Evaluation of FMI-based workflow for simulation and testing of industrial automation applications

Sara Gunnarsson

LUND UNIVERSITY

Department of Automatic Control

MSc Thesis
ISRN LUTFD2/TFRT--6002--SE
ISSN 0280-5316

Department of Automatic Control
Lund University
Box 118
SE-221 00 LUND
Sweden

Printed in Sweden by Tryckeriet i E-huset
Lund 2016

Abstract

The Modelica language is an object-oriented, equation-based language used for modeling and design in many different domains and for various physical applications. In order to exchange models both between Modelica-based tools and other tools the Functional Mock-up Interface (FMI) has emerged. This interface enables exchange of virtual models between manufacturers and/or divisions within a company. The industry is moving towards increased development of virtual models, both for understanding of systems but also because it lowers the development costs.

In this thesis a proof-of-concept of the newly released FMI support in Automation Studio is tested. With a physical model of a reaction wheel pendulum as base, a virtual model is designed in the Modelica-based simulation tool Dymola. A reaction wheel pendulum is a pendulum equipped with a motorized wheel, which is run in different directions in order to swing up the pendulum and to be able to balance it at its highest point. The Dymola model is converted into a Functional Mock-up Unit (FMU), which is the implementation of the FMI. Since not all values of the pendulum are known, an estimation script is written and run in JModelica.org. The FMU is then modified with these estimated values and tested with the controller included together with the reaction wheel pendulum in Simulink, in order to verify that these work together. Lastly the FMU is imported into Automation Studio together with the controller, imported with Automation Studio Target for Simulink, and simulated together there.

The workflow of the thesis was successful and the controller managed to control the FMU in Automation Studio.

Acknowledgement

Firstly I would like to thank my supervisor at Modelon, Maria Henningsson, for all her patience and help, without her this thesis would not have been completed. Secondly thank you to my supervisor Anders Robertsson for the patience and inputs that you assisted me with.

Also thanks to the people at B&R Automation and Modelon for having patience with my questions and the company you have provided.

And thank you Ylva Teleman for making me laugh when everything seemed to be going against me.

Table of Contents

Table of figures

Glossary

FMI (Functional Mock-up Interface): An interface for exchanging models between different tools in order to be able to simulate models together and in different environments.
FMU (Functional Mock-up Unit): The file that has been exported with the FMI is called FMU.
HiL (Hardware-in-Loop Simulation): When simulating the FMUs and/or controllers on the actual hardware or with the hardware included in the simulation.
JFMI (Java Wrapper for FMI): A wrapper for FMI in Java.
MiL (Model-in-Loop Simulation): When simulating the FMUs and/or controllers on a virtual model of the actual hardware, could be a virtual PLC (described below) like in this thesis.
OEM (Original Equipment Manufacturer): Usually used in the automotive industry describing a company that is assembling the product that is sold on the market. The components, like wheels and the motor, can be supplied by other manufacturers. Most car companies are OEMs.
PLC (Programmable Logic Controller): A programmable digital computer used for automation solutions. Could be controlling manufacturing machines in a production line for example. It can have several digital and/or analog inputs and outputs for processing data.
SiL (Software-in-Loop Simulation): When simulating the FMUs and the controllers in software.

1. Introduction

1.1 Background

This master thesis is a collaboration between B&R Automation and Modelon.

Modelon is a Swedish company specialized in simulation, modelling and optimization. The company is developing and supporting Modelica and FMI open standards, for example JModelica.org. They are experts in solutions for model-based systems and control design. Their customers are mainly in the automotive, aerospace, process and energy industry.

B&R Automation is an Austrian company with expertise in hardware and software solutions for automation and process technology. They customize their products for each customer and are specialized in controller, visualization and drive technology. They have modular systems for hardware and develop the software Automation Studio for implementation and testing of control systems.

B&R is developing an FMI (Functional Mock-up Interface) support for Automation Studio, to make it possible for customers to import FMUs (Functional Mock-up Units) from third party tools, like Dymola. The gain of doing this is to be able to test and validate complex systems virtually before they are actually built. Since Modelon is a great actor in this field they provide the knowledge and support of Dymola and of importing FMUs into new platforms.

FMI is an interface which makes it possible to share and transfer virtual models between users and platforms to be able to simulate together with other models or to continue building or simulating the model in another department of a company.

1.2 Overall objectives

The main goal with this thesis is to analyse the workflow, identify challenges and provide proof of concept for importing FMUs into Automation Studio. A reaction wheel pendulum, provided by B&R Automation, is used as a reference model throughout the thesis work as it is a fairly simple process but whose properties are well suited to illustrate the different steps and it is also available as a physical model for experiments. Included with the pendulum there is a controller, which makes it possible to focus the thesis on the FMI-support in Automation Studio.

1.3 Limitations

The thesis does not include designing the controller for the pendulum. In the material included with the pendulum a working controller is present and this is the one used in the thesis. Only FMUs generated in Dymola are tested in this thesis.

1.4 Approach

The model of the pendulum was built in Dymola and exported as an FMU. Some parameters were unknown which made it necessary to use a method to estimate these, and this was what the grey-box identification method was used for. A script that estimated the variables with respect to a cost function was run. When the parameters were estimated, the Dymola model was completed and converted into an FMU. To verify that the controller could control the FMU, the FMU was imported into Simulink and run with the controller there, a so-called Software-in-Loop. When this was accomplished the FMU was imported into Automation Studio and testing was made with a so-called Model-in-Loop. For the Model-in-Loop simulation the controller was exported with the Automation Studio Target for Simulink and both the model and the controller were simulated virtually in Automation Studio.

1.5 Outline of the thesis

The outline of the thesis is as follows. Chapter 2 describes the workflow and the software used in the different steps. Chapter 3 describes the background including the process, the model and its equations. Chapter 4 deals with the detailed description of the workflow that was carried out. In Chapter 5 the results are presented and in Chapter 6 the results are discussed. Conclusions are drawn in Chapter 7. A usability analysis of the FMI-support in Automation Studio is also included in Chapter 7. Finally, some future work is discussed.

2. Workflow

The focus of the thesis is the possibility to connect tools in order to get a workflow that makes it possible to simulate FMUs together with a controller in Automation Studio. In the figure below, Figure 1, the workflow tested in the thesis can be seen.

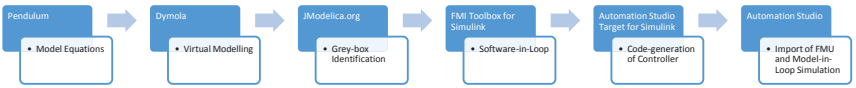

Figure 1: This is the conducted workflow, the blue boxes are tools or hardware and the white boxes are the activity connected to that specific tool.

2.1 Model Equations

The workflow begins with measuring and obtaining data from the physical pendulum. This data is used in the following step for building a virtual model. So the physical pendulum is only used as a model for the virtual model of the pendulum and in the step "Grey-box Identification" it is used for obtaining data, to have something to compare the virtual model with.

2.2 Virtual Modelling

In this step of the workflow the data from the physical pendulum is used for building a virtual model of the pendulum. This is done in the tool Dymola, described below. However, some parameters of the virtual model are not known, since it is not always possible to calculate or measure all values.

Dymola

Dymola [1] is a simulation and modelling tool built on the Modelica language. It makes it possible to connect many different areas of engineering. The available libraries are extensive and can be modified and new libraries can be created depending on the use. The software makes it possible to make virtual models of practically anything and therefore reduces the need for physical prototypes. The models can be tested through the whole procedure of developing new products.

It is possible to build models either by coding or by building with blocks. In this thesis the Electrical [2] and the 3D Mechanical Library [2] have been used to build a model of the pendulum. Since the FMU is to be used on virtual hardware it is necessary to include source code in the FMU.

Dymola supports both FMI import and export. Since the FMI support of Automation Studio does not include a solver, the Co-Simulation mode for the FMU was used. The solver CVode [3] is the one that is included and since the FMU needs source code, this is the only available solver option in Dymola.

2.3 Grey-box Identification

As described in Section 2.2, not all parameters could be measured or calculated. That is why a grey-box identification had to be carried out. For this the tool JModelica.org was used. The basis of the script used for the

grey-box identification was taken from a script included with JModelica.org and then modified to fit the calculations made here.

JModelica.org

JModelica.org [4] is a software for simulation, optimization and analysis of complex physical systems built on the Modelica language and integrated with Python [5]. JModelica.org is now maintained and developed by Modelon AB and is a result of research made at Department of Automatic Control, Lund University [4].

In this thesis the software was used for calibrating unknown parameters in the pendulum model. PyLab [6] together with an optimization script was used for the estimation. The script uses a cost function that is to be minimized. The script imports the FMU, uses experimental data from a .mat-file and estimates the cost function and then simulates the FMU with the estimated values and plots the measured values against the simulated values.

2.4 Software-in-Loop

When the Grey-box Identification was done, the virtual model in Dymola was completed and exported as an FMU, a Functional Mock-up Unit. To be able to verify that the FMU could be controlled by the controller, the FMU was imported into Simulink with the FMI Toolbox for MATLAB/Simulink, described below. Since both the FMU and the controller are simulated in software, this step is a so called Software-in-Loop simulation.

MATLAB/Simulink

MATLAB [7] is a commonly used software for calculation and simulation. For simulation Simulink [8] is a commonly used tool.

The FMI toolbox for MATLAB/Simulink [9] makes it possible to import and simulate FMUs (both Model Exchange and Co-Simulation ones) in both MATLAB and Simulink. In this thesis Simulink was used for simulation. It is possible to choose which variables should be used as outputs from the FMU. The input cannot be chosen but needs to be decided in the exporting tool.

2.5 Code-generation of Controller

This next step has to be done since the controller included with the physical pendulum, which is the controller that also should control the FMU, is provided in Simulink. In order to import the controller into Automation Studio, the Automation Studio Target for Simulink has to be used.

The Automation Studio Target for Simulink

The Automation Studio Target for Simulink [10] translates Simulink systems into C-code to be able to import Simulink systems to Automation Studio.

2.6 Import of FMU and Model-in-Loop Simulation

In the last step, the FMU is imported into Automation Studio using the new FMI-library. The controller was imported in the previous step of the workflow. To be able to simulate the FMU and the controller together they have to be coupled, this is easily made with the feature Mapping. Lastly they are simulated together, run on the virtual PLC that is Automation Studio. Since there is a model of the real hardware in the simulation, it is called a Model-in-Loop. Even though the PLC is virtual hardware, source code still needs to be included in the FMU.

Automation Studio

Automation Studio [11] is a tool for industrial automation solutions. The software is used for design, simulation and monitoring of processes. The programming language is either ANSI C or IEC 61131-3. It is possible to visualize the components in the tool which eliminates the need of using external visualization tools. [11]

In simple terms, Automation Studio can be described as a virtual or simulated PLC. Automation Studio has been used for testing the new FMI support and to simulate the Dymola-generated FMU together with the controller. In some stages of the work the software has been used to increase the understanding of the pendulum but also for monitoring the physical pendulum to extract data.

