Simulation-Based Engineering Lab
University of Wisconsin-Madison
Technical Report TR–2023–05

Particle Filter basic implementations

Stefan Caldararu, Huzaifa Unjhawala, Harry Zhang, Dan Negrut

Department of Mechanical Engineering, University of Wisconsin – Madison

July 6, 2023

Contents

1 Introduction

This report provides an overview of Particle Filter (PF) algorithm used to track a wheeled vehicle called the Autonomous Resaerch Testbed (ART) vehicle [1] using two inputs – throttle and steering control, and two measurements – GPS and Magnetometer readings. In the past, an Extended Kalman Filter was used for this purpose [2, 3]. The PF algorithm has advantages over an EKF one due to its modularity in design and handling of nonlinear sensor noise. In terms of the former aspect, it is very easy to have one component swapped out for an improved one for a filter that already works, and directly compare results with minimal additional tuning to already set-up components. With regard to the latter, in Sec. 5 we describe future work involving changes to the weighting model. The PF is not required to make assumptions of Normal Distribution noise in GPS measurements, leading to a possible weighting that assumes Random Walk noise as described in [4]. This can lead to future validation of the Random Walk GPS noise model, as we can gauge relative performance in reality of filters that make different assumptions about the noise model. In Sec. 2, we present a brief literature review and general background knowledge for a particle filter and its benefits. We direct the reader to [5] for a more detailed description and discussion of several examples. In Sec. 3 we provide a description of the model we use for the ART vehicle (called here "the ART"), and an explicit overview of each stage of the algorithm. Additionally, in Sec. 4 we provide a brief analysis of performance in a mock-up Python environment, and subsequently in the simulation engine CHRONO [6]. Finally, we provide a description of future work for this filter in Sec. 5.

2 Background

The Particle Filter fits in the class of Evolutionary Algorithms [7]. It maintains multiple "particles", each having its own estimation for the current state of the vehicle. The filter then uses the observation model along with sensor data readings to generate a weight which it assigns to each particle, representing the likelihood that this particle represents the true state of the vehicle. Finally, a weighted average between particles is returned, representing the Particle Filters state estimate.

The basic PF, as described in [5], has two key components: a prediction phase and a correction phase. The prediction phase functions similarly to that of the EKF, as we are just progressing our estimated state using the vehicle dynamics model [8]. Here, there are two differences, though. We are maintaining multiple particles, each having their own state; additionally, each of these particles has some form of randomness added in order to create a diverse population.

2.1 Basic Model Problems

As described in [5], there are several drawbacks that can hinder a PF, of which we discuss four. The first is the **degeneracy** problem. This happens when the particles start to diverge, and we only have one particle that gets weighted heavily, with the rest having negligible weights. In this case, we are essentially just doing dead reckoning. To solve this issue, we follow a similar method to the first solution described in [5]. Every five timesteps, we go through a resampling process, as described in Sec. 3.3. Here, the particles with low weight are pruned, and the particles with high weight are split.

Secondly, the basic PF suffers from the **impoverishment** problem. This is where we have the opposite problem, in that we are not introducing enough randomness to our system and so all of the particles follow very similar trajectories, drifting away from the true measurement. This issue is solved by ensuring that there is enough randomness being introduced to the system.

Finally, we have the **filter divergence** and **importance density** issues. The former occurs when our measurements are not adequate, or our weighting model is inaccurate and so the estimation diverges from reality. Issues with the importance density occur when we don't have sufficient particles to accurately represent enough distributions of states, and so we are unable to create an accurate depiction of the true state. Due to this, we often need to increase the number of particles we maintain as the number of states increases, making PF computationally expensive to the point of not being affordable in real life applications.

2.2 Model Solutions

As stated above, most of these problems can be solved by accurately tuning a model or changing various aspects of the algorithm. Specifically, the filter divergence problem can be avoided by ensuring that the weighting model actually does weight particles closer to the ground truth higher. The importance density issue is tackled by increasing the number of particles maintained. We discuss the randomness techniques used in Sec. 3.4 to avoid the Impoverishment Problem. Finally, the filter divergence problem is solved by resampling the particles at regular intervals, see discussion in Sec. 3.3.

