ISSN 0280-5316
ISRN LUTFD2/TFRT--5757--SE

Greybox Identification and Control Design with Dymola

Marco Bracci

Department of Automatic Control
Lund University
September 2005

Department of Automatic Control
Lund Institute of Technology
Box 118
SE-221 00 Lund Sweden

Document name
MASTER THESIS

Date of issue
September 2005

Document Number
ISRNLUTFD2/TFRT--5757--SE

Author(s)
Marco Bracci

Supervisor
Sven Erik Mattsson at Dynasim in Lund
Anders Robertsson and Karl-Erik Årzén at Automatic Control in Lund

Sponsoring organization

Title and subtitle
Greybox Identification and Control Design with Dymola (Gråboxidentifiering och Reglerdesign med Dymola)

Abstract
This Master Thesis has been done at the Department of Automatic Control, Lund Institute of Technology, and Dynasim AB, in Lund. Dynasim develops the simulation software called Dymola (Dynamic Modeling Laboratory). With the upcoming release 6, they have introduced the opportunity of interfacing their software with an optimization tool called MOPS (Multi-Objective Parameter Synthesis). This work shows how the application of this new feature to Greybox Identification and to the design of parameterized controllers subjected to some constraints on their performance, can provide excellent results. The procedures requested to perform identifications or control designs are carefully explained, step by step, with examples.

Keywords

Classification system and/or index terms (if any)

Supplementary bibliographical information

ISSN and key title
0280-5316

ISBN

Language
English

Number of pages
95

Recipient's notes

Security classification

Acknowledgments

I would like to begin saying that I've spent, in Sweden, one of the most beautiful and interesting periods of my life. So I thank Sweden and its people to have received me with their kindness in a marvellous place. I will miss Lund a lot...

Then I would like to thank the people which have helped me there. I will start with all the people at Dynasim AB, the company where I've spent the most of my studying time. I will never forget Sven-Erik (who helped me a lot in every kind of matters with my work), Hilding (the boss who helped me a lot, as well), Hans, Dag, José (with whom I spoke in Italian), Per, Jonas, Mats, Hubertus and Jessica. They have been very kind to me.

After that, I would like to thank all the people at Control Department of LTH for their courtesy and support. A special place in my memories belongs to Dr. Anders Robertsson, for the continuous help and encouragement he gave me in every situations. I would like to thank, then, Professor Rolf Johansson for his kind welcome at LTH and for his precise supervising to my work.

Special thanks go to my Italian supervisors Professor Edoardo Mosca and Ing. David Angeli which helped me a lot with the last matters of my work when I came back to Italy.

I would like to thank, then, my old friends and the new friends I've met in Sweden with which I've enjoyed my time there and I've learned a lot of new things (*"I learn, I learn, I learn"* was my motto). Special thanks go to my friend, flatmate, guru, and "stylist" Lorenzo and to my "little sister" Clara: they encouraged me very much and we were a very nice group of friends in Lund.

Finally, I would like to show my gratitude to my family. I've been able to live this very interesting and beautiful experience only thanks to their continuous support and efforts.

Contents

Chapter 1

Introduction

The Master Thesis that we are going to present here is the result of six months of studies in Lund, Sweden. This work has been done at the *Department of Automatic Control*, *Lund Institute of Technology* (Lunds Tekniska Högskola, LTH) and *Dynasim AB*, the company that has developed *Dymola* (Dynamic Modeling Laboratory), the simulation software that we have used all through this project.

The subjects of this work are *Modeling* and *Identification*. A good description of the meaning of modeling and identification is given in [18] and it is briefly reported subsequently.
Identification is the process of constructing a mathematical model of a dynamical system from observations and prior knowledge.
A model can be used in a lot of different fields of science and technology: for example scientific modeling helps to increase understanding of some mechanism by finding the connections between observations relating to it. Modeling is also used for Diagnosis of Faults and Inadequacies because of the eventuality, using it, of discovering shortcomings and anomalies; another fields of application are Simulation and Operator Training because it is generally more secure and, above all, less expensive exploiting a simulator in place of a real device; then it is often impossible, for diverse reasons, to have direct access to the real system: the system could be too delicate (and performing experiments on it could be too hazardous or expensive), not reachable (too far, too small, too dangerous, etc...) or even it could be also possible that the system does not

just exist, yet: this is a common case in the design of prototypes where some things must be tried before the actual construction of the object.
A curious and interesting fact that could happen during the study of models is, without any doubts, *Serendipity* which is that phenomenon for which one stumbles across something when looking for something entirely else: this often happens in astronomy, where realizing mismatches between the behaviour of the model and the behaviour of the real system, often makes it possible to make discoveries.
However, above all, the important fact is that Modeling and Identification are fundamental in our field of studies that is *Automation*. They are used for *Prediction* that helps in feed-forward control[1], where a disturbance is detected early in its progress through the system and is fed forward, suitably shaped and inverted, to cancel its effects further on; prediction is required also in self-tuning control which recalculates the control input to the system periodically by reference to a periodically updated prediction model of the effect of that input. Another important role of modeling in automation is *State Estimation* which is tracking variables which characterise some dynamical behaviour, by processing observations afflicted by errors, wholly or partly random.

The main issue of modeling is, then, to build a model that behaves, as much as it is possible, like the real system. Generally we have not a complete knowledge of the system because it is often hard knowing all the physical relations that the system has inside. Also having a deep cognition of it, that is a cognition of all the main features, it is impossible to have a complete knowledge of the "minor characteristics" such as frictions or less important physical relations between the internal components, etc. So we can have different approaches to the matter of modeling, depending a lot on our knowledge of the system: with a great cognition we could directly write the physical equations, otherwise we could try to think to a possible structure of the system and build some possible transfer functions. However it does not happen so frequently to have all the required parameters of these relations, so we must find them in some way. Here the theory of *Parametric Identification* gives help to us with all its methods. This subject is divided in two different branches: *Blackbox Identification* and *Greybox Identification*.

[1] also known as open-loop control

Blackbox Identification is applied to systems with a complex unknown inner structure: for this kind of systems it is difficult to describe the relations between their internal parts. In this situation it is unproductive to study these parts and, instead, we consider it as a closed box[2] and we think about a probable *a priori* structure which we must find the parameters of.

For systems with a simple or a well known internal structure, instead, it is utilized the Greybox Identification[3]: with this approach a so-called *a priori* information about the system and the presence of random disturbances are needed. The procedure begins in developing the models of the single system components (whose equations are known or have been obtained studying them as they were Blackbox systems) and the relations between them (these are the above a priori information), then we try to find the values of the parameters that we do not know through *Calibration*: it is for this task that disturbances are important; actually their presences makes the calibration and validation tasks different from a simple comparisons of the responses of model and system to known inputs (see [2]).

In this thesis we use the *Furuta Pendulum* as a case study. For this process we have a lot of *a priori* information and physical modeling can be used to capture the dominating nonlinear dynamics. Greybox Identification has been preferred to the Blackbox one: we have the a priori information, that is our knowledge of the device components (bars and joints) and of the relations between them; then we have also the required disturbances, that are frictions, the only unknown parts of the system.

What we have done in this work is therefore, performing a *Greybox Identification* through a modeling software called Dymola. The usage of this software is recommended for the opportunity of modeling by means of *Modelica*, a object-oriented modeling language, and of its very convenient *MultiBody Library* that permits to design models via a graphical user interface. Identification, then, has been made possible by the new upcoming feature of the software, that is the opportunity of interfacing itself with the optimization tool *MOPS* (Multi-Objective Parameter Synthesis). This has been used to min-

[2]the name comes just from the fact that in a black box one cannot see anything inside

[3]grey, in this case, means opaque: we have a box which we can see almost all in, except for some "little" things

imize the difference between the behaviours of the analyzed system and our model.

Moreover, it has been shown how the same tool can be used for the key task of Automation, that is *Control Design*: this use is possible because of the similarities between Greybox Identification and the design of a parameterized control which we already know the structure of.

This thesis has the following outline: Chapter 2 introduces the softwares that have been used for the identification, Chapter 3 regards the theory behind this work and identification in general, Chapter 4 shows our case study and Chapter 5 has examples about the usage of optimization in Control Design. Finally, in the last chapter, there are comments and conclusions about this work.

Chapter 2

Software

In this chapter we are going to make a brief introduction to the computer tools which have been used for this work.

Modelica is an object-oriented modeling language that permits to model complex systems with a reasonable easiness and velocity. The power of this language is exploited through the software *Dymola*, a tool that first makes the modeling even easier through the possibility of using Drag & Drop and then it offers a complete environment for simulation and analysis of the system.

MOPS is a powerful optimization software which now has the possibility to be interfaced to Dymola: this feature permits to this simulation software to be used also for Model Calibration and Design Optimization.

2.1 Modelica

Modelica is a freely available *object-oriented modeling language* for large, complex and heterogeneous physical systems. Its design was started in 1996 by Hilding Elmqvist and a lot of developers have joined in this effort during the years. Nowadays Modelica is developed and promoted by the *Modelica Association* [4], a non-profit, non-governmental organization that has its seat in Linköping, Sweden.
Tools and environments have been presented to make easier the use of this language and to capitalize its numerous capabilities; Dymola (see Chapter 2.1.2), the software used in this thesis, is one of those.

2.1.1 Overview

Modelica has a lot of interesting characteristics that suggest the utilization of it.

One of the most important features of Modelica is the fact that this language, differently from others, describes physical systems through equations and not through algorithms. This feature is very convenient since it gives the language a high level of abstraction, that means a more comprehensible and easier way of using it: the user has only to write the equations characterizing the system in their differential, algebraical or discrete forms, then there are some tools which translate these equations into very efficient code; Dymola is one of those.
An example, taken from [12], of what we have just said can be seen here below, where a model of a simple LC-oscillator written in Modelica language is shown:

```
model LC_oscillator
  parameter SI.Capacitance C = 1;
  parameter SI.Inductance L = 1;
  SI.Voltage u;
  SI.Current i;
equation
  L*der(i) = u;
  C*der(u) = -i;
end LC_oscillator;
```

The model declaration starts with the keyword `model` and the name of the model, and ends with the keyword `end` and the name. Subsequently parameters (using the keyword `parameter`) and variables[1] are declared. At the end, after the keyword `equation` we find all the equations that describe the system.

The major thing, however, is *Reusability*: Modelica is a an object-oriented language which implies that a model is defined as a class, and so it can be extended or just used, as an object, together with others to design a larger

[1] using the keyword `parameter` or `constant` a variable cannot be modified during the simulation

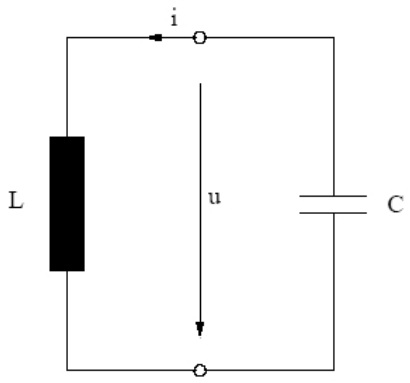


Figure 2.1: LC Oscillator

one. This is a very interesting and convenient fact because it means that, in designing, we can use works done by others, without having the need to restart everything from the beginning. For example, the model seen before could be described in this new way:

```
model LC_oscillator
  capacitor C1;
  inductor L1;
equation
  connect(C1.p, L1.n);
  connect(L1.p, C1.n);
end LC_oscillator;
```

We can see that now our oscillator is composed by two other models connected together: the one of the capacitor and the one of an inductor. The connection is created in the equation section where the function `connect()` generates the equations at the component pins.
The differential equations of the previous model are now part of the models of the two components:

```
model capacitor
  SI.Voltage u;
  SI.Current i;
  parameter SI.Capacitance C = 1;
```

```
  pin p;
  pin n;
equation
  C*der(u) = i;
  u = p.u - n.u;
  0 = p.i + n.i;
  i = p.i;
end capacitor;
```

In this model there are two variables declared of `pin` type. `pin` is not a model here, but it is defined as a *connector*, which is an entity that contains all the quantities needed to describe connections between components: connectors are connection interfaces between models. The declaration of a connector starts with the keyword `connector` and ends with the keyword `end` both of them followed by the name of the connector.

```
connector pin
  SI.Voltage u;
  flow SI.Current i;
end pin;
```

Being an object-oriented language, Modelica has also the feature of *Inheritance* that is the feature of extending the properties of a class. This feature is very convenient because it enhances the possibilities of reusability and it makes code thinner and then easier to understand and to maintain.

All these elements can be found in the *Modelica Standard Library*, a package containing constants, types, connectors, partial models and model components for various subject matters.

Another significant library, that has been used for this thesis, is the *MultiBody Library*. In [5] it is explained that this library helps the modeling of multi body systems, that are systems of rigid bodies connected to each other with certain degrees of freedom. A multi body system model consists of an inertial system, joints, massless bars and bodies with mass, connected together with the Modelica connect statement (see Figure 2.2). Every model must have exactly one instance of the class Inertial. This object defines the global coordinate system and gravitational force. All other objects are in some way

connected to the inertia system, either directly or through other objects. In order to build a rigid body with several joints connected to other bodies at different points, massless bars are used to put distance between the connection points. A body object is then connected at some point, defining the mass properties of the rigid body (see Figure 2.3).

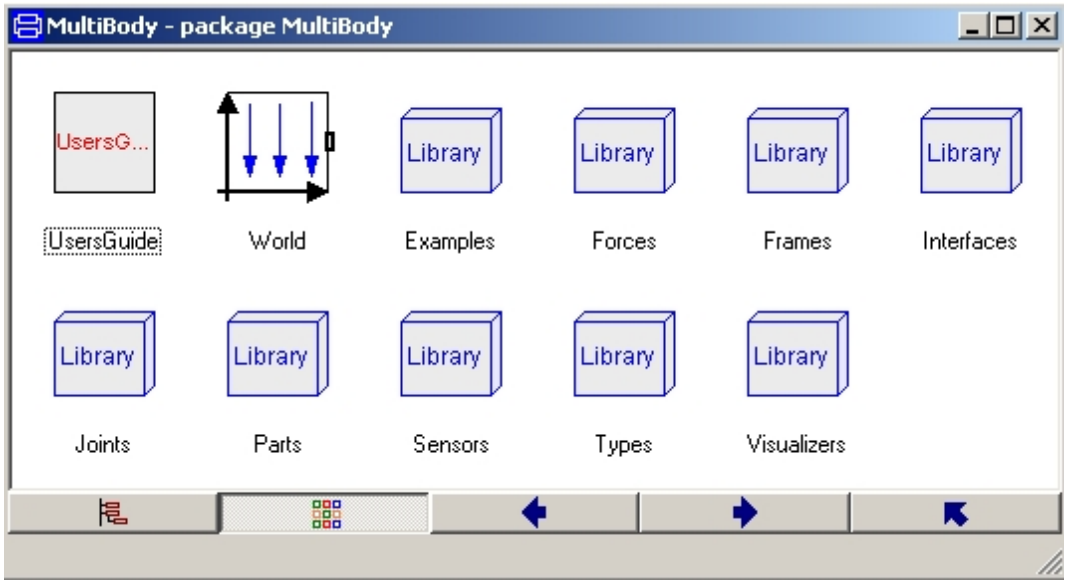


Figure 2.2: MultiBody Library

2.1.2 Dymola

Dymola is the *Simulation Environment* for Modelica utilized in this work. Modelica needs a translator to transform the model, that is the equations which describe it, into a form that can be efficiently simulated, that is a form which can be integrated with standard methods. Such translators are really complicated, so they require a lot of knowledge and implementation work.

Dymola is developed by Dynasim whose CEO, Dr Hilding Elmqvist, started the design of Modelica in 1996.

The tool contains a Modelica translator that can handle real time applications and systems with more then one hundred thousands equations.
It comes with a graphical editor that allows modeling just with the use of

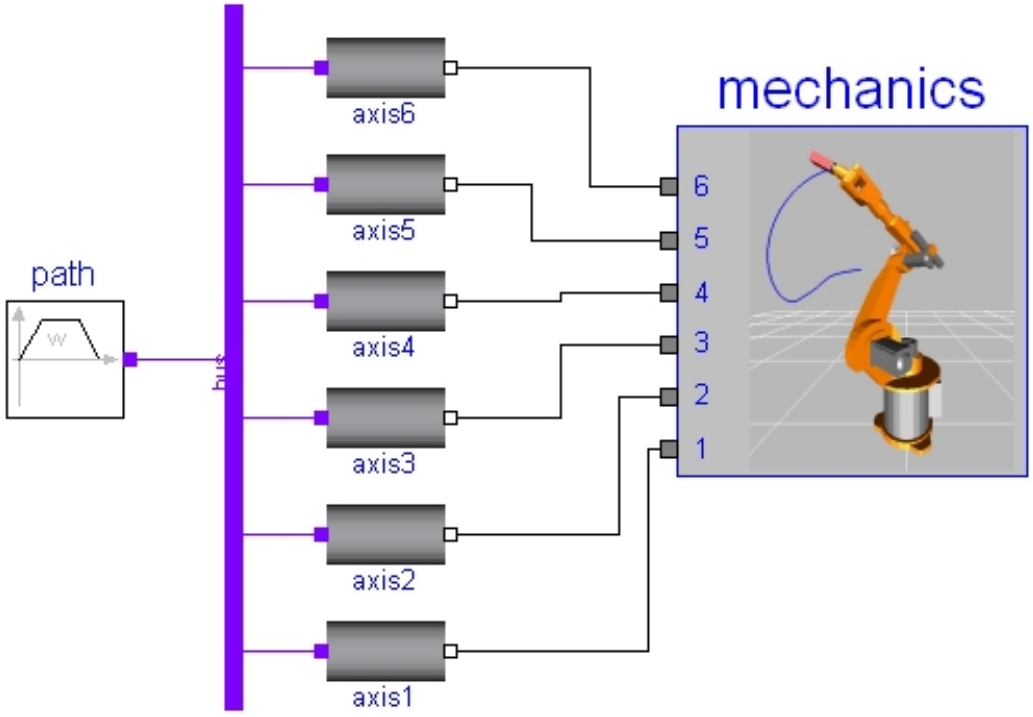
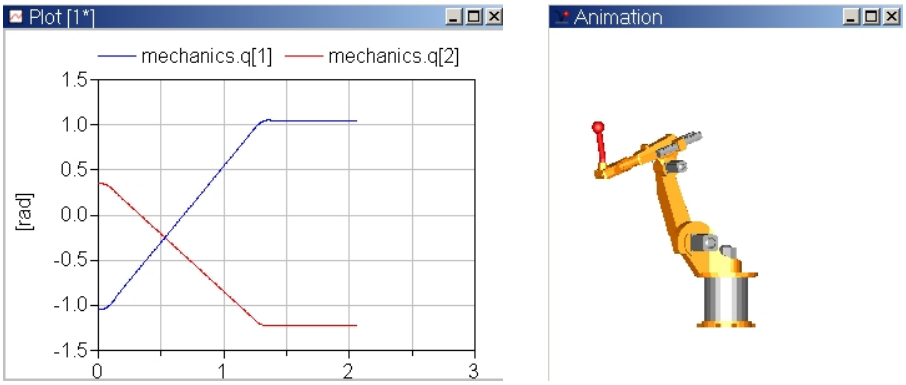


Figure 2.3: Example of a MultiBody Project (Manutec r3 robot)

Drag & Drop. This is a very convenient feature that permits to design models without any knowledge of the Modelica language.

Then, and this is the most important part, there is a simulation environment that has the task to translate the equations into efficient code and then to resolve them. This part manages also the visualization of the dynamic behaviours of the system, that is it deals with the plots of the time-response of variables (see Figure 2.4(a)). Another interesting feature, likewise linked to this environment, is the opportunity of viewing a *3D animation* of the system moving: this is possible because when we design a model, we can also add attributes regarding the shape and the colors of the utilized components (see Figure 2.4(b)).

It is also very interesting and useful with the possibility to link Dymola to Matlab and Simulink through simple interfaces. That is very convenient since we have the opportunity of using all the analysis and design capabilities that Matlab offers.



(a) Plots regarding Manutec r3 robot

(b) Animation of Manutec r3 robot

Figure 2.4: Analysis of Manutec r3 robot

2.2 MOPS

MOPS [6] is an optimization software developed at DLR (Deutsches Zentrum für Luft- und Raumfahrt) which has been in use and applied to challenging industrial problems since years. MOPS is an acronym that stands for *Multi-Objective Parameter Synthesis*.
It supports the design engineer in setting up hisher design problem as a properly formulated multi-objective optimization task. To this end, MOPS offers a basic control system criteria library, a generic multi-model structure for multidisciplinary problems and a generic multi-case structure for robust control law design, as well as visualization tools for monitoring the design progress.
MOPS also supports, and this is the feature mainly utilized in this work, parameter estimation in identification problems.

The first step in an optimization is the definition of the criteria: *MOPS criteria library* provides a complete set of basic functions for the most commonly used time and frequency domain criteria: this library may serve as a basis to define more application-oriented design criteria.

Since realistic control law design is a multidisciplinary task, it involves the simultaneous minimization of many design criteria in the presence of various constraints. Typically, the different criteria and constraints are evaluated using computational models developed for different engineering disciplines or resulting from different modelling formalisms. MOPS explicitly supports the

usage of different models from multiple disciplines (multi-models) to evaluate the design criteria. To each analysis model (e.g. non-linear simulation model, frequency domain models, etc.) a set of criteria is associated.

The optimization problem, that is the multi-objective/multi-model design problem, is then mapped to a weighted min-max optimization problem, which is solved in MOPS by using one of several available powerful optimizers, implementing local and global search strategies.

Chapter 3

Theory behind this work

In this chapter we are going to show some useful results that the theory about Identification has reached.

In the first section, the one about *Persistent Excitation*, we will show some results that are really useful when we are starting to prepare an identification experiment: we will discuss, doing some examples, about the most suitable input signals to give to our system for a correct parameter identification. In fact Persistent Excitation condition is sufficient to obtain consistent estimates for the least-squares method and maximum-likelihood identification. An accurate explanation of this topic can be found in [11].

The second section is about the concept of *Identifiability*: leaving out problems in the identification originated by lack of excitation that is the aspect largely discussed in Section 3.1, we discuss the aspect of identifiability regarding parameters, that is the uniqueness of the parametrization of a model. This part is covered in detail in [16] and [17].

The last section is about *Sensitivity* that is a very convenient "tool" which helps in the choice of parameters through the analysis of the behaviour of the minimization criterion as the parameters or their number change.

