Simulating Kalman Filter for Wheelchair Localization Using Simulated GPS and Wheel Encoders

Eldridge, Ethan
ENGR-E599
Indiana University
Bloomington, USA
etmeldr@iu.edu

***Abstract*—The Kalman filter is a powerful tool that can aid in the fusion of noisy signals into one well-informed signal output. This paper explores the inner workings of the Kalman filter by simulating the results of Garrett Leonard's 2014 paper, "Localization Using Kalman Filter with GPS and Wheel Encoders on a BCI Wheelchair."**

I. Introduction

The objective of this project is to simulate the results of a 2014 thesis project by Garrett Leonard at California State University, Northridge. In this project, Leonard used the Kalman filter, a powerful tool for signal processing, to enhance the postioning accuracy of the university's brain-computer interface (BCI) wheelchair. To achieve this, Leonard collected GPS and wheel encoder data from the wheelchair as it traveled down a series of predetermined paths. Leonard then created a Kalman filtering program that took this position information and filtered out noise to give a more accurate reading of the wheelchair's true location.

II. Background and Related Work

This project attempts to simulate the results of Garrett Leonard's 2014 thesis, "Localization Using Kalman Filter with GPS and Wheel Encoders on a BCI Wheelchair." The paper is heavily referenced in this report, and and the system dynamics models and experimental methods implemented here are taken from the paper.

III. Methods

A. Ground Truth Path Generation

Given that the scope of this project was limited to a simulation of Leonard's results, actual GPS and wheel encoder information could not be obtained. Instead, the positioning data was synthesized using a random walk algorithm that used Leonard's proposed system dynamics to simulate a ground truth path for the wheelchair. This algorithm started the virtual wheelchair out at (x=0,y=0) and, for a given number of steps, generated wheel encoder speeds at each step. These encoder speeds were then plugged into the system dynamics equation in order to obtain a ground-truth x, y, and θ value for the virtual wheelchair at each step.

The system dynamics for the wheelchair are as follows:

$$\begin{bmatrix} x(t+1) \\ y(t+1) \\ \theta(t+1) \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x(t) \\ y(t) \\ \theta(t) \end{bmatrix}$$

$$+ \begin{bmatrix} \frac{1}{2}\cos(\theta) * \Delta t & \frac{1}{2}\cos(\theta) * \Delta t \\ \frac{1}{2}\sin(\theta) * \Delta t & \frac{1}{2}\sin(\theta) * \Delta t \\ \frac{1}{0.6985}\Delta t & -\frac{1}{0.6985}\Delta t \end{bmatrix} \begin{bmatrix} V_R(t) \\ V_L(t) \end{bmatrix}$$

where $x(t)$ is the wheelchair's x-position at time t, $y(t)$ is the y-position, $\theta(t)$ is the orientation of the wheelchair relative to the x-axis, $V_R(t)$ is the right wheel's rotation speed in rad/s, and $V_L(t)$ is the left wheel's rotation speed. This dynamics model takes into account specific dimensions of the BCI wheelchair, such as the radius of each wheel, to derive an accurate measurement of the wheelchair's future state at time $t+1$. The dynamics model is derived from the more general Kalman filter state projection equation:

$$\hat{x}_{t+1}^{-} = A\hat{x}_t + Bu_t$$

where $\hat{x}_{t+1}^{-}$ is the system's state at time $t+1$, $\hat{x}_t$ is the system's state at time t, and u_t is the input value to the system at time t. A and B are matrices specific to a given system dynamics model that transform the system input and former state of the system into a usable format for the projection equation.

These system dynamics, as well as the A and B matrices for this dynamics model, were implemented in the algorithm $dynamics(V_L, V_R, \Delta t, lastEst)$, where the input Δt is the change in time between estimates in seconds (defined consistently in this project as 1s), and $lastEst$ is a 1x3 matrix that contains the prior state of the system at time t. The output for $dynamics()$ is the new projected state of the system, $[x(t+1), y(t+1), \theta(t+1)]$ at time $t+1$.