3. Background

3.1 Modelica

Modelica [12] is an object-oriented, equation-based language for modelling and simulation. In 1996, when the Modelica language was initiated, a group in a company wanted to be able to exchange models between the different working groups. Well-established features from other programming languages were collected and adapted into a new equation-based language for modelling, Modelica. However, several companies saw the opportunities with the language and it has been widely adopted since. Now the non-profit Modelica Association [13] is responsible for the development and maintenance of the language and a conference is held every 18 months, where both companies and academia can provide their contributions and views of Modelica [14].

The models are described by algebraic, discrete and differential equations [15]. Modelica uses non-causal modelling concepts, meaning that it is not necessary to define input and output relationship among the components. The data flow among model components is not always known, and does therefore not have to be specified. However, the interfaces the components have with the surroundings, called connectors, are usually well-defined. These connectors communicate with other components, symbolizing energy flows. The connectors are usually defined as containing two types of variables: energy and potential variables. Flow variables can be current or heat flow rate and in that case the corresponding potential variables are voltage and temperature.

When connecting two components with the same kind of connectors, two equations are formed. The energy variables are summarized and equals zero, according to the physical laws, and the potential variables are set equal to each other, see Figure 2 [14].

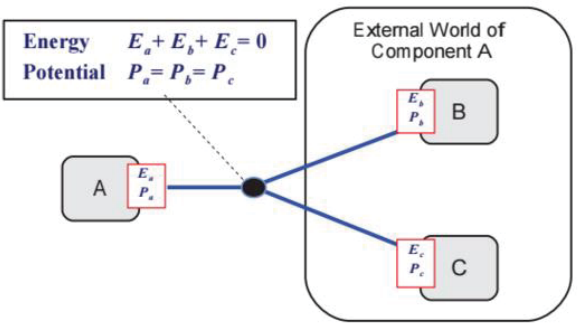

Figure 2: The overview of how components are connected and the equations that are generated when they are connected [14].

Models described by differential-algebraic equations (DAE) are difficult or sometimes even impossible to solve. However, with Modelica it is possible to solve them. A differential-algebraic equation is described by

$$F(\dot{x}, x, y, t) = 0$$

where t is time, and x and y are vectors. The elements of y are called algebraic variables since they have no derivatives in the equations, in contrast to the x elements, called dynamic variables, which have their time derivatives in the equations. The difference between a differential-algebraic equation and an ordinary differential

equation (ODE) is mainly that the Jacobian of F is singular for the DAE. If it had been non-singular it would have been possible to solve the equation to obtain an ODE. An ODE can always be written on the form

$$\dot{x} = f(x, t)$$

which is not the case for a DAE [16].

The concept of Modelica is based on tool-independent modelling that serves a plethora of applications, for example, electrical, mechanical, hydraulic, and thermal systems. The philosophy of Modelica is that no matter how complex a system is, it is possible to divide it into a set of smaller components. Components of the different areas of engineering are built in Modelica and then divided into libraries. Since the components are encapsuled, it is easy to reuse and further develop a component. This makes it easier to build new libraries, based on the basic components [14]. The basic components are found in the Modelica Standard Library [2], an open source library with more than 1200 components and about 900 functions, available for anyone who wants to use them [12]. Here the most basic DC-motors, revolutes and bodyshapes, among others, are located [14]. A benefit with the open source language of Modelica is that users are not dependent on one single tool vendor and can contribute themselves to the development of the language and the libraries [16] [17]. Based on this, several libraries, both commercial and free, have been developed. Modelon has developed the commercial Hydraulic Library [18], as an example. The Modelica Association has developed the free Power Systems library [19] [15].

In order to be able to use the libraries and actually design and develop models a simulation environment is needed. There are both free and commercial simulation environments. The JModelica.org is a free environment and Dymola is an example of a commercial one. The models can be implemented either by dragging and dropping blocks of various libraries in a graphical model editor to make a sketch of the model. Wires then connect the blocks, symbolizing for example fluid flow or an electrical wire, see Figure 3. The blocks are constructed by sub-components and can also be written by simply coding the equations and connections, see Figure 4.

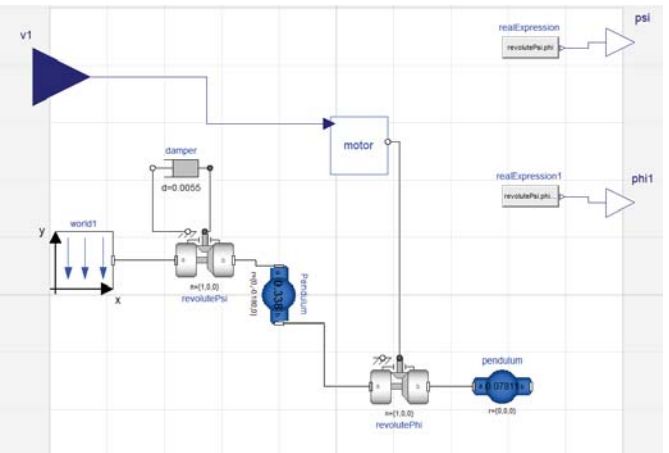

Figure 3: The block view of a model in Dymola.

```
model Pendulum

  parameter Modelica.SIunits.RotationalDampingConstant d
    "Damping constant for psi";
  parameter Modelica.SIunits.Inertia I_11=0.07375 "Inertia for motor";
  parameter Modelica.SIunits.Mass m=0.388 "Mass of motor";
  parameter Modelica.SIunits.Angle psiStart "Psi angle initial guess";
  Modelica.Blocks.Sources.RealExpression realExpression(y=revolutePsi.phi)
    annotation (Placement(transformation(extent={{60,70},{80,90}})));
  Modelica.Blocks.Sources.RealExpression realExpression1(y=revolutePsi.phi +
        revolutePhi.phi)
    ;
  Modelica.Blocks.Interfaces.RealOutput psi
    ;
  Modelica.Blocks.Interfaces.RealOutput phi1
    ;
  Modelica.Mechanics.Rotational.Interfaces.Flange_b support1
    "1-dim. rotational flange of the drive support
    (assumed to be fixed in the world frame, NOT in the joint)"
    ;
  Modelica.Mechanics.Rotational.Interfaces.Flange_a axis1
    "1-dim. rotational flange that drives the joint"
    ;
equation
    connect(revolutePhi.frame_b, pendulum.frame_a) annotation (Line(
        points={{30,-14},{38,-14},{46,-14}},
        color={95,95,95},
        thickness=0.5));
    connect(revolutePsi.frame_b, Pendulum.frame_a) ;
    connect(revolutePsi.frame_a, world1.frame_b) ;
    connect(damper.flange_a, revolutePsi.axis)
      ;
    connect(damper.flange_b, revolutePsi.support) ;
    connect(realExpression1.y, phi1) ;
    connect(realExpression.y,psi)  ;
    connect(Pendulum.frame_b, revolutePhi.frame_a) ;
    connect(revolutePhi.support, support1) ;
    connect(revolutePhi.axis, axis1) ;
    annotation (Diagram(coordinateSystem(preserveAspectRatio=false, extent={{-100,
              -100},{100,100}})));
end Pendulum;
```

Figure 4: The code view of a model in Dymola.

3.2 FMI/FMU

Background

A problem with simulation tools has been that there is usually no possibility to exchange the models between different tools, so the ability to simulate different components together is limited, unless the virtual models can be built in the same software. Since the trend is pointing towards more simulation during various steps of development of new products, this problem is in need of a solution. Simulation makes the development cheaper, since fewer physical models need to be built for testing, and it also deepens the understanding of the products.

In order for an interface to exchange models to become a viable and useful industry standard there are some criteria that need to be fulfilled:

- To be able to be tool independent, a standardization of files is necessary.
- A significant number of supporting tools needs to be available.
- An interface that is easy to use.
- The specific industry has to adopt the standard.
- The interface needs to be mature enough so there can be no errors when importing or simulating the FMUs and the results need to be reliable, no matter what tool is used for simulation.
- Documentation and a reference process is needed for black-box models. [20]

Since the development of different components within different engineering areas usually uses different simulation software it is not always simple to exchange models and couple different components to test them together. To be able to smoothly exchange models and couple components in order to test them together, a standard was needed. Daimler AG [21] wanted an easy way to do this and initiated the Functional Mock-up Interface (FMI) project. It started in 2010 and when it ended in 2011, the Modelica Association took over the maintenance and development of the standard [22].

A plant model that is exported from an FMI-supported tool is called FMU (Functional Mock-Up Unit) and is a .zip file that consists of an XML description and a set of C-functions that describe the simulation code [23]. The XML file contains descriptions of the variables used in the environment the FMU is supposed to be used in. In the file, information that is not needed during execution is stored. The benefit of this is that the tool that the FMU is imported to can use its favourite programming language to read the file. The C-functions are used for the setup and running of slaves in the Co-Simulation mode and to execute model equations in the Model Exchange mode. Other data can be included in the XML file such as model icon, tables, and/or all object libraries [22].

To be able to run an FMU on hardware, like a PLC, source code must be included. There is not that many tools that support this, on the FMI-standard webpage there is only Dymola [1] and MapleSim [24] listed.

Model Exchange and Co-Simulation FMUs

There are two so-called modes when exporting an FMU: Model Exchange and Co-Simulation. The difference between them is how the solver is provided. When a Model Exchange FMU is imported into a tool, the solver is provided by the tool, see Figure 5. But when a Co-Simulation FMU is imported into a tool, the FMU itself is providing the solver, see Figure 6. In this thesis the Co-Simulation mode was used since the FMI support in Automation Studio does not include a solver.

The Model Exchange mode for an FMU enables easy exchange of models between different tools. Since the solver is provided by the tool, this makes it possible to use the same solver for all models simulated in that specific tool.

The intention with Model Exchange is that the simulation environment or tool is capable of generating C-code of the dynamic model in order to form an input/output block that can be integrated in to the simulation and modelling tool [22]. The Model Exchange is useful for exchanging models between different working groups but

also when components are developed by other manufacturers and needs to be integrated into new simulation environments.

The Co-Simulation FMUs are used when the tool itself is not providing a solver or when several models needs to be coupled. The mode is based on a master-slave concept. The slaves simulate and solve sub-problems and the master is responsible for coordinating the overall simulations but also monitoring the data exchange between the subsystems. The data exchange is limited to discrete communication points, so in the time between two communication points the sub-systems are solved independently by their own individual solver. This means that each FMU handles one part of the problem.

This is a very useful feature of the Functional Mock-up Interface since it is not always possible to build all models in the same tool, but it is very common to have to simulate these components together or with real world system components, like in Hardware-in-Loop simulations [25].

Figure 5: The principle of the Model Exchange FMUs.

Figure 6: The principle of Co-Simulation FMUs.

Tools to help the standard

Since the combinations of FMUs and tools are almost infinite there is still the risk of errors when importing FMUs into tools; like failed assertions and unspecified errors at import.

