Fusion of sensors using Kalman and Particle Filters to estimate the simulated ball position

Yernar Kubegenov, Frank Deinzer
Faculty of Computer Science - University of Applied Science Würzburg-Schweinfurt
Wurzburg, Germany
yernar.kubegenov@study.thws.de

***Abstract*—Paper investigates use case of Kalman and Particle filters to estimate the trajectory of a thrown ball. These Filters are powerful techniques for sensor fusion to estimate the states of any system, and commonly used in various fields. They work by using the transitional model of the system and observations coming from sensors over time to iteratively estimate the state of the system. Both sensor fusion algorithms have been implemented on a commonly used ball-throwing example to estimate the position states of the ball from the noisy observation samples. In this paper, I will investigate basic use of the Kalman and Particle filter, as well as the assumptions made during their realization. In addition, this paper also examines filter's robustness to sensor measurement dropout, when the system state temporarily fails for a certain period of time leading to missing observations.**

***Index Terms*—kalman filter, particle filter, state estimation, projectile motion, ballistic trajectory**

I. Introduction

Estimating ballistic trajectories is a common problem in physics and engineering. It involves predicting the path that a projectile will follow after being launched into the air. This task is fundamental in various applications, including missile guidance, rocket flight, and sports analytics. In missile guidance, for example, accurate trajectory estimation is crucial to ensure that the missile hits its target. Similarly, in sports analytics, precise estimation of a ball's trajectory can provide insights to analyze and improve athletes' performance.

While tracking sensors can help with this task, there are many problems with their application across a variety of fields. For instance, sensors used to track the object's trajectory could easily fail due to system uncertainty or unsuitability of the environment, which reduces the reliability of sensor measurements. Furthermore, any measurement will be corrupted to some degree by noise, biases, and device inaccuracies. [1] Therefore, extracting reliable information from a noisy signal is crucial. In modern world, the stochastic filtering such as Kalman and Particle filter are commonly addressed as recursive solution to the problem above.

In this paper, overview of the these two filter are provided and their mathematical basis, as well as the assumptions made in their implementation. Simple experiment is conducted to investigate their performance in estimating the trajectory of a thrown ball under various levels of measurement noise and their robustness under total failure circumstances in state estimation. My results demonstrate the effectiveness of the filters in state estimation and shed light on their behavior in scenarios where measurements may be incomplete, noisy, or non-linear. I also leave a room for open questions after analysing the my findings after first using the filtering algorithms.

The Kalman Filter is a powerful tool for state estimation. Nowadays, Kalman filters have been applied in the various engineering and scientific areas, including communications, machine learning, neuroscience, economics, finance, political science, and many others. [2] It estimates the state of a system iteratively using a system model and measurements. The filter can optimally estimate the state of a linear system with given Gaussian noise and with a set of assumptions about the system and the measurement models.

The Particle Filter is a non-parametric filter that can handle non-linear and non-Gaussian models. It represents the posterior distribution of the state using a set of weighted particles, which are propagated through the system model and resampled based on the likelihood of the measurements. [3]

This paper has the following structure. The introduction section briefly describes the problem and introduces the filtering algorithms. In the following Sensor Fusion Methodology section, we present our simulated model of a thrown ball, including short descriptions of the Kalman and Particle Filters, as well as discrete mathematical implementations of filters applied to our example. In the Results and Discussion section, the accuracy and performance of both filters are evaluated under several circumstances. Finally, a brief summary of the paper is provided in conclusion section, along with questions for future research.

II. Sensor Fusion Methodology

A simulated ball experiment was conducted to investigate the performance of the Kalman Filter and Particle Filter in estimating the states of a thrown ball. The experiment involved launching a ball from various heights, angles, and velocities, and modeling it with both filters. The trajectory of simulated can be seen in Figure 1.

