Nonlinear State Estimation with FMI: Tutorial and Applications

Christopher R. Laughman Scott A. Bortoff

Mitsubishi Electric Research Laboratories, Cambridge, MA, USA
{laughman,bortoff}@merl.com

Abstract

One of the key uses enabled by the functional mockup interface (FMI) standard is the ability to combine Modelica models governed by differential-algebraic equations with measurement data to systematically estimate unmeasured quantities in physical systems. While it is clear how this might be done in theory, many implementation challenges can make this difficult in practice. This paper provides a tutorial connecting the mathematical formulation of two different estimators, the extended Kalman filter (EKF) and the ensemble Kalman filter (EnKF), to an FMI-based Modelica implementation of these estimators. The efficacy of these methods are demonstrated on an example of a small motor model and a larger thermodynamic model of a building, and some of the advantages and disadvantages of this FMI-based approach to estimation are discussed, as well as limitations of FMI associated with constraint management for these estimation methods. The code for the motor example is publicly available and is attached to this publication.

Keywords: observers, state estimation, extended Kalman filter, ensemble Kalman filter, functional mockup interface (FMI)

1 Introduction

Information about unmeasured physical quantities is often desired when designing complex engineered systems to improve system control, implement performance monitoring, or perform fault detection and analysis. Data about such variables may not be available for a variety of reasons, such as sensor cost or the fact that quantities of interest may be very difficult or impossible to measure directly. As an example, measurements of the amount of heating or cooling energy delivered by an HVAC system to an occupied space in a building would be useful in the design of improved air temperature controllers, but accurate estimates of this variable are difficult to obtain because they depend on local temperature differences and airflow rates that cannot be easily characterized.

This problem is commonly addressed by combining sensor data with a model of the system, thereby leveraging knowledge about the system structure which cannot be directly inferred from a given set of sensor data. Though a wide variety of modeling approaches can be used depending on the type of system under study, models of physical systems often benefit from the use of equation-oriented tools such as Modelica (Modelica Association, 2017) because their physics-based structure allows the internal states to have a meaningful interpretation, and their first-principles construction tends to produce good extrapolative performance over high-dimensional range of potential operating conditions.

A variety of estimation and observer-based techniques, including the range of Kalman filters and particle filters, have been developed since the mid-twentieth century to solve the problem of estimating unknown system quantities of interest given a model and a set of observations. These methods can be more precisely described by considering a system model described by a set of ordinary differential equations

$$\dot{x} = \mathcal{M}(x,u,t;\mu) + \mathcal{W} \quad (1)$$

$$y = \mathcal{H}(x,u,t;\mu) + \mathcal{V} \quad (2)$$

$$z = \mathcal{G}(x,u,t;\mu), \quad (3)$$

where $\mathcal{M}$ is the forward model operator, $\mathcal{H}$ is the observation operator, $\mathcal{G}$ is the map from the states and inputs to the performance variables of interest, x represents the system state, u represents the system input including both the control inputs and the unmeasured or measured disturbances, μ represents the system parameters (which we will assume for the present work are known), and $\mathcal{W}$ and $\mathcal{V}$ represent the process and observation noise with covariances Q and R, respectively. We assume that the physical system is governed by the model $(\mathcal{M},\mathcal{H})$, while the noise terms are included in our approximate model representation to describe model and measurement errors. In this context, these methods are designed to produce an estimate x that minimizes a metric related to the error between the model output y and the measured system output y_m, with the expectation that the estimated system state is sufficiently close to that of the plant that the performance variables will accurately describe the variables of interest.

Unfortunately, the Modelica language is not designed to implement these state estimation methods, as it cannot readily update the state vector for the compiled model to incorporate additional measurement information. The functional mockup interface (FMI) standard (Modelica Association, 2019) was thus created, in part, to enable the use of Modelica models in this setting. The co-simulation interface provides an efficient method to evaluate the right-hand sides of Equations 1-3 at a given time step and correct the state vector to assimilate updated sensor data, which enables the implementation of these state

estimation methods on top of the existing Modelica models.

Early work in using FMI 1.0 to implement state estimation methods was presented by Brembeck et al. (2011), which demonstrates the use of weighted least-squares and Kalman filters to estimate the state-of-charge for a lithium-ion battery. This work was significantly extended by Brembeck et al. (2014) to cover a draft implementation of FMI 2.0 standard for co-simulation, and test implementations of an extended Kalman filter and a moving horizon estimator are demonstrated on an electric vehicle application. Brembeck (2019) further develops this work and implements a number of practical refinements on the state-of-charge estimator. Other recent work done by Vytvytskyi and Lie (2019) compares the performance of unscented Kalman filters and ensemble Kalman filters for state estimation in hydropower plants and finds that these methods work well in this application. The present paper is an extension of the work presented by Bortoff and Laughman (2019), in which we demonstrated the use of a model-exchange functional mockup unit (FMU) to construct an extended Luenberger observer (ELO) with output injection to estimate the heat capacity of a water-cooled fan coil unit in a building.

While this prior work demonstrates the potential and some of the capabilities of using FMI to construct estimators based on Modelica models, an engineer with an interest in implementing these methods is faced with the daunting task of understanding the details of the FMI standard, as well as the API for the standard in a given language (e.g., Modelica, Python, or C++), before an estimator can be implemented for a given FMU. Moreover, a host of practical issues must be addressed in the process of this implementation that depend on the type of estimation method that is used. Larger Modelica models (tens to hundreds of states) are also associated with a variety of challenges that are not encountered for smaller models.

