Proceedings
of the 4th International Modelica Conference,
Hamburg, March 7-8, 2005,
Gerhard Schmitz (editor)

H. Elmqvist, H. Olsson, S.E. Mattsson, D. Brück, C. Schweiger, D. Joos, M. Otter
Dynasim AB, Sweden; DLR, Oberpfaffenhofen, Germany
Optimization for Design and Parameter Estimation
pp. 255-266

Program Committee

- Prof. Gerhard Schmitz, Hamburg University of Technology, Germany (Program chair).
- Prof. Bernhard Bachmann, University of Applied Sciences Bielefeld, Germany.
- Dr. Francesco Casella, Politecnico di Milano, Italy.
- Dr. Hilding Elmqvist, Dynasim AB, Sweden.
- Prof. Peter Fritzson, University of Linkping, Sweden
- Prof. Martin Otter, DLR, Germany
- Dr. Michael Tiller, Ford Motor Company, USA
- Dr. Hubertus Tummescheit, Scynamics HB, Sweden

Local Organization: Gerhard Schmitz, Katrin Prölß, Wilson Casas, Henning Knigge, Jens Vasel, Stefan
Wischhusen, TuTech Innovation GmbH

Optimization for Design and Parameter Estimation

Hilding Elmqvist[1], Hans Olsson[1], Sven Erik Mattsson[1], Dag Brück[1],
Christian Schweiger[2], Dieter Joos[2], Martin Otter[2]

[1]Dynasim AB, Lund, Sweden ({Elmqvist, Hans.Olsson, SvenErik, Dag}@Dynasim.se)
[2]DLR Institute of Robotics and Mechatronics, Oberpfaffenhofen, Germany
(Christian.Schweiger@DLR.de, Dieter.Joos@DLR.de, Martin.Otter@DLR.de)

Abstract

This paper describes new features of the Modelica environment Dymola to perform integrated computer experiments with Modelica models, in particular for model calibration, design optimization and robustness or sensitivity assessment based on multiple criteria and multiple simulation runs. The environment and especially the problem setup are demonstrated by several application examples.

1 Introduction

Recently, new features have been added to the Modelica simulation environment Dymola [2] to simplify experimentation with Modelica models significantly. Some basic ideas are from the optimization environment MOPS [3]. The central part is a Modelica model of a physical system with Modelica parameters that are not yet fixed. Several problem classes can now be conveniently solved:

- **Model Calibration (Parameter Estimation)**:
  Some Modelica parameters of the model are not known. Several simulation runs are performed and compared with measurement data that is available from equivalent dynamic behavior of the real device. Via optimization, the selected unknown parameters and initial conditions are determined such that the simulations and the measurement data are in good agreement. Also standard tasks such as fitting of functions to measurement data is supported.

- **Design Optimization (Parameter Optimization):**
  Selected Modelica parameters of the model are tuned to improve the system dynamics, e.g., by changing the parameters of a controller or some parameters of the physical device. This is performed by multi-criteria parameter optimization using one or several simulation runs to compute the desired criteria.

- **Assessment (Parameter Variation):**
  Selected Modelica parameters of the model are systematically changed within a given grid and for every fixed set of parameters, simulations are performed. This might be used, e.g., to evaluate a finished design by varying the operating points. For Monte Carlo simulations, the parameter values are chosen statistically. Simulations with small variations to the parameters can be used to determine how sensitive a design is.

All these experiment tasks utilize the same basic functionality that is defined once:

- **Tuner Parameters:**
  Modelica parameters that remain constant for a particular simulation but are varied by the experimentation environment to search for satisfactory solutions are called "tuners". E.g. via parameter estimation or optimization the tuners shall be determined such that criteria are minimized.

- **Criteria:**
  Criteria are used to compute quantitative values of achieved performance of a simulation run. Criteria are assumed to be positive and smaller values reflect better performance. Usually, several criteria are needed to express the desired behavior. For example, typical criteria of a system step response are over-shoot or settling time. By weighting each criterion individually by a fixed demand value (criterion_i/demand_i), where the demand value expresses the designer's notion of expected system performance, a clear preference list of the criteria is defined. For example, the demand value for the settling time might be 0.1 s, indicating that a value of 0.1 s is satisfactory from a users point of view.

  To solve the multi-criteria problem by means of standard numerical optimization an overall criterion to be minimized has to be formulated. This so called aggregation function is by default the maximum function, yielding a min-max optimization problem over the weighted criteria (= the

largest weighted criterion is minimized). Of course the maximum weighted criterion may change during optimization depending on the tuner values found. Note that an aggregation function value less than 1 implies that all criteria satisfy the demands.

There is an option to choose other aggregation function types like weighted sum. In any case the weights are formed by the reciprocal demand values.

- **Constraints:**
  Design specifications are often given as constraints such as actuator limits. If a certain level of compliance is achieved for a criterion, this level must be kept, and smaller (better) criterion values are not necessary. Constraints are formulated as criteria, which are requested to be smaller than the demand value. Optimization procedures account for constraints in their optimization strategy explicitly.

- **Indicator plots:**
  A criteria value results in one number to express the performance. This is necessary in order that an optimizer can be used. A human would like to evaluate a design by visual comparison of result plots of different designs, e.g., by viewing a whole step response curve and not only the overshoot value. Indicator plots for a model can be defined once and then reused, e.g., for online visualization of the optimization or parameter estimation process.

- **Model Cases:**
  In many applications, several simulation runs are necessary to evaluate a design or to estimate parameters. In the Dymola environment, several simulation runs are collected together to model cases: Exactly the same tuners, criteria and indicator plots are used for each model case. The model cases are distinguished only by a set of fixed Modelica parameters that define the different simulation runs. Often, these case parameters describe different operating conditions, e.g., different road or load conditions of a vehicle.

The paper shows for two application examples how to define the problem setup of tuners, criteria, cases and indicator plots by means of the graphical user interface. The defined problem setups are used to solve the calibration task of an under-actuated two joint Furuta Pendulum and then the parameter optimization for robot control laws. The solution of the multi-criteria optimization problem is discussed briefly in Section 4.

2 Application Examples

In this section, details of the experimentation environment are described elaborately by several examples.