These problems have resulted in an FMU Compliance Checker [26] and FMI Cross Checking Rules [27]. The FMU Compliance Checker was implemented by Modelon and tests the common errors occurring when importing an FMU, some problems when simulating an FMU and formal errors that can occur in an FMU. The FMI Cross Checking Rules focus on testing the quality of the importing and exporting into different tools. [20]

One of the greatest challenges connected to this is to make the simulations trustworthy independent of which tool is importing the FMU. The simulation needs to behave the same way in different tools and simulation environments, but also to make the solver accompanying the FMU (concerning the Co-Simulation) and the solver the tool is providing (concerning the Model Exchange) work properly. The problems encountered when simulating multiple FMUs are mainly new since models have neither been exchanged nor coupled in the past. [20]

On the FMI webpage [28] the tools that are listed are marked in green or orange. Green means that the tool has passed the FMI Cross Checking Rules and orange means that the FMI import or export is claimed to be available but has not passed the Cross Checking Rules yet, see Figure 7. To be marked as green, and thereby being listed as available on the FMI webpage, 12 FMUs must be exported or imported. This makes it possible for users to see which tools are reliable and compatible. [29]

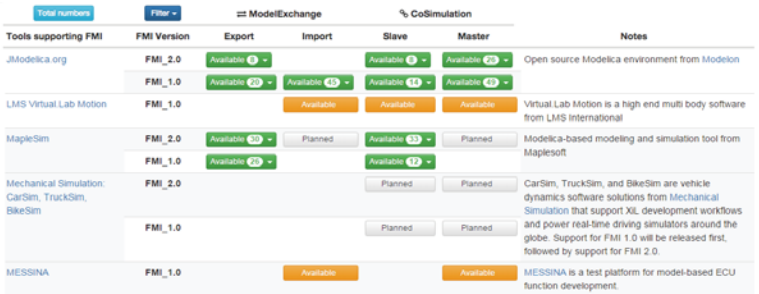

Figure 7: An example of tools that are listed as available and which have passed the compatibility test on the FMI-webpage. [29]

The FMI standard decreases the need for in-house solutions for exchange of models between teams and/or external partners and tool couplings. Since the need of exchange of simulation models are increasing, especially between Original Equipment Manufacturers (OEMs) and suppliers, the FMI standard is a solution to this. Although the standard still has some flaws, such as developing the FMI Cross checking to address more complicated problems and with multiple FMUs, and the model exchange process that needs to be improved together with the accompanying documentation [20].

Benefits

In only a couple of years the standard has been widely adopted in the industry. On the FMI-standard webpage [28] more than 40 tools are registered as available [29]. This proves that the standard was needed and really fulfils a purpose when it comes to exchanging models. Before the standard there were some problems that were especially cumbersome and error prone, and an example of that would be to go from a Model-in-Loop (MiL) to Hardware-in-Loop (HiL). The main challenge was that the models for the two cases came from different tools, which commonly were not compatible with each other, and to translate from one tool to another was time consuming. With the FMI standard it is possible to test MiL and HiL against the same FMU. However, to be able to test HiL, source code FMUs are needed and so far only a few tools support this [30].

Another benefit of using the FMI-standard is that it is possible to simulate advanced and complex systems with many components, to see how they behave in different situations and with different values of the specified parameters. When the model is designed there is a need to have control over which equations and parameters that build the model in Modelica. However, when the model is exported as an FMU and used in simulations the user is not interested in what the model consists of; it is more important that it works properly and behaves like a real physical model would [20].

Published used cases

The FMI standard is used in many fields of engineering, not only the most obvious. In the automotive industry the standard is well used. Modelon works in this area and there are several articles about this, for example Belmon et al. [31] who have written about Dongfeng Commercial Vehicles, regarding developing powertrain controls for hybrid light trucks. Dongfeng is using Modelica and FMUs for the virtual modelling of, for example, engine, gearbox and tires. The virtual model makes it possible to test and simulate this complex system that contains a lot of components, and see how it behaves and optimize, for example, gearshifts and hybrid drive strategies. For the different stages of building the model different tools are used, for example: ITI Simulation X [32] that is Modelica based and specialized in testing software and changes in parameters, and QTronic [33] that tests the hybrid motor's transitions when the charge of the battery is varying [31].

The components are built in different tools, either by different working group within the company or by a third party manufacturer, and needs to be simulated together in different environments in order to thoroughly test and analyse the models. For this application the Functional Mock-up Interface is very useful. In this case it is probable that both Model Exchange and Co-Simulation FMUs have been used. Model Exchange for the

components that needs to be simulated together with the same solver, in specific environments and Co-Simulation FMUs for components and models that need to be coupled but have their own solver.

Électricité de France (EDF) is the world's largest electricity producer with expertise in a wide range of areas and with its own research centres [34]. Energy efficient houses are getting more and more common and need to be evaluated to see how the electricity is used and how the people living in the house behave depending on the comfort and temperature in the house. Because of this, EDF is doing simulations of a model of an energy efficient house in Dymola and an Agent-Based Model [35] of the occupants of the house. These models are coupled and the electricity consumption is monitored. Here is a great example of when Co-Simulation FMUs are useful. There are two models that needs to be simulated together, they are both developed in totally different tools that probably will not support an exchange if it were not for the Functional Mock-up Interface. Since the tools are so very different from each other, the solvers might not be the same and maybe it is not desirable that they are the same, so the solver for each FMU is included and then the two models are simulated together.

Since the house is so energy-efficient the warming of the house comes from the people living in the house but also from lighting, white goods and other electrical equipment in the house. The Dymola model is exported as an FMU and put into the occupant simulator with the help of a JFMI wrapper. The occupants' behaviour is modelled after certain approaches, such as if an occupant is ironing, the ironing is preceded by the washing of the clothes and the drying [17].

4. Method

The basis of the thesis is to evaluate and provide a proof of concept of the Functional Mock-up Interface of Automation Studio. For this a simple model of a physical process was needed and since B&R already has a reaction wheel pendulum in their range, this was used.

4.1 Hardware

The pendulum is a reaction wheel pendulum [36], which is a simple physical process that is easy to model. The pendulum is attached to a rack and on the pendulum is a wheel and a motor attached, see Figure 8. The motor provides the wheel with momentum and makes it spin. It is with the help of the motor and the wheel the pendulum can swing up and balance in an inverted position.

The pendulum [37] is provided by B&R Automation and the equipment connected to it is: a power supply (art.nbr: PS1020), a PLC (art.nbr: X20CP1584) and a motor bridge module with four digital inputs (art.nbr: X20MM2436).

Figure 8: The pendulum. [38]

In the Simulink controller included with the pendulum the angles Ψ and φ are used to control the pendulum. Psi is the angle of the pendulum and phi is the angle of the wheel, see Figure 9. The inputs to the controller are the values of these two angles and to be able to control the pendulum the power of the motor, u, is used.

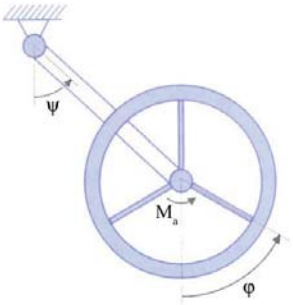

Figure 9: The definitions of Ψ and φ. [38]

The parameters of the pendulum were obtained in the documentation, most values were found in the datasheet [39] for the motor and in the initiation function script for the model in Simulink, see Appendix 1. However, some variables were not specified and could not be measured and these had to be estimated with the help of data from the real pendulum. But to be able to estimate the values a pendulum model had to be designed in Dymola to be able to decide exactly which values were to be estimated.

4.2 Modelling

A model of the pendulum was built in Dymola, see Figure 10, and for this the Mechanical 3D Library [2] and the Electrical Library [2] were used. From the Mechanical library the following components were used: two revolute joints, two bodyshapes, a damper and a world co-ordinate system, see Figure 10. The motor was built from a DC-motor with a permanent magnet, an ideal gear and an inertia component, see Figure 12.

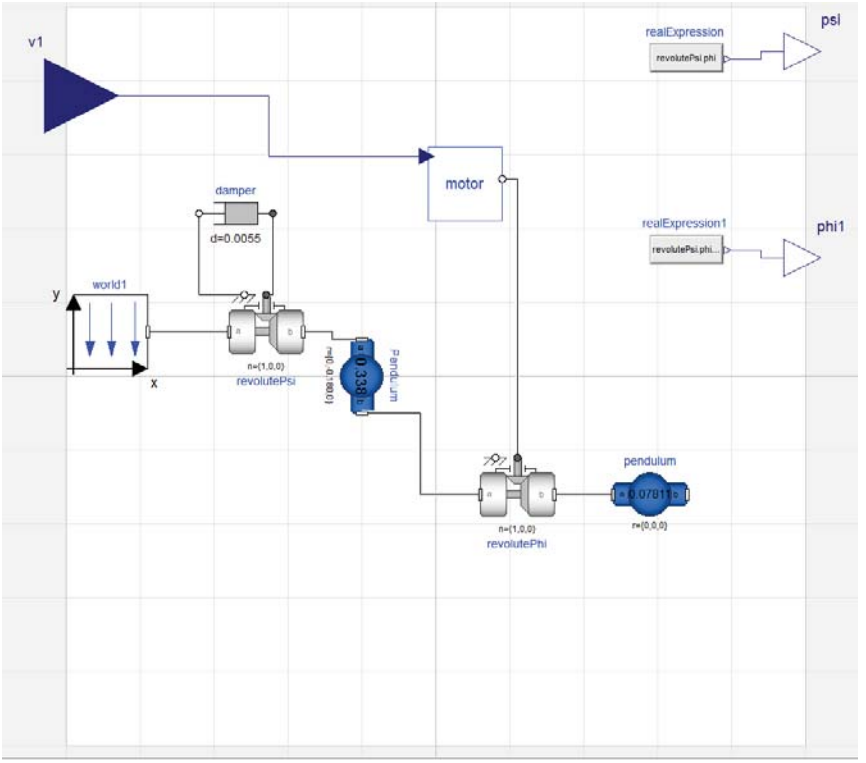

Figure 10: The model of the pendulum, built in Dymola. The blue spheres are bodyshapes.

The bodyshapes were used because of the possibility to specify inertia and mass without the components having a specific geometry. The damper represents the friction of the pendulum. The DC-motor with a permanent magnet was used since it makes it possible to change rotational direction, this is needed in order to control the pendulum. A screenshot of the pendulum in simulation mode in Dymola can be seen in Figure 11.

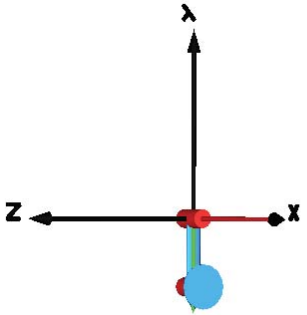

Figure 11: The simulation view of the pendulum in Dymola. Even though the bodyshapes do not have a specific geometry, it can be specified for the simulation view so it is easier to see how the model behaves.

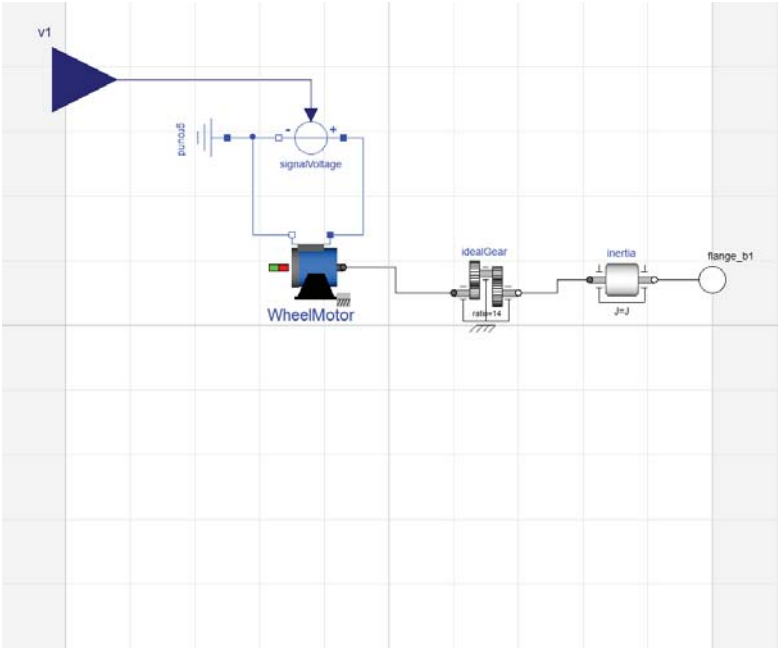

Figure 12: The motor of the pendulum, built in Dymola.