In this paper, we provide a tutorial-oriented description of two different types of estimators: an extended Kalman filter (EKF) (Simon, 2006) and a stochastic ensemble Kalman filter (EnKF) (Evensen, 2009a), so that a reader might obtain an improved understanding of the theory and process behind the implementation of these methods using the FMI. The EKF represents a traditional Kalman filter formulation for a nonlinear model, while the EnKF represents a particle-based approach which is similar to the EKF, but seeks a reduced computational effort by avoiding the integration of the covariance matrix. Both of these methods are first implemented in this paper on a simple model of an electrical machine, and then on a much larger model that describes the thermofluid dynamics of a building application, to provide an indication of the differences inherent in using them for practical Modelica models. While we could conceivably instantiate the FMU via any available API (e.g., pyFMI, FMPy, or other tools), we choose to instantiate the FMUs back into Modelica using the Dymola 2020 compiler (Dassault Systemes, 2019) because of the extensive Modelica support of the FMI interface and because this framework can easily be used to interface these estimators to other Modelica models.

The remainder of this paper is organized as follows: in Section 2, we will describe the theory of both the EKF and the EnKF in the context of the simple motor example, and connect the mathematical description of these estimators to the code listings provided in this paper. A complete implementation of this motor example is also available and is attached to this paper. Issues relating to the practical implementation of these estimators, as well as the simulation results from their application to the motor problem, will be discussed in Section 3. These same estimation methods will then be used in Section 4 to estimate the cooling capacity on a larger building model. Conclusions and directions for future development and work will briefly be discussed in Section 5.

2 Tutorial: EKF & EnKF with FMI

2.1 Extended Kalman Filter

As described in (Simon, 2006), the Kalman filter updates the state of a system model whenever measurements are available, and is the optimal variance-minimizing algorithm for linear systems with Gaussian process and measurement noise. While this state estimation method is thoroughly documented in many textbooks, it is helpful to review it because it is closely related to both the EKF and the EnKF. Assume a linear system model

$$x_k = Ax_{k-1} + Bu_{k-1} + w_k \quad (4)$$

$$y_k = Cx_k + v_k, \quad (5)$$

where w_k and v_k are the zero-mean, independent, and identically distributed Gaussian process and measurement noise with covariances Q and R. As a result we can characterize the statistical properties of the state by its mean and variance, e.g.,

$$\hat{x}_k = \mathbf{E}[x_k] \quad (6)$$

$$P_k = \mathbf{E}[(\hat{x}_k - x_k)(\hat{x}_k - x_k)^T]. \quad (7)$$

The Kalman filter proceeds in two phases to update and correct the state at time k given the state at $k-1$ and a set of measurements y_k. The first of these phases is referred to as the *forecast* step, which predicts the state at the next time step without the knowledge of any additional measurements, and the second phase is the *analysis* step, which corrects the state forecast using the received measurements. In the forecast step, we update the state from $k-1$ at time k by propagating forward the state and the covariance via the original system model, e.g.,

$$\hat{x}_k^f = A\hat{x}_{k-1}^a + Bu_{k-1} \quad (8)$$

$$P_k^f = AP_{k-1}^a A^T + Q, \quad (9)$$

where the covariance matrix P can be derived by explicitly calculating the variance from Equation 7. Now that the

state of the system has been propagated forward during the forecast step, we can correct the state during the analysis step using the measurements. This formulation minimizes the trace of the estimation error covariance at each time step.

$$\hat{x}_k^a = \hat{x}_k^f + K_k(y_k - C\hat{x}_k^f) \tag{10}$$

$$P_k^a = (I - K_k C)P_k^f \tag{11}$$

$$K_k = P_k^f C^T (CP_k^f C^T + R)^{-1} \tag{12}$$

The state is corrected at each time step k to optimally tradeoff between predictions of the model forecast and the information obtained from the measurements. The state will be biased towards the forecasts if the errors in the measurements are large, while the state will be corrected towards the measurements if the errors are small.

Because this estimator is designed for linear systems, it must be modified to work on the nonlinear models often studied in Modelica. The EKF represents perhaps the most straightforward modification of the basic Kalman filter equations for this purpose, which is to linearize the nonlinear system model at each time step, and correct the state based upon the predictions from the linearized model. Given the original system description in Equations 1-3, we can linearize the system at each time step k,

$$\dot{x} = \mathcal{M}_{(x,k)}x + \mathcal{M}_{(u,k)}u \tag{13}$$

$$y = \mathcal{H}_{(x,k)}x \tag{14}$$

where $\mathcal{M}_{(x,k)} = \partial\, \mathcal{M}/\partial x|_k$, $\mathcal{M}_{(u,k)} = \partial\, \mathcal{M}/\partial u|_k$, and $\mathcal{H}_{(x,k)} = \partial\, \mathcal{H}/\partial x|_k$. This model can then be discretized for a system with sample time T_s using the matrix exponential, e.g., $A = e^{\mathcal{M}_{(x,k)}T_s}$, which yields the linear system at this operating point represented by Equations 4 and 5. Once in this discrete-time representation, the previous Kalman filter equations (8-12) can be implemented.

In order to describe the implementation of the EKF as explicitly as possible, we present a simple nonlinear model of a two-phase permanent magnet machine from Simon (2006) to provide context for the derivation of these modifications, and to provide a concrete example for which we can develop the EKF and EnKF code listings of Figures 1 and 2. This model consists of four coupled ODEs that describe the electrical and mechanical dynamics of the machine, e.g.,

$$L\frac{di_a}{dt} = -Ri_a + \omega\lambda \sin\theta + v_a \tag{15}$$

$$L\frac{di_b}{dt} = -Ri_b - \omega\lambda \cos\theta + v_b \tag{16}$$

$$J\frac{d\omega}{dt} = \frac{3}{2}\lambda\left(-i_a \sin\theta + i_b \cos\theta\right) - B\omega \tag{17}$$

$$\frac{d\theta}{dt} = \omega, \tag{18}$$

where i represents electrical current, v represents voltage, R is the winding resistance, L is the winding inductance, λ is the flux linkage of the coil, J is the rotational inertia of the shaft, and B is the damping factor of the load.

