Modelica Libraries for Linear Control Systems

Marcus Baur, Martin Otter, Bernhard Thiele
German Aerospace Center (DLR), Institute of Robotics and Mechatronics, Germany
Marcus.Baur@DLR.de, Martin.Otter@DLR.de, Bernhard.Thiele@DLR.de

Abstract

This article presents and describes the new LinearSystems and Controller libraries which are developed to enhance analysis, design and simulation of linear control systems in Modelica. The LinearSystems library contains basic functions for linear system analysis and controller design for state-space, transfer-function, and zeros-and-poles representation. The library utilizes the operator overloading technique from Modelica 3.1. The Controller library provides input/output blocks for these basic system descriptions and allows to quickly switch between a continuous and a discrete representation.
Keywords: linear systems; control design; system control; sampled systems

1 Introduction

This article gives an overview of two new, open source Modelica libraries to enhance the analysis, design and simulation of linear control systems in Modelica, i.e. the LinearSystems library and the Controller library. The LinearSystems library contains about 180 Modelica functions for the analysis and design of linear control systems in different description forms. The Controller library contains about 30 controller blocks where it is easy to switch between a continuous and a discrete representation of the blocks. The Controller library is based on the description forms provided by the LinearSystems library.

All numerical functions of the LinearSystems library are natively implemented in Modelica, with exception of linear algebra functions (e.g. solving linear systems of equations) that use the LAPACK library [6]. Therefore, the functions of the LinearSystems library can be used in the production code of a controller, e.g., to design at every sample instant a linear observer for a non-linear plant model that is linearized around the actual operating point. The goal is to use this functionality in combination with the Modelica_EmbeddedSystems library [4] for advanced embedded control systems, where non-linear inverse plant models are present in a controller.

Parts of the functions and blocks of the two libraries are based on the Modelica_LinearSystems library described in [9]. It is planned that both libraries will be included in one of the next versions of the Modelica Standard Library. Currently, they are again collected in one library called Modelica_LinearSystems2 for the LinearSystems library and the sublibrary Modelica_LinearSystems2.Controller for the Controller library.

2 LinearSystems library

The LinearSystems library contains currently about 180 functions, whereas the previous version had about 20 functions. A screen shot of the first hierarchical level of the library is shown in Figure 1. Most important are the four records that contain the basic data structures and functions for linear control systems according to the following mathematical description forms:

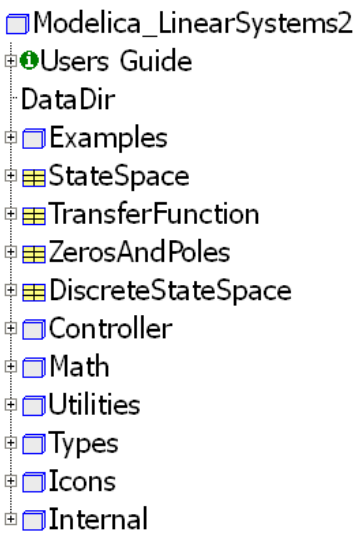

Figure 1: LinearSystems library

DOI: 10.3384/ecp09430068

- StateSpace

$$\begin{aligned}\dot{\mathbf{x}}(t) &= \mathbf{A}\mathbf{x}(t)+\mathbf{B}\mathbf{u}(t)\\ \mathbf{y}(t) &= \mathbf{C}\mathbf{x}(t)+\mathbf{D}\mathbf{u}(t)\end{aligned}$$

- TransferFunction

$$y = \frac{n(s)}{d(s)}\cdot u$$

- ZerosAndPoles

$$y = k\cdot\frac{\prod(s+n_{1i})\cdot\prod(s^2+n_{2j}s+n_{3j})}{\prod(s+d_{1k})\cdot\prod(s^2+d_{2l}s+d_{3l})}\cdot u$$

- DiscreteStateSpace

$$\begin{aligned}\mathbf{x}_d(t_{k+1}) &= \mathbf{A}\mathbf{x}_d(t_k)+\mathbf{B}\mathbf{u}(t_k)\\ \mathbf{y}(t_k) &= \mathbf{C}\mathbf{x}_d(t_k)+\mathbf{D}\mathbf{u}(t_k)\\ \mathbf{x}(t_k) &= \mathbf{x}_d(t_k)+\mathbf{B}_2(t_k)\end{aligned}$$

with the Laplace variable s. Note, that the matrix $\mathbf{B}_2$ to derive the continuous state space vector $\mathbf{x}(t_k)$ from discrete $\mathbf{x}_d(t_k)$ depends on the linearization method. The users view of the ZerosAndPoles data structure are the zeros, poles and the gain of the transfer function. Internally, the transfer function is represented by the real coefficients of first and second order polynomials, so that the operations on this representation results in transfer functions with real coefficients. If complex poles and zeros would be used, inaccuracies in computational calculation could result in systems with complex poles without the conjugated complex counterpart.

The data of the mathematical description forms are stored in the respective record, e.g., the matrices **A**, **B**, **C**, **D** are the data of the state space representation stored in the StateSpace record. Additionally, the signal names can be optionally stored as well. When linearizing a Modelica model, e.g., with `StateSpace.Import.fromModel(..)`, the full signal names of the original model are automatically included in the linear system record.

StateSpace-, TransferFunction-, and ZerosAndPoles-records have the same basic structure. As an example, the first hierarchical level of the StateSpace record consisting of several sublibraries is shown in Figure 2.

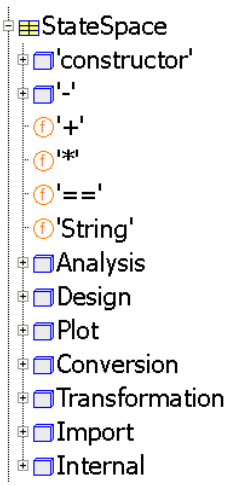

Figure 2: StateSpace record

The first few elements, coated with quotes, are made for the usage of operator overloading, see [8], in order that the elementary operations $+,-,*,==$, String(..) on the data structure can be conveniently carried out.

