Model-based design of industrial automation solutions using FMI

Charlie Erwall

Oscar Mårtensson

LUND
UNIVERSITY

Department of Automatic Control

MSc Thesis
TFRT-6016
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

This thesis defined and investigated a general workflow based on model-based design using the Functional Mock-up Interface (FMI), involving Hardware-in-the-Loop (HiL) simulation. The thesis was a direct continuation of Sara Gunnarsson's master's thesis "Evaluation of FMI-based Workflow for Simulation and Testing of Industrial Automation Applications", where a Software-in-the-Loop (SiL) simulation of the B&R Reaction Wheel Pendulum was conducted in Automation Studio using a model imported with FMI. A HiL simulation of the pendulum was performed to complete the work done by Sara, thus showcasing the strength and possibilities of using FMI in testing. The performance of the HiL results were evaluated by comparing the settling time with the SiL test and the real process swing-up.

In addition to the pendulum work, this thesis also aimed to perform model-based tests of the ABB IRB340 FlexPicker robot, including SiL and HiL simulations. This was done in order to define a general workflow for conducting tests using FMI, and to verify the approach on a more complex process than the pendulum. A MapleSim model of the robot was exported as a Functional Mock-up Unit and imported in Automation Studio, where the testing was done.

The results of the pendulum test showed that a HiL simulation with an FMU can be performed. The HiL simulation produced a settling time of 2.55 s at best, compared to 2.46 s of the SiL simulation and 2.28 s of the process. For the FlexPicker, the SiL and HiL tests were never run due to a lack of time. Instead, a recommended approach for implementing the SiL and HiL test—along with two less promising approaches tested—were discussed and evaluated. The conclusion is that the workflow and model-based design using FMI is a promising way of conducting tests, but that there is more implementational work needed before SiL and HiL results of the FlexPicker can be successfully collected.

Acknowledgements

Hereby, we would like to extend our thanks to our supervisors Anders Robertsson at LTH and Christian Tallner at B&R Automation for giving us the opportunity to do this master's thesis and for providing us with much needed help along the way. Also, thank you to Kurt Zehetleitner and Christoph Neukamp for supporting us from the B&R headquarters in Austria, and for providing a direction for the project from B&R's side. Also from B&R Austria, we thank Leopold Griessler for help with the simulation mode in Automation Studio. Thank you to Elias Palmqvist, Maria Henningsson and Per-Ola Larsson at Modelon for allowing us to access the material from Sara's thesis, and all the help regarding FMI and Dymola. Thank you to Patrik Lilja and Johan Malmberg at the B&R office in Malmö for being available and helping us with Automation Studio. Lastly, a thank you to Adam Bäckström for clarifications regarding both his and Kristofer Rosquist's theses.

Acronyms

AR Automation Runtime

AS Automation Studio

CPU Central Processing Unit

EPL Ethernet POWERLINK

FMI Functional Mock-up Interface

FMU Functional Mock-up Unit

GUI Graphical User Interface

HiL Hardware-in-the-Loop

ICN Intelligent Control Node

I/O Input / Output

PLC Programmable Logic Controller

SDRAM Synchronous Dynamic Random-Access Memory

SiL Software-in-the-Loop

TCP Tool Center Point

Contents

1

Introduction

1.1 Background

In today's industry, simulation of large complex systems become increasingly important in the development stage of production. Interaction between components can be investigated without the need of actual hardware which makes it possible to detect and correct design flaws already at an early stage of development. Testing can also be done in a safer way, since there is no risk of damaging expensive components.

As of today, there are many simulation software solutions which are very good at modeling specific functions and sub-parts of a complete system such as electrical and thermal behavior, deformation, multibody systems etc. What lacks is a general ability to easily connect all of these programs and subsystem models together to simulate them as a whole unit. For example, a car is built up by several sub-parts such as motor, gear box, chassis, exhaust system, which all need to be tested together to make sure that all communication between these sub-systems work as intended. The solution to this problem is the Functional Mock-up Interface (FMI) [Blochwitz et al., 2012], which is a standardized interface that allows all models—regardless of which modeling program that is used—to be able to communicate with each other.

