Available online at www.sciencedirect.com

ScienceDirect

Procedia CIRP 99 (2021) 656–661

www.elsevier.com/locate/procedia

14th CIRP Conference on Intelligent Computation in Manufacturing Engineering, CIRP ICME '20

Estimation of unknown system states based on an adaptive neural network and Kalman filter

Christoph Kellermann[a,]*, Jörn Ostermann[b]

[a]Gerresheimer Bünde GmbH, Erich Martens Str. 26-32, 32257 Bünde, Germany
[b]Institut für Informationsverarbeitung - Leibniz Universit at Hannover, Appelstr. 9A, 30167 Hannover, Germany

* Corresponding author. Tel.: +49 5223 164 233; fax: +49 5223 164 211. *E-mail address:* christoph.kellermann@gerresheimer.com

Abstract

In the field of industry 4.0 the number of sensors increases steadily. The sensor data is often used for system observation and estimation of the system parameters. Typically, Kalman filtering is used for determination of the internal system parameters. Their accuracy and robustness depends on the system knowledge, which is described by differential equations. We propose a self-configurable filter (FNN-EKF) which estimates the internal system behavior without knowledge of the differential equations and the noise power. Our filter is based on Kalman filtering with a constantly adapting neural network for state estimation. Applications are denoising sensor data or time series. Several bouncing ball simulations are realized to compare the estimation performance of the Extended Kalman Filter to the presented FNN-EKF.

Keywords: Adaptive extended Kalman filter; Neural Network; Robustness and stability metric; Denoising

1. Introduction

Extended Kalman Filters are widely used for sensor fusion and denoising sensor data in areas of signal processing and prediction. This technique is often used before a noisy time line series can be used for information extraction. Although great progress has been achieved with classical Kalman filters and neuronal network techniques for time series approximation and regression were developed. Both techniques need suitable background information like physical assumptions or huge training datasets.

In practice these assumptions or training datasets are often not available. The target is to denoise field sensors in industrial machinery without information of the theoretical optimal time series. A real case is e.g. further processing for features or mechanical fading.

In this paper we introduce a system that takes directly raw input data from sensors and calculates signal characteristics for denoising the inputs without knowing the system behavior. We generally assume that this is in practice a nonlinear system. The input time series is separated in origin and noise component. Internal parameters like the probability density function of noise and power are calculated. The estimated robustness and accuracy of the neuronal network which is used for filtering, and the related observation are published to the user.

This paper is divided into 6 sections. In section 2 we recall the Extended Kalman Filter (EKF) first followed by a short introduction into back propagation of feed forward neuronal networks (FNN). Section 3 contains the derivation of our approach based on EKF and FNN techniques. The role of system observability is also discussed in this context. In section 4 the characteristics of the implementation with respect to computational power and numerical stability is shown. In section 5, we compare our approach to an EKF simulation scenario. The conclusions are drawn in section 6.

2. Related Work

In this subsection we derive a filtered discreet signal (y_k) which contains superimposed noise at time k.

$$y_k = x_k + w_k + v_k \qquad (1)$$

The signal is divided into the true value (x_k) and two noise components, the process noise (w_k) and measurement noise (v_k).

2.1. Discrete Extended Kalman Filter

There are many examples for successful Kalman filtering in control system applications with noisy input data since the original paper by Kalman [1]. Later on, the Kalman filter problem was transferred to a nonlinear model called “Extended Kalman Filter” (EKF). Most EKFs employ a first order linearization using the Taylor series expansion. Errors in the estimated parameters may lead to divergences sometimes [2]. In addition a second-order version of EKF exists but the calculation complexity makes it unfeasible in practical usage, especially for real-time and high dimensional systems [3].
This subsection focuses on the derivation of the discrete-time EKF.

Nomenclature

$\mathbb{E}$	expectation operator
$\mathbf{J}$	symbolizes Jacobian Matrix
x_k	unknown states
y_k	observations/measurements, input matrix
$\boldsymbol{W_k}$	process noise
$\boldsymbol{V_k}$	measurement noise
$\boldsymbol{J_f}(\cdot)$	process nonlinear function matrix
$\boldsymbol{J_h}\,(\cdot)$	observation nonlinear function matrix
$\boldsymbol{Q_k}$	process noise covariance matrix
$\boldsymbol{R_k}$	measurement noise covariance matrix
$\boldsymbol{P_k}$	approximate error covariance matrix
$\boldsymbol{K_k}$	Kalman gain matrix

Variables with a hat stand for estimated values.
Vectors and matrices are printed in bold letters.