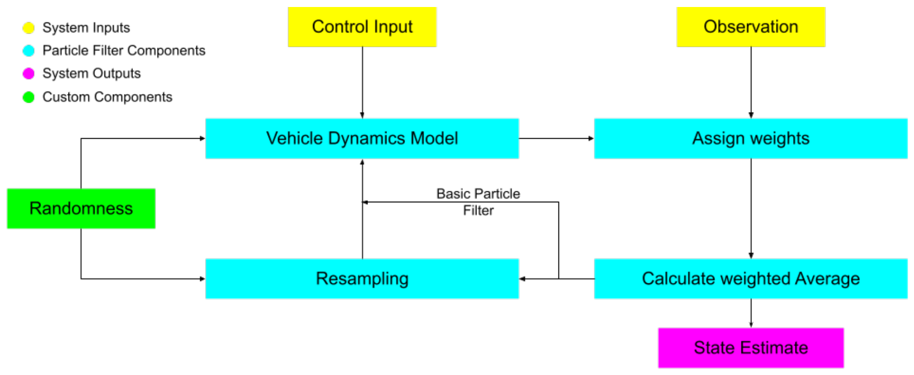


Figure 1: High-level overview of information flow in a solution that uses a PF for state estimation.

3 Model Description

In this section, we present our formulation of a particle filter, designed for the ART and dART vehicles (dART is a digital twin of ART, which is used by the state estimator to propagate the state of a particle forward in time). At timestep t, this model takes a control input $\mathbf{u}^t = [\alpha^t, \delta^t]$ consisting of the steering (α) and throttle (δ) control input; as its observation, the model knows $\mathbf{z}^t = [x_m^t, y_m^t, \theta_m^t]^T$, consisting of x and y coordinates, and heading angle θ – this is essentially GPS and Magnetometer information. The subscript m represents the *measurement*. Provided with this information, i.e., input and measurement, the particle filter generates a state estimate $\mathbf{q}_m^t = [x_s^t, y_s^t, \theta_s^t, v_s^t]^T$, consisting of the x and y coordinates, heading angle θ, and vehicle velocity v.

In Sec. 3.1, we describe the 4 Degree of Freedom (4DOF) vehicle dynamics model used in this implementation. We describe the weight assignment algorithm used in Sec. 3.2. In Sec. 3.3, we discuss the resampling technique used to avoid the degeneracy problem. Finally, in Sec. 3.4, we describe where in the model we include randomness in order to avoid impoverishment.

3.1 Vehicle Dynamics Model

The Particle Filter is used in conjunction with a 4DOF model that propagates forward in time the dynamics of the vehicle. This model updates states in accordance with Eq. 1b. The first three equations for this model are a standard bicycle model, with the last equation modeling the full powertrain of the vehicle with regard to the throttle input and model parameters. Most 4DOF bicycle models assume that the velocity of the vehicle is provided as an input, but we found this to be inadequate for our applications. We direct the reader to [8] for a further discussion of this topic.

$$\dot{\mathbf{q}} = \mathbf{f}(\mathbf{q}, \mathbf{u}) \; , \tag{1a}$$

where

$$\mathbf{q} \equiv \begin{bmatrix} x \\ y \\ \theta \\ v \end{bmatrix} \qquad \text{and} \qquad \mathbf{f}(\mathbf{q}, \mathbf{u}) \equiv \begin{bmatrix} cos\theta \cdot v \\ sin\theta \cdot v \\ \frac{v \cdot tan(\delta)}{l} \\ \frac{R_{wheel} \cdot \gamma}{I_{wheel}} \cdot [\alpha \cdot f_1(v) - \frac{v c_1}{R_{wheel} \gamma} - c_0] \end{bmatrix} , \tag{1b}$$

and

$$f_1(v) = -\frac{\tau_0}{\omega_0 \cdot R_{wheel} \cdot \gamma} \cdot v + \tau_0 \; . \tag{1c}$$