While the FMI has already gained ground in the automotive industry, it is still not widespread in the automation industry. B&R is the first automation company to support FMI [FMI-Standard, 2016], which has been released in the latest version of their software Automation Studio [Bernecker & Rainer Industrie-Elektronik GMBH, 2016]. As a way of showcasing the versatility and power of the FMI interface in the automation area, this thesis focuses on defining a workflow for model-based testing using FMI, and also to perform two types of simulations to verify this workflow: Software-in-the-Loop (SiL) and Hardware-in-the-Loop (HiL). In the SiL simulation, the setup consists of the controller and the model, which is simulated

exclusively in a software environment. In the HiL simulation, hardware is added to the loop to gain more information about the real setup. The SiL and HiL simulations will be performed for two different processes: a reaction wheel pendulum [B&R Automation, 2009] by B&R Automation and an ABB IRB340 FlexPicker industrial delta robot [ABB Robotics Products AB, 2000].

1.2 Previous Work

This master's thesis is a continuation of Sara Gunnarsson's joint master's thesis "Evaluation of FMI-based Workflow for Simulation and Testing of Industrial Automation Applications" [Gunnarsson, 2016] between Modelon and B&R Automation. Sara verified the use of FMUs in Automation Studio (AS) by performing a SiL simulation of a B&R reaction wheel pendulum. The pendulum model was built by Sara in Dymola and exported as an FMU, and the controller was C-code generated from Simulink using the B&R Automation Studio Target for Simulink. The process was successfully controlled by the controller inside the AS environment, and produced satisfactory results. The pendulum model developed by Sara is further used for HiL tests in this project.

Another two master's theses titled "Modelling and Control of a Parallel Kinematic Robot" [Rosquist, 2013] made by Kristofer Rosquist for B&R Automation, and "Time-Optimal Control by Iterating Forward and Backward in Time" by Adam Bäckström also contributed to this project. The aim for Kristofer's thesis was to model the kinematics of an ABB IRB340 robot [ABB Robotics Products AB, 2000], and implement the control using software and hardware from B&R Automation. The implementation of the robot dynamics (inverse kinematics) and control in Automation Studio is what was primarily used from his thesis. Adam's thesis concerned a way of generating time-optimal trajectories, which is not a focus of this thesis, but he also worked with the ABB IRB340 and improved upon Kristofer's code, kinematics and dynamics which was useful for this thesis.

1.3 Goals

The primary goal of this thesis is to define a workflow for conducting model-based design consisting of FMUs exported from different software, and that they can be imported in Automation Studio and act as process substitutes for HiL testing. The first sub-goal is to verify that the virtual model of the pendulum that Sara Gunnarsson created in her thesis can be used in a HiL test. After that, focus lies on verifying that the HiL simulation—using the same workflow—works for a more advanced process that has more practical value than a pendulum. The choice of this process is the previously mentioned ABB IRB340 industrial robot. This is a suitable choice of

process since both Kristofer Rosquist's and Adam Bäckström's theses produced material which can be reused to reduce the scope of the project. In addition, the robot is used in today's industry and would be a relevant showcase for B&R Automation.

1.4 Limitations

This thesis project contains no controller design neither for the pendulum nor the FlexPicker robot. The pendulum uses a B&R controller that comes with the pendulum, and the FlexPicker robot utilizes the built-in cascade control structure found in the B&R ACOPOS servo drives [B&R Automation, 2016a]. In general, time will primarily be spent on implementation utilizing existing functions in order to reach the goals that were set up for this project.

The modeling software used is restricted to Dymola [Dassault Systèmes, 2016] and MapleSim [Maplesoft, 2016] since they support FMU export and licenses are available for use in this thesis. Since this thesis is a collaboration between the Department of Automatic Control at Lund University and B&R Automation, Automation Studio studio is the software that is used for writing code for the PLCs and for testing. It is also the reason why both the studied processes use B&R control systems.

1.5 Division of labor

Throughout the project, in general, the students have not been assigned sub-tasks within specific areas. The focus has always been for both the students to have knowledge of all parts of the project, but the workload has of course always been split between the students in order to increase work efficiency. For example, during the work with the FlexPicker robot, Charlie focused more on figuring out what to salvage of Kristofer's and Adam's AS programs while Oscar focused more on the MapleSim models and how to adjust them for the needs of this specific thesis.