To make the pendulum model behave like the real pendulum the length of point of mass had to be calculated. This was done by using the equations for the pendulum and the values of the Simulink script, see calculations below. The variables and how they were retrieved can be viewed in Table 1. The variables and what they represent are presented in Figure 13.

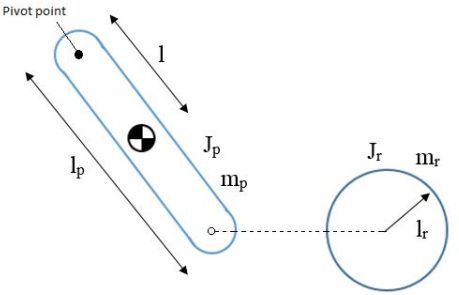

Figure 13: The measurements of the calculations below.

The length of the center of mass, l_p, is calculated as

$$ml = m_p l_p + m_r l_r$$

$$l_p = \frac{ml - m_r l_r}{m_p} = \frac{0.416 * 0.175 - 0.18 * 0.078}{0.338} = 0.1738$$

The moment of inertia, J_p, of the pendulum is calculated from [41] as

$$J = J_p + m_p l_p^2 + m_r l_r^2$$

$$J_p = J - m_p l_p^2 - m_r = 0.0135 - 0.338 * 0.1738^2 - 0.078 * 0.18^2 = 0.0058$$

Table 1: The parameters used for designing a model of the pendulum in Dymola, their values, and where they were retrieved from.

Parameter	*Description*	*Value*	*Obtained from*
m_p	Mass of the pendulum	0.338 kg	Initialization script, see Appendix 1
m_r	Mass of the wheel	0.18 kg	Initialization script, see Appendix 1
$m = m_r+m_p$	Mass of pendulum and wheel	0.416 kg	Initialization script, see Appendix 1
J_p	Moment of inertia of the pendulum about its center of mass	0.0058 kgm^2	Calculated above
J	Moment of inertia of complete pendulums	0.0135 kgm^2	Initialization script, see Appendix 1
l_p	Distance from pivot to the center of mass of the pendulum	0.1738 m	Calculated above
l_r	Distance from pivot to the center of mass of the wheel	0.078 m	Measured on the physical pendulum
l	Distance from pivot to the center of mass of the pendulum and wheel	0.175 m	Initialization script, see Appendix 1

4.3 Grey-box identification

Since the equations for the pendulum were known as well as the values of some of the parameters, like weight, lengths and diameters, grey-box identification was used. Grey-box identification is an identification method used when a physical system is partly known. To solve this by hand is time consuming and sometimes even impossible so JModelica was used to optimize the values of the unknown parameters.

The basis of the script was taken from the scripts included in the JModelica package and describes the estimation of variables of a Furuta pendulum [41]. The script minimizes a cost function. It was modified to fit the estimation made for the reaction wheel pendulum and methods for calculating frequency and damping were written to make the estimated values more accurate.

The script needs data from a real process to compare with so Automation Studio was used to log data from when the physical pendulum was let go from an angle of about -70 degrees and oscillating until reaching balance. The motor is not used here at all. This data, put into a .mat-file, is used as a reference for the estimation of the variables of the Dymola model. The Dymola model is included in the script as an FMU.

The script did not work properly when the motor was included in the FMU so a damper on the revolute for the wheel replaced it, see Figure 14. The variables that were to be estimated were the damping of the pendulum and the damping of the wheel. The damper attached to the wheel was removed and replaced by the motor after the estimation was done.

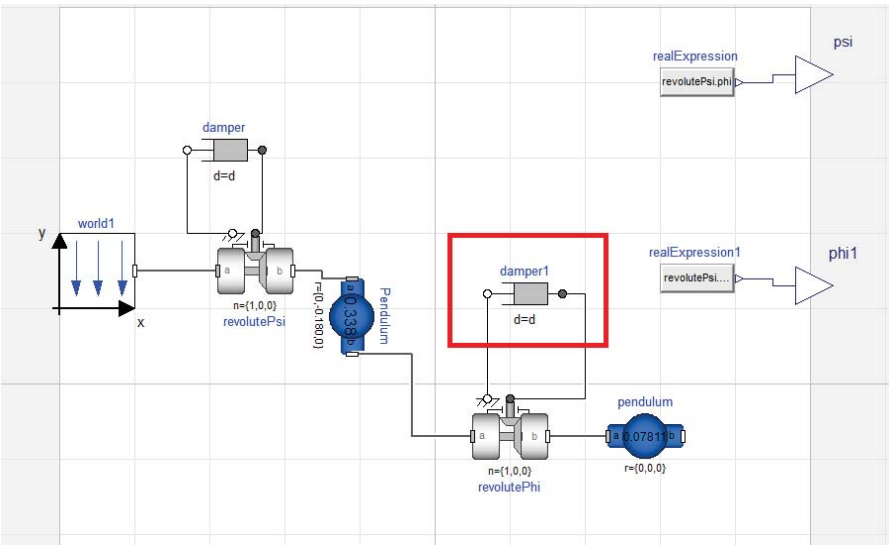

Figure 14: The model used in JModelica with the motor detached and a damper attached instead.

The cost function used is a quadratic loss function.

$$f(x) = (t - x)^2$$

The minimizing function uses the Nelder-Mead simplex function [42], which is used in non-linear optimizations.

The script works as follows:

- It calculates damping and frequency of Ψ and φ of the physical pendulum with the values from the .mat-file.
- The FMU is simulated and damping and frequency of Ψ and φ is calculated from the values of the simulation.
- The above calculated values are put into the cost function.
- The Nelder-Mead method obtains the value of the cost function and fits the behaviour of the real process to the model.
- To be able to decide if the script has made a good estimation, the behaviour of the real process and the model are plotted, see Figure 22 of this in Chapter 5.

The code is attached in Appendix 2.

When the behaviour of the real process and the model matches, the values calculated are set in the Dymola model and the motor is added. The model is tested in Dymola to see that the behaviour is the same here and then transformed into an FMU.

4.4 Controller

The Simulink model included with the pendulum can be seen in Figure 15. The controller, referenced to as Extended Controller in Figure 15, consists of two parts, one for the swing-up of the pendulum and one for the balancing in the top most position, see Figure 16. The controller uses Ψ to decide if the pendulum is in "swing-up" or in "balancing" mode. If abs(Ψ-π) is greater than 0.25 the controller for "swing-up" is used, otherwise the controller for "balancing" is used. The "swing-up" controller can be viewed in Figure 17 and the balancing controller can be viewed in Figure 18.

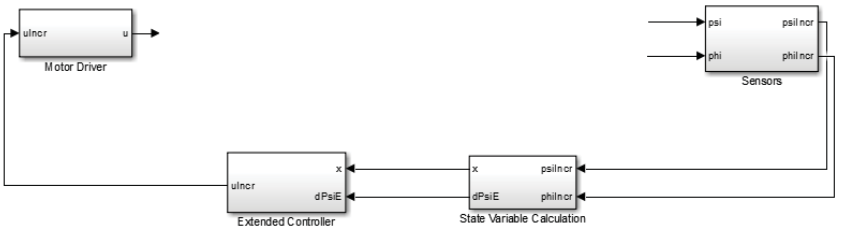

Figure 15: This is the Simulink model included with the pendulum. The Sensors, State Variable Calculation and Motor Driver are functions to help virtual models behave like the physical pendulum. The Extended Controller is described further in Figure 16 and Figure 17.

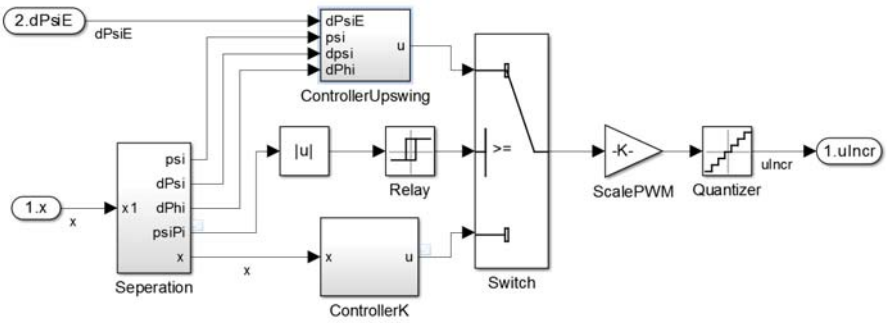

Figure 16: The Extended Controller-block contains two parts, the ControllerUpswing for swinging up the pendulum and the ControllerK for controlling the pendulum in its upright position.

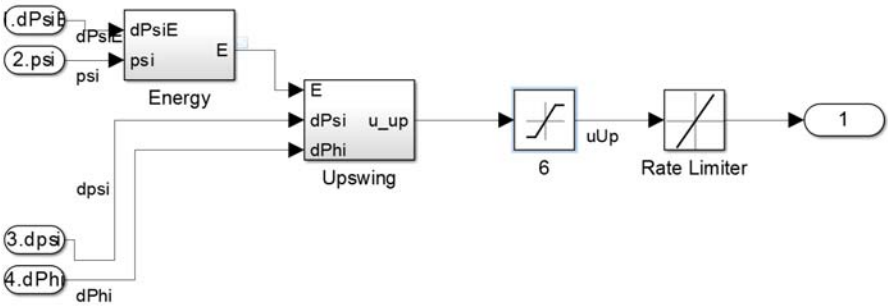

Figure 17: The "swing-up" controller.

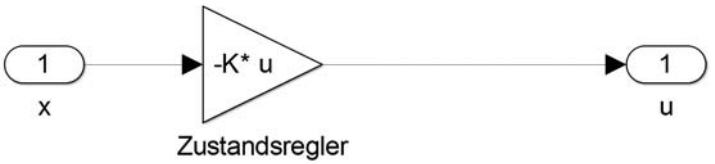

Figure 18: The controller for balancing the pendulum in its upright position, the K is the controller matrix.The x-matrix contains Ψ-π, dΨ (the change of Ψ) and dφ (the change of φ).

The balancing controller uses the Ackermann formula [43] to calculate the controller matrix, K in Figure 18. This matrix is then multiplied by the control signal u. The controller is implemented in Simulink and was included with the physical pendulum so the design of it was not a part of the thesis. The initialization script with declarations of the variables of the two controllers is attached in Appendix 1.

