An FMI-based Framework for State and Parameter Estimation

Marco Bonvini, Michael Wetter, Michael D. Sohn
Simulation Research Group, Lawrence Berkeley National Laboratory
1 Cyclotron Road, 94720, Berkeley, CA

Abstract

This paper proposes a solution for creating a model-based state and parameter estimator for dynamic systems described using the FMI standard. This work uses a nonlinear state estimation technique called unscented Kalman filter (UKF), together with a smoother that improves the reliability of the estimation. The algorithm can be used to support advanced control techniques (e.g., adaptive control) or for fault detection and diagnostics (FDD). This work extends the capabilities of any modeling framework compliant with the FMI standard version 1.0.

Keywords: Nonlinear State and Parameter Estimation; Unscented Kalman Filter (UKF); Smoothing; Functional Mockup Interface (FMI); Fault Detection and Diagnosis (FDD)

1 Introduction

In many applications, after the system has been designed, controls and/or fault detection and diagnostics (FDD) algorithms are developed and deployed. These techniques should be able to leverage the models developed during the earlier design stages, thereby increasing the productivity of the overall product development. Advanced control (such as adaptive control or model predictive control) and FDD techniques require an enhanced knowledge of the system state. For example, the flight controller of an airplane should try to estimate the real velocity and position of the aircraft while compensating for measurements errors and sensor noises. When dealing with dynamic system, having an enhanced knowledge about the system state means estimating its state variables with associated error bounds.

This paper proposes a solution for creating a model-based state estimator for dynamic systems described using the FMI standard. This work extends the capabilities of any modeling framework compliant with the FMI standard version 1.0. The FMI is a standard that allows to embed a simulation model within a unified interface in order to couple simulation models developed using different simulation programs. Although the FMI standard has been created mainly for co-simulation, we leverage this standard for providing an algorithm that is compatible with a large number of simulation and modeling platforms, including Modelica-based ones.

There are several characteristics intrinsic of any model-based state estimation technique. These characteristics are related to the model properties (e.g., the Kalman filter is applicable just to linear models), the assumptions introduced when describing the probability distribution of the state variables (e.g., assuming they are Gaussian) and the computational performance of the underlying algorithm (e.g., the number of simulations or computations to be done in order to provide an estimation).

The state estimation technique used in this work is the unscented Kalman filter (UKF) [1, 2]. The UKF is able to deal with nonlinear systems and it just requires to perform function evaluations of the model in order to compute the evolution of its state variables and the value of its outputs. The UKF has less requirements about the knowledge of the model with respect to other nonlinear state estimation techniques. For example the extended Kalman filter needs to linearize the model [3]. The computational performances of the UKF are modest with respect to other Monte Carlo based techniques (like particle filters [4]), enabling its use for real-time applications.

The proposed work leverages the UKF technique and provides a state and parameter estimation algorithm for a dynamic system (e.g., modeled with Modelica or Matlab) embedded according to the FMI standard as a Functional Mockup Unit (FMU). The model, once exported as FMU can be used to set up a state and parameter estimator to

- calibrate the model during the commissioning phase in order to check if it performs as expected,
- compute a probabilistic estimation of unknown

variables of the system (e.g., observable but not measured) for control or FDD.

The paper is structured as follows. Section 2 starts with a brief introduction about the state estimation and it continues with a description of the unscented Kalman filter and smoothing algorithm, including the modifications required to extend the state estimation procedure to the parameters. Subsection 2.5 gives more information about the implementation details, while subsection 2.6 contains a code snippet that shows how to use the proposed algorithm. Section 3 contains an example that shows how the proposed algorithm can be used to identify faulty operation in a valve in the presence of measurement errors.

2 Method

The FMI-based state and parameter estimation algorithm consists of two components: (i) a filter for the state and parameter estimation, (ii) a smoother that improves the quality of the estimation when the measurements are noisy and sometimes erroneous. The proposed algorithm has been written in Python and uses PyFMI [5]. Some of the basic methods and classes provided by PyFMI have been extended to fit our purposes. For example, we modified how FMUs are executed in parallel.

2.1 State Estimation

Kalman Filter (KF) [3] are often used to estimate state variables. However, as they are only applicable for linear systems, they are not suited for our applications. For systems that are described by nonlinear differential equations, the state estimation problem can be solved using an Extended Kalman Filter (EKF) [3]. The EKF linearizes around the current state estimate the original nonlinear model. However, in some cases, this linearization introduces large errors in the estimated second order statistics of the estimated state vector probability distribution [6]. Another approach is to simulate sample paths that generate random points in the neighborhood of the old posterior probability, for example by using Monte Carlo sampling, and adopting particle filters for the state estimation [4]. These techniques are robust with respect to model nonlinearities, but they are computationally expensive. The UKF faces the problem representing the state as a Gaussian random variable, the distribution of which is modeled non parametrically using a set of points known as sigma points [1]. Using the sigma points, i.e., by propagating a suitable number of state realizations through the state and output equations, the mean and the covariance of the state can be captured. The favorable properties of the UKF makes its computational cost far lower than the Monte Carlo approaches, since a limited and deterministic number of samples are required. Furthermore, the UKF requirements fit perfectly with the infrastructure provided by PyFMI since it provides an interface to the FMU model that allows to set state variables, parameter and running simulations.