The movement of the ball is modelled on the basis of the physical based projectile motion, which involves two dimensional horizontal and vertical displacements. Here are key formulas used to model the simulation. [4] The horizontal component is defined as following:

$$x = v_{0x}t \tag{1}$$

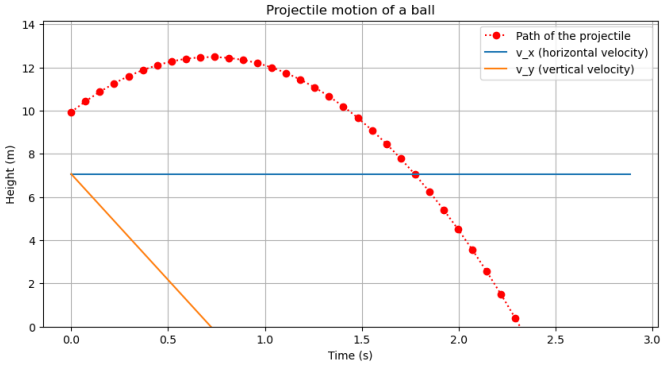

Fig. 1. Trajectory of simulated ball-throwing example with launch parameters like *launch height*, *launch speed*, and *launch angle* of the ball. The y axes is limited to y=0 to have imaginary ground.

where x is the horizontal displacement, v_{0x} is the initial horizontal velocity, and t is the time. The vertical component is defined by:

$$y = v_{0y}t - \frac{1}{2}gt^2 \tag{2}$$

where y is the vertical displacement,v_{0y} is the initial vertical velocity, g is the acceleration due to gravity (approx. 9.8 m/s² on the surface of the Earth), and t is the time. The v_{0x} and v_{0y} are calculated using the initial velocity v_0 and the launch angle θ. This simulated trajectory of thrown ball will serve as ground truth to evaluate the experiment results.

A. Kalman Filter

The methodology for the Kalman Filter involves estimating the state of the ball in the x and y directions. In real-world scenarios, direct tracking of motion is not possible, and thus measurements and sensors are used. However, sensors can be noisy, leading to inaccuracies in measurements. To address this challenge, the Kalman Filter leverages a discrete white noise model to represent the prediction and measurements of the ball's motion in maximum real-world scenarios where the ball's motion is subject to wind, air resistance, or imperfect knowledge of initial values. The goal of this approach is to apply the key features of Kalman Filtering to continuously estimate the state of the ball's motion under some uncertainty and evaluate its performance.

In general, Kalman Filter Algorithm consist computationally efficient 5 major steps, where **predict** and **update** steps are iteratively does the state estimation from previous to new state. [5]

Step 1: Building a Model and Initializing System States

This step involves setting up the state space model and the initial values for the system state.

The state space model equations are as follows:

$$\begin{aligned} \hat{q}_{k+1} &= F\hat{q}_k + Bu_k + \epsilon_k, \\ o_k &= Hq_k + \delta_k, \end{aligned} \tag{3}$$

where q_k represents the state mean vector at the k^{th} time, F is the state transition matrix, B corresponds to the control input matrix, u_k is the control vector, ϵ_t denotes process noise with covariance matrix Q , o_k indicates observation vector at k^{th} time, H is the observation model (measurement matrix), and δ_k symbolizes observation noise with covariance matrix R.

Initial conditions are set as:

$$\begin{aligned} q_0 &= \text{Initial state estimate}, \\ P_0 &= \text{Initial error covariance estimate}. \end{aligned}$$

Step 2: Start the System

Here, we start iterating over the observations o_k in order to estimate the state q_k for each time step k.

Step 3: Predict System State Estimate

This step involves predicting the next system state and error covariance, given the current system state and error covariance.

Predicted state estimate:

$$\hat{q}_{k+1|k} = A\hat{q}_{k|k} + Bu_k, \tag{4}$$

and predicted estimate covariance:

$$P_{k+1|k} = AP_{k|k}A^T + Q \tag{5}$$

where P is arbitrary covariance matrix and Q is the covariance of the process noise ϵ_k.

Step 4: Compute the Kalman Gain

The Kalman gain is calculated using the predicted estimate covariance and the observation model. It serves as balancing factor between the predicted state and the new observation.