After the system dynamics had been implemented, it was possible to begin simulating the ground-truth path of the virtual wheelchair. The algorithm $simPath(steps)$ takes in an integer number of steps as input. For each step, simulating the passing of Δt seconds in the wheelchair's trajectory, the algorithm decides the direction of the wheelchair's next movement by choosing one of three weighted directions: forward (F), right (R), or left (L). These directions are weighted at 0.95, 0.025, and 0.025, respectively, in order to simulate a more realistic path for the wheelchair than a purely random walk would. Although the filtering algorithm would still function on such a path, a spiraling and erratic trajectory would be much less common for a wheelchair to take than a more continuous path with only a handful of major turns.

Another feature adding to the realism of the path simulation is implementing adaptive changes to each direction's weight following decisions to turn. In the real world, once a vehicle initiates a turn, it usually continues to turn for several seconds more in order to orient itself in the direction it wants to turn. This cannot happen in the simulation with static weights, however, as the "L" and "R" weights are very low, making it improbable that the same turn would be selected again at the next step. This means that, when the "L" direction is selected for example, that the virtual wheelchair will turn only slightly to the left, and that the wheelchair will still be traveling in the general "forward" direction. It follows that in the entire trajectory of the wheelchair, only small changes to the orientation of the wheelchair would be made, and that no major turns would occur in the simulation. This is not the desired behavior for the simulation, as we would instead like to see the wheelchair make full 45º-90+º turns from time to time to simulate a normal indoor or outdoor environment. To address this problem, at step s, the algorithm keeps track of the decision made at step $s-1$. If the last decision was "L" or "R," that decision is weighted significantly more. Although "F" is still the most probable option, this strategy allows the simulation to recreate more realistic turns and behavior across all steps.

Upon choosing a direction, the $simPath()$ algorithm then sets ground-truth encoder values for that step accordingly. If the choice was "L," for example, the right wheel would be traveling faster than the left wheel, so the encoder values reflect the difference in wheel speed. These values are then input into the $dynamics()$ algorithm to get ground truth x, y, and θ values for that step. The θ value is then adjusted to reflect the orientation of the system dynamics model relative to the x-axis, and the ground truth state estimate and encoder values are finally recorded for the step. The algorithm then repeats this process for the remaining number of steps, recording the ground truth state estimates and encoder values at each step.

$simPath()$ outputs two lists: $positions$ and $encoders$. These two lists represent the ground-truth (x, y) positions and (L, R) encoder values, respectively, of the simulated wheelchair along the generated path. Each call of this function produces a different stochastically generated path. The $STEPS$ constant can be changed to generate longer or shorter paths.

B. Signal Data Generation

After the ground truth path had been generated for the virtual wheelchair, the positioning data needed to be generated for use in the filter. This was done using a small noise-adding algorithm, $add\_noise(vals, stdev)$ which simply takes in a one-dimensional list of values and adds Gaussian noise to each value. The amount of noise added can be controlled via the $stdev$ parameter, which represets the standard deviation of the Gaussian distribution that the noise is drawn from. Using this function, the program is able to take the ground truth position and encoder outputs from $simPath()$ and add Gaussian noise to them to represent the noisy GPS and wheel encoder values that the filter will be receiving. Noise for each sensor can be specified using the GPS_NOISE and ENC_NOISE constants.

With the noisy GPS data and wheel encoder values recreated, visualizations could be made to compare the signal data to the ground truth. The noisy GPS data could be plugged directly into the visualization given that it was already in (x,y) position format. The noisy wheel encoder information, however, had to be ran back through the system dynamics model in order to calculate new (x,y) positions corresponding to the noisy encoder values at each step in the path.