3.2 Weight Assignment

The current model for weighting assumes a normal distribution for the GPS noise model. When the observation is acquired, each particle computes the distance between its estimate and the observed value. Here, we use the standard Euclidean distance between the two points on a 2D Local Tangent Plane (LTP). With this model, we place a plane tangent to the earth's surface at our starting location. We then project all Latitude and Longitude coordinates onto this plane, and compute the Cartesian coordinates of this point. While this model will distort distances far away from the origin, it works well for short distances which satisfy the needs of this application.

We first compute the distance d_i^{t+1} between the ith particle prediction, and current measurement. Note that $1 \leq i \leq N_p$, where N_p represents the number of particles used by the PF. We follow a similar idea for the heading to generate h_i^{t+1}:

$$d_i^{t+1} = \sqrt{(x_i^{t+1} - x_m^{t+1})^2 + (y_i^{t+1} - y_m^{t+1})^2} \tag{2a}$$

$$h_i^{t+1} = |\theta_i^{t+1} - \theta_m| \tag{2b}$$

$$D_i^{t+1} = D_i^t \ \cup \ \{d_i^{t+1}\} \setminus d_i^{t-(n_h-1)} \tag{3a}$$

$$H_i^{t+1} = H_i^t \ \cup \ \{h_i^{t+1}\} \setminus h_i^{t-(n_h-1)} \ , \tag{3b}$$

where n_h is the number of particle maintained in the history (herein we used $n_h = 10$). We then update distribution D and H maintained by each particle in accordance with Eq. 3. The most recent n_h "distance to measurement" calculations are maintained for each particle as a distribution, a fact indicated by subtracting the oldest term, $\setminus d_i^{t-(n_h-1)}$. Given the sensor noise model assumed (typically provided by the manufacturer), we would expect to see a normal distribution with 0 mean, and standard deviation around 1m. A particle i will have a higher weight if its distribution resembles the known distribution associated with the sensors used on the vehicle. Therefore, to compute the weighting of each particle, first we need to evaluate the Earth Mover's distance (EMD) between our particle's distribution (D and H) and the distribution associated with the noise model [9]. An efficient EMD computation algorithm involves a network flow computation. We define the flow constraints as follows:

$$f_{ij} \geq 0, \ 1 \leq i \leq m, \ 1 \leq j \leq n \tag{4a}$$

$$\Sigma_{j=1}^{n} f_{ij} \leq 1, \ 1 \leq i \leq m \tag{4b}$$

$$\Sigma_{i=1}^{m} f_{ij} \leq 1, \ 1 \leq j \leq n \tag{4c}$$

$$\Sigma_{i=1}^{m} \Sigma_{j=1}^{n} f_{ij} = min(m, \ n) \ . \tag{4d}$$

Here, we have two distributions of points, $R = \{x_1, x_2 ... x_m\}$ and $Q = \{y_1, y_2 ... y_n\}$. Constraint 4a ensures that all flows are nonnegative, while constraints 4b and 4c ensure that the flows through each node do not exceed 1. Finally, constraint 4d ensures that we get a total matching equal to the number of points in the smaller distribution. This is a simplification of the network flow formulation described in [9]. This will give us a family of possible flows, $\mathbb{F}$.