The Kalman gain, K_{k+1}, is computed as:

$$K_{k+1} = P_{k+1|k}H^T(HP_{k+1|k}H^T + R)^{-1} \tag{6}$$

where R is the covariance of the observation noise δ_k.

Step 5: Update System State and System State Error Covariance Matrix

In this step, the predicted system state and error covariance are updated using the new observation and the Kalman gain. Equation to update the system states as follows:

$$\hat{q}_{k+1|k+1} = \hat{q}_{k+1|k} + K_{k+1}(o_{k+1} - H\hat{q}_{k+1|k}) \tag{7}$$

and updated covariance matrix:

$$P_{k+1|k+1} = (I - K_{k+1}H)P_{k+1|k} \tag{8}$$

where I represents the identity matrix.

After updating, the iteration continues to the next time step (returns to Step 3), using the updated system state and error covariance.

There are many forms of equations that used in many literature, but for this paper the same as Wikipedia notations are used. [6] For the simplicity and deprecate worrying about the specific of the linear algebra, already pre-defined Kalman Filter implementation is used from '*filterpy*' library in Python. [7] Our implementation purely involves the definition of several matrices to represent the system dynamics, noise characteristics, and initial state estimation and uncertainty.

To apply the Kalman Filter properly, we need to design the state (**q, P**), process (**F, Q**), measurement (**R**), action matrix

and control vector(**B, u**) and measurement matrix (**H**) with input vector (**u**) accordingly to our 'Ball-thrown' example. They are defined as following, excluding the covariance matrices. In our example of a ball's motion, the state is defined as a four-dimensional vector.

$$\hat{q}_k = \begin{pmatrix} x \\ y \\ v_{0x} \\ v_{0y} \end{pmatrix}_k$$

The state matrix F is:

$$F = \begin{pmatrix} 1 & 0 & \Delta t & 0 \\ 0 & 1 & 0 & \Delta t \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

The action control matrix B and control input vector u are:

$$B = \begin{pmatrix} 0 & 0 & 0 \\ 0 & 0.5\Delta t^2 & 0 \\ 0 & 0 & 1 \\ 0 & \Delta t & 0 \end{pmatrix}$$

$$u = \begin{pmatrix} 0 \\ -g \\ 0 \end{pmatrix}$$

The matrixes F and B is designed based on projectile motion of the ball. [4] The measurement matrix H is:

$$H = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{pmatrix}$$

which allows us to track the x and y of the ball state components. Equation 10 demonstrates what a comprehensive transition model looks like:

$$\hat{q_k} = \begin{pmatrix} x \\ y \\ v_{0x} \\ v_{0y} \end{pmatrix}_k = \begin{pmatrix} 1 & 0 & \Delta t & 0 \\ 0 & 1 & 0 & \Delta t \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ v_{0x} \\ v_{0y} \end{pmatrix}_{k-1} + \\ + \begin{pmatrix} 0 & 0 & 0 \\ 0 & 0.5\Delta t & 0 \\ 0 & 0 & 1 \\ 0 & \Delta t & 0 \end{pmatrix} \begin{pmatrix} 0 \\ -g \\ 0 \end{pmatrix}_{k-1} + \epsilon_{k-1} \tag{10}$$

The **predict** and **update** functions are implemented in *FilterPy*. [7]

Lets go back to our simulation. The simulation will run until the Ball hits imaginary floor with Δt steps. The simulation generates noisy observations of the ball's position, which are then used by Kalman Filter to estimate the ball's x and y positions. The estimated position is compared with the true position to evaluate the performance of the filter.

B. Particle Filter

Our methodology employs a particle filter to accurately estimate the trajectory of two simultaneously launched balls. Particle filtering, also known as Sequential Monte Carlo estimation, is a technique used to solve filtering problems arising in signal processing and Bayesian statistical inference. [8] It is particularly beneficial in non-linear and non-Gaussian systems. In comparison, Kalman filter provides an optimum estimate only for unimodal linear systems with Gaussian noise.

Particle filters work on the principle of recursive Bayesian estimation. They use a set of particles (random samples) to represent the posterior distribution of some stochastic process given by noisy and partial observations. Each particle represents a possible state of the system. As new observations become available, the particles are updated and resampled based on their likelihood with respect to these observations.