A set of EKF equations is available to perform the estimation of state. The equations are given as follows [4], [5].
Model and observation states:

$$\boldsymbol{x_k} = \boldsymbol{J_f}\,(\boldsymbol{x_{k-1}}) + \boldsymbol{W_{k-1}} \quad (2)$$

$$\boldsymbol{y_k} = \boldsymbol{J_h}\,(\boldsymbol{x_k}) + \boldsymbol{V_k} \quad (3)$$

Initialization:

$$\boldsymbol{\hat{x}_0} = \mathbb{E}[\boldsymbol{x_0}] \quad (4)$$

$$\boldsymbol{\hat{P}_0} = \mathbb{E}[(\boldsymbol{x_0} - \boldsymbol{\hat{x}_0})(\boldsymbol{x_0} - \boldsymbol{\hat{x}_0})^T] \quad (5)$$

Time update/Forecast:

$$\boldsymbol{\hat{x}_k} = \boldsymbol{F}(\boldsymbol{\hat{x}_{k-1}}) \quad (6)$$

$$\boldsymbol{\hat{P}_k} = \boldsymbol{J_f}(\boldsymbol{\hat{x}_{k-1}})\boldsymbol{P_{k-1}}\boldsymbol{J_f^T}(\boldsymbol{\hat{x}_{k-1}}) + \boldsymbol{Q_{k-1}} \quad (7)$$

Measurement update/corrector:

$$\boldsymbol{\hat{x}_k} = \boldsymbol{\hat{x}_{k-1}} + \boldsymbol{K_k}\left(\boldsymbol{y_k} - \boldsymbol{J_h}\,(\boldsymbol{\hat{x}_k})\right) \quad (8)$$

$$\boldsymbol{K_k} = \boldsymbol{\hat{P}_k}\boldsymbol{J_h}\,(\boldsymbol{\hat{x}_k})\boldsymbol{E_k} \quad (9)$$

$$\text{with } \boldsymbol{E_k} = \left[\boldsymbol{J_h^T}(\hat{x}_k)\boldsymbol{\hat{P}_k}\boldsymbol{J_h}\,(\boldsymbol{\hat{x}_k}) + \boldsymbol{R_k}\right]^{-1} \quad (10)$$

$$\boldsymbol{P_{k+1}} = \boldsymbol{\hat{P}_k} - \boldsymbol{K_k}\boldsymbol{J_h}\,(\boldsymbol{\hat{x}_k})\boldsymbol{\hat{P}_k} \quad (11)$$

The coefficients of $\boldsymbol{P_{k+1}}$ are estimated at time k based on the new observations and the state estimations are propagated to time $k+1$. The maximum likelihood function is estimated by the coefficients of the Kalman gain matrix $\boldsymbol{K_k}$. The Kalman gain matrix $\boldsymbol{K_k}$ is based on the assumption reflected in the noise covariance matrices $\boldsymbol{R_k}$ and $\boldsymbol{Q_k}$. These three matrices are important for the stability of the EKF [5], [6]. We will concretize this in chapter 3.4.
Equation (8) denotes the state matrix prediction. The part $\boldsymbol{y_k} - \boldsymbol{J_h}\,(\boldsymbol{\hat{x}_k})$ of equation (8) is called state error vector. Special attention will be paid to the error vector in chapter 3.

Constraints to the EKF model are denoted in the following. Process noise $\boldsymbol{W_k}$ and measurement noise $\boldsymbol{V_k}$ must be temporally uncorrelated zero-mean Gaussian random sequences. Matrices $\boldsymbol{Q_k}$ and $\boldsymbol{R_k}$contain the known covariances of the process and the measurement noise, respectively.

$$\mathbb{E}\left[W_i V_j^T\right] = 0\; for\; all\; i\; and\; j \quad (12)$$

$$\mathbb{E}[W_k W_k^T] = \begin{cases} \boldsymbol{Q_k}\; for\; i = j \\ 0\;\; for\; i \neq j \end{cases} \quad (13)$$

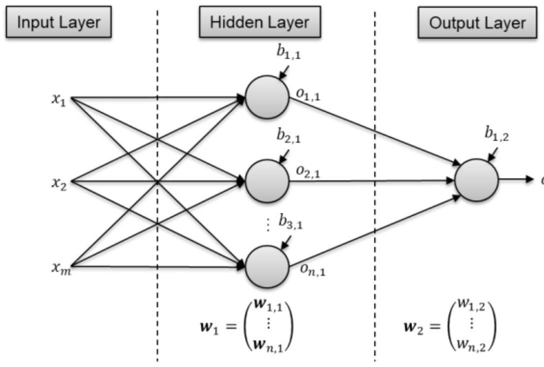
$$\mathbb{E}[V_k V_k^T] = \begin{cases} \boldsymbol{R_k}\; for\; i = j \\ 0\;\; for\; i \neq j \end{cases} \quad (14)$$

The estimation performance of an EKF is directly related to the correct $\boldsymbol{Q}$and $\boldsymbol{R}$ noise covariance matrices [7].