2.1 Parameter Estimation of an under-actuated two joint Furuta Pendulum

Consider the Furuta pendulum demonstration of the Department of Automatic Control, Lund Institute of Technology, Lund, Sweden. The pendulum, consisting of 2 revolute joints and 2 moving bodies, is shown in the figure. Only the first joint (vertical axis) is driven by a DC motor. Experiments and controller have been designed and evaluated in [1].
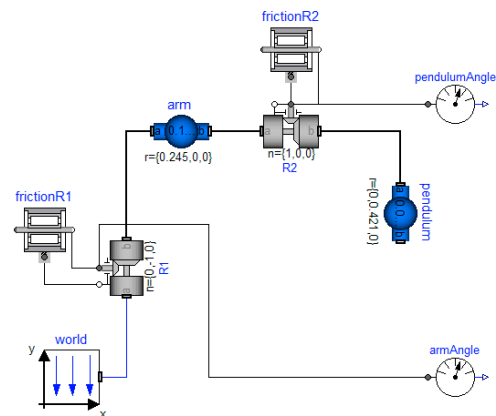
Within Dymola, a model of the Furuta pendulum is easily constructed by dragging, dropping and connecting body and joint components from the Multi-Body library. However, it is also necessary to set physical parameters in the model. Some of these parameters such as the length of the arm or the length of the pendulum are easily measured on the system. Direct measurements of the weights of the parts would require the system to be dismounted (in other cases it is easy to measure the mass or determine it from a CAD system). Moreover, it is not simple to measure the inertia of the parts or the friction characteristics of the two joints.

The new Dymola experimentation environment has been used for parameter identification. For this, the movement of the 2 body pendulum has been recorded by sensors in the joints. The same movements are performed with a simulation model and the friction parameters are optimized such that the measured and the simulated movements closely agree.

Selection of parameters to be tuned

When estimating parameters from measurements, a basic question is "Which parameters can be estimated from the measurements?" Changing a parameter to be estimated must of course influence the output. However, this is not enough if several parameters are to be estimated. Consider the arm of the Furuta pendulum. It rotates around a vertical axis. The model of the arm uses

Modelica.MultiBody.Parts.BodyShape

It has the following parameters: mass, position of center of mass and inertia tensor with respect to center of mass. Since the arm is rotating around a fix axis, it is only the inertia with respect to this axis that influences the behavior of the system. Inertia with respect to an axis being orthogonal to the vertical axis does not influence the motion at all. The criterion is independent of its value. We will discuss a more general case. Let J_c denote the inertia with respect to a vertical axis through the center of mass. Let m denote the mass of the arm and r_c denote the distance between the point of rotation and the center of mass. The inertia of the arm with respect to the point of rotation, J_a, is then

$$J_a = J_c + m \cdot r_c^2$$

It is this inertia that is of importance for the rotation of the arm. It is this parameter that can be estimated from measurements. It is not possible to estimate m, J_c or r_c uniquely from measurements of positions or velocities.

What will happen if we try to estimate J_c and m? In the best case we will get a good estimate of J_a, but J_c and m may be really different from the their real physical values. One approach to investigate if we are about to estimate too many parameters is to make estimation experiments on the model. First one could use the model with its nominal parameter values to produce simulated "measurement data". Then one would set the model parameters that are going to be estimated to other values, and run the calibration procedure using the previously produced "measurement data". If the estimated parameters give the same simulation results but are significantly different from the "true" nominal ones, then this indicates overparametrization, because the nominal behavior can be reproduced by other parameter values than the nominal ones used to produce the "measurement data".

Assume r_c = 0.1225 and being known. Assume the nominal values J_c= 0.0014 and m = 0.165. Assume the criterion to be the integrated squared error of the pendulum angle and arm angle. Let the pendulum start in horizontal position and use the simulation time 5 s. It is illustrative to plot the criterion versus J_c and m as shown below.
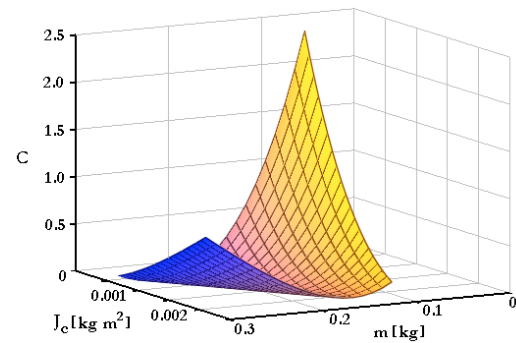
We see a valley where the criterion is unchanged. Along the valley the effective inertia, J_a, is unchanged with peaks on either side where it has been changed. The error is not symmetric with respect to large variations in J_a and the error increases more when J_a decreases.

To compute this map we used a gridding function that takes the calibration task (as described below) and additionally the gridding parameters as inputs, i.e. we do not have to define the calibration task twice.

A further possibility is to investigate the sensitivity of the criterion with respect to the parameters estimated. In particular we can calculate the sensitivity matrix (the Hessian of the minimization problem). Dymola calculates the sensitivity matrix by simulating for disturbed parameters and taking differences of the resulting criterion values obtained. The sensitivity matrix was found to be

[431064, 6456.3;

6456.3, 97.0671],

The eigenvalues being 431161 and 0.37 are very different in magnitudes. Considering the numerical accuracy, the small eigenvalue may be considered as being zero. The eigenvector having the large eigenvalue is {0.99989, 0.01498}. It means a large sensitivity in the direction

{0.99989, 0.01498}*{J_c, m}

Recall J_a = J_c + 0.01501*m which shows that the criterion is sensitive for variations in J_a. The second eigenvector {-0.01498, 0.99989} is orthogonal. (For a symmetric matrix, all eigenvalues are real and eigenvectors are orthogonal.) The small eigenvalue, being very close to zero, indicates that the measured behavior is insensitive to variations in the direction of the second eigenvector as is also shown in plot above.

Since the optimization problem is a non-linear least-squares problem, $\sum f_i^2(p)$, we can alternative com-

pute the insensitive direction (in a more numerically robust way) as the approximate null-space of $\sum \partial f_i(p)/\partial p$.