3.1 Persistent Excitation

3.1.1 Finite Impulse Response Model

Let us consider a Finite Impulse Response (FIR) Model, that is a system with the impulse response that goes to zero and may then be truncated; this can be described by the following equation:

$$y(t) = b_1 u(t-1) + b_2 u(t-2) + \cdots + b_n u(t-n)$$

or

$$y(t) = \varphi^T(t-1)\theta$$

where

$$\theta^T = [b_1 \dots b_n]$$

$$\varphi^T(t-1) = [u(t-1) \dots u(t-n)]$$

We must impose some conditions on the input signal, otherwise we could be not able to determine the parameters of this model: for example, if we took zero as input, we would not be able to determine any parameters.
Theorem 3.1 in [11] says that the condition for uniqueness of the least-squares estimate is that the matrix

$$\begin{pmatrix} \sum_{k=n+1}^{t} u^2(k-1) & \sum_{k=n+1}^{t} u(k-1)u(k-2) & \dots & \sum_{k=n+1}^{t} u(k-1)u(k-n) \\ \sum_{k=n+1}^{t} u(k-1)u(k-2) & \sum_{k=n+1}^{t} u^2(k-2) & \dots & \sum_{k=n+1}^{t} u(k-2)u(k-n) \\ \vdots & \vdots & \vdots & \vdots \\ \sum_{k=n+1}^{t} u(k-1)u(k-n) & \dots & \dots & \sum_{k=n+1}^{t} u^2(k-n) \end{pmatrix} \tag{3.1}$$

has full rank (*excitation condition*).
We can see that introducing

$$\Phi = \begin{pmatrix} \varphi^T(n) \\ \vdots \\ \varphi^T(t-1) \end{pmatrix}$$

matrix 3.1 is equal to $\Phi^T\Phi$.
For long data sets the end effects are negligible so we can take k from 1 to t and we have:

$$C_n = \lim_{t\to\infty} \frac{1}{t}\Phi^T\Phi = \begin{pmatrix} c(0) & c(1) & \dots & c(n-1) \\ c(1) & c(0) & \dots & c(n-2) \\ \vdots & \vdots & \vdots & \vdots \\ c(n-1) & c(n-2) & \dots & c(0) \end{pmatrix} \tag{3.2}$$

where $c(k)$ are the *empirical covariances* of the input, that is:

$$c(k) = \lim_{t\to\infty} \frac{1}{t} \sum_{i=1}^{t} u(i)u(i-k)$$

Thus, we come to the following definition:

Definition 3.1 (Persistent excitation) *A square summable signal* u *is called* persistently exciting *(PE) of order* n *if the matrix C_n given by Eq. 3.2 is positive definite.*

and to the following theorem:

Theorem 3.1 (Consistency for FIR models) *Consider least-squares estimation of the parameters of a finite impulse response model with* n *parameters. The estimate is consistent*[1] *and the variance of the estimates goes to zero as 1/*t *if the input signal is persistently exciting of order* n*.*

3.1.2 Transfer Function Models

Let us now consider a dynamical system described by the model:

$$A(q)y(t) = B(q)u(t)$$

or

$$y(t) + a_1 y(t-1) + \cdots + a_n y(t-n) = b_1 u(t-1) + \cdots + b_n u(t-n)$$

[1] *consistency* is the property for which an estimate converges to the true parameter value as the number of observations increases towards infinity

where A and B are polynomials of order n and *n-1*, respectively. The sequence of inputs $\{u(1), u(2), \ldots, u(t)\}$ has been applied to the system and the corresponding sequence of outputs $\{y(1), y(2), \ldots, y(t)\}$ has been observed. Introducing the parameter vector

$$\theta^T = [a_1 \ldots a_n \quad b_1 \ldots b_n]$$

and the regression vector

$$\varphi^T(t-1) = [-y(t-1) \ldots - y(t-n) \quad u(t-1) \ldots u(t-n)]$$

the model can be rewritten as a regression model

$$y(t) = \varphi^T(t-1)\theta$$

So, the parameters can be estimated through *least squares*. We can assume that disturbances are described as *white noise* added to the system output $x(t)$:

$$x(t) + a_1 x(t-1) + \cdots + a_n x(t-n) = b_1 u(t-1) + \cdots + b_n u(t-n)$$

Determining the parameters that minimize the criterion

$$\sum_{k=1}^{t} (y(k) - x(k))^2 \qquad \text{where} \quad y(t) = x(t) + e(t)$$

is a least-squares problem whose solution is:

$$\hat{\theta}(t) = \hat{\theta}(t-1) + P(t)\varphi(t-1)\varepsilon(t) \tag{3.3}$$

$$P(t) = \frac{1}{\lambda}\left(P(t-1) - \frac{P(t-1)\varphi(t-1)\varphi^T(t-1)P(t-1)}{\lambda + \varphi^T(t-1)P(t-1)\varphi(t-1)}\right) \tag{3.4}$$

where $\hat{\theta}(t)$ is the estimation of $\theta(t)$ and

$$\varepsilon(t) = y(t) - \varphi^T(t-1)\hat{\theta}(t-1)$$

3.1.3 Other Methods

Least-squares estimation can be applied to all those models that can be written as regression models $y(t) = \varphi^T(t-1)\theta$. The resulting algorithms are the same of that described by Eq. 3.3 and 3.4 apart the fact that θ, φ and ε are different for the different kinds of model. Some examples of these models are some Nonlinear models and the Stochastic ones.

3.1.4 Persistently exciting signals

Considering the model

$$y(t) = \frac{B(q)}{A(q)}u(t) + \frac{B(q)}{A(q)}e(t)$$

we have the following important result:

Theorem 3.2 (Persistently exciting signals) *A square summable signal* u *is persistently exciting of order* n *if and only if:*

$$\lim_{t\to\infty} \frac{1}{t}\left(\sum_{k=1}^{t} A(q)u(k)\right)^2 > 0 \tag{3.5}$$

for all nonzero polynomials A of degree n*-1 or less.*

An easy calculation shows that:

$$\lim_{t\to\infty} \frac{1}{t}\left(\sum_{k=1}^{t} A(q)u(k)\right)^2 =$$

$$= \lim_{t\to\infty} \frac{1}{t}\left(\sum_{k=1}^{t} a_0u(k+n-1) + \cdots + a_nu(k)\right)^2 = a^T C_n a$$

where C_n is the matrix given by Eq. 3.2.

This useful theorem gives us interesting information about the signals which we should perform an experiment of identification with, here some examples follow:

- **Pulse**: From Eq. 3.5 it follows that $C_n \to 0$ for all n if u is pulse, so this cannot be PE for any n.
- **Step**: Let $u(t)$ a step, that is $u(t) = 1$ for $t > 1$ and zero otherwise. It follows that:

  $$(q-1)u(t) = \begin{cases} 1 & t = 0 \\ 0 & t \neq 0 \end{cases}$$

  So, a step can at most be PE of order 1. But since

  $$C_1 = \frac{1}{t}\sum_{k=1}^{t} u^2(k) = 1$$

  it follows that is PE of order 1.

- **Sinusoid**: Let $u(t) = \sin \omega t$, we have that:

$$(q^2 - 2q \cos \omega + 1)u(t) = 0$$

This implies that a sinusoid can at most be PE of order 2. Since

$$C_2 = \frac{1}{2} \begin{pmatrix} 1 & \cos \omega \\ \cos \omega & 1 \end{pmatrix}$$

it follows that a sinusoid is actually PE of order 2.

- **Periodic Signal**: Let $u(t)$ be periodic with period n. It then follows that:

$$(q^n - 1)u(t) = 0$$

So the signal can at most be PE of order n.

- **Random Signals**: Consider a mean square ergodic stochastic process with nonvanishing prediction error. Since the signal cannot be predicted, it follows that Eq. 3.5 holds. The signal is thus PE of any order.

3.1.5 Some remarks about Persistent Excitation

It is important to tell some facts about the topic of this section, that is Persistent Excitation.

The main matter is that the results of this theory are valid as time goes to infinity that is, when we cannot see the transitory effects anymore. Otherwise taking advantage of those effects we can properly identify parameters even with signal of a smaller Persistent Excitation order than the number of parameters.

For example, a step, with its transitory effect, is not bad at all for identification, and it can identify well more than one parameter. In Figure 3.1 we can see the result of this kind of experiment. We have taken a simplified version of the model used for our case study (see Chapter 4) and we have added some noises to generate "experimental data", then with these data we have tried to identify two parameters of the model, more exactly the inertias of the two pendula.

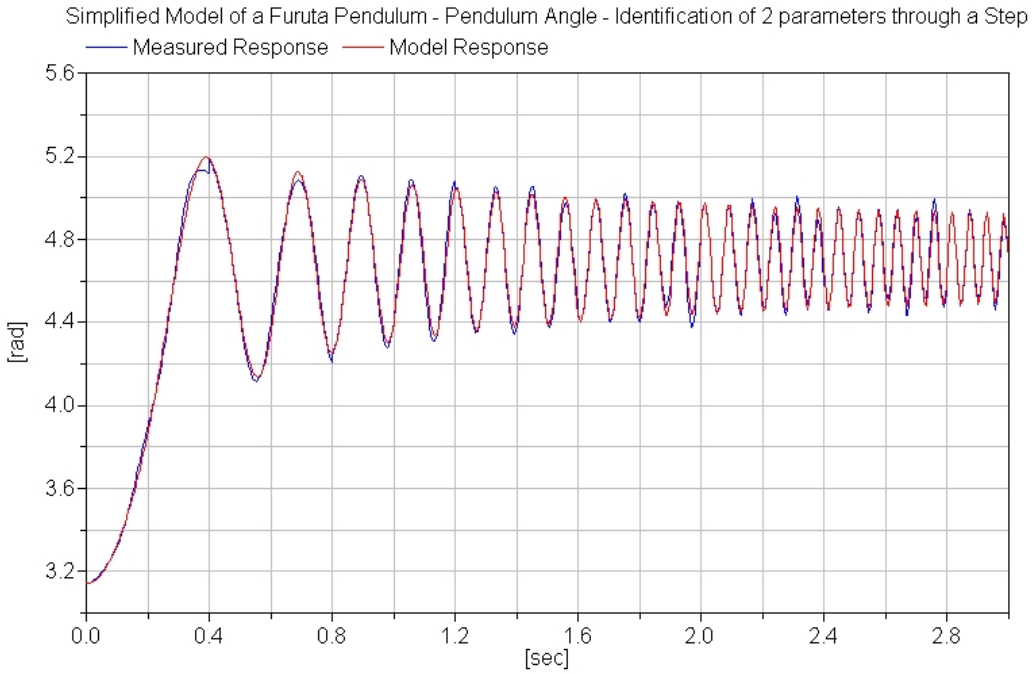


Figure 3.1: Identification of two parameters through a Step

The nominal values of the two parameters are:

$$Pendulum\ Inertia = 0.0008\ kg \cdot m^2$$
$$Arm\ Inertia = 0.02\ kg \cdot m^2$$

Here inertia is meant with respect to the center of mass of the bar.
Performing an optimization along the first 3 seconds of the simulation and starting with a guess of 0.0007 for Pendulum Inertia and 0.005 for the Arm one, we have obtained the following values, that are really good if compared with the nominal ones:

$$Pendulum\ Inertia = 0.000833446\ kg \cdot m^2$$
$$Arm\ Inertia = 0.0199541\ kg \cdot m^2$$

On the contrary the results obtained in the same conditions with a sinusoidal signal have not been that good. This can exactly be explained with the good transitory effects of Step signal and with the validity of persistent

excitation only for time that goes to infinity, that is when transitory effects are finished.

3.2 Identifiability

Identifiability is a fundamental concept in identification problems. This matter can be simply explained saying that a system is identifiable if it exists a procedure that brings to an unique value of the parameter vector θ and to a resulting model with the same behaviour of the real system. The main aspects that regard this issue are two: the first involves the suitable choice of the experimental conditions, that is we want to know if our data set is informative enough to discriminate between nonequal models; the second aspect, instead, assuming that our experiments are suitable, studies the possibility to find a unique parameter vector θ.
We omit, now, the experimental problem, which we have already discussed in Chapter 3.1 and we want to concentrate only on the lack of identifiability due to the parametrization problem: the so-called *structural identification problem*. We must notice, as written in [18], that the identification of an usable model does not always require structural identifiability.

Considering a set of models M, where each model $M(\theta)$ is described by its own parameter vector θ: the parametrization is said to be unique only if, for two parameter vectors θ_1 and θ_2, it holds that

$$M(\theta_1) = M(\theta_2) \Rightarrow \theta_1 = \theta_2$$

So, avoiding poor identifiability due to lack of excitation, we say that we have identifiability if a unique *a priori* system representation exists and it is independent of the experimental procedure.

One example of non structural identifiability is given in Figure 3.2. That picture shows two transfer functions that could describe the same system: actually the second one differs from the first, which is co-prime instead, only for a constant that multiplies both the numerator and the denominator.
While the first system is structurally identifiable, the second one is not, because for example any value of the parameter k fits the model.

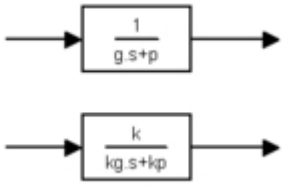


Figure 3.2: Example of non structural identifiability

3.3 Sensitivity

Sensitivity is an important aspect that we need to evaluate when we are designing a model. It can help us in the choice of the parameters, telling us which ones are relevant and which ones are not.

Sensitivity studies the effects of parameter variations on the value of the criteria function.
The simplest way of studying Sensitivity is to investigate the criteria function with respect to the parameters: we could vary slightly the value and see what happens; a great variation of the value of the criteria function means which that parameter is relevant in the identification, otherwise if the variation is small, it means that parameter is not influential.

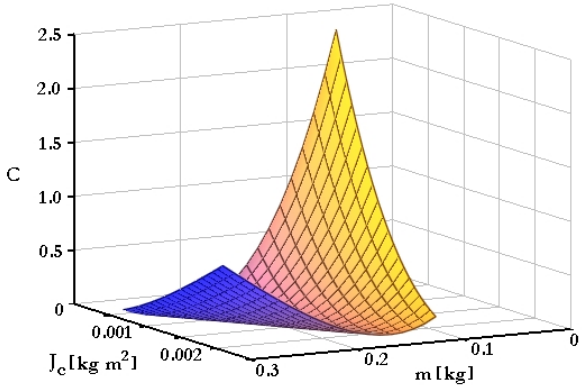


Figure 3.3: Plot of a Criteria Function

That is illustrated in Figure 3.3 where we can see this method applied to the case study analyzed in Chapter 4. The figure shows the value of the criteria

function with respect to J_c, that is the inertia of the arm with respect to a vertical axis through the center of mass, and to m that is the mass of the arm. Being r_c the distance between the point of rotation and the center of mass and being J_a the inertia of the arm with respect to the point of rotation, we have the following relation:

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

From the picture we can notice that there is a valley where the function is minimum and constant: which means that there are several couples of values of J_c and m for which the criteria function is minimum. We can also see that the function increases when J_a decreases; this implies that J_c and m cannot be estimated separately, while the parameter that can be estimated is actually J_a.

Another possible way to investigate the sensitivity of the criteria function is calculating the *Sensitivity Matrix* that is the Hessian of the minimization problem.

Let us assume $r_c = 0.1225$ and known, then let us assume the nominal values $J_c = 0.0014$ and $m = 0.165$. As we can see in [13] the sensitivity matrix is:

$$\begin{pmatrix} 431064 & 6456.3 \\ 6456.3 & 97.0671 \end{pmatrix}$$

Its eigenvalues are 431161 and 0.37 where the second can be considered zero considering the numerical accuracy. The eigenvector having the large eigenvalue is $\{0.99989, 0.01498\}$ that means a large sensitivity in the direction

$$\{0.99989, 0.01498\} \cdot \{J_c, m\}$$

Recall $J_a = J_c + 0.01501 \cdot m$ which shows that the criterion is sensitive for variations in J_a. Since a symmetric matrix has real eigenvalues and orthogonal eigenvectors, $\{-0.01498, 0.99989\}$ is orthogonal and being its eigenvalue very close to zero, it means that the measured behaviour is insensitive in that direction as confirmed by Figure 3.3.

Chapter 4

Case Study

In this chapter we are going to present our case study: the so-called *Furuta Pendulum*. After a short introduction about the system and its advantages followed by a quite detailed description of the device used for our experiments, we talk about all the process of identification and validation that has been conducted: the experiments performed, the simulations made and the final results obtained.

4.1 Furuta Pendulum

4.1.1 Presentation of the System

In this section we will speak about a system known as *"Furuta Pendulum"*. This denomination comes from the name of its creator, Professor Katsuhisa Furuta.

It is a simple variation of the classical inverted pendulum: it consists of an inverted pendulum connected to the end of an actuated horizontal bar, also known as *Arm*; this one has a servo-motor attached roughly to one of its ends and it has the task to balance the un-actuated pendulum attached to the other its end.

There are some advantages that make the use of this kind of device more interesting in respect to the classical inverted pendulum on a cart. The first one is just that we haven't the cart so we do not need of a lot of space to make it move: the cart moves on a guiding rail and if this is too short, the

experiment does not work; moreover we have not problems regarding wheel slippage anymore and finally, in the Furuta Pendulum there are not any motors to move with all the advantages that it implies[1].
Nonetheless, in this system, there is also the disadvantage of the introduction of big rotational nonlinearities into the dynamics and thus into a possible control law.

4.2 Experiments

4.2.1 Device

The experiments have been performed in one of the laboratories of Lund Institute of Technology where we can find an instance of the system. The device can be seen in Figure 4.1. This device is connected, through a 12 bit

Figure 4.1: Furuta Pendulum in LTH Lab

[1]there are, however, instances of the inverted pendulum with and without the motor on the cart

AD/DA-converter board, to a standard Pentium PC running Linux. Matlab with Simulink is used for acquisition and storage of the measurements.

The interface manages twelve signals: one for the Ground, one for the Control Signal, four of them required by a joystick and the others regarding the four states of the system (Table 4.1). The joystick would be used to generate a reference trajectory, but it is not interesting for these experiments. To notice that the signals used for the state of the pendulum are four instead of two because there are two more sensors for a better measurement around the upright position[2] in view of a possible stabilization and velocity control.

I/O Connection	Pendulum Signal
AI2	Pendulum Angle (Top)
AI3	Pendulum Velocity (Top)
AI4	Arm Position
AI5	Arm Velocity
AI6	Pendulum Angle (360 degrees)
AI7	Pendulum Velocity (360 degrees)
AO0	Control Signal
Ground	Ground

Table 4.1: Signals regarding the two Pendula

Once started the simulation, the signals are acquired and then transformed from Volt to Radian by the complex Simulink diagram in Figure 4.2. Since the angle of the pendulum, θ, is defined to be zero when the pendulum is in the upright position and to be positive when the pendulum is moving clockwise, an offset is added to each signal and then, what we obtain is multiplied for a gain to have a range of 2π rad for the angles of the two pendula, a value of 0 rad for the pendulum angle in the upright position, a value of π rad for the vertically downward position and a value of 0 rad/s for the velocity of the two pendula when they are standing still.

So, after some transformations we obtain the six signals. Thus, in order to have, in the end, only the four needed measures corresponding to the four states of the system, the six signal are propagated to a Simulink block (see

[2]these measurements are indicated in Table 4.1 with *Top*

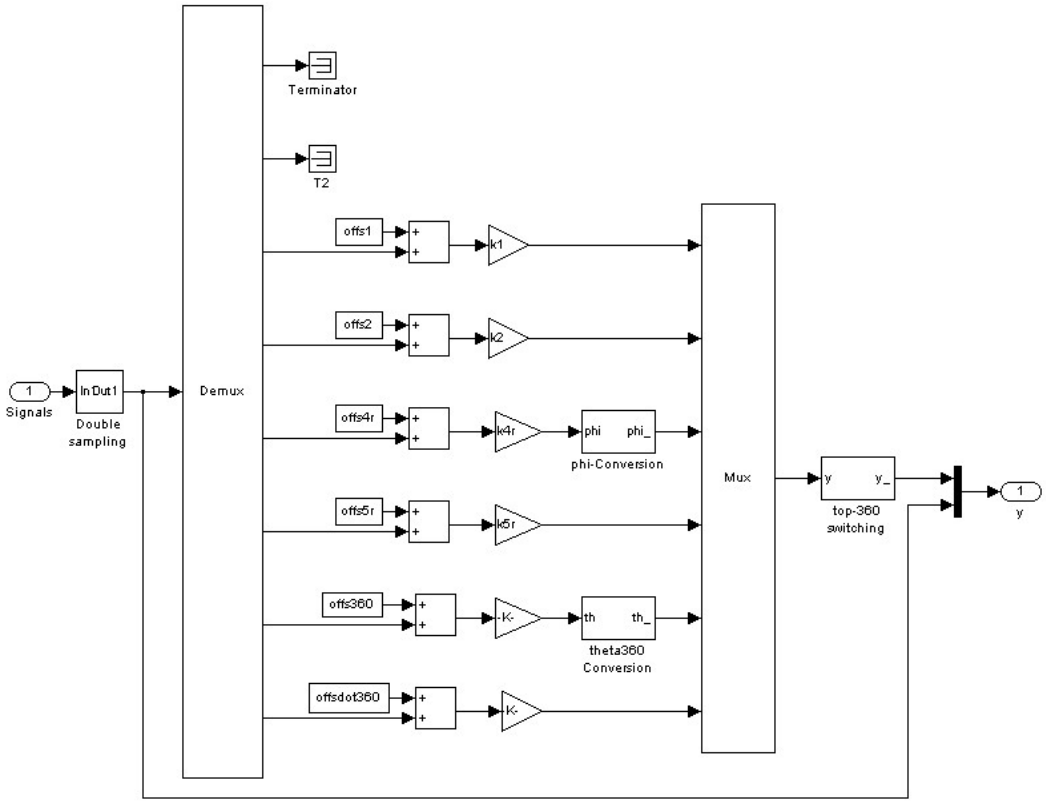


Figure 4.2: Conversion of Inputs

Figure 4.3) which has the task to select between the two measurements of the sensors that are the one for the entire range of angles (*360 Degrees*) and the one for the upright position (*Top*).

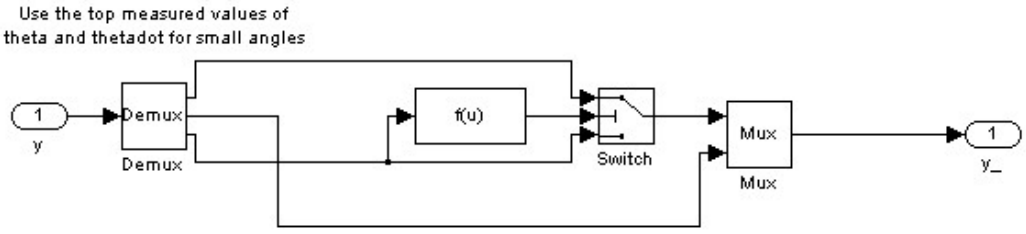


Figure 4.3: Simulink Block switching between sensors

4.2.2 Execution of the Measurements

Once illustrated the utilized device, we can now explain how the experiments have been performed. We will start with all the steps needed to run the simulation:

1. Switch on the computer and, after the boot, run Matlab[3]
2. Load the Simulink diagram called *simuldiagram.mdl* typing `open 'simuldiagram'` in Matlab
3. Click on *Simulation* in the just opened Simulink window, then on *Simulation parameters...* and, finally, set *Stop time* to 50 sec that is about the time that pendula need to stop. Leave the other parameters to their default values (Figures 4.4 and 4.5)
4. In the *simuldiagram* window (Figure 4.4), double-click on the block *Scope* and after, in the new opened window, click on *Parameters*[4], then on *Data History* and set parameters as seen in Figure 4.6[5]. Leave the others unchanged.
5. Click again on *Simulation*, then on *Start* (Figure 4.4) and, finally, let the pendulum swing

We must notice that the file *simuldiagram.mdl* is an adjustment of the file *pendlib.mdl* that can be found at [7]; in this version the block *Conversion of Inputs* is changed: we have added some connections so to have also the raw data in the outputs (see Figure 4.2).