2.2. Backpropagation for Forward Neural Network

Feed forward neural networks (FNN) with an input, one hidden and an output layer can theoretically approximate every function [8]. Fig. 1 shows a FNN with only one hidden layer.
A given input pattern $\boldsymbol{x} = \{x_1, \cdots, x_m\}$ leads after propagation through the FNN to an output $\boldsymbol{o} = \{o_{1,2}, \cdots, o_{n,2}\}$. The nomenclature for FNN is first index of the node followed by the index of the layer.
Inside the network consists of different neurons which have weights and an additive bias. The result of weighted sum with the bias is passed to an activation function F_j before the output o_j can be computed [9] (Fig. 2). The output of a neuron with index j is calculated by

$$o_j = \mathcal{F}_j\left(\left(\textstyle\sum_{i=1}^{n} x_i w_{i,j}\right) + b_j\right) \quad (15)$$



Input Layer
Hidden Layer
Output Layer
x_1
x_2
x_m
$b_{1,1}$
$o_{1,1}$
$b_{2,1}$
$o_{2,1}$
$b_{3,1}$
$o_{n,1}$
$b_{1,2}$
$o_{1,2}$
$\boldsymbol{w}_1 = \begin{pmatrix} \boldsymbol{w}_{1,1} \\ \vdots \\ \boldsymbol{w}_{n,1} \end{pmatrix}$
$\boldsymbol{w}_2 = \begin{pmatrix} w_{1,2} \\ \vdots \\ w_{n,2} \end{pmatrix}$

Fig. 1. Feed forward network with m inputs and one output. Connection matrices are labelled with w. In the hidden layer the $\boldsymbol{w}_{i,j}$is a row vector with length m.

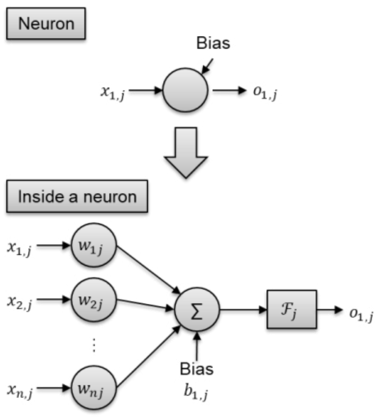


Fig. 2. Structure of a neuron is shown. The first index stands for the node followed by the index of the layer.

The goal is to estimate the correct weights and bias to minimize the mean square error of the overall outputs and their corresponding targets defined by the training set [9]. The training dataset $\{(\boldsymbol{x}_1, \boldsymbol{t}_1), \cdots, (\boldsymbol{x}_p, \boldsymbol{t}_p)\}$ consists of p pairs (input and target patterns) of m- and n-dimensional vectors.

The most popular learning algorithms for FFNs is the gradient descent error backpropagation (GDB) described hereinafter [10]. Every weight and bias is updated per iteration with the set of equations (16) to (23) [10] [11]. The adjustments of weights and bias are based on gradient descent. η represents the learning rate which defines the step size of each iteration. In this example the output layer has no activation function (Eq. 23).

For the hidden layer:

Weight update: $\Delta w_{i,1} = -\eta \delta_{i,1} o_{i,0}$ (16)

Output init.: $o_{i,0} = x_i$ (17)

Bias update: $\Delta b_{i,1} = -\eta \delta_{i,1}$ (18)

$$\delta_{i,1} = \mathcal{F}_1'(o_{i,1}) w_{i,2} \delta_{i,2} \quad (19)$$

For the output layer:

Weight update: $\Delta w_{i,2} = -\eta \delta_{i,2} o_{i,1}$ (20)

Bias update: $\Delta b_{i,2} = -\eta \delta_{i,2}$ (21)

$$\delta_{i,2} = \mathcal{F}_2'(o_{i,2}) * (o_{i,2} - t_i) \quad (22)$$

$$\mathcal{F}_2'(o_{i,2}) = 1 \quad (23)$$

In case of multiple input/output patterns, each pattern is used to compute the error function separately. Each pattern calculates a new weight correction. This is called “batch updates”.

This pure backpropagation has a problem with convergence over time in some cases. In this senario the error during training is constant over hunderts of interations before the error drops. Sapkal et. al. [10] show that by adding gaussian noise to weighted sum entity Eq. (15) the convergence rate can be improved.

Furthermore, this kind of backpropagation gives unsatisfactory final results when the training set is affected by heavy noise (low SNR).

3. Methodology of denoising time series with EKF-FNN backpropagation

Therefore a learning algorithm is presented that is based on EKF and the backpropagation for FNNs. Thus, we denoise raw signals affected by noise.

3.1. EKF for parameter estimation

We formally show how to combine the EKF equations (chapter 2.1) and the backpropagation (chapter 2.2) for parameter estimation. Technical details can be found in chapter 4.

