



D3.1.3 – Estimation blocks for continuous-time FMUs

WP 3.1 Continuous Systems

WP 3 State estimation and system monitoring

MODRIO (11004)

Version 2.0

Date Nov. 30, 2015

Authors

Andreas Pfeiffer, Jonathan Brembeck, Martin Otter
German Aerospace Center (DLR)
Institute of System Dynamics and Control (DLR-SR)
Oberpfaffenhofen
Germany

Executive Summary

This document describes the open source Modelica library "KalmanFilterOpenSource". The library provides the functionality to use an Extended Kalman Filter model of a nonlinear Modelica model in Dymola. Basically, any Modelica model with continuous-time states can be used. The process is as follows:

1. Starting point is a Modelica model with top level inputs and outputs, as well as continuous-times states.
2. This plant model needs to be converted to an FMU for Co-Simulation 2.0. This FMU has inlined integrators embedded and can be viewed as a discrete-time model of the plant model. It requires the non-standard (Dymola-specific) feature that at a communication point the internal states of the model can be changed from the outside.
3. The generated FMU is imported in to a Modelica package of the user.
4. A discrete-time state estimation Modelica model is generated in the package that utilizes the FMU and uses the extended Kalman Filter algorithm for the state estimation. Basically, this filter integrates from one sample point to the next. At every sample point the nonlinear discrete-time system is (numerically) linearized and a Kalman Filter for the linearized system is designed and used for this sample period.
5. An instance of the state estimation model can be used in a Modelica controller to estimate the full state vector of the (nonlinear) plant from measurements.

This approach is demonstrated by means of a Double Pendulum example that is mounted on a trolley. The FMU of the Double Pendulum is included (for Windows).

Additionally, this report describes shortly an improved version of the above process that will become available in a commercial Modelica library from DLR-SR: Steps 2-4 above are automatically performed by appropriate Dymola/Modelica scripts. Furthermore, more advanced state estimation algorithms are supported, such as the Unscented Kalman Filter. For more information, contact sr-modelica@dlr.de.

Contents

Executive Summary	2
Contents	3
1 Open Source Kalman Filter Library	4
1.1. Kalman Filter in General	4
1.2. Basic Structure of the Library	5
1.3. Structure of a Tailored Kalman Filter Model	5
1.4. Usage of Kalman Filter Models	7
1.5. Kalman Filter Parameters	8
1.6. Simulation Results	11
2. Advanced Version of Kalman Filter Library	12
2.1. Plant Model for a Kalman Filter in Modelica	12
2.2. Automatic Generation of Tailored Kalman Filter Models	13
3. References	14

1 Open Source Kalman Filter Library

The open source Kalman Filter library is a Modelica package providing tools to apply Kalman filter models based on FMUs (Functional Mock-up Units).

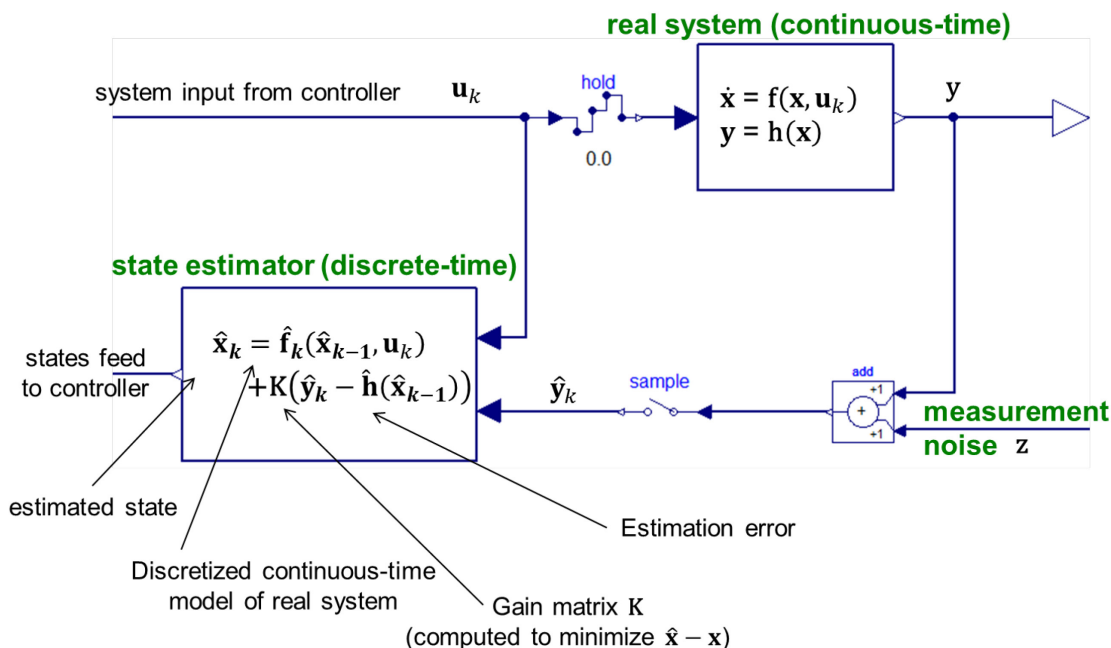
In the first sub-sections of this document some general information about Kalman filters and the basic structure of the Kalman Filter library are summarized. In the following the document aims to address