2.2 The Unscented Kalman Filter

The Unscented Kalman Filter is a model based-techniques that recursively estimates the states (and with some modifications also parameters) of a nonlinear, dynamic, discrete-time system. This system may for example represent a building, an HVAC plant or a chiller. The state and output equations are

$$\mathbf{x}(t_{k+1}) = f(\mathbf{x}(t_k), \mathbf{u}(t_k), \Theta(t), t) + \mathbf{q}(t_k), \tag{1a}$$

$$\mathbf{y}(t_k) = H(\mathbf{x}(t_k), \mathbf{u}(t_k), \Theta(t), t) + \mathbf{r}(t_k), \tag{1b}$$

with initial conditions $\mathbf{x}(t_0) = \mathbf{x}_0$, where $f\colon \Re^n \times \Re^m \times \Re^p \times \Re \to \Re^n$ is nonlinear, $\mathbf{x}(\cdot) \in \mathbb{R}^n$ is the state vector, $\mathbf{u}(\cdot) \in \mathbb{R}^m$ is the input vector, $\Theta(\cdot) \in \mathbb{R}^p$ is the parameter vector, $\mathbf{q}(\cdot) \in \mathbb{R}^n$ represents the process noise (i.e. unmodeled dynamics and other uncertainties), $\mathbf{y}(\cdot) \in \mathbb{R}^o$ is the output vector, $H\colon \Re^n \times \Re^m \times \Re^p \times \Re \to \Re^o$ is the output measurement function and $\mathbf{r}(\cdot) \in \mathbb{R}^o$ is the measurement noise.

The UKF is based on the typical prediction-correction style methods:

1. PREDICTION STEP: predict the state and output at time step t_{k+1} by using the parameters and states at t_k.

2. CORRECTION STEP: given the measurements at time t_{k+1}, update the posterior probability, or uncertainty, of the states prediction using Bayes' rule.

The original formulation of the UKF imposes some restrictions on the model because the system needs to be described by a system of initial-value, explicit difference equations (1). A second drawback is that the explicit discrete time system in (1) cannot be used to simulate stiff systems efficiently. The UKF should be translated in a form that is able to deal with continuous time models, possibly including events.

Although physical systems are often described using continuous time models, sensors routinely report

DOI 10.3384/ECP14096647

time-sampled values of the measured quantity (e.g. temperatures, pressures, positions, velocities, etc.). These sampled signals represent the available information about the system operation and they are used by the UKF to compute an estimation for the state variables.

A more natural formulation of the problem is represented by the following continuous-discrete time model

$$\frac{d\,\mathbf{x}(t)}{dt} = F(\mathbf{x}(t),\mathbf{u}(t),\Theta(t),t), \tag{2a}$$

$$\mathbf{x}(t_0) = \mathbf{x}_0, \tag{2b}$$

$$\mathbf{y}(t_k) = H(\mathbf{x}(t_k),\mathbf{u}(t_k),\Theta(t),t_k) + \mathbf{r}(t_k), \tag{2c}$$

where the model is defined in the continuous time domain, but the outputs are considered as discrete time signals sampled at discrete time instants t_k. The original problem described in equation (1) can be easily derived as

$$\begin{aligned}\mathbf{x}(t_{k+1}) &= f(\mathbf{x}(t_k),\mathbf{u}(t_k),\Theta(t_k),t_k)\\ &= \mathbf{x}(t_k) + \\ &\quad \int_{t_k}^{t_{k+1}} F(\mathbf{x}(t),\mathbf{u}(t),\Theta(t),t)\,dt\end{aligned} \tag{3}$$

Our implementation uses this continuous-discrete time formulation and the numerical integration is done using PyFMI that works with a model embedded as an FMU. Despite not shown in (2) and (3) the model may contain events that are handled by the numerical solver provided with the PyFMI package.

The UKF is based on the the Unscented Transformation [1] (UT), which uses a fixed (and typically low) number of deterministically chosen sigma-points[1] to express the mean and covariance of the original distribution of the state variables $\mathbf{x}(\cdot)$, exactly, under the assumption that the uncertainties and noise are Gaussian [1]. These sigma-points are then propagated simulating the nonlinear model (2) and the mean and covariance of the state variables are estimated from them. This is significantly different from Monte Carlo approaches because the UKF chooses the points in a deterministic way. One of the most important properties of this approach is that if the prior estimation is distributed as a Gaussian random variable, the sigma points are the minimum amount of information needed to compute the exact mean and covariance of the posterior after the propagation through the nonlinear state function [6].

[1]The sigma-points can be seen as the counterpart of the particles used in Monte Carlo methods.

Figure 1 illustrates the filtering process which we will now explain. At time t_k, a measurement of the outputs $\mathbf{y}(t_k)$, the inputs and the previous estimation of the state are available. Simulations are performed starting from the prior knowledge of the state $\hat{\mathbf{x}}(t_{k-1})$, using the input $\mathbf{u}(t_{k-1})$. Once the results of the simulations $\hat{x}_{sim}(t_k)$ and $\hat{y}_{sim}(t_k)$ are available, they are compared against the available measurements in order to correct the state estimation. The corrected value (i.e. filtered) becomes the actual estimation. Because of its speed, the estimation can provide near-real-time updates, since the time spent for simulating the system and correcting the estimation is typically shorter than the sampling time step, in particular for building or HVAC applications, where computations take fractions of second and sampling intervals are seconds or minutes.

The Algorithm 1 summarize the steps performed by the UKF. The interested reader can find more information and details of the actual implementation in [7].