C. Kalman Filtering Simulation

Once all of the proper data had been generated, it was finally possible to implement the Kalman filter simulation itself. The implementation is relatively straightforward and deviates only slightly from the normal Kalman filter equation flow:

1) Get initial estimates for system state ($\hat{x}_t$) and error covariance matrix (P_t).
2) State projection:
$$\hat{x}^-_{t+1} = A\hat{x}_t + Bu_t$$
3) Error covariance projection:
$$P^-_{t+1} = AP_tA^T + Q$$
4) Kalman gain computation:
$$K_{t+1} = P^-_{t+1}H^T(HP^-_{t+1}H^T + R)^{-1}$$
5) State projection update using measurement z_{t+1} (GPS data):
$$\hat{x}_{t+1} = \hat{x}^-_{t+1} + K_{t+1}(z_{t+1} - H\hat{x}^-_{t+1})$$
6) Error covariance update:
$$P_{t+1} = (I - K_{t+1}H)P^-_{t+1}$$
7) Repeat steps 2-6

The only modification to this order is an update to the value of θ during the state projection phase in order to align it with the dynamics model's orientation relative to the x-axis.

D. Kalman Filtering Simulation with Signal Obstruction

In addition to filtering data from a relatively signal-consistent wheelchair trajectory, Leonard's thesis also simulated a trajectory that passed through obstacles that would obstruct the wheelchair's GPS signal. This simulation was performed by adding additional Gaussian noise to the GPS data of certain sections of the ground truth path. A similar implementation of this simulation was created for this project, where each ”dead zone” along the path is highlighted in red.

IV. Results

The simulation provided a variety of observable results. For each ground truth path generated by the simulation, the following visualizations were created:

1) Simulated Ground Truth Path
2) Noisy Encoder Path vs Ground Truth Path
3) Noisy GPS Data vs Ground Truth Path
4) Final Simulation of Kalman Filter Along Path
5) Noisy GPS Data vs Partially Obstructed Ground Truth Path
6) Final Simulation of Kalman Filter Along Partially Obstructed Path

Four experiments were ran on the implementation in an attempt to derive the effects of different parameters on the filtering algorithm: one baseline run as a control result, one with increased measurement noise (higher GPS noise), one with increased process noise (higher encoder noise), and one with increased iterations (more steps). The visualizations for each of these experiments were compiled and can be found at the end of the paper.

V. Discussion

The results of the experiments were very helpful in understanding the significance of each feature in the Kalman filtering algorithm. Starting with the control group results (Fig. 1), we can see that the filter performs well with small amounts of measurement and process noise. Even in the dead zone simulation, the control group was able to follow the general direction of the ground truth path and was able to pick it back up almost immediately after leaving each dead zone.

Next, the increased measurement noise group (Fig. 2) revealed a great deal about the importance of the measurement noise assumptions made by the filter. Although the filter was able to withstand extremely noisy GPS data in the normal simulation, the exaggerated measurement noise in the dead zone simulation seemed to have thrown the filter off drastically, as can be seen in Fig. 2f. This is likely due to the fact that the measurement noise covariance matrix, R, was not initialized properly so as to handle this amount of noise. With proper investigation and testing, one would likely be able to converge on a measurement noise covariance matrix that could offer a better solution to this path.

The increased process noise group (Fig. 3) showed an equally interesting set of results. Because the noisy encoder path was so drastically different from the ground truth (Fig. 3b), the filter was unable to closely approximate the true path of the wheelchair in either simulation (normal and dead zone, Figs. 3d and 3f). When comparing the results of this experimental group to that of the increased measurement (IM) noise group, however, one can see that the filter path is affected differently in each case. In the dead zone simulation of the IM group (Fig. 2f), the filter began to turn back on itself erratically. The filter in this case completely loses any sense of the encoder path and its past movement trends. In the case of the increased process noise group, however, the error seems a bit more controlled. By this, I mean that the filter paths in both simulations (Figs. 3d and 3f) seem to at least follow the general shape of the ground truth path, though displaced by a distance approximate to the distance between the encoder path and the ground truth path (Fig. 3b).

Finally, the increased steps group (Fig. 4) seemed to show that with an increased number of filtering iterations, the filter is better able to determine which signals (measurement or process input) it should listen more closely to. The overall trends of the lines followed more closely to the ground truth than in any of the other experimental groups and were more robust to increased noise. This was particularly evident, in the dead zone simulation (Fig. 4f), where the filter was almost completely unfazed by the extra noise in the second dead zone. I hypothesize that this result is likely due to the extra steps that the filter had observed up to that point, allowing it to get a better understanding of the more accurate signal to listen to between the GPS and encoder data.