We assume that the sampling rate of the measured items is constant and has an insignificant time jitter. For this purpose the $\boldsymbol{J_f}(\boldsymbol{\hat{x}_{k-1}})$ in the time update of the EKF becomes approximately an identity matrix.

Based on (7) and

$$\boldsymbol{J_f}(\boldsymbol{\hat{x}_{k-1}}) = \boldsymbol{I} \quad (24)$$

we find

$$\boldsymbol{\hat{P}_k} = \boldsymbol{P_{k-1}} + \boldsymbol{Q_{k-1}} \quad (25)$$

This step affects the Kalman gain matrix (Eq. 9) and the update of the error covariance matrix (Eq. 11). These effects simplify implementation.

3.2. Adaptive EKF

In Almagbile et al. [7] $\boldsymbol{Q}$ and $\boldsymbol{R}$ are estimated and their values are checked for plausibility. This leads to dynamic minimazation and measurement update errors with filter convergence problems.

For initialization the diagonal of $\boldsymbol{R_k}$ matrix is set to the estimated standard deviations of the input signals. For the estimation of the standard deviation the target is to select a sliding window of a feasible length N, such that the standard deviation of the window is representative for the complete signal. The selected window size affects the estimation performance of the calculated standard deviation. It should be correctly determined to get high performance estimation because the result is directly linked to the $\boldsymbol{Q}$ and $\boldsymbol{R}$ noise covariance matrices. In theory there is not a known way to choose the right window size. The window size depends on the dynamics of the system which is also depends on the concrete application. In [7] and [11] it is shown that a large window size enhances the ability to converge while a small window size causes the filter to diverge. In further iterations of the EKF the matrices are adjusted. In the implementation chapter 4.1 the usage and link to the covariance matching is explained.

3.3. Junction between FNN and EKF

Kalman filter solves the process and measurement equations for unknown states with an optimal approach. We want to use EKF parameter estimation for gradient descent backpropagation to find weights and bias for the FNN. Therefore, the weights and bias values are denoted as states of the EKF algorithms. The backpropagation of the FNN is similar to the measurement update of the EKF. $\boldsymbol{J_h}$ is updated by the forward propagation of the FNN. For the shown FNN in

Fig. 1 above the state vector $\widehat{\boldsymbol{x}}_{\boldsymbol{k}}$ of the EKF is defined as $\widehat{\boldsymbol{x}}_{\boldsymbol{k}}(w_k, b_k) =$
$(w_{1,1} \; \cdots \; w_{n,1} \; b_{1,1} \; \cdots \; b_{n,1} \; w_{1,2} \; \cdots \; w_{n,2} \; b_{1,2})^T$
This choice of the state vector leads to several similarities of the EKF and GDB.

The state error vector of the EKF $\boldsymbol{y_k} - \boldsymbol{J_h}\,(\widehat{\boldsymbol{x}}_{\boldsymbol{k}})$ corresponds to the derivative of the GDB output layer $\delta_{i,2} = (o_{i,2} - t_i)$ (Eq. 22). The measurement update from EKF is quite equal to the GDB $\widehat{\boldsymbol{x}}_{\boldsymbol{k+1}} - \widehat{\boldsymbol{x}}_{\boldsymbol{k}} = \Delta\boldsymbol{w} \;\; = \boldsymbol{K} \;\; \cdot \left(\boldsymbol{y_k} - \boldsymbol{J_h}\,(\widehat{\boldsymbol{x}}_{\boldsymbol{k}})\right)$ with

$$\boldsymbol{J_h} = \left(\Delta w_{1,1} \ldots \Delta w_{n,1} \, \Delta b_{1,1} \ldots \Delta b_{n,1} \, \Delta w_{1,2} \ldots \Delta w_{n,2} \, \Delta b_{1,2}\right)^T$$

The Kalman gain matrix (Eq. 9) is used to calculate the step size for the weight and bias correction. In the following the algorithms is called FNN-EKF.

3.4. Observability – Robustness and sensibility metric

This section is focused on observability of the training process. For finding suitable internal parameters and for the overall rating of the performance metrics of FNN-EKF models are essential. For this purpose, the following metrics for robustness and measurement of sensitivity are introduced.

In general the robustness of the EKF is significantly influenced by the covariance matrices **Q** and **R** [6] [7]. Thus, the FNN-EKF algorithm converges without inconsistencies. All covariant matrices must be positive definite. Therefore, $\boldsymbol{Q_k}$, $\boldsymbol{R_k}$ have to be symmetric positive definite matrices.