For example, the following transfer functions

$$G_0(s) = \frac{s+1}{s^2+3s-2}$$

and

$$G_1(s) = \frac{G0}{1+G0}$$

can be easily defined in the interactive environment of Dymola [3] in the following way:

```
import tf =
Modelica_LinearSystems2.TransferFunction;
s  = tf.s();
G0 = (s+1)/(s^2+3*s-2);
G1 = G0 / (1 + G0);
G1
  // = "(s^3 + 4*s^2 + s - 2)/
  //    (s^4 + 7*s^3 + 9*s^2 - 11*s + 2)"
```

The new Command window of Dymola shows such results appropriately rendered, e.g. Figure 3.

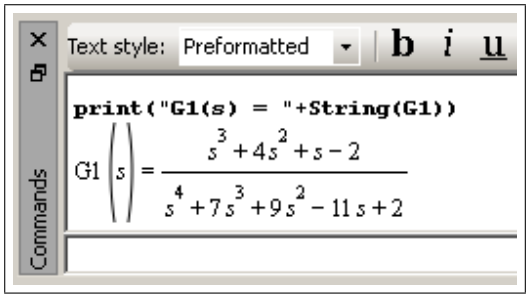

Figure 3: Dymola Command window

The further sublibraries structure the available functions:

- **Analysis** contains functions to compute eigenvalues, poles, zeros, or properties like controllability.
- **Design** contains functions to design control systems, e.g., with the pole placement or the Riccati method.

- **Plot** contains functions to compute and plot poles and zeros, frequency responses, step responses etc.
- **Conversion** contains functions to convert from one data structure to another description form, e.g., from StateSpace to TransferFunction.
- **Transformation** contains functions to perform a similarity transformation, e.g., to transform to controllability form.
- **Import** contains functions to import the data structure from a model (by linearization) or from a file.

All functions of the library are Modelica functions. For basic linear algebra computations, like solutions of linear systems of equations, or eigenvalue computations, the standard numerical library LAPACK [6] is used. Besides LAPACK, no other external functions are utilized.

In the following sections, some of the above quoted sublibraries are described in more detail for StateSpace systems.

2.1 StateSpace.Analysis

The Analysis package of StateSpace contains functions to compute eigenvalues, invariant zeros, and various system properties.

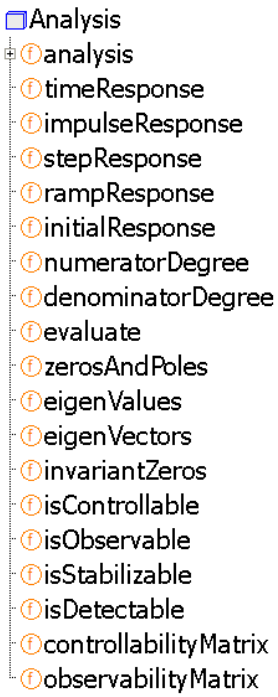

Figure 4: Package StateSpace.Analysis

The eigenvalues of a state space system, which are the eigenvalues of the system matrix $\mathbf{A}$, are calculated with the LAPACK function dggevx, i.e., using the basic eigenvalue computation with an additional balancing transformation to improve the conditioning of the eigenvalues. The principle of the algorithm is to reduce matrix $\mathbf{A}$ to an upper Hessenberg form first. The QR algorithm is then used to further reduce the matrix to a real Schur form (RSF) from which the eigenvalues are easily computed.

Beside the eigenvalues, also the invariant zeros play an important role for linear dynamic systems. They identify those exponential input signals that are completely blocked by the system. A complex number $s = z_k$ is an invariant zero of a state space system

$$\begin{aligned} \dot{\mathbf{x}} &= \mathbf{A}\mathbf{x} + \mathbf{B}\mathbf{u} \\ \mathbf{y} &= \mathbf{C}\mathbf{x} + \mathbf{D}\mathbf{u} \end{aligned}$$

if the Rosenbrock matrix

$$\mathbf{P}(s) = \begin{pmatrix} s\mathbf{I} - \mathbf{A} & \mathbf{B} \\ -\mathbf{C} & \mathbf{D} \end{pmatrix}$$

is rank deficient for $s = z_k$, i.e.

$$rank(\mathbf{P}(z_k)) < \max_s (rank(\mathbf{P}(s))) \qquad (1)$$

(note, that the modification of the signs in the matrix do not change the rank). If the system has the same number of inputs and outputs, the invariant zeros are the generalized eigenvalues of the pair of square matrices $(\mathbf{L}, \mathbf{M})$ with

$$\mathbf{L} = \begin{pmatrix} \mathbf{A} & \mathbf{B} \\ \mathbf{C} & \mathbf{D} \end{pmatrix}, \quad \mathbf{M} = \begin{pmatrix} \mathbf{I} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} \end{pmatrix}.$$

In this case the generalized eigenvalues, i.e., the invariant zeros, could be computed with a standard QZ-algorithm, e.g., with LAPACK function dggev. However, this algorithm fails if the system has not the same number of inputs and outputs or if one of the matrices $\mathbf{B}$ or $\mathbf{C}$ does not have full column or full row rank respectively. For this reason, in the LinearSystems library the algorithm from [5] is used to calculate the invariant zeros of arbitrary StateSpace systems, i.e., with arbitrary numbers of inputs and outputs and rank deficient matrices. The approach is to compress the matrices $(\mathbf{A}, \mathbf{B}, \mathbf{C}, \mathbf{D})$ with QR-decompositions to $(\mathbf{A}_r, \mathbf{B}_r, \mathbf{C}_r, \mathbf{D}_r)$ such that a reduced order system matrix

$$\mathbf{P}_r(s) = \begin{pmatrix} s\mathbf{I} - \mathbf{A}_r & \mathbf{B}_r \\ -\mathbf{C}_r & \mathbf{D}_r \end{pmatrix}$$

with invertible matrix $\mathbf{D}_r$ have the same zeros as $\mathbf{P}(s)$. After another transformation that compresses the columns of $[\mathbf{C}_r, \mathbf{D}_r]$ to $[\mathbf{0}, \mathbf{D}_f]$ such that

$$\begin{pmatrix} \mathbf{A}_f & * \\ \mathbf{0} & \mathbf{D}_f \end{pmatrix} = \begin{pmatrix} \mathbf{A}_r & \mathbf{B}_r \\ \mathbf{C}_r & \mathbf{D}_r \end{pmatrix} \mathbf{V}$$

and

$$\begin{pmatrix} \mathbf{B}_f & * \\ \mathbf{0} & \mathbf{0} \end{pmatrix} = \begin{pmatrix} \mathbf{I} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} \end{pmatrix} \mathbf{V}$$

the QZ-algorithm is applied to calculate the generalized eigenvalues of the pair (A_f, B_f). These generalized eigenvalues are the invariant zeros of the system.