Given a flow $F \in \mathbb{F}$, the amount of work done by this flow is defined as

$$WORK(F, R, Q) = \Sigma_{i=1}^{m} \Sigma_{j=1}^{n} f_{ij} \Delta_{ij} \tag{5}$$

Where f_{ij} is the flow going from x_i to y_j, and Δ_{ij} is the distance between the two points in the distribution. Finally, the EMD between our two distributions is given by solving the optimization problem described in Eq. 6:

$$EMD(\mathbb{F}, R, Q) = \frac{min_{F \in \mathbb{F}} WORK(F, R, Q)}{min(m, \ n)} \tag{6}$$

At initialization, we also created a sample distribution that fits the normal distribution for our GPS noise model, and a similar one for our heading noise model. These will be named N_{GPS} and N_{MAG}, where

$$N_{GPS} \equiv |N(0, 0.8)| \qquad \text{and} \qquad N_{MAG} \equiv |N(0, 0.1)| \ . \tag{7}$$

It is important to note that in practice N_{GPS} and N_{MAG} are represented as 100 sampled points from each distribution. This is done in order to make comparison with D_i and H_i easier. We now use the EMD to compare distributions for each particle, and assign a weight to each particle as follows:

$$w_i^{t+1} = \frac{0.9}{EMD(D_i^{t+1}, N_{GPS})} + \frac{0.1}{EMD(H_i^{t+1}, N_{MAG})} \tag{8}$$

The easiest way of thinking about the EMD is we want to minimize the amount of work required to get the distributions to match. If we have very different distributions, then the EMD will be very high, giving a low particle weight. Particle weights are then normalized.

Additional Reduction

We additionally provide a reduction of the Earth Mover's distance simplified above to an $O(n \log(n))$ computation with n being the number of points we maintain in our distributions (here 100 points). This reduction is possible due to the fact that we are comparing a finite number of equally weighted points in our distributions. This means that when we "move matter" from locations in one distribution to another, we are just doing a direct matching of points in the distribution. This means the network flow computation described above creates a bijective matching between the two sets. Following this, it is clear to see that the minimum cost matching can be found by sorting the values in the two distribution sets, and then matching them by index. Following this, the EMD will be the distance between values with the same index in the two distributions. This computation is described in Eq. 9.

$$\text{QuickSort}(R),\ \text{QuickSort}(Q) \tag{9a}$$

$$EMD(\mathbb{F}, R, Q) = \Sigma_{i=1}^{n} |x_i - y_i| \tag{9b}$$

This is a significant improvement in performance relative to the standard EMD computation. While this is yet to be implemented, it has no effect on the filters performance other than computation time, which has not yet been analyzed.

This is a fairly different weighting scheme than the standard Bayesian approach, as outlined in [5]. We do this as we believe the EMD provides good maintenance of previous particle information, and accurate weight computation. This is presently an hypothesis that remains to be validated.

3.3 Resampling

The current Resampling approach is similar to the one suggested in [5]. However, rather than resampling every timestep we do so every five timesteps. After assigning weights to each of our N_p particles, we create a new generation of particles based off of the distribution of weights. We will resample particles randomly with replacement, with probability proportional to a particle's weight. The easiest way of depicting this method is by an example with three particles. Suppose our particles, P_1, P_2, P_3, have weights $w_1 = 0.2$, $w_2 = 0.5$, $w_3 = 0.3$. We first calculate the cumulative weights of each particle:

$$w_1^{cumm} = w_1 = 0.2 \tag{10a}$$

$$w_2^{cumm} = w_1 + w_2 = 0.7 \tag{10b}$$

$$w_3^{cumm} = w_1 + w_2 + w_3 = 1.0 \tag{10c}$$

The cumulative weights lead to a "range" that is assigned to each particle. P_1 is assigned the range $[0, 0.2)$, P_2 is assigned the range $[0.2, 0.7)$, and P_3 is assigned the range $[0.7, 1.0]$. Next, we sample three random values from a uniform distribution over $[0, 1]$. Each particle has a likelihood of the sampled point falling in its range equal to its weight. Suppose the three points we sample are 0.126, 0.545, and 0.698. We want to find the particle which is assigned the range which each random sample falls in. We can do this by finding the first cumulative weight which is larger than the random sample, assuming they are ordered w_1^{cumm}, w_2^{cumm}, ..., $w_{N_p}^{cumm}$. We will propagate the particle which satisfies this condition. Therefore, in this example we will be propagating particle P_1 once, and particle P_2 twice to generate three new particles. This method will have a high likelihood of generating many of the particles with high weights, and pruning particles with low weights, while still allowing room for outliers to be maintained.

