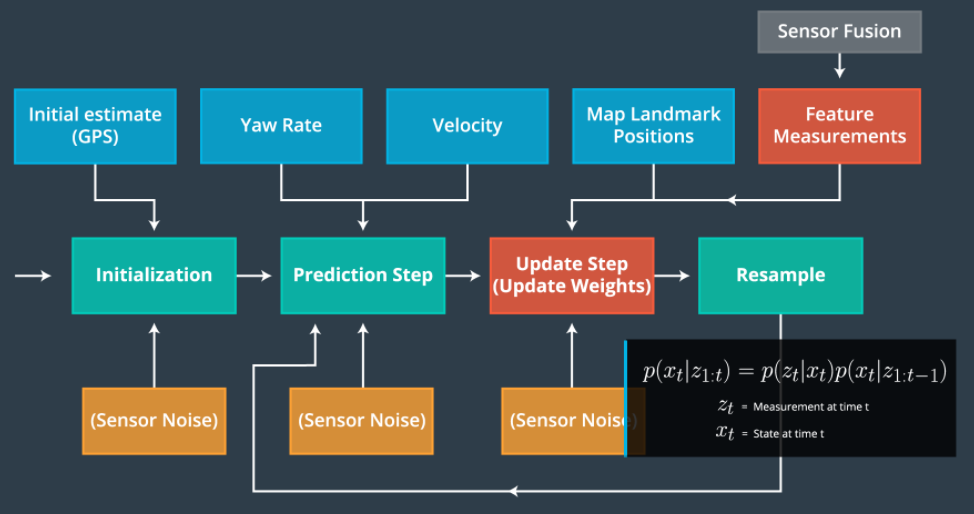


Some problems with Nearest Neighbour are:

1. If there is a high density of Landmarks, we won’t be able to distinguish them.
2. We require a High Signal-to-Noise ratio for sensors.
3. We need a very accurate motion model.
4. It is not very efficient due to the fact that for every Map Landmark, we need to search through all of the measurements in order to find the smallest distance one. Complexity of O(m\*n)
5. Nearest Neighbour isn’t robust, not does it account for the sensors noise/uncertainty.

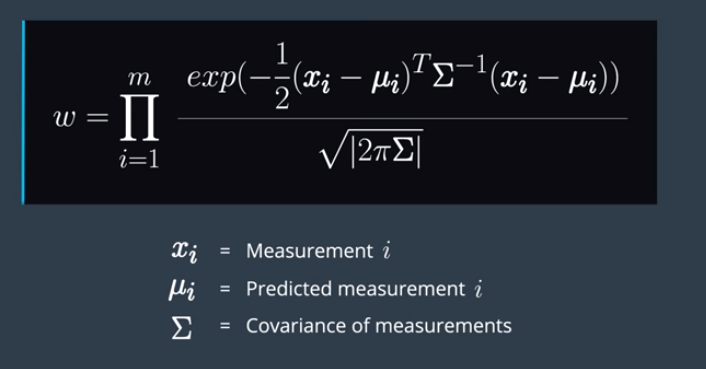


**Step 3: Update Step**



We must update particle weights based on LIDAR and RADAR readings of landmarks. By calculating particle weights and using the Resampling Wheel.

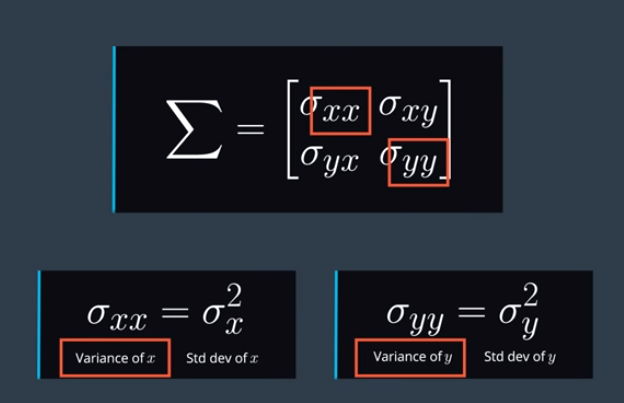
Landmark measurements are used to update the weight of each particle.



We can use the Multivariate Gaussian Probability Density Function to update the weights of each particle. The product of all the measurements gives us the likelihood of each particle given the sensors have Gaussian noise, we use the current predicted state of the car. We also assume that each landmark measurement is independent. This formula allows a recursive structure for updating weights of each particle.

m is the total number of measurements for each particle.