2.3 Smoothing to Improve UKF Estimation

In this subsection, we discuss an additional refinement procedure to the UKF. The distribution $P(\mathbf{x}(t_k)|\mathbf{y}(t_1),\ldots,\mathbf{y}(t_k))$ is the probability to observe the state vector $\mathbf{x}(t_k)$ at time t_k given all the measurements collected. By using more data $P(\mathbf{x}(t_k)|\mathbf{y}(t_1),\ldots,\mathbf{y}(t_k),\ldots\mathbf{y}(t_{k+N}))$, the posterior distribution can be improved through recursive smoothing. Hence, the basic idea behind the recursive smoothing process is to incorporate more measurements before providing an estimation of the state. Figure 2 represents the smoothing process. While the filter works forwardly on the data available, and recursively provides a state estimation, a smoothing procedure back-propagates the information obtained during the filtering process, after some amount of data becomes available, in order to improved the estimation previously provided [8].

The smoothing process can be viewed as a delayed, but improved, estimation of the state variables. The longer the acceptable delay, the bigger the improvement since more information can be used. For example if, at a given time, a sensor provides a wrong measurement, the filter may not be aware of this and it may provide an estimation that does not correspond to the real value (although the uncertainty bounds will still be correct). The smoother observes the trend of the estimation will reduce this impact of the erroneous data, thus providing an estimation that is less sensitive to measurement errors.

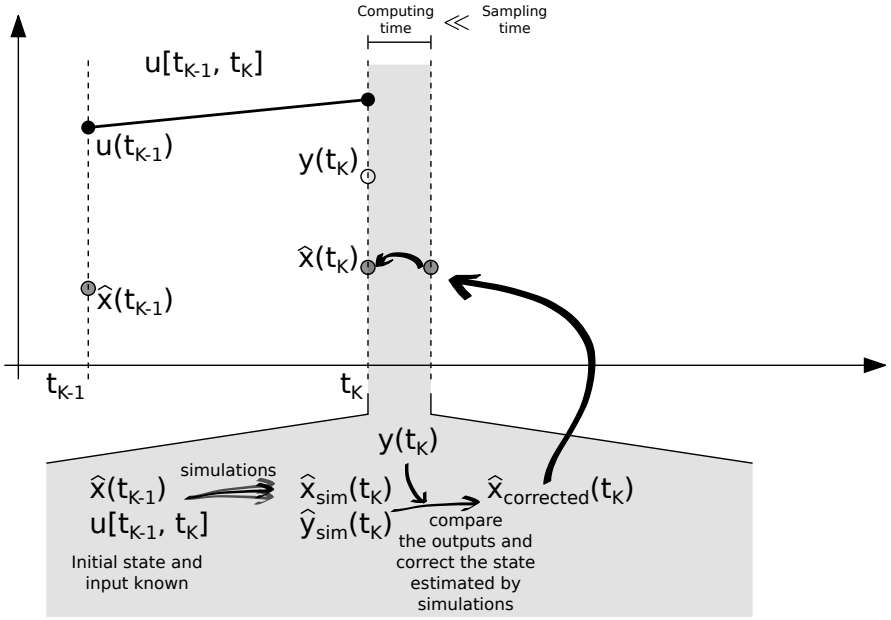

Figure 1: On-line state estimation filtering procedure.

Procedurally, the smoothing algorithm starts from a user-specified point in the data stream and back updates the previously filtered estimation. Algorithm 2 describes the smoothing procedure.

2.4 Parameter Estimation

The importance of the state estimation has been stressed, and we described the UKF and Smoother as solutions to this problem. While state estimation is particularly important for controls, parameter estimation is important for model calibration and fault detection and diagnostics. Consider, for example an heat exchanger. Suppose it is characterized by one heat exchange coefficient that influences the heat transfer rate between the two fluids. During the design of the heat exchanger it is possible to compute an approximation of it. However, it is not possible to know its value exactly. After the heat exchanger is created, identifying the value of it is important to verify if the design requirements have been met. Another example is real-time monitoring in which it is continuously monitored during the operation in order to continuously check if it has been reduced by fouling and the heat exchanger need to be serviced.

Continuous parameter estimation is possible by extending the capabilities of the UKF and Smoother to estimate not just the state variable, but also the parameters of the system. The approach is to include the parameter in an augmented state $\mathbf{x}^A(\cdot)$, defined as

$$\mathbf{x}^A(\cdot) = \begin{bmatrix}\mathbf{x}(\cdot) & \mathbf{x}^P(\cdot)\end{bmatrix}^T, \tag{4}$$

where $\mathbf{x}^P(\cdot) \subseteq \Theta(\cdot)$ is a vector containing a subset of the full parameter vector $\Theta(\cdot)$ to be estimated. The new components of the state variables need a function that describe their dynamics. Since in the normal operation, these values are constant, the associated dynamic is

$$\frac{d\,\mathbf{x}^P(t)}{dt} = \mathbf{0}, \tag{5}$$

where $\mathbf{0}$ is a null vector. These null dynamics have to be added (2). The result is a new continuous-discrete time system

$$\frac{d\,\mathbf{x}^A(t)}{dt} = F_A(\mathbf{x}^A(t), \mathbf{u}(t), \Theta(t), t) \tag{6a}$$

$$\mathbf{y}(t_k) = H(\mathbf{x}^A(t_k), \mathbf{u}(t_k), \Theta(t_k), t_k) + \mathbf{r}(t_k), \tag{6b}$$

with

$$F_A(\mathbf{x}^A(t), \mathbf{u}(t), \Theta(t), t) = \begin{bmatrix} F(\mathbf{x}(t), \mathbf{u}(t), \Theta(t), t) \\ \mathbf{0} \end{bmatrix} \tag{7}$$

Note that augmenting the state variables leads to a nonlinear state equation even if $F(\cdot,\cdot,\cdot,\cdot)$ is a linear function. Therefore, for parameter estimation, a nonlinear filtering and smoothing technique is required.

2.5 Implementation

The former sections explained on the state and parameter estimation. This section describes the software implementation and describes specific issues that we addressed.