While the FNN-EKF iterates the covariance matrix $\boldsymbol{P_k}$ has to be checked to be positive definite. By the numerical inaccuracy over the iterations due to the use of finite-word length arithmetic $\boldsymbol{P_k}$ can become indefinite[12]. If $\boldsymbol{P_k}$ becomes indefinite the Kalman filter becomes unstable.

In terms of robustness and sensitivity of system performance monitoring the RMSE is often used as general performance metric. We need a metric to obtain a desired trade-off between robustness and sensitivity in our algorithm. An observability metrics that represents the robustness and sensitivity of the Kalman filter is introduced in [6]. Saha et al. [6] shift the focus from the estimated states to the innovation in getting suitable performance metrics. Those metrics are related to the innovation covariance. The state error vector $\boldsymbol{y_k} - \boldsymbol{J_h}\,(\widehat{\boldsymbol{x}}_{\boldsymbol{k}})$ of Eq. (8) can also be called "innovation" because it adjusts the new states in the current iteration. This innovation and its covariance $\boldsymbol{E_k^{-1}}$ from Eq. (10) provide a deterministic basis for robustness and sensibility of the metrics. For that Saha et al. define two terms $\boldsymbol{A_k}$ and $\boldsymbol{B_k}$ respectively derived from the innovation covariance in [6].

$$\boldsymbol{E_k^{-1}} = \boldsymbol{A_k} + \boldsymbol{B_k} + \boldsymbol{R_k} \qquad (26)$$

Metrics J_{1k} and J_{2k} are defined:

$$J_{1k} = tr([\boldsymbol{A_k} + \boldsymbol{B_k} + \boldsymbol{R_k}]^{-1} \cdot \boldsymbol{R_k}) \qquad (27)$$

$$J_{2k} = tr([\boldsymbol{A_k} + \boldsymbol{B_k}]^{-1} \cdot \boldsymbol{B_k}) \qquad (28)$$

J_{1k} shows the effect of the measurement noise because it is attributed to $\boldsymbol{R_k}$. It assumes the sensitivity of the EKF. J_{1k} indicates any mismatch between assumed measurement and real noise.

J_{2k} is distinctive for robustness and is linked to the process noise covariance $\boldsymbol{Q_k}$. Hence J_{2k} indicates a mismatch between the real noise and the modeling noise.

The combination of J_{1k} and J_{2k} leads to the number of measurements.

$$m = J_{1k} + J_{2k} + tr(\boldsymbol{N_k}) \qquad (29)$$

With $\boldsymbol{N_k} = [\boldsymbol{A_k} + \boldsymbol{B_k}]^{-1} \cdot \left[\boldsymbol{J_h}\,(\widehat{\boldsymbol{x}}_{\boldsymbol{k}}) \cdot \left(\boldsymbol{J_f}(\widehat{\boldsymbol{x}}_{\boldsymbol{k-1}}) \cdot \boldsymbol{P_{k-1}} \cdot \boldsymbol{J_f^T}(\widehat{\boldsymbol{x}}_{\boldsymbol{k-1}}) - \boldsymbol{P_k}\right) \cdot \boldsymbol{J_h^T}(\widehat{\boldsymbol{x}}_{\boldsymbol{k}})\right]$ (30)

$$\boldsymbol{A_k} + \boldsymbol{B_k} = \boldsymbol{J_h}\,(\widehat{\boldsymbol{x}}_{\boldsymbol{k}})\widehat{\boldsymbol{P}}_{\boldsymbol{k}}\boldsymbol{J_h^T}(\widehat{\boldsymbol{x}}_{\boldsymbol{k}}) \qquad (31)$$

$$\boldsymbol{B_k} = \boldsymbol{J_h}\,(\widehat{\boldsymbol{x}}_{\boldsymbol{k}})\boldsymbol{Q_{k-1}}\boldsymbol{J_h^T}(\widehat{\boldsymbol{x}}_{\boldsymbol{k}}) \qquad (32)$$

With Eq. (24):

$$\boldsymbol{N_k} = [\boldsymbol{A_k} + \boldsymbol{B_k}]^{-1} \cdot \left[\boldsymbol{J_h}\,(\widehat{\boldsymbol{x}}_{\boldsymbol{k}}) \cdot (\boldsymbol{P_{k-1}} - \boldsymbol{P_k}) \cdot \boldsymbol{J_h^T}(\widehat{\boldsymbol{x}}_{\boldsymbol{k}})\right] \qquad (33)$$

4. Implementation

In this chapter the implementation of the FNN-EKF algorithms is introduced. In addition to the use of the algorithm, the internal monitoring of stability (as described in chap. 3.4) and the computational power of the execution are considered.

The block-diagram representation of the FNN-EKF algorithm with its observation metrics and auto-parameter adjustments is given in Fig. 3. This setup can be used especially for time series filtering. Input parameter $\boldsymbol{y(k)}$ is a vector or matrix of noisy data with m features and n samples per feature. $\boldsymbol{k}$ is an input vector which contains the corresponding time stamp of the measurements.