In general, electrical variables can be observed easily and reliably, whereas the mechanical variables are more expensive to measure. We consequently assume that we have observations of the the input voltages v_a and v_b, as well as the currents i_a and i_b, but want to obtain estimates of the shaft speed ω. This model can thus be written down in the form of Equations 1-3, where the state $x = [i_a\ i_b\ \omega\ \theta]^T$ and the input $u = [v_a\ v_b]$. Equations 15-18 are straightforward to implement as a model in Modelica; we chose to avoid the specification of the phase voltages as explicit inputs, and instead defined them as time-varying real variables.

We create an estimator from a Modelica model on the basis of the above theory by first exporting the underlying Modelica model (representing $(\mathcal{M}, \mathcal{H})$) from a Modelica tool as an FMU, reimporting the same FMU back into Modelica as a co-simulation FMU, and then implementing the EKF by modifying the Modelica code that is autogenerated upon reimportation. The co-simulation format is needed because of the discrete-time formulation of the EKF. This autogenerated code contains a wide variety of helper functions (often encapsulated in their own package, such as `fmiFunctions` in Dymola) that are needed to interface with the FMI API and run the FMU in the Modelica tool, and which are the building blocks from which the estimators are built. These helper functions do not necessarily have the same API as that which is described in the FMI standard because they provide an interface between the specific tool and the functions defined by the standard. However, the functions used in the estimator are relatively straightforward, and would be expected to be found in most complete FMI implementations.

Two aspects of this reimported FMU are of particular note. First, the variables in the Modelica-instantiated FMU are referred to by 9-digit integer labels, rather than by their original names. The `modelDescription.xml` file included in the FMU lists the correspondence between these labels and their names, but use of particular variable names in the estimator requires explicitly re-establishing this correspondence in the estimator code (Brembeck et al., 2014). Second, though the derivative variables are separately enumerated in the XML file in the element `modelStructure`, the FMU does not maintain an easily parsed list of the integer labels corresponding to the state variables. The set of state variable integer labels can instead be obtained by importing the FMU in the model-exchange format for this express purpose, though this model-exchange FMU must be unloaded so that it can be reimported in the co-simulation format to create the estimator code. Once loaded, the initialization section of the model-exchange FMU contains a list of the state variables selected during compilation as well as their associated integer labels. Because the process of reading and correcting the state in the EKF requires the manipulation of these integer labels (which are often

sequential for the state variables), this is of major importance when implementing the estimator.

The essential code implementing the EKF is provided in Figure 1, while the complete code is provided in an attachment to this paper. To make this code more readable, certain simplifications were adopted with the hope that they do not obfuscate the overall intent. Some boilerplate code, such as variable definitions, was eliminated when the information about the variable could be inferred from its context. In addition, some sections of code that were autogenerated during the FMU import were also commented out. Finally, integer labels were shortened to improve readability.

This code excerpt begins in lines 1-13 with the definition of the fmiFunctions section (`fmiF`), which is autogenerated upon FMU import, and the user-specified definition of a number of important variables for the EKF, such as the number of states and outputs `Nx` and `Ny`, the process noise covariance matrix Q, and the measurement noise covariance R. We also create a new set of inputs and outputs `y` and `yhat` (lines 15-16), since we will be reading the measurements of the actual plant that will be assimilated by the estimator in variable `y`, and we want to study the performance variables `yhat`. Once these variables are set up, the FMU enters a `when` loop that will step through the stimulation and is initialized in an autogenerated block of code represented by line 21.

Lines 23-41 create of the linearized forward model operator $\mathcal{M}_{(x,k)}$ and linearized observation operator $\mathcal{H}_{(x,k)}$. The `fmiGetDirectionalDerivative` function takes the directional derivative of the integer labels corresponding to the derivatives of the state variables (in the array of the second argument) with respect to the dot product of the integer labels corresponding to the state variables (the third argument) and the column of the identity matrix $I^{N_x \times N_x}$ corresponding to the number of the column (the fourth argument). While this function could be used to take the mixed derivative with respect to multiple variables, the array `IdentityMatrixA` is used in this case to construct the Jacobian by computing the gradient of the state vector with respect to each individual state variable. Note that $\mathcal{M}_{(x,k)}$ is of size $N_x \times N_x$, whereas $\mathcal{H}_{(x,k)}$ is of size $N_y \times N_x$, and the entries of the output vector in lines 37-38 are the same as the state vector because the states i_a and i_b are assumed to be measured.

Once these linearized Jacobians have been created for the current time step, we apply the correction terms calculated at the end of the last time step and step the simu-

```
1  import MSL.Math.Matrices.exp;
2  import MSL.Math.Matrices.inv;
3
4  package fmiF
5  ...
6  end fmiF;
7  public
8  parameter Integer Nx=4, Ny=2;
9  parameter Real Hz[1,Nx]=cat(2,
10   {{1}}, {{1}}, zeros(1,2));
11 parameter Real Q[Nx,Nx]=transpose(Hz)*Hz;
12 parameter Real R[Ny,Ny]=identity(Ny);
13 // {Other variable declaration code}
14
15 MSL.Blocks.Interfaces.RealInput y[2];
16 MSL.Blocks.Interfaces.RealOutput yHat[2];
17
18 algorithm
19 when {initial(),
20       sample(startTime, stepSize)} then
21 // {Initialization/slave-mode code}
22
23 for i in 1:Nx loop
24   dz := fmiF.fmiGetDirectionalDerivative(
25     fmi,
26     {560, 561, 562, 561},
27     {432, 433, 434, 435},
28     IdentityMatrixA[i,:]);
29   dF[:,i] := dz;
30 end for;
31
32 F := exp(dF*stepSize);
33
34 for i in 1:Ny loop
35   dh := fmiF.fmiGetDirectionalDerivative(
36     fmi,
37     {432, 433},
38     {432, 433, 434, 435},
39     IdentityMatrixC[i,:]);
40   H[:,i] := dh;
41 end for;
42
43 PPred := F*PCorr*transpose(F) + Q;
44 K := PPred*transpose(H)*inv((H*PPred*
       transpose(H)) + R);
45
46 // Apply state correction after 5 steps
47 if time >= (startTime + 5*stepSize) then
48   fmiF.fmiSetReal(fmi,
49     {432, 433, 434, 435}, xCorr);
50 end if;
51
52 // {Step forward with fmiDoStep()}
53
54 xPred := fmiF.fmiGetReal(
55   fmi, {432, 433, 434, 435});
56 yPred := fmiF.fmiGetReal(
57   fmi, {432, 433});
58 xCorr := xPred + K*(y - yPred);
59 PCorr := (identity(4) - K*H)*PPred;
60
61 // {Variable allocation code}
62 equation
63 yHat[1] = 'x[1]';
64 yHat[2] = 'x[2]';
```