It may be remarked that for the case where the criterion is independent of a parameter, there will be a valley parallel to the axis representing the parameter and the sensitivity matrix the vector in the direction of the parameter will be an eigenvector with zero eigenvalue.

Thus for the arm we set $r_c = 0$ and interpret the estimated inertia as being J_a. It means that we introduce a top-level parameter I_arm and set arm.r_CM to {0,0,0}, and arm.I_22 to I_arm.

The pendulum consists of a cylinder having a small mass at the end. It is natural to assume its inertia with respect to all axes perpendicular to its length axis to be equal, call it I_pendulum. Set pendulum.I_11 = I_pendulum and set pendulum.I_33 = I_pendulum. The inertia sensed by the rotating arm depends on the angle of the pendulum, which means that for the pendulum, we can estimate also mass (pendulum.m) and position of the center of mass, r_CM_pendulum. We set pendulum.r_CM to {0, r_CM_pendulum, 0}.

Parameters in the friction model for the joints can also be estimated. We assume Coulomb friction with a linear dependence on velocity. For the arm joint we introduce tau11 and tau12 and set frictionR1.tau_pos to [0, tau11; 1, tau12] and similarly for the pendulum joint.

Setting up the calibration task

To set up the calibration task, select CalibrateModel from optimization package. Click the right mouse button. Select Call Function. A dialog is shown:
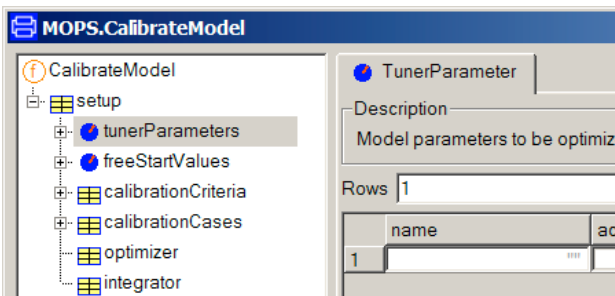

It shows that the calibrate function has one argument, setup. Clicking on the "+" opens it and shows that it is a record with five elements, which describes various aspects of the calibartion task including which parameters to tune, criterion and which measurement data to use. We will discuss the specification of these elements in turn. The calibrate function assumes the current model (last translated

model). This allows easy reference to parameters when selecting tuners as described below.

To specify which parameters to tune, click on the tunerParameters and the right pane of the dialog shows
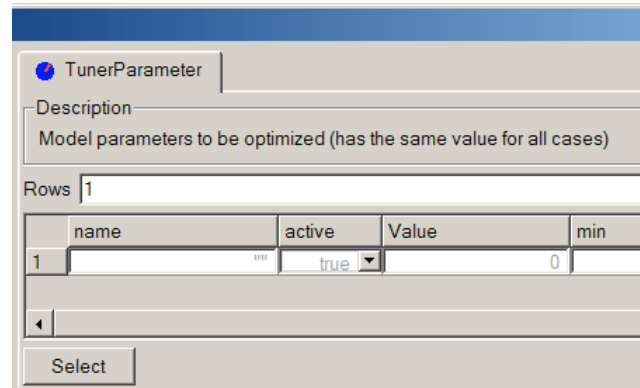
Clicking on Select gives a browser for simple selection where we tick the parameters as decided.
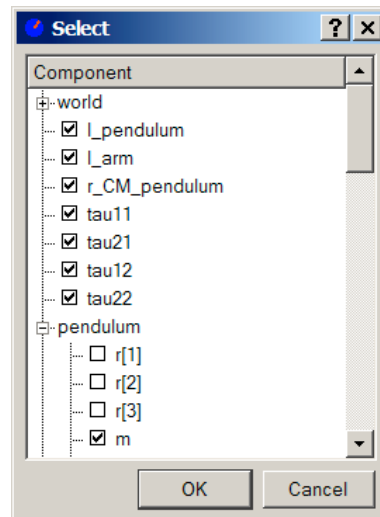
Clicking OK fills the tunerParameter dialog as shown in the following image.
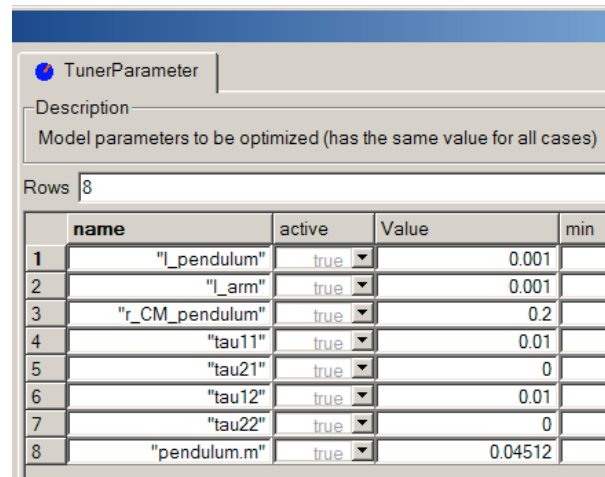
Estimating initial conditions

The initial conditions will be different for different experiments, for example, if we let the pendulum start at different angles. To get good fits it is advisable to estimate the initial conditions. We can give reasonable guesses, because we try to start the system in a well-defined state. For some experiments the initial conditions are known accurately and they should then be given as part of the setup of the calibration cases, but this is not the case here.

The joint models allow specification of initial conditions in terms of parameters. These parameters can be tuned. For each of the parameter discussed previously we would like to have a common tuned value for all cases.