Controllability, observability, stability, detectability, and stabilizability are computed with numerically effective staircase algorithms. All properties can be computed with the convenience function "analyze" that calls all functions from the Analysis package and presents the results in a nice layout in html format. The requested results are selected by a menu (see Figure 5). Furthermore, the dynamics of the states are being analyzed in relation to the dynamics of the modal states, i.e., the states of the corresponding similar uncoupled modal system representation. This helps to understand which eigenvalue and/or eigen response is associated with which variable.

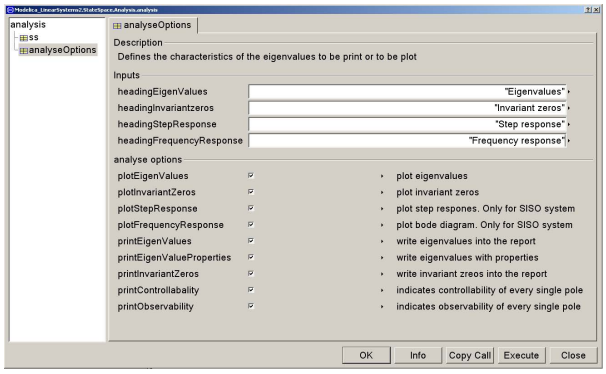

Figure 5: Menu of Analysis.analysis function

The following example illustrates this relation: The state space system with the matrices

$$A = \begin{pmatrix} -3 & 2 & -3 & 4 \\ 0 & 6 & 7 & 8 \\ 0 & 13 & 34 & 0 \\ 0 & -17 & 0 & 0 \end{pmatrix}, \; B = \begin{pmatrix} 1 & 0 \\ 0 & 1 \\ 1 & 0 \\ 0 & 1 \end{pmatrix}$$

$$C = \begin{pmatrix} 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 \end{pmatrix}, \qquad D = \begin{pmatrix} 0 & 0 \\ 0 & 0 \end{pmatrix}$$

has the four eigenvalues

$$\lambda_1 = -3, \; \lambda_2 = 36.65, \; \lambda_{3,4} = 1.67 \pm 11.11\, i.$$

The system analysis function reports that this system is neither stable nor observable but it is controllable and therefore it is stabilizable and it is detectable. In particular:

	characteristics
λ_1	stable, controllable, not observable
λ_2	not stable, stabilizable, detectable
$\lambda_{3,4}$	not stable, stabilizable, detectable

The contribution of the modal states $z_i(t)$ to the system states $x_i(t)$ and the characteristics of the dynamic behavior of the modal states is given by the following tables on the result file in html format:

i	λ_i	T[s]	z[i] contributes
1	-3	0.333	to x[1] with 100 %
2	36.65	0.273	to x[3] with 72.8 % to x[2] with 14.8 %

i	λ_i	z[i] contributes
3/4	$1.67 \pm \mathrm{i}\, 11.11$	to x[4] with 43.9 % to x[2] with 29 %

i	frequency[Hz]	damping
3/4	1.7877	-0.1492

The relation of the system states to the modal states, i.e., the composition of the system states is shown in the next table:

System state	is composed of
x[1]	60.6 % by z[1] 34.9 % by z[3/4]
x[2]	83.9 % by z[3/4] 16.1 % by z[2]
x[3]	71.2 % by z[2] 28.8 % by z[3/4]]
x[4]	94.4 % by z[3/4] 5.6 % by z[2]

The meaning of contribution and composition is explained subsequently. The idea of this approach is, that the zero input response

$$\mathbf{x}_h(t) = e^{\mathbf{A}t}\mathbf{x}_0$$

of a state space system can be represented by a transformation

$$\mathbf{x}_h(t) = \tilde{\mathbf{V}}\mathbf{z}_h(t) \tag{2}$$

of the decoupled zero input responses $\mathbf{z}_h(t)$ of the corresponding modal system

$$\dot{\mathbf{z}} = \tilde{\mathbf{V}}^{-1}\mathbf{A}\tilde{\mathbf{V}}\mathbf{z} = \tilde{\mathbf{\Lambda}}\mathbf{z}$$

with

$$z_k(t) = e^{\lambda_i t} z_{k_0}$$

for a single real eigenvalue λ_k or

$$\begin{pmatrix} z_k(t) \\ z_{k+1}(t) \end{pmatrix} = e^{\mathbf{\Gamma}_k t} \begin{pmatrix} z_{k_0} \\ z_{k+1_0} \end{pmatrix}$$

for a single pair of conjugated complex eigenvalues $(\lambda_k, \lambda_{k+1}) = \alpha_k \pm i\beta_k$ respectively. The matrix

$$\mathbf{\Gamma}_k = \mathbf{T}^{-1}\mathbf{\Lambda}_k\mathbf{T} = \begin{pmatrix} \alpha_k & \beta_k \\ -\beta_k & \alpha_k \end{pmatrix}$$

with

$$\mathbf{T} = \frac{1}{2}\begin{pmatrix} 1 & -j \\ 1 & j \end{pmatrix} \text{ and } \mathbf{\Lambda}_k = \begin{pmatrix} \alpha_k + j\beta_k & 0 \\ 0 & \alpha_k - i\beta_k \end{pmatrix}$$

results from a transformation of the complex eigenvalue matrix $\mathbf{\Lambda}_k$ into the corresponding real form. The matrix $\tilde{\mathbf{V}}$ is given by