4.5 Software-in-Loop: Simulation in Simulink

When the model of the pendulum was complete it was transformed into an FMU and imported into Simulink, see Figure 19, with the FMI Toolbox for MATLAB/Simulink, to test the controller. Since everything is virtual Sensors, State Variable Calculation, Controller and the Motor Driver have to be included in the project in order to be able to simulate the pendulum. The controller was tested and evaluated with the Simulation Data Inspector and the voltage saturation of the swing-up controller were changed from 11 to 15 to make the model swing up and being able to balance. Plots of this were made, in order to see that the behaviour was the same or at least close to the same as the physical pendulum's behaviour.

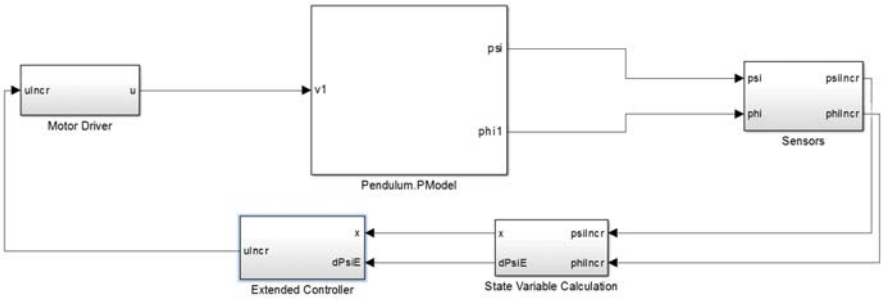

Figure 19: The test system in Simulink. The FMU is Pendulum.PModel and the other components are part of the controller. The block "Sensors" calculates the change is Ψ and φ. "State Variable Calculation" uses a Bessel filter [44] to calculate Ψ, dΨ and dφ from the incoming signals to get the right unit when forwarding the values into the controller. The motor driver calculates u (control signal and voltage to the motor) from uIncr, which is the signal from the controller.

4.6 Model-in-Loop: Simulation in Automation Studio

The FMU is imported to Automation Studio and the controller is imported from Simulink via the Automation Studio Target, Figure 20. Since the FMI-support in Automation Studio is a beta version, many settings had to be done by hand. The FMU is a Co-Simulation FMU, which includes a solver, this is because Automation Studio does not include a solver. Source code is included in the FMU, since the simulation will be run on virtual hardware.

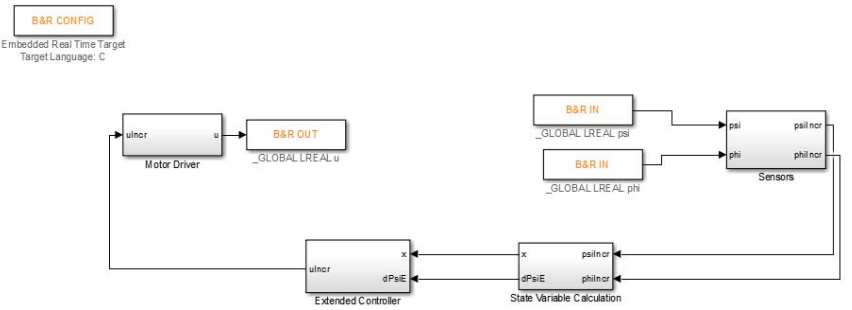

Figure 20: The controller imported to Automation Studio with the Automation Studio Target for Simulink.

To connect the controller and the FMU the feature Mapping in Automation Studio was used. The output of the controller, u, was connected with the input to the FMU, also called u. The inputs of the controller, Ψ and φ, were connected to the outputs of the FMU, also called Ψ and φ, according to the same principle used in Simulink, seen in Figure 19.When these were connected the simulation was started and the Trace function was started. The Trace function plots chosen signals, in order to get a graphical view of how the system behaved. The plots were used to see that the behavior of the FMU was the same or close to the same as the physical pendulum's behavior.

5. Results

In the workflow the steps “Grey-box Identification”, “Software-in-Loop” and “Import of FMU and Model-in-Loop simulation”, seen in the picture below, gave results that will be displayed in this chapter.

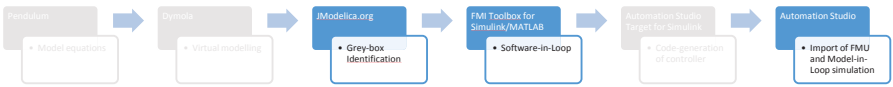

Figure 21: The workflow with the steps where a result is retrieved is marked.

5.1 Grey-box Identification

Below are the results of the estimation of the FMU in JModelica, step three in the workflow, see Figure 21. Here the pendulum was let go from 70 degrees and swung freely until reaching the start position. The reason for choosing 70 degrees instead of 90 was that there were irregularities in the first seconds of data of the physical pendulum. The green curve is the physical data, meaning the data from the physical pendulum and the blue curve is the optimal data, the data from the FMU when the script has decided on a value for the damper. The estimated value of the damper can be seen in Table 2.

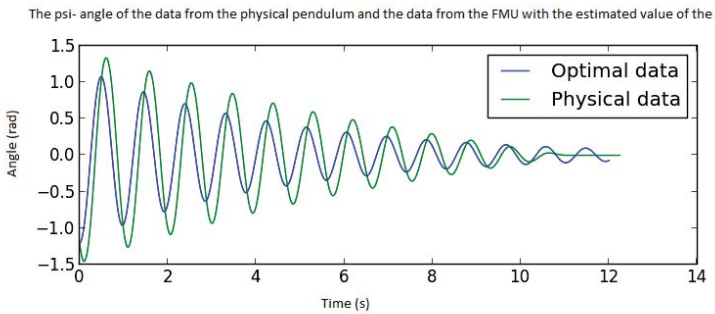

Figure 22: The plot from the estimation with JModelica.org. The green curve represents the data from the physical pendulum and the blue curve represents the data of the FMU with the estimated value of the damper. (Optimal data: Ψ Period time=0.91 s, Physical data: Ψ Period time= 0.91 s)

.

Table 2: The values calculated by the estimation run in JModelica.

VARIABLE	VALUE	UNIT
THE DAMPER OF THE PENDULUM	0.0055	Nms/rad

5.2 Software-in-Loop

The Software-in-Loop is the next step in the workflow, see Figure 21. Here the FMU is simulated together with the controller in Simulink. The controller manages to swing up the pendulum, as can be seen in Figure 23. The pendulum swings back and forth until the pendulum arm angle (Ψ) reaches minus π radians, which is the top most position. This shows that the controller can control the FMU of the pendulum.

The plot of u, see Figure 24, shows the control signal sent to the virtual motor in order to swing up the FMU. The saturation voltage is 15 V, this slight modification is described in Section 4.5 .

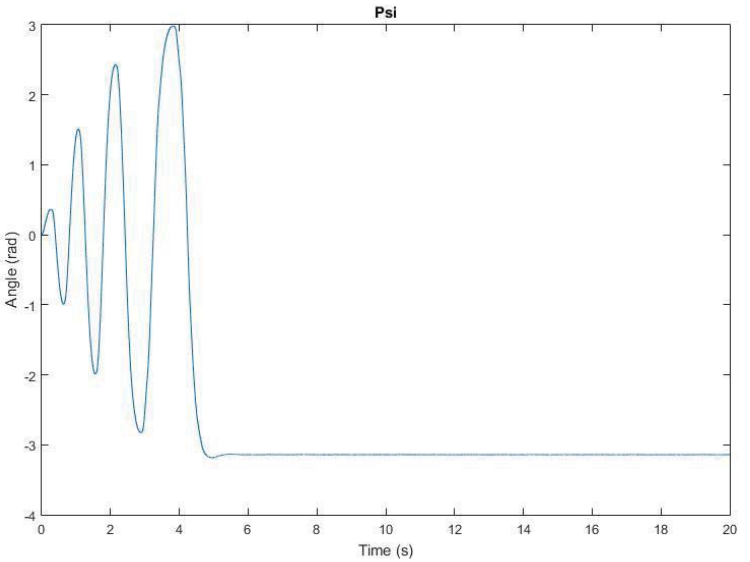

Figure 23: Ψ of the FMU and controller run in Simulink when swinging up.

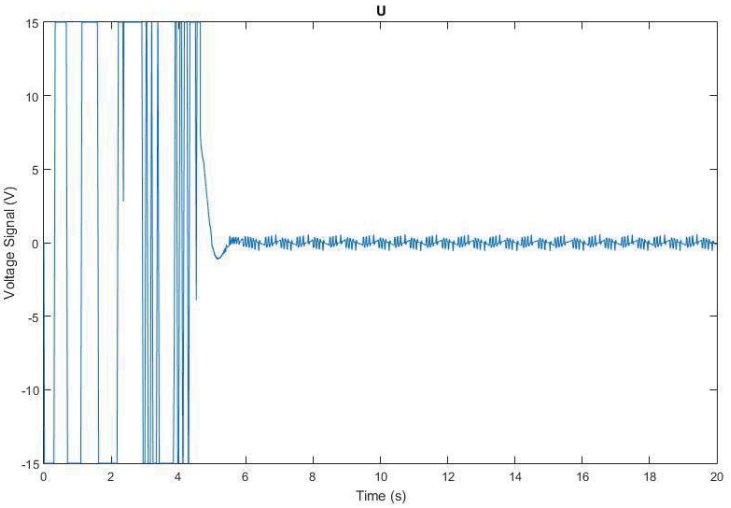

Figure 24: The voltage signal to the motor when the FMU and controller are run in Simulink during swing up.

5.3 Model-in-Loop

The plots seen below are generated when the FMU and the controller are simulated in Automation Studio. There is no hardware connected. The first plot, see Figure 25, is of the pendulum arm angle (Ψ) of the FMU when it swings up in Automation Studio. As can be seen the curve goes up and down and then stabilizes at 3.14 radians, which is the top most position of the pendulum. This proves the controller can control the pendulum in Automation Studio.

The plot of u, see Figure 26, shows the control signal sent to the virtual motor in order to swing up the FMU. Here the saturation voltage is 15 V, as seen in Figure 24 above, since the same controller was used here and this plot is displayed in order to verify that the controller behaves similar in both Simulink and Automation Studio.

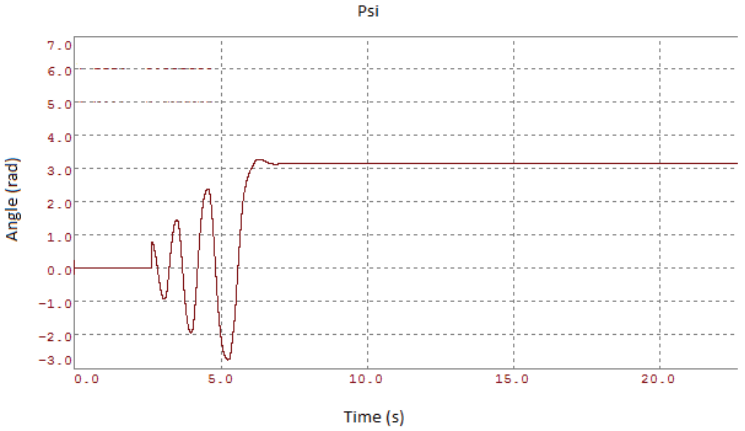

Figure 25: The pendulum arm angle (Ψ) of when the FMU and the controller are simulated in Automation Studio. This is the swing up of the pendulum.