- the structure of tailored Kalman filter models in Modelica,
- the usage of Kalman filter models within Modelica models,
- the parameters of a Kalman filter model and
- some simulation results

by means of a double pendulum model.

1.1. Kalman Filter in General

The methodical description of Kalman filters can be found in [2]. Here, only a very short summary is given to define the nomenclature. In the following figure a signal flow diagram is sketched that shows the configuration of a state estimator in a discrete time system. From a controller a system input u_k is commanded to the real system and also fed to the inputs of the prediction model of the state estimator. Note that it is always assumed that the inputs have no direct feed through to the outputs. This is a valid assumption for all physical models and controlled plants. Due to this input and the system dynamics we can measure some outputs y of the real system that may be disturbed through a measurement noise z ; in an additive case this yields the sampled measurement vector \hat{y}_k . From the discrete prediction model \hat{f}_k and the output function \hat{h}_k we get an estimate of the output. To estimate the true states of the real system we try to reduce the error between measurement and prediction. This is copied via calculation of the gain K in the Kalman filter algorithm. It is determined in a way that it minimizes the error between $\hat{x} - x$ under statistical assumptions believing in the underlying prediction model and the measured values.



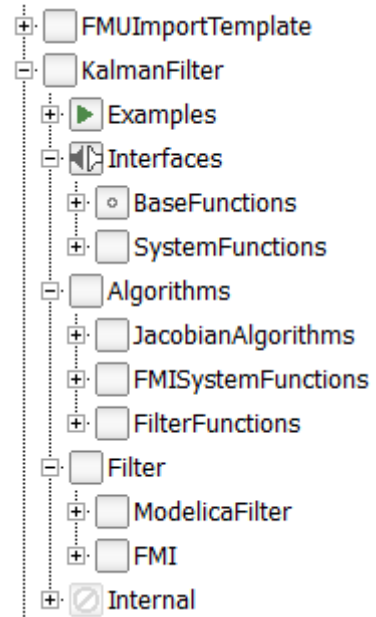
1.2. Basic Structure of the Library

The package `KalmanFilterOpenSource.mo` consists of the libraries

- `FMUImportTemplate`,
- `KalmanFilter`.

The main library is `KalmanFilter` whereas `FMUImportTemplate` is an interface library for imported FMUs, see [1] for details.

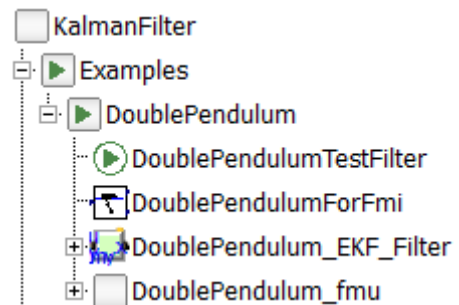
The `KalmanFilter` library mainly consists of the sub packages `Interfaces`, `Algorithms` and `Filter`. In `Interfaces` the base classes and function prototypes for the Kalman filter system functions are defined – see [1] for details. Sub package `Algorithms` contains the numerical algorithms of the library. Especially, algorithms to compute the Jacobians of system functions and the algorithmic parts of the Kalman filter techniques [2] can be found here. In `Algorithms.FMISystemFunctions` the Kalman filter relevant generic system functions for an imported FMU are collected – see also [1]. In the sub package `Filter`, `ModelicaFilter` a base class for an Extended Kalman Filter (EKF) model is contained. This base class is also extended in the FMI based EKF base model in `Filter.FMI`.



1.3. Structure of a Tailored Kalman Filter Model

There is a whole process defined in [1] how to interface FMUs for estimation purposes. To apply filter models of the open source Kalman Filter library it is assumed that an FMU for the plant model is provided according to the `FMUImportTemplate` (also part of the open source Kalman Filter package). For the double pendulum example the plant model `DoublePendulum.DoublePendulumForFmi` has been exported to an FMU and has been re-imported to the package `DoublePendulum.DoublePendulum_fmu`.