After having described how to run a simulation, we must explain which experiments have been performed.
At the beginning we made all those procedures concerning the setup of the device. We have found the right values for the gains and the offsets needed in

[3] on the LTH Lab computer type *matlab -R12.1 -nojvm* in the command shell

[4] the second icon on the left

[5] the interface for MOPS in Dymola requires that the name of the matrix of the data in the Matlab file is *Data* (with the first letter in upper case). Besides data must be stored in *Array* format

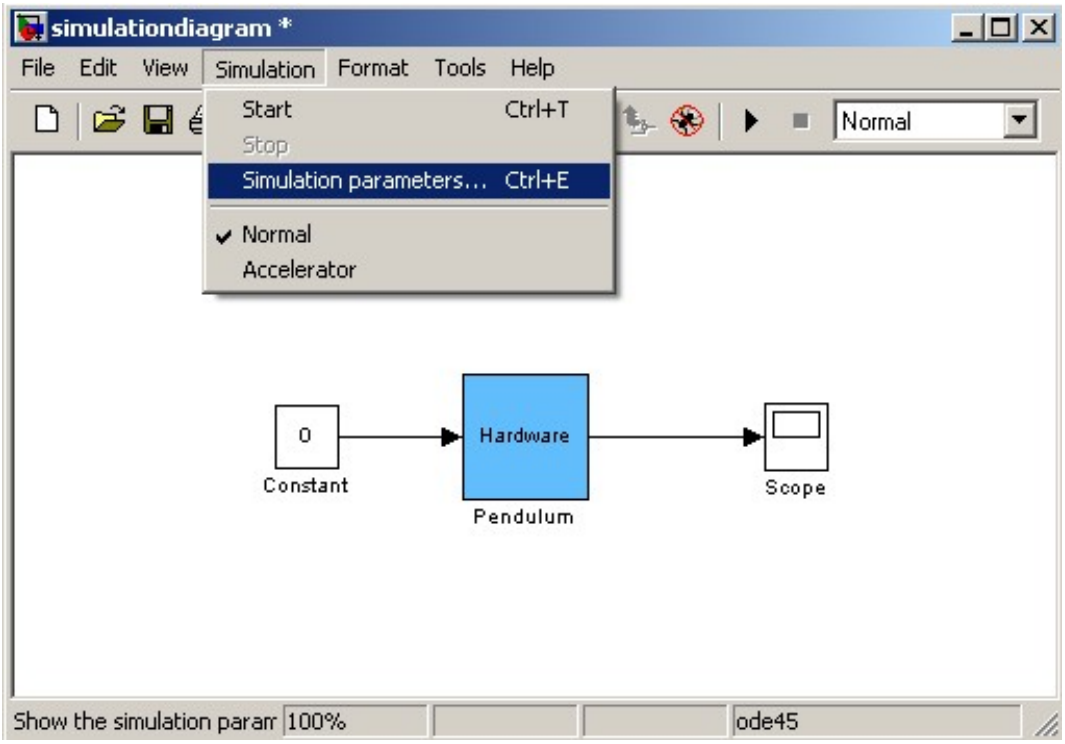


Figure 4.4: Simulink Diagram used for simulation

the Simulink diagram of Figure 4.2. The gains are estimated making the pendulum turn 360 degrees and looking then at the range of the raw data (Figure 4.7): the gain is 2π divided by the range of the raw data. Since the sensor does not reset itself after a full turn, we must make the pendulum turn exactly 360 degrees and so we had to put some landmarks to indicate the place where we started turning the pendulum. As landmark, it can be taken any object with an opportune shape. This calibration has not been so easy to perform because, also with the landmark, it was difficult to stop the pendulum precisely in the right place because of the swinging of the pendulum itself. The offsets, instead, have been found forcing the conditions mentioned in Section 4.2.1 that is, the constraints on angles and angular velocities of the two pendula.
All these parameters have to be changed in the Matlab initialization file called

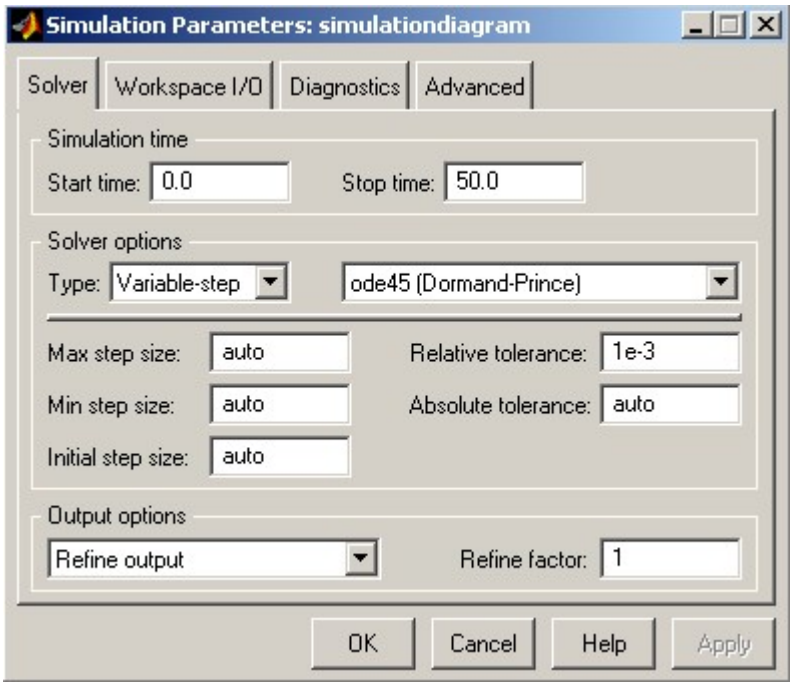


Figure 4.5: Parameters of the Simulation

pendinit.m[6] that contains the values of all the parameters used in the file *simuldiagram.mdl*.
Moreover, we have looked also at the voltage of the pendulum standing still, to have an estimation of the noise in our measurements. It is important to notice that all these parameters should be re-tuned whenever we perform new experiments because of the interaction of the device with the external environment that brings to the changing of its parameters.

After the setup we have started performing the experiments. To prevent measurement errors and also to have data for the validation task, those have been repeated several times. The chosen initial conditions are: pendulum in the upright position, in the horizontal position, that is parallel to the ground, and some other "random ones", where random means not naked-eye determinable angles; for each one of these initial conditions we have taken at least three measurements.

[6]also this file can be found at [7]; this file can be loaded from the Matlab prompt just typing its name without extension

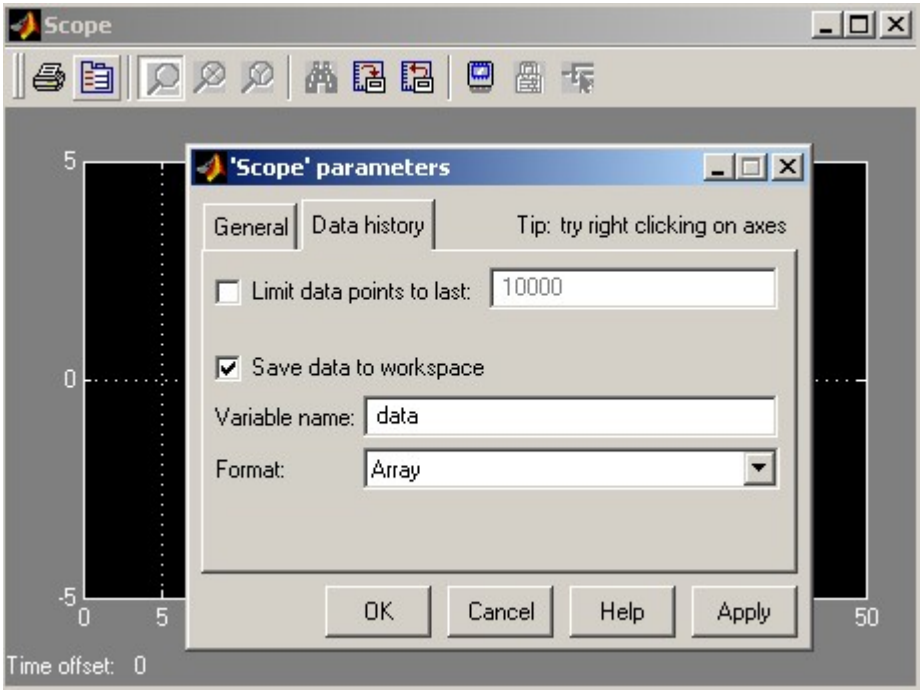


Figure 4.6: Parameters of the block Scope

It is very difficult to perform experiments manually: at the beginning we have to start the log of data and then we have to release the pendulum; it is impossible to do these two things at the same time, so after the experiment we have to edit the file with measurements and cut out the relevant part (see Section 4.2.3). Those troubles caused also initial conditions different from the "nominal[7]" ones so that we always had a velocity different from zero and a starting angle quite different from the "nominal" one.
Besides, the lightness of the pendulum body, combined with the quite big freedom of its joint not only on the x-axis, but even along the z-axis, causes a lot of vibrations on this last axis. Likewise, probably because of vibrations and excessive freedom of the joint, the arm seems to get stuck and to move jerkily.
All these considerations will bring us to the idea of adding, to our model, another joint for the z-axis (see Chapter 4.4.4).

[7]the exact horizontal or upright position

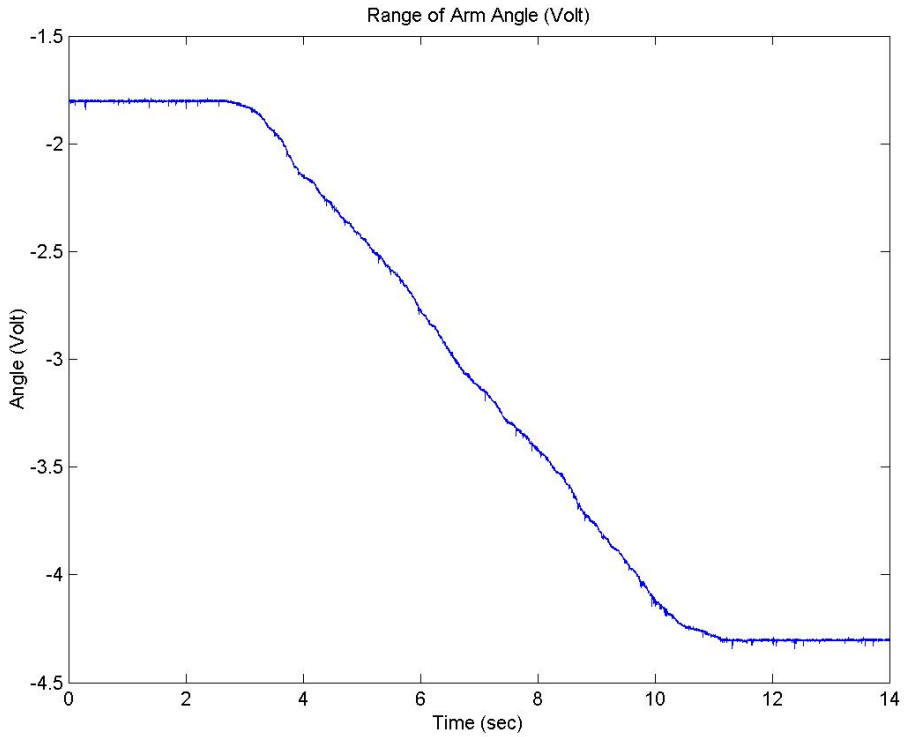


Figure 4.7: Range of Arme Angle

After the experiments we find, in the Matlab Workspace, a matrix with thirteen columns. The first one is time: we have chosen a sampling time of 0.01 seconds because a shorter one would have been quite useless and it would have led to a very big amount of data; the second and the third are respectively Pendulum Angle and Speed and the fourth and the fifth are Arm Angle and Speed. The remaining eight columns are raw data and they are ordered as in Table 4.1.

The matrix exists only in the Matlab Workspace, at the moment. We must, then, save it on a file to be able to recall it with Dymola or to open it again in the future. To succeed in opening a Matlab file, Dymola requires that this is saved in *Matlab 4 format*. This operation can be performed with the following Matlab command: `save filename -v4`.

4.2.3 Data Adjustments

In performing measurements and processing data we have had to use some practical tricks.

For example we have noticed that, when the sensors start to measure, the very first value of these measurements, is completely wrong, being, this, very distant from the values that follow it. Furthermore, performing the experiments alone, it was impossible for me to release the pendulum and start the data acquisition at the same time. The release itself was troublesome: holding the pendulum with my hand or with a pen implied, when I left the pendulum swing, that I impressed a certain force on the bar, so to have a certain initial acceleration of it. This difficulties have been partially solved starting the acquisition of the data some seconds before releasing the pendulum.

All these things have meant that we have cut the first instants of the measurements, obtaining, again, starting angles quite different from the "nominal" ones. This action has been carried out in Matlab with some swaps of matrix[8]. We have then adjusted the offset in the measurement to obtain a value of π rad for the angle of the pendulum in the vertically hanging downward position and a value of 0 rad/s for the velocity of the two pendula when they are standing still.

After, we have chosen to focus only upon the first three seconds of the measurements because the behaviour of the two pendula is more distinctive and interesting here. Specially the motion of the arm is very hard to follow during the first seconds of the experiments.

Moreover raw data have been left out to obtain a smaller and more manageable file being the optimization a very CPU-stressing and memory-greedy task.

At the end of all these procedures, we have chosen which experiments to use for our tasks. We have decided to perform our calibration on one of the experiments with the pendulum starting in the "horizontal position" (the file is called *HorLight.mat*) and the validation on one of the experiments with the pendulum starting in the "upright position" (the file is called *Up3Light.mat*). The real initial conditions in these files are showed in Table 4.2.

[8]in addition to cutting the data, we must also reset the time in the matrix

	HorLight.mat	***Up3Light.mat***
Arm Angle (rad)	-0.0122	-0.0851
Arm Speed (rad/s)	-0.0635	-2.1246
Pendulum Angle (rad)	1.6803	-2.0698
Pendulum Speed (rad/s)	1.7812	-8.7596

Table 4.2: Real Initial Conditions of experiments

4.3 Choice of the Parameters

A very important problem in this work is the choice of the parameters. This choice is problematic because we have to decide which and how many parameters to take.

Too many parameters imply, maybe, a very good, almost perfect, fit for the calibration task but, probably, a very bad behaviour during the validation process: this is the real issue, because the model, of course, should work well in every situations, otherwise this would be useless for any tasks regarding controls and automation. Actually, also a perfect estimation of the model parameters could be useless for those tasks: it is not important that the values of the found parameters are equal to the actual ones in fact, due to simplifications and approximations in the model, model parameters are often the result of a combination of a lot of real parameters; the important thing, the goal of identification, is instead having a model that works well in every conditions, so that it can almost supersede the real system.

Speaking about the physical parameters, we can say that there are several papers that examine the Furuta Pendulum in the lab. Some of those relate values of the real system parameters slightly different from those which can be found in [9], that is our most important reference about the real device. The values that can be found in this paper are:

m_{pa}	$0.02\ kg$	Mass of pendulum
M	$0.015\ kg$	Mass of pendulum weight
l_p	$0.421\ m$	Pendulum length
r	$0.245\ m$	Arm length
r_{cm}	$0.44\ m$	Distance from center of rotation to center of mass of arm

m_a	$0.165\ kg$	Mass of the arm
J_m	$0.0000381\ kg \cdot m^2$	Moment of inertia of motor and tachometer, from data sheet

Since we are not completely sure about these values, we have decided to trust only to those which can be easily measured by ourselves. Considering the fact that the only values that can be easily measured, without dismantling the device, are lengths and that the only length that we have used in our model is the Arm one, the sole parameter that we are going to consider known is the length of the Arm, whose value is really $r = 0.245$.

Regarding the Arm, it has been made the approximation that its center of rotation corresponded with its center of mass and that it was at the beginning of the bar, which is the origin of our coordinate system: this has been made setting $Arm.r\_CM = \{0, 0, 0\}$, where $Arm.r\_CM$ is the vector from the beginning of the bar to the center of mass of itself.
The other parameters of the Arm are its mass and its inertia; since they cannot be estimated distinctly (as we can see in [13] and in Chapter 3.3), we have taken the inertia as a parameter and imposed the mass as seen above. Since the arm rotates around the y-axis, it is only the inertia with respect to that axis that influences the movements of the system. So the only not null element of the inertia tensor is $I_{2,2}$ which has been set as a parameter to find ($Arm.I\_22 = inertiaarm$ as we can see in Figure 4.8).

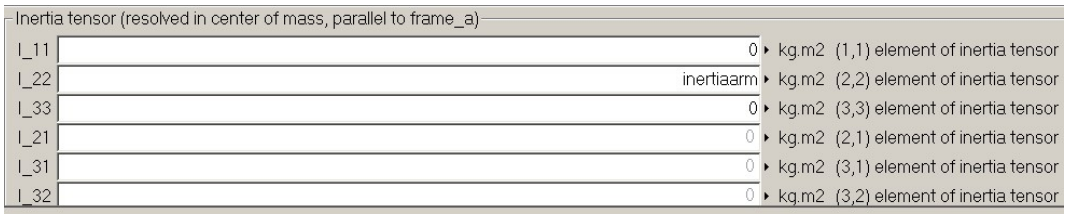
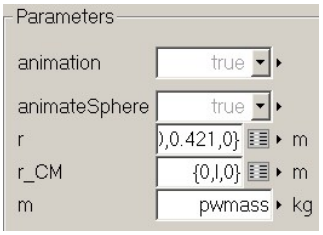


Figure 4.8: Choice of Arm Parameters

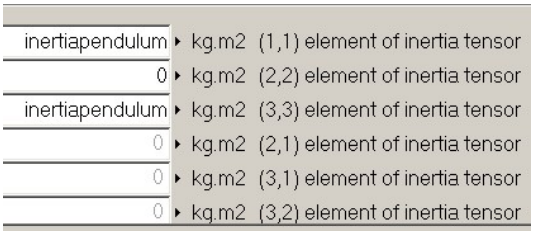
Concerning the Pendulum, the parameters that have been picked for optimization are its inertia, its mass and the position of its center of mass.
The inertia with respect to all axes is perpendicular to the pendulum length axis ($Pendulum.I\_22 = 0$) and it has been assumed equal (we have set $Pendulum.I\_11 = Pendulum.I\_33 = inertiapendulum$) as shown

in Figure 4.9(b)).
For the pendulum it is also possible to estimate the position of its center of mass (we have set $Pendulum.r\_CM = \{0, l, 0\}$) and its mass ($Pendulum.m = pwmass$), depending, the inertia sensed by the arm, on the angle of the pendulum (see Figure 4.9(a)).

It is important to notice that, connected to the pendulum, there would be also a small weight: it could be considered as a point-mass object. Since this is directly attached to the pendulum bar, it is impossible to estimate its own mass, so, simplifying, we have preferred to remove this object from the model and considering it as if it was part of the pendulum itself. Thus, the pendulum mass is actually the pendulum mass plus the weight mass.



(a) Parameters regarding Mass



(b) Parameters regarding Inertia

Figure 4.9: Choice of Pendulum Parameters

All these parameters are, of course, subjected to the constraint to be strictly positive due to the fact the inertias and masses cannot be negative. Then, the parameter l, which is the position of the pendulum center of mass, other than having to be positive, must also be smaller than the length of the pendulum because the center of mass cannot stay out of the pendulum bar.

About the choice of the parameters regarding the friction it will discuss when we will extensively analyze the MultiBody Models, that will be in Chapters 4.4.3 and 4.4.4.

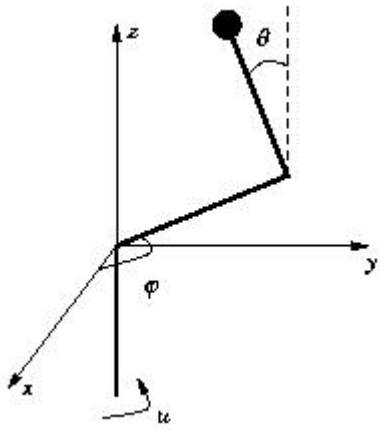


Figure 4.10: A schematic picture of the Furuta Pendulum

4.4 Modeling

4.4.1 Simple Mathematical Model (without Friction)

In Figure 4.10 we can see a simple scheme of this device. Let the length of the pendulum be l, the mass of the weight M, the mass of the pendulum m, its moment of inertia J and the moment of inertia for the arm J_p. The length of the arm is r. The angle of the pendulum, θ, is defined to be zero when in upright position and positive when the pendulum is moving clockwise. The angle of the arm, φ is positive when the arm is moving in counter clockwise direction. Further, the central vertical axis is connected to a DC motor which adds a torque proportional to the control signal u.