$$\tilde{\mathbf{V}} = \mathbf{V}\begin{pmatrix} \hat{\mathbf{T}} & \mathbf{0} \\ 0 & \mathbf{I} \end{pmatrix}$$

where $\hat{\mathbf{T}}$ is a block diagonal matrix of order $2r$ with r matrices $\mathbf{T}$ in its diagonal. Note, that the matrix $\mathbf{\Lambda} = \mathrm{diag}(\lambda_k)$ is assumed to be appropriately sorted, i.e. the first $2r$ elements of the diagonal are the r conjugated complex pole pairs of the system. Considering (2), each element $x_{h_k}(t)$ of $\mathbf{x}_h(t)$ is represented by a linear combination $\tilde{\mathbf{v}}_k^T\mathbf{z}_h$ of the zero input responses $z_{h_k}(t)$, where $\tilde{\mathbf{v}}_k^T$ indicates the k'th row of matrix $\tilde{\mathbf{V}}$. Furthermore, x_{h_k} is composed by $p_{kl} = 100 \cdot \left(|\tilde{v}_{kl}|/|\tilde{\mathbf{v}}_k^T|\right)\%$ by the element z_{h_l}. On the other hand, equation (2) can be written as

$$\mathbf{x} = \tilde{\mathbf{V}}\mathbf{z} = \sum_{k=1}^{n} z_k\tilde{\mathbf{v}}_k,$$

i.e., for each modal state z_{h_k} of $\mathbf{z}_h$ the elements $q_{lk} = 100 \cdot |\tilde{v}_{lk}|/|\tilde{\mathbf{v}}_k|\%$ of the corresponding vector $\tilde{\mathbf{v}}_k$ indicates the proportion of z_{h_k} that is contributed to the state x_{h_l}. Since the state x_k is assigned to a system variable, which can probably be assigned to a physical component or a certain subsystem, then p_{kl} and q_{lk} help to indicate the influence of this subsystem to the dynamical behavior of the system which, mathematically, is composed of the dynamics of the modal states $\mathbf{z}_k$ associated to the eigenvalues λ_k.

Currently, the poles are only considered as real poles or pole pairs with multiplicity 1. Systems with eigenvalues of higher multiplicity are processed as multiple eigenvalues of multiplicity 1. This should be improved in the future.

2.2 StateSpace.Plot

The sublibrary Plot contains functions to plot poles, zeros, frequency responses and various time responses. For the plotting, a separate small package is provided that must be adapted to the features of the used Modelica tool, since plotting is not standardized in Modelica. Currently, the plot package is only provided for Dymola.

Figure 6: Package StateSpace.Plot

For example, the following function call computes the step responses of a state space system from every input to every output and plots the corresponding curves:

```
StateSpace.Plot.step(
     ss=ss, dt=0.05,tSpan=10);
```

Figure 7 shows a step response of a SISO state space system with the matrices

$$\begin{aligned} \mathbf{A} &= \begin{pmatrix} 0 & 1 \\ -25 & -1.5 \end{pmatrix}, \mathbf{B} = \begin{pmatrix} 0 \\ 25 \end{pmatrix}, \\ \mathbf{C} &= \begin{pmatrix} 1 & -0.5 \end{pmatrix}, \quad \mathbf{D} = \mathbf{0}. \end{aligned}$$

Figure 8 shows the corresponding bode diagram generated with `StateSpace.Plot.bodeSISO(ss=ss);`

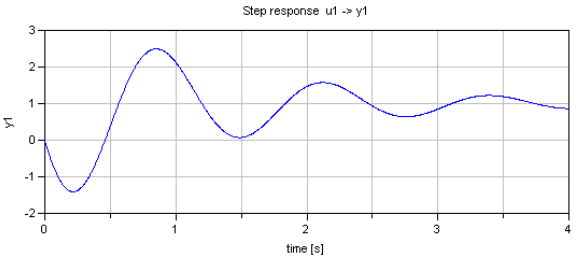

Figure 7: Step response

2.3 StateSpace.Design

The Design sublibrary (see Figure 9) provides standard controller design methods for linear systems. The pole assignment design function `StateSpace.assignPolesMI()` for StateSpace systems with one or more inputs is based on the

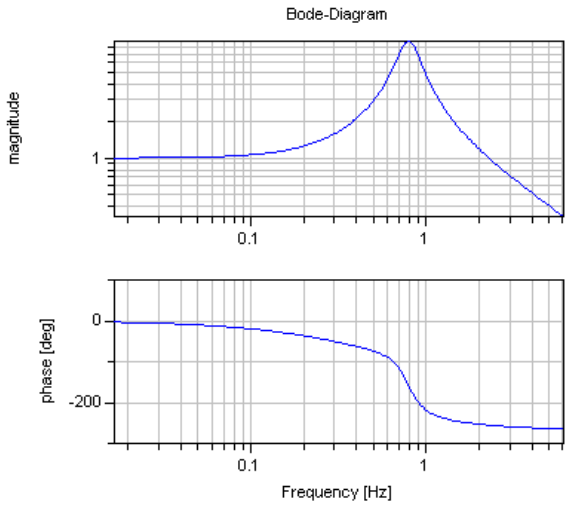

Figure 8: Bode diagram

approach described in [11], where the system matrix $\mathbf{A}$ is reduced to a real Schur form that allows sequential and/or partial eigenvalue assignment.