Figure 1. Extended Kalman filter (EKF) code.

lation forward. The Kalman gain matrix and the forecast for the covariance matrix are calculated in lines 43-44, and the state correction is applied to the state integer labels in lines 46-50 by using `fmiSetReal`. The `if` statement in line 47 is used to allow the values of the covariance matrix to accumulate before applying the corrections to the state variables. Once the state vector has been corrected, the FMU is stepped forward to compute the forecast step using `fmiDoStep()` auto-generated code. After this step, the corrections to the state vector $\hat{x}$ and the covariance matrix P for the analysis step are computed in lines 54-61 for application to the next cycle. Finally, we need to add two equations in line 63-64 to assign the outputs to be the state estimates that we want to examine.

2.2 Ensemble Kalman Filter

While the EKF is a popular estimation technique because it embodies a logical extension of the Kalman filter to nonlinear systems, it has been shown to have a few drawbacks. First, the entire covariance matrix $N_x \times N_x$ matrix P must be integrated at each time step; while this does not present a burden for small systems, the quadratic growth of the size of this matrix with the length of the state vector poses significant computational barriers for large-scale problems involving thousands of states. Second, the development of the EKF as an extension of the linear KF uses a linearized equation to describe the propagation of the error covariance matrix. This may not be a valid assumption, and can result in unbounded linear instabilities of the error evolution (Evensen, 2009b).

An alternative approach created to address the size limitations of the EKF is called the ensemble Kalman filter, or EnKF (Evensen, 2009b). Rather than directly propagate all of the covariances forward at each time step, the EnKF uses a sequential Monte Carlo approach to propagate forward an ensemble of statistically sampled state vectors, or particles, through the system dynamics and directly estimate the covariance matrix from this distribution. This avoids the computation of the Jacobians for the forward model and the observation operator. This algorithm is summarized with a range of variants by Vetra-Carvalho et al. (2018), and will be briefly presented here.

As indicated above, the essential distinction between the EKF and the EnKF can be seen by examining the formulation of the covariance matrices. Whereas the EKF formulates the covariance matrix P as

$$P_k = \mathbf{E}[(\hat{x}_k - x_k)(\hat{x}_k - x_k)^T], \tag{19}$$

the EnKF uses the fact that it propagates forward an ensemble of particles to define the ensemble covariance matrices around the ensemble mean rather than the true mean, e.g.,

$$P_k = \mathbf{E}\left[(\hat{x}_k - \mathbf{E}[\hat{x}_k])\,(\hat{x}_k - \mathbf{E}[\hat{x}_k])^T\right]. \tag{20}$$

This allows us to approximate the error covariance matrices by using simple multiplications, rather than integrating the full set of differential equations for the covariance matrix forward at every time step. This method has been successfully used in numerical weather prediction models with more than 10^6 states, which would not be generally feasible with an EKF.

Because the ensemble of particles evolves forward with time, it is also necessary that we formulate the problem in terms of the ensemble mean and the ensemble perturbations. Taking X^a as N_e particles sampled from a state space of N_x so that X^a has dimension $N_x \times N_e$, we construct the analysis mean and perturbation ensemble by

$$X^a = \overline{X}^a + X'^a. \tag{21}$$

As a result, the covariance matrix P^a can be written as

$$P^a = \frac{X'^a (X'^a)^T}{N_e - 1}. \tag{22}$$

We summarize the construction of the filter as follows, emphasizing the similarity to the linear Kalman filter. The ensemble forecast is constructed by propagating the nonlinear forward operator and observation operator by a step, e.g. integrating forward the model equations

$$X_k^f = \mathcal{M}(X_{k-1}^a, u_{k-1}, t_{k-1}; \mu) \tag{23}$$

$$HX_k^f = \mathcal{H}(X_k^a, t_k; \mu) \tag{24}$$

where HX_k^f indicates the integation of the nonlinear observation operator forward on the ensemble by one time step. We then calculate the output error covariance HPH_k and the innovation term C_k by computing

$$HX_k'^f = HX_k^f - \overline{HX}_k^f \tag{25}$$

$$HPH_k = \frac{HX_k'^f (HX_k'^f)^T}{N_e - 1} \tag{26}$$

$$C_k = (HPH + R)^{-1}\left(Y_k - HX_k^f\right) \tag{27}$$

We construct the ensemble perturbation matrix $X_k'^f = X_k^f - \overline{X}_k^f$ and compute the correction ensemble

$$X_k^a = X_k^f + \frac{1}{N_e - 1} X_k'^f (HX_k'^f)^T C_k \tag{28}$$

which completes the calculation of the analysis step for the EnKF.

As was the case for the EKF, much of the implementation of the EnKF using the FMU is relatively straightforward. Many of the specific modifications needed to implement the EnKF by using the autogenerated Modelica code pertain to the generation and propagation of the particles used to characterize the estimator's statistical properties, as well as the computation of the ensemble mean. While random number generation code from the Modelica Standard Library (MSL) (included in `Modelica.Math.Random` and `Modelica.Math.Distributions`) was used in the

creation of the perturbations, separate implementations of the noise generation functions were required because the models in `Modelica.Blocks.Noise` could not be included in the algorithm sections of the FMU.