We are not going to examine here the complete derivation of the Furuta pendulum dynamics. This matter, based on Lagrange theory, has been discussed by Gäfvert in [14] and brings to these nonlinear equations describing the behaviours of the two pendula:

$$(J_p + Ml^2)(\ddot{\theta} - \dot{\varphi}^2 \sin\theta\cos\theta) + Mrl\ddot{\varphi}\cos\theta - gl(M + m/2)\sin\theta = 0$$
$$Mrl\ddot{\theta}\cos\theta - Mrl\dot{\theta}^2\sin\theta + 2(J_p + ml^2)\dot{\theta}\dot{\varphi}\sin\theta\cos\theta + (J + mr^2 + Mr^2 +$$
$$+ (J_p + ml^2)\sin^2\theta)\ddot{\varphi} = u$$

For simplifying, we can take:

$$\begin{array}{ll} a = J_p + Ml^2 & b = J + Mr^2 + mr^2 \\ c = Mrl & d = lg(M + m/2) \end{array}$$

So, the equations can be rewritten:

$$a\ddot{\theta} - a\dot{\varphi}^2 \sin\theta\cos\theta + c\ddot{\varphi}\cos\theta - d\sin\theta = 0$$
$$c\ddot{\theta}\cos\theta - c\dot{\theta}^2\sin\theta + 2a\dot{\theta}\dot{\varphi}\sin\theta\cos\theta + (b + a\sin^2\theta)\ddot{\varphi} = u$$

4.4.2 Linearized Model

We can try to linearize the equations around an equilibrium point and see what we obtain.
We take the following state:

$$x = (\theta \quad \dot{\theta} \quad \varphi \quad \dot{\varphi})$$

and linearize around this point:

$$x = (0 \quad 0 \quad 0 \quad 0)$$

which is the upright position of the pendulum with zero velocity. We obtain:

$$\dot{x} = Ax + Bu = \begin{pmatrix} 0 & 1 & 0 & 0 \\ \frac{bd}{ab-c^2} & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ \frac{-cd}{ab-c^2} & 0 & 0 & 0 \end{pmatrix} x + \begin{pmatrix} 0 \\ \frac{-c}{ab-c^2} \\ 0 \\ \frac{a}{ab-c^2} \end{pmatrix} u$$

We can now calculate the characteristic polynomial of the system that is the determinant of $(A - \lambda I)$:

$$p(\lambda) = \det(A - \lambda I) = \det \begin{pmatrix} 0 & 1 & 0 & 0 \\ \frac{bd}{ab-c^2} & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ \frac{-cd}{ab-c^2} & 0 & 0 & 0 \end{pmatrix} - \lambda \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} =$$

$$= \det \begin{pmatrix} -\lambda & 1 & 0 & 0 \\ \frac{bd}{ab-c^2} & -\lambda & 0 & 0 \\ 0 & 0 & -\lambda & 1 \\ \frac{-cd}{ab-c^2} & 0 & 0 & -\lambda \end{pmatrix} = \lambda^4 - \frac{bd\lambda^2}{ab - c^2}$$

This polynomial becomes zero for the following four values of λ:
$\lambda = \pm\sqrt{\frac{bd}{ab-c^2}}$ and $\lambda = 0$ (twice).
So the linear system has a pole with a positive real part and then the equilibrium point, for Lyapunov, is surely unstable.

4.4.3 First MultiBody Model

The first presented MultiBody model is simply composed by two Actuated Revolute Joints[9] and two Rigid Bodies[10]: the one that we call "*Arm*" rotates standing parallel to the ground; on the contrary, the other, that we call "*Pendulum*", rotates around the arm. Thus, being the arm parallel to the ground, the joint to which it is attached spins around the y-axis and the other joint, the one to which the pendulum is attached, spins around the x-axis since the pendulum hangs down vertically.

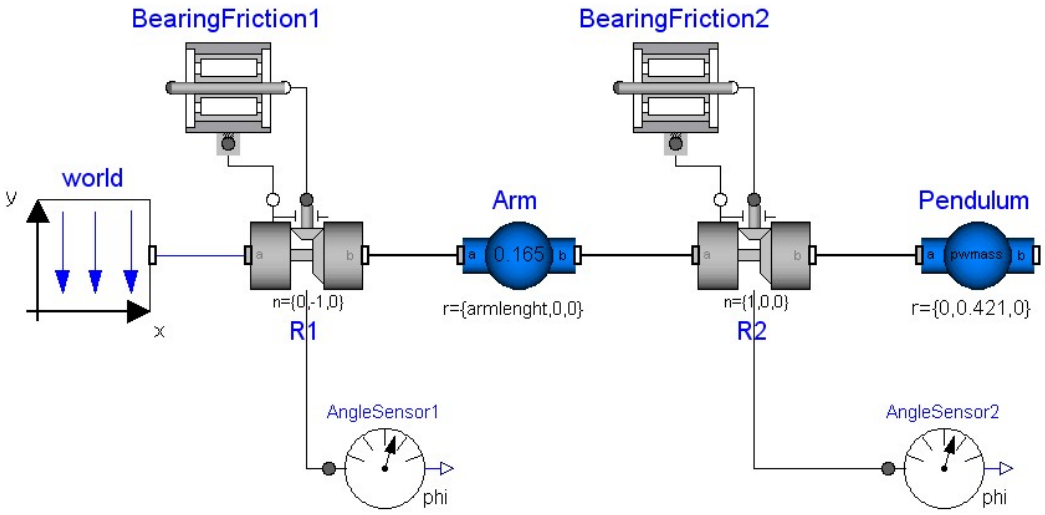


Figure 4.11: First MultiBody Model (2 Joints)

The two pendula, rotating, are subjected to a friction force that we have

[9] the Modelica class used is Modelica.Mechanics.MultiBody.Joints.ActuatedRevolute (see Appendix B.2)

[10] the Modelica class used is Modelica.Mechanics.MultiBody.Parts.BodyShape(see Appendix B.3)

modeled as a *Bearing Friction*[11]. This element describes Coulomb friction in

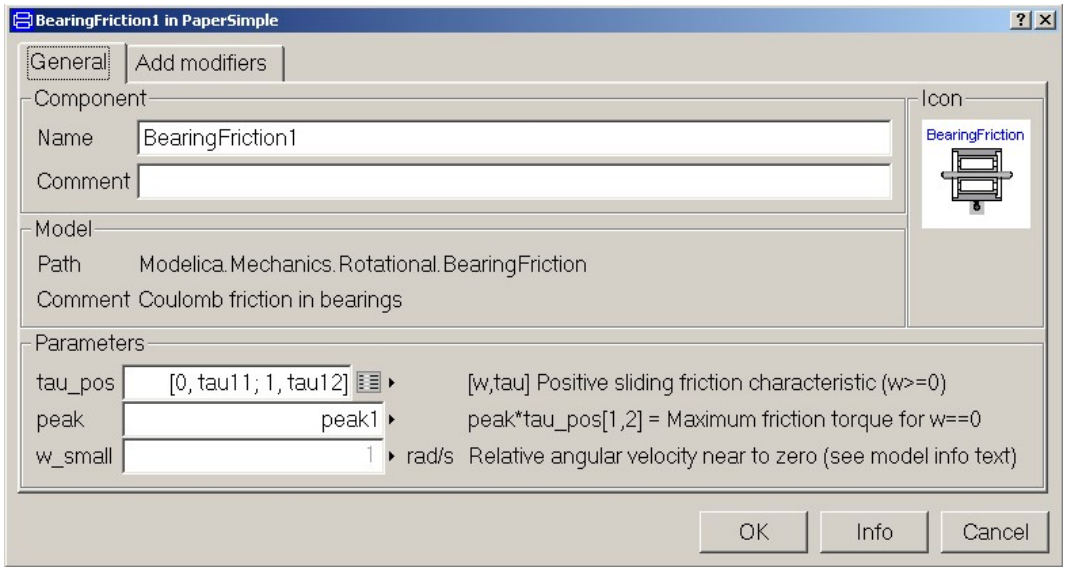


Figure 4.12: Parameter Choice for the BearingFriction element

bearings, and it has, as we can see in Appendix B.4 or in Figure 4.12, at least 3 parameters to set (*tau_pos[:,:]*, *peak* and *w_small*, where the first one is actually a table, so in general we have more than 3 parameters). In this first model we have taken the force with a *linear dependence on velocity*, so we have decided to leave w_small to its default value ($w\_small = 1$) and to choose *tau_pos[:,:]* and *peak* as parameters of the optimization. Since, as we have just said, the frictional torque in this model has a linear dependence on velocity, we have taken tau_pos[:,:] like in Table 4.3.

Joint N	
ω	τ
0	tauN1
1	tauN2

Table 4.3: tau_pos[:,:] for Joint N

[11] the Modelica class used is Modelica.Mechanics.Rotational.BearingFriction (see Appendix B.4)

We can see in the graph of Figure 4.13 that (0,tauN1) and (1,tauN2) are the points which the straight line, representing the trend of the frictional torque in respect to the angular velocity, passes for. In fact we have that for $\omega = 0$, won the initial static friction given by $peak \cdot tauN1$, the joint starts rotating subjected to a dampening torque that at the beginning is equal to tauN1 and grows as $(tauN2 - tauN1)\omega$. The first Joint R1 is then connected to

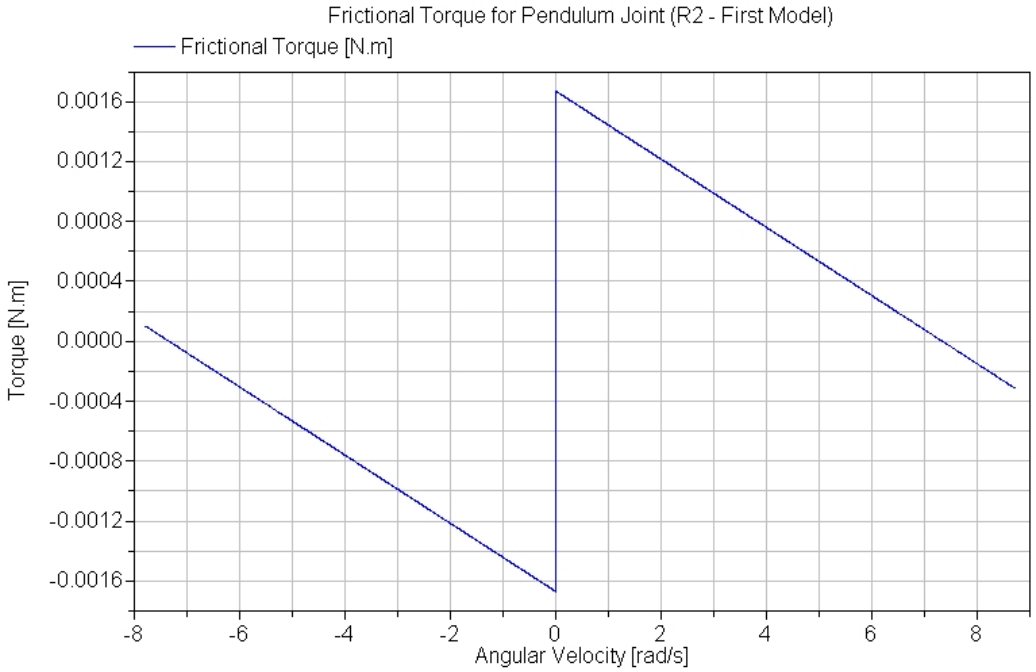


Figure 4.13: Frictional Torque for Pendulum Joint

an element called *World*[12]: this component, as we could understand from its name, represents the outside world for our model, in fact it gives to the model a global coordinate system fixed in ground and a gravity field.

Furthermore an "ideal" sensor[13] is linked to each joint to measure the angles of the two pendula that will be used in comparison with the measured response of the real device to calculate the criteria function which will be used for the

[12]the Modelica class used is Modelica.Mechanics.MultiBody.World (see Appendix B.1)

[13]the Modelica class used is Modelica.Mechanics.Rotational.Sensors.AngleSensor (see Appendix B.6)

optimization: our criteria function is a weighted sum of the integrated square difference with respect to measured value for each variable. The sensor has been said "ideal" because it actually measures the angle of a model and not of a real system, that is it is a result obtained from a simulation. To notice that, though we have also the measurements of the angular velocities, we have relied only on pendulum angles which are actually the integrals of those velocities: practically we have decided to trust in the measurements of the angle sensors leaving out the others, that would not add further knowledge on the system.

4.4.4 Second MultiBody Model

This model is an evolution of the previous one. We have observed that, when we let the pendulum hang down, there are a lot of vibrations of the pendulum. These vibrations obviously interfere with the movements of both pendula. Then we have decided to model, in some way, these chatters to see if it helps with the identification of the system.

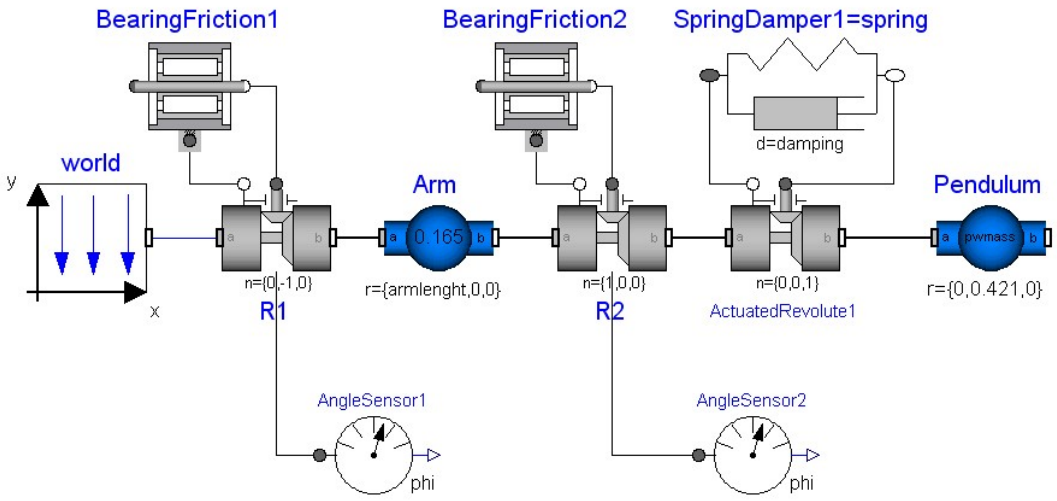


Figure 4.14: Second MultiBody Model (3 Joints)

All the things have been left as in the previous model except for the insertion of an extra joint to the pendulum that rotates around the z-axis. A spring

and a damper[14] are linked to this joint, instead of a Bearing Friction element, as in the other cases; this because, with these vibrations, the pendulum seems really to behave like it was connected to a spring.

Also the model of the friction is changed. The behaviour of the arm was not completely satisfying, so we have tried with something different for it. The matters regarding the pendulum, whose behaviour was almost perfect, have on the contrary been left unchanged except, obviously, for the value of the parameters which has been changed by the new optimization. We have left the same Bearing Friction elements, but, for the arm, we have added some parameters to make the model of its own friction more realistic. In this case the frictional torque has not a simply linear dependence on angular velocity, but, instead, a *Piece Wise Linear* (PWL) dependence[15] (see Figure 4.15).

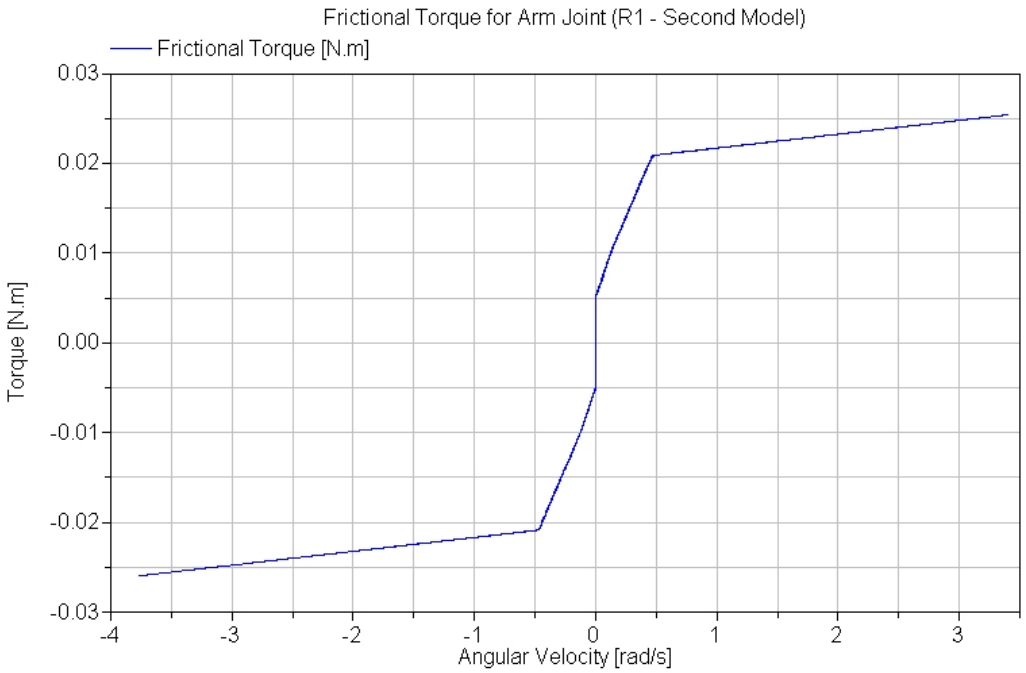


Figure 4.15: Frictional Torque for Arm Joint

To have such a trend for the torque we have added some parameters for the

[14]the Modelica class used is Modelica.Mechanics.Rotational.SpringDamper; this element is composed by a spring and a damper connected in parallel (see Appendix B.5)

[15]Dymola currently supports only linear interpolation in the table

making of the tau_pos[:,:] table: we have added new points of interpolation in the torque curve to reach a final number of four points, two more in respect to the previous model. It must also be noticed that, aside from the first point that requires only the ordinate, all the other points have abscissa and ordinate as parameters while the linear dependence of the previous model required only ordinates being required only two points (that can have any abscissa) to identify a straight line. The insertion of these new five parameters produces the Table 4.4.

Joint 1	
ω	τ
0	tau11
point0	tau01
point1	tau05
point2	tau12

Table 4.4: tau_pos[:,:] for Joint 1

Further experiments have been conducted adding some more points of interpolation in the torque curve, but results have been almost the same, so to justify our choice not to add other parameters which would increase uselessly the computational time and some possible behaviour problems in the validation.

The new element introduced in this model and called SpringDamper is a Spring with a Damper in parallel, so to have a dampened vibrating effect on the pendulum. This element has three parameters: c (the spring constant), phi_rel (the unstretched spring angle) and d (the damping constant); we have left the second to its default value ($phi\_rel = 0$) and taken the others as variables.

4.5 Optimization

4.5.1 Setting up the Optimization

After the description of our physical system and our models, in this section we will discuss about the main part of this work: the optimization process.

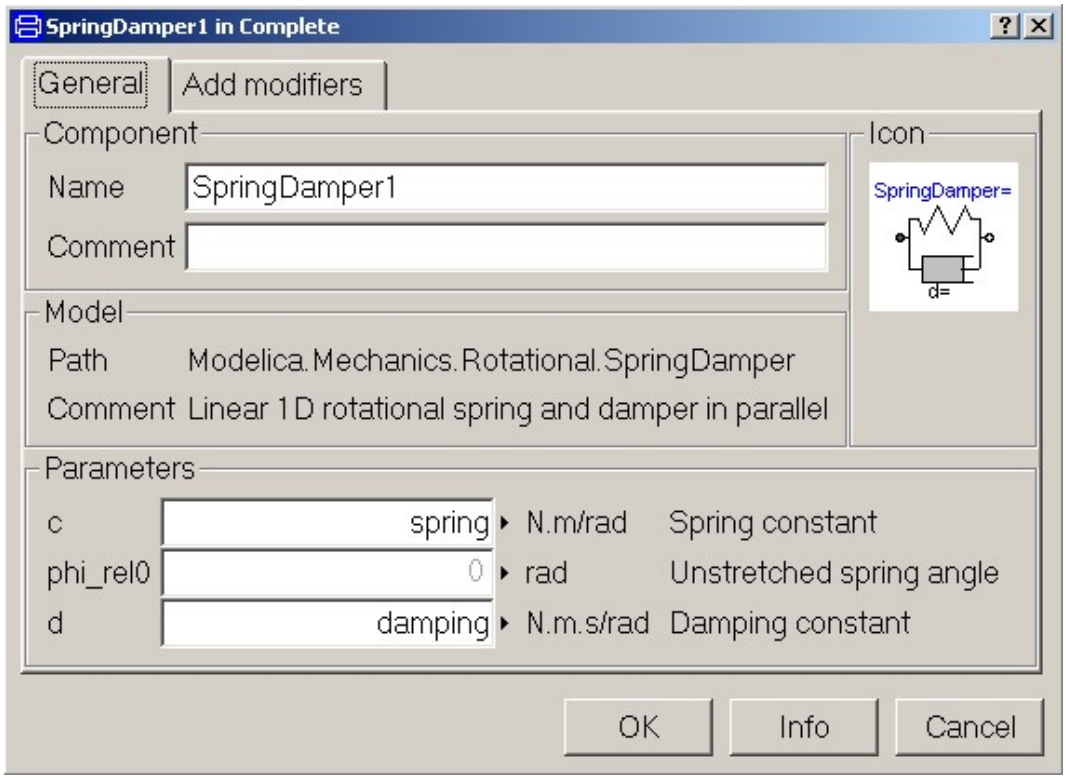


Figure 4.16: Parameter Choice for the SpringDamper element

In the last versions of Dymola a simple optimizer called *Basic Optimizer* was present, but now, for the upcoming release of Dymola 6, Dynasim is working on a GUI[16] that links Dymola to the much more powerful optimizer called MOPS. Thus, being this GUI still in a development phase, it is possible that some of the operations described here could be slightly different from those which will be needed for working with the final release of the software[17]. On the other hand, since MOPS will be totally integrated with Dymola, the whole installation procedure will not be needed anymore and it will be completely skipped.

Installing MOPS needs two files named *package.mo* and *mops.dll*. Called

[16]Graphical User Interface, that is a tool that eases the utilization of a software: in this case it also links the optimizer to Dymola

[17]we are using *Dymola 5.3a+*

path the directory where we have installed Dymola[18], the first file has to be placed in a new directory called *path**Modelica**Library**MOPS* and the second in *path**bin*. After having copied the files in their respective directories the last step for the installation of MOPS is to insert it in the Dymola Library Menu so to be able to call it from the program of modeling. To do this, we have to open the file *path**Insert**dymodraw.ini* and to add `optpackage MOPS MOPS` as explained in this comment taken from the file itself:

```
# ------------------------------------------------------------
# SETTING UP LIBRARY AND DEMO MENUS
# ------------------------------------------------------------
# Definitions for building the File/Libraries menu.
#
# Syntax for packages found in MODELICAPATH:
#
#   package <packagename> <description>
#
# For optional packages:
#
#   optpackage <packagename> description
```

Once restarted Dymola, we have to go in *Simulation Mode*[19] and to type `Hidden.Dymola6=true;` at the prompt to activate some hidden features that will be present in the new Dymola 6.