DOI 10.3384/ECP14096647

Algorithm 1: Filtering

Notation: The superscript $^{(i)}$ indicates that the quantity is related to the $i-th$ sigma point, $w_m^{(i)}$ and $w_c^{(i)}$ are the weights associated to the $i-th$ sigma point, n is the dimension of the state vector $\mathbf{x}(\cdot)$, and $[\cdot]_i$ is an operator that if applied to a matrix A returns its $i-th$ column. Vectors are indicated with bold characters.

Given the initial knowledge of the state distribution $\mathbf{x}(t_0) \sim N(\mu_0, P_0)$, and given the output measurement covariance matrix R_0

1. Initialize $k = 0$ and set parameters α, β, λ (with $0 \leq \alpha \leq 1$ – other configuration details in [2])
2. Define $2n+1$ sigma-points

$$\begin{aligned}
\mathbf{x}(t_k)^{(0)} &= \mu_k, \\
\mathbf{x}(t_k)^{(i)} &= \mu_k + \left[\sqrt{(n+\lambda)P_k}\right]_i, \; i = 1 \ldots n, \\
\mathbf{x}(t_k)^{(i)} &= \mu_k - \left[\sqrt{(n+\lambda)P_k}\right]_{i-n}, \; i = n+1 \ldots 2n.
\end{aligned}$$

3. Compute the weights associated to each sigma-point

$$\begin{aligned}
w_m^{(0)} &= \lambda/(n+\lambda), \\
w_c^{(0)} &= \lambda/(n+\lambda) + (1-\alpha^2+\beta), \\
w_m^{(i)} &= 1/2(n+\lambda), \; i = 1 \ldots 2n, \\
w_c^{(i)} &= 1/2(n+\lambda), \; i = 1 \ldots 2n.
\end{aligned}$$

4. Compute the predicted state (i.e. perform a simulation) for each sigma-point, and the predicted weighted mean, and the predicted covariance

$$\begin{aligned}
\mathbf{x}(t_{k+1})^{(i)} &= f(\mathbf{x}(t_k)^{(i)}, \mathbf{u}(t_k), \Theta(t_k), t_k), \; i = 0 \ldots 2n+1, \\
\mu_{k+1}^- &= \sum_{i=0}^{2n+1} w_m^{(i)} \mathbf{x}(t_{k+1})^{(i)}, \\
P_{k+1}^- &= P_k + \sum_{i=0}^{2n+1} w_c^{(i)} \left(\mathbf{x}(t_{k+1})^{(i)} - \mu_{k+1}^-\right)\left(\mathbf{x}(t_{k+1})^{(i)} - \mu_{k+1}^-\right)^T.
\end{aligned}$$

5. Redefine the new sigma-points $\mathbf{x}(t_{k+1})^{(i)}$ using the predicted mean μ_{k+1}^-, and covariance P_{k+1}^- as shown in step (1).
6. Compute the measured outputs using the new sigma-points, then compute the mean output $\hat{\mathbf{y}}_{k+1}$ and its covariance S_{k+1}

$$\begin{aligned}
\mathbf{y}(t_{k+1})^{(i)} &= H(\mathbf{x}(t_{k+1})^{(i)}, \mathbf{u}(t_{k+1}), \Theta(t_{k+1}), t_{k+1}), \; i = 0 \ldots 2n+1, \\
\hat{\mathbf{y}}_{k+1} &= \sum_{i=0}^{2n+1} w_m^{(i)} \mathbf{y}(t_{k+1})^{(i)}, \\
S_{k+1} &= R_0 + \sum_{i=0}^{2n+1} w_c^{(i)} \left(\mathbf{y}(t_{k+1})^{(i)} - \hat{\mathbf{y}}_{k+1}\right)\left(\mathbf{y}(t_{k+1})^{(i)} - \hat{\mathbf{y}}_{k+1}\right)^T.
\end{aligned}$$

7. Compute the cross covariance between the state and the output

$$C_{k+1} = \sum_{i=0}^{2n+1} w_c^{(i)} \left(\mathbf{x}(t_{k+1})^{(i)} - \mu_{k+1}^-\right)\left(\mathbf{y}(t_{k+1})^{(i)} - \hat{\mathbf{y}}_{k+1}\right)^T.$$

8. Compute the filter gain and update the predicted mean and covariance of the state

$$\begin{aligned}
K &= C_{k+1} S_{k+1}^{-1}, \\
\mu_{k+1} &= \mu_{k+1}^- + K\left[\mathbf{y}_{k+1} - \hat{\mathbf{y}}_{k+1}\right], \\
P_{k+1} &= P_{k+1}^- - K\, S_{k+1}\, K^T.
\end{aligned}$$

9. Compute the state estimation as $\hat{\mathbf{x}}(t_{k+1}) \sim N(\mu_{k+1}, P_{k+1})$.
10. Increment k, and go to step (2).

