Kidnapped Vehicle Project

The goals/steps of this project are the following:

- In this project, you will implement a 2-dimensional particle filter in C++. Your particle filter will be given a map and some initial localization information (analogous to what a GPS would provide). At each time step your filter will also get observation and control data.
- The project should satisfy the project rubric.

Files Submitted & Code Quality

My project includes the following files:

- main.cpp containing the script to read the map data and ground truth data. The main.cpp runs the particle filter and calculates the weighted error at each time step.
- particle_filter.cpp which contains the implementation of the prediction and update steps of the particle filter.

Implementation approach

Initialization

- The initialization step of the particle filter is implemented in the **ParticleFilter::init()** function from lines 27-61.
- First, the number of particles is set. If the number of particles is too small, the car will not localize correctly and if the number is too large, it will slow down the filter and real-time localization may not be possible. The number of particles needs to for the project was set to 100.
- The initial location and the heading of all the particles are set using the GPS data provided as input. To account for GPS measurement uncertainty, Gaussian noise is added to the position estimate.
- For initialization, all the particles are assigned a weight of 1.

Prediction

- The prediction step of the particle filter is implemented in the **ParticleFilter::prediction()** function from lines 63-101.
- The prediction state uses the vehicle velocity and yaw rate is input to predict the location and heading of each particle.
- The prediction is done using the equations from the bicycle motion model.
- The bicycle model equations used are shown below:

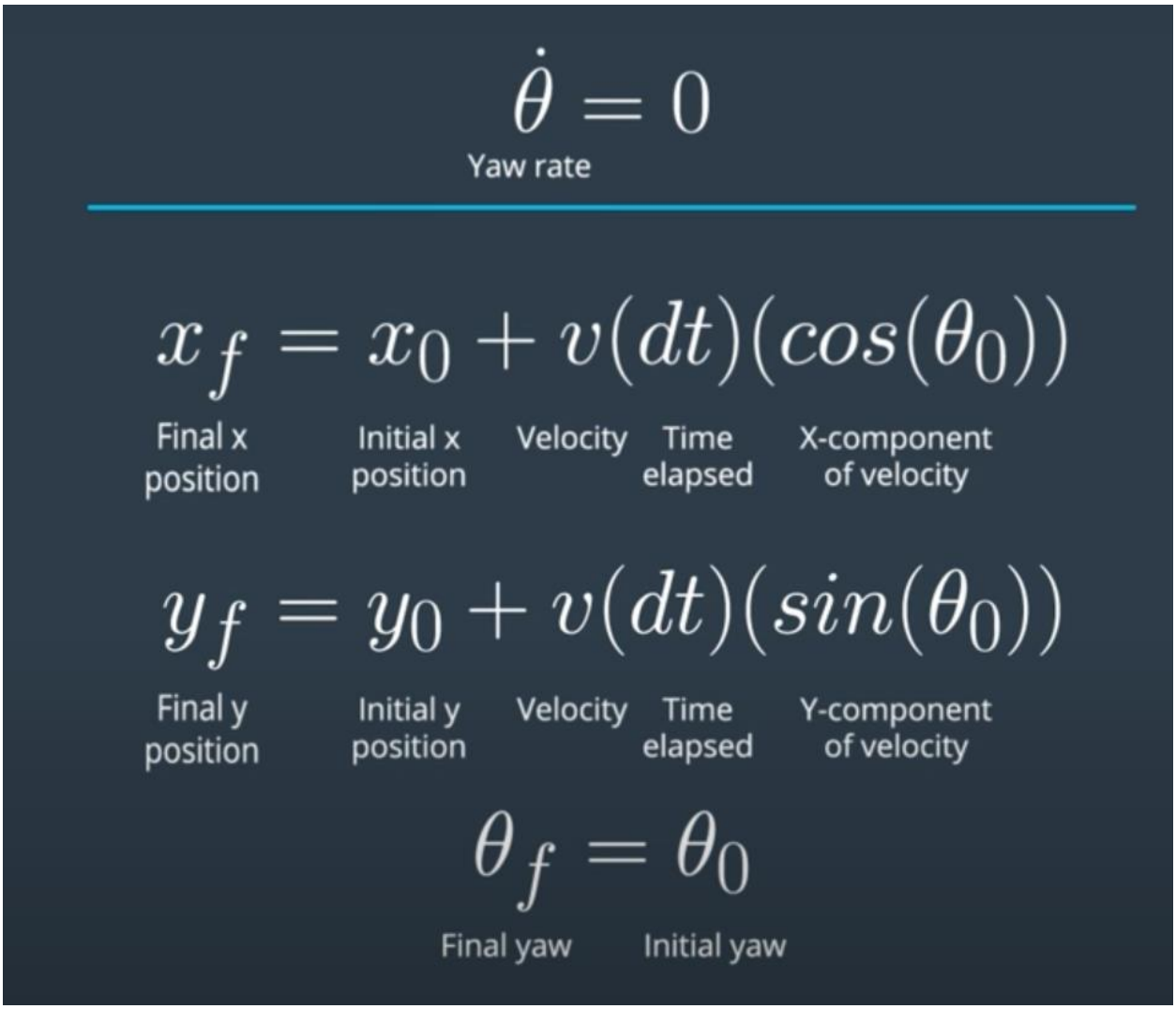

Fig 1. Equations used for zero yaw rate

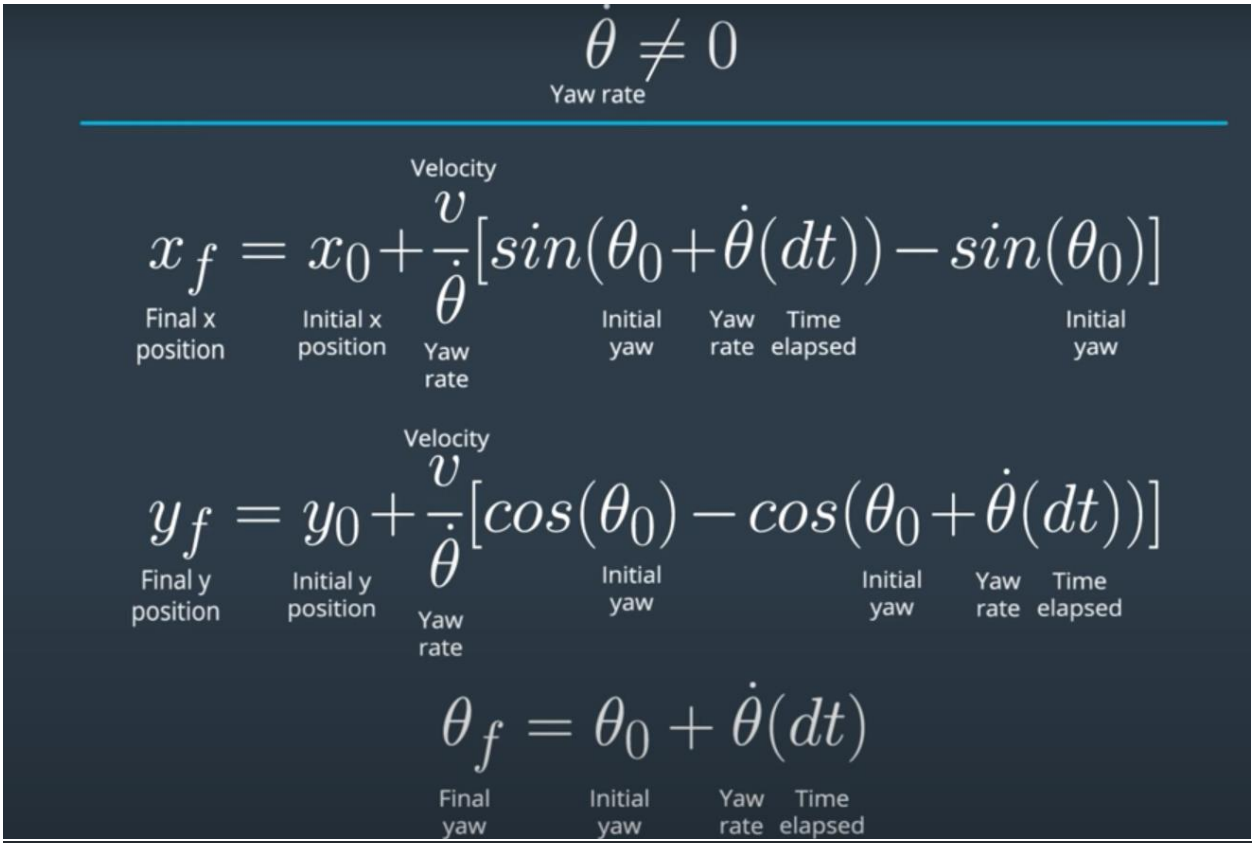

Fig 2. Equations used for varying yaw rate

- To account for velocity and yaw rate measurement uncertainty, Gaussian noise is added to the predictions using the measurement standard deviations provided as input.

Data Association

- The data association step of the particle filter is implemented in the **ParticleFilter::dataAssociation()** function from lines 102-126.
- For each observation (which has been transformed from vehicle co-ordinates to map co-ordinates), landmarks on the map are associated with it based on the predicted measurements for the landmarks. This is done using the nearest neighbor method, in which the landmark with the lowest Euclidean distance is associated with the observation.