4.1. Initialization and data preparation

First step is initialization of the FNN and the parameter settings for the FNN-EKF algorithms. The weights and the bias of the FNN are initialized with gaussian noise.

The first step of the parameter settings is scaling the feature input $\boldsymbol{y(k)}$, so that every feature has a same mean and variance. Hence the relevance of all feature is equal. In order to calculate the initial $\boldsymbol{Q}$, $\boldsymbol{R}$ and $\boldsymbol{P}$ for backpropagation computation the standard deviation of each feature is calculated within a sliding window over the signal. The selection of the right window size already is described in Chap. 3.2. We assume that the noise is according to the specifications Eq. (12)-(14) at the end of Chap. 2.1. While the distribution is measured within a sliding window, the variation of the distribution is providing the step size for the backpropagation. The range of values of the step size is $[0, 1]$. A small variation of input parameters tends to a low volatile value of deviation in every window. This leads to step size 1. This means a high variation leads to a small step size.

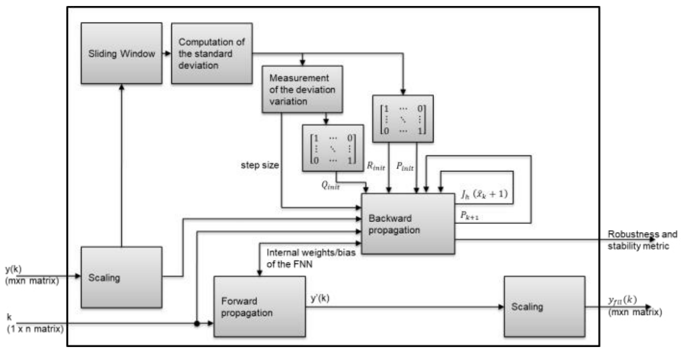

Sliding Window
Computation of the standard deviation
Measurement of the deviation variation
$\begin{bmatrix}1 & \cdots & 0\\ \vdots & \ddots & \vdots\\ 0 & \cdots & 1\end{bmatrix}$
$\begin{bmatrix}1 & \cdots & 0\\ \vdots & \ddots & \vdots\\ 0 & \cdots & 1\end{bmatrix}$
step size
R_{init}
P_{init}
Q_{init}
$J_h\,(\hat{x}_k + 1)$
P_{k+1}
Backward propagation
Robustness and stability metric
Internal weights/bias of the FNN
y(k) (mxn matrix)
Scaling
k (1 x n matrix)
Forward propagation
y'(k)
Scaling
$y_{fil}(k)$ (mxn matrix)

Fig. 3. Block-diagram of the FNN-EKF with auto-parameter adjustment and observation metrics

In order to select a good starting point the target is to select the sliding window of length N in a right way to receive a representative standard deviation. The length N of the sliding window is initialized with $N = n/10$ and in later iteration it is adjusted by the metric J_{1k} from Chap. 3.4.

4.2. Training

After initialization and pre-processing the forward propagation is executed to obtain the outputs of every layer $\boldsymbol{y}'(\boldsymbol{k})$. The outputs are needed for the backpropagation with the FNN-EKF algorithm (Chap. 2.2 and 3.3). The time stamp vector $\boldsymbol{k}$ is fed to the input side of the FNN and the noisy data $\boldsymbol{y}(\boldsymbol{k})$ is connected to the output of the FNN. By every iteration of the backpropagation the error covariance matrix $\boldsymbol{P_k}$ is updated and the FNN approximates to the raw signal affected with noise.

In terms of computational performance the calculation time for the $\boldsymbol{P_k}$ matrix is cost-intensive (Eq. 11) because the matrix $\boldsymbol{J_h}$ gets rapidly bigger with every input feature and increasing network size (Chap. 3.3). With respect to the computation time the training phase is split up into two modes – fast training and normal training. The first fast training focuses on fast approximation with less accuracy. This is due to the fact that at the beginning, when the FNN is initialized, the $\boldsymbol{P_k}$ matrix only needs to be a rough approximation of the final target matrix for fast convergence. Thus, the P_k matrix is updated in an interval i_P per iteration. The calculation index per epoch is shifted by the interval of i_{Epoch}. This means, for example, if the training vector of length $n = 10$, $i_P = 1$ and $i_{Epoch} = 1$ are selected, the $\boldsymbol{P_k}$ matrix has been updated after 10 epochs for each value.