2

Theory

2.1 Functional Mock-up Interface

Functional Mock-up Interface (FMI) [Blochwitz et al., 2012] is a standard for model exchange and co-simulation of dynamic models that had its first version released in January 2010, followed by a second version in July 2014. As more tools begin to support FMI, it becomes viable to use as part of a tool-chain for testing and improving industrial control systems. To have the possibility of using many different tools using a common interface makes it much easier to integrate models into the development process of automation applications. More information on the interface and its development can be found on the FMI web page [FMI-Standard, 2016].

Another large benefit of using FMI is that many of the tools that support it are based on the Modelica language [The Modelica Association, 2012], which builds models upon sets of equations instead of causal relations. This gives the user freedom to choose the input and output of the model when exporting it using FMI. This is highly beneficial when using models in a control architecture with feed-forward, since the same model that can be used in testing can also be used in the controller, only with different export settings. For instance, if the tool of a milling robot is replaced with a cutting tool the model could be modified accordingly, a new FMU exported to replace the old one and thus the load feed-forward has been corrected for the new tool, without the hassle of recalculating the dynamics equations by hand.

A component that follows the FMI standard is called a Functional Mock-up Unit (FMU) and consists of a zip-file, although using .fmu as file extension instead. The archive contains an XML file and C-functions in source code or binary format. The XML file contains a model description and variable declarations. The C-functions provide the model equations and optionally the solver [Blochwitz et al., 2012].

2.2 Hardware

B&R reaction wheel pendulum

The B&R Reaction wheel pendulum (Figure 2.1) consists of a base segment upon which a pendulum arm is attached. At the end of the arm, an aluminium wheel is mounted. When the aluminium wheel is spun, it creates a torque in the opposite direction which forces the pendulum arm to start swinging. With the right controller action, one can force the pendulum arm to swing to an upwards position, and make it balance there. This is the control goal for the pendulum in this project. The process is suitable for study since it is a relatively simple and well known one.

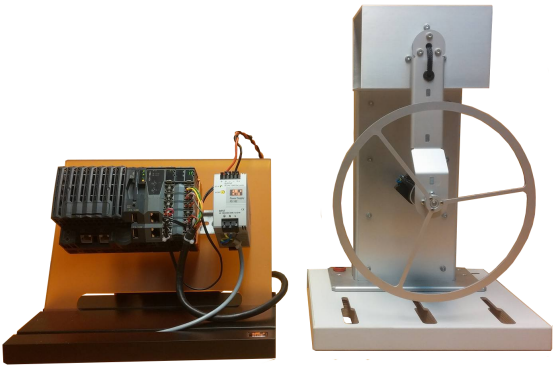

Figure 2.1 The B&R reaction wheel Pendulum with included PLC and power supply unit.

Controller

The controller used for the pendulum is a Simulink controller designed by B&R that comes with the pendulum and is known to perform well. For the sake of an overview, the control structure will be briefly presented here.

The controller is built up by two parts: a bang-bang controller for the swing-up part, and a state feedback controller for balancing the pendulum arm in its upright position. The states of the state feedback are the arm angle (ψ), the wheel angle (ϕ) and the respective angular velocities ($\dot{\psi}$ and $\dot{\phi}$). The angle measurements are supplied by encoders on the motor axis and the arm joint. The controller switches from the bang-bang controller to the balancing controller once the arm angle comes within a certain threshold according to $||\psi| - \pi| < 0.25$, i.e., when the arm is approximately 15 degrees from its upright position.

An overview of the controller can be seen in Figure 2.2. Here, a *State Variable Estimation* block can be seen as well, which will not be commented on. A more

detailed view of the *Extended Controller* block can be seen in Figure 2.3.

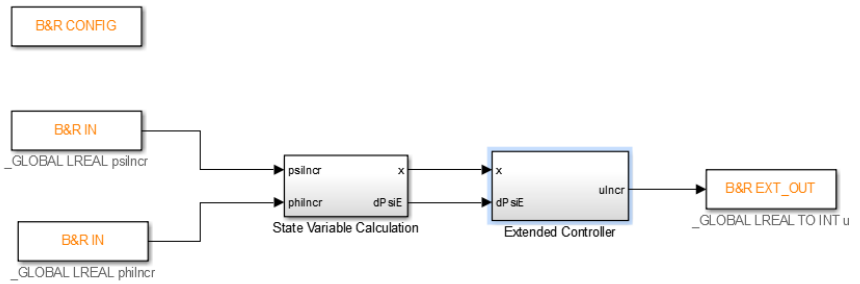

Figure 2.2 An overview of the pendulum controller. The input can be seen to the far left, followed by a *State Variable Calculation* block and after that the *Extended Controller* block, which is the core control structure. Lastly, the output can be seen to the far right.