Weights update

- The weights update step of the particle filter is implemented in the **ParticleFilter::updateWeights()** function from lines 127-187.
- To update the weights of each particle, first, all the map landmarks that are within the sensor range are added to the predictions vector.
- Next, all the observations are transformed from the vehicle co-ordinate system to the map coordinate system.
- Using the predictions and transformed observations, the data association is done for each observation.
- After the data association is complete, the weight of each particle can be calculated is the multi-variate Gaussian distribution.
  The equation for calculating the same is shown below:

$$P(x,y) = \frac{1}{2\pi\sigma_x\sigma_y} e^{-\left(\frac{(x-\mu_x)^2}{2\sigma_x^2} + \frac{(y-\mu_y)^2}{2\sigma_y^2}\right)}$$

  Where, σ_x = standard deviation in x-direction

  σ_y = standard deviation in y-direction

  x = observation measurement in x-direction

  y = observation measurement in y-direction

  μ_x = x co-ordinate of landmark position

  μ_y = y co-ordinate of landmark position

- The weight of the particle calculated using the multi-variate Gaussian, will assign high probabilities to particles for which the given landmark positions are most likely for the current set of observations.

Resampling

- The weights update step of the particle filter is implemented in the **ParticleFilter::resample()** function from lines 189-215.
- The particles are resampled based on replacement with probability proportional to their weight.
- The resampling ensures particles with low weights are discarded and those with higher weights are more frequently picked.
- As a result, all the particles finally converge to the correct position of the car.
- The resampling was achieved using the discrete_distribution function in C++ to pick particles based on their weight.

Output

The screenshot below shows that the vehicle was successfully localized using the particle filter implementation. The blue circle shows the location as estimated by the particle filter.

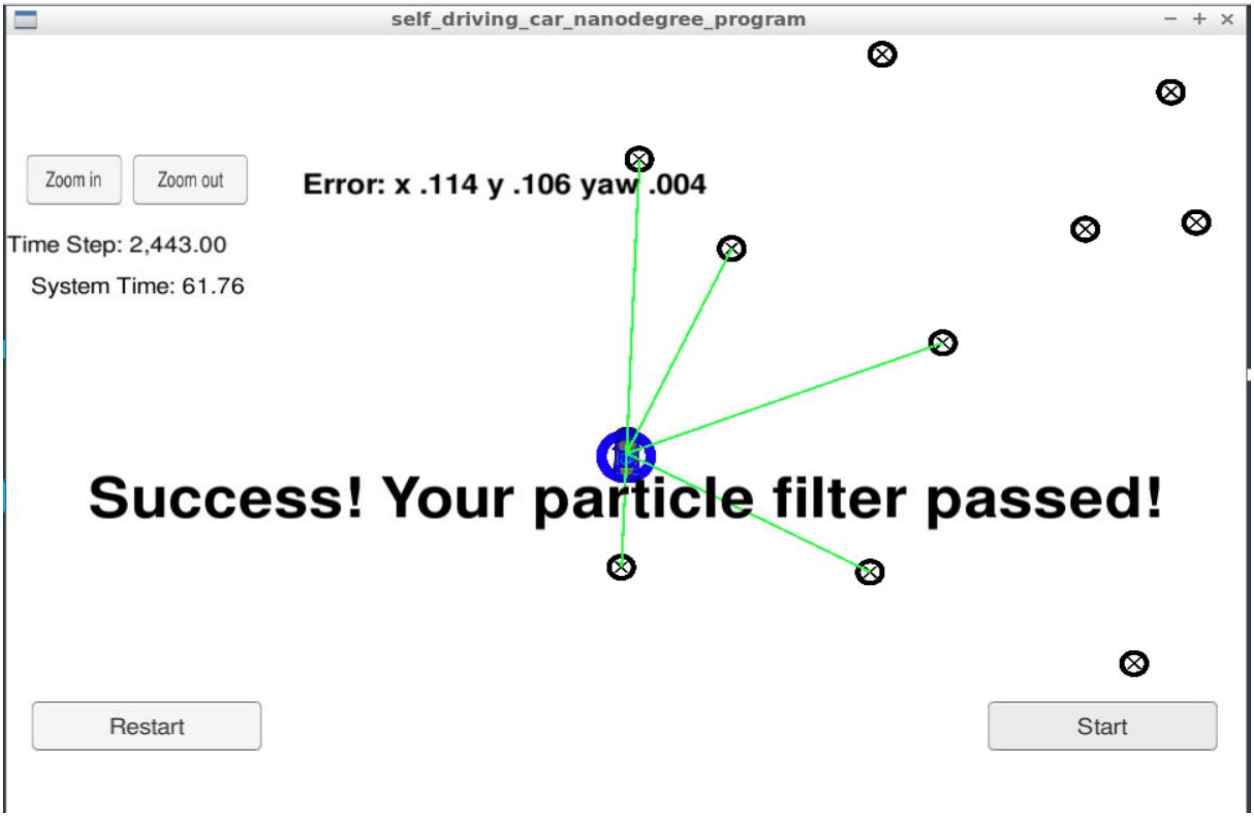

Fig 3. Output Visualization