The EnKF code described in Figure 2, and which is also available in its entirety in the attachment, bears many similarities to the EKF code discussed previously. Many of the variables are first defined in lines 1-21, with some of the variable definitions commented out for the sake of brevity. The random number generation code imported from the MSL is abbreviated as `MSL*` for similar reasons, though these functions can be quickly found through a search. Two variables (`initState` and `nextState`) were also required to maintain and evolve the state of the random number generator. The inputs and outputs are defined in lines 20-21, and the model is initialized using the autogenerated code in lines 23-26.

The initial ensemble of particles is generated in lines 28-35 during the initialization phase of simulation by using the `MatrixPerturbation` function. This function takes a set of initial guesses for the state variables and creates N_e perturbed instances of this $N_x \times 1$ state vector by adding zero-mean noise with a standard deviation based upon the order of magnitude of the initial guesses. This scaling of the particle perturbations is important, as perturbations that are too small will not provide sufficient diversity to estimate the covariance, while perturbations that are too large could result in incorrect model behavior or otherwise reduce the information provided by these initial guesses.

```
1  import distribution=MSL*.quantile;
2  import generator=MSL*.Xorshift128plus;
3  import MSL*.impureRandomInteger;
4  package fmiF
5  ...
6  end fmiF;
7  public
8  parameter Integer Nx=4, Ne=5, Ny=2;
9  parameter Real R[Ny,Ny]=[0.01, 0; 0, 0.01];
10 parameter Real[Nx] x_init={0, 0, 0, 0};
11 parameter Real[Nx,Ne] X_init=[x_init,
       x_init, x_init, x_init, x_init];
12 parameter Real mu=0, sigma=0.1;
13
14 MSL.Blocks.Noise.GlobalSeed globalSeed;
15 parameter Integer actualGlobalSeed=
       globalSeed.seed;
16 final parameter Integer localSeed=
       impureRandomInteger(
       globalSeed.id_impure)
17 Integer initState[generator.nState];
18 Integer nextState[generator.nState];
19 // {Other variable declaration code}
20 MSL.Blocks.Interfaces.RealInput y[Ny];
21 MSL.Blocks.Interfaces.RealOutput yHat[Ny];
22
23 algorithm
24 when {initial(),
25         sample(startTime, stepSize)} then
26 // {Initialization/slave-mode code}
27
28 if initial() then
29   // Create particle distribution
30   initState := generator.initialState(
31     localSeed, actualGlobalSeed);
32   (X, nextState) := MatrixPerturbation(
33     X_init, initState);
34   initState := nextState;
35 end if;
36
37 if time >= startTime + (5*stepSize) then
38   X := XCorr;
39 end if;
40
41
42 for i in 1:Ne loop
43   fmiF.fmiSaveFMUState(fmi);
44   fmiF.fmiSetReal(fmi,
45     {432, 433, 434, 435}, X[:,i]);
46   // {Step ensemble member with fmiDoStep}
47   X[:,i] := fmiF.fmiGetReal(fmi,
48     {432, 433, 434, 435});
49   HX[:,i] := fmiF.fmiGetReal(fmi,
50     {432, 433});
51   if i<Ne then
52     fmiF.fmiRestoreFMUState(fmi);
53   else
54     continue;
55   end if;
56 end for;
57
58 // EnKF computations
59 X_bar := MatrixMean(X,1);
60 HX_bar := MatrixMean(HX,1);
61
62 for i in 1:Ne loop
63   X_prime[:,i] := X[:,i]-X_bar;
64   HX_prime[:,i] := HX[:,i]-HX_bar;
65 end for;
66
67 HPH:=1/(Ne-1)*HX_prime*transpose(HX_prime);
68 A := HPH + R;
69 (Y, nextState) := MatrixPerturbation(
70   [y, y, y, y, y], initState, sigma=0.01);
71 initState := nextState;
72
73 D := Y - HX;
74 C := MSL.Math.Matrices.solve2(A,D);
75 E := transpose(HX_prime)*C;
76 XCorr := X + (Ne-1)*X_prime*E;
77 yHat := HX_bar;
78
79 // {Variable allocation code}
80 equation
81 yHat[1] = 'x[1]';
82 yHat[2] = 'x[2]';
```

Figure 2. Stochastic ensemble Kalman filter (EnKF) code.

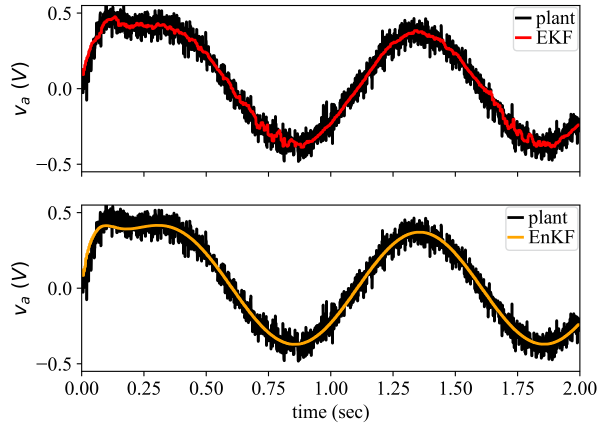

Figure 3. Estimated and measured phase A currents for the EKF and EnKF experiments.

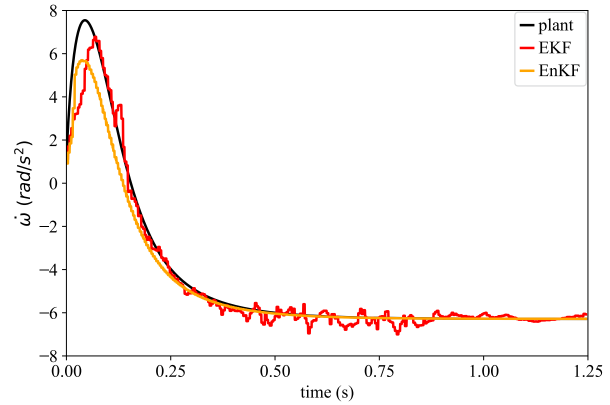

Figure 4. Shaft speed.