The standard design method of a "linear quadratic optimal controller" computes the matrix $\mathbf{K}$ of the state-feedback law $\mathbf{u} = -\mathbf{K}\mathbf{x}$ in such a form that a quadratic cost function with symmetric weighting matrices $\mathbf{Q}$ and $\mathbf{R}$

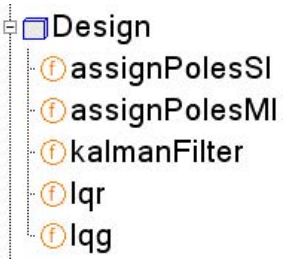

Figure 9: Package StateSpace.Design

$$J(\mathbf{x}) = \int_0^\infty \mathbf{x}^T\mathbf{Q}\mathbf{x} + \mathbf{u}^T\mathbf{R}\,\mathbf{u}\,dt \qquad (3)$$

is minimized. The feedback matrix $\mathbf{K}$ is obtained from the solution $\mathbf{X}$ of the algebraic Riccati equation

$$\mathbf{Q} + \mathbf{A}^T\mathbf{X} + \mathbf{X}\mathbf{A} - \mathbf{X}\mathbf{B}\mathbf{R}^{-1}\mathbf{B}^T\mathbf{X} = \mathbf{0}. \qquad (4)$$

In the LinearSystems library the linear quadratic optimal controller design is performed with the function call `(K, sslqr, X, evlqr) = StateSpace.lqr(ss, Q, R)`. The first output of the function is the optimal and stabilizing gain matrix $\mathbf{K}$. It is calculated from the solution $\mathbf{X}$ of (4) by $\mathbf{K} = \mathbf{R}^{-1}\mathbf{B}^T\mathbf{X}$. Additionally, the complex output vector `evlqr` contains the closed loop system eigenvalues, i.e., the eigenvalues of the matrix $\mathbf{A} - \mathbf{B}\mathbf{K}$. The record `sslqr` contains the system representation of the closed loop system

$$\begin{aligned} \dot{\mathbf{x}} &= (\mathbf{A} - \mathbf{B}\mathbf{K})\mathbf{x} + \mathbf{B}\mathbf{u} \\ \mathbf{y} &= (\mathbf{C} - \mathbf{D}\mathbf{K})\mathbf{x} + \mathbf{D}\mathbf{u}. \end{aligned}$$

The corresponding control system structure for the state feedback control is depicted in Figure 11 and is available as a model template in the Controller library. Alternatively, a feedback control structure with observer, as shown in Figure 12, is also available.

Kalman filter design is related to the LQR-problem. Actually, the corresponding design function makes use of `StateSpace.Design.lqr()` but with a dual system for the observer problem. Kalman filters are computed with `(L, sskf)=StateSpace.kalmanFilter(ss, Q, R)`. Beside the filter matrix $\mathbf{L}$ the output `sskf` represents the system together with the Kalman filter, i.e.

$$\begin{aligned} \dot{\mathbf{x}} &= (\mathbf{A} - \mathbf{L}\mathbf{C})\mathbf{x} + \begin{pmatrix} \mathbf{B} - \mathbf{L}\mathbf{D}, & \mathbf{L} \end{pmatrix} \begin{pmatrix} \mathbf{u} \\ \mathbf{y} \end{pmatrix} \\ \mathbf{y} &= \mathbf{C}\mathbf{x} + \begin{pmatrix} \mathbf{D}, & \mathbf{0} \end{pmatrix} \begin{pmatrix} \mathbf{u} \\ \mathbf{y} \end{pmatrix}. \end{aligned}$$

LQG design determines the matrices $\mathbf{K}_c$ and $\mathbf{K}_f$ for linear quadratic gaussian problems (LQG), i.e., the minimization of the expected value of a cost function under the assumption of stochastically disturbed states and outputs. Therefore, it is a combination of LQR design and Kalman filter design. The function `(Kc, Kf, sslqg)=Design.lqg(ss, Q, R, V, W)` returns the controller matrix $\mathbf{K}_c$ and the filter matrix $\mathbf{K}_f$. Again, the matrices $\mathbf{Q}$ and $\mathbf{R}$ are weighting matrices in the cost function of the controller. The matrices $\mathbf{V}$ and $\mathbf{W}$ are assumed to be the covariance matrices of the disturbances $\mathbf{v}$ and $\mathbf{w}$ respectively but due to the lack of deeper insight are usually treated as weighting matrices. The output record `sslqg` represents the estimated system

$$\begin{aligned} \dot{\hat{\mathbf{x}}} &= (\mathbf{A} - \mathbf{K}_f\mathbf{C} - \mathbf{B}\mathbf{K}_c + \mathbf{K}_f\mathbf{D}\mathbf{K}_c)\hat{\mathbf{x}} + \mathbf{K}_f\mathbf{y} \\ \hat{\mathbf{y}} &= (\mathbf{C} - \mathbf{D}\mathbf{K}_c)\hat{\mathbf{x}} \end{aligned}$$

with $\mathbf{y}(t)$, the output of the original system, as input.

The solution of the Riccati equation (4) is provided by `X = Modelica_LinearSystms2.Math.Matrices.care(A, B, R, Q)`, which computes the solution with a Schur vector method for the continuous algebraic Riccati equation (CARE) [2, 7]. The approach is to form the $2n \times 2n$ Hamilton matrix (note that matrix $\mathbf{A}$ has order n)

$$\mathbf{H} = \begin{pmatrix} \mathbf{A} & -\mathbf{B}\mathbf{R}^{-1}\mathbf{B}^T \\ -\mathbf{Q} & -\mathbf{A}^T \end{pmatrix}$$

and to transform it into an ordered real Schur form

$$\mathbf{T} = \begin{pmatrix} \mathbf{T}_{11} & \mathbf{T}_{12} \\ \mathbf{0} & \mathbf{T}_{22} \end{pmatrix} = \mathbf{U}^T \mathbf{H} \mathbf{U}$$

where $\mathbf{T}_{11}$ contains the n eigenvalues of $\mathbf{H}$ with negative real parts [2]. Considering an appropriate partitioning of matrix $\mathbf{U}$

$$U = \begin{pmatrix} \mathbf{U}_{11} & \mathbf{U}_{12} \\ \mathbf{U}_{21} & \mathbf{U}_{22} \end{pmatrix},$$

with $n \times n$ matrices $\mathbf{U}_{ij}$, the solution $\mathbf{X}$ can be computed by solving

$$\mathbf{X}\mathbf{U}_{11} = \mathbf{U}_{21}.$$

Additionally, an optional refinement function based on Newton's Method with exact line search is provided [1].