Now we are ready for the optimization:

1. Open the model (come back to the *Model Editing Mode*[20] and click on $File \rightarrow Open$, or, more quickly, type Ctrl+O and, finally, choose the file and click on the Open button)

2. Translate the model[21] (come back to the *Simulation Mode* and click on *Translate* button on the Toolbar)

[18] the default directory for a common Windows machine is *C:**Program Files**Dymola*

[19] click on the *Simulation* tab or Ctrl+F2 as well

[20] click on the *Modeling* tab or Ctrl+F1 as well

[21] the translation of the model is required for the selection of the parameters in the GUI

3. Load the MOPS Library (return to the *Model Editing Mode* and click on $File \rightarrow Libraries \rightarrow MOPS$)

4. Set up the optimizer (in Packages click on $CalibrateModel \rightarrow Call\ Function$ as seen in Figure 4.17)

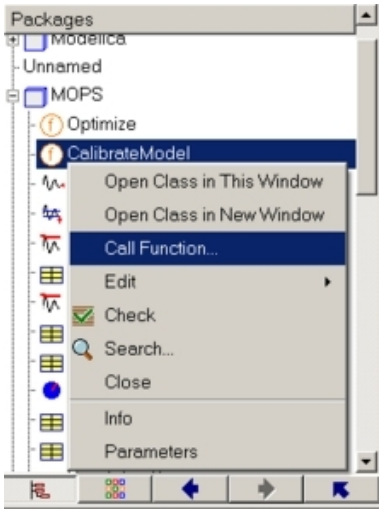


Figure 4.17: Calling MOPS

After all these steps, let us concentrate on the setup of MOPS. At the beginning, we find a dialog box as the one in Figure 4.18. We can see that the

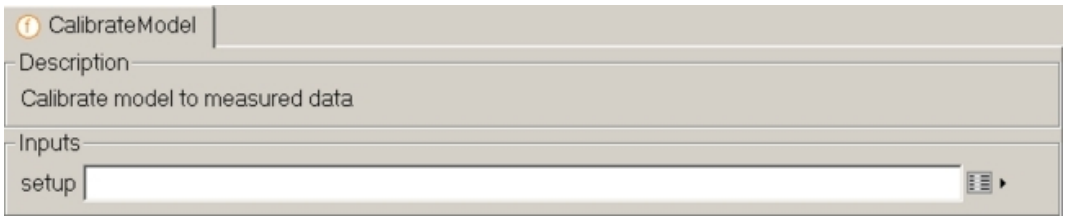


Figure 4.18: The initial dialog box of the MOPS GUI

function CalibrateModel has *setup* as unique argument. It would be possible to fill in this box directly by hand, inserting the right value of the argument, but it is very difficult because that argument is usually very long and complicated, so we are going to use the whole potentiality and easiness of the GUI.

Then, we have to click on the small grey square called *Edit* on the left of the text box, and the dialog box of Figure 4.19 will appear. This dialog box shows the most important aspects of the configuration of the optimizer: we must choose the parameters to optimize, the data which we are going to use for calibration and validation and finally we have also to configure the integrator and the optimizer about tolerances, precision, utilized algorithm, etc.

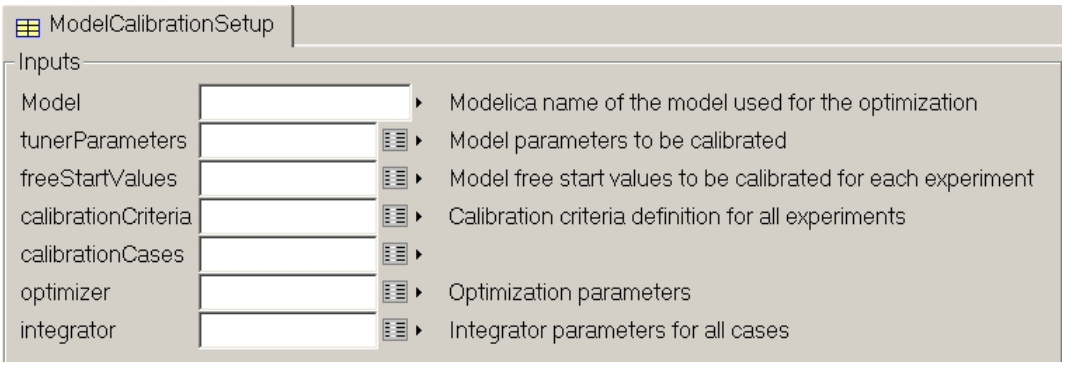


Figure 4.19: The main dialog box of the MOPS GUI

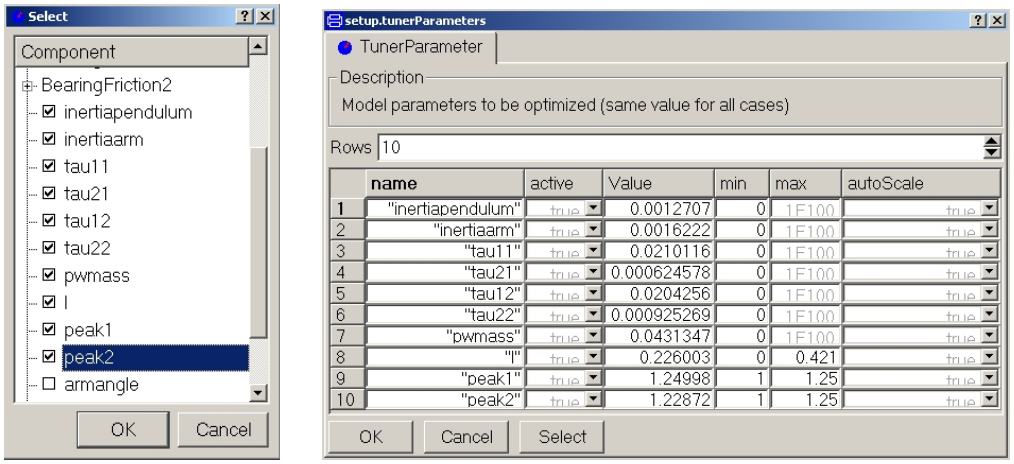
- **Model**: Insert the name of the model between quotations marks (")[22].
- **tunerParameters**: Choose the parameters that we want to optimize[23]. A window like the one in Figure 4.20(b) is opened, the table is now empty and we should click on the *Select* button to fill in it. After clicking on *Select*, a window like the one in Figure 4.20(a) appears: here, we can tick the check boxes to select the parameters to optimize[24]. Then our previous empty table becomes very similar to the one in Figure 4.20(b), except for the fact that, all the cells in the two columns called *min* and *max*, are set to their default value that is $\pm 1E100$ that here is used in place of $\pm\infty$. These two columns set respectively a lower and a higher

[22]it is very important, in Dymola, that every strings are inserted between quotation marks

[23]at the beginning, also the parameters that we want to optimize, must have a value from which the optimization can start

[24]this example refers to the first MultiBody Model that have been discussed in Chapter 4.4.3, so we have ten parameters to check off

bound on the values of the parameters in the quest for the optimum[25]. The first column, instead, represents the name of the parameters and the third, the initial value of those. The other two columns *active* and *autoScale* must be left unchanged.



(a) Window for the selection of the parameters to optimize

(b) Chosen parameters with constraints for the optimization

Figure 4.20: Selection of the parameters to optimize

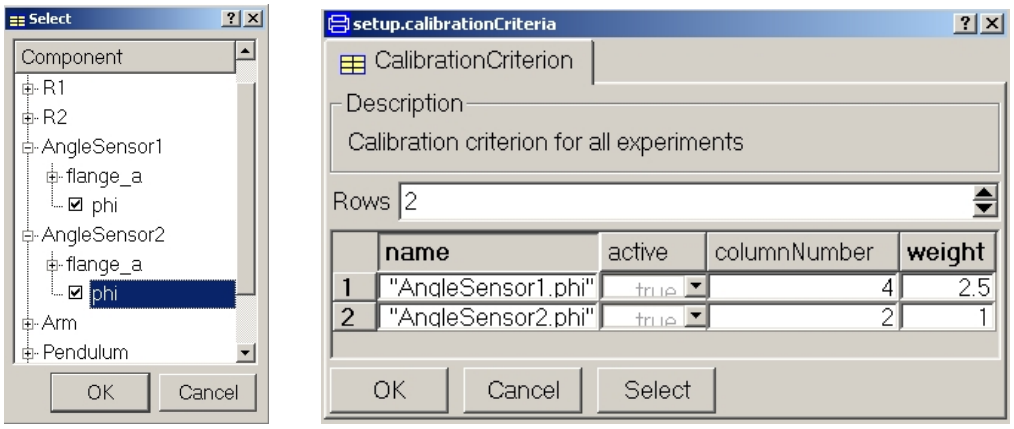
- **freeStartValues**: Select the initial conditions to estimate. In this way, Dymola tries to estimate also the initial conditions that fit better with the measured data.

- **calibrationCriteria**: Pick the variables for the criterion used in the calibration task. In this part, we choose the variables of the model that will be compared with the real data that are the ones which we can measure by means of our sensors[26], in fact our criterion is a weighted sum of the integrated square difference with respect to measured value for each variable. We have to fill in the table shown in Figure 4.21(b) by

[25] as seen in Chapters 4.4.3 and 4.4.4 all the parameters are positive and *l*, *peak1* and *peak2* have an upper bound

[26] in this case, these variables are the angles of the two pendula

ticking the chosen variables in the window reproduced in Figure 4.21(a)[27]. The first column represents the names of the variables, the third tells to Dymola in which columns of the matrix in the Matlab files to find the real measured data of the respective variable[28] and, finally, the fourth column shows the weights used for the calculation of the criterion[29].



(a) Window for the selection of the variables for the criterion

(b) Chosen variables with weights

Figure 4.21: Selection of the variables for the criterion

- **calibrationCases**: Select the set of data for calibration and validation. As we can see in Figure 4.22, this time there a lot of things to set:
  - **experimentNames**: In this dialog we tell to MOPS in which Matlab files it can find the data of measurements: as we can see in Figure 4.23(a), we have chosen, this time, two files called respectively *HorLight.mat* and *Up3Light.mat*.

[27] here *AngleSensor*1.*phi* is the variable relative to the Arm angle, while *AngleSensor*2.*phi* is relative to the Pendulum angle

[28] these file will be indicated in the *calibrationCases* window as we will see in a while; we have also to recall, as said in Chapter 4.2.2 that the Matlab files must contain a matrix called *Data* (with the first letter in upper case)

[29] here the Arm variable has a weight of 2.5 because the range of it is about 2.5 times littler than the Pendulum variable

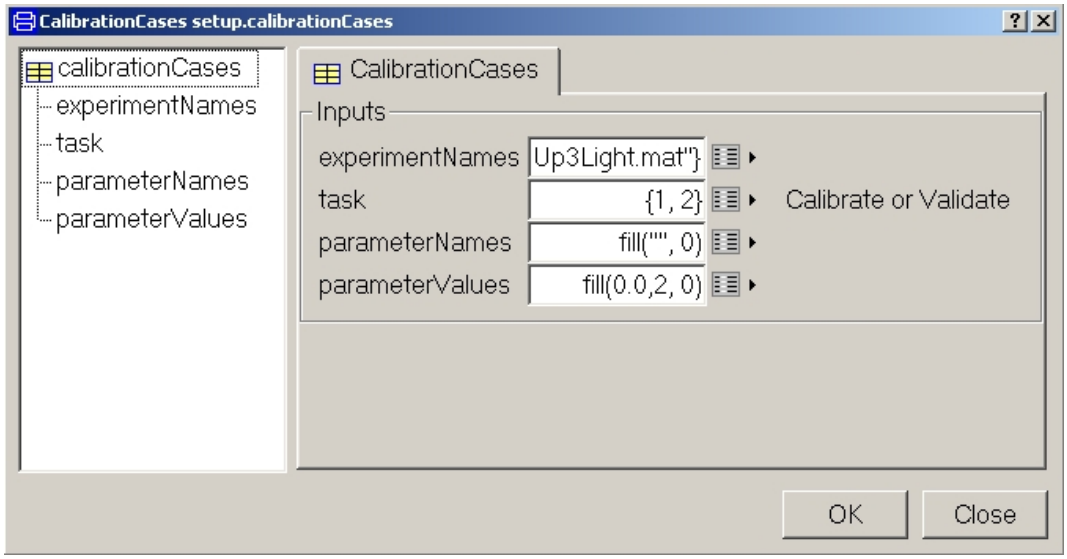
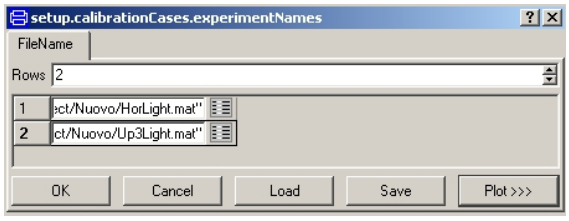
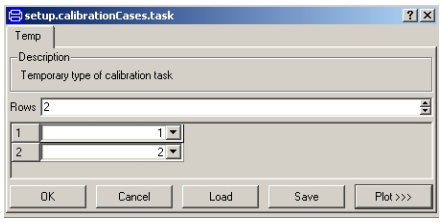


Figure 4.22: Dialog box for calibrationCases

– **task**: Here we choose which files, of the one picked before, to take for calibration or for validation. In Figure 4.23(b), we see that the file *HorLight.mat* has been used for Calibration[30], while *Up3Light.mat* has been utilized for Validation[31].



(a) Window for the selection of files to use for calibration and validation



(b) Window for the assignment of tasks to the files

Figure 4.23: Selection of the files for calibration and validation

– **parameterNames**: In this dialog we choose the parameters which we change the value to, through the experiments that are represented by the different Matlab files. This feature has not been used

[30] *1* in the vector of Figure 4.23(b) means *Calibration*
[31] *2* in the vector of Figure 4.23(b) means *Validation*

in this work so we have had to create an "empty vector" with 0 rows: the vector, to understand, is actually a matrix 0×0, that has neither rows nor columns.

- **parameterValues**: The same matter as in the previous paragraph: here we have to assign values to the parameters taken before. Since we have two experiments, but we have not chosen any parameters, we must create an "empty matrix" with 2 rows and 0 columns[32].
- **Optimizer**: This window allows to change the advance setting of the optimizer. For our work, the only parameters that we could want to change are *tolerance* and *maxEval*[33] that are fundamental for the end of the optimization. Of course to obtain better results it is advisable to take a smaller tolerance. It is also best to choose a large value for the maximum number of evaluations of the criteria function not to risk to terminate the optimization before having reached the optimum: we should say that there is actually the possibility that MOPS stops anyway, so it is advisable restarting the optimization from the parameter values found in the last run optimization and repeating this procedure until these values remain the same.
- **Integrator**: Here we can change settings for the integrator. Also in this case, we have left all the parameters to their default value except *startTime* and *stopTime* which are the starting and ending times of the simulation. Since we have taken only the first three seconds of our data, in our case, *startTime* is 0 and *stopTime* is 3.

4.5.2 Results for the First MultiBody Model

In this section we are going to show the results obtained with the use of the First MultiBody Model.

[32] 2 is the number of the experiments, 0 is the number of the parameters taken, in fact we do not have to assign any values

[33] this is the maximum number of evaluations of the criteria function: once reached this number Mops stops

In Figures 4.24 we can see the results of the Calibration: the graphs have the same scale for a correct comparison. Results are quite good for both the angles, especially for the pendulum one (Figure 4.24(b)); the identification of the arm behaviours is, instead, slightly more imprecise (Figure 4.24(a)). However, at the end, the important characteristics, the ones interesting for Automation, that are period and amplitude, are well identified as we can see in the graphs.

So, as we can see, the model fits measured data quite correctly. This, of course, happens almost every time, unless the model is completely wrong, since the model has been calibrated on those data. The real identification problem is to find a model that works well also with data which it has not been calibrated on. So we have to do a sort of control about that: we must check the validity of our model on other sets of data. There are a lot of way to validate a model, the one used in this work, the easiest, is to check the behaviour of the model working on another set of data, directly looking at the resulting graphs: if the model still fits the data quite well it means that the model is good, otherwise we have to change it.
In Figures 4.25 we can see the behaviour of the model on the validation data. Results are good since, also in this case, period and amplitude have been identified quite well.

We need to know, that the pendulum angle is measured with a potentiometer which has a small dead-zone: its effects can be clearly seen in Figure 4.25(b). This dead-zone has not been considered in our model, in fact, in the same figure, we can notice that the irregularity of the data trend disappears in the trend of the model.

Table 4.5 shows the values that have been found with calibration.

4.5.3 Results for the Second MultiBody Model

We are going to analyze now the behaviour of the second model. In this model there is a slightly more complex modelization of the friction and the addition of a new joint with a spring and a damper for modeling the vibrations that we can see when the two pendula move. Thus this model has more parameters than the other one, and then we expect a better fit to the data. The result can

1st MultiBody Model	
Device Parameters	
Parameter	*Value*
Pendulum Inertia $[kg \cdot m^2]$	0.000464
Arm Inertia $[kg \cdot m^2]$	0.000227
Pendulum and Weight Mass $[kg]$	0.015102
Pendulum Center of Mass $[m]$	0.170544
Friction Parameters	
tau11 $[N \cdot m]$	0.005485
tau21 $[N \cdot m]$	0.001670
tau12 $[N \cdot m]$	0.005420
tau22 $[N \cdot m]$	0.001443
peak1	1.249996
peak2	1.228720
Initial Conditions for Calibration	
Arm Angle $[rad]$	-0.071575
Arm Speed $[rad/s]$	-0.083589
Pendulum Angle $[rad]$	1.671170
Pendulum Speed $[rad/s]$	1.870203
Initial Conditions for Validation	
Arm Angle $[rad]$	-0.100236
Arm Speed $[rad/s]$	-1.620246
Pendulum Angle $[rad]$	-1.940164
Pendulum Speed $[rad/s]$	-7.661688
Criteria Function Value (Error)	0.014406

Table 4.5: Parameters of the First MultiBody Model

be seen in Figures 4.26. As anticipated the fit is almost perfect and the model response can almost be overlapped to the calibration data as for the arm as for the pendulum.

A bigger number of parameters brings to a better calibration but in addition we can have some problems with validation: in trying to follow also the behaviour of the validation data, the fitting curve can have peculiar oscilla-

tions that are sometimes very noticeable and make the model unemployable. Thus it is often not recommended to use models with too much parameters. Let us see then, if this is our situation, that is let us see if the addition of these new parameters has implied troubles in the validation. In Figures 4.27 we can see what we have obtained: there are some little oscillations in the Arm graph (Figure 4.27(a)), but the fit is very good also with this set of data.

In Table 4.6 instead, we can see the values of the parameters that we have found through the calibration.

4.5.4 Comparison between the two Models

In this section we will make a short comparison of the results obtained from the two models: this will show us some important aspects of this identification.

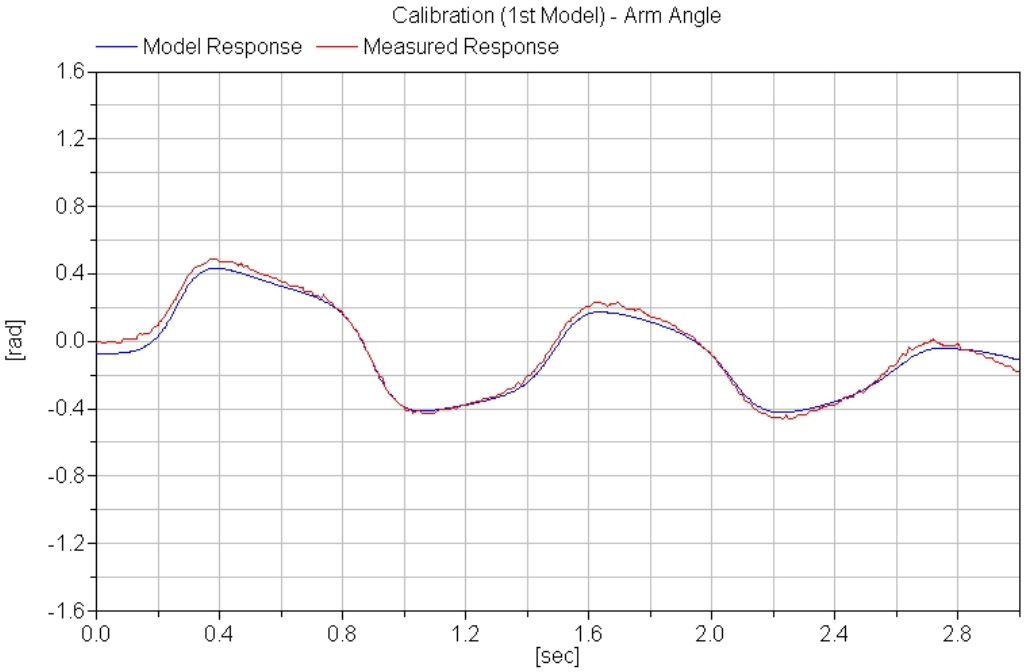
Looking at the Tables 4.5 and 4.6 we can see that the values of the parameters that we have found are quite different in the two cases. These values are also different from those shown at Page 36 which would be the real values. This fact has been already explained in Chapter 3.2, the one on Identifiability: the analyzed model has not this property, so there are several values of the parameter vector θ which produce the same minimum value of the criteria function. But, as already said in that chapter, the identification of an usable model does not always require structural identifiability, so, since we have obtained good results as in the calibration as in the validation, we can use our models without any problems.

Another interesting thing to know is why the second model works better. Observing the device moving we can see, as soon as we release the pendulum, a lot of vibrations that decreased more and more as the experiment proceeds. The idea of modeling these vibrations with a new joint linked to a spring and to a damper comes natural.

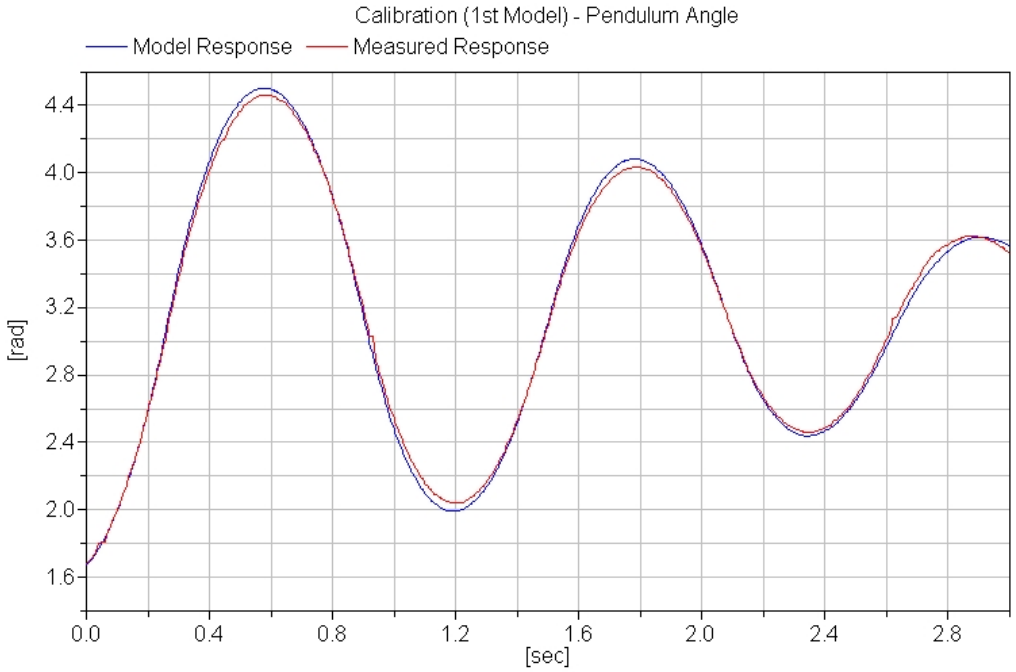
Looking then at the trend of the frictional torque in the first model (Figure 4.28(a)), we can notice that this friction as a quite particular trend, in fact it decreases when the angular velocity increases. Thus we have decide to slightly change our model passing from a linear dependence with respect of angular velocity to a piece wise linear one. The effect of this change can be seen in Figure 4.28(b) where it is shown that, in this case, frictional torque increases

when also angular velocity increases.