After creating this ensemble, the corrected state vector is applied to the system in lines 37-39; this is performed after 5 time steps as was the case for the EKF. Lines 42-56 then use this ensemble and the corrected state vector during model integration to evolve each particle forward according to the dynamics of the nonlinear model. This is accomplished by

1. Saving the state of the FMU using `fmiSaveFMUState`,
2. Setting the integer labels of the state variables to the values of the current particle,
3. Stepping the system dynamics forward with this particle using `fmiDoStep()`,
4. Saving the results of the integration step to the variables `X` and `HX`, and
5. Resetting the state of the FMU using `fmiRestoreFMUState` so the next particle can be integrated forward.

Note that the evolution of the model does not depend on the evaluation of the Jacobians, which can potentially reduce the amount of computation required. However, the requirement that each particle be individually stepped forward in time can also present significant computational requirements, depending on the implementation of the integrator and the relative scales of the state variables.

The analysis phase of the EnKF is completed in lines 59-77 of the EnKF example. This implementation follows Vetra-Carvalho et al. (2018) closely, and culminates in the computation of the corrected state vector in line 76. The outputs of interest are then assigned in lines 81-82 to complete the construction of this estimator.

3 Simple Motor System

The construction of these estimators was based upon a Modelica model of the simple motor described in Equations 15-18, with model parameters set to the values provided in Table 1. After generating a co-simulation FMU for this model, we saved the model into a separate folder that was expressly set up for the storage of FMUs, as FMUs cannot be stored within the package directory structure. This imported Modelica model was then duplicated and modified to create the EKF and the EnKF estimators, which remained distinct from the original imported FMU. Note that the path to the FMU DLL is stored as a text string in the autogenerated Modelica code, and must be changed manually if the FMU is moved after it is reimported.

We evaluated these estimators by creating a test model that included both the original Modelica motor model and the new estimator. Zero-mean noise with a standard deviation of 0.05 A was added to the motor current observations before feeding these signals into the estimator to evaluate the robustness of the estimates. In addition, the states of the initial motor model were each initialized to zero, but the states of the estimator model were each initialized to 1 to test the estimator's robustness to initial state errors. The EnKF was configured to use 5 particles in its ensemble; we found that the performance and convergence of this filter was dependent upon the number of particles used.

One minor observation that merits a highlight is that the simulation time is set in the FMU when it is reimported, and is not set by the annotations usually used to capture information in Modelica models. As such, if the user wants to change the start or end time of a simulation, or the number of time steps used, this must be done both in the simulation window, as usual, and by setting the appropriate parameters of the FMU.

Figure 3 demonstrates the ability of both estimators to construct good estimates of state i_a from noisy observa-

μ	Value	μ	Value
v_a	$\sin(2\pi t)$	λ	$0.1\ V \cdot s$
v_b	$\cos(2\pi t)$	J	1.8e-4 $\text{kg} \cdot \text{m}^2$
R	1.9 Ω	B	1e-3 $\text{kg} \cdot \text{m/s}$
L	3 mH		

Table 1. Motor parameters.

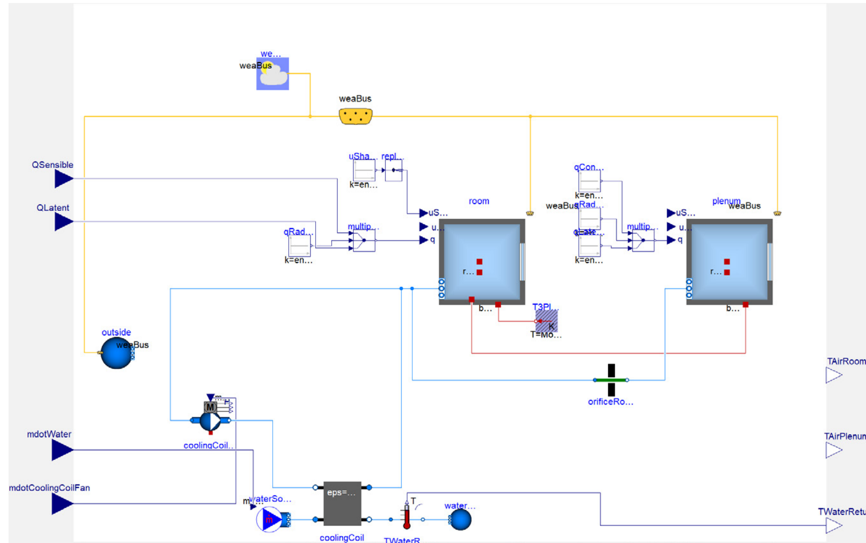

Figure 5. Two zone (room/plenum) model including a fan coil unit.

tions of the original motor model. The upper plot in this figure shows the performance of the EKF, while the lower plot shows the performance of the EnKF; the state estimate for both systems can be seen to be close to the mean of the input signal.

Figure 4 illustrates the estimates of the motor speed state ω as well as the estimates produced by the EKF and EnKF. It is clear from a comparison between the original plant signal and the estimates that both filters can produce good estimates of this state without its measurement. It is interesting to note that the estimate for this state from the EKF is somewhat noisier than those estimates from the EnKF, but that the mean of both of these estimates is quite close to the state of the original plant.

4 Building System

Though we can gain significant insights into the process of designing and implementing estimators by using the previous simple motor model, our primary interest in this technology lies in the ability to leverage large-scale Modelica system models for use in estimation problems. As a case study of this application, we constructed a two-zone model of one floor of a commercial office building, including both the occupied space and a plenum above, by using the models provided by the Modelica Buildings library (Wetter et al., 2014) with the objective of estimating the sensible heat load in the occupied space. This model is very similar to that which was used by (Bortoff and Laughman, 2019).