3.4 Randomness

A big factor determining the efficacy of the PF is the mechanism used to introduce randomness to the particles. As mentioned previously, if we were to not do this at all then we would end up with a system that performs the same as dead reckoning. If we have all particles start in the same state, after propagating through the motion model they will end up in the *same* new state. They would then all receive the same weight, and this cycle would continue. The PF discussed has three stages where randomness is injected into the algorithm.

First, randomness is added in the control input. This helps diversify the particles, and also will help capture terrain profiles. It is difficult to capture terrain information (such as driving at an incline) within a vehicle dynamics model that is as basic as the one used here and described in Eq. 1. Feeding some particles lower throttle inputs can be an easy way of correcting for this, as the vehicle model will slow down. This is modeled by Eq. 11, which generates the new state of a particle at the next timestep:

$$\mathbf{q}_i^{t+1} = \mathbf{q}_i^t + h\mathbf{f}(\mathbf{q}_i^t, \mathbf{u}^{t+1} + [N(0, \sigma_\alpha), N(0, \sigma_\delta)]^T) \ , \tag{11}$$

where in the Forward Euler scheme used above in conjunction with Eq. 1a, h is the integration step size, and the value $N(0, \sigma)$ is a single point sample from the normal distribution, with σ_α and σ_δ being values predetermined during filter tuning.

Second, when we initialize the filter we initialize the particles' location randomly around the first measurement. This helps create a diverse initial population as we are not initializing all particles at the same point.

Thirdly, when we resample the particles, we add some randomness. We have two equations for generating resampled particles:

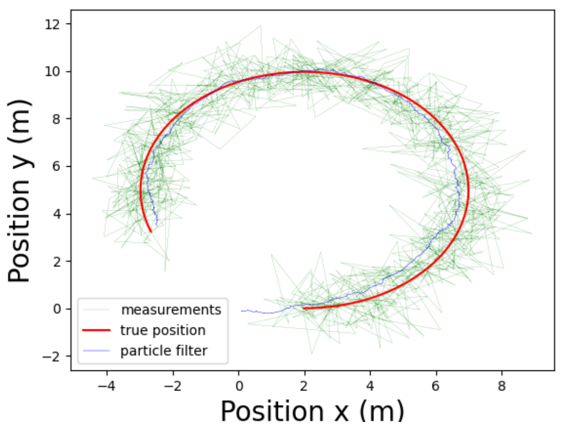
$$P^{new}(P_i) = P_i \tag{12a}$$

$$P^{new}(P_i) = P_i + [N(0, 0.1), N(0, 0.1), 0, 0]^T \ , \tag{12b}$$

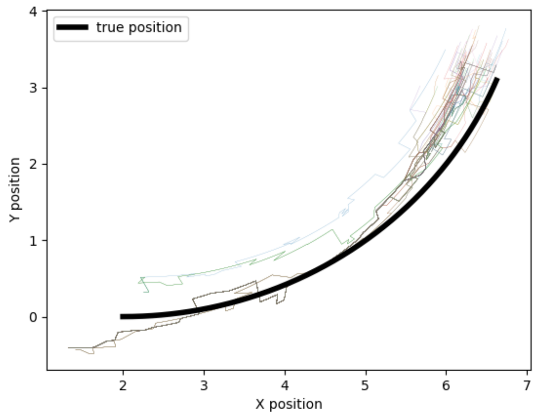
where P^{new} is the new particle we are generating, and P_i is the particle we are using to generate it. If a particle only gets propagated to the next generation once, then we use Eq. 12a. If a particle has multiple children getting propagated into the next generation (e.g., P_3 in the example in Sec. 3.3), then we have its first child get propagated with Eq. 12a, and the remaining children get propagated with Eq. 12b. This method helps prevent impoverishment, while additionally allowing for the filter to "correct" itself when it has all of the particles diverging from the ground truth.