Thus, at the end, we can say that it is better to utilize the second model since it gives better results and a more faithful description of the behaviour of the device. However, we have to say that also the other model behaves well and it can be used as well.

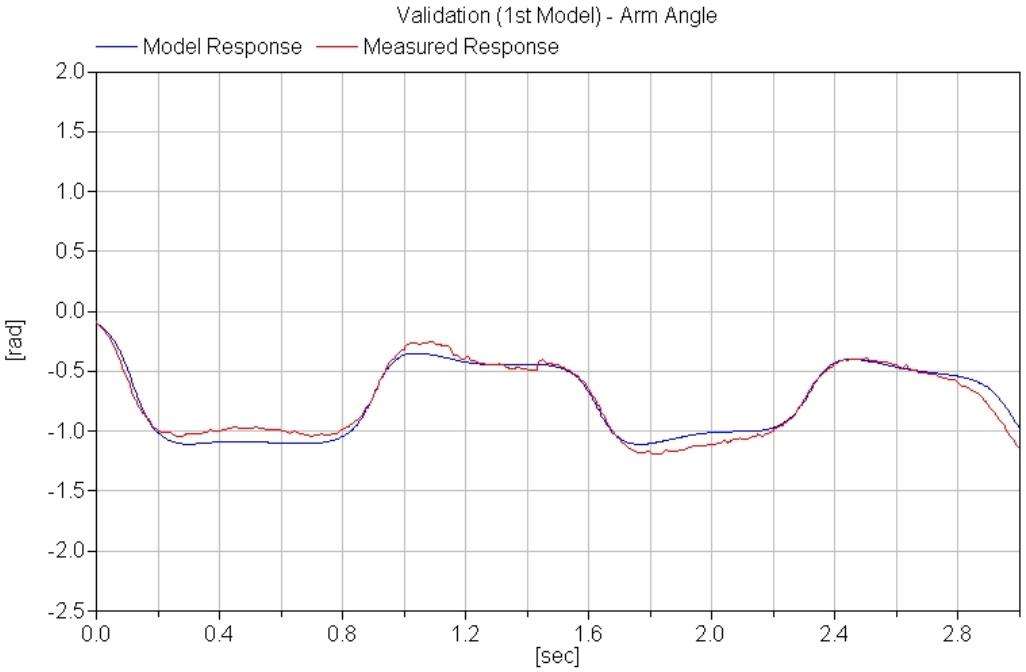


(a) Arm Angle

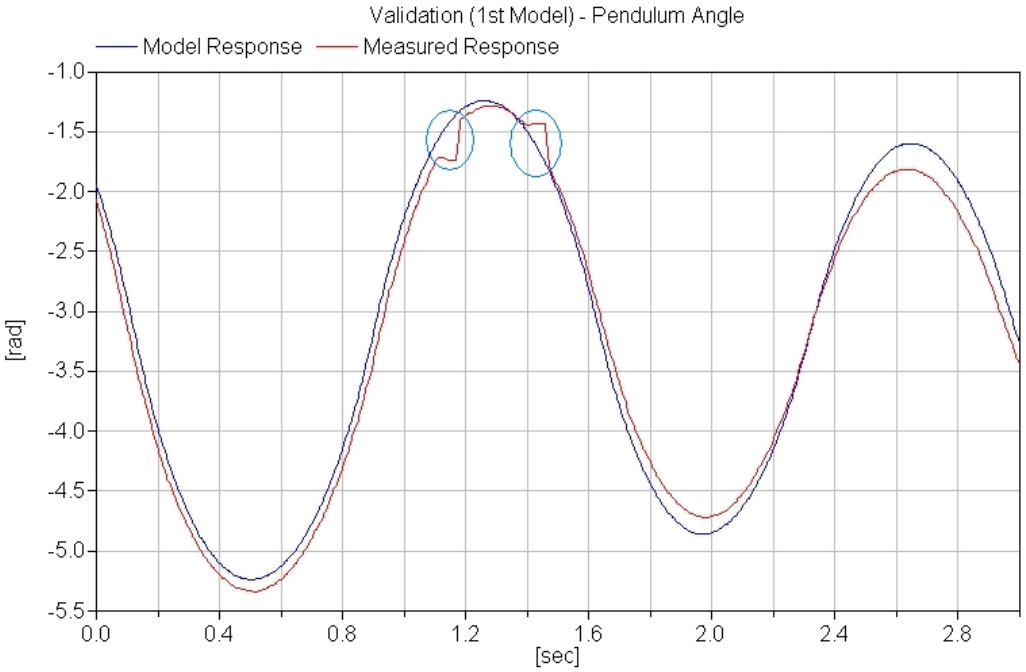


(b) Pendulum Angle

Figure 4.24: Calibration of the First MultiBody Model

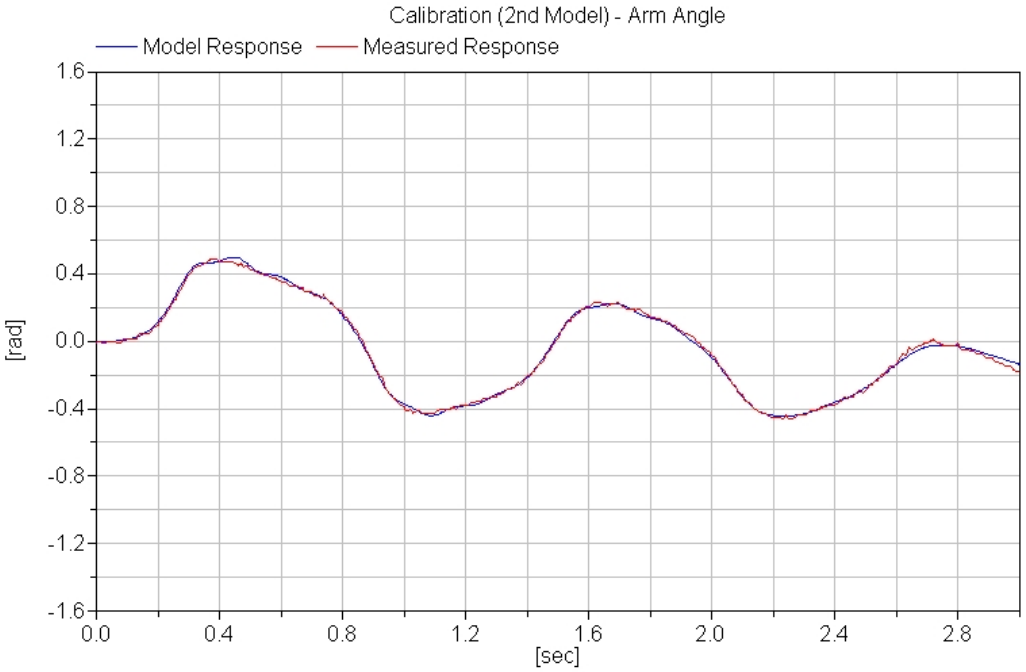


(a) Arm Angle

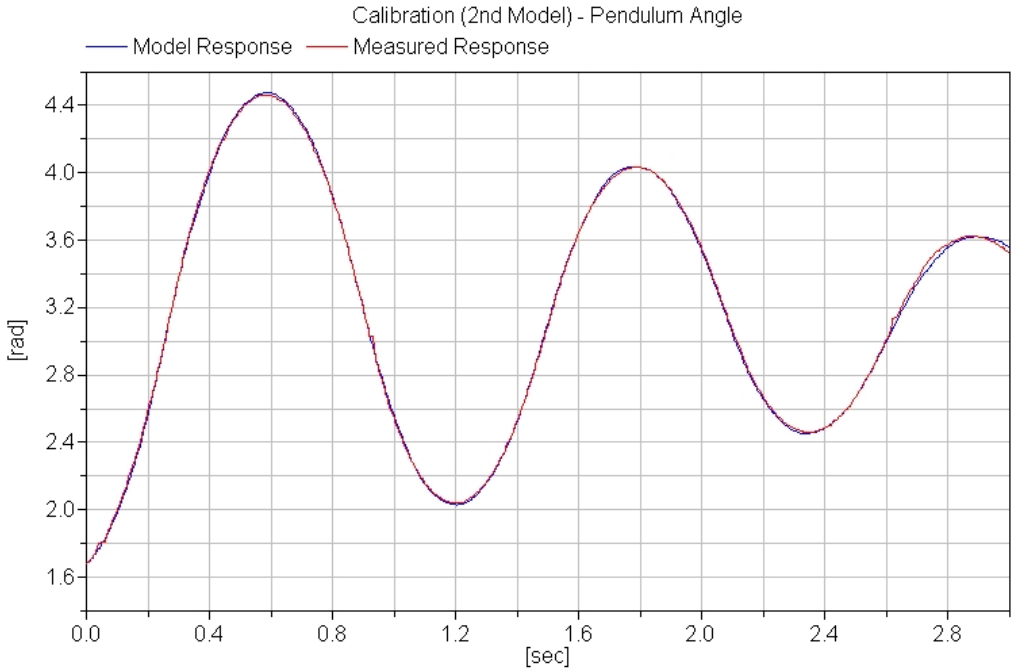


(b) Pendulum Angle. We can notice the effects of the dead-zone discussed in the text

Figure 4.25: Validation of the First MultiBody Model

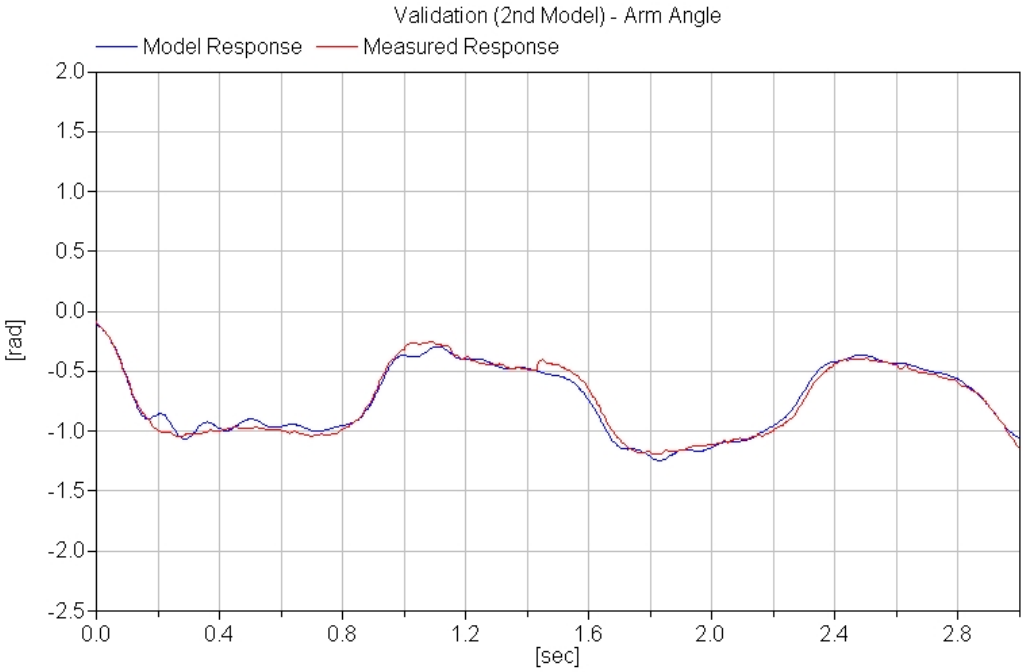


(a) Arm Angle

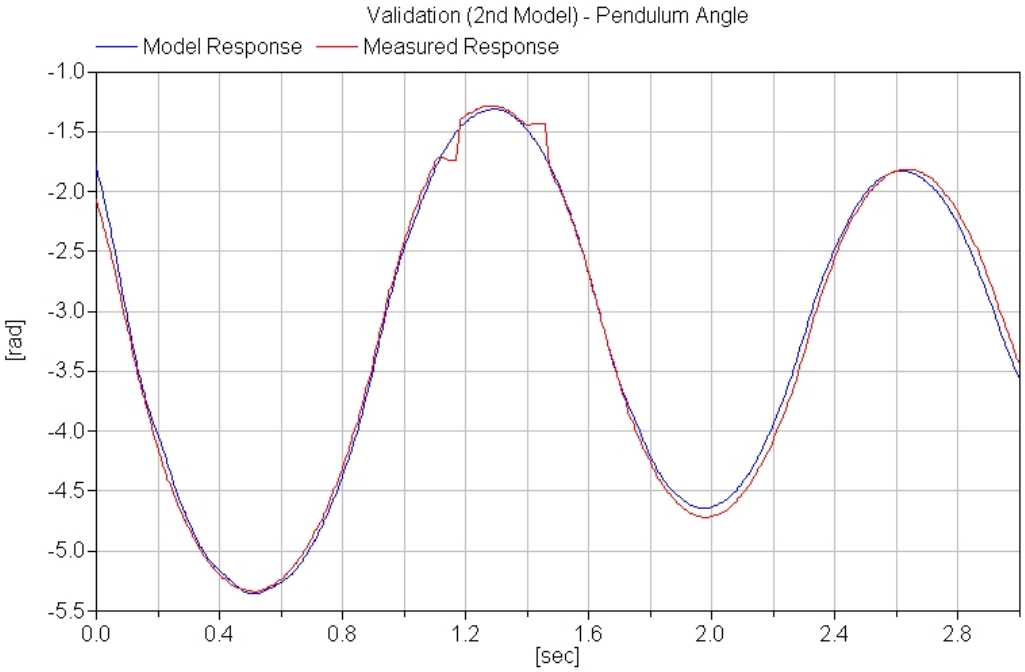


(b) Pendulum Angle

Figure 4.26: Calibration of the Second MultiBody Model



(a) Arm Angle

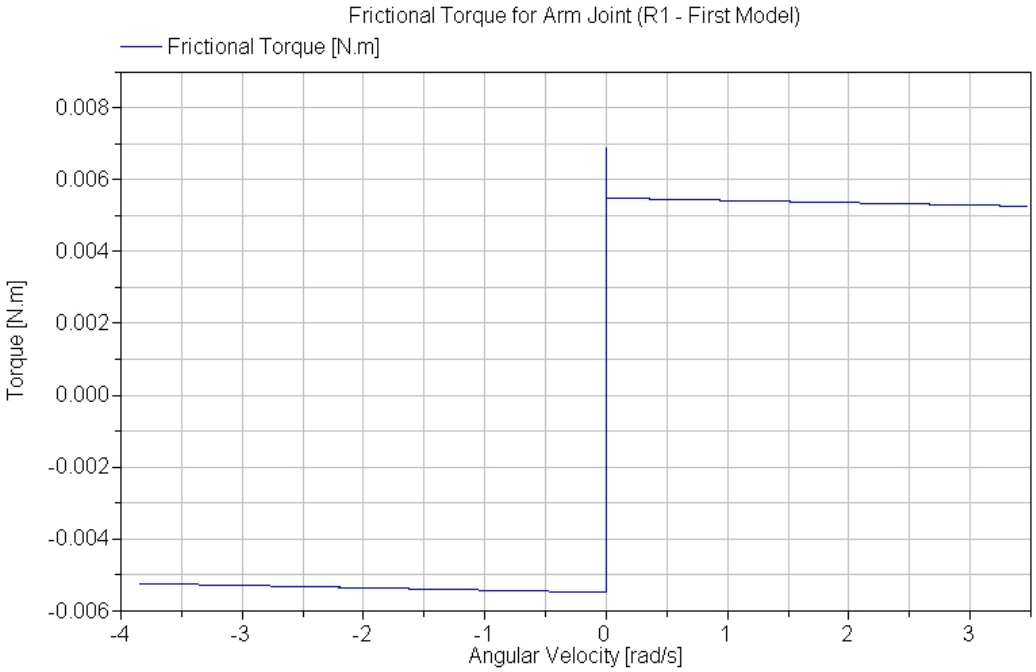


(b) Pendulum Angle. We can notice the effects of the dead-zone discussed in the previous section

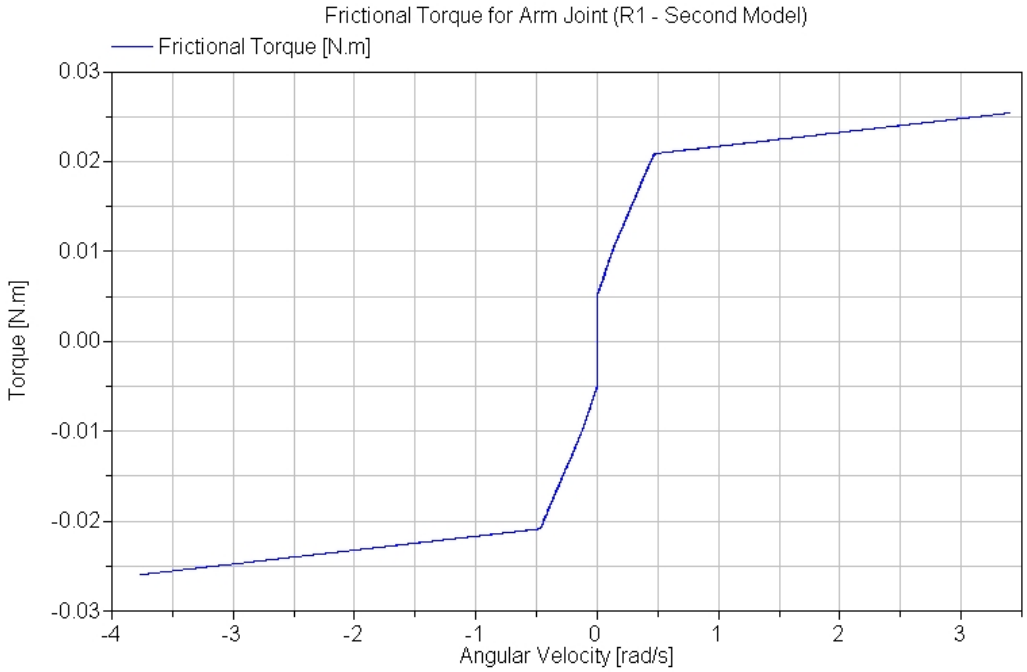
Figure 4.27: Validation of the Second MultiBody Model

2nd MultiBody Model	
Device Parameters	
Parameter	*Value*
Pendulum Inertia $[kg \cdot m^2]$	0.001082
Arm Inertia $[kg \cdot m^2]$	0.001656
Pendulum and Weight Mass $[kg]$	0.040414
Pendulum Center of Mass $[m]$	0.238454
Friction Parameters	
tau11 $[N \cdot m]$	0.005055
tau21 $[N \cdot m]$	0.004626
tau12 $[N \cdot m]$	0.034065
tau22 $[N \cdot m]$	0.003449
tau01 $[N \cdot m]$	0.010220
tau05 $[N \cdot m]$	0.020867
point0	0.130807
point1	0.468552
point2	9.011384
peak1	1.000119
peak2	1.228720
Spring Constant $[N \cdot m/rad]$	3.81909
Damping Constant $[N \cdot m \cdot s/rad]$	0.010007
Initial Conditions for Calibration	
Arm Angle $[rad]$	-0.002626
Arm Speed $[rad/s]$	-0.0362232
Pendulum Angle $[rad]$	1.677478
Pendulum Speed $[rad/s]$	1.640014
Initial Conditions for Validation	
Arm Angle $[rad]$	-0.106993
Arm Speed $[rad/s]$	-1.340885
Pendulum Angle $[rad]$	-1.785922
Pendulum Speed $[rad/s]$	-10.443861
Criteria Function Value (Error)	0.002380

Table 4.6: Parameters of the Second MultiBody Model



(a) Frictional Torque for Arm of the First Model



(b) Frictional Torque for Arm of the Second Model

Figure 4.28: Comparison between the Frictional Torques of the two Models

Chapter 5

Control Design

This chapter will show us how the interface between Dymola and MOPS can be conveniently used for *Control Design*.

The idea is to choose the layout of the controller, forcing some constraints on the system response and, finally, letting the computer find the correct value of the controller parameters that meets the imposed requests. This way of designing is very similar to the previous task performed in this work, that is Identification: imposing some constraints and finding the right values of the parameters that satisfy them through optimization, it is approximately the same of the previously performed Greybox Identification, where the minimum difference between the behaviours of the system and the model was found by MOPS.

The chapter will begin with the application of this feature to a simple linear system controlled by a PID and it will continue with the control of our case study.

5.1 Overview

5.1.1 Simple Linear System

The system that we are going to analyze in this section is the linear system described by the following transfer function:

$$G(s) = \frac{1}{s(s+1)(s+4)}$$

The response of this system to a *Step* as input, without any kinds of controller, is shown in Figure 5.1: the system response grows unbounded due to the pole in $s = 0$.

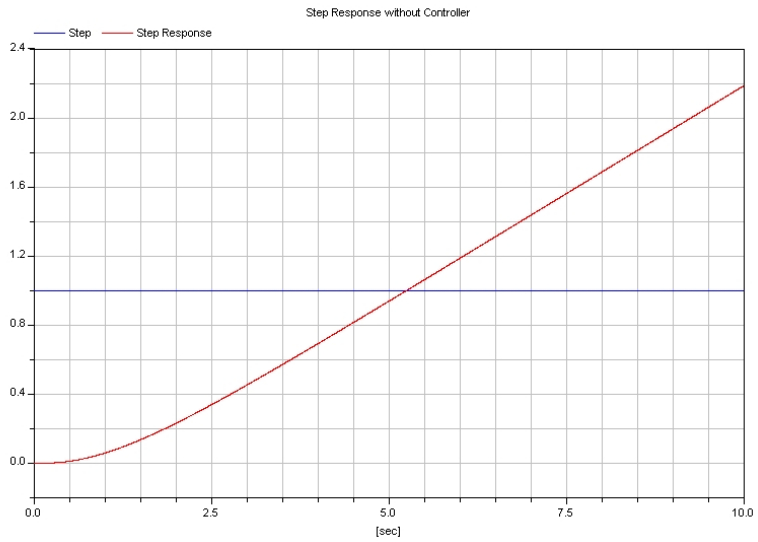


Figure 5.1: Step Response of the System without Controller

Let us apply now a PID controller to the system. Setting the following initial values for the three parameters of the PID, we obtain the Step Response shown in Figure 5.2:

k	4	Gain
Ti	2 s	Time constant of Integrator
Td	1 s	Time constant of Derivative block

As we can see in that picture, the system is now stable and it follows the reference signal.
We have however, in this case, a quite large value of the *Overshoot*[1] (over 30%) and the response in general could be improved.
It is in this situation that the new design feature of Dymola can be useful to an engineer: instead of a lot of calculations which can be often long and complicated, computer can quickly solve the design problem in place of him.