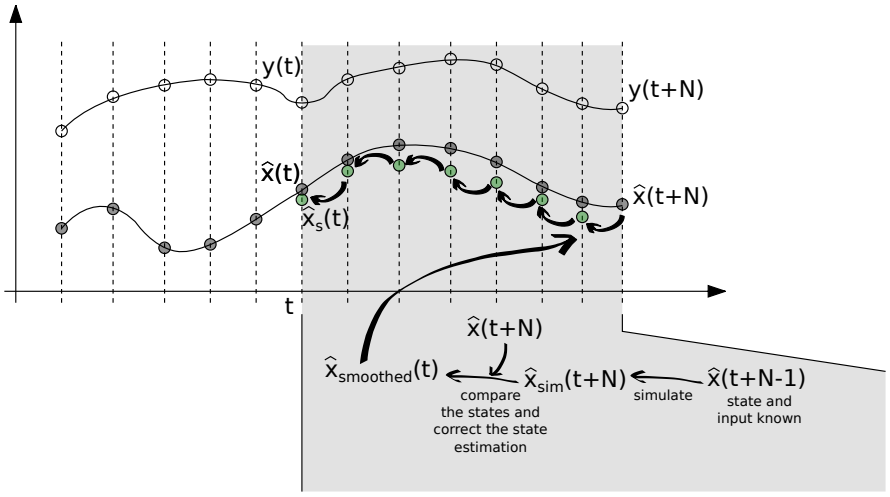

Figure 2: Improving the estimation using the smoothing. Backward propagations of the filtering results can improve the former estimation.

Algorithm 2: Smoothing

1. Initialize $k \geq 0$ and define the amplitude of the smoothing window $N > 0$ such as data at time t_{k+N} are available.
2. Initialize $j = 1$.
3. Draw the $1+2n$ sigma-points $\mathbf{x}(t_{k+N-j})^{(i)}$ using $\mathbf{x}(t_{k+N-j})$ as mean value, and P_{k+N-j} as covariance matrix.
4. Propagate the sigma points through the dynamic model (prediction step) and compute the mean, covariance and cross covariance

$$\begin{aligned}
\mathbf{x}(t_{k+N-j+1})^{(i)} &= f(\mathbf{x}(t_{k+N-j})^{(i)}, \mathbf{u}(t_{k+N-j}), \Theta(t_{k+N-j}), t_{k+N-j}) \ , \ i = 0 \ldots 2n+1, \\
\mu^{-}_{k+N-j+1} &= \sum_{i=0}^{2n+1} w_m^{(i)} \mathbf{x}(t_{k+N-j+1})^{(i)}, \\
P^{-}_{k+N-j+1} &= P_k + \sum_{i=0}^{2n+1} w_c^{(i)} \left(\mathbf{x}(t_{k+N-j+1})^{(i)} - \mu^{-}_{k+N-j+1} \right) \left(\mathbf{x}(t_{k+N-j+1})^{(i)} - \mu^{-}_{k+N-j+1} \right)^T, \\
C_{k+N-j+1} &= \sum_{i=0}^{2n+1} w_c^{(i)} \left(\mathbf{x}(t_{k+N-j+1})^{(i)} - \mu^{-}_{k+N-j+1} \right) \left(\mathbf{x}(t_{k+N-j})^{(i)} - \mathbf{x}(t_{k+N-j}) \right)^T.
\end{aligned}$$

5. Compute the smoother gain, and post-correct the previous estimation providing the smoothed mean and covariance

$$\begin{aligned}
K &= C_{k+N-j+1} \left[P^{-}_{k+N-j+1} \right]^{-1}, \\
\mu^{s}_{k+N-j} &= \mu^{-}_{k+N-j+1} + K \left[\mathbf{x}(t_{k+N-j+1}) - \mu^{-}_{k+N-j+1} \right], \\
P^{s}_{k+N-j} &= P_{k+N-j} + K \left[P^{s}_{k+N-j+1} - P^{-}_{k+N-j+1} \right] K^T.
\end{aligned}$$

6. If $j < N$ then set $j := j+1$ and go to step (2), otherwise exit.

N.B. At the end of each time step the state is estimated as $\hat{\mathbf{x}}^s(t_{k+N-j}) \sim N(\mu^s_{k+N-j}, P^s_{k+N-j})$.

The first barrier to overcome was that the state estimation procedure refers to the full vector of state variables $\mathbf{x}(\cdot)$. However in many applications, the number of state variables is higher than the ones to be estimated. An example are sensors that sometimes contain a unitary gain first order filter. Including all the state variables in the estimation is not desirable because

- the number of sigma points required by the UKF, and thus the number simulations to be run, grows with the number of state variables.
- for every state variable or parameter estimated the user must provide an initial guess for mean value and the covariance.

Therefore, the user is allowed to select a subset of the state variables and parameters to be estimated by the UKF algorithm.

Similarly, an FMU may have several outputs, but only few of them may be measured and used by the UKF. The outputs for which a measurement is available and used by the UKF are denominated measured

DOI 10.3384/ECP14096647

outputs. The measured outputs requires additional information such the covariance (i.e., the uncertainty associated to the measure) and a time serie containing the data.

All the inputs and the measured outputs have to be associated with a time serie. This can be done by associating them with a specific column of a csv file.

Another issue has been encountered regarding the range of validity of states and parameters. For example, the level of water contained in a tank has to be positive but not greater than the height of the tank. We addressed this issue by constraining the sigma points within user specified upper and lower limits. As default, the min and max values specified in the FMU model description file are used.

We also implemented the ability to run the simulations in parallel. The computationally demanding part of the UKF and Smoother is the time integration done using PyFMI. However this section of the algorithm is entirely parallelizable. By default, our implementation uses one less thread as there are processors to run the simulation, while a thread manages the simulation and collects the results. The simulation results generated by PyFMI can optionally be written to files or not.

All these functionalities have been embedded in few classes that use PyFMI. In particular we have defined a new class representing the FMU model in a more general term since the state estimation is a task that involves more than a simple simulation, which is the aim of PyFMI.