A schematic diagram of this building model is illustrated in Figure 5, which shows the two zones as well as a water-source dry-coil fan coil model that is used to manage the room air temperature. The use of the dry coil model is somewhat atypical, and will be elaborated later in this section. This fan coil is connected to a fixed temperature water source (10 °C) and sink (16 °C), and the room temperature is regulated by a PI controller that adjusts the mass flow rate of water through the coil to maintain the set point. The area of the floor is 415 m^2, while the room height is 2.6 m and the plenum height is 1.3 m; there is also a total of 83.6 m^2 of window area on the external walls. The heat load varies between 1.66 kW (4 W/m^2) between the hours of 7pm and 8am, and 4.15 kW (10 W/m^2) between the hours of 8am and 6pm, with smooth ramps during the transition hours. The Tokyo TMY3 weather file is used to provide the ambient conditions, and this simulation is run for 5 days from June 12-17 to study its behavior over a practical duration of time.

The connections between the building model and the estimator are illustrated in Figure 6, which illustrate the use of the original Modelica plant model and the FMI-based estimator. This estimator is based on four measurements

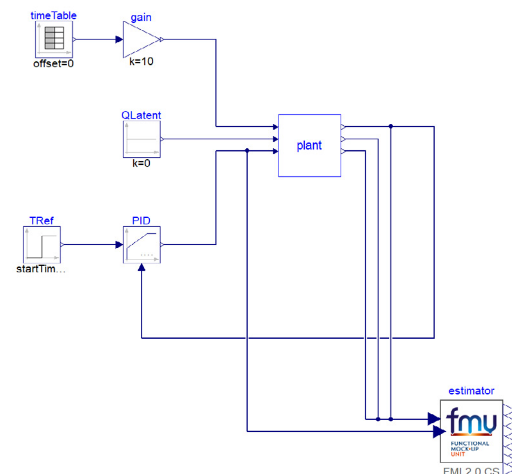

Figure 6. Estimator.

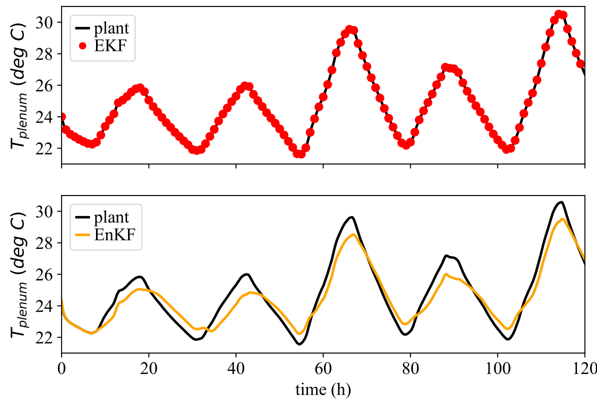

Figure 7. Estimated and measured plenum air temperatures.

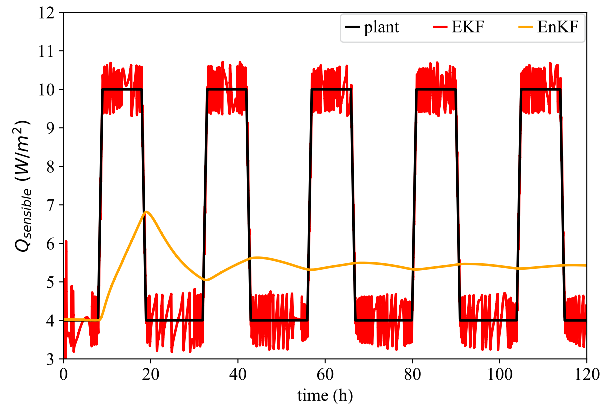

Figure 8. Imposed and estimated heat loads.

from the plant: a measurement of the water flow rate provided by the control signal for the fan coil unit, the room air temperature, the plenum air temperature, and the return water temperature. We then designed an EKF and an EnKF to estimate the heat load on the room. This objective necessitated a small modification of the model used for the estimator; while the sensible load is customarily specified as an input to the model, we added an integrator driven by a constant value to the sensible load input for the estimator model. By incorporating this integrator, we force the compiler to include the heat load as a state which can then be estimated. The scales of the states also required specific attention in the construction of the EnKF. In general, the states in these thermodynamic models are poorly scaled; for example, the internal energy of the water is on the order of 10^7 J/kg, while the humidity ratio of the room is on the order of 10^{-3} kg w/kg da. Perturbations of the state vector therefore must be carefully designed so they do not dominate its mean, which tends to be constructed carefully in consideration of physical intuition.

We tested the EKF by adding 0.01 °C of quantization noise to the measured outputs of the plant; while this amount of quantization is small, it was chosen to ensure that the observed variations in room and return water temperatures (which were small due to the large coil capacity) would not be hidden by the quantization. In using these quantized inputs, we examined the ability of the EKF to properly estimate both the states characterized by the inputs as well as the heat load. The upper plot of Figure 7 illustrates the measured and estimated plenum temperatures for the EKF, and it is clear that the estimator can accurately capture the dynamics of the original plant. Moreover, Figure 8 demonstrates the rapid convergence of the estimates to the imposed heat load, suggesting that the EKF is effective for solving this estimation task.

In comparison, the estimates of both the plenum air temperature and the sensible heat load from the EnKF are relatively poor. While the EnKF captures the general shape of the plenum air temperature dynamics, there are significant errors; moreover, the errors in the heat load estimates are substantial, and completely miss the time-varying behavior of the heat load towards the end of the simulation.

The poor performance of the EnKF is fundamentally related to interactions between the ensemble perturbations and the state and output constraints of the model. Many models of thermodynamic systems have limits of validity on their variables; for example, the dry air model used in the Buildings library is only accurate down to temperatures of 200 K, while the humidity ratio in a space cannot physically go below zero. However, the ensemble perturbations in the EnKF are not designed to satisfy these constraints in a statistically valid manner. We found many circumstances in which small perturbations in the ensemble led to constraint violations either of the state variables directly (e.g., humidity ratio), or of algebraically related output variables (e.g., temperatures), causing the simulations to crash. The need to place stringent limits on the dispersion of the particles thus prevented the covariance matrix from accurately characterizing the system dynamics and producing an accurate estimate of the heat load.