[1]the difference, in percentage, between the maximum value of the response and its asymptotic response

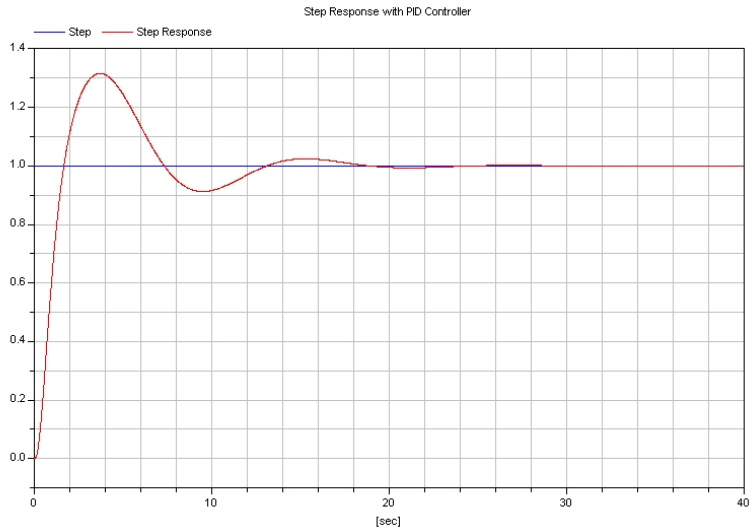


Figure 5.2: Step Response of the System with a non-optimized PID Controller

In the following sections we are going to show how this task could be done.

5.1.2 Multi-Criteria Design

In this section we want to improve the step response obtained in the previous section.

Let us impose, then, some constraints on this response. We can suppose, for example, to want a lower value of the *Overshoot* and a certain value for the *Settling Time*[2].
Our requests could be the following:

- $Overshoot = 10\%$
- $Settling\ Time\ (1\%) = 15s$

This is a typical example of *Multi-Criteria Design* that is the design of a controller with the observance of some constraints or *Criteria* (in this case Overshoot and Settling Time). Such a problem can be easily solved by Dymola

[2]the time after which the response definitively remains in a range around the asymptotic response smaller than a certain prefixed value

adding some components for criteria calculation to the system diagram, as we can see in Figure 5.3.

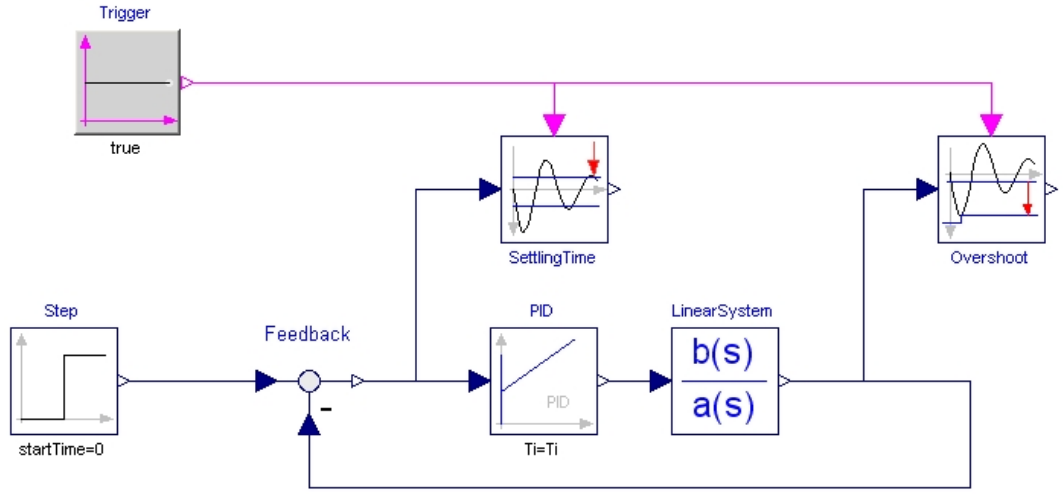


Figure 5.3: System Diagram with PID Controller and Criteria Calculation Components

In this figure the Linear System is represented by the Transfer Function block[3] called *LinearSystem* which is controlled through feedback by a PID[4] to manage to follow the Step that we have chosen as input signal.
The two blocks called *Overshoot*[5] and *SettlingTime*[6] are the ones which will permit to Dymola to find the right values of the parameters of the PID to satisfy our requests: we need to notice that these two blocks must be activated by the block named *"Trigger"*.

The operations to perform this computer aided design are pretty the same of those explained in Section 4.5.1:

1. Open the model

2. Translate the model

[3]the Modelica class used is Modelica.Blocks.Continuous.TransferFunction (see Appendix B.7)

[4]the Modelica class used is Modelica.Blocks.Continuous.PID (see Appendix B.8)

[5]the Modelica class used is Design.Criteria.Overshoot (see Appendix B.9)

[6]the Modelica class used is Design.Criteria.SettlingTime (see Appendix B.10)

3. Load the Design Library

4. Set up the optimizer (in Packages click on *optimize* → *Call Function*)

Also the optimizer setup is very similar to the one done in the section mentioned above. The GUI, in this case, appears as in Figure 5.4.

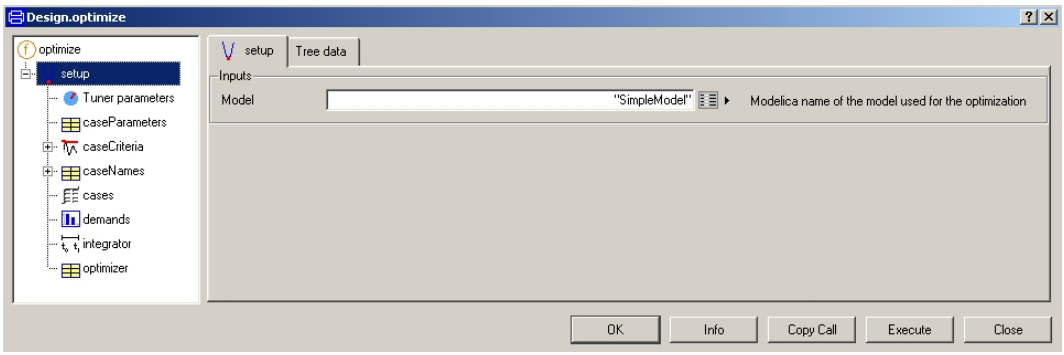


Figure 5.4: The initial dialog box of the Optimizer GUI

For a simple Multi-Criteria Design the only interesting settings to change are the following:

- **Setup**: Insert the name of the model between quotation marks (").

- **Tuner Parameters**: Choose the parameters that we want to optimize. The choice is made through a form in which we have to select the parameters of the controller to find. To comply with physical reasons or simply to ease the task of the optimizer, it is also possibile to impose a range for the values of the parameters.

- **CaseCriteria**: Select the criteria (or constraints) which have to be satisfied. To notice that these criteria can be chosen from those offered with Dymola (the most common and useful ones, such as *Overshoot*, *Settling Time*, *Rise Time*, etc...) or they can be conceived by the engineer himself. This selection is made through the dialog box in Figure 5.5: we have to fill the *name* box with the name of the variable which is our criterium and then we must select the appropriate analysis rule of this variable in

the *criteria* menu[7]. About the *criterionUsage*, we can decide to minimize the value of the variable or to make it equal (equality constraint) to, or smaller (inequality constraint) than a certain requested value.

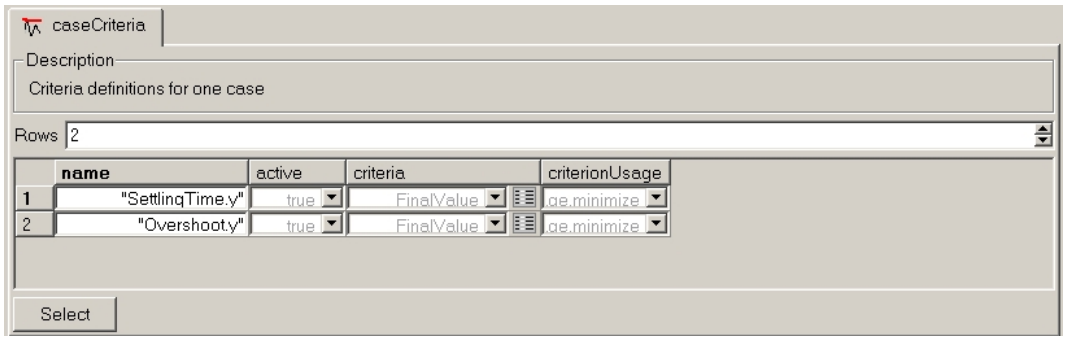


Figure 5.5: Dialog box for the selection of the Criteria

- **demands**: Impose the request value of the criteria i.e. the constraints (Figure 5.6).

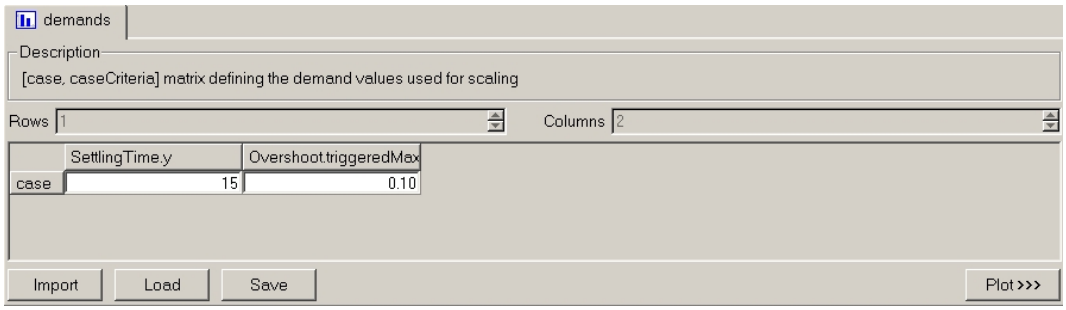


Figure 5.6: Dialog box for imposing constraints

Returning to our requests, we have chosen to optimize the three PID parameters i.e. the gain k, the time constant of integrator Ti and the time constant of the derivative block Td.

[7]it depends from how the criterium is calculated. For example, in the case of Overshoot and Settling Time blocks, we have the correct values of these quantities only at the end of the simulations, so *"FinalValue"* is needed

Then we have started the optimization imposing our constraints and deciding to minimize the value of the Overshoot and Settling Time.

It is important to know that, during the optimization, the value of the criteria is scaled with its demand value so that every criteria are correctly weighted and, at the end, we obtain a solution which has the characteristic that no criterium can be further minimized without degrading at least one other criterium (*Pareto-Optimal Solution*).

So our task is solving the following *min-max problem*:

$$\min_{k,Ti,Td} \left(\max \left(\frac{Overshoot}{0.1}, \frac{Settling\ Time}{15} \right) \right)$$

The results reached are summarized in Table 5.1.

Tuner Parameters		**Criteria**		
Name	**Value**	**Name**	**Value**	**Scaled Value**
k	5.48078	Overshoot	0.0863809	0.863809
Ti $[s]$	4.6697	Settling Time (1%)	12.9538 s	0.863584
Td $[s]$	1.41883			

Table 5.1: Parameters and Results reached with minimization of both criteria

We can see that both criteria widely satisfy our requests and that the scaled values of those are nearly the same (within computational accuracy): we would have had to expect this last fact due to the attainment, with the optimization, of a Pareto-Optimal solution.

Let us suppose now, that we are satisfied by a settling time of $15s$ but we would want to improve the overshoot as much as possible. We must then change our requests, setting inequality for rising time and letting minimization for the overshoot. In this situation, our problem becomes:

$$\min_{k,Ti,Td} \left(\max \left(\frac{Overshoot}{0.1} \right) \right) \text{ subject to } \frac{Settling\ Time}{15} \leq 1$$

The new results can be read in Table 5.2, while in Figure 5.7 it is shown a graphical visualization of them. Settling time has now reached its maximum allowed value (in fact the scaled value is 1) and the overshoot is approximately improved of 15% with respect to the previous optimization.

Tuner Parameters		Criteria		
Name	**Value**	**Name**	**Value**	**Scaled Value**
k	5.52434	Overshoot	0.0736137	0.736137
Ti [s]	5.93807	Settling Time (1%)	15 s	1
Td [s]	1.34739			

Table 5.2: Parameters and Results reached with minimization of Overshoot and inequality for Settling Time

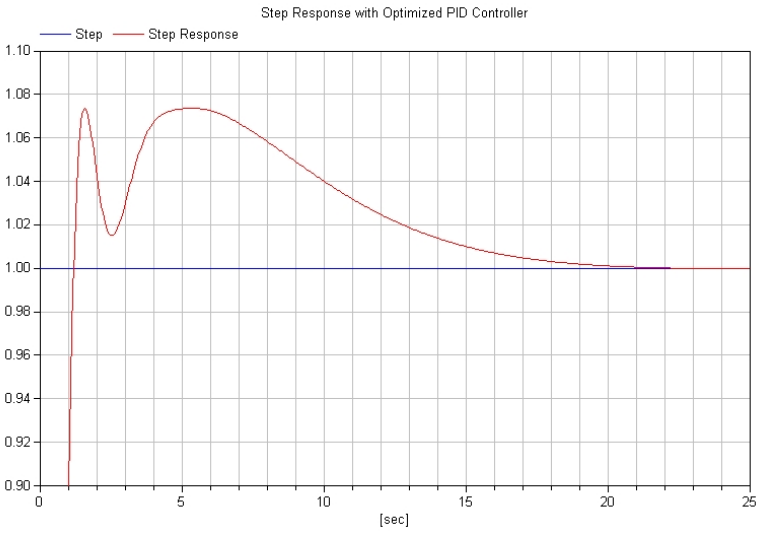


Figure 5.7: Step Response of the System with an optimized PID Controller

5.1.3 Multi-Case Optimization

Let us speak, in this section, about Multi-Case Optimization.
As we can understand by the name, the optimization is performed on various cases. This feature is very convenient to design a controller that can work in different situations.

The parameters of a system are usually known in a range of values. If this range is not too large and so the characteristics of the system do not vary appreciably, we can try to use the optimizer to find a controller that can satisfy our requests in the whole range.

For example, we could write our linear system in the form

$$G(s) = \frac{1}{(s-a)(s-b)(s-c)}$$

and say that its parameters (its poles) are real (for simplicity) and stay in the following ranges:

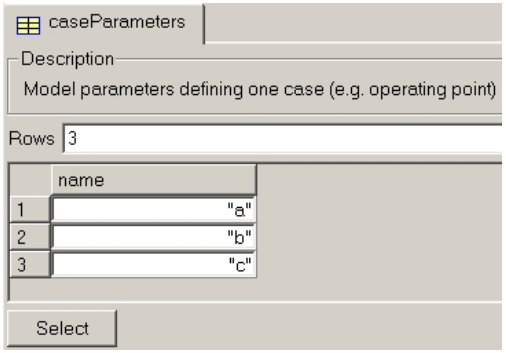
$$a = 0 \pm 0.2$$
$$b = -4 \pm 0.8$$
$$c = -1 \pm 2$$

We need to notice that the real part of the first pole can become positive, increasing the instability of the system. Furthermore, running some simulations, we have found that we obtain the maximum value of the Settling Time ($23.37s$) for $[a = 0.2, b = -4.8, c = -1.2]$ and the maximum of Overshoot (39.46%) for $[a = 0.2, b = -3.2, c = -0.8]$.

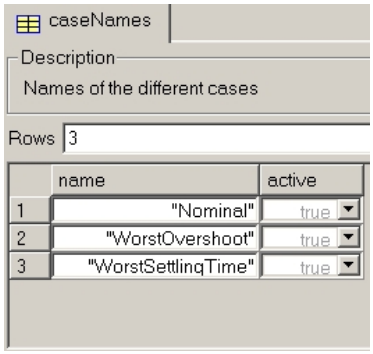
Repeating the steps explained in the previous section and changing the settings of the optimization as shown in Figure 5.8, Dymola tries to find a controller that satisfies our requests.
In Figure 5.8(a) is shown the dialog box for selecting the parameters that vary; in the dialog box of Figure 5.8(b) we have to choose the name of the different cases, that will be defined as in Figure 5.8(c). Finally the constraints will be imposed in a dialog box like the one in Figure 5.8(d).

So we have set the two worst cases *"WorstSettlingTime"* and *"WorstOvershoot"* and the nominal one *"Nominal"* and then, set the same constraints for all of them. We have had also to increase the maximum number of function evaluations in the settings of the optimizer, to allow the optimizer to finish its task: actually it finds the optimum after a few iterations, but it needs more than 1400 to verify that. Thus, at the end, we have obtained the results shown in Table 5.3.

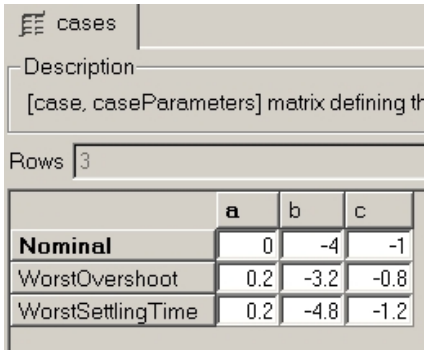
For this optimization we have decide to simply respect the Settling Time constraint and to minimize the value of the Overshoot. Looking at the table, we discover that our request for Settling Time has been satisfied for all the three cases, while the Overshoot is near to our request only in the nominal case, in the other cases, instead, it remains quite large. The same results are



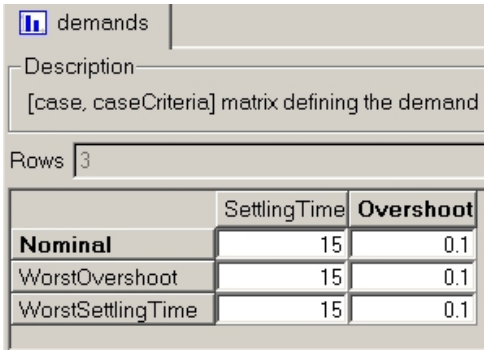
(a) Name of the case parameters



(b) Name of the cases



(c) Definition of the cases



(d) Demands for the different cases

Figure 5.8: Dialog Boxes of Multi-Case Optimization

Tuner Param.		**Criteria**			
Name	**Value**	**Name**	**Nominal**	**Settling T.**	**Overshoot**
k	5.81163	Overshoot	0.105139	0.307592	0.410065
Ti $[s]$	6.53286	Settling T. (1%)	14.9438 s	14.6882 s	14.4761 s
Td $[s]$	0.931463				

Table 5.3: Parameters and Results reached with Multi-Case Optimization

graphically shown in Figure 5.9.
Since we have chosen minimization for Overshoot, it is important to notice that these results cannot be improved unless to reduce our request on Settling Time.

It is then up to the designer to decide if these results can be accepted or, otherwise, trying to reduce the constraint on the Settling Time and finding a

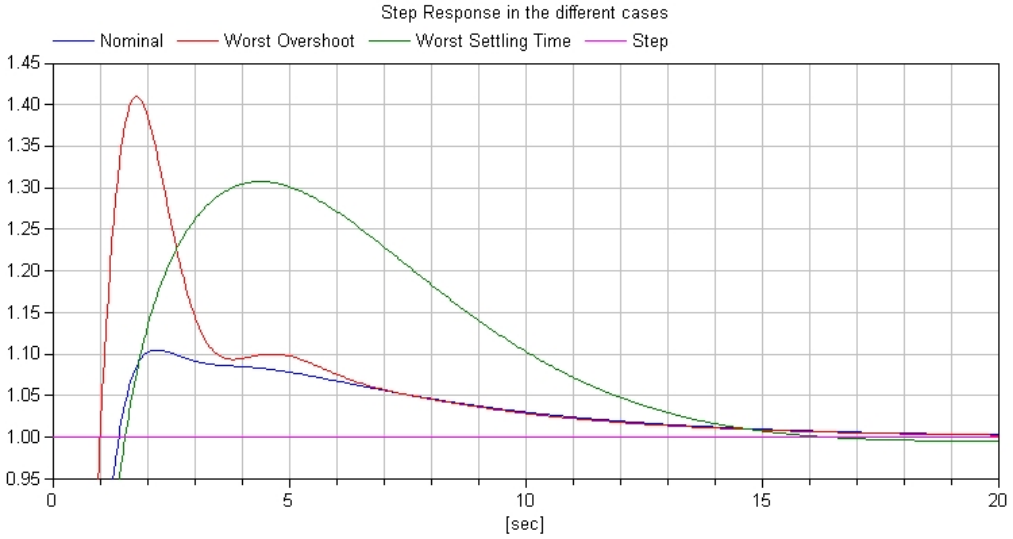


Figure 5.9: Step Response of the three cases

controller with a better performance for Overshoot.

5.2 Case Study

In this section we are going to apply the above analyzed feature to a simplified version of our case study.

The simplification regards the frictions applied to the joints[8]: we have replaced the two Bearing Friction components, which are discontinuous, with a simpler friction model, that is a Spring and Damper component. This change allows to linearize the system and then, to find a controller, using the *Pole-Placement* technique.
The parameters of the two new blocks have been found minimizing the difference between the behaviour of the two models with a process very similar to the one explained in Section 4.5.

Entering the Simulation Mode and selecting *Simulation* → *Linearize*, Dymola creates a Matlab file called *dslin.mat* containing the linearized model and the name and the number of the states of the model.

[8]the new model can be seen in Figure 5.10

In our case, called φ the arm angle, θ the pendulum angle and u the input applied to the arm, linearizing around the equilibrium point $(\varphi\ \dot{\varphi}\ \theta\ \dot{\theta}) = (0\ 0\ 0\ 0)$, we obtain the following linearized system:

$$\begin{pmatrix} \dot{\varphi} \\ \ddot{\varphi} \\ \dot{\theta} \\ \ddot{\theta} \end{pmatrix} = \begin{pmatrix} 0 & 1 & 0 & 0 \\ -0.1082 & 0 & -25.4814 & 0.0255 \\ 0 & 0 & 0 & 1 \\ 0.0756 & 0 & 45.7597 & -0.0458 \end{pmatrix} \begin{pmatrix} \varphi \\ \dot{\varphi} \\ \theta \\ \dot{\theta} \end{pmatrix} + \begin{pmatrix} 0 \\ 1444.1 \\ 0 \\ -1008.5 \end{pmatrix} u$$

Once in Matlab, with the command *place*, we can design a controller using the Pole-Placement technique. This approach, under some constraints on the characteristics of the system, permits to place the poles of the controlled system transfer function, wherever we want.