2.6 Code snippet

This subsection contains a code snippet that illustrates how the FMU-based state and parameter estimation framework works. In this example the FMU model represents a mass-spring-damper system, where the input is the force F applied to the mass, and the measured output is the mass acceleration a. The two corresponding data series are stored in a csv file named data.csv. The states to be estimated are the position x and its velocity v.

```
# Path of the FMU model
filePath = "./model.fmu"

# Instantiate the model
m = Model(filePath, atol=1e-5, rtol=1e-4)

# Path of the CSV file that contains the data series
csvPath = "./data.csv"

# Associate the columns of the csv file to the input
input = m.GetInputByName("F")
input.GetCsvReader().OpenCsv(csvPath)
input.GetCsvReader().SetSelectedColumn("Force")

# Associate the columns of the csv file to the output
output = m.GetOutputByName("a")
output.GetCsvReader().OpenCsv(csvPath)
output.GetCsvReader().SetSelectedColumn("acceleration")

# Specify that this output has to be compared against
# measured data contained in the CSV file
output.SetMeasuredOutput()

# Specify output measurement covariance
output.SetCovariance(1.0)

# Specify the subset of the states to be estimated
m.AddVariable(m.GetVariableObject("x"))
m.AddVariable(m.GetVariableObject("v"))

# Get a reference to the states to be estimated
var_x = m.GetVariables()[0]
var_v = m.GetVariables()[1]

# Specify initial value for the position
# and the boundary limits (e.g., position x must be positive)
var_x.SetInitialValue(2.5)
var_x.SetCovariance(0.5)
var_x.SetMinValue(0.0)

# Specify initial value for the velocity
var_y.SetInitialValue(0.0)
var_y.SetCovariance(0.2)

# Initialize simulator
m.InitializeSimulator()

# Instantiate UKF and pass to it the model
UKF = ukfFMU(m)

# Run the filter from 0.0 to 10.0 seconds
time, X, Sx, y, Sy = UKF.filter(start = 0.0, stop = 10.0)

# plotting...
```

This framework reduces the effort to set up a state or parameter estimation. The model can be created in Modelica and directly imported as an FMU, avoiding the user to rewrite the model in the right format required by the UKF. Other functionalities provide an easy way to specify the state variables or parameter to be estimated, together with the data series to be used as inputs and outputs.

3 Application: FDD

This section contains an example that shows how the FMU-based state and parameter estimation algorithm can be used for fault detection and diagnosis.

FDD algorithms based on state estimation techniques are known to be more suitable than other approaches based on neural networks or principal component analysis for detecting multiple faults [9, 10, 11], they can compute the fault probabilities and they provide an indication of what is not working as expected in the system. All these features are possible thank to the probabilistic description of the state variables and parameters the estimation techniques provide.

A drawback of state estimation based strategies is that they require a slightly higher modeling effort, e.g., to include explicit fault descriptions in the model itself. However, various open-source modeling tools (e.g. OpenModelica and JModelica), and modeling libraries for buildings (Modelica Buildings library [12]) are available, and they provide two main advantages: reducing the effort required to set up the model, and

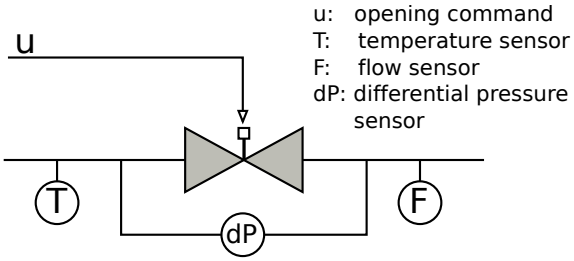

Figure 3: Schematic of the system.

reducing the risk of modeling errors since they are typically validated and tested.

The availability of these tools and libraries together with the FMU-based state and parameter estimation algorithm put in place a framework for creating with a low level of effort a model-based FDD algorithm.

The considered system is a valve that regulates the water flow rate in a water distribution system (see Figure 3). The system is described by the following equations

$$\dot{m}(t) = \phi(x(t))A_v\sqrt{\rho(t)}\sqrt{\Delta p(t)}, \tag{8a}$$

$$x(t) + \tau\dot{x}(t) = u(t), \tag{8b}$$

where $\dot{m}(\cdot)$ is the mass flow rate passing through the valve, $\Delta p(\cdot)$ is the pressure difference across it, $u(\cdot)$ is the valve opening command signal, $x(\cdot)$ is the valve opening position, τ is the actuator time constant, $\phi(\cdot)$ is the power-law opening characteristic, A_v is the flow coefficient and $\rho(\cdot)$ is the fluid density (please note that the square root of the pressure difference is regularized around zero flow in order to prevent singularities in the solution). The system has three sensors (see Figure 3) that respectively measure the pressure difference across the valve, the water temperature $T(\cdot)$ and the mass flow rate passing through it. All the sensors are affected by measurement noise. In addition, the mass flow rate sensor is also affected by a thermal drift.

$$T^N(t) = T(t) + \eta_T(t) \tag{9a}$$

$$\Delta p^N(t) = \Delta p(t) + \eta_P(t) \tag{9b}$$

$$\dot{m}^{N+D}(t) = (1 + \lambda(T(t) - T_{ref}))\dot{m}(t) + \eta_m(t) \tag{9c}$$

The measurement equations are described in (9), where the superscript ^{N} indicates a measurement affected by noise, the superscript $^{N+D}$ indicates the presence of both noise and thermal drift, T_{ref} is the reference temperature at which the sensor has no drift, λ is the thermal drift coefficient and $\eta_T(\cdot)$, $\eta_P(\cdot)$, and $\eta_m(\cdot)$ are three uniform white noises affecting respectively the temperature, pressure and mass flow rate