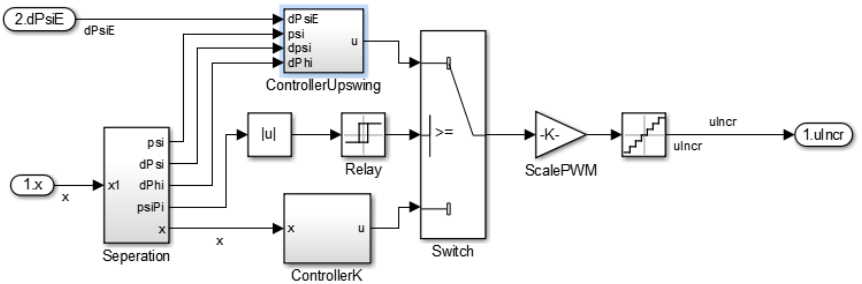

Figure 2.3 A more detailed view of the *Extended Controller* block seen in Figure 2.2. The *ControllerUpswing* block is the bang-bang controller that controls the swing-up of the pendulum, and the *ControllerK* is the state feedback controller that controls the balancing of the pendulum arm when in upright position. Between the two of them, there is a *Switch* block that switches between the two controllers

ABB IRB340 FlexPicker

The IRB340 FlexPicker is an industrial parallel-kinematic robot developed and manufactured by ABB. The robot excels at handling smaller objects up to 1 kg such as assorted chocolates or mini-sausages. With light weight and strong carbon fibre arms, the robot can accelerate the tool up to 15 g's of acceleration [ABB Robotics Products AB, 2000], enabling it to work at a very fast pace while still maintaining stability and high precision. The robot can be seen in Figure 2.4.

Figure 2.4 The ABB IRB340 FlexPicker robot [Bäckström, 2014].

B&R ACOPOS 1045 servo drive

8V1045.00-2 The ACOPOS 8V1045.00-2 drive (Figure 2.5) is a servo drive from B&R used in this project to supply a motor with power. The drive has 4 plug-in module slots, where for example POWERLINK and resolver interfaces can be installed. The drive also has a built-in 4th order trajectory generator which ensures a smooth transition from the input value to the output value of the ACOPOS. More details can be found in the ACOPOS User's Manual [B&R Automation, 2015].

Figure 2.5 The ACOPOS 1045 Drive (left) [B&R Automation, 2015] and the ACOPOS setup used to control the FlexPicker robot (right).

8AC114.60-2 Plug-in module for ACOPOS drives providing POWERLINK interface.

8AC122.60-3 Plug-in module that provides the drive with a 10kHz resolver interface used for communicating with the motors.

Controller

In addition to physical features, the ACOPOS drive house a cascade controller consisting of position, speed and torque/current loops. The overall structure can be seen in Figure 2.6. The position controller is a PI controller which runs on 400μs cycles. It is also equipped with both anti-windup and feed forward control. The position feedback comes directly from the resolver that reads the motor position. This value is then both fed to the position controller, and differentiated and sent to the speed controller. More detailed views of the position controller, speed controller and torque controller can be seen in Figure 2.7, Figure 2.8 and Figure 2.9 respectively [B&R Automation, 2016a].

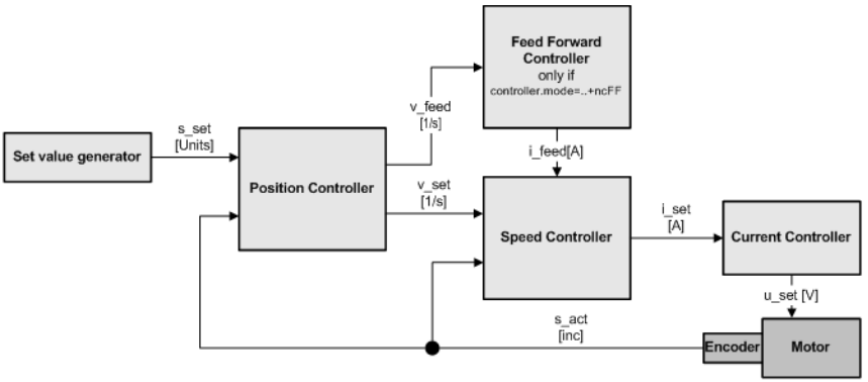

Figure 2.6 An overview of the cascade controller available in the ACOPOS drive [B&R Automation, 2016a].