Choosing $(-5,\ -8,\ -10,\ -14)$ as poles, with the Matlab command `K=place(A,B,[-5,-8,-10,-14])`, we obtain a Pole-Placement controller defined by the vector $K = (-0.1387\ -0.0690\ -0.7345\ -0.1354)$. The choice of these poles is arbitrary, excluding the fact that we must have a quite fast controller to avoid the fall of the pendulum from the upright position. However, for the same reason, our controller must not be very strong, otherwise a too large input would produce the tumble of the pendulum.

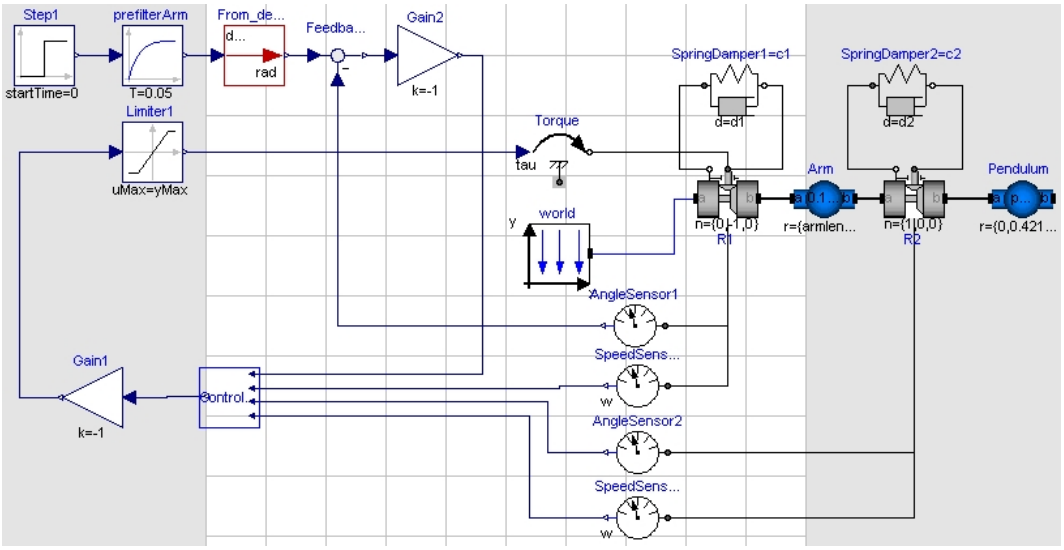
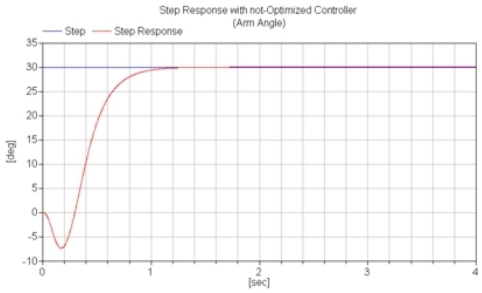


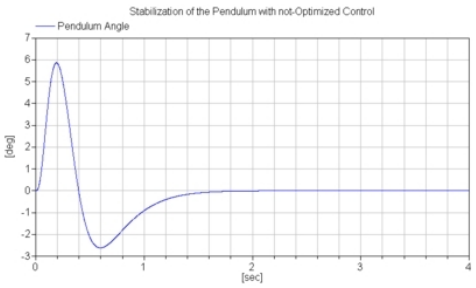
Figure 5.10: Simplified System with Pole-Placement Controller

Applying the controller[9] to our system as shown in Figure 5.10, we want to stabilize the pendulum in the upright position and drive the arm to a certain position (in degrees) selected changing the value of the Step in the picture.

We must notice that the step has been filtered by a block called *prefilterArm* that is a first order system attenuating the input not to let the pendulum fall down. For the same reason, also the torque given by the controller is limited to values between -0.05 and $0.05\ N \cdot m$.



(a) Step Response with non-Optimized Controller (Arm Angle)



(b) Stabilization of the Pendulum with non-Optimized Controller

Figure 5.11: Results with non-Optimized Controller

The results for this controller, shown in Figure 5.11, are quite good with a Rise Time[10] of 0.41 s and a Settling Time (1%) of 1.11 s for the Arm to perform a rotation of 30 degrees. Regarding the Pendulum, instead, we have an Overshoot of 5.86 deg with respect to the stabilization position that is 0 degrees. Even if the controller works, the stabilization of the Pendulum is the most critical problem in controlling this system: a too large oscillation could cause the fall of the pendulum itself; for this reason, thinking to apply the controller to the real system, it could be interesting to reduce the Overshoot value of the pendulum and so, the maximum amplitude of the oscillations of it around its instable equilibrium position.

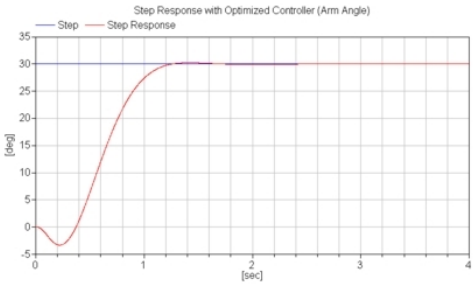
Not to degrading too much the arm performance, instead of choosing minimization, we have chosen to request an Overshoot value smaller than 3 deg (we want to reduce the previous obtained value by 50%). It must be noticed

[9] the controller is described in Appendix B.12

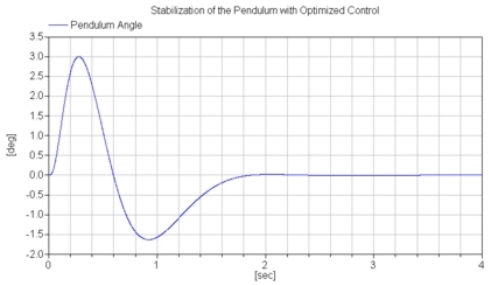
[10] the time it takes for the step response to grow from 10% to 90% of its final asymptotic value

that some conditions on the angle response must be obligatorily imposed, otherwise the optimizer would find a solution like $K = (0\ 0\ k_3\ k_4)$ that respects the request on the Overshoot value, but that does not let the arm move.

So imposing a maximum value of 3 *deg* for Pendulum Overshoot and choosing to minimize Rise and Settling Times of the arm we obtain the results shown in Figure 5.12 and summarized in Table 5.4.



(a) Step Response with Optimized Controller (Arm Angle)



(b) Stabilization of the Pendulum with Optimized Controller

Figure 5.12: Results with Optimized Controller

Tuner Parameters		**Criteria**	
Name	**Value**	**Name**	**Value**
k_1	-0.0330268	Overshoot (Pendulum)	3
k_2	-0.0218451	Settling Time (Arm - 1%)	1.2 s
k_3	-0.318308	Rise Time (Arm)	0.54 s
k_4	-0.0614688		

Table 5.4: Parameters and Results reached with Optimization

As we can read in the previous table, the Settling Time of the arm is increased of 8% and its Rise Time of 35%, while the pendulum Overshoot is halved, as requested: these results can be accepted since the degradation of the arm performance is quite small.

Chapter 6

Conclusions

In this chapter we are going to comment the results obtained with this work regarding identification and control design using the software Dymola.

6.1 Comments

The main purpose of this thesis has been performing Greybox Identification. Greybox modeling is very relevant when we are working on a system whose inner structure and physical relations are well-known, allowing the designer to exploit some *a priori* information that, of course, help in the design itself and are really useful to obtain better identification results.

The main topic of this work is however performing the two tasks of Calibration and Validation through a new feature of the modeling software Dymola that can be interfaced to the optimization tool called MOPS with an easy to use GUI.

Dymola is a very convenient software for modeling, since it permits the utilization of the whole power and features of the modeling language Modelica, without requiring the user to know such language. This fact really makes the task of designing quicker and feasible by almost anyone. The model can, of course, be simulated through a very efficient Simulation Environment that allows the user to plot the values of the quantities regarding the model and even to see an animation representing the behaviours of the model itself: these characteristics are very useful since the designer can be helped in the valuation

of his model with just a visual inspection.

The newly introduced capacity, that is the interface to MOPS, allows now to perform Parameter Estimation and Design Optimization of parameterized controllers, directly in the Dymola environment making this software a complete tool for the whole designing process.
In this work all the steps of the Parameter Estimation and Control Design are illustrated in details through the analysis of a case study, that is a Furuta Pendulum.

Regarding the identification task, beginning with the setup of the physical device, we have continued with the description of the experiments, the presentation of the two utilized models and finally the results obtained, passing through a detailed description of the usage of the GUI.

The interface allows a lot of choices on the different settings of the optimization: we can set the parameters, of course, with some constraints on their values, we can perform calibration and validation on several sets of data, we can perform calibration even of the initial conditions of our model. Obviously also the setup of the optimizer and of the integrator can be done easily, choosing tolerance and so on.

The results achieved for our case study are, as we can see in the pictures of Sections 4.5.2 and 4.5.3, excellent, with an almost perfect fit in the case of calibration and an optimal fit in the case of validation. The second model, described in Chapter 4.4.4, has obtained, of course better results because of a more complex model of the friction with a following larger number of parameters: we have however to remember that, increasing the number of parameters a lot, is not a good way of performing an identification since this brings troubles validating the model with data different from those used for calibration.

The same explanations given for identification, have been given also for control design: starting with a simple linear system, we have shown all the steps needed to perform a Multi-Criteria or a Multi-Case Optimization of a parameterized controller. Since the two issues are very similar, these steps are not very different from the ones already seen in the case of identification. Then, the knowledge acquired with the simple linear system, has been applied to the Furuta Pendulum.

We can say that the results achieved in this thesis entirely satisfy the

expectations which we thought about, at the beginning of this work.

6.2 Future Works

Regarding the identification problem in general, the work done in this thesis could be extended with the analysis and the use of different calibration and validation softwares such as, for example, *MoCaVa* that has been briefly presented in Appendix A and offers the interesting possibility to receive Dymola designed models as inputs.
Considering, on the contrary, our case study, we could try to improve the identification of the Furuta Pendulum using a more refined friction model as, for example, we can see in [15], even if we can consider the already achieved results more than sufficient for our purpose.

Thinking about the design problem, it could probably be found a more appropriate example than Furuta Pendulum and Pole-Placement controller, which the new feature of Dymola can be applied to.

Appendix A

MoCaVa

The *MoCaVa* software [2] is a tool for calibrating and validating tentative model structures using one or more samples of discrete stimulus and response data. The tentative model structures comprise systems of components describing the physical units or phenomena that together constitute the process to be modelled.

MoCaVa is a "grey-box" program, so it needs, to performing identification, an *a priori information* and the *presence of random disturbances*.

Disturbances are important because it is their presence that makes the calibration and validation tasks different from those derived from straightforward comparisons of the responses of model and object to known stimuli.

In MoCaVa prior information is nothing else than a set of sub-models that after will be assembled together into a simulation model for the integrated process. Initially it is not known which of these sub-models will be in the final, satisfying model, so the software recursively test and try to fit various models structures starting with the simplest and most reliable sub-models (for example the ones obtained from mass or energy conservation).
Then the structure develops through a process of *"pruning and cultivation"*: the sub-models considered not suitable are eliminated from consideration, while the others continue to be candidates for further refinement. MoCaVa goes to the final model expanding the intermediate ones appending components modelling separate process units or, otherwise, refining the model by adding components that model internal physical phenomena.

MoCaVa attends to the elimination of sub-models and suggests to the user some convenient refinements to make in the next step. At this point, the user can decide to follow the advice of the software or to choose another way: the design of MoCaVa is based on *the belief that an engineer is usually good at amending models*.

The software is composed by a great number of independent script files, but it exist a shell that guarantees that scripts are executed in correct order. These files are Matlab scripts: this implies portability.

MoCaVa can import models from Dymola, but they have to be of "all continuous" type, that is, "hybrid" models cannot be processed, and neither can models containing serious discontinuities.

Appendix B

Used Dymola Classes

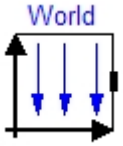
In this appendix we are going to present the Modelica components that we have utilized for the design and the control of our models.

There are components used to model the systems themselves (rigid bodies, joints, frictions, springs, dampers and transfert functions) and the one used to describe their external environment (world).

About control components, there will be depicted the PID controllers and the criteria used in the optimization: in this work we have used only Overshoot, Settling Time and Rise Time blocks, but there are others already available in the Design library and new ones can be designed as well.

Further details on these blocks can be found in the Dymola Online Help, from which these short descriptions have been taken.

B.1 World



Modelica.Mechanics.MultiBody.World

The component *World* represents a global coordinate system fixed in ground. It is used as the *inertial system* in which the equations of all elements of the MultiBody library are defined. It furthermore represents the *gravity field* for a MultiBody model.

By default the gravity field is uniform and the gravity acceleration is vector g

is, of course, the same at every position with a value of $9.81 m/s^2$. However, a point gravity field can also be selected.

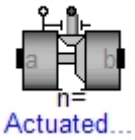
The other function is defining the default settings of the animation properties of the model and representing axes and gravity field in the animations.

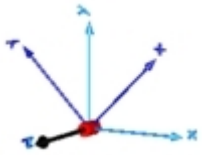
Since the gravity field is required by all bodies with a mass, and since the animation settings are required by almost all components, exactly one instance of World must be present in every model. So it has to be declared with:

```
inner MultiBody.World world
```

To notice that it has to be declared as an *inner* variable so that it is global and it can be accessed from all the objects in the model. This declaration is automatic when we drag this component from the package browser into a model.

B.2 Actuated Revolute Joint



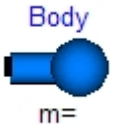


Modelica.Mechanics.MultiBody.Joints.ActuatedRevolute

This is a joint where *frame_b* rotates around axis n which is fixed in *frame_a*. The two frames coincide when $phi + phi\_offset = 0$, where phi_offset is a parameter with a zero default and *phi* is the rotation angle.

The revolute joint has two additional 1-dimensional mechanical flanges (flange *axis* represents the driving flange and flange *bearing* represents the bearing) where it can be driven with elements of the *Modelica.Mechanics.Rotational* library.

B.3 Rigid Body



Modelica.Mechanics.MultiBody.Parts.BodyShape

It is the model of a rigid body with mass, inertia tensor and two frame connectors. All parameter vectors have to be resolved in *frame_a*. The inertia tensor has to be defined with respect to a coordinate system that is parallel to *frame_a* with the origin at the center of mass of the body. The coordinate system *frame_b* is always parallel to *frame_a*.

B.4 Bearing Friction

Modelica.Mechanics.Rotational.BearingFriction

This element describes Coulomb friction in bearings, that is a frictional torque acting between a flange and the housing. The positive sliding friction torque *tau* has to be defined by table *tau_pos* as function of the absolute angular velocity ω.

tau_pos	
ω	*tau*
0	0
1	2
2	5
3	8

For example, the table above, gives the following value of *tau_pos*:

$$tau\_pos = [0, \quad 0; \quad 1, \quad 2; \quad 2, \quad 5; \quad 3, \quad 8]$$

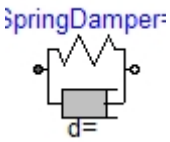
Currently, only linear interpolation in the table is supported. Outside of the table, extrapolation through the last two table entries is used. It is assumed

that the negative sliding friction force has the same characteristic with negative values. Friction is modelled such that when the absolute angular velocity ω is not zero, the friction torque is a function of ω and of a constant normal force. This dependency is defined via table *tau_pos* and can be determined by measurements, that is by driving the gear with constant velocity and measuring the needed motor torque (= friction torque).

When the absolute angular velocity becomes zero, the elements connected by the friction element become stuck, that is the absolute angle remains constant. In this phase the friction torque is calculated from a torque balance due to the requirement, that the absolute acceleration shall be zero. The elements begin to slide when the friction torque exceeds a threshold value, called the maximum static friction torque, computed via:

$$maximum\_static\_friction = peak \cdot sliding\_friction(\omega = 0) \qquad (peak >= 1)$$

B.5 Spring & Damper



Modelica.Mechanics.Rotational.SpringDamper

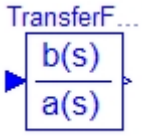
They are a spring and a damper connected in parallel. It can be used to model friction.

B.6 Angle Sensor

Modelica.Mechanics.Rotational.Sensors.AngleSensor

It measures the *absolute angle phi* of a flange in an ideal way and provides the results as output signal *phi*.

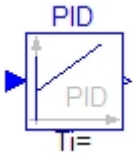
B.7 Transfer Function



Modelica.Blocks.Continuous.TransferFunction

This block defines the transfer function between its input and its output. This definition is made setting the numerator and denominator coefficients.

B.8 PID



Modelica.Blocks.Continuous.PID

This block defines the PID controller shown in Figure B.1.

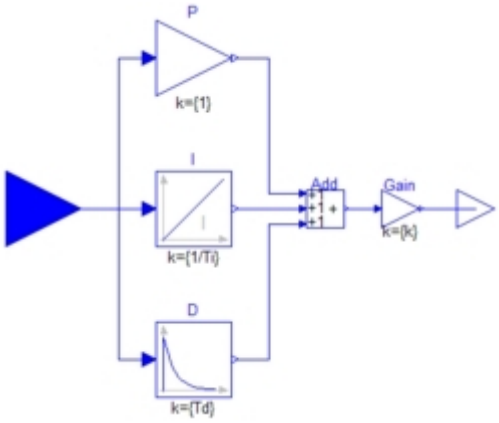


Figure B.1: PID Controller in Additive Form

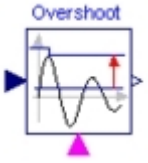
It is useful to know the following equivalences:

$k = K_P$ Gain

$T_I = \frac{K_P}{K_I}$ Time Constant of Integrator [s]

$T_D = \frac{K_D}{K_P}$ Time Constant of Derivative block [s]

B.9 Overshoot



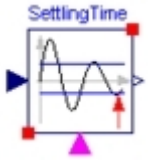
Design.Criteria.Overshoot

This block calculates the Overshoot of the input signal with respect to the value set by the *reference* parameter.

The Overshoot is provided as output signal y and it is computed as long as the simulation runs: the value of y is updated every time that the value of the step response of the system becomes greater than the current value of y. All that implies that the simulation time must be long enough not to incur in a wrong value of the Overshoot.

It is important to notice that the calculation must be activated by a boolean signal and that y starts with an initial value defined by the parameter *y0*.

B.10 Settling Time



Design.Criteria.SettlingTime

This block calculates the Settling Time of the input signal which must be the difference between the reference signal and the step response of the system.

The settling time is provided as output signal y and it is computed as long as the simulation runs: in fact the value of y is updated every time that the input signal has a distance from 0 smaller than the tolerance requested by the parameter called *tolAbs*; y, however, returns to its initial value *initialSettling-Time* (that must be high), whenever the distance from 0 of the input signal becomes greater than *tolAbs*. All that implies that the simulation time must be long enough not to incur in a wrong value of the Settling Time.

It is important to notice that the calculation must be activated by a boolean signal.

B.11 Rise Time

Design.Criteria.RiseTime

This block calculates the Rise Time of the input signal with respect to the value set by the *finalValue* parameter. The fractions of the final value where Rise Time starts and where it ends are set respectively with the *lowValue* and *highValue* parameters and the output signal is then calculated as $highValue \cdot finalValue - lowValue \cdot finalValue$.

B.12 Pole-Placement Controller

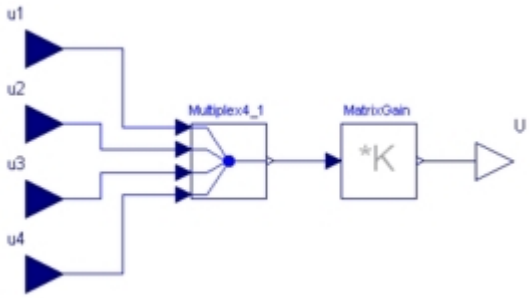


Figure B.2: Diagram of the Pole-Placement Controller

Figure B.2 represents the diagram of a Pole-Placement Controller with four inputs and one output. This block is not present in the Dymola libraries but it has been designed ad hoc for this work.

The four inputs are joined in a vector $u = [u_1,\ u_2,\ u_3,\ u_4]^T$ and then multiplied for the vector $K = [k_1,\ k_2,\ k_3,\ k_4]$, where the k_i parameters have been found imposing the eigenvalues of the matrix $A - B \cdot K$. Thus, the output value of the block is $U = K \cdot u$.

Bibliography

[1] www.dynasim.se.

[2] http://www.s3.kth.se/control/projects/mocava/MoCaVa.htm.

[3] http://www.modelica.org/documents/ModelicaOverview14.pdf.

[4] http://www.modelica.org.

[5] www.ida.liu.se/~vaden/teaching/modelica/lecture6/lecture.html.

[6] www.dlr.de/rm/en/Desktopdefault.aspx/tabid-401/.

[7] http://www.control.lth.se/~kursolin/labs/lab2/lab2-02/lab2.html.

[8] J. Åkesson. Safe manual control of unstable systems. Master's thesis ISRN LUTFD2/TFRT--5646--SE, Department of Automatic Control, Lund Institute of Technology, Lund University, Lund, Sweden, September 2000.

[9] J. Åkesson. Operator interaction and optimization in control systems. Licentiate thesis ISRN LUTFD2/TFRT--3234--SE, Department of Automatic Control, Lund Institute of Technology, Lund University, Lund, Sweden, December 2003.

[10] J. Åkesson and K.J. Åström. Safe manual control of the Furuta pendulum. In *Proceedings 2001 IEEE International Conference on Control Applications (CCA'01)*, pages 890–895, Mexico City, Mexico, September 2001.

[11] K.J. Åström and B. Wittenmark. *Adaptive Control*. Addison-Wesley, Reading, Massachusetts, 1989.

[12] I. Dressler. Code generation from jgrafchart to modelica. Master's thesis ISRN LUTFD2/TFRT--5726--SE, Department of Automatic Control, Lund Institute of Technology, Lund University, Lund, Sweden, March 2004.

[13] H. Elmqvist, H. Olsson, S.E. Mattsson, D. Brück, C. Schweiger, D. Joos, and M. Otter. Optimization for design and parameter estimation. In *Proceedings of the 4th International Modelica Conference*, pages 255–266, Hamburg, Germany, mar 2005.

[14] M. Gäfvert. Derivation of furuta pendulum dynamics. Report ISRN LUTFD2/TFRT--7574--SE, Department of Automatic Control, Lund Institute of Technology, Lund University, Lund, Sweden, 1998.

[15] M. Gäfvert, J. Svensson, and K.J. Åström. Friction and friction compensation in the Furuta pendulum. In *Proc. 5th European Control Conference (ECC'99)*, Karlsruhe, Germany, 1999.

[16] R. Johansson. *System Modeling and Identification*. Prentice Hall, Englewood Cliffs, New Jersey, 1993.

[17] L. Ljung. *System Identification—Theory for the User*. Prentice Hall, Englewood Cliffs, New Jersey, 1987.

[18] J.P. Norton. *An Introduction to Identification*. Academic Press Inc., London, UK, 1986.