Further, a tailored filter model has to be programmed to utilize the imported FMU package and the generic Kalman filter algorithms, see also [1]. An example how such a filter model should look like is given in `DoublePendulum.DoublePendulum_EKF_Filter` (see the icon of the model below).



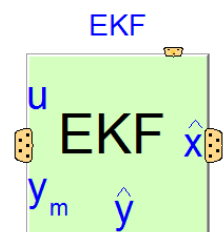
Firstly, parameter matrices `xData` and `yData` have to be defined as follows for the double pendulum:

```
model DoublePendulum_EKF_Filter
  parameter Real xData[nx,4] = zeros(nx,4);
  parameter Real yData[ny,1] = zeros(ny,1);
```

Some important annotations of the parameter definitions (which are explained in [1]) can be found in the source code of the filter model for the double pendulum.

The inheritance of the base EKF algorithm and the instantiation of the FMU initialization model are organized as follows:

```
extends KalmanFilter.Filter.FMI.PartialFilter.EKF_FMIBase(
  redeclare package FMUPackage = DoublePendulum_fmu,
  redeclare DoublePendulum_fmu.fmiModel fmi=ModelParameters.fmi,
  fmi_initializationExited = ModelParameters.fmi_initializationExited);
DoublePendulum_fmu.InitializationModel ModelParameters;
```



The filter model must have a bus connector `inBus` on the left side and a bus connector `outBus` on the right side of the model icon. The corresponding bus classes are `InBus` and `OutBus` within the filter model. The class `InBus` contains all the plant model input and output variables. The output variables in the `inBus` correspond to the measured model outputs. For the double pendulum example, the class `InBus` looks like the following:

```
encapsulated expandable connector InBus
  import Modelica;
  extends Modelica.Icons.SignalBus;
  // Model Inputs
  Real u;
  // Measured Model Outputs
  Real s;
  Real phi;
end InBus;
```

In the class `OutBus`, there are variables for all continuous time states of the plant model and for all output variables of the plant model. All these variables correspond to estimated variables of the Kalman filter. For the double pendulum example we have the following `OutBus`:

```
encapsulated expandable connector OutBus
  import Modelica;
  extends Modelica.Icons.SignalBus;
  // Estimated Model States
  Real prismatic_s;
  Real prismatic_v;
  Real rev_phi;
  Real rev_w;
  Real revolute2_phi;
  Real revolute2_w;
  // Estimated Model Outputs
  Real s;
  Real phi;
end OutBus;
```

Then, the buses have to be instantiated:

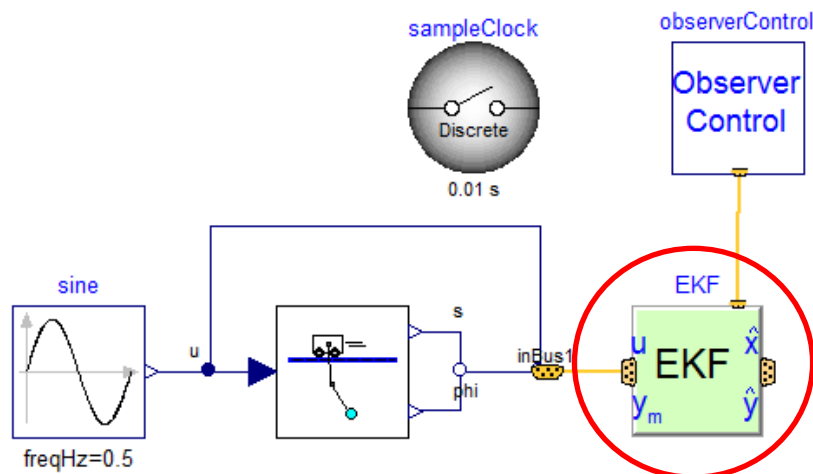
```
input InBus inBus;
output OutBus outBus;
```

In the equation part of the filter model the connections between the bus variables (with plant model specific names) and the anonymous vectors of the filter algorithms are made:

```
equation
  outBus.s = y_est[1];
  outBus.phi = y_est[2];
  outBus.prismatic_s = x_est[1];
  outBus.prismatic_v = x_est[2];
  outBus.rev_phi = x_est[3];
  outBus.rev_w = x_est[4];
  outBus.revolute2_phi = x_est[5];
  outBus.revolute2_w = x_est[6];
  inBus.u = u[1];
  inBus.s = y_m[1];
  inBus.phi = y_m[2];
end DoublePendulum_EKF_Filter;
```