3 Controller library

The former beta version of the library Modelica_LinearSystems [9] contained a sub-package *Sampled* with input/output blocks for continuous and discrete linear systems simulation. This package was extended and was adapted to the Modelica_LinearSystems2 library sketched in the previous section. Furthermore, the package was renamed and is now called "Modelica_LinearSystems2.Controller". The Controller library contains input/output blocks for StateSpace, TransferFunction and ZerosAndPoles systems, as well as PI, PID, FirstOrder, SecondOrder, Integrator, Filter blocks etc. Every block is available in a continuous and a discrete (sampled) representation, where the representation can be chosen by a Boolean parameter. By specifying a discretization method and a sample time, the discrete representation is automatically derived from the continuous form.

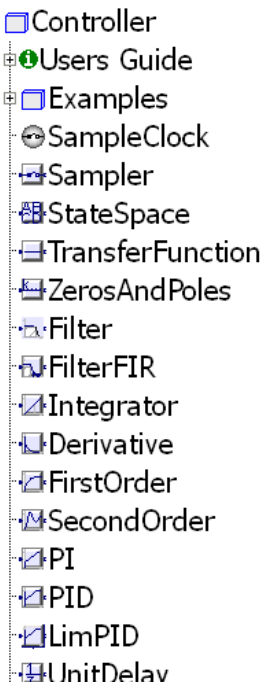

Figure 10: Controller library

Besides standard input/output blocks, especially for the data structures of the LinearSystems library, a "Template" sublibrary is present which provides standard controller structures with replaceable components. As an example, a state feedback control is shown in Figure 11, and state feedback control based on estimated states using an observer is shown in Figure 12.

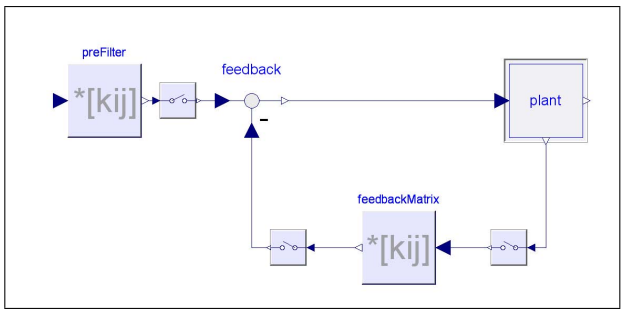

Figure 11: Template for state feedback control

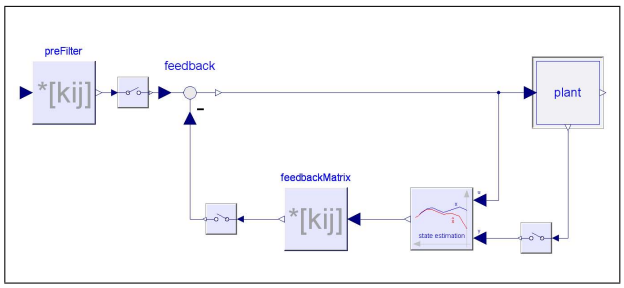

Figure 12: Template for state feedback control with observer

The observer structure is also provided as a template (Figure 13) which automatically adapts the dimensions to the loaded system.

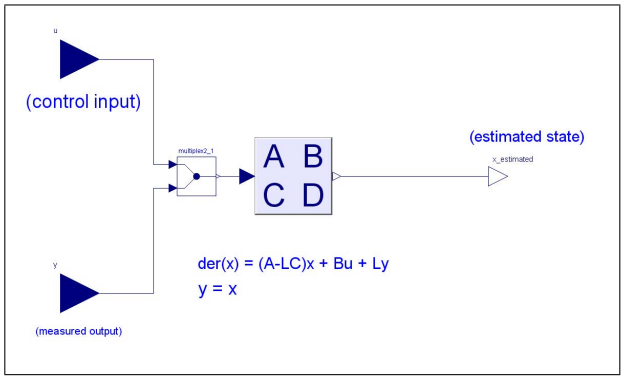

Figure 13: Template for observer

Figure 14 shows a two degree of freedom controller template with a (usually non-linear) in-

verse system model in the feed forward loop. How

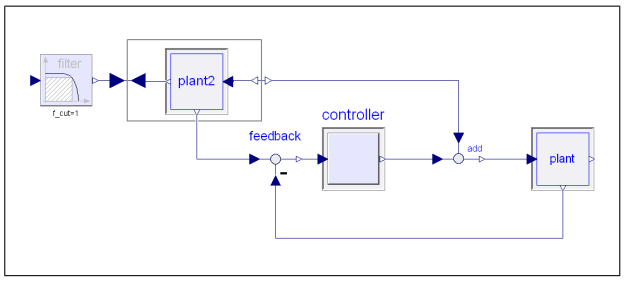

Figure 14: Template for a two degree of freedom controller.

to utilize non-linear inverse (Modelica) models in controllers is described in [10]. The templates are provided to support quick implementations of controllers by simple redeclarations of the replaceable components.

4 Example

Figure 15 shows the multi-body model of a simple inverse double pendulum and Figure 16 shows the corresponding animation of the simulated system.

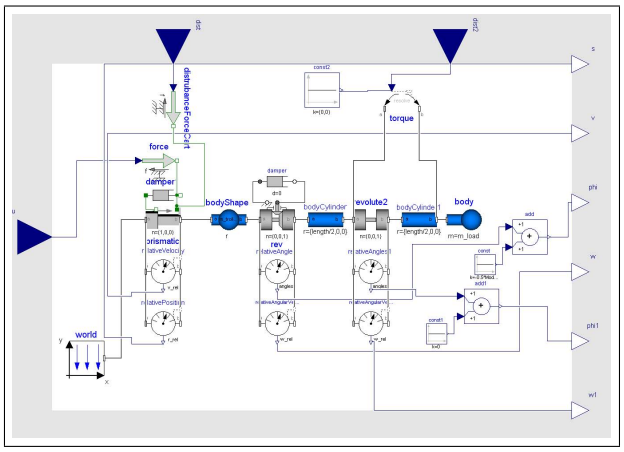

Figure 15: Multi-body model of inverse double pendulum

The input of the system is a one-dimensional horizontal force to move the cart. It is assumed that the position and the angle between the lower rod and the cart are measurable. Hence, the velocity, the angle of the upper joint, and the angular velocities of the joints have to be estimated by an observer. Disturbances can be conditionally activated and can be added to the horizontal force input and as additional torque at the upper joint applied at the upper rod. The control task is to