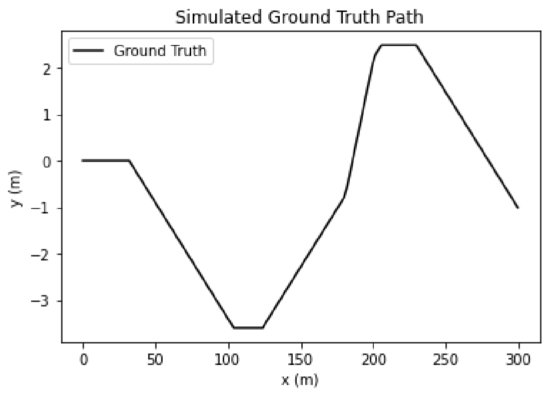
From the results of these experiments, it would be reasonable to conclude that for a Kalman filter to work properly, one should first choose a dynamics model with inputs that have low process noise. Then, one should take the time to properly initialize the measurement noise covariance matrix. Finally, a high number of iterations should be used (either implementing the filter for a longer period of time, or updating the filter at a higher frequency) in order to create the most robust filtering algorithm possible for a given application.

VI. Conclusion

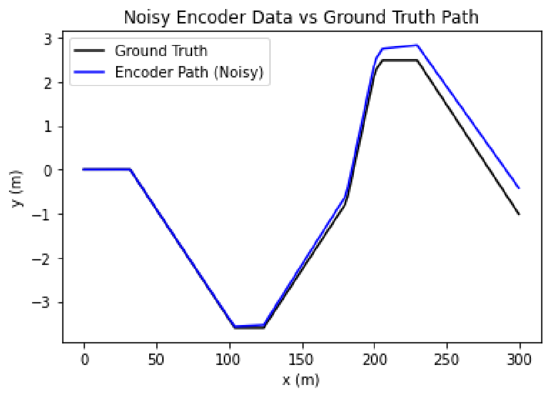
This project was a helpful experience in learning the inner workings of the Kalman filter as seen in Leonard’s 2014 thesis project. The path and positional data generation algorithms were sufficient in creating data that successfully demonstrated the effectiveness of the Kalman filter. Additionally, the experimental results shared an insightful look into the major factors affecting the performance of the filter.

References

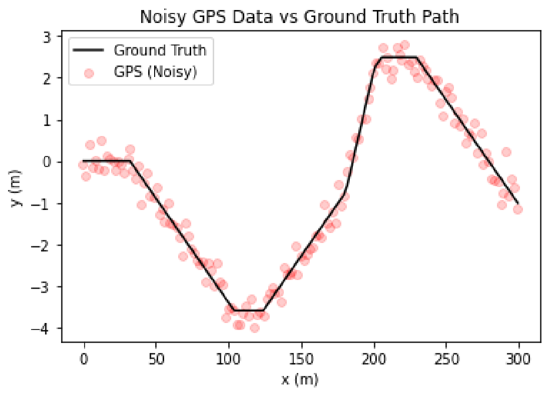
[1] G. Leonard, “Localization Using Kalman Filter with GPS and Wheel Encoders On A BCI Wheelchair,” Dec. 2014. [Online]. Available: https://scholarworks.csun.edu/bitstream/handle/10211.3/132976/Leonard-Garrett-thesis-2015.pdf?sequence=1 [Accessed: 17-Dec-2021]