This behavior was apparent in all of the experiments with the EnKF. We found that there were effectively no useful perturbations that could be applied to a wet-coil fan coil model that would not violate the constraint that the humidity ratio needs to be positive, and that even the room with the dry coil model often had problems in which the the mixed air humidity ratio would tend toward zero. While the EnKF does demonstrate promise for problems in which a suitably large ensemble can be used, it appears to have significant limits when used in an FMI-based context with constraints on state and output variables.

While the design of the FMI standard is suited to describe sets of differential equations for the purpose of simulation as well as estimation using Kalman filters and EKFs, the fact that it does not provide a direct association between variables and their constraints imposes a crucial limitation in the practical implementation of constrained estimators (Simon, 2010) or particle filters for large-scale problems (Van Leeuwen et al., 2019). For example, an

estimator designed for the system

$$\frac{dx_1}{dt} = f_1(x_1, x_2, x_3; \mu) \quad (29)$$

$$\frac{dx_2}{dt} = f_2(x_1, x_2, x_3; \mu) \quad (30)$$

$$x_3 = f_3(x_1, x_2; \mu) \quad (31)$$

$$a_1 \leq x_1 \leq b_1 \quad (32)$$

$$a_3 \leq x_3 \leq b_3 \quad (33)$$

must take the state and inequality constraints imposed by Equations 32 and 33 into account when generating the perturbation ensembles for the state vector $[x_1 \ x_2]$. While these constraints could be incorporated manually into an estimator for a small system when modifying the Modelica code generated upon import of the co-simulation FMU, the application of these estimation methods would be much more practical if the FMU included the infrastructure needed to systematically associate the variables with their constraints.

5 Conclusions & Discussion

While the basic infrastructure used to construct FMI-based system models might not always have the most intuitive interface, their capabilities make them excellent candidates for use in a variety of estimation problems. We found that the implementation of the extended Kalman filter is relatively straightforward once the FMI API is fully understood, and that these filters demonstrated good performance on both a small test problem and a larger estimation problem that utilized the capabilities of Modelica for building models of complex physical systems. We also found that the FMI interface also enabled the construction of other types of estimators, such as the ensemble Kalman filter, suggesting that there is potential in the further investigation of other FMI-based interfaces for estimation applications.

However, this study also revealed some important limitations of FMI for the application of constrained estimation approaches, such as ensemble Kalman filters or other particle filtering methods. The lack of a direct association between a state variable and its constraints posed significant difficulties in the implementation of ensemble-based methods. Future work on systematically accommodating such constraints in FMI could have a significant impact on FMI's use on the range of estimation problems, especially for the large-scale applications to which Modelica models are so well-suited.

References

S.A. Bortoff and C.R. Laughman. An extended Luenberger observer for HVAC application using FMI. In *Proceedings of the 13th International Modelica Conference*, pages 149–155, 2019. doi:10.3384/ecp19157149.

J. Brembeck. A physical model-based observer framework for nonlinear constrained state estimation applied to battery state estimation. *Sensors*, 19, 2019. doi:10.3390/s19204402.

J. Brembeck, M. Otter, and D. Zimmer. Nonlinear observers based on the Functional Mockup Interface with applications to electric vehicles. In *Proceedings of the 8th International Modelica Conference*, pages 474–483, 2011. doi:10.3384/ecp11063474.

J. Brembeck, A. Pfeiffer, M. Fleps-Dezasse, M. Otter, K. Wernersson, and H. Elmqvist. Nonlinear state estimation with an extended FMI 2.0 co-simulation interface. In *Proceedings of the 10th International Modelica Conference*, pages 53–62, 2014. doi:10.3384/ECP1409653.

Dassault Systemes. Dymola 2020, 2019.

G. Evensen. *Data Assimilation: The Ensemble Kalman Filter*. Springer, 2 edition, 2009a.

G. Evensen. The ensemble Kalman filter for combined state and parameter estimation. *IEEE Control Systems Magazine*, pages 83–104, 2009b. doi:10.1109/MCS.2009.932223.

Modelica Association. Modelica specification, Version 3.4, 2017. URL `www.modelica.org`.

Modelica Association. Functional Mockup Interface for Model Exchange and Co-Simulation, Version 2.0.1, 2019. URL `www.fmi-standard.org`.

D. Simon. *Optimal State Estimation: Kalman, H-∞, and Nonlinear Approaches*. John Wiley & Sons, 2006.

D. Simon. Kalman filtering with state constraints: A survey of linear and nonlinear algorithms. *IET Control Theory and Applications*, 4(8):1303–1318, 2010. doi:10.1049/iet-cta.2009.0032.

P. J. Van Leeuwen, H. R. Künsch, L. Nerger, R. Potthast, and S. Reich. Particle filters for high-dimensional geoscience applications: a review. *Quarterly Journal of the Royal Meteorological Society*, 145(723):2335–2365, 2019. doi:10.1002/qj.3551.

S. Vetra-Carvalho, P.J. Van Leeuwen, L. Nerger, A. Barth, M.U. Altaf, P. Brasseur, P. Kirchgessner, and J.-M. Beckers. State-of-the-art stochastic data assimilation methods for high-dimensional non-Gaussian problems. *Tellus A*, 70:1–43, 2018. doi:10.1080/16000870.2018.1445364.

L. Vytvytskyi and B. Lie. Combining measurements with models for superior information in hydropower plants. *Flow Measurement and Instrumentation*, 69, 2019. doi:10.1016/j.flowmeasinst.2019.101582.

M. Wetter, W. Zuo, T. Nouidui, and X. Pang. Modelica buildings library. *Journal of Building Performance Simulation*, 7(4): 253–270, 2014. doi:10.1080/19401493.2013.765506.