However, for the initial conditions we need individual estimation for each case. Moreover, we would also need them to be tuned for the evaluation where the parameters tuned are kept fixed. The initial conditions to be estimated for each measurement case are specified by the element freeStartValues. Select the element freeStartValues in the tree browser. It is specified in the same way as done for tunerParameters. In the select dialog we tick the start values for the angle and velocity of the two joints.
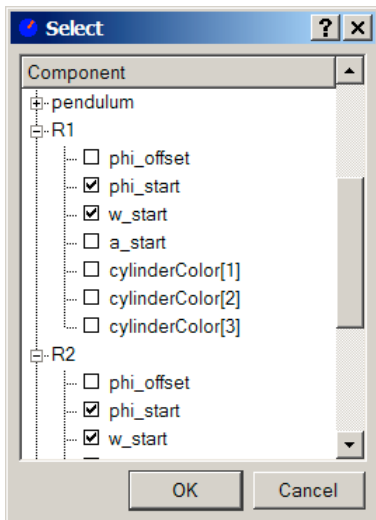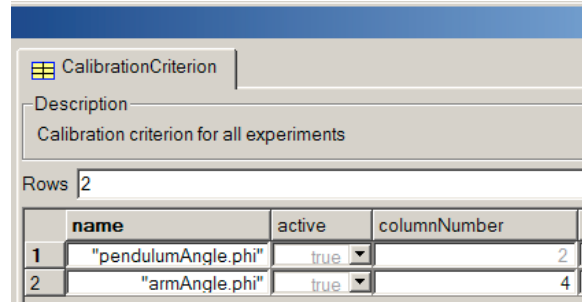
Clicking OK fills in the variable names in the first column and start values in the "Value" column.
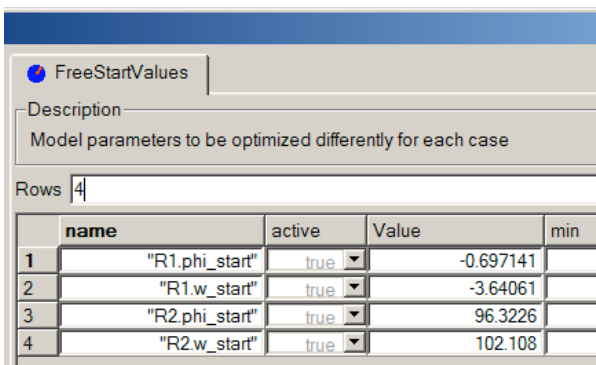
To specify the criterion, select calibrationCriteria. Clicking on Select displays a Select browser where we check pendulumAngle.phi and armAngle.phi. This fills the "name" column.

The criterion is a weighted sum of the integrated square difference with respect to measured value for each variable.

The measurement files to be used are specified by the element calibrationCases. It also specifies whether a file is to be used for calibration or validation.
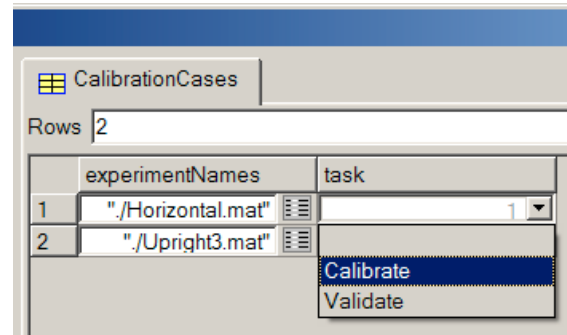
The elements *optimizer* and *integrator* allows advanced setting of optimization and simulation parameters. We use their default settings, except for simulation time that we need to specify.

Running the calibration

When all the input data have been entered, the calibration is started by clicking "Execute" on the dialog. The two next plots compare the simulation result with the obtained tuned parameters and measured data where the pendulum starts in a horizontal
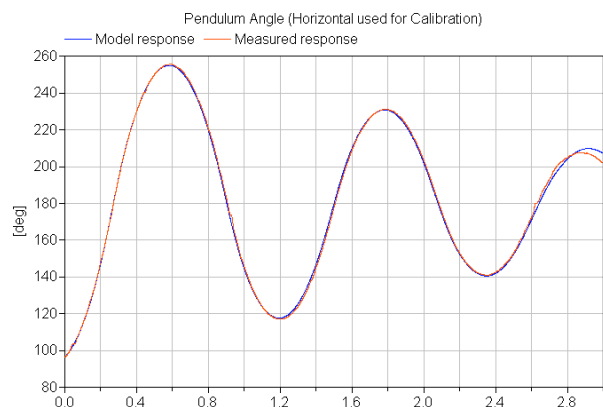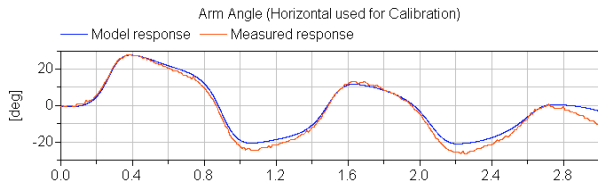
position. The result shows a good agreement. The tuned parameters were validated against another experiment where the pendulum starts in a more upright position. The system is nonlinear and it is of interest to validate the model for cases where the amplitudes of the pendulum are large.
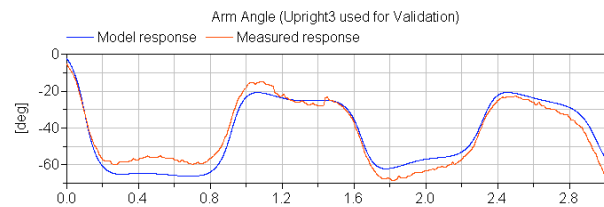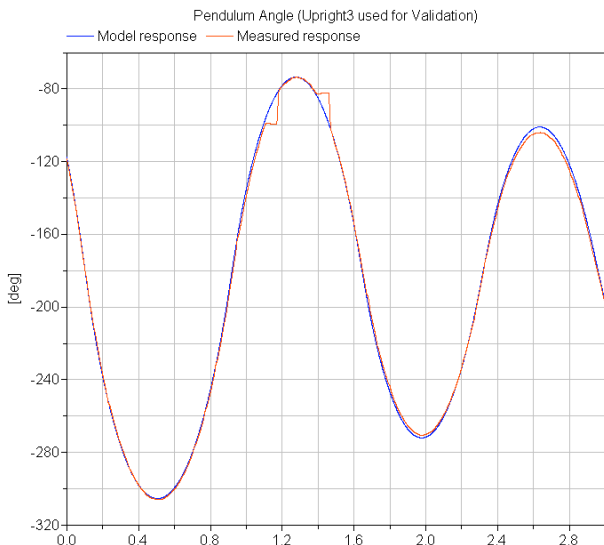
The jump in the pendulum angle in the validation case between –100° and –80° is due to problems of handling wrap-around in the measurement device. The agreement is good, in particular for the pendulum motions. The modeling of the arm may be improved by a more elaborate modeling of the arm friction. However, that is out of the scope of this paper.