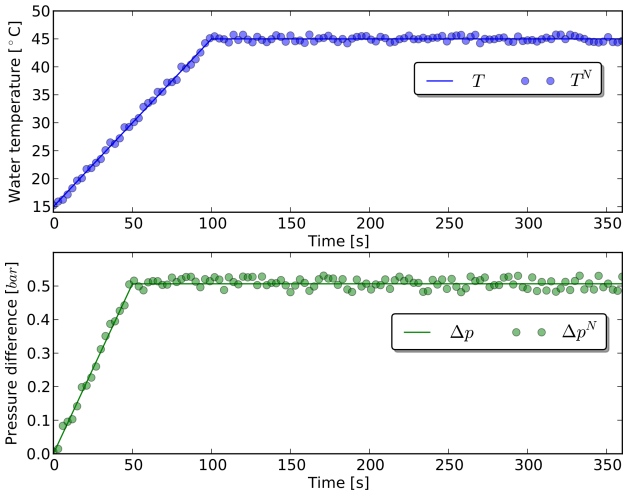

Figure 4: Input signals of the faulty valve model: water temperature (blue) and pressure difference (green). The lines represent the data generated by simulation, while the dots represent the sampled and noisy version provided to the UKF.

measurements. These signals are sampled every two seconds.

Suppose during the operation, at t = 80 s, the valve becomes faulty. The fault affects the ability of the valve to control its opening position. The valve opening cannot go below 20% (causing a leakage) and over 60% (it gets stuck). At t = 250 s, the valve stuck position moves from 60% to 90%.

The fault identification procedure is asked to identify whether the valve not works as expected, that is its opening position follows the command signal. The fault identification is performed using the UKF that uses as input signals for its model the noisy pressure difference (see Figure 4), the noisy water temperature (see Figure 4) and the command signal (see Figure 6). The command signal is noise free because it is computed by some external controller and not measured. The UKF compares the output of its simulations with the measured mass flow rate (see Figure 5) that is affected by both noise and thermal drift. The effect of the thermal drift is visible in Figure 5 where the green dots represent the measured mass flow rate while the green line is the actual mass flow rate passing through the valve.

The UKF and the smother estimate the value of the state variable $x(t)$ representing the valve opening position and the parameter λ, the thermal drift coefficient. The length of the augmented state is $n = 2$. Hence for every estimation step, the UKF performs $1 + 2 \times 2 = 5$ simulations. The UKF has the initial conditions $x(0) \sim N(0.8, 0.05)$ and $\lambda \sim N(0, 0.7 \cdot 10^{-3})$, the output noise

DOI 10.3384/ECP14096647

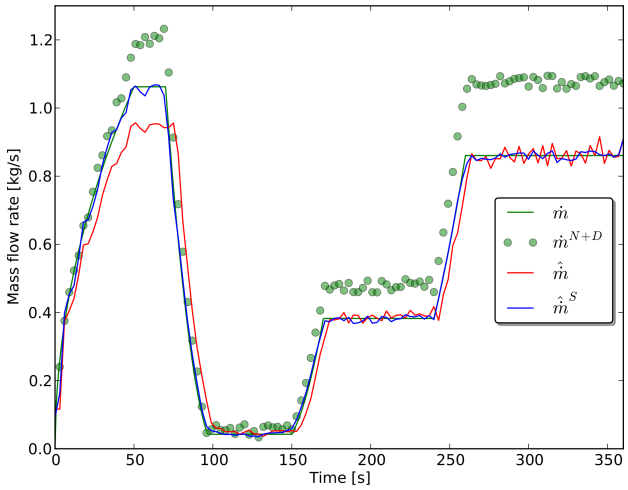

Figure 5: The green line represent the mass flow rate that is passing through the valve. The green dots are the measurements of the mass flow rate (affected by noise and sensor thermal drift) used by the UKF. The red line is the UKF estimation, while the blue line is the smoother estimation.

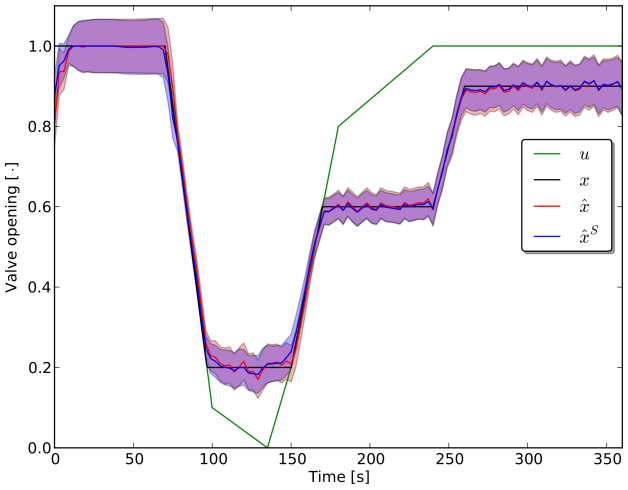

Figure 6: The green line is the opening valve signal. The blue line is the actual opening value affected by faults. The red and blue lines are the UKF and smoother estimations of the valve opening position (the area surrounding the estimation is the σ-confidence interval).

covariance matrix is $R = [0.05]$, and the filter coefficients are $\alpha = \frac{1}{\sqrt{3}}$, $\beta = 2$ and $k = 3 - n = 1$.

As shown in Figure 5, the measured mass flow rate (green dots) are far from the real mass flow rate (green line). Despite the measurement error, the UKF and the smoother provide an estimation with a good accuracy (red and blue lines in Figure 5). The mass flow rates computed by the UKF, $\hat{\dot{m}}$, and the smoother, $\hat{\dot{m}}^S$, are close to the real one, $\dot{m}$, because they are able to estimate both the valve opening position and the sensor drift coefficient, as shown in Figures 6 and 7. As expected the smoother is able to provide a better estimation since it uses more data. The time spent by the UKF to perform the simulations and computing the estimations was about 0.05 s, that is lower than the sampling time of 2 s. The speed of this UKF based FDD algorithm allows a real-time implementation for this particular application.

