A Neural-Network-Based Gaussian Nonlinear Filter

Felipe Giraldo-Grueso*, Andrey A. Popov† and Renato Zanetti‡
The University of Texas at Austin, Austin, TX 78712

Nonlinear estimation scenarios often present challenges for current state-of-the-art filtering techniques, potentially resulting in issues such as elevated computational costs and divergence problems. This paper introduces a new and improved learning framework designed for a neural-network-based Gaussian nonlinear filter, which demonstrates promising improvements in terms of consistency and accuracy when compared to prior approaches and other Gaussian filters. In this approach, a neural network is used to approximate a nonlinear measurement update equation, while the estimated state uncertainty is characterized using two different techniques: the unscented transform and Monte Carlo sampling. Testing is conducted using the Lorenz '96 equations and a linear measurement model, indicating better performance and improved consistency in comparison to the Unscented Kalman filter and the Gaussian particle filter. The performance of the filter is also evaluated under a nonlinear measurement model, showing consistent results that suggest reasonable generalization capabilities.

I. Introduction

Estimating quantities of interest from dynamic models and noisy measurements can be particularly challenging in nonlinear problems. If the system in question is linear and Gaussian, that is, linear dynamics and measurement models driven by additive Gaussian noise, the optimal estimator is the Kalman filter [1]. In practice, most dynamics and measurement models are typically nonlinear. In such cases, various adaptations of the Kalman filter exist to account for the nonlinearities present in the models. The extended Kalman filter (EKF) is a widely used linearized filter for nonlinear problems. The EKF linearizes the dynamics and measurement models using a first-order Taylor series expansion about the current state estimate [2]. However, neglecting higher-order terms in the Taylor expansions of the dynamics and measurement models can lead to sub-optimal filtering performance. The unscented Kalman filter (UKF) addresses this limitation by using the unscented transform [3] to retain higher-order terms. Both filters assume a linear measurement update equation based on the linear minimum mean squared error (LMMSE) formulation [1, 3].

Filters that incorporate the measurement in a nonlinear fashion typically aim to approximate the complete posterior distribution. Examples of these filters include particle filters [4] and Gaussian sum filters [5]. These filters can sometimes be more computationally expensive but can provide better accuracy when the dynamics or measurement models are nonlinear. Rather than approximating arbitrary distributions, it is also possible to assume them to be Gaussian, thus providing conservative approximations. The conservatism to this approximation comes from the fact that the Gaussian distribution maximizes the entropy over all distributions with a given covariance [6, 7].

Although assuming a Gaussian distribution can often result in a less accurate estimate, including the measurement nonlinearly can generally lead to an improvement in the accuracy of the estimate. The Gaussian particle filter (GPF) is a popular implementation for nonlinear estimation among Gaussian filters. This filter implements a re-sampling step of the particles by making a Gaussian assumption, which allows for a more conservative distribution constraint while preserving particle diversity [8]. Other nonlinear filters under Gaussian assumptions include the Gaussian high-order polynomial update filter, which approximates the posterior probability density function using polynomial functions of a Gaussian random vector, accompanied by a polynomial measurement update [9]. Recently, neural-network-based filters have been proposed to tackle nonlinear estimation problems. Yun and Zanetti [10] proposed a nonlinear estimator (ANNF) that approximates the posterior distribution as Gaussian while using a neural network to approximate a nonlinear measurement update equation.

The main contribution of this work is a new learning framework and training strategy for the nonlinear estimator based on neural networks, resulting in a more consistent filter. By incorporating the diagonal elements and correlation coefficients of the covariance matrix into the input vector of the neural network, an improvement in filtering performance

*Ph.D. Student, Department of Aerospace Engineering and Engineering Mechanics.
†Postdoctoral Fellow, Oden Institute for Computational Engineering and Sciences.
‡Associate Professor, Department of Aerospace Engineering and Engineering Mechanics, AIAA Associate Fellow.

is obtained. The proposed filter, referred to as CovNNF, is tested using the Lorenz '96 [11, 12] model and evaluated under two different uncertainty quantification (UQ) techniques.

II. Bayesian Estimation with Neural Networks

Nonlinear filters aim to establish a nonlinear function with respect to the measurement, to map the prior state estimate to the posterior estimate. Neural networks are known to be universal function estimators [13], making them a valuable tool for learning this mapping. Consequently, the approach presented in this work involves leveraging the function approximation capabilities of neural networks to accurately approximate the relationship between the prior state estimate and the posterior estimate. Note that the most computationally intensive task (i.e. training the network) is performed offline, making the filter computationally efficient for online use.

A. Learning Framework

Consider a system being driven by the following dynamics (f) and measurement (h) models:

$$\boldsymbol{x}_k = f\left(\boldsymbol{x}_{k-1}, \boldsymbol{\eta}_{k-1}\right), \tag{1}$$

$$\boldsymbol{y}_k = h\left(\boldsymbol{x}_k, \boldsymbol{w}_k\right), \tag{2}$$

where $\boldsymbol{x}_k$ is the true state at time step k, $\boldsymbol{x}_{k-1}$ is the true state at time step $k-1$, $\boldsymbol{y}_k$ is the measurement at time step k, and $\boldsymbol{\eta}_{k-1}$ and $\boldsymbol{w}_k$ represent uncertainty in the state and measurement respectively. Following the minimum mean squared error formulation (MMSE) [1], the linear measurement update equation is a function of the posterior estimated state $\left(\hat{\boldsymbol{x}}_{k-1}^{+}\right)$ and covariance $\left(P_{k-1}^{+}\right)$ from the previous time step, as well as the prior estimate $\left(\hat{\boldsymbol{x}}_{k}^{-}\right)$ and innovations $(\boldsymbol{\nu}_k)$ at the current time step, where the prior estimate and innovations are defined as:

$$\hat{\boldsymbol{x}}_k^{-} = f\left(\hat{\boldsymbol{x}}_{k-1}^{+}, \boldsymbol{0}_{n_v \times 1}\right), \tag{3}$$

$$\boldsymbol{\nu}_k = \boldsymbol{y}_k - h\left(\hat{\boldsymbol{x}}_k^{-}, \boldsymbol{0}_{n_z \times 1}\right), \tag{4}$$

with n_v and n_z representing the size of the process and measurement noise respectively. Being consistent with the MMSE formulation, to approximate a nonlinear measurement update equation, the input to the neural network $(\boldsymbol{i}_k)$ is defined as:

$$\boldsymbol{i}_k = \left[\hat{\boldsymbol{x}}_k^{-\mathrm{T}} \quad \operatorname{diag}\left[P_{k-1}^{+}\right]^{\mathrm{T}} \quad \overbrace{\operatorname{corrcoef}\left[P_{k-1}^{+}\right]^{\mathrm{T}}}^{\text{new addition}} \quad \boldsymbol{\nu}_k{}^{\mathrm{T}}\right]^{\mathrm{T}}, \tag{5}$$

where $\operatorname{diag}\left[P_{k-1}^{+}\right]$ are the diagonal elements of the posterior covariance matrix at the previous time-step and $\operatorname{corrcoef}\left[P_{k-1}^{+}\right]$ are the correlation coefficients of this matrix. This input vector represents all the non-redundant information available at the time step k. It is important to note that the diagonal and correlation coefficients of the covariance matrix are used as inputs to include all information without using the full matrix as an input. In previous versions of this filter, the posterior estimated state at time step $k-1$ was included in the input vector, but for this work, including this state resulted to be unnecessary. Additionally, for the previous design, the correlation coefficients of the covariance matrix were not included in the input vector. Instead, either the Cholesky factorization of the covariance matrix or the diagonal elements of the minimum diagonal covariance matrix were used [10]. These approaches yielded filters that were either too smug or too conservative.

Rather than predicting the posterior estimate (which is unknown), the output of the network $(\boldsymbol{o}_k)$ is designated as the difference between the prior estimate $(\hat{\boldsymbol{x}}_k^{-})$ and the true state $(\boldsymbol{x}_k)$ at the current time, i.e.:

$$\boldsymbol{o}_k = \boldsymbol{x}_k - \hat{\boldsymbol{x}}_k^{-}. \tag{6}$$

With this, the posterior estimate is defined as:

$$\hat{\boldsymbol{x}}_k^{+} = \hat{\boldsymbol{x}}_k^{-} + \boldsymbol{o}_k. \tag{7}$$

B. Uncertainty Quantification

Since the network only outputs a correction to the prior estimate, different techniques can be used to quantify the uncertainty in the posterior estimate for online use. Consider the network as a nonlinear function such that:

$$\boldsymbol{o}_k = \mathcal{N}(\boldsymbol{i}_k), \tag{8}$$

with $\boldsymbol{i}_k$ representing the input vector as defined in (5), $\mathcal{N}$ the neural network, and $\boldsymbol{o}_k$ the output vector defined in (6). Calculating the probability density function of $\boldsymbol{o}_k$ can often be intractable. Regardless, Monte Carlo (MC) methods can best approximate this probability density function [14]. These methods involve drawing a large number (N) of random samples from the known distribution of $\boldsymbol{i}_k$ and then propagating them through the nonlinear function to obtain samples from the distribution of $\boldsymbol{o}_k$. As the number of samples increases, the results converge to the true distribution of $\boldsymbol{o}_k$. To use MC sampling to quantify the uncertainty in the posterior estimate, the following procedure is used. First, N samples are drawn from the assumed Gaussian distribution of $\hat{\boldsymbol{x}}_{k-1}^{+}$, such that:

$$\hat{\boldsymbol{X}}_{k-1}^{+} = \left\{\hat{\boldsymbol{x}}_{k-1}^{+(1)}, \hat{\boldsymbol{x}}_{k-1}^{+(2)}, \cdots, \hat{\boldsymbol{x}}_{k-1}^{+(N)}\right\}, \tag{9}$$

where the superscript in parenthesis represents the sample number. Next, each sample is propagated using the dynamics model to obtain a set of prior estimates:

$$\begin{aligned} \hat{\boldsymbol{X}}_{k}^{-} &= \left\{f\left(\hat{\boldsymbol{x}}_{k-1}^{+(1)}, \boldsymbol{\eta}_{k-1}^{(1)}\right), f\left(\hat{\boldsymbol{x}}_{k-1}^{+(2)}, \boldsymbol{\eta}_{k-1}^{(2)}\right), \cdots, f\left(\hat{\boldsymbol{x}}_{k-1}^{+(N)}, \boldsymbol{\eta}_{k-1}^{(N)}\right)\right\}, &(10)\\ &= \left\{\hat{\boldsymbol{x}}_{k}^{-(1)}, \hat{\boldsymbol{x}}_{k}^{-(2)}, \cdots, \hat{\boldsymbol{x}}_{k}^{-(N)}\right\}, &(11)\end{aligned}$$

where each $\boldsymbol{\eta}_{k-1}^{(i)}$ represents a different process noise realization. The set of prior estimates is used to construct a set of innovations:

$$\begin{aligned} \boldsymbol{N}_{k} &= \left\{\boldsymbol{y}_k - h\left(\hat{\boldsymbol{x}}_{k}^{-(1)}, \boldsymbol{w}_{k}^{(1)}\right), \boldsymbol{y}_k - h\left(\hat{\boldsymbol{x}}_{k}^{-(2)}, \boldsymbol{w}_{k}^{(2)}\right), \cdots, \boldsymbol{y}_k - h\left(\hat{\boldsymbol{x}}_{k}^{-(N)}, \boldsymbol{w}_{k}^{(N)}\right)\right\}, &(12)\\ &= \left\{\boldsymbol{\nu}_{k}^{(1)}, \boldsymbol{\nu}_{k}^{(2)}, \cdots, \boldsymbol{\nu}_{k}^{(N)}\right\}, &(13)\end{aligned}$$

with the variable $\boldsymbol{w}_{k}^{(i)}$ denoting different measurement noise realizations. With a set of posterior estimates at time step $k-1$ and set of prior estimates and innovations at time step k, a set of inputs to the neural network can be assembled. The i-th sample for the set of inputs is given by:

$$\boldsymbol{i}_{k}^{(i)} = \left[\hat{\boldsymbol{x}}_{k}^{-(i)\mathrm{T}} \quad \operatorname{diag}\left[\hat{P}_{k-1}^{+}\right]^{\mathrm{T}} \quad \operatorname{corrcoef}\left[\hat{P}_{k-1}^{+}\right]^{\mathrm{T}} \quad \boldsymbol{\nu}_{k}^{(i)\mathrm{T}}\right]^{\mathrm{T}}. \tag{14}$$

Notice that the covariance-related terms (the diagonal and correlation coefficients) remain the same for all samples in the set. The set of inputs is then propagated through the neural network to obtain a set of posterior estimates, where:

$$\begin{aligned} \hat{\boldsymbol{X}}_{k}^{+} &= \left\{\hat{\boldsymbol{x}}_{k}^{-(1)} + \mathcal{N}\left(\boldsymbol{i}_{k}^{(1)}\right), \hat{\boldsymbol{x}}_{k}^{-(2)} + \mathcal{N}\left(\boldsymbol{i}_{k}^{(2)}\right), \cdots, \hat{\boldsymbol{x}}_{k}^{-(N)} + \mathcal{N}\left(\boldsymbol{i}_{k}^{(N)}\right)\right\}, &(15)\\ &= \left\{\hat{\boldsymbol{x}}_{k}^{-(1)} + \boldsymbol{o}_{k}^{(1)}, \hat{\boldsymbol{x}}_{k}^{-(2)} + \boldsymbol{o}_{k}^{(2)}, \cdots, \hat{\boldsymbol{x}}_{k}^{-(N)} + \boldsymbol{o}_{k}^{(N)}\right\}, &(16)\\ &= \left\{\hat{\boldsymbol{x}}_{k}^{+(1)}, \hat{\boldsymbol{x}}_{k}^{+(2)}, \cdots, \hat{\boldsymbol{x}}_{k}^{+(N)}\right\}. &(17)\end{aligned}$$

Finally, the set can be reduced by calculating the sample mean and covariance, such that:

$$\hat{\boldsymbol{x}}_{k}^{+} = \frac{1}{N}\sum_{i=1}^{N}\hat{\boldsymbol{x}}_{k}^{+(i)}, \tag{18}$$

$$P_{k}^{+} = \frac{\alpha}{N-1}\sum_{i=1}^{N}\left[\hat{\boldsymbol{x}}_{k}^{+(i)} - \hat{\boldsymbol{x}}_{k}^{+}\right]\left[\hat{\boldsymbol{x}}_{k}^{+(i)} - \hat{\boldsymbol{x}}_{k}^{+}\right]^{\mathrm{T}}, \tag{19}$$

where α is a constant slightly greater than one to account for the fact that the posterior covariance estimate is itself a random variable. Although effective, Monte Carlo methods are usually expensive, requiring numerous samples to converge to the true distribution. In this sense, less expensive algorithms such as the unscented transform (UT) can be used to approximate the first two moments of the probability distribution function of $\boldsymbol{o}_k$. In the UT, a small set of so-called sigma points, chosen deterministically, are propagated through the nonlinear mapping. The statistics of these propagated sigma points can then be used to estimate the nonlinear mean and covariance [3]. To use the UT to quantify the uncertainty in the posterior estimate, few changes have to be done to the procedure followed for MC sampling. First, the posterior estimate and covariance are augmented to include the process noise and the measurement noise:

$$\hat{\boldsymbol{x}}_{a,k-1}^{+} = \begin{bmatrix} \hat{\boldsymbol{x}}_{k-1}^{+\mathrm{T}} & \mathbf{0}_{n_v \times 1}^{\mathrm{T}} & \mathbf{0}_{n_z \times 1}^{\mathrm{T}} \end{bmatrix}^{\mathrm{T}}, \tag{20}$$

$$P_{a,k-1}^{+} = \begin{bmatrix} P_{k-1}^{+} & 0_{n_x \times n_v} & 0_{n_x \times n_z} \\ 0_{n_v \times n_x} & Q & 0_{n_v \times n_z} \\ 0_{n_z \times n_x} & 0_{n_z \times n_v} & R \end{bmatrix}, \tag{21}$$

where n_x is the size of the state vector, Q and R represent the process noise and measurement noise covariances respectively. To build the set of posterior estimates at time step $k-1$, the following equations are used:

$$\boldsymbol{\mathcal{X}}^{(0)} = \hat{\boldsymbol{x}}_{a,k-1}^{+}, \tag{22}$$

$$\boldsymbol{\mathcal{X}}^{(i)} = \hat{\boldsymbol{x}}_{a,k-1}^{+} + \left(\sqrt{(L+\lambda)\, P_{a,k-1}^{+}}\right)^{(i)}, \quad i = 1, ..., L, \tag{23}$$

$$\boldsymbol{\mathcal{X}}^{(i)} = \hat{\boldsymbol{x}}_{a,k-1}^{+} - \left(\sqrt{(L+\lambda)\, P_{a,k-1}^{+}}\right)^{(i)}, \quad i = L+1, ..., 2L, \tag{24}$$

resulting in a set of $2L+1$ sigma points. The scaling parameter λ is defined as $\lambda = \alpha^2 (L+\kappa) - L$, where α and κ are secondary scaling parameters and $L = n_x + n_v + n_z$. The superscript (i) denotes the i-th column of the matrix in parenthesis. The parameter α controls the spread of points around $\hat{\boldsymbol{x}}_{a,k-1}^{+}$ and is typically set to a small positive value ($10^{-4} \leq \alpha \leq 1$), and κ is usually chosen as $3 - L$ [3]. The set of posterior estimates at time step $k-1$ is given by:

$$\hat{\boldsymbol{\mathcal{X}}}_{k-1}^{+(i)} = \boldsymbol{\mathcal{X}}^{(i)}(1 : n_x), \tag{25}$$

where the notation $\boldsymbol{\mathcal{X}}^{(i)}(a : b)$ indicates the selection of elements a to b of the vector $\boldsymbol{\mathcal{X}}^{(i)}$. The set of posterior estimates is propagated using the dynamics model to obtain a set of prior estimates at time step k, such that:

$$\hat{\boldsymbol{\mathcal{X}}}_{k}^{-(i)} = f\left(\hat{\boldsymbol{\mathcal{X}}}_{k-1}^{+(i)}, \boldsymbol{\mathcal{X}}^{(i)}(n_x + 1 : n_x + n_v)\right). \tag{26}$$

And the set of innovations is constructed as:

$$\boldsymbol{N}_{k}^{(i)} = \boldsymbol{y}_k - h\left(\hat{\boldsymbol{\mathcal{X}}}_{k}^{-(i)}, \boldsymbol{\mathcal{X}}^{(i)}(n_x + n_v + 1 : n_x + n_v + n_z)\right). \tag{27}$$

Once the sets have been obtained, the set of inputs can be assembled in the same way as in (14) and propagated through the neural network. Finally, the posterior estimate and covariance are calculated using a weighted average based on the spread of the sigma points.

$$\hat{\boldsymbol{x}}_{k}^{+} = \sum_{i=0}^{2L} w_m^{(i)} \hat{\boldsymbol{x}}_{k}^{+(i)}, \tag{28}$$

$$P_{k}^{+} = \sum_{i=0}^{2L} w_c^{(i)} \left[\hat{\boldsymbol{x}}_{k}^{+(i)} - \hat{\boldsymbol{x}}_{k}^{+}\right]\left[\hat{\boldsymbol{x}}_{k}^{+(i)} - \hat{\boldsymbol{x}}_{k}^{+}\right]^{\mathrm{T}}, \tag{29}$$

where the weights are given by [15]:

$$w_m^{(0)} = \frac{\lambda}{L+\lambda}, \tag{30}$$

$$w_c^{(0)} = \frac{\lambda}{L+\lambda} + 1 - \alpha^2 + \beta, \tag{31}$$

$$w_m^{(i)} = w_c^{(i)} = \frac{1}{2(L+\lambda)}, \quad i = 1, ..., 2L. \tag{32}$$

The constant β is used to incorporate prior knowledge of the probability distribution. Under the assumption of a scalar and Gaussian prior distribution, $\beta = 2$ is usually regarded as optimal since it can cancel second-order error terms [15]. However, it is important to note that even for scalar priors, $\beta = 2$ may not always be optimal, as its effectiveness highly depends on the nonlinear function (in this case, the neural network), so tuning β should consider both the prior distribution and the function. [16]