The covariance matrix, sigma, is a symmetric square matrix, that contains the variance or uncertainty of each variable in the sensor measurement, as well as the correlation between these variables.

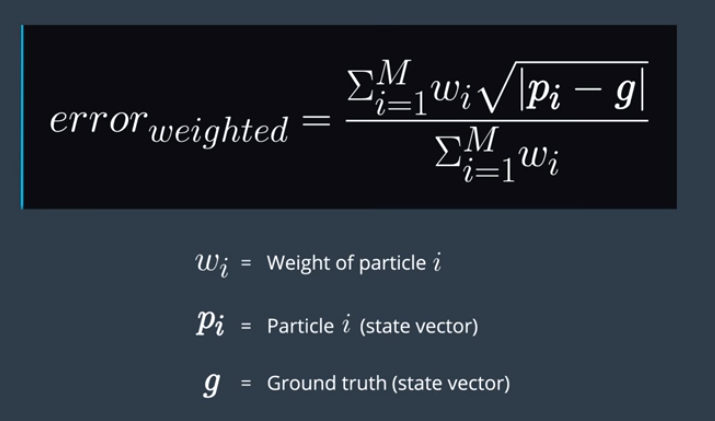


The off diagonal terms are correlation values. While the diagonal values are the reliability of the x and y measurements.

After we update our weights, we move on to the resampling step, where we resample our particles based on the calculated weights.

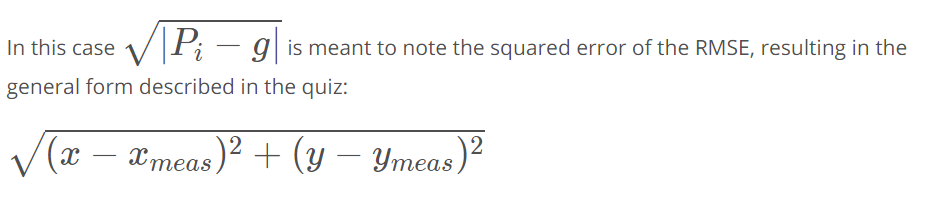
**Calculating Error**

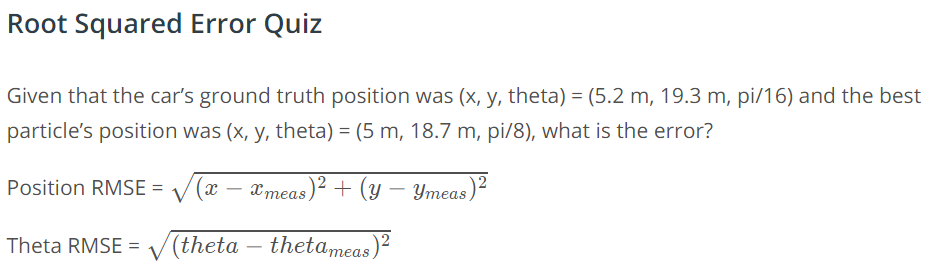
We can show the error in our localization from the ground truth value in a couple of ways.

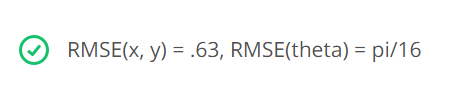


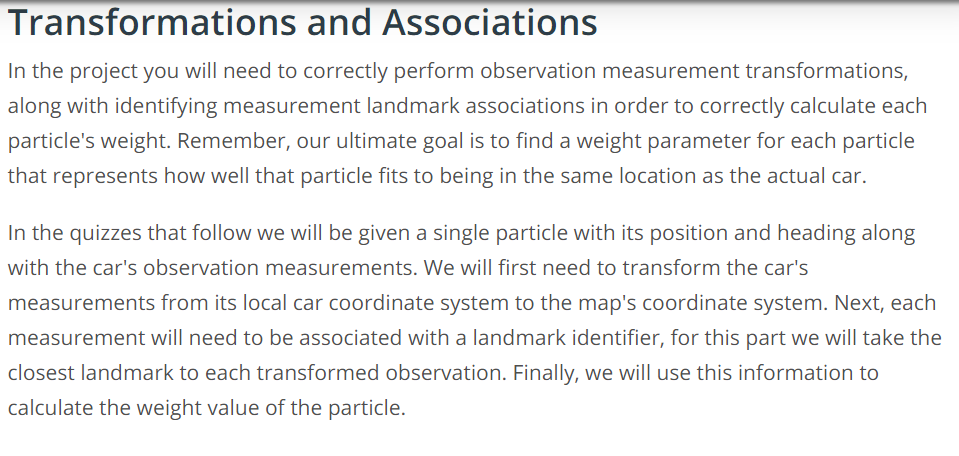
One way is to take the weighted average error of every particle.