The implemented particle filter algorithm to our example consists of several key steps,they are explained as follows:

Initialization: Particles are initialized with Gaussian noise around the initial conditions of the balls. The number of particles (N) was set to 500 in our experiments. Each particle has four state variables of the ball: **x** (horizontal position), **y** (vertical position), **vx** (horizontal velocity), and **vy** (vertical velocity).

Prediction: The particles are moved according to the projectile motion model from physics, with noise added. [4] The noise is collected from a Gaussian distribution with an average of 0 and a standard deviation equal to the standard deviation of the noise. It can be set as a hyperparameter. The process noise represents uncertainties in the model, such as wind and air resistance.

$$\text{particles}[x] \mathrel{+}= \text{particles}[vx] \cdot \Delta t + \\ + \text{np.random.normal}(0, x_{\text{noise}}, \text{size} = N) \tag{11}$$

$$\text{particles}[y] \mathrel{+}= \text{particles}[vy] \cdot \Delta t - 0.5 \cdot g \cdot \Delta t^2 + \\ + \text{np.random.normal}(0, y_{\text{noise}}, \text{size} = N) \tag{12}$$

$$\text{particles}[vx] \mathrel{+}= 0 \text{ and particles}[vy] \mathrel{+}= -g \cdot \Delta t \tag{13}$$

where, the x_{noise} and y_{noise} are the parameters for process noise.

Update: The weights of the particles are updated based on the difference between the predicted and measured positions. The weight of each particle represents how well it matches the observation. The weights are normalized so that they sum to one, which gives a probability distribution. The weights are used to determine which particles to keep and which to discard. The weights calculated using a Gaussian likelihood:

$$weights[i] = \frac{1}{\sqrt{2\pi z_{\text{noise}}^2}} \exp\left(-\frac{\text{distance}^2}{2z_{\text{noise}}^2}\right) \tag{14}$$

where, $distance$ is difference between particle and measurement, z_{noise} is measurement noise.

Resampling: Particles are resampled according to their weight. This means that particles with higher weights are more likely to be selected during the resampling process. We want to discard all the particles with lowest weights. To avoid the problem of particle degeneracy, we use resampling wheel method. This part plays significant role in filtering as there are many resampling algorithms can be applied. One of the simplest, so called *multinomial resampling* from "Stochastic Processes" presentation is used in this task. [10]

Estimate: The state estimate is computed as the weighted mean of the particles. This estimate represents the most probable state of the system given the observations so far.

$$\mu = \frac{1}{N} \sum_{i=1}^{N} w^i x^i \tag{15}$$

Here, the notation x^i to indicate the $\mathtt{i}^{th}$ particle.

Effective Sample Size (ESS): The ESS is calculated to evaluate the quality of the particle set. If the ESS falls below a certain threshold, the particles are resampled to avoid weight degeneracy. As a starting point the threshold assumed N/2.

$$\hat{N}_{\text{eff}} = \frac{1}{\sum w^2} \tag{16}$$

The particle filter is run in a loop where the aforementioned steps are repeated at each time step. In our experiments, we consider different scenarios for the two balls, varying their initial launch positions, directions, velocities, and the time intervals between observations. We also simulate periods of observation failure to test the filter's performance under these conditions. This can be achieved introducing a dropout period in update loop where updating the weights for some time steps are skipped.

To validate our results, we simulate real measured trajectories of the balls, adding Gaussian noise to represent measurement errors. The estimated trajectories are then compared first with these noisy measurements, then compared with ideal path of the ball , providing a performance benchmark.

It is worth noting that the particle filter algorithm does not require the measurements or system to be Gaussian or linear. It can handle any probability distribution and highly nonlinear problems. But for this experiment for simplicity, Gaussian distribution is mainly used.

In terms of software, our filters are implemented in Python using the *NumPy* library for numerical computations and *Matplotlib* for visualizations.