2.2   Robot optimization

In this section it is demonstrated how to optimize the controller parameters of a robot for a set of cases consisting of different loads and reference motions.

Description of the robot

The model is chosen as the r3-robot from the Modelica standard library. As described in its documenta-

tion this was originally a Manutec r3-robot. It was then updated with incorporating CAD-data from a KUKA-robot, and the geometry was modified to fit the CAD-data.
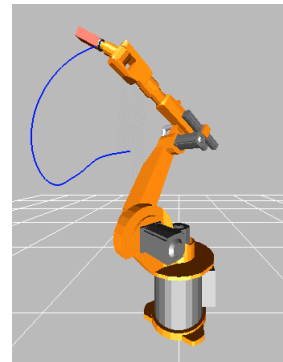

Figure 1 Animation of the robot

The robot consists of a 6 degree-of-freedom mechanical structure modelled as the multibody system "mechanics" (= bodies connected together by revolute joints). The calculation of the animation can be optionally switched off to increase simulation speed, which is especially important during an optimization run. Every joint is driven by a drive train called "axis1", "axis2", ..., in the next figure:
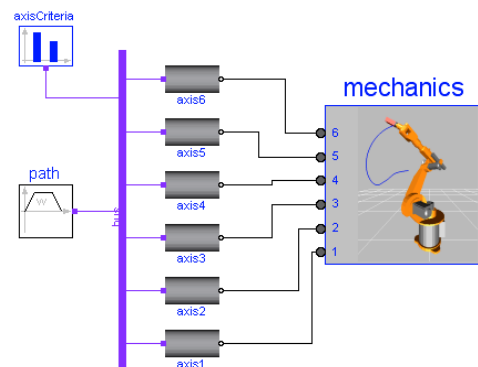

Figure 2 Composition diagram of robot

The desired reference motion is generated in component "path" and as input it has the start and end angle of each axis. All signal data is communicated via a "data bus".
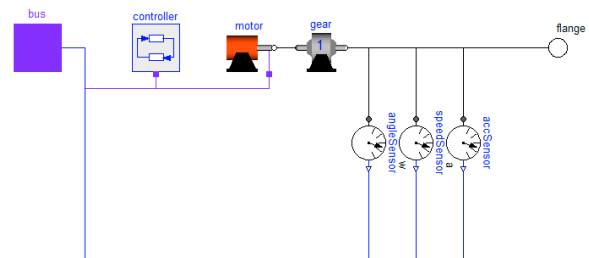


Figure 3 Drive train of one axis

One drive train consists of a controller, an electrical motor, a gearbox (which includes the bearing friction). The controller of one axis is a P-PI cascade controller, i.e., has the simplest form useful for a servo drive (in most industrial robots, more sophisticated controllers are used). The goal of the optimization is to tune the values of the 3 controller parameters of every axis:

- kp – gain of position controller
- ks – gain of speed controller
- Ts – integrator time constant of speed controller

The difficulty is that this controller should work well with a fixed set of values for all paths and operating conditions encountered by the robot. This makes it impossible to use standard, linear control design methods.

Optimization setup

The structure of the optimization setup is similar to the calibration setup:
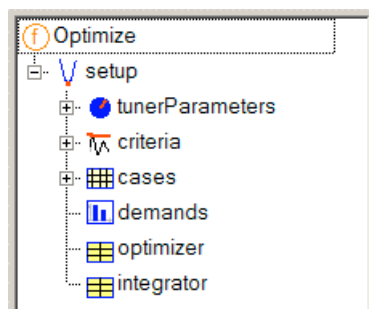
The tuner parameters are the 3 controller parameters of each axis as discussed above. They are easily entered in the tunerParameter dialog by using the Sellect dialog. The snapshot below shows the dialog when the controller parameters of axis 2 have been selected. Numeric values have also been entered for "Value" to be used as the start of the optimization and minimum and maximum values that are used as box constraints during optimization and optionally also for scaling:

In order to specify the different cases of the optimization, the case parameters are first selected by using

the Select dialog to browse the model parameters. The parameter values for the different cases used to optimize the design of the controller are then given by filling out the cases form as below (the selected parameter names appear as column headings):

Cases

Rows 4

	Case	Active	startAngle2	startAngle3	endAngle2	endAngle3	mLoad
1	"High 1"	true	90	0	0	0	15
2	"Low 1"	true	90	-90	0	-90	0
3	"High 2"	true	0	90	30	90	15
4	"Low 2"	true	0	0	30	0	0

Simulation cases are defined by specific settings of model parameters that are given as labels "startAngle2", "startAngle3" etc. in the figure above. Note that in the reference trajectory axis 2 goes through different movements in the different cases, whereas axis 3 is fixed in different positions (all other axes are fixed to the default reference angle in all cases). In practice, robots are optimized for a larger number of cases.

Optimization criteria

If we compare the actual axis speed with the reference speed we normally get the following:
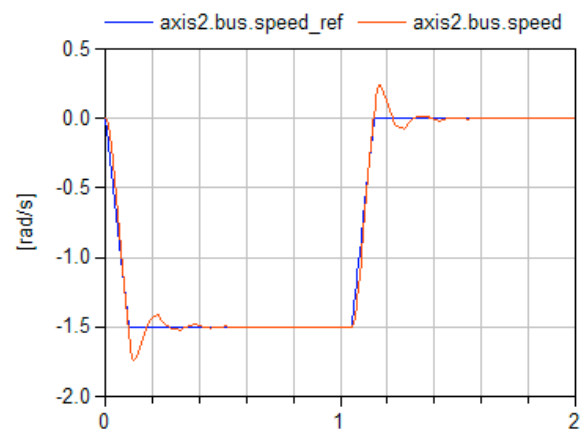
For design optimization the goal is in general *not* to simply minimize these errors, as it would be for a calibration, but something more advanced.