C. Online use

As discussed earlier, once the neural network has undergone offline training, it becomes suitable for online estimation. By shifting the computationally intensive tasks to the offline phase, the trained model can be used efficiently for real-time state estimation. The computational cost of a filtering algorithm during online operations plays a significant role in achieving optimal performance and facilitating autonomous decision-making. A computationally efficient filter can enable real-time data processing and help minimize bandwidth usage for effective communication [17]. In this regard, the offline training of the network contributes to a fast and accurate filter when deployed for online use. To make use of the trained neural network, the next algorithm can be followed:

Algorithm 1 Online state estimation using the trained neural network

```
1:  Initialize state estimate and covariance $(\hat{\boldsymbol{x}}_0^+, P_0^+)$
2:  for $k = 1, 2, \ldots$ do
3:      Draw $2L + 1$ or $N$ samples from the approximated distribution of $(\hat{\boldsymbol{x}}_{k-1}^+, P_{k-1}^+)$ using the UT or MC sampling, respectively:
            $$\hat{\boldsymbol{X}}_{k-1}^+ = \left\{\hat{\boldsymbol{x}}_{k-1}^{+(1)}, \hat{\boldsymbol{x}}_{k-1}^{+(2)}, \cdots, \hat{\boldsymbol{x}}_{k-1}^{+(N)}\right\}$$
4:      Propagate each sample using the forward dynamics to obtain a set of prior estimates:
            $$\hat{\boldsymbol{X}}_k^- = \left\{\hat{\boldsymbol{x}}_k^{-(1)}, \hat{\boldsymbol{x}}_k^{-(2)}, \cdots, \hat{\boldsymbol{x}}_k^{-(N)}\right\}$$
5:      Use the measurement at time $k$ to form a set of innovations:
            $$\boldsymbol{N}_k = \left\{\boldsymbol{\nu}_k^{(1)}, \boldsymbol{\nu}_k^{(2)}, \cdots, \boldsymbol{\nu}_k^{(N)}\right\}$$
6:      Arrange the samples to create an set of inputs to the network where:
            $$\boldsymbol{i}_k^{(i)} = \left[\hat{\boldsymbol{x}}_k^{-(i)\mathrm{T}} \quad \mathrm{diag}\left[P_{k-1}^+\right]^{\mathrm{T}} \quad \mathrm{corrcoef}\left[P_{k-1}^+\right]^{\mathrm{T}} \quad \boldsymbol{\nu}_k^{(i)\mathrm{T}}\right]^{\mathrm{T}}$$
7:      Propagate each vector through the neural network to obtain a set of posterior estimates:
            $$\hat{\boldsymbol{X}}_k^+ = \left\{\hat{\boldsymbol{x}}_k^{-(1)} + \boldsymbol{o}_k^{(1)}, \hat{\boldsymbol{x}}_k^{-(2)} + \boldsymbol{o}_k^{(2)}, \cdots, \hat{\boldsymbol{x}}_k^{-(N)} + \boldsymbol{o}_k^{(N)}\right\}$$
8:      Reduce the set to obtain a posterior estimate and covariance $(\hat{\boldsymbol{x}}_k^+, P_k^+)$ using the UQ technique selected.
9:      Update state for next iteration $(\hat{\boldsymbol{x}}_{k-1}^+, P_{k-1}^+) \leftarrow (\hat{\boldsymbol{x}}_k^+, P_k^+)$.
10: end for
```

Figure 1 shows a flowchart of the online state estimation using the trained neural network. The algorithm adheres to a fundamental structure: initial samples are acquired, undergo propagation through the dynamics, and subsequently, the neural network processes these samples to generate updated samples. These updated samples are then reduced to derive a state estimate along with its corresponding covariance.

D. Filter Consistency

A filter is considered consistent when estimating a constant parameter if its estimate converges to the true parameter. However, when estimating a dynamic state, it is not always possible for the estimate to converge to the true state. In such cases, a filter is considered consistent if the following conditions are met [1]:

- The errors in the state estimate are zero-mean and have a magnitude that is consistent with the state covariance estimated by the filter.
- The innovations, or the differences between the predicted and measured states, also have zero mean and are considered white, meaning they are uncorrelated over time.

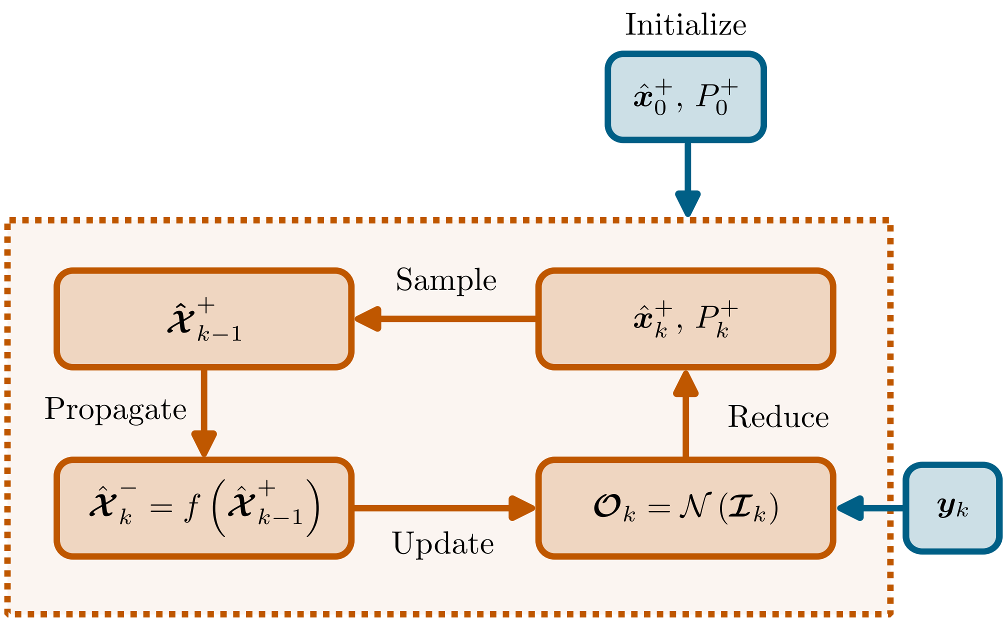
Initialize
$\hat{x}_0^+, P_0^+$
Sample
$\hat{\mathcal{X}}_{k-1}^+$
$\hat{x}_k^+, P_k^+$
Propagate
Reduce
$\hat{\mathcal{X}}_k^- = f\left(\hat{\mathcal{X}}_{k-1}^+\right)$
$\mathcal{O}_k = \mathcal{N}(\mathcal{I}_k)$
y_k
Update

Fig. 1 Flowchart of the online state estimation algorithm using the trained neural network.

III. Data Generation

Typically, training a neural network requires a large number of samples. In this case, since the nonlinear measurement update equation is an unknown function, there is no way to generate training data that maps the prior estimate to the posterior. However, the forward mapping, where measurements can be directly calculated from the state, is often known. By using this known function to map states to measurements, it is possible to generate enough training data to train a neural network offline to approximate a nonlinear measurement update equation.

In this work, the Lorenz '96 system is used to generate training data and, after training, test the network in an online fashion. This system is widely used as a test case for new data assimilation and forecasting techniques [11, 12], described by a set of coupled ordinary differential equations that model the transfer properties of an atmospheric quantity. The model accounts for advection, dissipation, and external forcing and is defined as follows [10]:

$$\frac{d}{dt}x_i(t) = x_{i-1}(t)\left[x_{i+1}(t) - x_{i-2}(t)\right] - x_i(t) + F, \tag{33}$$

with the cyclic boundary conditions $x_{-1} = x_{n-1}$, $x_0 = x_n$, $x_1 = x_{n+1}$, and F representing the external forcing term. For this work, a 4-dimensional vector is used, where $\boldsymbol{x}(t) = [x_1(t), x_2(t), x_3(t), x_4(t)]$. The measurement model, used to simulate possible measurements, is defined as a discrete linear combination of the elements of the state vector with odd indices such that:

$$\boldsymbol{y}_k = H\boldsymbol{x}_k + \boldsymbol{w}_k, \quad H = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix}, \tag{34}$$

where $\boldsymbol{w} = [w_{k,1}, w_{k,2}]$ represents the measurement noise. The training data generation process, as illustrated in Figure 2, involves propagating various trajectories over time. As the true state evolves through time, a prior estimate is stored by randomly sampling a point from a Gaussian distribution with the true state as the mean and a selected covariance. At each time step, the true state, prior estimate, covariance, and innovations are recorded.

For each trajectory, the true state was initiated at $\boldsymbol{x}_0 = [F, F, F + 0.01, F]$, bounded by an initial covariance of $P_0 = 14I_{4\times4}$. To choose the covariance of each point, the diagonal of the matrix was randomly chosen from a Gamma distribution in the range of $P_{\text{min}} = 10^{-1}$ and $P_{\text{max}} = 14$. The upper value was obtained by calculating the auto-covariance of one single trajectory, and the lower value was manually tuned until arriving at a satisfactory performance of the trained network. After setting the diagonal of the covariance matrix, a random correlation matrix was generated by using vine theory [18], thus generating a full covariance matrix. Each trajectory was simulated for a duration of 40 time units, with a time step of 0.5 time units, using a Runge-Kutta 4-5 integrator [19]. Two measurements were obtained per time unit using the measurement model outlined in (34). It is essential to note that both process and measurement noise were assumed to be Gaussian, uncorrelated and white with zero mean and covariance matrices of $Q = 10^{-6} I_{4\times4}$ and $R = I_{2\times2}$, respectively. The forcing term was set to $F = 14$. This choice was informed by the observation that it represents the minimal threshold for inducing chaotic dynamics in a four-variable state within the context of the Lorenz '96 problem, thus yielding a more challenging filtering problem. In total, the model was simulated for 10^3 distinct trajectories, resulting in a training dataset comprising 8×10^4 instances. This new data generation scheme results in a reduced requirement for training points when compared to prior methods [10].