III. Results and Discussion

A. Kalman Filter

The Kalman Filter implementations on our simulation demonstrates excellent performance in estimating the projectiles' trajectory. The figure 2 below shows the effectiveness of the filter visually in tracking trajectories despite noisy remnants of observation.

The performance of a Kalman filter depends largely on the setting of its parameters. Selecting these parameters may affect

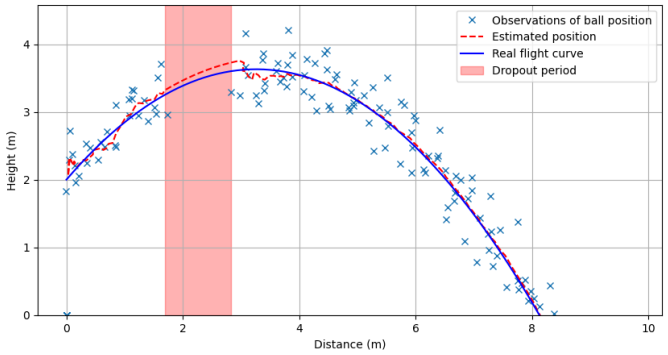

Fig. 2. Estimated position of the Ball using Kalman Filter with 0.01 time step, initial parameters are tuned to get such a accurate precision in estimation.

the ability of the filter to accurately assess the state of the dynamic system. Here we investigate the impact of various parameters and scenarios on the performance of the Kalman filter.

- **Launch Positions, Directions, and Velocities:** The Kalman filter is quite robust to variations in launch positions, directions, and speed of the ball. It uses the initial measurements to estimate the state of the system from even from a bad position and then updates these estimates as new measurements are obtained. Even though wrong initial assumptions are given to filter, it could go back to track very quickly once a good measurements was provided. This allows it to adapt to different launch conditions and still provide accurate estimates of the ball's trajectory. [9]
- **Time Intervals Between Observations:** The performance of the Kalman filter can be affected by the time intervals between observations. If the time intervals are too long, the filter may not be able to accurately track the ball's trajectory. However, if the intervals are too short, the filter may be overly sensitive to noise in the measurements. Therefore, it was important to choose an appropriate time interval for our application.
- **Observation Failure:** The Kalman filter can handle occasional observation failures by relying on its internal model to predict the state of the system. However, if the observations fail for an extended period, the filter's estimates may become increasingly inaccurate. We have noticed that without updating step the filter prediction can deviate over time from the tracking path.
- **Initial Parameters:** The initial parameters of the Kalman filter, such as the initial state estimate and the process and measurement noise covariance, can significantly affect its performance. If these parameters are not chosen correctly, the filter may not converge to the true state or may be overly sensitive to noise. Especially this part of Kalman Filter needed a broad knowledge of how to tune wisely to get the exact accuracy, as each parameter could be unique for each system.
- **Measurement noise R and process noise Q:** The

performance of the Kalman filter is tremendously influenced by its starting parameters, which include the initial estimation of the state and the covariance matrices of both process and measurement noise. Small experiments holding one parameter while varying second variable shows that larger Q and smaller R is telling rely less on its internal model or prediction, and more on incoming measurements. If R gets larger and Q smaller, the filter still was able to handle the tracking process. Conversely, a larger R signals to the filter that more trust should be placed in the prediction model and less in the measurements. Therefore, it's crucial to choose these parameters carefully based on the physical implication of specific application and the characteristics of the noise.

- **Covariance Matrix P:** The initial value of the covariance matrix P may have an important impact on filter performance. A high initial value can cause the filter to rely more heavily on the measurements for the initial estimates, while a low initial value can cause the filter to rely more heavily on the model predictions. Therefore, it was important to specify a more reasonable value for P, because in scenarios where the initial estimates were terribly wrong, the filter took a very long time to converge back to Ball's path.

In general, the performance of the Kalman filter is significantly influenced by various parameters and scenarios. Careful analysing the system for tuning and optimization of these parameters can greatly enhance the filter's performance in estimating the state of a system. Even for a simple linear task, the filter requires extensive analysis and a good understanding of the system variables, such as state vector, state models, noise covariances, etc. In our case, we have assumed that our observation has Gaussian noise, which slightly simplifies the filtering task in the simulation. However, in real world it is complicated and sophisticated, which will require rigorous investigation to get the filter work right. Making confident estimate of all parameters are "art" of science.