1.4. Usage of Kalman Filter Models

A tailored filter model as described in the previous Section 1.3 can directly be used in another Modelica model (we call it *test model*) by drag and drop in Dymola, see for example the EKF filter of the double pendulum in the following figure (model `KalmanFilter.Examples.DoublePendulum.DoublePendulumTestFilter`):



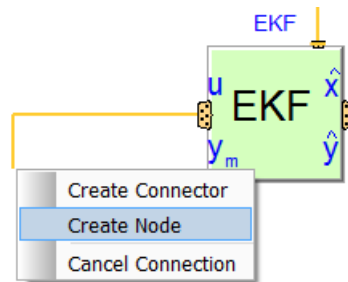
To build a new test model, the following steps are necessary:

- Instantiate the model `KalmanFilter.Internal.Utilities.observerControl` and connect the bus of this model with the bus `ObserverControl` of the filter model on the upper border of the model icon. The parameters of the model `observerControl` should not be changed in this version of the Kalman Filter library.
- Instantiate the model `Modelica_LinearSystems2.Controller.SampleClock` and select the following in the parameter menu of the model `sampleClock`:

Model		
Path	Modelica_LinearSystems2.Controller.SampleClock	
Comment	Global options for blocks of Controller library (in particular sample clock)	
Parameters		
blockType	<input type="radio"/> Continuous <input checked="" type="radio"/> Discrete	Type of Sampled blocks
methodType	Modelica_LinearSystems2.Types.Method.Trapezoidal	Discretization method for discrete blocks
sampleTime	0.01 s	Base sample time for discrete blocks
initType	Modelica_LinearSystems2.Controller.Types.Init.SteadyState	Type of initialization of Sampled blocks

The **sample time** specified here is known by the Kalman filter model. It is the main sample time of the overall discrete system in which the Kalman filter is a part of. The real sample time of the Kalman filter model can be influenced by another parameter (`sampleFactor`) in the filter model, see Section 1.5 for details.

- Make a connection line from the bus connector `inBus` (on the left side of the filter model) and press the right mouse button:



- Select "Create Node" and press OK in the appearing menu. Then a bus node with the name `inBus1` will be created.


Before the test model can be simulated the remaining part to do is to connect all variables of the created node `inBus1` to some sort of sources. If measurements are available, then the measurements should be fed into the Kalman filter model, e.g. by using `Modelica.Blocks.Sources.CombiTimeTable` and by connecting it to the corresponding variables in the node `inBus1`. For the double pendulum example the original plant model is used to "simulate" measurements. The input signal for the plant model and the Kalman filter model are the same.

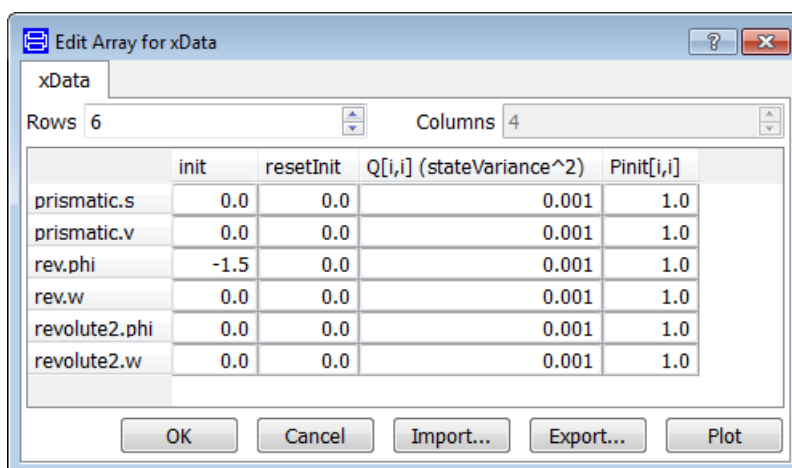
At this stage the test model can be run in Dymola as any other Modelica model. To get meaningful results, the parameters of the Kalman filter have normally to be edited, see the following section.