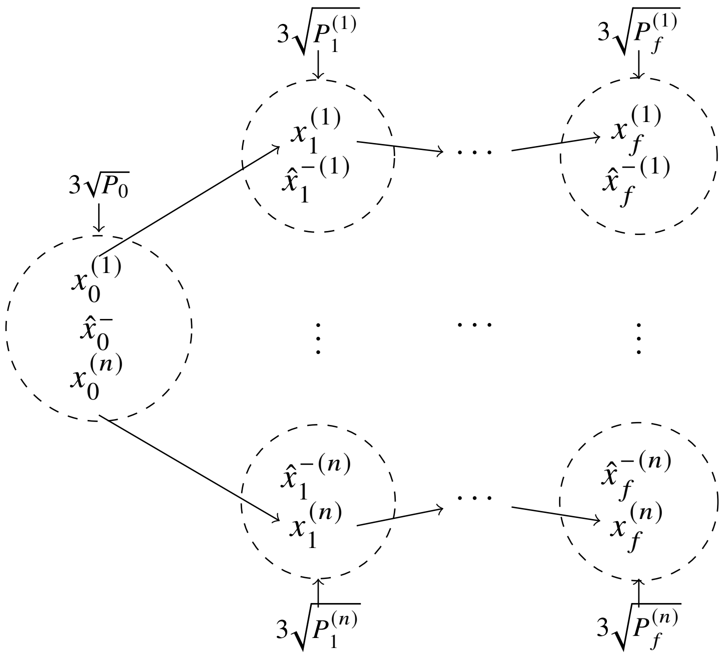

Fig. 2 Training data generation. Figure adapted from Yun and Zanetti [10].

IV. Training and Testing

A. Training

The network architecture used for this work, depicted in Figure 3, consists of two hidden layers, each with 100 neurons and using a hyperbolic tangent activation function. The operations performed by the network are described by the following equations:

$$\begin{aligned} \boldsymbol{a}_k^{(1)} &= \tanh\left\{W^{(1)}\boldsymbol{i}_k + \boldsymbol{b}_k^{(1)}\right\}, \\ \boldsymbol{a}_k^{(2)} &= \tanh\left\{W^{(2)}\boldsymbol{a}_k^{(1)} + \boldsymbol{b}_k^{(2)}\right\}, \\ \boldsymbol{o}_k &= W^{(o)}\boldsymbol{a}_k^{(2)} + \boldsymbol{b}_k^{(o)}, \end{aligned} \tag{35}$$

where $\boldsymbol{a}_k^{(1)} \in \mathbb{R}^{100}$ are the neurons in the first hidden layer, $W^{(1)} \in \mathbb{R}^{100\times16}$ is the matrix of weights for the first layer, $\boldsymbol{i}_k \in \mathbb{R}^{16}$ represents the input and $\boldsymbol{b}_k^{(1)} \in \mathbb{R}^{100}$ are the biases for the first layer. Similarly, $\boldsymbol{a}_k^{(2)} \in \mathbb{R}^{100}$ are the neurons in the second hidden layer, $W^{(2)} \in \mathbb{R}^{100\times100}$ is the matrix of weights for the second layer, and $\boldsymbol{b}_k^{(2)} \in \mathbb{R}^{100}$ are the biases for the second layer. Lastly, $\boldsymbol{o}_k \in \mathbb{R}^4$ is the output of the network, and $W^{(o)} \in \mathbb{R}^{4\times100}$ and $\boldsymbol{b}_k^{(o)} \in \mathbb{R}^4$ are the weights and biases for the output layer, respectively.

To facilitate the learning process of the network, the training data was scaled within a range of $[-1, 1]$. The mean squared error loss function was used and an Adam-based [20] optimizer was employed during training. The learning rate was dynamically adjusted based on a specific function, which is illustrated in figure 4, thus eliminating the need for manual tuning. To mitigate potential issues related to exploding gradients, the weights and biases were initialized

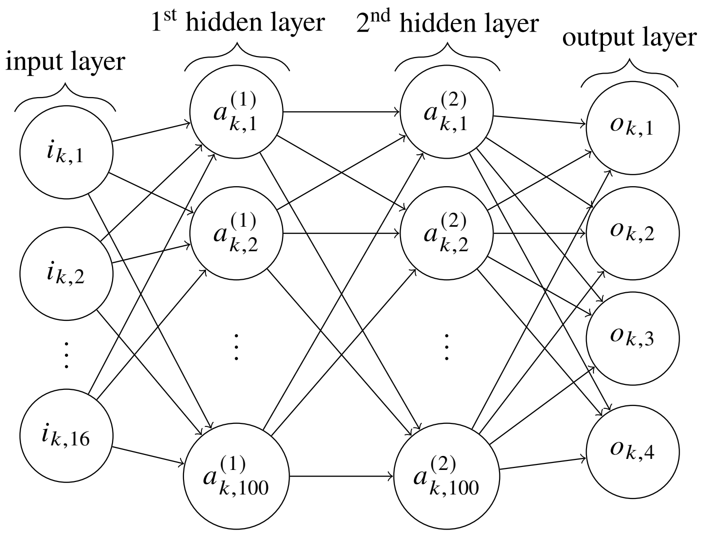

Fig. 3 Network architecture used to predict the difference between the true state and the prior estimate. The input is a 16-dimensional vector, and the output is a 4-dimensional vector.

using Xavier initialization [21]. The network was trained for 250 epochs with a mini-batch size of 2^{10}, resulting in 79 batches. It is important to note that two different networks were trained in order to understand the effects of including the correlation coefficients in the input vector as described in (5). The only difference in the second network is that the correlation coefficients were not included in the input vector. To differentiate between the two trained neural networks, the filter using the full input vector as described by (5) will be referred to as CovNNF, and the filter using the neural network without the correlation coefficients in the input will be referred to as ANNF2 (the "2" signifies that this network mirrors the concept introduced by Yun and Zanetti [10], but has improvements in the data generation and training process).

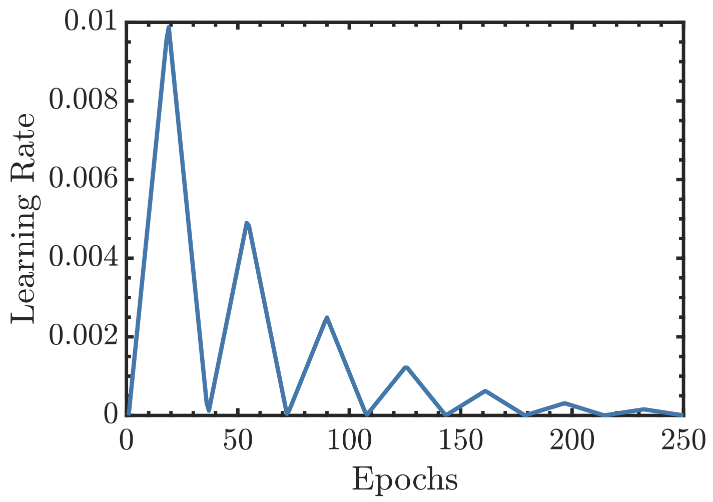

Fig. 4 Learning rate as a function of training epochs.

B. Testing

To test the trained network, an MC performance test was conducted. In this test, 1000 random scenarios following the models presented in (33) and (34) were simulated using the neural network filter as the online estimator. It is worth emphasizing that, before feeding the input vector into the network, a scaling process was applied, utilizing information derived from the training data. Additionally, the output given by the network underwent a descaling procedure to ensure consistent units with the states being estimated. To set the initial conditions for each scenario, the initial state was randomly sampled from a propagated true trajectory and the initial covariance was set to $P_0^+ = 10\, I_{4\times4}$. Just as with the training data generation, each trajectory was simulated for a duration of 40 time units, with a time step of 0.5 time units, obtaining two measurements per time unit. Both process and measurement noise were assumed to be uncorrelated, white, and Gaussian with zero mean and the same covariance matrices used for the data generation. Throughout these tests, the UT and MC sampling techniques were used to obtain the first two moments of the estimate undergoing the nonlinear mapping characterized by the neural network. The trained model was also tested on a MC performance test with a nonlinear measurement model to assess its robustness and generalization capabilities. The measurement model used for this purpose can be seen below described as an element-wise operation [22]:

$$\tilde{\boldsymbol{y}}_k = \frac{\boldsymbol{y}_k}{2}\left\{1 + \left(\frac{|\boldsymbol{y}_k|}{10}\right)^{\gamma-1}\right\} + \boldsymbol{w}_k, \tag{36}$$

where $\boldsymbol{y}_k$ denotes the output of the linear measurement model in (34) without noise. In this equation, the second term within the brackets is meant to be of approximately the same order of magnitude as the linear term to avoid problems with numerical overflow [22]. The parameter γ controls the nonlinearity of the model with $\gamma = 1$ exactly corresponding to (34).

V. Results

Figure 5 shows the training mean squared error as a function of the number of epochs for the two neural network configurations: ANNF2 and CovNNF, the latter incorporating correlation coefficients into the input vector. No validation data was used during training as overfitting could be easily detected while testing the neural network in the deployed filter. From this figure it can be seen that the network with the correlation coefficients commences training with a higher loss. However, it rapidly converges to and maintains a lower training loss compared to ANNF2 throughout the entire training process. This observation strongly suggests that the incorporation of correlation coefficients provides the network with valuable information, enhancing its ability to approximate a more accurate MMSE estimator. Nonetheless, assessing the performance of the network during online deployment solely based on this figure and the final loss values is somewhat limited. As mentioned before, to properly evaluate the performance of the network, different MC tests were performed to evaluate the network on different trajectories that were not used during training.

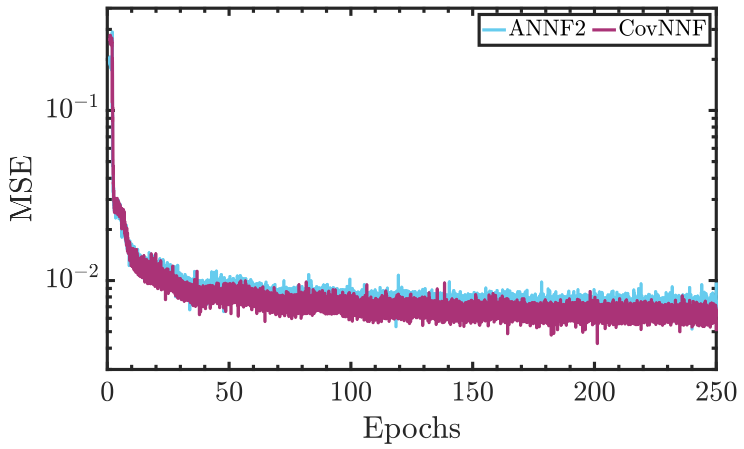

Fig. 5 Training mean squared error as a function of training epochs for CovNNF and ANNF2.