For the design optimization we thus have to introduce additional blocks to measure the performance of the axis, these are introduced by extending the robot-model with an additional performance component for the reference signals (containing different performance indicator blocks).
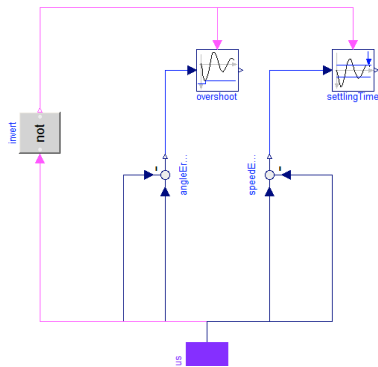
In the performance Modelica model, the following performance indicator blocks are present (these blocks are from a criteria library that contains several predefined useful criteria blocks):

- Block "Overshoot" computes the overshoot (here: the maximum deviation from the reference angle) after the referenced motion has come to a rest. This is performed very precisely by triggering an event whenever the derivative of the input signal is zero (here: whenever the speed is zero), i.e., a minimum or maximum of the signal is reached and storing the corresponding signal, if its absolute value is larger than the previously stored value. The last stored value is the overshoot which shall be minimized.

- Block "SettlingTime" computes the time until a signal stays completely within a tolerance band around zero, after the reference motion is in rest. This is performed by triggering an event whenever the input signal passes through this band and storing the corresponding time instants. The last stored value is the settling time which shall be minimized.

The usage of these two blocks is shown above. As can be seen the angle error is used as input to the "overshoot" block whereas the speed error is used as input in to the "settlingTime" block.

This is connected to the bus to get the reference signals and the actual values for the specific axis. We then select two reference indicators, the overshoot and the settling time, in the optimizer:
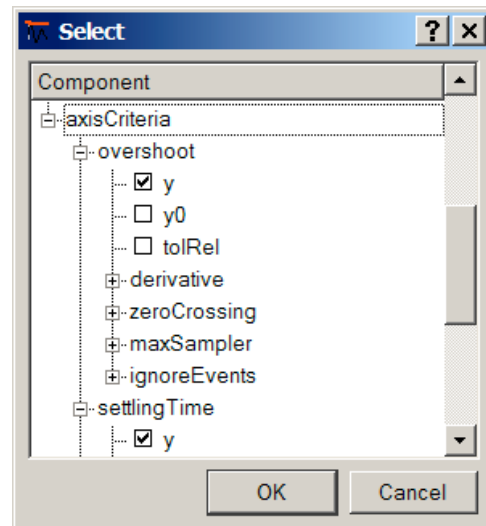
As an alternative to using criteria blocks in the model one can compute the criteria in a post-processing function that operates on the simulation results and which is selected in the drop down menu shown below. In this case we select the final value for these two criteria:
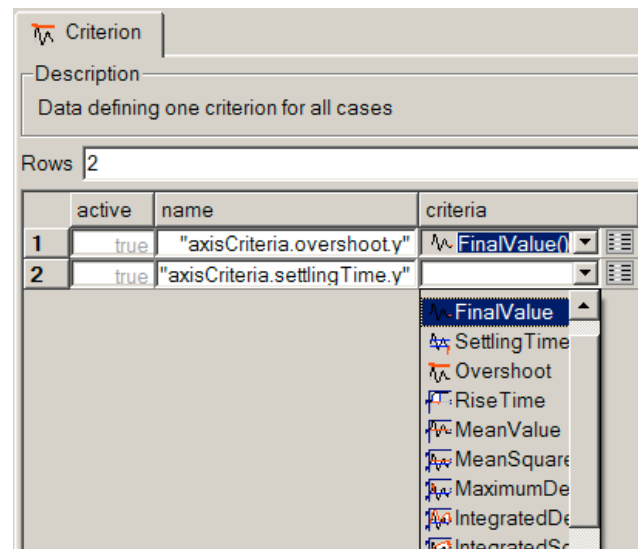
We also have to set the demand values. Based on first simulations and specifications we set the overshoot demand to 3/1000 [rad/s] and the settling time to 0.3 [s]:

	axisCriteria.overshoot.y	axisCriteria.settlingTime.y
High 1	0.003	0.3
Low 1	0.003	0.3
High 2	0.003	0.3
Low 2	0.003	0.3

Applying these values as demand values results in the following weighted criteria:

case	overshoot	settling time
High 1	0.6854	0.7588
Low 1	1.5106	0.9855
High 2	0.8332	0.5579
Low 2	1.7887	1.0468

The values indicate satisfactory behavior in cases with load (cases "High 1" and "High 2"), but bad overshoot performance in unloaded cases (cases "Low 1" and "Low 2").

After finalizing the optimization setup, a click on "Execute" starts the design optimization. As a result of the first optimization run, which converges after 30 function evaluations, we obtain the following tuner and corresponding weighted criteria values:

kp2	3.0776
ks2	2.5089
Ts2	0.08896

	overshoot	settling time
High 1	0.5929	0.8998
Low 1	1.0315	1.0902
High 2	0.6510	0.7157
Low 2	1.1956	1.1955

Overshoot has been improved but settling time is slower. The equal and largest criteria values in case "Low 2" indicate a conflict between the 2 criteria which usually can only be solved when one criterion is eased off.

We decide to force the overshoot criteria that they reach their demand values and to put lower emphasis on settling time. This is accomplished by applying all overshoot criteria as *inequality constraints* during the next optimization run forcing the optimizer to perform improvements in these criteria until the demand value is reached. Criteria to be minimized (here settling time) may increase. The next run results in:

kp2	4.0722
ks2	4.8897
Ts2	0.070565

	overshoot	settling time
High 1	0.7623	0.8965
Low 1	0.8845	1.1855
High 2	0.6175	0.7161
Low 2	0.9985	1.2027

The overshoot demand is satisfied, The settling time slightly increases to 1.2027 for the worst case "Low 2". We might stop the design optimization here. In other cases, one might change demands, select other simulations cases and criteria. In the next figure, results for the 3 runs (initial, first and second optimization run) are shown for case "Low 2":
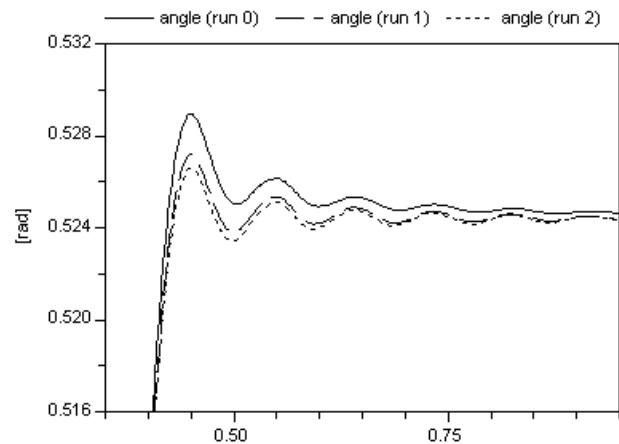
3   Customizable user interfaces

Experimentation includes operations that require rich interfaces to supply all the information needed to perform the task.