1.5. Kalman Filter Parameters

The parameters of each Kalman filter model can be edited by a double click with the mouse on the model instance icon in the diagram layer of Dymola. For the Kalman filter model EKF the parameter menu looks like the following:

Model		
Path	DoublePendulumFilters.EKF	
Comment	Extended Kalman filter	
Parameters		
xData	.001, 1.0; 0.0, 0.0, 0.001, 1.0; 0.0, 0.0, 0.001, 1.0	Filter parameters w.r.t. model states
yData	[0.2; 0.01]	Filter parameters w.r.t. model outputs
sampleFactor	1	Sample factor ($T_s = \text{sampleFactor} * \text{sampleClock.sampleTime}$)
ModelParameters		Parameters and start values of the estimation model

The parameter **xData** is a matrix for the estimated filter states that can be edited by clicking on the corresponding button :




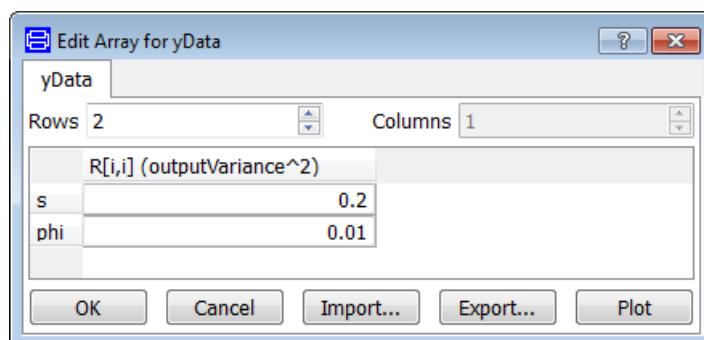
Dialog box titled "Edit Array for xData". It shows a table with 6 rows and 4 columns. The columns are labeled: "init", "resetInit", "Q[i,i] (stateVariance^2)", and "Pinit[i,i]". The rows are labeled: "prismatic.s", "prismatic.v", "rev.phi", "rev.w", "revolute2.phi", and "revolute2.w".

	init	resetInit	Q[i,i] (stateVariance^2)	Pinit[i,i]
prismatic.s	0.0	0.0	0.001	1.0
prismatic.v	0.0	0.0	0.001	1.0
rev.phi	-1.5	0.0	0.001	1.0
rev.w	0.0	0.0	0.001	1.0
revolute2.phi	0.0	0.0	0.001	1.0
revolute2.w	0.0	0.0	0.001	1.0

Buttons at the bottom: OK, Cancel, Import..., Export..., Plot.

Each row of the matrix corresponds to one state of the filter (names on the left). In the column "init" the initial values for the states in the filter can be specified. The column "resetInit" can be ignored for this version of the Kalman Filter library. In the column "Q[i,i]" the diagonal elements of the covariance matrix of the model states can be edited. Small values mean a high model accuracy of the plant model. In the column "Pinit[i,i]" the initial values for the diagonal elements of the estimated covariance matrix may be edited.

The parameter **yData** is a matrix for the estimated filter outputs that can be edited by clicking on the corresponding button :



Dialog box titled "Edit Array for yData". It shows a table with 2 rows and 1 column. The column is labeled: "R[i,i] (outputVariance^2)". The rows are labeled: "s" and "phi".

	R[i,i] (outputVariance^2)
s	0.2
phi	0.01

Buttons at the bottom: OK, Cancel, Import..., Export..., Plot.

The rows of the matrix correspond to the measured outputs of the plant model and in the column "R[i,i]" the diagonal elements of the covariance matrix of the measurement noise can be edited. The higher the values are, the more uncertainty is contained in the measurement signals.

The parameter **sampleFactor** of the Kalman filter model is multiplied by the sample time of the model **sampleClock** in the test model. This product defines the sample time of the Kalman filter model, i.e. the repeating time instant when one step of the Kalman filter algorithm is executed.

The parameter **ModelParameters** covers all model parameters of the original plant model. They can be edited by clicking on the small arrow behind the input field of **ModelParameters**. Then the user should select "Edit". For a filter model of the double pendulum the following list of parameters appears and can be edited:

ModelParameters in KalmanFilter.Examples.DoublePendulum.DoublePendulum_xKF_FMI

General FMI Add modifiers

Component

Name ModelParameters

Comment

Model

Path DoublePendulum_fmU_black_box.InitializationModel

Comment

Parameters

Parameter	Value	Unit	Description
world_g	9.81	m/s ²	Constant gravity acceleration
world_mue	398600000000000.0	m ³ /s ²	Gravity field constant (default = field constant of earth)
world_driveTrainMechanics3D	false		= true, if 3-dim. mechanical effects of Parts.Mounting1D/Rotor1D/BevelGear1D shall be taken into account
world_gravityArrowTail_1_	0	m	Position vector from origin of world frame to arrow tail, resolved in world frame
world_gravityArrowTail_2_	0	m	Position vector from origin of world frame to arrow tail, resolved in world frame
world_gravityArrowTail_3_	0	m	Position vector from origin of world frame to arrow tail, resolved in world frame
world_gravitySphereDiameter	12742000	m	Diameter of sphere representing gravity center (default = mean diameter of earth)
world_defaultWidthFraction	20	1	Default for shape width as a fraction of shape length (e.g., for Parts.FixedTranslation)
world_defaultFrameDiameterFraction	40	1	Default for arrow diameter of a coordinate system as a fraction of axis length
world_defaultSpecularCoefficient	0.7		Default reflection of ambient light (= 0: light is completely absorbed)
world_defaultN Oto 0m	1000	N/m	Default scaling of force arrows (length = force/defaultN_to_m)
world_defaultNm Oto 0m	1000	N m/m	Default scaling of torque arrows (length = torque/defaultNm_to_m)

Icon

Initialization...

FMI

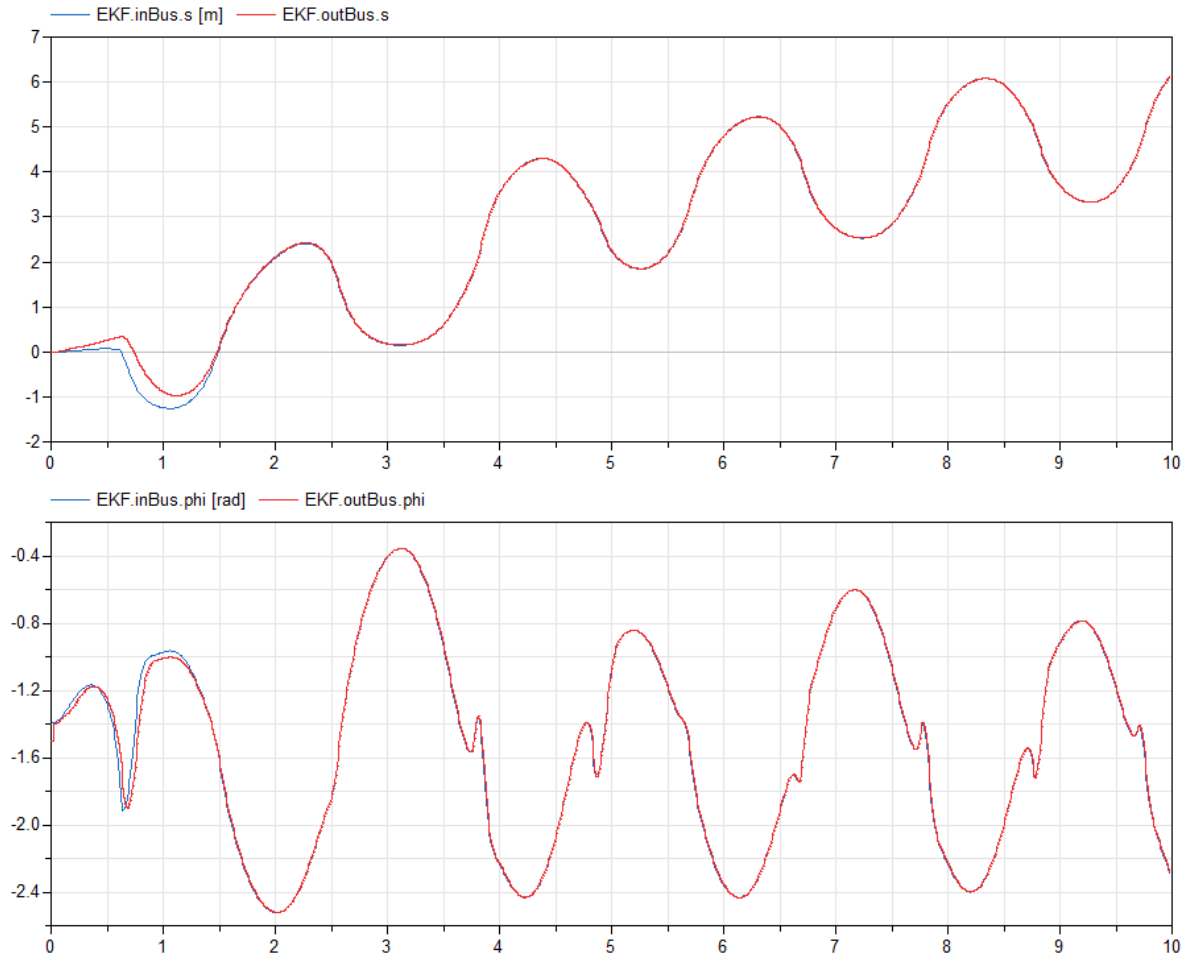
ModelName

Model

OK Info Cancel

1.6. Simulation Results

The test model for the EKF algorithm applied to the double pendulum model is `KalmanFilter.Examples.DoublePendulum.DoublePendulumTestFilter`. If the test model is simulated for 10 seconds, then the following simulation results for the `inBus` and `outBus` variables can be inspected:



In each plot the measured outputs are compared with the estimated outputs of the filter. After a short settling time the measured and estimated output variables coincide very well as expected.