A. Linear Measurement Model

Figure 6 shows 100 MC runs out of the 1000 total runs for ANNF2, with the UT as the UQ technique used. Similarly, figure 7 presents the performance test results for CovNNF, also using the UT. For both filters, the UT parameters were tuned to maximize consistency.

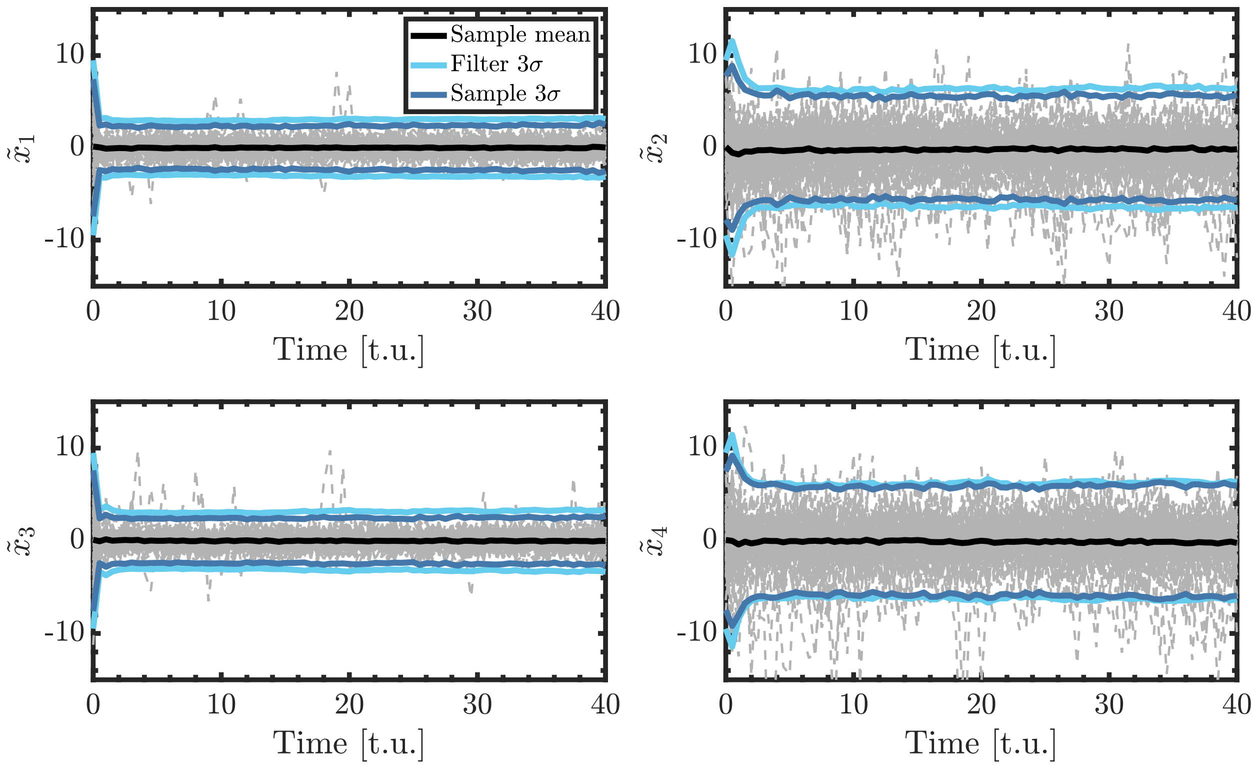

Fig. 6 Performance test for the ANNF2 for 100 Monte Carlo runs.

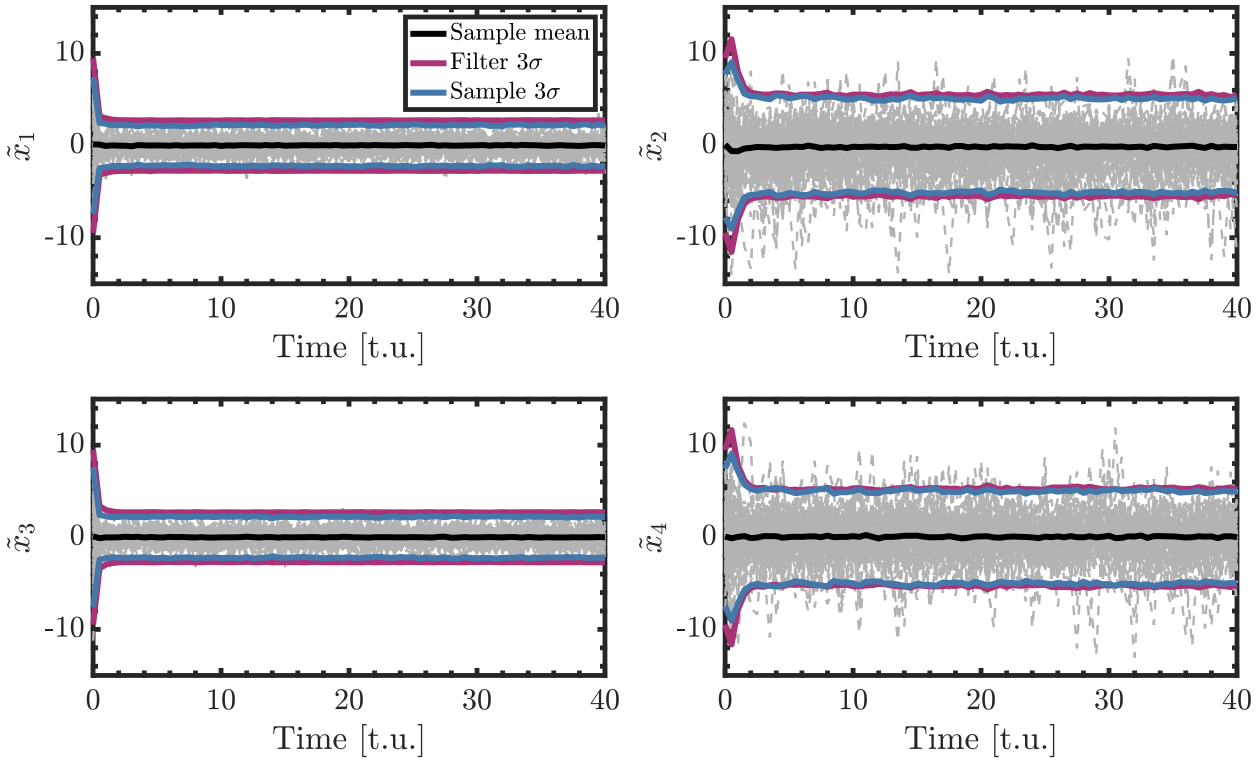

Fig. 7 Performance test for the CovNNF for 100 Monte Carlo runs.

A comparison between the results presented in figures 6 and 7 shows that both ANNF2 and CovNNF successfully predict zero-mean error estimates, as indicated by the solid black line. However, it is apparent that the ANNF2 exhibits a higher occurrence of MC runs surpassing the covariance bounds when compared to the CovNNF. As the measurement model used (34) includes sparse observations, it is evident that the estimation error for the states that are not directly measured is higher, as seen in both figures. Regardless, the results obtained with the CovNNF show that including the correlation coefficients in the neural network can yield lower estimation errors and more consistent results. To comprehensively assess the consistency of these filters, metrics such as the root sum squared (RSS) and root mean squared error (RMSE) can be used. RSS, commonly used for filter consistency evaluation, is defined as the square root of the trace of the covariance matrix, providing valuable insights into filter performance when states share similar orders of magnitude [23]. The formulas used to calculate the predicted and effective RSS are shown below.

$$\text{RSS Pred.}(k) = \sum_{j=1}^{N} \frac{1}{\mathrm{N}} \sqrt{\sum_{i=1}^{n_x} P^{+}_{i,i}(k)^{(j)}}\ , \tag{37}$$

$$\text{RSS Eff.}(k) = \sum_{j=1}^{N} \frac{1}{\mathrm{N}} \sqrt{\sum_{i=1}^{n_x} \left[x_i(k)^{(j)} - \hat{x}^{+}_i(k)^{(j)}\right]^2}, \tag{38}$$

where N is the total number MC runs and n_x is the total number of states being estimated. This metric serves as a quantification of the consistency of the filter. The objective is to minimize the disparity between the predicted RSS and the effective RSS, with the aim of bringing both metrics as close to zero as possible. Figure 8 illustrates the temporal evolution of both the effective RSS and the RSS predicted by each filter. In this figure, the unscented Kalman filter (UKF), the Gaussian particle filter (GPF), and the bootstrap particle filter (BPF) are included for comparison purposes. As mentioned previously, the UKF is a Gaussian linear filter, the GPF is a Gaussian nonlinear filter and the BPF is a sequential importance resampling (SIR) filter. Just as with the ANNF2 and the CovNNF, the UT parameters in the UKF were manually tuned until satisfactory results were obtained. For the GPF, the importance function was selected as the predictive density and the filter was run with 1500 particles. To enable a fair comparison with the other Gaussian filters, the prior estimate of the GPF was Gaussianized, yielding comparable results across all Gaussian filters. For the BPF, 1500 particles were used including a systematic resampling and regularization.