In normal training mode the goal is to get a high accuracy. Here the algorithms are used as it has already been described in Chap. 2.1. $\boldsymbol{P_k}$ matrix is updated every iteration. While the training phase is running the numerical stability of $\boldsymbol{P_k}$ in every iteration is traced with $rang(\boldsymbol{P_k}) = m$ with $m = 3$ for 3 inputs to ensure the stability of the EKF shown in paragraph 3.4. Additionally, the RMSE is traced per measurement in every iteration and compared to an expected RMSE per measurement dimension because the target RMSE depends on the application. If a fault is detected the process is stopped and the user is notified.

4.3. Using the filter

After the training sequence the FNN model approximates the raw signal affected with noise. The filtering of the noisy input data is done by forward propagation. Finally $\boldsymbol{y}'(\boldsymbol{k})$ is scaled to the means and variance of the input feature.

5. Simulation Results and Observations

In this section, a ball throw simulation should show the effectiveness of the proposed FNN-EKF algorithms. The benchmark is a standard EKF implementation for estimating the trajectory of the ball. This scenario is well known from various aviation publications [15]. By fusing the measurement data form the acceleration sensor and camera based position system the EKF calculation gets the velocity $(\dot{x}, \dot{y}, \dot{z})$ as well as the position (x, y, z). For better visualization $y = 0$ is chosen.

In all simulations the setup from the FNN-EKF is the same. The feed forward network has 50 neurons in the hidden layer with “thanh”-activation function. The network is updated in 100 epochs.

5.1. Estimation performance

In the first simulation trial standard deviation is set to 0.2 ($SNR = 22.25\ dB$) and is kept constant during this simulation experiment.

Fig. 4 shows the trajectories of the ball hitting the ground at $z = 0$. The orange solid line shows the true path of the ball. The gray dots are the measured position.

Fig. 6 shows the comparison between the real EKF and the FNN-EKF. The RMSEs can be found in Tab. 1. Especially for low SNR, FNN-EKF shows an advantage.

In another simulation trial, the standard deviation of the noise was changed and it produces the results of Table 1.

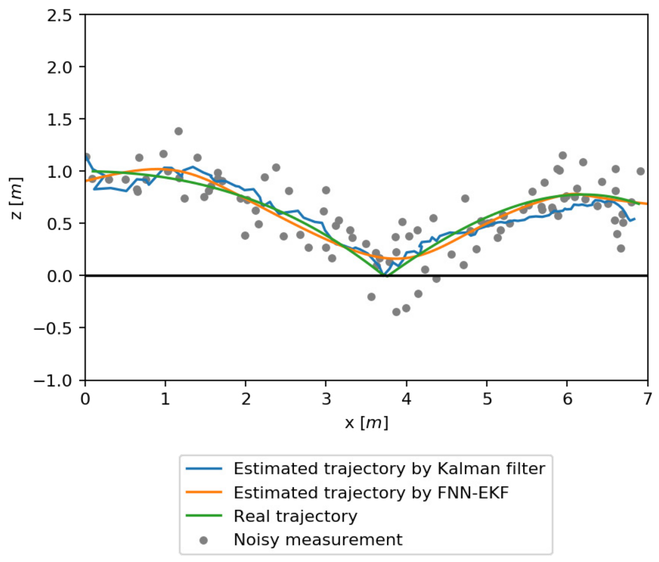


Fig. 4. Trajectories of the first simulation of a hitting and a SNR of 22.25 dB

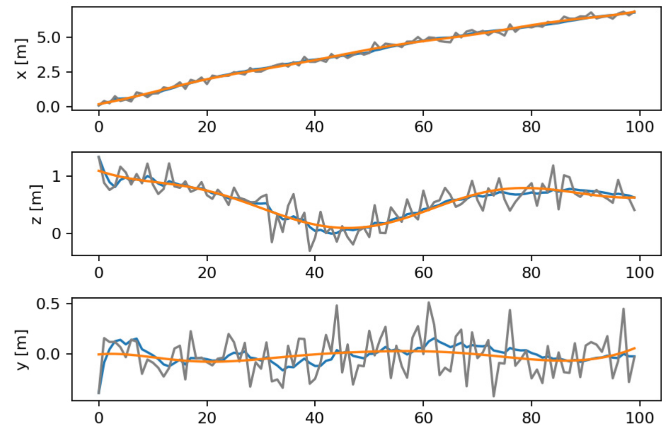


Fig. 5. Component wise plot of the FNN-EKF estimation per sample. Grey line is the noisy input data, blue line is the estimation of Kalman filter and orange line is the estimation of FNN-EKF

Table 1. Different standard deviation for the input dataset

Algorithm	Stand. devi. [m]	SNR [dB]	RMSE
EKF FNN-EKF	0.1	28.49	2.356 4.193
EKF FNN-EKF	0.2	22.25	6.825 4.036
EKF FNN-EKF	0.3	18.46	5.203 5.744
EKF FNN-EKF	0.4	15.59	16.77 11.05
EKF FNN-EKF	0.5	14.30	13.31 10.89

In most cases the self-configurable FNN-EKF filter is quite near or sometime even better than the real EKF which requires the knowledge of the internal system behavior given by the differential equations and the noise power.