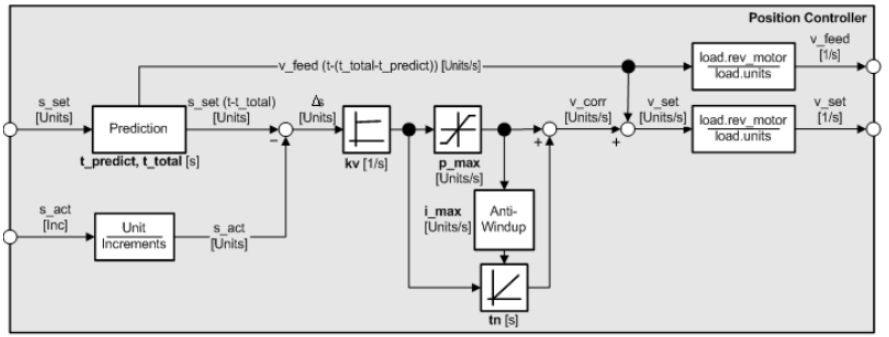

Figure 2.7 A more detailed view of the position controller block from Figure 2.6 [B&R Automation, 2016a].

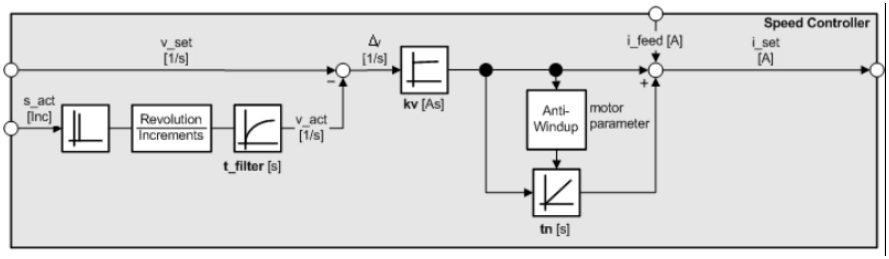

Figure 2.8 A more detailed view of the speed controller block from Figure 2.6 [B&R Automation, 2016a].

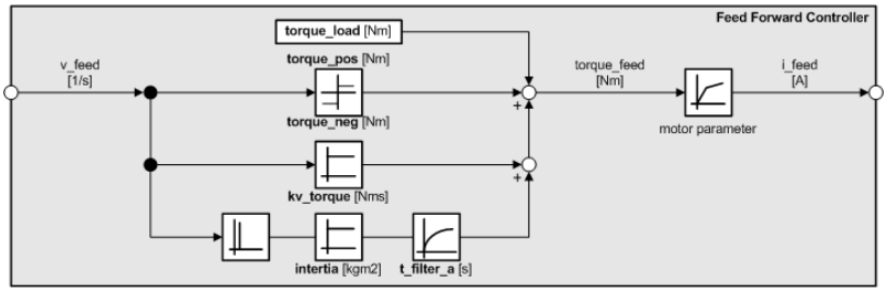

Figure 2.9 A more detailed view of the feed forward torque controller block from Figure 2.6 [B&R Automation, 2016a].

ACOPOS simulation

The ACOPOS drive has the possibility to be run in a simulation mode. This is for example useful when the hardware is absent or when one wants the whole functionality of the ACOPOS drive, but does not want to deal with having to connect to the drives. The ACOPOS can be put in two different simulation modes: standard and complete. In order to simulate the ACOPOS, the target system needs to support an AR version newer than or equal to A4.04.

Standard simulation is the most primitive mode, and does not utilize the cascade control structure of the ACOPOS drive seen in Figure 2.6. Instead, the input value is directly copied to the actual value. This simulation mode requires low CPU time, and does not take potential error conditions such as lag and overheating into account.