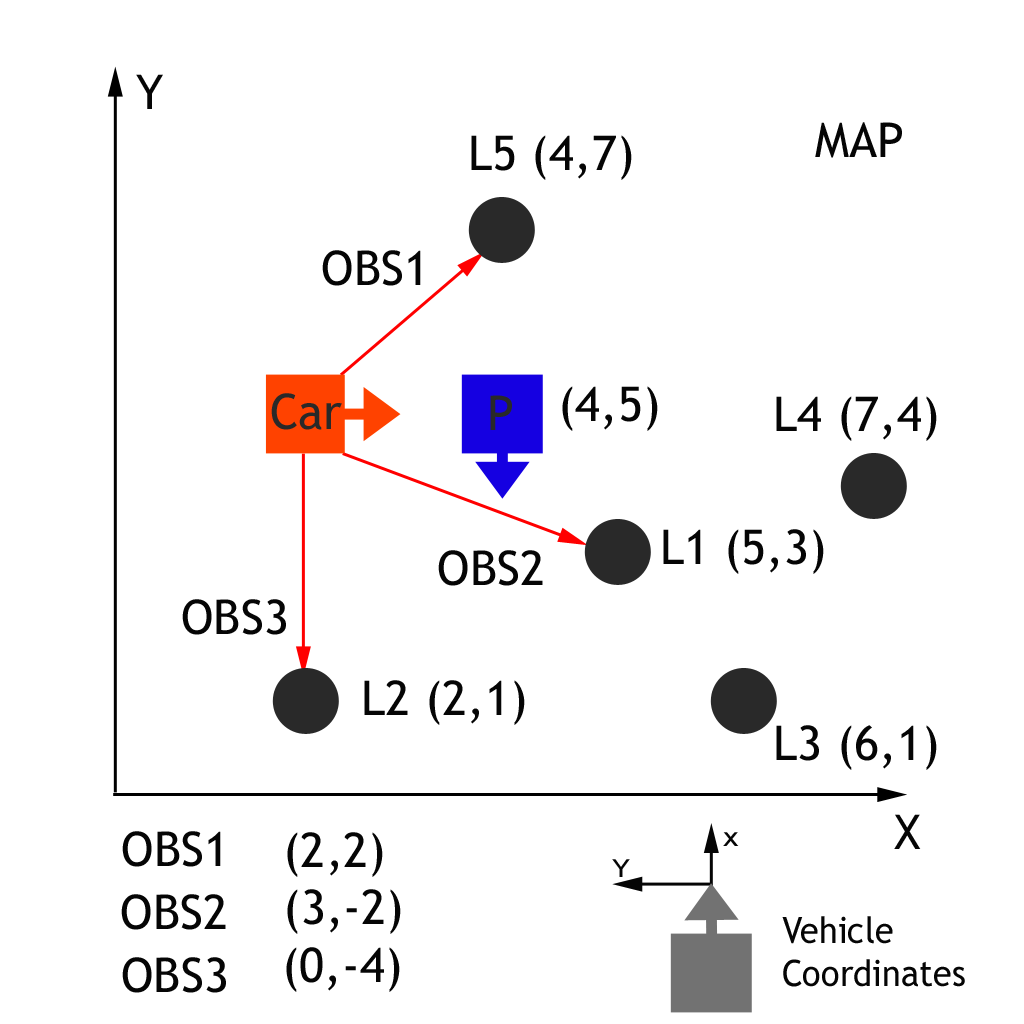
Another way is to just look at the best/highest weighted particle, and look at its root squared error.

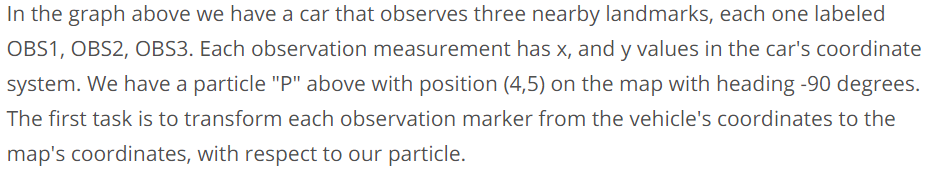












We need to map any observation measurement (x,y) with the transformed observation (X,Y) for that particle. That way we can transform particle co-ordinates to map coordinates with a function.