For model components, parameters have been visually split into groups and tabbed pages, representing logical grouping of primary and secondary parameters. However, the individual data items comprise a relatively "flat" structure.

For calibration and optimization the interface contains a much deeper hierarchical structure, and the complexity at each level is also greater. For example, it is common that subitems contain variable amount of data, typically represented by arrays of records.

To handle the increased complexity, the graphical user interface of Dymola has been extended in two dimensions:

- The nested structure is visualized by a tree, which makes relationships easier to understand and allows easy navigation between data items.

- Specialized GUI elements, for example, for file and color selection, can be enabled by annotations, which facilitate common input operations.

Several of these improvements are useful also for simpler data structures, and the specialized GUI elements can also be used for parameters of models. In

other words, all the features discussed below can be easily utilized by every user. It is even possible to make a copy of the calibration and/or optimization setup and adapt the user interface to the specific needs of an end-user with more specialized menus.

3.1 Nested structures

Model calibration will be used as an example. The function CalibrateModel takes a record (setup) as its input parameter. The structure of the record contains among other elements an array of type Tuner-Parameter, which in turn contains several attributes.

```
function CalibrateModel
  "Calibrate model to measured data"
  input ModelCalibrationSetup setup;
  ...
end CalibrateModel;

record ModelCalibrationSetup
  String Model;
  TunerParameter tunerParameters[:];
  ...
  Optimizer optimizer;
  Integrator integrator;
end ModelCalibrationSetup;

record TunerParameter
  "Model parameter to be optimized ..."
  String name="" "Full name of ...";
  Boolean active=true "true, if ...";
  ...
end TunerParameter;
```

The GUI is automatically built from the data structure declarations. The nested structure of the input to CalibrateModel is evident in the tree view at the left.
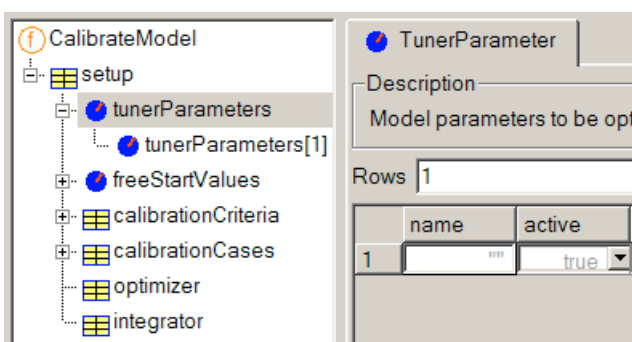

The tree serves two puposes. First it shows the structure and makes it easier to understand what information must be provided. The tree view corresponds exactly to the data structure. Second, it is used to navigate between multiple "pages" (input forms) that are swapped into the space at the right.

The component tunerParameters is an array of records, each containing several variables. The user can choose a combined tabular view of the array (as

shown above), which offers maximum overview in a compact format.

Alternatively it is possible to inspect and edit individual array elements, which has the advantage of displaying descriptions for each input field and that data can be grouped and put under different tabs. Each page corresponds to one row in the combined view.
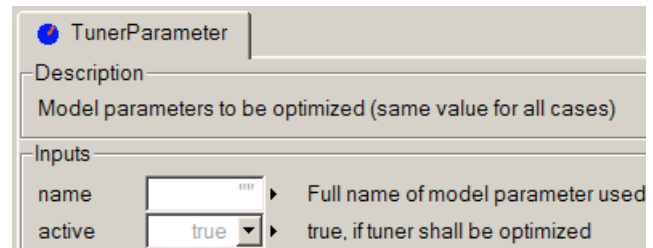
The implementation of the tree view also ensures that data filled out by the user is propagated. For example, changes to an individual tuner parameter must be visible when the user switches to the tabular view of all tuner parameters. Changes in a modifier at a high level is propagated down to more detailed views.

3.2 New GUI elements

The graphical user interface can greatly simplify certain input tasks with some additional support. Although it is always difficult to strike a balance between features and complexity, the following operations have been found useful in the experimentation environment.

The deployment of these GUI elements is controlled by model annotations, either at the class level or on individual variables.

Predefined choices. A list of values suitable to a particular type or application are presented. A simple example is "true" and "false" for Boolean. Selection of a criteria function is specified by this annotation:

```
CriterionSpecification criteria
  "Criteria specification"
    annotation (choices(
      choice=FinalValue(),
      choice=SettlingTime(),
      choice=Overshoot()));
```

To the user the choices are presented in a drop-down combobox:
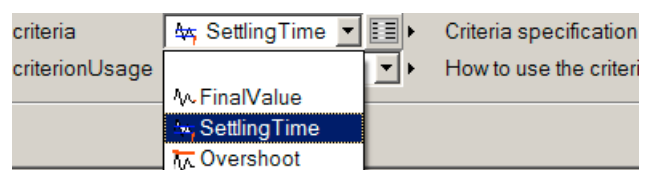
Because parameters which depend on the selected function must also be specified, pressing the "edit" button will display a dialog for the parameters to the chosen function.

Color selection. Colors can typically be represented by RGB (red, green, blue) or HSV (hue, saturation, value) tuples. Although they could be specified numerically by the user, a colorful dialog makes selection much easier.

File selection. Standard dialogs for selecting files either for reading or writing data. The filename is stored in the corresponding variable.

Variable selection. Several operations in the experimentation environment involves the selection of variables from the model, for example, parameters to optimize. A specialized selection dialog simplifies the task considerably. Furthermore, additional data, such as, start/min/max values can be extracted.