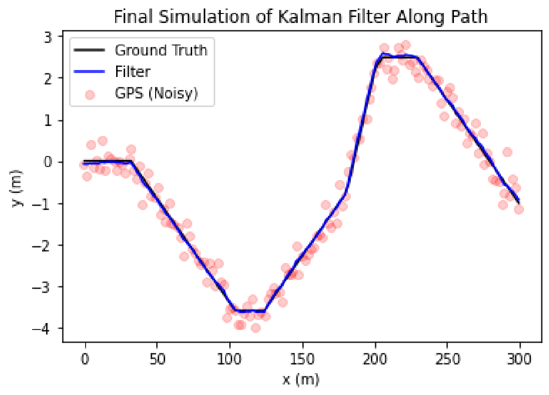
(a) Ground Truth



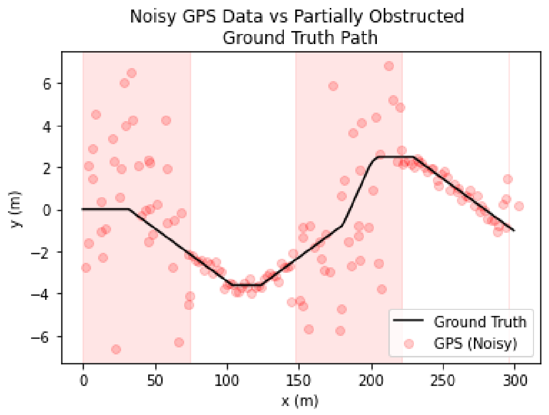
(b) Noisy Encoder Path vs GT



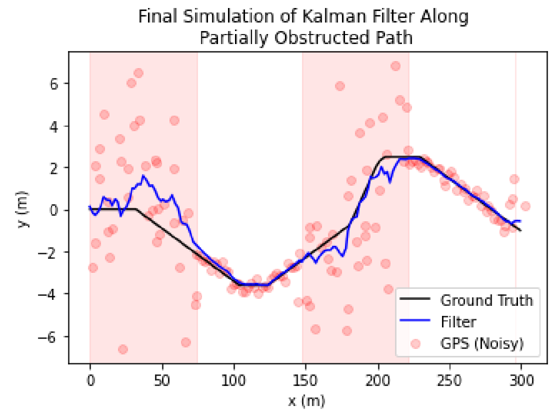
(c) Noisy GPS Data vs GT



(d) Normal Kalman Sim

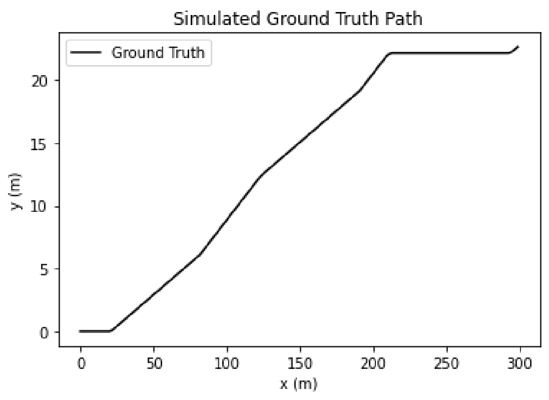


(e) Partially Obstructed GPS Data vs GT

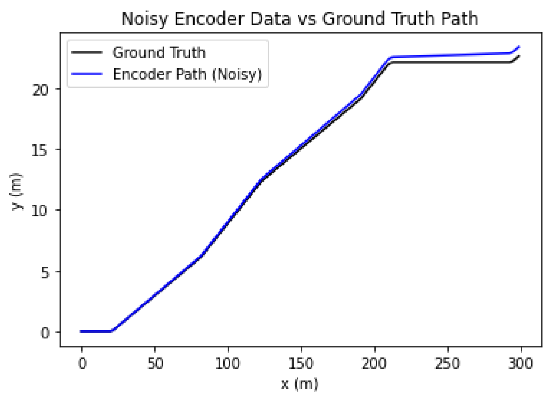


(f) Partially Obstructed Kalman Sim

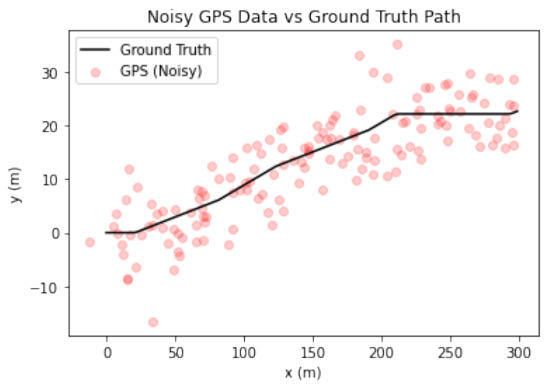
Fig. 1: Results from Control Group (STEPS=150, GPS_NOISE=0.25, ENC_NOISE=0.0065)