4 General Applicability

The UKF and in general state/parameter estimation techniques are well known solutions used in many fields. We presented an approach that reduces the effort needed to set up a state and parameter estimation for models created with simulation programs that are FMI compliant. The example shows how this tool can be used for FDD purposes in the context of HVAC systems. However, this tool can be used in other contexts. For example, state and parameter estimation techniques are used to support guidance and control systems in the automotive, robotic and aerospace industries.

5 Conclusion

We proposed a model-based state and parameter estimator for dynamic systems described using the FMI standard. The paper explained the nonlinear state estimation and smoothing techniques employed by the algorithm, together with the details necessary to implement it. The last section shows how the FMU-based state and parameter estimator can be used to set up a fault detection algorithm capable of identifying faults in a valve, even in presence of wrong and noisy measurements.

This algorithm extends the functionalities of any simulation program that implements the FMI standard version 1.0. As shown in the example, this algorithm has a direct application in FDD, but it can also be used for model calibration and process control together with adaptive or model predictive control schemes. The main advantage of this algorithm is that it allows to reuse models from the design phase, it extends the capabilities of FMI compliant modeling frameworks and it reduces the time and investments necessary to develop advanced control strategies, calibration and FDD.

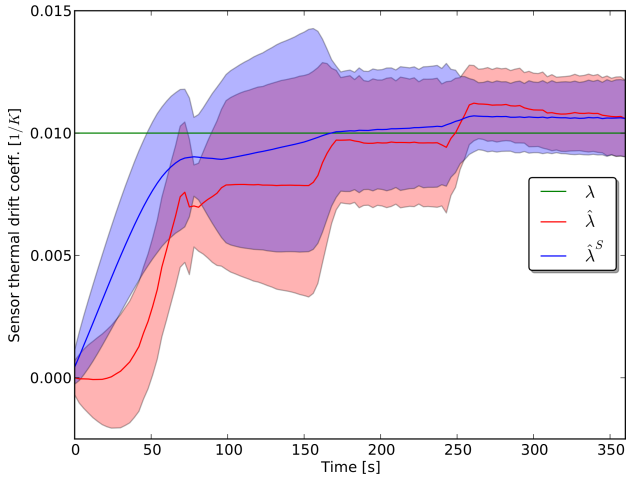

Figure 7: The green line is the sensor thermal drift coefficient, while the red and blue lines are the estimations of the thermal drift coefficient computed by the UKF and smoother (the area surrounding the estimation is the σ-confidence interval).

6 Acknowledgements

This research was supported by the Assistant Secretary for Energy Efficiency and Renewable Energy, Office of Building Technologies of the U.S. Department of Energy, under Contract No. DE-AC02-05CH11231. The research was also supported by the U.S. Department of Defense under the ESTCP program.

The authors thank Mary Ann Piette, Jessica Granderson, Oren Shetrit, Wangda Zuo, and Rong Lily Hu for the support provided through the project.

References

[1] S. J. Julier and J. K. Uhlmann. A general method for approximating nonlinear transformations of probability distributions. *Robotics Research Group Technical Report, Department of Engineering Science, University of Oxford*, pages 1–27, November 1996.

[2] S.J. Julier. The scaled unscented transformation. In *American Control Conference, 2002. Proceedings of the 2002*, volume 6, pages 4555–4559 vol.6, 2002.

[3] Simon S Haykin et al. *Kalman filtering and neural networks*. Wiley Online Library, 2001.

[4] D. Crisan and Arnaud Doucet. A survey of convergence results on particle filtering methods for practitioners. *Signal Processing, IEEE Transactions on*, 50(3):736–746, 2002.

[5] Modelon AB. PyFMI a package for working with dynamic models compliant with the functional mock-up interface standard, September 2013.

[6] E.A. Wan and R. Van der Merwe. The unscented kalman filter for nonlinear estimation. In *Adaptive Systems for Signal Processing, Communications, and Control Symposium 2000. AS-SPCC. The IEEE 2000*, pages 153–158, 2000.

[7] S. Sarkka. On unscented kalman filtering for state estimation of continuous-time nonlinear systems. *Automatic Control, IEEE Transactions on*, 52(9):1631–1641, 2007.

[8] S. Sarkka. Unscented rauch–tung–striebel smoother. *Automatic Control, IEEE Transactions on*, 53(3):845–849, 2008.

[9] Venkat Venkatasubramanian, Raghunathan Rengaswamy, Kewen Yin, and Surya N. Kavuri. A review of process fault detection and diagnosis: Part I: Quantitative model-based methods. *Computers & Chemical Engineering*, 27(3):293 – 311, 2003.

[10] Venkat Venkatasubramanian, Raghunathan Rengaswamy, and Surya N Kavuri. A review of process fault detection and diagnosis: Part II: Qualitative models and search strategies. *Computers & Chemical Engineering*, 27(3):313 – 326, 2003.

[11] Venkat Venkatasubramanian, Raghunathan Rengaswamy, Surya N. Kavuri, and Kewen Yin. A review of process fault detection and diagnosis: Part III: Process history based methods. *Computers & Chemical Engineering*, 27(3):327 – 346, 2003.

[12] Michael Wetter, Wangda Zuo, Thierry S Nouidui, and Xiufeng Pang. Modelica buildings library. *Journal of Building Performance Simulation*, (In press):1–18, 2013.