B. Particle Filter

The Python implementation of the particle filter gives estimates of the positions of the two balls at each time step. Our study of the two-ball scenario shows that the particle filter with appropriate hyper-parameters can estimate the two objects at each time step, even if there are significant errors in the system and in the sensor measurements. The plot illustrates the effectiveness of the filter in tracking trajectories of multiple objects and demonstrates the multimodality of the filter.

In Figure 3, we see the observed path of each projectile (red and blue solid lines), the estimated path of our particle filter model (red and blue crosses connected by a dotted line), and the true path (red and blue unfilled circles). The cyan and yellow dots show the state of the particles for the two spheres over time. The dropout period of time is indicated with red square are. Next, let's look at the impact of different scenarios on the filtering work.

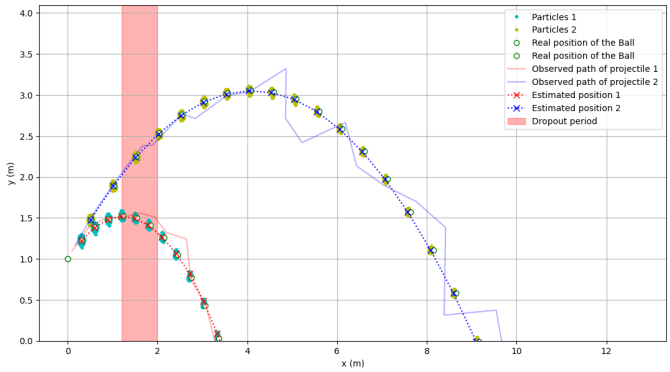

Fig. 3. Trajectory plot of the two balls with different launch variables

- **Different Launch Positions, Directions, and Velocities:** The initial launch parameters play an important role in the trajectory of the projectiles. We tested our model under different initial conditions. We varied the starting positions, starting angles (directions) and speeds of the two balls. Subsequently, it was observed that the particulate filter systematically provided accurate estimates of projectile trajectories under these variable initial conditions. This demonstrates the filter's ability to adapt to different initial states, highlighting its suitability for a range of physical scenarios. However, it's important to note that while the particle filter shows admirable adaptability to various initial states, it is sensitive to inaccuracies and noise in the initial conditions. Misestimations or excessive noise in the initial variables can lead the particle filter to a false track, thereby compromising the estimation accuracy of the entire system. Therefore, while the filter has a certain degree of robustness, the precision and reliability of the initial parameters are crucial for the success of the particle filter. This highlights the importance of precise system modeling and initial parameter configuration in the actual implementation of the particulate filter.
- **Time Intervals between Observations:** The time interval between observations affects how well the filter can track the projectiles' paths. As expected, smaller intervals resulted in more accurate estimations as there was less time for the projectiles to deviate from their predicted paths. Conversely, larger intervals made the filter estimates less accurate due to as much variation as possible in the actual trajectory between observations. However, the particle filter was still able to provide reasonable estimates, showing its robustness to changes in the observation frequency.
- **Failure of Observations:** During these periods, the filter had to rely solely on its motion model (prediction step) and the previously assigned weights. Naturally, the estimates started to deviate from the actual paths over time, as filter stops updating the weights, as there are no observations and leads to accumulating prediction errors.

However, when the observations were resumed, the filter quickly corrected itself and returned to accurate tracking. Hence, this demonstrates the particle filter's resilience and its ability to recover from periods of no observation only for a short period of dropouts. If the dropout period lasts longer, it severely undermines the effectiveness of the filter. This scenarios must to be treated with other strategies.