5.2. Computational performance

The computational performance is compared over 100 epochs of normal training and the fast training explained in chapter 4.2. The fast training is approximately two time faster due to its reduced calculation complexity. A complete epoch update takes 2.1s on one core of an Intel Xeon CPU with 2.30GHz. For further speedup special indexes could be updated after the network is pretrained.

6. Conclusion

In this paper we derive a feed forward network with an extended Kalman filter optimizer and described an algorithm for a self-adaptable denoising filter. The FNN-EKF was compared to a standard extended Kalman filter.

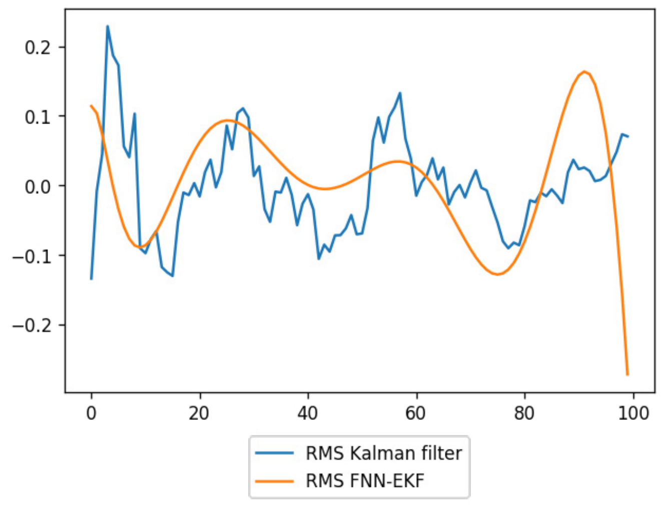


Fig. 6. Deviation from real trajectory comparison per sample

The weights and bias values of the FNN are used as states of the Kalman filter. This enables the network to compute the observations of the filter.

We show that the approximation accuracy of the FNN-EKF is similar to the standard EKF. However, FNN-EKF needs no knowledge of the system behavior. Throughout various simulations, it is shown the efficiency of the FNN-EKF to extract the complex dynamics of the system for denoising nonlinear time series from noisy multi-dimensional input data. First experiments for using the FNN-EKF to denoise real sensor data from a torque motor are already started.

References

[1] Kalman RE. A new approach to linear filtering and prediction problems. Transaction of the asme journal of basic; 1960.

[2] Julier SJ, Uhlmann JK, Durrant-Whyte HF. A new approach for filtering nonlinear systems. In Proceedings of 1995 American Control Conference-ACC'95; 1995.

[3] Dhall S, Lakshmivarahan S, Lewis JM. Dynamic Data Assimilation: A least squares approach. Cambridge University Press; 2006.

[4] Terejanu G. Extended Kalman Filter Tutorial; 2009.

[5] Reif K, Gunther S, Yaz E, Unbehauen R. Stochastic stability of the discrete-time extended Kalman filter. IEEE Transactions on Automatic control, Vol. 44; 1999. p. 714-728.

[6] Saha M, Ghosh R, Goswami B. Robustness and Sensitivity Metrics for Tuning the Extended Kalman Filter. IEEE Transactions on Instrumentation and Measurement, Vol. 63; 2014. p. 964-971.

[7] Almagbile A, Wang J, Ding W. Evaluating the performances of adaptive Kalman filter methods in GPS/INS integration. Journal of Global Positioning Systems, Vol. 9; 2010. p. 33-40.

[8] Funahashi KI. On the approximate realization of continuous mappings by neural networks. Neural networks, Vol. 2; 1989. p. 183-192.

[9] Goodfellow I, Bengio Y, Courville A. Deep learning. MIT press; 2016.

[10] Rojas R. The backpropagation algorithm. In Neural networks, Springer; 1996. p. 149-182.

[11] Chen X, Ren H, Liu J. Intelligent structural rating system based on backpropagation network. Journal of Aircraft, Vol. 50; 2013. p. 947-951.

[12] Sapkal A, Kulkarni UV. Modified Backpropagation with Added White Gaussian Noise in Weighted Sum for Convergence Improvement. Procedia computer science, Vol. 143; 2018. p. 309-316.

[13] Mohamed AH, Schwarz KP. Adaptive Kalman filtering for INS/GPS. Journal of geodesy, Vol. 73; 1999. p. 193-203.

[14] Maybeck PS. Stochastic models estimation and control. Vol. 141, Elsevier; 1979. p. 368-409.

[15] Cross R. Ball Trajectories. In Physics of Baseball & Softball, New, York: Springer New York; 2011. p. 37–57.