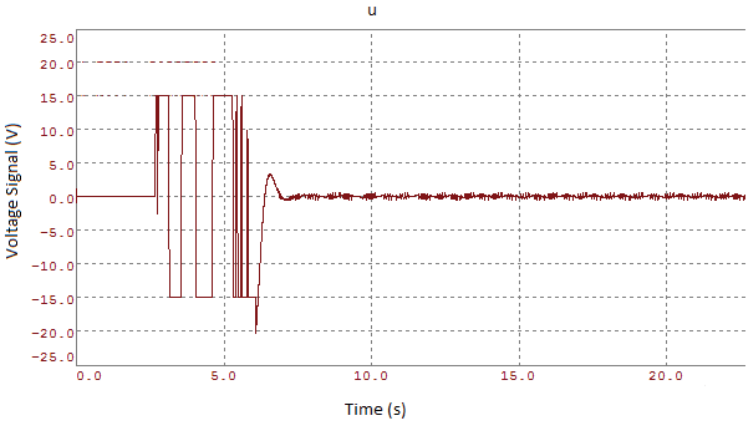

Figure 26: The Voltage Signal of when the FMU and the controller are simulated in Automation Studio. This is the swing up of the pendulum.

5.4 The real pendulum

The plot below are generated when the physical pendulum is run with the controller and is to be seen as a reference for the above generated results. The angle of the pendulum arm is displayed in Figure 27 and shows the swing up of the real process. The physical pendulum does not provide the control signal, u, as a monitored value, that is why there is no such plot included here.

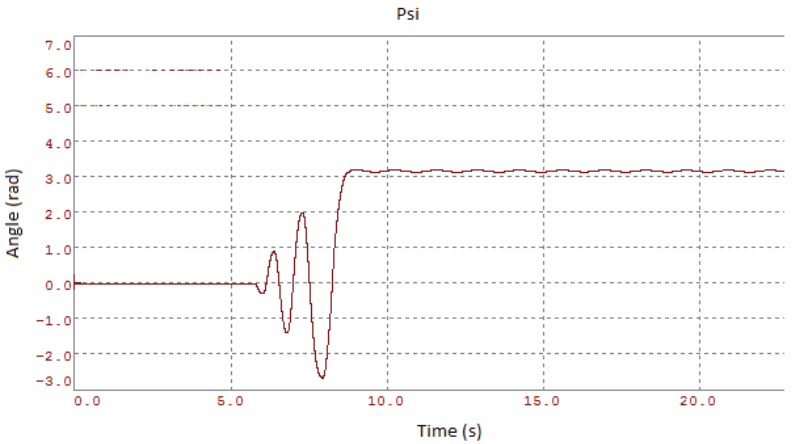

Figure 27: The pendulum arm angle (Ψ) of the physical pendulum when swinging up.

5.5 Comparison of the simulation of the FMU in different tools

The plots below picture the pendulum arm angle (Ψ) of the pendulum when it is let go from 90 degrees. These plots are included in order to see how the FMU behaves in different tools compared to the physical pendulum.

The first plot, see Figure 28, is of the physical pendulum. The plot has a time delay in the beginning, this is because the feature Trace, used for plotting in Automation Studio, has to be started before the process can be started, and it takes some time to change view in Automation Studio in order to start the process.

The second plot, see Figure 29, is of the FMU simulated in Dymola.

The third and last plot, see Figure 30, is of the FMU simulated in Simulink. Since the sampling routine did not include a lot of points, the Simulink plot and the Dymola plot have pointy curves. This was set by Simulink and then the same number was inserted into Dymola.

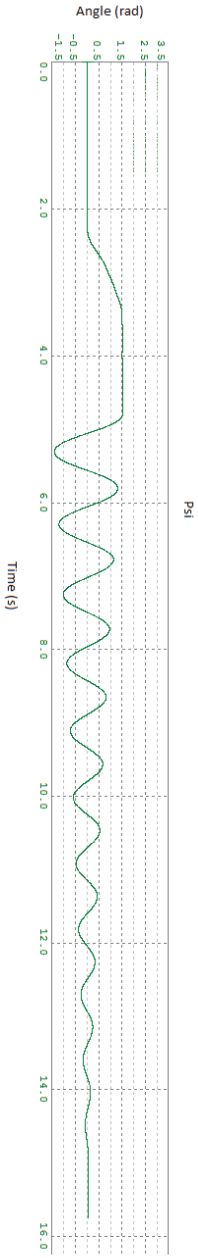

Figure 28: Plot of Ψ when the physical pendulum is let go from 90 degrees. The plot is made with Automation Studio (Period time=0.89 s).

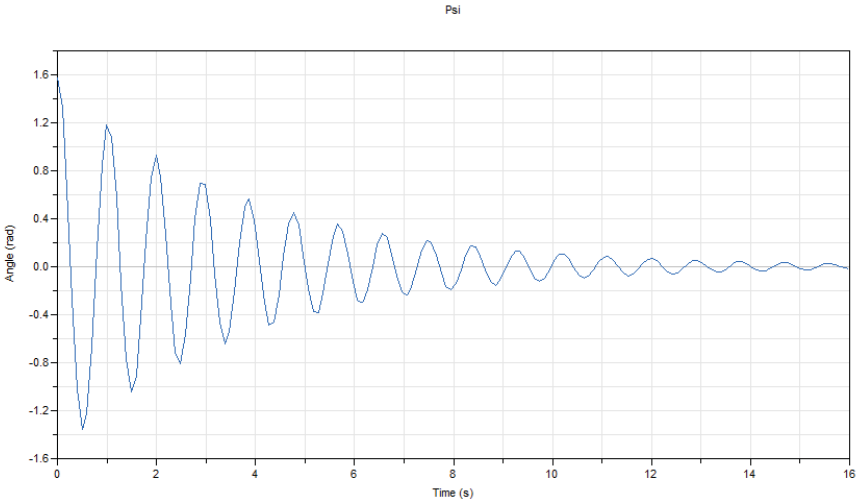

Figure 29: Plot of Ψ of the Dymola model when let go from 90 degrees (Period time= 0.89 s)

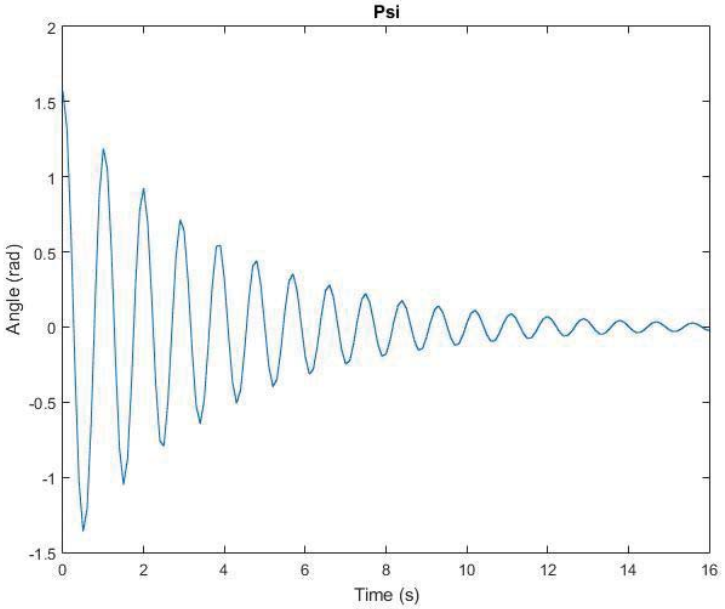

Figure 30: Plot of Ψ of the FMU simulated in Simulink when let go from 90 degrees (Period time=0.89 s).

6. Discussion

6.1 Results

In Section 5.5 the plots of the pendulum arm angle are similar and show that the difference between the tools is very small. Results from Automation Studio would have been a good complement to these plots, however there was no time in the end for this.

The plots of the swing up of the pendulum, seen in Section 5.2-5.4, differ a bit. The physical pendulum takes less time and does only need six oscillations before the pendulum is in its upright position. An explanation for this is that the controller is optimized for the physical pendulum. The other two cases need seven versus eight oscillations and take about one second longer to get the pendulum to its upright position. The possible explanation for this is that the controller is not optimized for the FMU, but also the fact that the FMU probably is heavier or has a higher friction than the physical model. A motivation for this is that the voltage saturation for the controller had to be raised from 11 to 15 in order for the controller to swing up the pendulum. However, the difference is a small detail since the controller can control the FMU even though it is not designed especially for the FMU. That proves that the virtual model is a good enough representation of the physical model.

In the result from JModelica, see Figure 22, the curves of the "optimal data", or rather the estimated data, and the physical data does not follow each other completely but has the same behaviour in damping and frequency. These features were the ones that the estimation was based on and is the important features to consider. This estimation provided the FMU with the value of the damper in order to proceed in the workflow.

6.2 Workflow

The tools used in this workflow are in no way new or revolutionary, however the way they are connected and used together in order to produce a virtual model of a physical process and simulate it, is new. The workflow as it is accounted for here was put together during the work with the thesis, since new needs came up and needed to be taken care of. This workflow has proved to be successful and can be used for this kind of verification and simulation of a virtual model. Below the different tools and their features are discussed.

To produce source code FMUs is not a common feature among the tools supporting FMI, on the FMI-standard webpage there is only Dymola and MapleSim [24] listed as available. Automation Studio has recently been added to the page and is listed as "Planned", this will however change to "available" when enough FMUs have been imported and exported. Since it is not uncommon to have a need for source code FMUs, it is plausible that more tools will include this as a feature in the future.

Dymola can, for now, only provide source-code FMUs with one kind of solver, the CVode. It works but it might be good with other options as well. Automation Studio is a bit limited in the way that it can only import Co-Simulation FMUs, which means that the exporting tool where the FMU is constructed is the one deciding what solver is included with the FMU. An extension with the possibility to import Model Exchange FMUs would be a natural next step. That would mean that Automation Studio would be the tool deciding what solver should be used, meaning that the choice would be up to the one using the FMU not the one making it. Of course this would be a challenge since the solver has to make the Model Exchange FMU to behave the same or at least very similar to the tool were the FMU was constructed. Further in the future the possibility to design and export FMUs would be a great complement to the support. Of course this involves a lot of testing and developing, but would add extra quality to Automation Studio as a tool.

The design of the Dymola model, later turned into an FMU, was the most time consuming part of the process. It was not as easy to design a virtual model and understand all the different features effect the physical abilities of the pendulum, even though there was a documentation of the pendulum. The documentation could have been more extensive; it would be good to know how the Simulink model was designed, why certain components have been chosen, why the calculations were made in a certain order and so on. However, this would probably not have sped up the process, but would have been a good complement and deepened the understanding of the physical and virtual system.

6.3 Usability analysis

Usability is the notion that user interfaces should be efficient, easy to learn and satisfying to use [45].

It is currently laborious to import an FMU to Automation Studio, but since this is only at beta-version it is assumed that this will change when it becomes a "real" version of Automation Studio. Problems were encountered when importing FMUs from Dymola, FMUs from MapleSim worked fine (however this was not tested in this thesis). This shows that the import mechanism works but as for now, it works best with MapleSim generated FMUs. The problems with the Dymola generated FMUs are intended to be resolved.

Adding the library is simple enough and does not have to become an automatic feature. However, removing the DYN_MULTISTANCE-flag and delete all the files included in the program is a bit too risky to have the user do, there are too many elements of risk included. Also, adding files to the software (PC_ANY) is too time consuming, to add all four in one step would be one solution.