- **Different Initial Parameters:** We also experimented with different initial parameters for the particle filter itself, such as different numbers of particles and initial noise levels. Increasing the number of particles generally improved the accuracy of the estimation at the expense of computational resources. The level of initial noise added a degree of randomness to the initial states of the particles. A high level of noise made the initial particle states widely spread, which can be beneficial for scenarios with high uncertainty. Conversely, low noise levels resulted in particles that closely followed the initial conditions, which can be advantageous when the initial state is known with high accuracy. In addition, performance of a particle filter are strongly influenced by the measurement and process noise levels. High measurement noise can lead to broad weight distribution, making the filter less precise, while low noise allows for a more sharp weight distribution, focusing on particles that closely match the observed data. Similarly, high process noise ensures adaptability to dynamic changes at the expense of tracking precision, whereas low noise is beneficial for tracking consistent systems but may struggle with sudden changes. Thus, careful tuning of both noise levels was essential to balance accuracy, adaptability, and avoid overfitting risk.

The full Python code for this experiment is available on my GitHub repository. Here: https://github.com/yerkgb/sensor_fusion_portfolio

C. Discussion

From the plots, it is evident that the Kalman filter and the particle filter effectively follow the path of the balls. In spite of the noise inherent in the observations, the filter is capable of providing an approximation very close to the true path. The particle's filter's ability to control multimodal and dealing with non-Gaussian uncertainties is a significant advantage over the Kalman filter, making it more suitable for non-linear systems. Our experiment also found that filters remarkable strength of the filters when dealing with dropouts, which makes the filters high reliable in scenarios when the system receives no measurement signals or is corrupted by noise. Furthermore, the time intervals between observations play a crucial role in the accuracy of both filters. In general, the intelligent use of such filters can considerably improve the reliability and accuracy of monitoring and evaluation tasks.

IV. Conclusion

In conclusion, our study has vividly showcased the robustness and versatility of the Particle and Kalman Filters in a context of ballistic motion, dealing adeptly with both noise and measurement dropouts. Their success in such a simple fundamental problem affirms their utility in complex real-world applications where the system's behavior might encompass nonlinearities, non-Gaussian characteristics, or both. This paves the way for an array of potential applications, positioning these filters as key computational tools for state estimation across a spectrum of fields. As we move forward, an intriguing question arises: How can we further optimize these filters to improve their performance under varying conditions? How highly non-linear Paricle Filter models implementation differ from Extended Kalman Filter algorithm? How the filters' performance diverge under varying types of measurement noise models. These question opens up a new avenue for future research, inviting further exploration into the fascinating world of sensor fusion and state estimation.

References

[1] Maybeck, Peter S. Stochastic models, estimation, and control. Academic press, 1982.

[2] Chen, Zhe. "Bayesian filtering: From Kalman filters to particle filters, and beyond." Statistics 182.1 (2003): 1-69.

[3] A. Doucet, N. de Freitas, and N. Gordon, Eds., Sequential Monte Carlo Methods in Practice. New York, NY, USA: Springer-Verlag New York, Inc., 2001.

[4] "Projectile motion" Wikipedia, The Free Encyclopedia, Wikimedia Foundation, [Online]. Available: https://en.wikipedia.org/wiki/Projectile_motion [Accessed: June 14, 2023]

[5] G. Welch, G. Bishop, "An introduction to the Kalman filter." (1995): 2.

[6] "Kalman Filter." Wikipedia, The Free Encyclopedia, Wikimedia Foundation, [Online]. Available: https://en.wikipedia.org/wiki/Kalman_filter#Details [Accessed: June 25, 2023]

[7] "FilterPy Documentation: KalmanFilter Class," FilterPy documentation. [Online]. Available: https://filterpy.readthedocs.io/en/latest/kalman/KalmanFilter.html. [Accessed: June 17, 2023].

[8] Dimitrov, Martin. "Particle Filters." STOR-i Student Sites, 14 May 2021, Available: www.lancaster.ac.uk/stor-i-student-sites/martin-dimitrov/2021/05/14/particle-filters/.

[9] Kim, Phil Young and Lynn Huh. "Kalman Filter for Beginners: with MATLAB Examples." (2011).

[10] F. Deinzer. "Reasoning and Descision Making under Uncertainty" Lecture Slides. Available: https://elearning.fhws.de/course/view.php?id=23366