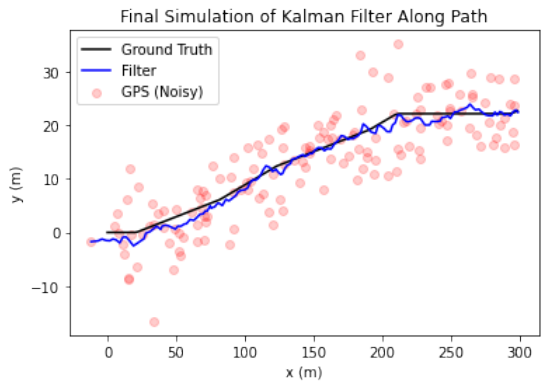
(a) Ground Truth



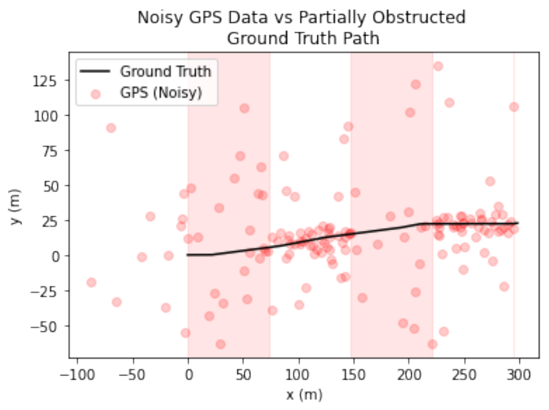
(b) Noisy Encoder Path vs GT



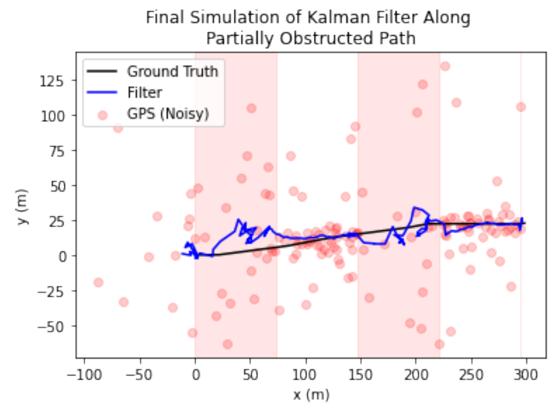
(c) Noisy GPS Data vs GT



(d) Normal Kalman Sim

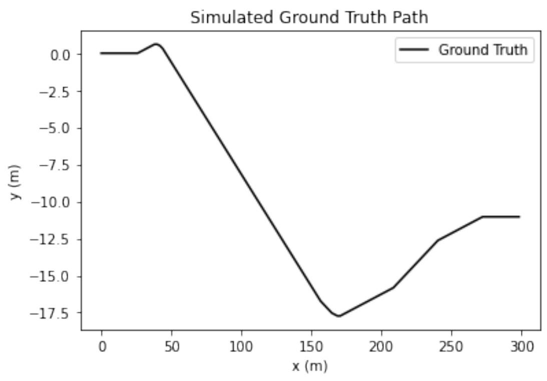


(e) Partially Obstructed GPS Data vs GT

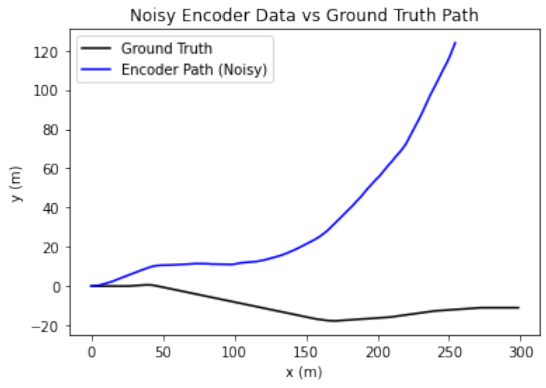


(f) Partially Obstructed Kalman Sim

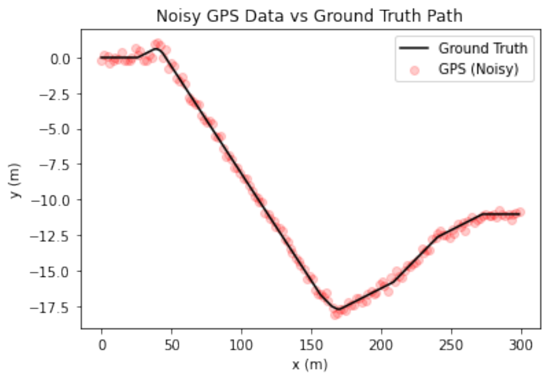
Fig. 2: Results from Increased Measurement Noise Group (STEPS=150, GPS_NOISE=0.5, ENC_NOISE=0.0065)