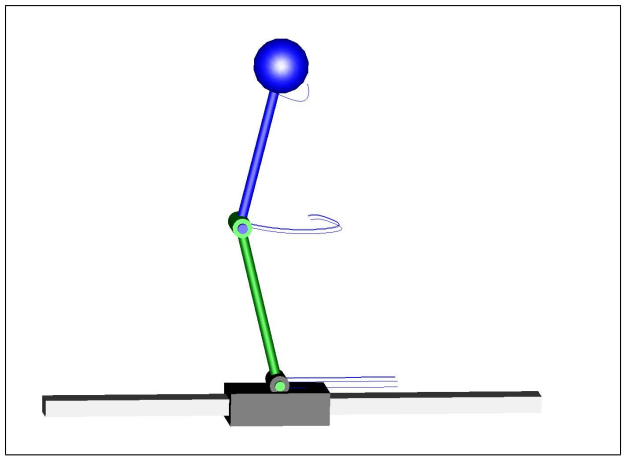

Figure 16: Screenshot of animated simulation

track the cart to a given (time varying) position without dropping the pendulum.

4.1 Controller design

Linearization of the system around the vertical position of the double pendulum is performed numerically with function `StateSpace.Import.fromModel(modelName)`, which returns an appropriate instance of the StateSpace record. Both, the controller and the observer are designed by pole assignment. With the state space system `ss` of the linearized model and the desired poles p_c the feedback matrix K_{p_c} is calculated by calling `K_pc := assignPolesMI(ss, pc);`

Due to the duality of controllability and observability, function `assignPolesMI()` can also be used to design the observer feedback matrix $\mathbf{K}_{p_o}$. The inputs are the dual state space system $\left(\mathbf{A}^T, \mathbf{C}^T, \mathbf{B}^T, \mathbf{D}^T\right)$. Finally, the pre filter, see Figure 12, is designed such that the model would follow a constant input in case of steady state behavior:

```
M_pa := -inv(ss.C*Matrices.solve2(
  ss.A - ss.B*K_pc, ss.B));
```

4.2 Simulation

The model of the controlled system (Figure 17) is extended from the corresponding template shown in Figure 12. To fit the template, the physical system model must be put into an appropriate form as depicted in Figure 18. The data of the controller are directly copied into the controller

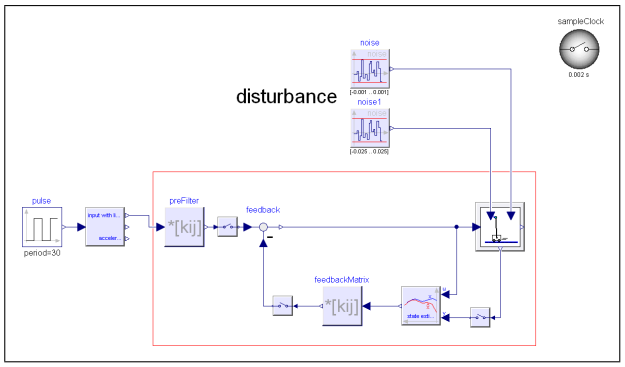

Figure 17: Block diagram of the controlled system

matrix or loaded from a file.

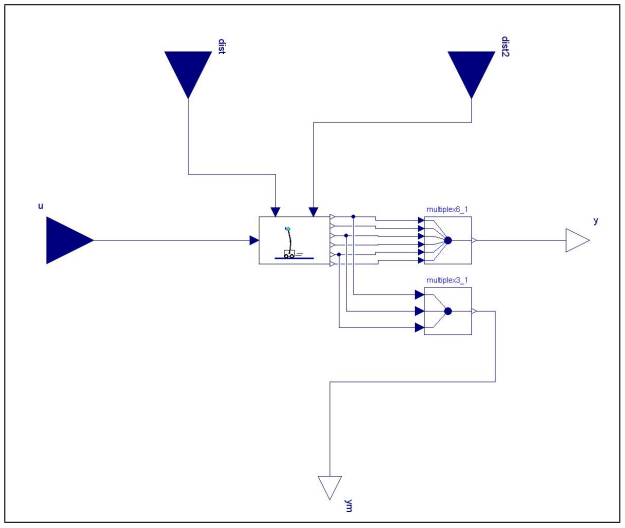

Figure 18: Model to fit the physical model of the template shape

The characteristics of the undisturbed controlled continuous system is shown in Figure 19. In the upper picture, the position of the inverted pendulum and its set point are depicted. The lower picture shows the corresponding input force. Figure 20 shows the same signals of a disturbed system with a discrete controller.

The disturbances have of course much impact on the angle of the upper joint. By comparison, Figure 21 shows the first 15 seconds of this angle and the corresponding estimated value for the undisturbed system (upper picture) and the disturbed system (lower picture).

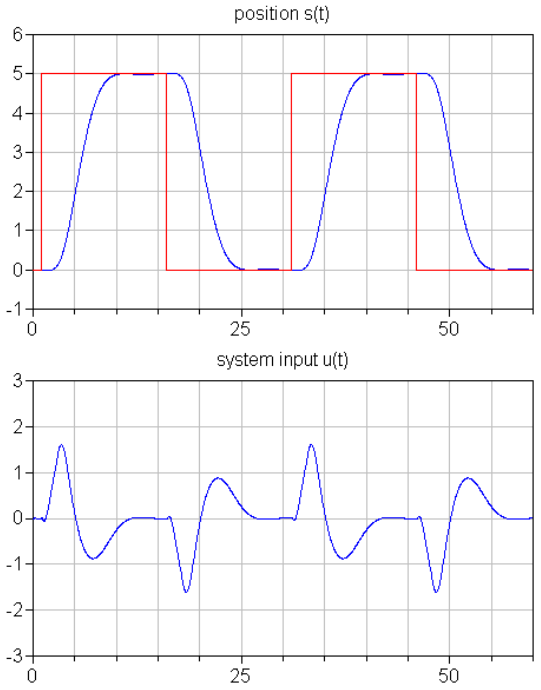

Figure 19: Controlled system without disturbances and with continuous controllers. Position of the inverted pendulum and its setpoint (above) and the input force (below)