2. Advanced Version of Kalman Filter Library

In this section some features of an advanced version of the Kalman Filter library are described. The advanced version is not open source and will become available as a commercial Modelica library. For more information, contact sr-modelica@dlr.de.

The advanced version of the library provides the framework to *automatically generate* a Modelica Kalman filter model from any Modelica plant model. The process described in [1] is implemented in the library by Modelica scripting functions for steps B, C and D as follows:

- A Model a plant in Modelica with top level inputs and outputs.
- B Export this plant model to an FMU for Co-Simulation 2.0 with some extensions.
- C Import this FMU into a Modelica package based on `FMUImportTemplate`.
- D Generate a tailored filter model.
- E Instantiate the tailored filter model into an overall Modelica model.

This section includes the description of modeling the plant in Modelica and the description of the user interface for the automatic export of the plant model, the re-import to an FMU package and the generation of the tailored filter model.

2.1. Plant Model for a Kalman Filter in Modelica

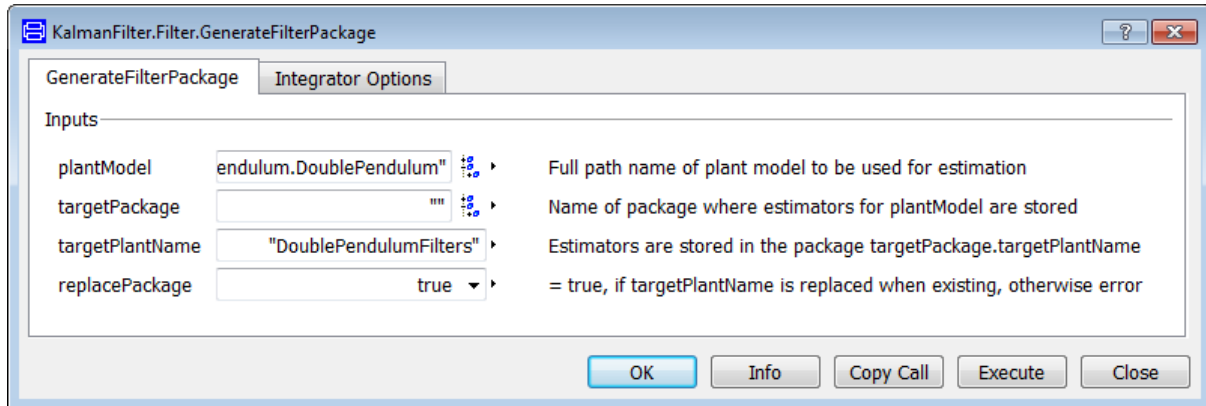
In step A the user has to provide a Modelica plant model of the system that shall be used within the Kalman filter. Theoretically, the system functions f and h (which are automatically determined by Dymola) of the model have to be sufficiently smooth to work within the Kalman filter. In practice, the filter can also be tested with plant models containing events (e.g. introduced by "if-then-else" parts in the model equations). The plant model has to have **top level inputs** and **top level outputs**. These variables are used as inputs u and measurement variables y in the Kalman filter, see the figure in Section 1.1. In addition, the Modelica plant model has to have **continuous-time states**. These are automatically used in the estimation algorithms as states x . If the plant model does not have any continuous-time states, the estimation algorithms cannot be applied.