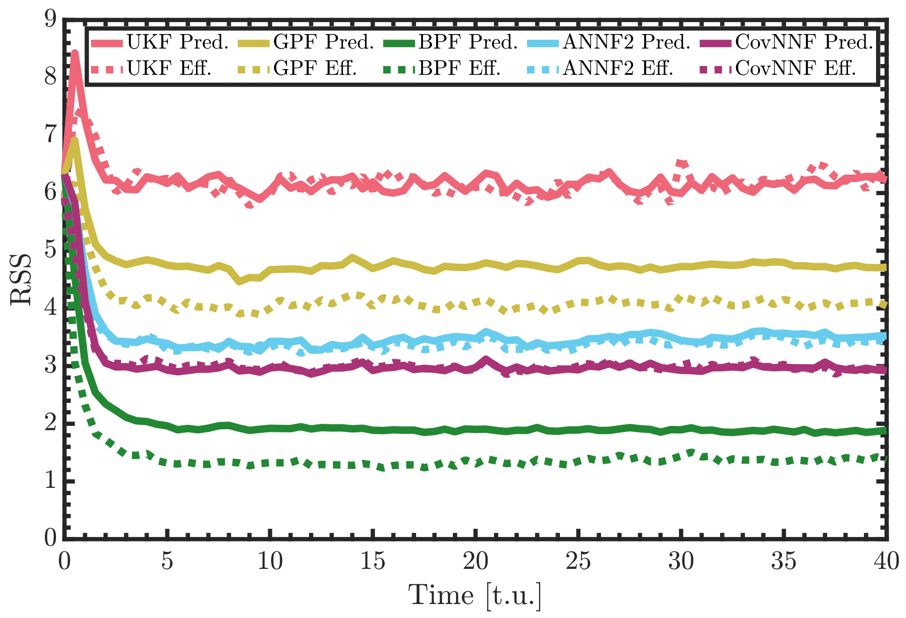

Fig. 8 Effective RSS and predicted RSS for each filter tested as a function of time.

As mentioned before, a perfectly consistent filter should exhibit a match between the dashed and solid lines [23]. Conservative filters have the solid bar above the dashed line, while smug filters have the solid bar below the dashed line. As observed in the figure, the UKF, GPF, ANNF2, and CovNNF show consistent estimates, coherent with their Gaussian assumption. The GPF, ANNF2 and CovNNF achieve lower RSS values than the UKF, attributed to their nonlinear nature, in contrast to the linear update in the UKF. The green line represents results obtained using the BPF, which also shows conservative results due to the regularization of the particles, but offers insight into the theoretical best achievable performance. These results show the superior performance of the CovNNF over the ANNF2, GPF and UKF, more closely approaching the BPF results.

Table 1 shows the time-averaged RSS, displayed as mean values with their corresponding standard deviations. This table reaffirms the closest proximity of CovNNF to the theoretical ideal represented by the BPF. Additionally, it highlights its consistent performance by achieving lower RSS predictions and accurately aligning with the effective RSS values, performing better than the ANNF2, GPF, and UKF in this context.

Table 1 Time averaged root sum squared for each of the filters tested.

Filter	Effective RSS	Predicted RSS
UKF	6.1863 ± 0.2699	6.2010 ± 0.3030
GPF	4.1510 ± 0.3467	4.7951 ± 0.3259
ANNF2	**3.4469 ± 0.3542**	**3.5370 ± 0.4385**
CovNNF	**3.0615 ± 0.5055**	**3.0676 ± 0.4093**
BPF	1.4506 ± 0.5545	2.0230 ± 0.5883

Figure 9 illustrates the temporal evolution of root mean squared error (RMSE) for each filter, defined as:

$$\mathrm{RMSE}(k) = \sum_{j=1}^{N} \frac{1}{\mathrm{N}} \sqrt{\sum_{i=1}^{n_x} \frac{\left[x_i(k)^{(j)} - \hat{x}_i^{+}(k)^{(j)}\right]^2}{n_x}}, \tag{39}$$

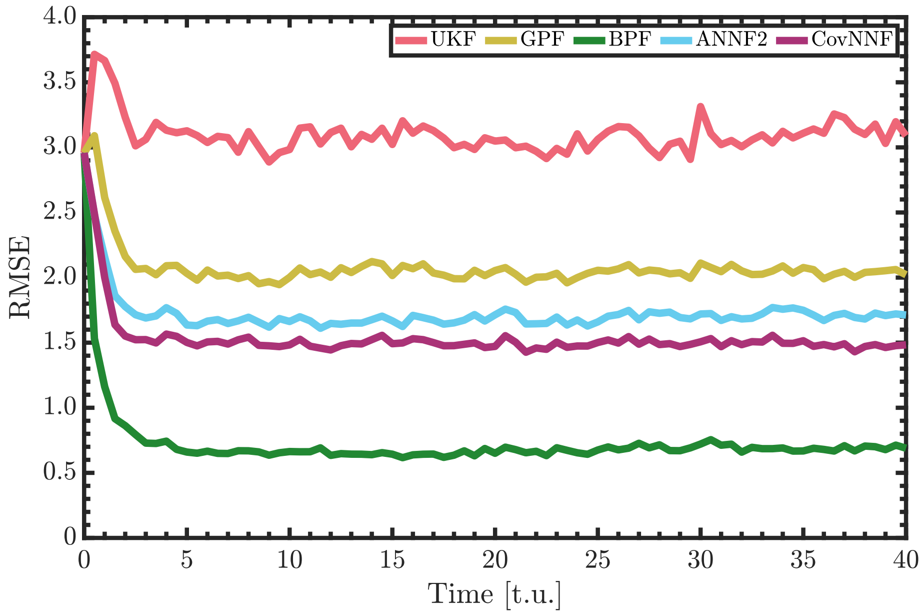

Fig. 9 RMSE for each filter tested as a function of time.

Compared to the RSS, the RMSE provides a normalized error which can be interpreted more easily in order to assess the accuracy of each filter. Similar to figure 8, the UKF exhibits the poorest performance among all the filters studied, while the CovNNF approaches the BPF results the closest. These results suggest that the nonlinearities present in the neural network leads to improved filtering performance compared to other Gaussian linear and nonlinear filters. Furthermore, using the inclusion of all information of the covariance matrix as part of the neural network input yields a more consistent and accurate filter. Table 2 displays the time-averaged RMSE values for each filter, presented as mean values with corresponding standard deviations.

Table 2 Time averaged root mean squared error for each of the filters tested.

Filter	RMSE
UKF	3.0932 ± 0.1349
GPF	2.0755 ± 0.1733
ANNF2	**1.7234 ± 0.1771**
CovNNF	**1.5338 ± 0.2047**
BPF	0.7253 ± 0.2772

Taking into account the distinct nature of these filters, which handle operations either for a limited number of sigma points or particles, it becomes important to quantify their operational speed for the evaluation of their online performance. Particle filters are known to require more computational resources, a fact made evident in table 3. This table presents the average run-time for the filters depicted in Figures 8 and 9 for a single propagation and update, essentially one time step in the simulation. As anticipated, the UKF demonstrates the most rapid performance due to its lower number of sigma points. On the other hand, ANNF2 and CovNNF operate at a slower pace but notably faster than the implemented particle filters. It is important to note that run-time can be significantly affected by the specific implementation, and the implementation used for this research did not aim to optimize the efficiency of these filters. Notwithstanding, these results show that neural-network-based filters can provide a fast solution and surpass both linear and nonlinear Gaussian counterparts in terms of efficiency and performance.

Table 3 Average run-time of one propagation and update for each filter.

Filter	Run-time (s)
UKF	0.0014 ± 0.0002
GPF	0.1091 ± 0.0123
ANNF2	**0.0109 ± 0.0013**
CovNNF	**0.0101 ± 0.0012**
BPF	0.1675 ± 0.0189

To validate the use of the UT, the same MC performance test was conducted for the CovNNF using MC sampling to estimate the uncertainty. Figure 10 illustrates the time-averaged RSS as a function of the number of samples in MC sampling. From the figure, it can be seen that as more samples are added to the MC sampling, the filter becomes more consistent as the effective RSS and RSS predicted by the filter begin to converge. The straight lines represent time-averaged RSS obtained using the UT, as seen in table 1. The results obtained with MC sampling show that using more samples can capture the nonlinearities in the neural network better than the UT, when estimating the first two moments of the resulting set. While some performance might be lost using the UT, it is important to note that the UT can run faster than MC sampling since less samples are being propagated through the dynamics, measurement model and the neural network. The UT exhibits reduced run-time, approximately three times faster compared to the optimal MC sampling (using 150 MC samples resulted in a run-time of 0.0323 ± 0.0033 seconds), while maintaining consistent results. This reduction in computational resources makes it an attractive option, over MC sampling, for onboard applications [17].

B. Nonlinear Measurement Model

To test the robustness of the trained neural network and its ability to generalize to other measurement models, a different MC performance test was conducted using the nonlinear measurement model described in (36). Figures 11 and

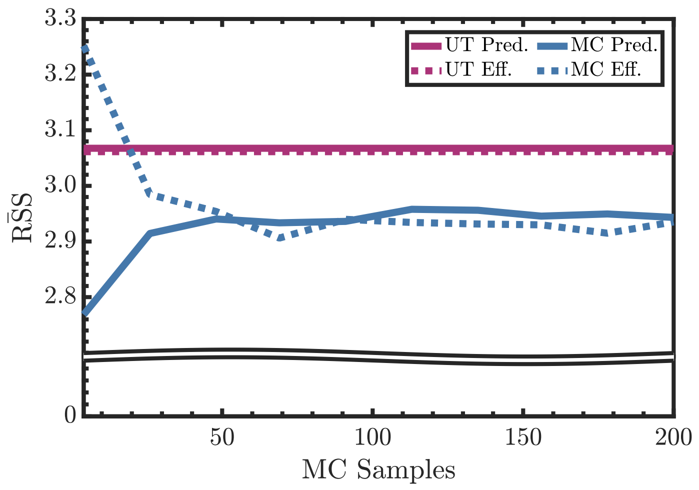

Fig. 10 Time average of the RSS as a function of the number of samples in MC sampling. Overlaid is the time average of the RSS obtained using the UT.

12 show the performance test for the ANNF2 and CovNNF, respectively. Just as with the linear measurement model, the UT parameters were tuned to maximize consistency. From these figures it can be observed that the CovNNF is capable of generalizing better to a nonlinear measurement when compared to the ANNF2, even though both networks were trained with the linear measurement model. The ANNF2 exhibits a more conservative behavior, as the predicted covariance is higher than the variance of the estimation errors. This behavior suggests that using the correlation coefficients in the input of the network can help generalize better to nonlinearities in the measurement and yield a more consistent filter.