4 Demonstration

We provide a number of tests designed to gauge the performance of this algorithm prior to deployment in reality. First, we demonstrate the performance of this model in a mock-up Python environment. This fake environment generates control inputs, and then uses the 4DOF model to propagate a vehicle forward in time. We start the true position of the vehicle 2 meters ahead of where the filter thinks the vehicle is starting, and add Standard Gaussian distribution noise to the x and y coordinates and the heading angle in order to generate "measurements". In Fig. 2a, we demonstrate the final state estimates for this model. The red line shows the ground truth, blue shows the filter's position estimate, and green line shows the measurements. In Fig. 2b, we show the states of each individual particle at a timestep partway through the state estimation process. In these examples, we have $N_p = 100$. The black line shows the ground truth. We can see that most particles share a common ancestor, and we can also see the particles "jump around" due to the randomness added at the resampling stage.



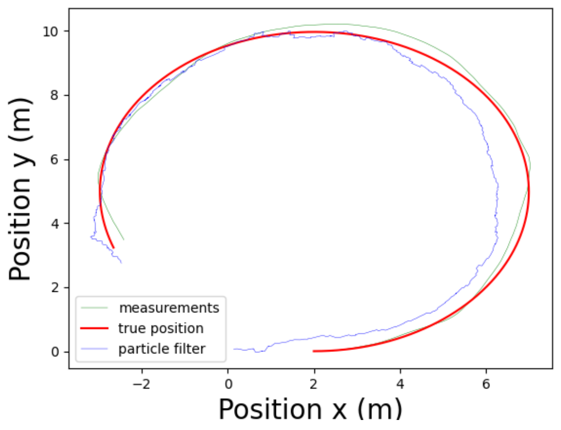
(a) Filter Final Result



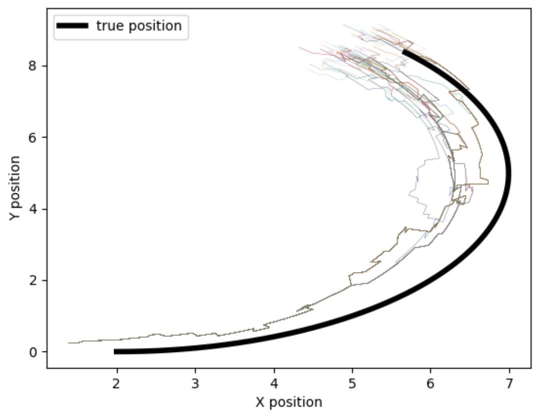
(b) Depiction of Particles

Figure 2: Results obtained with the proposed PF when used in a mock-up Python environment.

Additionally, we show the filter's performance with our Random Walk noise-based GPS model [4]. That GPS noise model, which is used in Chrono [6,10], more accurately represents true GPS measurements, as opposed to the standard normal distribution noise. As can be seen in Fig. 3, the filter sometimes overfits the measurement, and overall performs worse than in the previous test, as the filter is assuming normal distribution noise, see Eq. 7. The tests shown in Fig. 3 are once again run in a faked Python environment.



(a) Filter Final Result with Random Walk noise



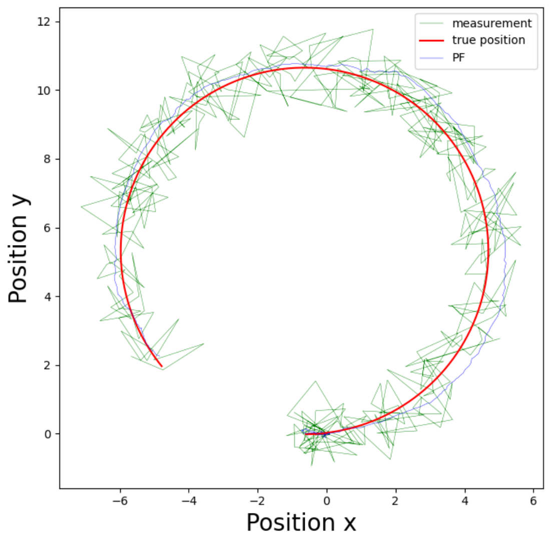
(b) Depiction of Particles with Random Walk noise

Figure 3: Results obtained when the PF discussed was used in conjunction with a Chrono GPS model.