The double pendulum plant model `DoublePendulum.DoublePendulumForFmi` consists of one input variable: the horizontal force on the moving trolley. At the trolley a double pendulum hangs and moves a load mass at the end of the double pendulum. The output variables of the plant model are the relative angle ϕ in the upper joint and the horizontal position s of the trolley. The continuous-time states of the double pendulum model can be shown in Dymola:

```
prismatic.s  
prismatic.v  
rev.phi  
rev.w  
revolute2.phi  
revolute2.w
```

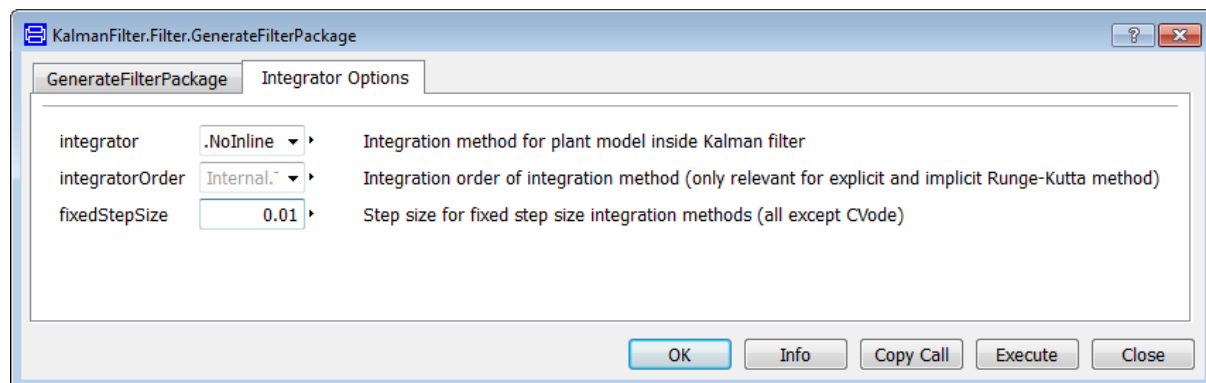
2.2. Automatic Generation of Tailored Kalman Filter Models

Steps B, C and D of the process above can be controlled by the following menu of the Kalman Filter library:

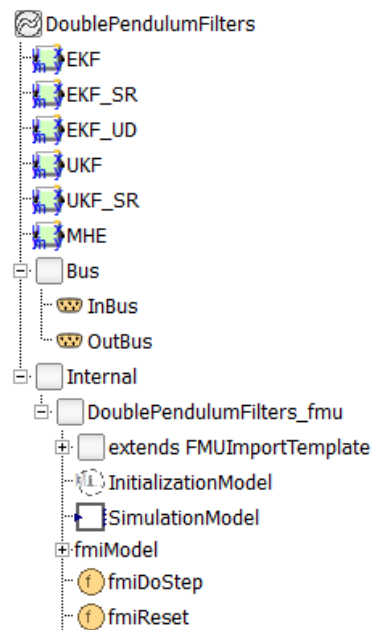


There are two tabs in this menu: *GenerateFilterPackage* and *Integrator Options*. In the tab *GenerateFilterPackage* the user has to open a model browser and select the plant model by clicking on the button  behind the input field of **plantModel**. The **target package** is the name of an existing package / sub-package where the new generated Kalman filter package (containing the filter models) will be stored. The default is "", i.e. the filter package is generated on top level of Dymola's package browser. The intention of the target package is to collect filter packages of different plant models in one package to keep the overview. The string of **targetPlantName** is the name of the Kalman filter package to be generated by the function. If the user sets **replacePackage** = true, then an existing filter package given by the name in **targetPlantName** will be replaced during the generation process (e.g. because the filter package is newly generated after changing some details in the plant model) – otherwise an error message will appear.

In the second tab *Integrator Options* the user can select the numerical **integration** algorithm that is applied to the plant model inside the Kalman filter. The default setting is CCode / NoInline, i.e. a multistep method with variable order and variable step size, similar to the DASSL implementation. The other methods that can be selected are inline integrators. All the inline integrators are **fixed step size** methods for which the step size has to be specified in the GUI. For implicit and explicit Runge-Kutta methods the **integration order** has to be given in the range of 2 – 4.



After this configuration the user can press the "OK" button and a Modelica package with the specified name is automatically generated. The result looks like the following:



The import parts are the filter models `EKF`, `EKF_SR`, ... and the buses `Bus.InBus`, `Bus.OutBus`. The parts of the package `Internal` are automatically used by the filter models and shall not be applied directly in other models.

By this automatic generation approach, handling of different Kalman filter models for a specific Modelica plant model is rather user convenient. Also, testing different types of Kalman filters is directly provided by the advanced version of the Kalman Filter library.

3. References

- [1] Deliverable D3.1.2: *Standardized interfaces for continuous-time models as needed for state and parameter estimation in the MODRIO tool chains*, ITEA2 project MODRIO (11004), Version 2.0, 2014.
- [2] Deliverable D3.1.1: *Method to extend models for system design to models for system operation*, ITEA2 project MODRIO (11004), Version 1.0, 2013.