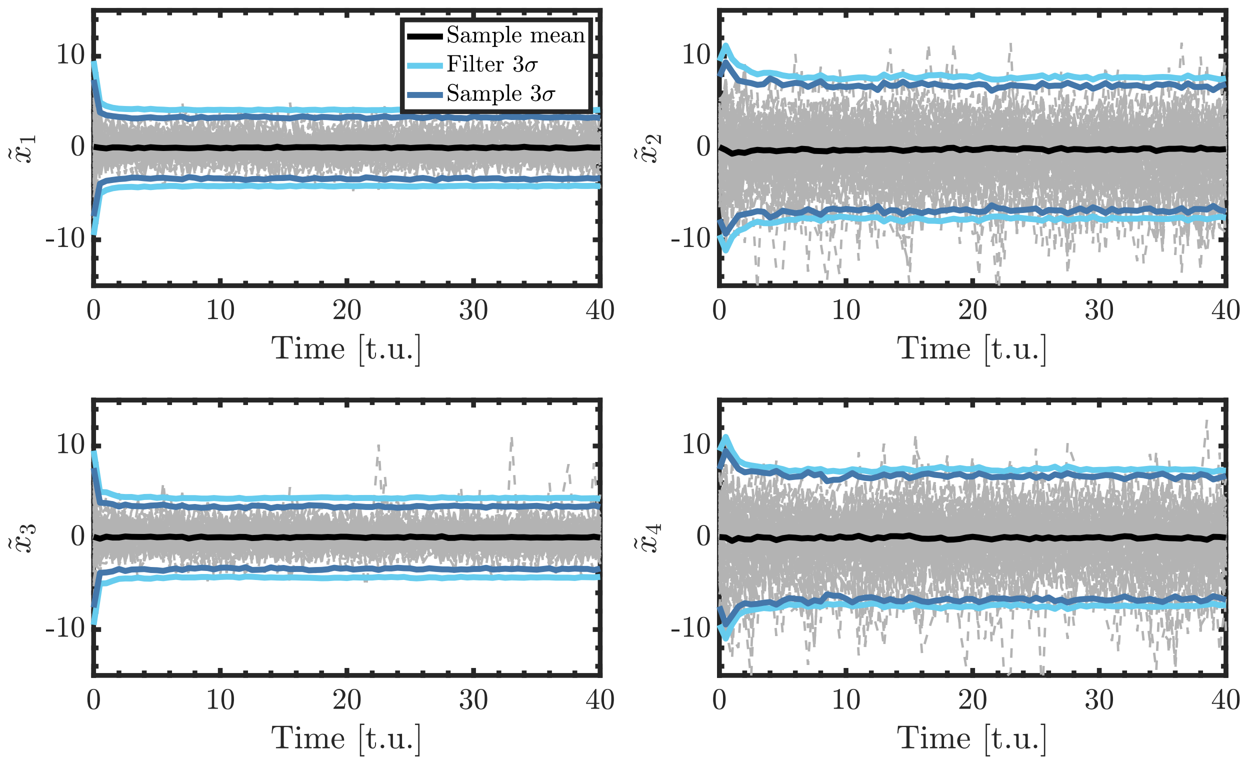

Fig. 11 Performance test for the ANNF2 for 100 Monte Carlo runs with $\gamma = 2$.

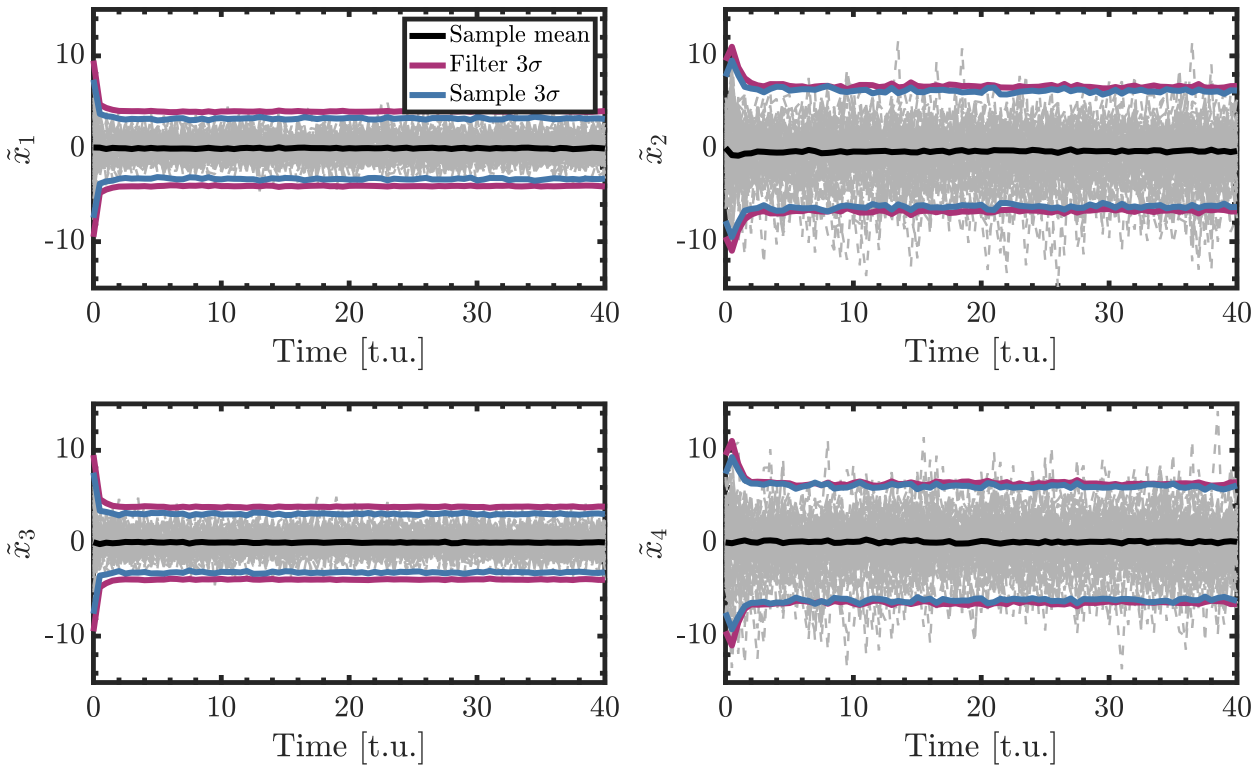

Fig. 12 Performance test for the CovNNF for 100 Monte Carlo runs with $\gamma = 2$.

Deeper understanding of this behavior can be observed in Figure 13, which shows the time evolution of both the effective RSS and the RSS predicted by each filter.

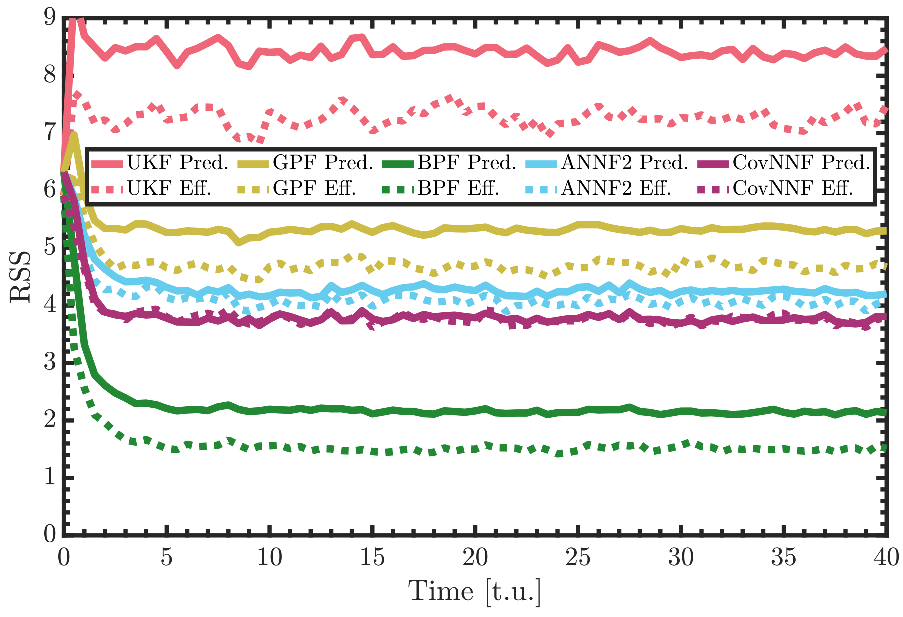

Fig. 13 Effective RSS and predicted RSS for each filter tested as a function of time with $\gamma = 2$.

As observed earlier, it becomes apparent that RSS predictions of the ANNF2 show a conservative tendency when compared to its effective RSS counterpart. Conversely, the RSS predicted by the CovNNF more closely tracks its effective RSS, indicating that the incorporation of correlation coefficients into the input vector contributes to the ability of the filter to maintain consistency in the presence of nonlinearities within the measurement model. Furthermore, it should be noted that the CovNNF consistently outperforms the UKF (which diverges due to the nonlinearity within the measurement model), the GPF and the ANNF2, as it remains closest to the BPF. Table 4 illustrates the time-averaged RSS values, represented as mean ± standard deviation. The results in this table provide further validation that the CovNNF filter is the closest match to the theoretical best. Additionally, it demonstrates the CovNNF remains the most consistent by achieving RSS predictions and effective RSS values that closely align, surpassing the consistency of the ANNF2, the GPF and the UKF in this regard.

Table 4 Time averaged root sum squared for each of the filters tested with $\gamma = 2$.

Filter	Effective RSS	Predicted RSS
UKF	7.2559 ± 0.2275	8.4026 ± 0.2832
GPF	4.7294 ± 0.2597	5.3586 ± 0.2313
ANNF2	**4.1371 ± 0.2841**	**4.3300 ± 0.3222**
CovNNF	**3.8292 ± 0.3200**	**3.8471 ± 0.3801**
BPF	1.6228 ± 0.5410	2.2835 ± 0.5634

Figure 14 shows the evolution of the RMSE with respect to time, defined in (39), for each of the filters used. Similar to figure 13, it is evident that the UKF exhibits divergence when confronted with the nonlinear measurement model. Additionally, the CovNNF maintains a slight edge in accuracy over the ANNF2 and the GPF. This indicates that, despite the initial training with a linear measurement model, the network demonstrates reasonable generalization to the more complex nonlinear measurement model. Table 5 shows the time averaged RMSE for each of the filters presented as mean ± standard deviation.