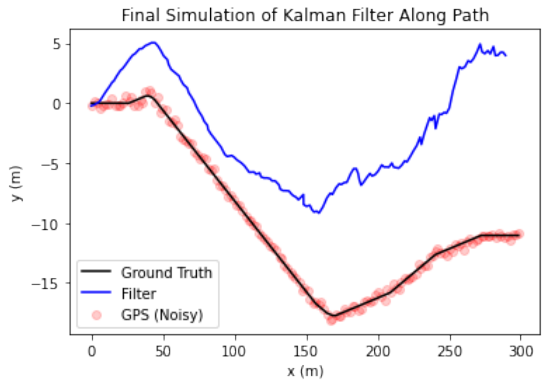
(a) Ground Truth



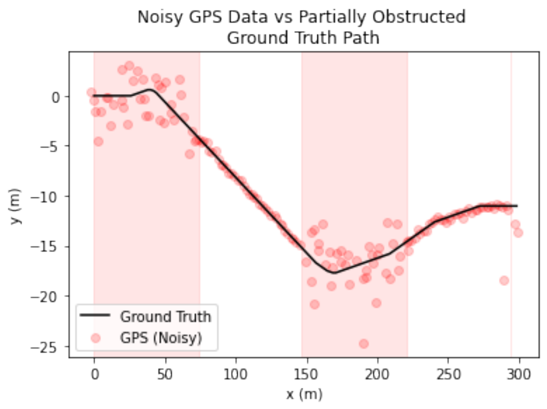
(b) Noisy Encoder Path vs GT



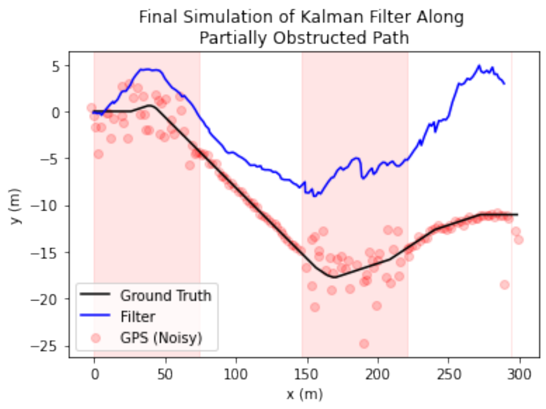
(c) Noisy GPS Data vs GT



(d) Normal Kalman Sim

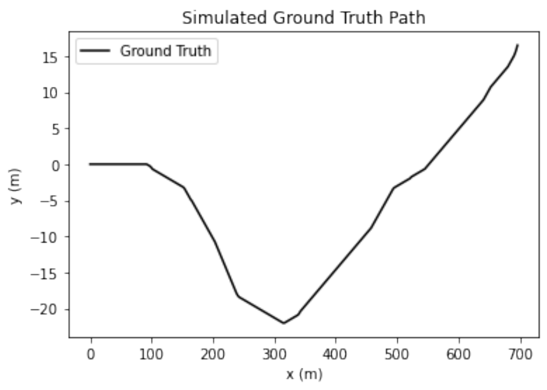


(e) Partially Obstructed GPS Data vs GT

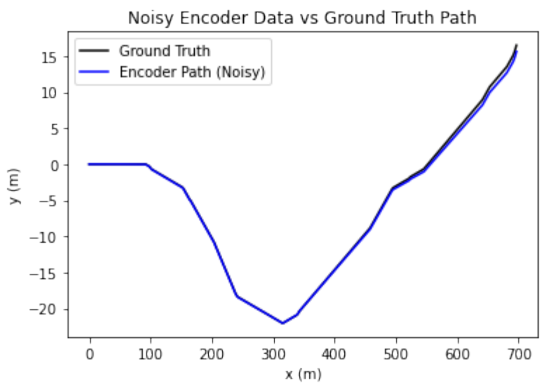


(f) Partially Obstructed Kalman Sim

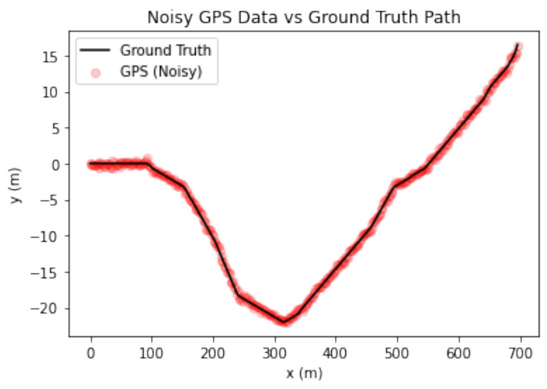