When starting the simulation the “State”-field is marked and changed manually from zero to one. This is a solution but not a very user-friendly one, it would be great with a button to start the simulation. However, if the customers are used to use the program like this, then this might not be a problem. Another problem connected to this was the fact that the PLC got into service mode when trying to simulate the FMU directly after a simulation ended. A warm restart solved the problem, but made simulations time consuming.

Some problems with the licenses were encountered too. Two licenses are needed, one for Automation Studio and one for the Automation Studio Target for Simulink. It was especially the latter that was troublesome and it would be easier to have only one, but it is presumed that this is the solution that is tested and works best.

7. Conclusion

The workflow implemented in this thesis followed the following pattern. First a virtual model was designed, then this was converted into an FMU, which was tested using a software-in-loop approach and then, finally, imported into a tool that allowed for model-in-loop testing. It worked well, although some problems were encountered along the way. This is, however, not surprising since in some cases prototype tools were used. It is likely that these problems could be removed with some further development. If this would be a real development of a new product the virtual model would have been designed first and then the physical model would be based on the virtual model. Then it is more likely that the physical model would work as expected, since it has been thoroughly tested. Then the challenge would be to make the physical model work, not the virtual one, and that is probably a greater challenge, however it is easier to change values and behavior since there is no template to start from.

The plants and the physical pendulum have similar behavior but does not match entirely, this is not a problem since it is impossible to make an exact virtual model of a physical one, and there is too many parameters to take into account. This is, however, not the goal, the goal is to make the virtual model as close as possible to the virtual one. Since the FMU can be controlled by the same controller as the physical pendulum, the goal is reached.

This thesis has shown that FMI is a working solution to exchange of virtual models and that the number of supporting tools are increasing. To be able to test a model as thoroughly as it is possible with the Functional Mock-up Interface decreases the development time for new products, deepens the understanding and is more environmental friendly, since less prototypes have to be built. The addition of a tool supporting source code FMUs is welcome and it is possible that the number of tools supporting source code will increase in the near future. FMI is developing fast and will probably be a natural part of simulations and product development in the future.

7.1 Continuing work

There is a lot of work that can continue after this thesis. Some examples are:

- Testing Hardware-in-Loop by running the FMU and the controller on a real PLC.
- Design a controller optimized for the FMU, with the help of the book “The Reaction Wheel Pendulum” [36] where the whole pendulum and how to design an accompanying controller, is documented.
- Coupling two or more FMUs and run them together. The FMUs can be two models or a model and a controller.
- Coupling two FMUs and run them with a code-generated object.

8. References

[1] Dassault Systemes, "Dymola," 2015. [Online]. Available: http://www.3ds.com/products-services/catia/products/dymola. [Accessed 6 November 2015].

[2] Modelica Association, "Standard Library," 2015. [Online]. Available: https://github.com/modelica/Modelica. [Accessed 27 November 2015].

[3] Lawrence Livermore National Laboratory, "Sundials; Description of CVode," 2015. [Online]. Available: https://computation.llnl.gov/casc/sundials/description/description.html#descr_cvode. [Accessed 6 November 2015].

[4] Modelon, "JModelica startpage," 2015. [Online]. Available: http://www.jmodelica.org/. [Accessed 6 November 2015].

[5] Python Software Foundation, "Python: About Python," 2015. [Online]. Available: https://www.python.org/about/. [Accessed 13 November 2015].

[6] Python Software Foundation, "PyLab," 2015. [Online]. Available: https://pypi.python.org/pypi/pylab. [Accessed 13 November 2015].

[7] MathWorks, "MATLAB," 2015. [Online]. Available: http://se.mathworks.com/products/matlab/. [Accessed 6 November 2015].

[8] MathWorks, "Simulink," 2015. [Online]. Available: http://se.mathworks.com/products/simulink/. [Accessed 6 November 2015].

[9] Modelon, "FMI Toolbox for MATLAB/Simulink," 2015. [Online]. Available: http://www.modelon.com/products/fmi-toolbox-for-matlab/. [Accessed 6 November 2015].

[10] B&R Automation, "Automation Studio Target for Simulink," 2015. [Online]. Available: http://www.br-automation.com/en/products/software/automation-studio-target-for-simulink/. [Accessed 6 November 2015].

[11] B&R Automation, "Automation Studio," 2015. [Online]. Available: http://www.br-automation.com/en/products/software/automation-studio/. [Accessed 27 June 2015].

[12] Modelica Association, "Modelica; Startpage," 2015. [Online]. Available: https://www.modelica.org/. [Accessed 6 November 2015].

[13] Modelica Association, "About the Modelica Association," 2015. [Online]. Available: https://www.modelica.org/association. [Accessed 10 November 2015].

[14] A. Elsheikh, M. U. Awais, E. Widl and P. Palensky, "Modelica-Enabled Rapid Prototyping of Cyber-Physical Energy Systems Via The Functional Mockup Interface," 2013.

[15] M. Otter and D. Winkler, "Modelica Overview," 2013.

[16] S. E. Mattsson, H. Olsson and H. Elmqvist, "Dynamic Selection of States in Dymola," in *Modelica Workshop 2000*, Lund, Sweden, 2000.

[17] R. Samlaus and M. Strach, "Static Validation of Modelica Models for Language Compliance and Structural Integrity," University of Nottingham, Nottingham, 2013.

[18] G. Plessis, É. Amouroux and Y. Haradji, "Coupling occupant behaviour with a building energy model - A FMI application," in *10th International Modelica Conference*, Lund, 2014.

[19] Modelon, "Modelon Hydraulics Library," 2015. [Online]. Available: http://www.modelon.com/products/modelica-libraries/hydraulics-library/. [Accessed 6 November 2015].

[20] Modelica Association, "Documentation of the Power Systems Library," 2015. [Online]. Available: https://build.openmodelica.org/Documentation/PowerSystems.html. [Accessed 6 November 2015].

[21] C. Bertsch, E. Ahle and U. Schulmeister, "The Functional Mockup Interface - seen from an industrial perspective," in *10th International Modelica Conference*, Lund, Sweden, 2014.

[22] Daimler, "Startpage," 2015. [Online]. Available: https://www.daimler.com/. [Accessed 4 December 2015].

[23] T. Blochwitz, M. Otter, J. Akesson, M. Arnold, C. Clauss, H. Elmqvist, M. Friedrich, A. Junghanns, J. Mauss, D. Neumerkel, H. Olsson and A. Viel, "Functional Mockup Interface 2.0: The Standard for Tool independent Exchange of Simulation Models," in *9th International Modelica Conference* , Munich, 2012.

[24] R. Braun and P. Krus, "Tool-Independent Distributed Simulations Using Transmission Line Elements and the Functional Mock-up Interface," Linköping, 2015.

[25] Maple Soft, "MapleSim," 2015. [Online]. Available: http://www.maplesoft.com/products/maplesim/. [Accessed 4 December 2015].

[26] J. Bastian, C. Clauss, S. Wolff and P. Schneider, "Master for Co-Simulation Using FMI," Modelica Proceedings, Dresden, 2011.

[27] Modelica Association, "Downloads and Information," 2015. [Online]. Available: https://www.fmi-standard.org/downloads. [Accessed 13 November 2015].

[28] A. Junghanns, "FMI Cross Checks: How to Improve FMI Compliance," 2015. [Online]. Available: https://svn.fmi-standard.org/fmi/branches/public/CrossCheck_Results/FMI_Cross_Check_Rules_v3_2014_07_31.pdf. [Accessed 6 November 2015].

[29] Modelica Association, "FMI; Start page," 2015. [Online]. Available: https://fmi-standard.org/. [Accessed 27 November 2015].

[30] Modelica Association, "FMI; Tools," 2015. [Online]. Available: https://fmi-standard.org/tools. [Accessed 6 November 2015].

[31] E. Drenth, M. Törmänen and K. Johansson, "Consistent Simulation Environment with FMI based Tool Chain," in *10th International Modelica Conference*, Lund, Sweden, 2014.

[32] L. Belmon, Y. Geng and H. He, "Virtual Integration for hybrid powertrain development, using FMI and Modelica models," in *10th International Modelica Conference*, Lund, Sweden, 2014.

[33] ITI, "Simulation X," 2015. [Online]. Available: https://www.simulationx.com/simulation-software.html. [Accessed 13 November 2015].

[34] QTronic, "Qtronic," 2015. [Online]. Available: https://www.qtronic.de/en/weaver.html. [Accessed 15 November 2015].

[35] Électricité de France, "EDF at a glance," 2015. [Online]. Available: https://www.edf.fr/en/the-edf-group/world-s-largest-power-company/edf-at-a-glance. [Accessed 9 October 2015].

[36] E. Bonabeau, "Agent-based modeling: Methods and techniques for simulating human systems," in *Adaptive Agents, Intelligence, and EMergent Human Organization: Capturing Complexity through Agent-Based Modeling*, Irvine, CA, 2001.

[37] D. J. Block, K. J. Åström and M. W. Spong, The Reaction Wheel Pendulum, University of Illinois: Morgan & Claypool Publilshers, 2007.

[38] B&R Automation, "Reaction Wheel Pendulum Demo System," 2015. [Online]. Available: http://www.br-automation.com/en/career/working-at-br/academy/br-automation-academy/eta-system/system-overview/0dswheel01-flywheel-pendulum-demo-system/. [Accessed 6 November 2015].

[39] L. Obersamer, "Mechatronic Systems Control Utilizing the Automatic Code Generation for B&R Systems," Salzburg, 2009.

[40] Faulhaber, "DC-Micromotors; Model 2642...," 2015. [Online]. Available: https://fmcc.faulhaber.com/details/overview/PGR_3877_13818/PGR_13818_13813/en/GLOBAL/. [Accessed 6 November 2015].

[41] D. J. Block, K. J. Åström and M. W. Spong, "Modeling: Equations of Motion," in *The Reaction Wheel Pendulum*, Morgan & Claypool, 2007, p. 12.

[42] M. Gäfvert, "Department of Automatic Control," April 1998. [Online]. Available: http://www.control.lth.se/documents/1998/7574.pdf. [Accessed 13 November 2015].

[43] J. A. Nelder and R. Mead, "A simplex method for function minimization," *The Computer Journal*, vol. 7, no. 4, pp. 308-313, 1965.

[44] J.-F. Monin, Understanding Formal Methods, M. Hinchey, Ed., Springer, 2003.

[45] J. G. Proakis and D. G. Manolakis, Digital Signal Processing, Pearson Education, 1992.

[46] J. Nielsen, "Usability 101: Introduction to Usability," 4 January 2012. [Online]. Available: http://www.nngroup.com/articles/usability-101-introduction-to-usability/. [Accessed 4 December 2015].

[47] Dassault Systemes, "Dymola," 2015. [Online]. Available: http://www.3ds.com/products-services/catia/products/dymola. [Accessed 30 June 2015].