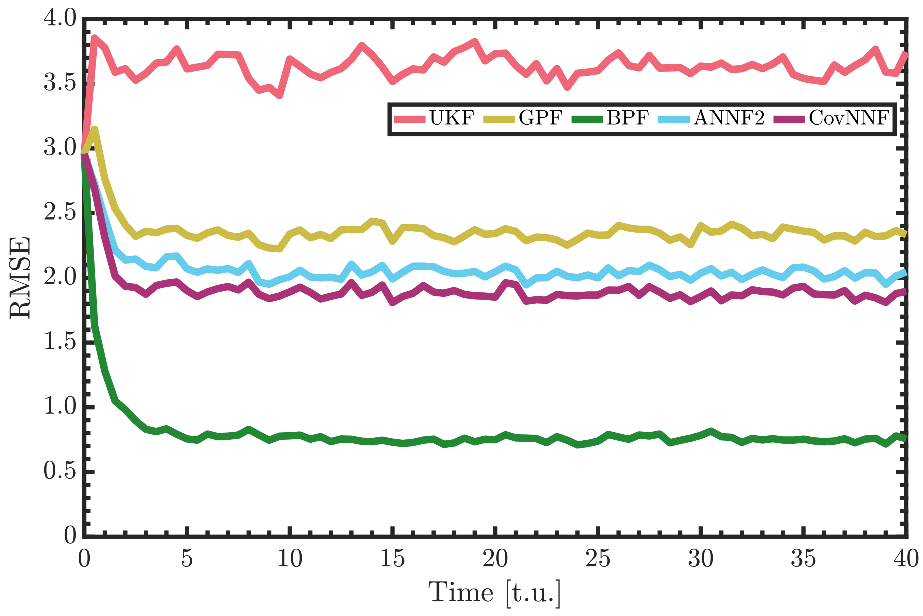

Fig. 14 RMSE for each filter tested as a function of time with $\gamma = 2$.

Figure 15 shows the time averaged RSS as a function of the parameter γ in (36) using the CovNNF as the online filter. As it can be seen, the filter performs well up to $\gamma = 3$ where the predicted RSS matches the predicted RSS of the GPF for $\gamma = 2$. For values of $\gamma > 3$, the performance of the filter degrades rapidly. Therefore, this figure shows a trust region where the neural network, which was trained using a linear measurement model, can be generalized to nonlinear measurement models, yielding consistent results.

Table 5 Time averaged root mean squared error for each of the filters tested with $\gamma = 2$.

Filter	RMSE
UKF	3.6280 ± 0.1137
GPF	2.3647 ± 0.1298
ANNF2	**2.0685 ± 0.1421**
CovNNF	**1.9146 ± 0.1600**
BPF	0.8114 ± 0.2705

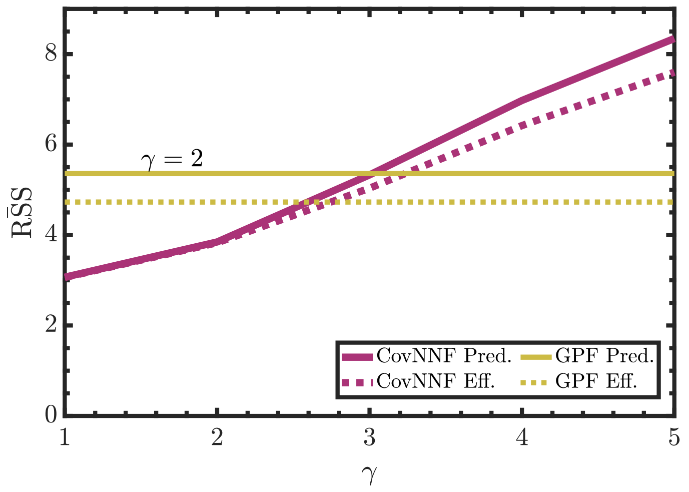

Fig. 15 Time average of the RSS as a function of the parameter γ. The yellow lines indicate the predicted and effective RSS obtained with the GPF with $\gamma = 2$, as shown in table 4.

VI. Conclusions and Future Work

This work introduces an improved learning framework and training technique tailored to neural-network-based Gaussian nonlinear filters. Notably, the incorporation of correlation coefficients into the input vector of the neural network leads to the development of a more accurate and consistent filter. The proposed filter underwent training utilizing synthetic data generated from the Lorenz '96 model and a linear measurement model. Its performance was assessed through a Monte Carlo performance test, wherein two distinct uncertainty quantification techniques, the unscented transform and Monte Carlo sampling, were employed to approximate the uncertainty in estimated states. The results obtained from both methods were observed to be in agreement. Furthermore, the unscented transform showed superior efficiency for online applications, reducing run-time by a factor of three when compared to optimal Monte Carlo sampling.

In a comparative analysis against other filters, the proposed filter consistently outperformed the unscented Kalman filter, the Gaussian particle filter and an improved implementation of previous work (ANNF2), offering results that approached the theoretical best (as calculated from the bootstrap particle filter). To determine the robustness of the trained neural network, the filter was evaluated using a Monte Carlo performance test with a nonlinear measurement model. The filter exhibited good performance even in scenarios characterized by nonlinear measurement models. This shows the ability of the filter to generalize effectively to more challenging problem domains. This new iteration of the neural-network-based filter introduces a different perspective on achieving consistent estimation in nonlinear problems. In the future, it would be intriguing to apply this methodology to real-world applications, thus providing further insights into the performance of the filter in practical scenarios.

Acknowledgment

This material is based on research sponsored by Air Force Office of Scientific Research (AFOSR) under award number: FA9550-22-1-0419.

References

[1] Bar-Shalom, Y., Li, X. R., and Kirubarajan, T., *Estimation with Applications to Tracking and Navigation*, John Wiley & Sons, Inc, 2001.

[2] Sorenson, H. W., *Kalman Filtering: Theory and Application*, IEEE Press, 1985.

[3] Julier, S., and Uhlmann, J., "Unscented filtering and nonlinear estimation," *Proceedings of the IEEE*, Vol. 92, No. 3, 2004, pp. 401–422.

[4] Liu, J. S., and Chen, R., "Sequential Monte Carlo Methods for Dynamic Systems," *Journal of the American Statistical Association*, Vol. 93, No. 443, 1998, pp. 1032–1044.

[5] Alspach, D., and Sorenson, H., "Nonlinear Bayesian estimation using Gaussian sum approximations," *IEEE Transactions on Automatic Control*, Vol. 17, No. 4, 1972, pp. 439–448.

[6] Jaynes, E., "On the rationale of maximum-entropy methods," *Proceedings of the IEEE*, Vol. 70, No. 9, 1982, pp. 939–952.

[7] Cover, T. M., and Thomas, J. A., *Elements of Information Theory*, Wiley-Interscience, 2006.

[8] Kotecha, J., and Djuric, P., "Gaussian particle filtering," *IEEE Transactions on Signal Processing*, Vol. 51, No. 10, 2003, pp. 2592–2601.

[9] Servadio, S., Zanetti, R., and Jones, B. A., "Nonlinear Filtering with a Polynomial Series of Gaussian Random Variables," *Proceedings of the 2020 IEEE International Conference on Information Fusion*, Virtual Conference, 2020.

[10] Yun, S., and Zanetti, R., "Bayesian Estimation with Artificial Neural Network," *Proceedings of the 2021 IEEE International Conference on Information Fusion*, Sun City, South Africa, 2021.

[11] Lorenz, E. N., and Emanuel, K. A., "Optimal Sites for Supplementary Weather Observations: Simulation with a Small Model," *Journal of the Atmospheric Sciences*, Vol. 55, No. 3, 1998, pp. 399–414.

[12] Van Kekem, D., "Dynamics of the Lorenz-96 model: Bifurcations, symmetries and waves," Ph.D. thesis, 2018.

[13] Goodfellow, I., Bengio, Y., and Courville, A., *Deep Learning*, MIT Press, 2016. `http://www.deeplearningbook.org`.

[14] Wübbeler, G., Krystek, M., and Elster, C., "Evaluation of measurement uncertainty and its numerical calculation by a Monte Carlo method," *Measurement Science and Technology*, Vol. 19, No. 8, 2008, p. 084009.

[15] Wan, E. A., and van der Merwe, R., *Kalman Filtering and Neural Networks*, John Wiley & Sons, Ltd, 2001.

[16] Nielsen, K., Svahn, C., Rodriguez-Deniz, H., and Hendeby, G., "UKF Parameter Tuning for Local Variation Smoothing," *Proceedings of the 2021 IEEE International Conference on Multisensor Fusion and Integration for Intelligent Systems*, 2021.

[17] Dempster, A., "Computationally Efficient Non-linear Kalman Filters for On-board Space Vehicle Navigation," Ph.D. thesis, 2017.

[18] Lewandowski, D., Kurowicka, D., and Joe, H., "Generating random correlation matrices based on vines and extended onion method," *Journal of Multivariate Analysis*, Vol. 100, No. 9, 2009, pp. 1989–2001.

[19] Prince, P., and Dormand, J., "High order embedded Runge-Kutta formulae," *Journal of Computational and Applied Mathematics*, Vol. 7, No. 1, 1981, pp. 67–75.

[20] Kingma, D. P., and Ba, J., "Adam: A Method for Stochastic Optimization," *Proceedings of the 3rd International Conference for Learning Representations*, San Diego, CA, 2017.

[21] Glorot, X., and Bengio, Y., "Understanding the difficulty of training deep feedforward neural networks," *Proceedings of the 2010 International Conference on Artificial Intelligence and Statistics*, Sardinia, Italy, 2010, pp. 249–256.

[22] Asch, M., Bocquet, M., and Nodet, M., *Data Assimilation: Methods, Algorithms and Applications*, Society for Industrial and Applied Mathematics, Philadelphia, 2016.

[23] D'souza, C., and Carpenter, J. R., "Navigation Filter Best Practices," 2018.