Fig. 3: Results from Increased Process Noise Group (STEPS=150, GPS_NOISE=0.25, ENC_NOISE=1)



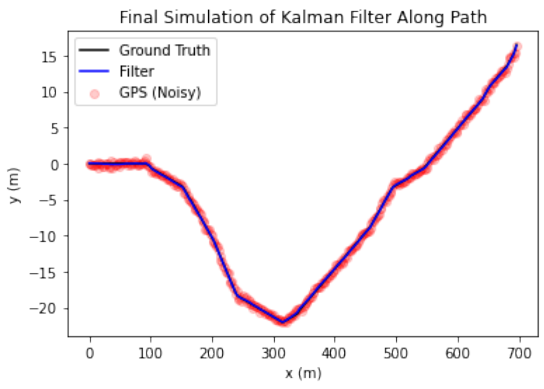
(a) Ground Truth



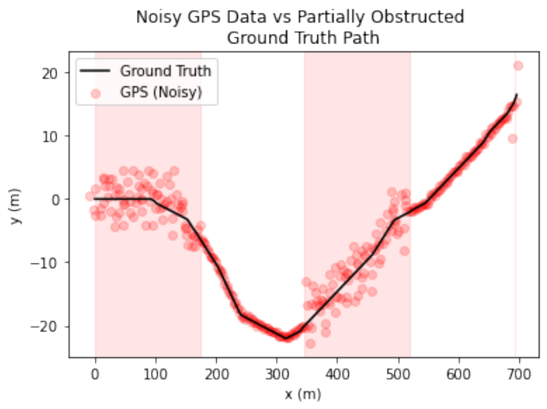
(b) Noisy Encoder Path vs GT



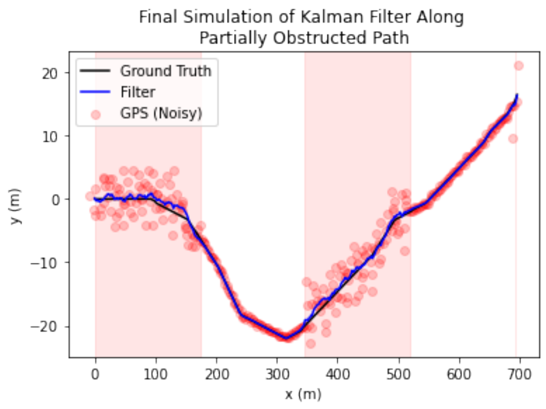
(c) Noisy GPS Data vs GT



(d) Normal Kalman Sim



(e) Partially Obstructed GPS Data vs GT



(f) Partially Obstructed Kalman Sim

Fig. 4: Results from Increased Steps Group (STEPS=300, GPS_NOISE=0.25, ENC_NOISE=0.0065)