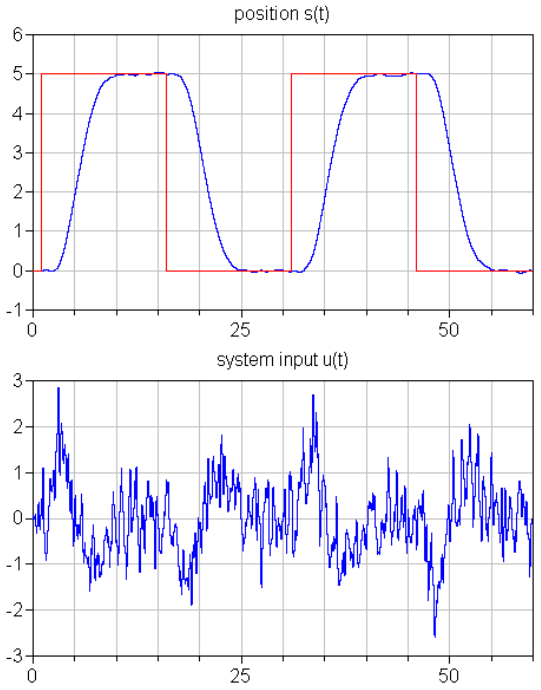

Figure 20: Controlled system with disturbances and with discrete controllers

5 Conclusions

This paper has sketched the Modelica libraries LinearSystems and Controller, i.e. a library for linear systems analysis and synthesis and a library to model continuous and discrete linear controller blocks. Compared with the former version

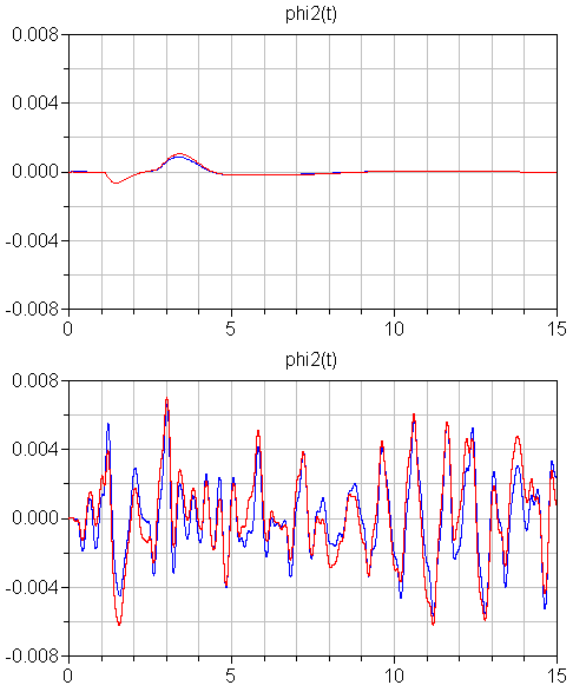

Figure 21: Angle φ_2 (upper joint) and estimated value (red line) for undisturbed (above) and disturbed (below) system

of the LinearSystems library, the current version has been extensively restructured and new functions have been added. The most important functions of this library have been described for the case of state space systems and the mathematical approaches have been outlined.

Concerning the Controller library, the new feature to use templates for common control system structures have been introduced. An example of the design of a controller for a inverse double pendulum demonstrates the usage of the libraries. It is planned to include the libraries in the Modelica Standard library. Currently, they are available from http://www.Modelica.org/libraries.

Acknowledgements

Partial financial support of DLR by BMBF (BMBF Förderkennzeichen: 01IS07022F) for this work within the ITEA project EUROSYSLIB (http://www.itea2.org/public/project_leaflets/EUROSYSLIB_profile_oct-07.pdf) is highly appreciated.

We would also like to thank Hilding Elmqvist and Hans Olsson from Dassault Systèmes (Dynasim) for their support and for discussions especially related to operator overloading.

References

[1] Benner P. and Byers, R. (1998): **An Exact Line Search Method for Solving Generalized Continuous-Time Algebraic Riccati Equations**. IEEE Transactions on Automatic Control, vol 43, pp. 101-107

[2] Datta B. N. (2004): **Numerical Methods for Linear Control Systems**. Elsevier Academic Press.

[3] Dymola (2009): **Dymola Version 7.3**. Dassault Systèmes, Lund, Sweden (Dynasim). http://www.dymola.com.

[4] Elmqvist H., Otter M., Henriksson D., Thiele B., and Mattsson S. E. (2009): **Modelica for Embedded Systems**. In: Proc. of the 7th Modelica Conference 2009, Como, Italy, Sept. 20-22. http://www.modelica.org/events/modelica2009.

[5] Emami-Naeini, A. and Van Dooren, P. (1982): **Computations of zeros of linear multivariable systems**, Automatica 26, pp. 415-430

[6] LAPACK (2009): http://www.netlib.org/lapack/.

[7] Laub A. J. (1979): **A Schur Method for Solving Algebraic Riccati equations**. IEEE Trans. Auto. Contr., vol 24, pp. 913-921.

[8] Olsson H., Otter M., Elmqvist H., and Brück D. (2009): **Operator Overloading in Modelica 3.1**. In: Proc. of the 7th Modelica Conference 2009, Como, Italy, Sept. 20-22.

[9] Otter M. (2006): **The LinearSystems library for continuous and discrete control systems**. In: Proc. of the 5th Modelica Conference 2006, Wien, Austria, Sept. 4-5. http://www.modelica.org/events/modelica2006/-Proceedings/sessions/Session5c1.pdf

[10] Thümmel M., Looye G., Kurze M., Otter M., and Bals J. (2005): **Nonlinear Inverse Models for Control**. In: Proc. of the 4th Int. Modelica Conference 2005, Hamburg, March 7-8. http://www.modelica.org/events/Conference2005/-online_proceedings/Session3/Session3c3.pdf

[11] Varga A. (1981): **A Schur method for pole assignment**. IEEE Trans. Autom. Control, Vol. AC-26, pp. 517-519.