Appendix 1

Initialization code included in with the controller in Simulink.

```
% Reaction Wheel Pendulum

%clc;
%clear;

% Informatoin about the plant

dPsiEncoder = 4096; % Resolution of the incremental encoder of the pendulum
dPhiEncoder = 2000; % Resolution of the incremental encoder on the motor

vPWM = 24; %[V], Voltage of the positive PWM power signal
dPWM = 32767; % Resolution of the pisitive PWM power signal

Ts = 0.001; % [s], Sample time
rrt = 14; % Reduction ratio of the transmission
eta = 0.80; % Efficiency of the transmission
k = 34.6*10^-3; % [Nm/A], Torque constant of the motor
Ra = 5.78; % [Ohm], TTerminal resistance of the motor
Ja = 0.0012; % [kg*m^2], Torque of inertia of motor and rotor
g = 9.81; % [m/s^2], Gravitational acceleration

mr = 0.07811; % [kg], Masse Rotor
m = 0.338+mr; % [kg], Mass of the complete pendulum
l = 0.175; % [kg], Reduced pendulum length
J = 0.0135; % [kg*m^2], Torque of inertia of the complete pendulum

% Linearized Model
A = [0 1 0;
(m*g*l)/J -k^2*rrt*eta/(J*Ra) k^2*rrt^2*eta/(J*Ra);
0 k^2*rrt*eta/(Ja*Ra) -k^2*rrt^2*eta/(Ja*Ra)];
B = [0; -k*rrt*eta/(J*Ra); k*rrt*eta/(Ja*Ra)];
C = [1 0 0];

% Eigenvalues
poles=-[10 10 10];
K = acker(A,B,poles) % Calculation of the controller matrix with the formula of Ackerman

% Aufschwingen mittels Energiebetrachtung

%Eref = 2*m*g*l; % Energie des Pendels in völliger Ruhe in  %instabiler Ruhelage
Eref = 1.47;
kv = 0.1;
ke = 700;

Pc=[B A*B (A^2)*B] % Calculation of the controllability matrix
Pcdet=det(Pc) % Calculation of the determinant of the controllability matrix

Po=[C;C*A;C*(A^2)] % Calculation of the controllability matrix
Podet=det(Po) % Calculation of the determinant of the controllability matrix
```

```
% Bessel filter
n = 8; % Order of the Bessel-Filter
wt = 2*pi*(0.1/Ts); % [rad], Cut-off frequency in

a = 1.0e+022 * [0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0001 0.0237 2.4291];
b = 1.0e+022 * [0 0 0 0 0 0 0 0 2.4291];
```

Appendix 2

The code written in Python and used by JModelica. The basis of the code is taken from a script included in the JModelica package but most of the code is written by the author of this thesis.

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-
# Copyright (C) 2010 Modelon AB
# This program is free software: you can redistribute it and/or modify it under the terms of the GNU
General Public License as published by
# the Free Software Foundation, version 3 of the License.
# This program is distributed in the hope that it will be useful,
# but WITHOUT ANY WARRANTY; without even the implied warranty of
# MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
# GNU General Public License for more details.
# You should have received a copy of the GNU General Public License
# along with this program. If not, see <http://www.gnu.org/licenses/>.

import os
from scipy.io.matlab.mio import loadmat
import numpy as N
import matplotlib.pyplot as plt
from pymodelica import compile_fmu
from pyfmi import load_fmu
from pyfmi.fmi import FMUModelME1
from pyjmi.optimization import dfo
from collections import OrderedDict
curr_dir = os.path.dirname(os.path.abspath(__file__));

# Load measurement data from file
data = loadmat(os.path.join(curr_dir, 'thebest.mat'), appendmat=False)
# Extract data series
t_meas = data['Time'][:,0]
phi_meas = data['phi'][:,0]
psi_meas = data['psi'][:,0]


# Find the highest points, the first low and the places in the vector
# to be able to get the time (for frequency calculations later)
# n = the number of high, voft = vector for finding the times later
# dof a= dictionary for the highs of the curve, to calc amplitude
# top = highest value (to calculate midsection of curve)
# bottom = lowest value (to calculate midsection of curve)
    def damping_calc(list):

f=0
b=0
flag=0
t=0
top=0
bottom=0
element=0
dofa=[]
damp=[]
    for i in range(3, list.size-1):
            if list[i]<list[i+1] and b!=1: # first bottom
                    bottom=list[i]
                    b=1
                    flag = 1
```

```
                elif list[i]>list[i+1] and f!=1 and flag == 1: # first top
                        dofa.append(list[i])
                        top=list[i]
                        f=1
                elif list[i]>list[i+1] and t == 0:   # rest of tops
                        dofa.append(list[i])
                        t=1
                elif list[i]<list[i+1] and t == 1:
                        t=0
                else:
                        print

# Calculate midesection
        mid=(top+bottom)/float(2)
        # Calculate the damping of the data for psi
        for j in range(0, len(dofa)-1):
                first=dofa[j]-mid
                second=dofa[j+1]-mid
                damp.append(first/second)

        for k in range (0,len(damp)):                   #gör vektorerna lika långa
                if len(damp)>4 and element<=1:
                        del damp[0]
                        element+=1
                elif len(damp)>4:
                        damp.pop()
                return damp

def frequency_calc(lista, lis):
        flagg=0
        element=0
        voft=[]
        freq=[]
        for i in range(3, lista.size-1):
                if lista[i]>lista[i+1] and flagg!=1:
                        voft.append(i)
                        flagg=1
                elif lista[i]<lista[i+1] and flagg==1:
                        flagg=0
                else:
                        print

# Calculate the frequency of the data for psi
                for k in range(0,len(voft)-1):
                fir=lis[voft[k]] #collects the time from the index of voft
                sec=lis[voft[k+1]]
                period=sec-fir
                freq.append(1/float(period))

        for j in range (0,len(freq)):  #make the vectors get equal length
                if len(freq)>4 and element<=1:
                        del freq[0]
                        element+=1
                        elif len(freq)>4:
                        freq.pop()
                return freq

# Define the objective functiondofa
def cost(x,with_plots=True):
        freq_meas = []
```

```
    damp_meas = []
    freq_m_phi=[]
    damp_m_phi=[]

    freq_sim=[]
    damp_sim=[]
    freq_s_phi=[]
    damp_s_phi=[]

    damp_meas = damping_calc(psi_meas)
    freq_meas = frequency_calc(psi_meas, t_meas) damp_m_phi=damping_calc(phi_meas)
    freq_m_phi=frequency_calc(phi_meas,t_meas)

# Setting the values of the mass and the damping in the FMU-model
    damping_psi = x[0]
    damping_phi=x[1]

# Collect the FMu from the path
     model = FMUModelME1(os.path.join(curr_dir, 'Pendulum_PModel.fmu'))

# Set new parameter values into the model
     model.set('damper.d', damping_psi)
     model.set('damper1.d', damping_phi)

# Create options object and set verbosity to zero to disable printouts
    opts = model.simulate_options()
    opts['CVode_options']['verbosity'] = 50
    opts['ncp'] = 800
    opts['filter'] = ['revolutePhi.phi', 'revolutePsi.phi']

# Simulate model response with new parameter values
                res = model.simulate(start_time=0., final_time=12, options=opts)

# Load simulation result
   phi_sim = res['revolutePhi.phi']
   psi_sim = res['revolutePsi.phi']
   t_sim  = res['time']

# Calculate the damping of the model psi and phi
    damp_sim=damping_calc(psi_sim)
    damp_s_phi=damping_calc(phi_sim)

# Calculate the frequency of the model psi and phi
    freq_sim=frequency_calc(psi_sim,t_sim)
    freq_s_phi=frequency_calc(phi_sim,t_sim)

    # Evaluate the objective function
    y_meas = N.vstack((freq_meas, damp_meas,freq_m_phi, damp_m_phi))
    y_sim = N.vstack((freq_sim, damp_sim,freq_s_phi, damp_s_phi))
    t_s=[1, 2, 3, 4] #t_s=[1, 2, 3, 4, 5, 6, 7, 8]

    freq_cost = dfo.quad_err_simple(t_s, y_meas, t_s, y_sim)
    return freq_cost

def run_demo(with_plots=True):
    # Choose starting point (initial estimation)
    x0 =N.array([0.001, 0.001])
    # Choose lower and upper bounds (optional)
    lb = N.array([0.0001,0.0001])
    ub = N.array([0.01,0.9])
```

```
x_opt,f_opt,nbr_iters,nbr_fevals,solve_time = dfo.nelme_modified(cost,xstart=x0,lb=lb,ub=ub,x_tol=1e-
3,f_tol=1e-2,debug=False)
[damping_psi_opt, damping_phi_opt] = x_opt

# Load model
model=load_fmu(os.path.join(curr_dir, 'Pendulum_PModel.fmu'))

# Set optimal parameter values into the model model.set('damper.d',damping_psi_opt)
model.set('damper1.d', damping_phi_opt)
opts = model.simulate_options()
opts['filter'] = ['revolutePhi.phi','revolutePsi.phi']

# Simulate model response with optimal parameter values
res = model.simulate(start_time=0., final_time=12)
# Load optimal simulation result
phi_opt = res['revolutePhi.phi']
psi_opt = res['revolutePsi.phi']
t_opt  = res['time']

if with_plots:
        plt.figure(1)
        plt.subplot(2,1,1)
        plt.plot(t_opt, psi_opt, linewidth=1, label='Optimal data')
        plt.plot(t_meas, psi_meas, linewidth=1, label='Physical data')
        plt.legend()

            plt.subplot(2,1,2)
        plt.plot(t_opt, phi_opt, linewidth=1, label='Optimal data')
        plt.plot(t_meas, phi_meas, linewidth=1, label='Physical data')
        plt.legend()
        plt.show()

if __name__=="__main__":
        run_demo()
        pass
```

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

Document name
MASTER´S THESIS

Date of issue
February 2016

Document Number
ISRN LUTFD2/TFRT--6002--SE

Author(s)
Sara Gunnarsson

Supervisor
Maria Henningsson, Modelon
Anders Robertsson, Dept. of Automatic Control, Lund University, Sweden
Karl-Erik Årzén, Dept. of Automatic Control, Lund University, Sweden (examiner)

Sponsoring organization

Title and subtitle

Evaluation of FMI-based workflow for simulation and testing of industrial automation applications

Abstract

The Modelica language is an object-oriented, equation-based language used for modeling and design in many different domains and for various physical applications. In order to exchange models both between Modelicabased tools and other tools the Functional Mock-up Interface (FMI) has emerged. This interface enables exchange of virtual models between manufacturers and/or divisions within a company. The industry is moving towards increased development of virtual models, both for understanding of systems but also because it lowers the development costs.

In this thesis a proof-of-concept of the newly released FMI support in Automation Studio is tested. With a physical model of a reaction wheel pendulum as base, a virtual model is designed in the Modelica-based simulation tool Dymola. A reaction wheel pendulum is a pendulum equipped with a motorized wheel, which is run in different directions in order to swing up the pendulum and to be able to balance it at its highest point. The Dymola model is converted into a Functional Mock-up Unit (FMU), which is the implementation of the FMI. Since not all values of the pendulum are known, an estimation script is written and run in JModelica.org. The FMU is then modified with these estimated values and tested with the controller included together with the reaction wheel pendulum in Simulink, in order to verify that these work together. Lastly the FMU is imported into Automation Studio together with the controller, imported with Automation Studio Target for Simulink, and simulated together there.

The workflow of the thesis was successful and the controller managed to control the FMU in Automation Studio.

Keywords

Classification system and/or index terms (if any)

Supplementary bibliographical information

ISSN and key title
0280-5316

ISBN

Language
English

Number of pages
1-53

Recipient's notes

Security classification