In this case a detailed specification (in the form of an annotation) is needed to move data into the right elements of a table, and if needed resize the table.

User-defined labels. The default labels used in the tree view or in the combined tabular view are constructed from variable names found in the data structure. By use of annotations, other labels can be specified or even extracted from actual data in the structure.

4 Solving the Multi-Criteria Optimization problem

In a multi-criteria optimisation problem setup all weighted criteria $q_{ij} = c_{ij}/d_{ij}$, $ij \in S_{min}$ can be combined to a vector $\mathbf{q}$, where S_{min} denotes the set of all criteria (i) to be minimised defined in all simulation cases (j). In order to decide whether a solution $\mathbf{q}^I$ is better than a solution $\mathbf{q}^{II}$, these vectors should be completely comparable. However, comparing each vector component individually, some components can be better, others can be worse. To make criteria vectors completely comparable a vector norm has to be introduced.

We prefer to use the max-norm, because weighted criteria with positive 'the smaller the better' values and quality limiting demands as upper bounds yield a most visible comparative satisfaction assessment of design alternatives in case of the max-norm. Define for all weighted criteria

$$\alpha := \max_{ij}\{q_{ij}\}, \quad ij \in S_{min},$$

then requirements' satisfaction of a design alternative (*II*) is said to be *better* than of a design alternative (*I*) if $\alpha^{(II)} < \alpha^{(I)}$. If $\alpha \leq 1$, the design alternative is called a satisfactory solution, because in that case each criterion is less than the respective demand value. In particular, a *best possible* design alternative is characterised by $\alpha^* = \min\{\alpha\}$, yielding the overall constraint optimization problem which can be solved by standard optimization methods:

$$\min_{T} \max_{ij \in S_{min}}\{c_{ij}(T)/d_{ij}\}$$
$$c_{ij}(T) \leq d_{ij}, \qquad ij \in S_{inequality}$$
$$c_{ij}(T) = d_{ij}, \qquad ij \in S_{equality} \qquad (1)$$
$$T_{min,k} \leq T_k \leq T_{max,k}$$

The disadvantage of this approach is the lack of differentiability of the aggregation function. Thus, methods relying on gradients (like SQP methods) can encounter difficulties in this case.

To overcome the problem of differentiability we provide two mechanisms: an exponential approximation of the max-function yielding a smooth overall criteria and a reformulation by '*equivalent constraints*'. In the latter case the min-max problem can be reformulated by an equivalent constrained problem. Let γ be a new variable for which we impose that $\gamma \geq \max\{q_{ij}(T), ij \in S_{min}\}$. Instead of (1), we can solve an equivalent optimization problem with the extended parameter vector $x = [T, \gamma]$. The aggregation function to be minimized is simply $\alpha(x) = \gamma$.

Defined inequality and equality constraints are applied as in (1) while the components to be minimized are added as additional constraints as

$$q_{ij}(x) \leq \gamma, \qquad ij \in S_{min}$$

The main advantage of this formulation is that the functions are differentiable provided the defined problem criteria are differentiable. The disadvantage is that a problem of higher dimension is solved and additional constraints are added. However, the application of this formulation of the min-max problem is recommended whenever a gradient based optimization method is used. There is an option to choose other aggregation functions like weighted sum:

$$\bar{\alpha} := \sum_{ij}|q_{ij}|, \quad ij \in S_{min}$$

Optimization Methods

At the moment 5 different methods to solve the optimization problem (1) are implemented for use with Modelica:

1. Sequential quadratic programming (SQP)
2. Quasi Newton (Bounds)
3. Pattern search (Pattern)
4. Simplex method (Simplex)
5. Genetic algorithm (GA)

The *Sequential Quadratic Programming* (SQP) approach can be used to solve the general optimization problem and has usually a super-linear convergence (= faster than linear, and slower than quadratic convergence). Bounds on tuners and linear equality and inequality constraints are met exactly during the iterations. The SQP approach needs gradients of functions and constraints. SQP in combination with the reformulation of the min-max optimization problem as an equivalent constraint problem is the method of choice for general optimization or calibration problems.

The *Quasi Newton method* (Bounds) is an algorithm intended to solve large optimization problems but can only handle simple bounds constraints on the tuners. "Bounds" needs also gradient information of the aggregation function.

The *Pattern Search* approach is a derivative free search method. It is numerically more robust in tackling with non-smooth criteria than other methods.

The *Simplex* approach is also a derivative-free algorithm and employs linear approximations to the objective and constraint functions. The main advantage of SIMPLEX over many of its competitors is that it treats each constraint individually when calculating a change to the variables, instead of lumping the constraints together into a single penalty function. A drawback of this method is that even bound constraints can be violated during computation.

The *genetic algorithm* (GA) is a global optimization technique. The basic algorithm allows only simple bounds on the variables. Thus, to address more general constraints, penalty function techniques are employed. The genetic algorithm search method is based on evolution principles which guarantee the survival of the fittest individual. The use of GA for optimization is normally quite costly in terms of function evaluations.

5 Conclusions

An environment was presented to optimize Modelica models in Dymola, especially with regards to design optimizations and calibration of unknown model parameters.

It is possible to prepare a customized GUI in Dymola for specific tasks such as optimization. This makes it possible to customize the optimization menus. Since the customization is performed in Modelica (annotations) it is possible for an end-user to also adapt this to his/her particular needs.

Acknowledgements

Measurements and calibration of the Furuta pendulum was performed by Marco Bracci as a part of his master-thesis project at the Department of Automatic Control, Lund Institute of Technology, Lund, Sweden.

References

[1] Åkesson J. (2000): **Safe Manual Control of Unstable Systems**. Master Thesis, ISRN LUTFD2/TFRT--5646—SE, Department of Automatic Control, Lund Institute of Technology, Lund, Sweden.

[2] Dynasim (2005): **Dymola - Users' Manual**

[3] Joos H.-D., Bals J., Looye G., Schnepper K., Varga A. (2002): **A multi-objective optimisation based software environment for control system design**. Proc. IEEE International Conference on Control Applications, Glasgow, Scotland, Sept. 18-20, pp. 7-14.