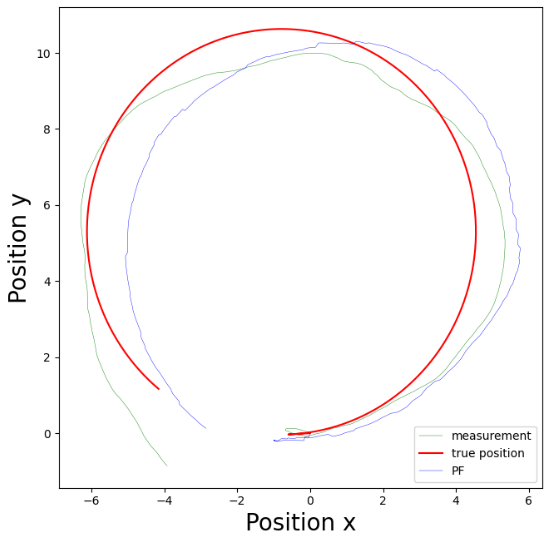
Finally, we present results run in the simulation engine Chrono [6], using the virtual vehicle dART, which is the digital twin of ART. This is the next step in the algorithms development, before it can be deployed on the real vehicle ART. We once again have the ground truth represented by the red line, GPS measurements represented by the green line, and filter estimates represented by the blue line. As can be seen, the filter performs much better in Fig. 4a, where the simulation is generating measurements using Normal Distribution Noise. In Fig. 4b, the same simulation is run, but with Random Walk Noise generated. The filter seems to perform much worse, even with an overall smaller total amount of noise, similar to the results shown in the fake data generated above.

5 Future Work

There are multiple aspects of this Particle Filter algorithm that can be improved. To begin, this algorithm should be run in a real system in order to compare it to current implementations of the EKF [12]. Additionally, incorporating the Random Walk noise model described in [4] into the weighting model should generate better performance. Implementing such a model, and comparing the its differences with a Particle Filter assuming Normal Distribution noise could serve to validate the Random Walk GPS noise model. As shown above, the current filter, PF^{ND} which assumes normal distribution noise performs very well in simulations with normal distribution noise. But, when we use the Random Walk noise model in simulation, this filter overfits the measurements. Once we implement a filter that assumes the Random Walk noise model, it should be able to better predict what measurements *should* look like. This filter, PF^{RW}, should perform well with the Random Walk noise model, and may not perform as well with the normal distribution noise model. After sufficient simulation testing and calibration, we would be able to begin reality testing. Using an RTK GPS with centimeter scale accuracy, we would be able to gauge how well each filter (PF^{RW} and PF^{ND}) performs in reality. We hope to see similar performance in reality as we do in simulation. This would mean that PF^{ND} would overfit the measurements provided by a standard GPS in comparison to the RTK GPS and PF^{RW} would be able to generate accurate



(a) Normal Distribution noise in Simulation



(b) Random Walk noise in Simulation

Figure 4: Results obtained with the proposed PF when used in Chrono with the dART vehicle. Left: normal distribution in sensor noise. Right: measurements provided by the GPS model available in Chrono::Sensor [11].

state measurements from the standard GPS.

Further improvements of the Particle Filter could include different added randomness to particles, and particularly including a "genetic signature" for each particle. Having different particles intrinsically behave differently could prove beneficial when traveling on different types of terrain, or at different levels of incline. The largest drawback to the Particle Filter is the high computational cost. Transferring the algorithm to a C++ environment would greatly improve the speed the filter is able to operate at. Additionally, this filter fits well into a framework which runs different particles in parallel, and may benefit from further GPU parallelization. Lastly, it would be interesting to use the PF in conjunction with the IMU for state estimation. Encoroprating orientation information into the dynamics model's characteristics would potentially allow for large improvements in position estimation.

References

[1] A. Elmquist, A. Young, I. Mahajan, K. Fahey, A. Dashora, S. Ashokkumar, S. Caldararu, V. Freire, X. Xu, R. Serban, and D. Negrut, "A software toolkit and hardware platform for investigating and comparing robot autonomy algorithms in simulation and reality," *arXiv preprint arXiv:2206.06537*, 2022.

[2] R. E. Kalman, "A new approach to linear filtering and prediction problems," *Transactions of ASME, Journal of Basic Engineering*, no. 82, pp. 35–45, 1960.

[3] Y. Pei, S. Biswas, D. S. Fussell, and K. Pingali, "An elementary introduction to kalman filtering," 2017.

[4] S. Caldararu, H. Zhang, I. Mahajan, T. Hansen, S. Chatterjee, N. Batagoda, R. Serban, and D. Negrut, "Using random walks to simulate gps sensing for applications in robotics and autonomous vehicles." `https://sbel.wisc.edu/wp-content/uploads/sites/569/2023/03/TR-2022-02.pdf`, 2022.

[5] J. Elfring, E. Torta, and R. v. d. Molengraft, "Particle filters: A hands-on tutorial," *Sensors (Basel)*, no. 438, 2021.

[6] A. Tasora, R. Serban, H. Mazhar, A. Pazouki, D. Melanz, J. Fleischmann, M. Taylor, H. Sugiyama, and D. Negrut, "Chrono: An open source multi-physics dynamics engine," in *High Performance Computing in Science and Engineering – Lecture Notes in Computer Science* (T. Kozubek, ed.), pp. 19–49, Springer International Publishing, 2016.

[7] A. Rodríguez and F. Moreno, "Evolutionary computing and particle filtering: A hardware-based motion estimation system," *IEEE Transactions on Computers*, vol. 64, no. 11, pp. 3140–3152, 2015.

[8] H. Zhang, H. Unjhawala, S. Caldararu, I. Mahajan, L. Bakke, R. Serban, and D. Negrut, "Simplified 4dof bicycle model for robotics applications," tech. rep., Simulation-Based Engineering Laboratory, University of Wisconsin-Madison, 2023. `https://sbel.wisc.edu/wp-content/uploads/sites/569/2023/06/TR-2023-06.pdf`.

[9] E. Levina and P. Bickel, "The earth mover's distance is the mallows distance: some insights from statistics," in *Proceedings Eighth IEEE International Conference on Computer Vision. ICCV 2001*, vol. 2, pp. 251–256 vol.2, 2001.

[10] H. Mazhar, T. Heyn, A. Pazouki, D. Melanz, A. Seidl, A. Bartholomew, A. Tasora, and D. Negrut, "Chrono: a parallel multi-physics library for rigid-body, flexible-body, and fluid dynamics," *Mechanical Sciences*, vol. 4, no. 1, pp. 49–64, 2013.

[11] A. Elmquist, R. Serban, and D. Negrut, "A sensor simulation framework for training and testing robots and autonomous vehicles," *Journal of Autonomous Vehicles and Systems*, vol. 1, no. 2, p. 021001, 2021.

[12] H. Zhang, S. Chatterjee, T. Hansen, S. Caldararu, I. Mahajan, N. Batagoda, L. Fang, R. Serban, and D. Negrut, "Formulating model predictive control (mpc) strategies in conjunction with error dynamics based waypoint-seeking to model robust vehicle control," tech. rep., Simulation-Based Engineering Laboratory, University of Wisconsin-Madison, 2023. `https://sbel.wisc.edu/wp-content/uploads/sites/569/2023/03/TR-2023-01.pdf`.