Complete simulation is a more advanced simulation mode. In contrast to the standard mode, it utilizes the whole cascade controller of the ACOPOS drive. In addition to the controller, this mode simulates the currents going to and from the motor, the motor temperature and bus processing stops. Complete mode is therefore necessary if maximum information about the system is desired, and more suitable for HiL simulations than the standard mode.

B&R PLCs

A Programmable Logic Controller is a computation unit commonly used in the automation industry. The PLC can be used to for example host controllers, visualizations, FMUs or other software programs. The X20 is a PLC serie with highly compact and modular PLCs ranging from low to high performance. In addition, each PLC can be equipped with several expansion modules, such as analog or digital I/Os or bus modules. A figure of a B&R X20CP158x PLC can be seen in Figure 2.10, but all PLCs used in the thesis look the same way on the outside. Choosing a PLC with the right performance can be crucial so that there is no lack of memory or computation power to perform the task running on the PLC.

X20CP1484 An industrial PLC from B&R Automation suitable for less computationally heavy tasks. The PLC is equipped with a 266 MHz Celeron 266 CPU and has 16 MB SDRAM. In addition, it has an EPL interface which allows it to communicate with for example the ACOPOS drives. This PLC is suitable for hosting FMUs or controllers that don't need much computation power.

X20CP1486 A more powerful PLC than the X20CP1484 with 650 MHz Celeron 650 CPU and 64 MB SDRAM. Just like the X20CP1484, this PLC is equipped with an EPL interface. This PLC is suitable for hosting programs that require more memory than the X201484 has, such as visualization.

Figure 2.10 A B&R X20CP158x PLC [B&R Automation, 2016b].

X20CP1584 A newer generation PLC than the X20CP148x ones. Equipped with an ATOM E620T 600 MHz processor and 256 MB DDR2 SDRAM. The most important feature is that the X20CP1584 supports AR version A4.04 and later, which allows the PLC to host a complete ACOPOS simulation.

2.3 Software

Automation Studio

Automation Studio is a software development environment created by B&R. The software can be used for configuring drives, controllers and visualizations, as well as debugging and generating data files and plots. The ability to use one single environment for many stages of an implementation process makes Automation Studio a very powerful tool in automation. B&R has recently begun supporting the import of FMUs in Automation Studio, which further pushes the advantages of the software and enables the possibility to simulate larger systems more easily.

The Automation Studio version used during this thesis was initially the Beta version of AS 4.2.5, which was later upgraded to the official 4.2.5.388 version once it was released.

Automation Runtime

Automation Runtime is the software kernel which allows applications to be run on a target system. It runs on all of B&R's target systems, and makes the applications hardware-independent. It is based on a real-time operating system and ensures deterministic execution of the task cycles. In addition, it supports an array of programming languages such as C, Structured Text and Sequential Function Chart to name a few.

Ethernet POWERLINK

Ethernet POWERLINK (EPL) [Ethernet POWERLINK Standardisation Group, 2013] is an open-source deterministic, real-time Ethernet protocol maintained by the EPL Standardization Group and is the standard used by B&R. It is based on the

existence of a managing node (master) and one or several controlled nodes (slaves). The cycle time for the network is decided by the managing node and it can be used as the synchronization base for the PLC as well. Additional PLCs can be added to an EPL network as controlled nodes in Automation Studio by adding Intelligent Controlled Node (ICN) modules in the hardware configuration.

MapleSim

MapleSim [Maplesoft, 2016] is a modeling, simulation and analysis tool developed by Maplesoft. The software is built on the mathematical engine of Maple, which allows it to utilize the symbolic computations and equation simplification features of Maple. The modeling is done graphically by dragging and dropping component blocks with different attributes, which are then connected with each other to form larger systems. This makes MapleSim suitable for modeling hydraulic, multibody and electrical systems to name a few. The blocks are described by acausal equations, which means that energy can flow in both directions in the system, and that no direction for the flow of energy in the system needs to be chosen. For example, one could build a model of a wind turbine meant for generating a current by feeding the rotor with an air flow. With the acausal feature, the turbine could also be fed with a current, which would then cause the rotor to rotate.

MapleSim Connector for FMI

In addition to being a powerful modeling tool, MapleSim has an add-on which allows the models to be exported as FMUs, both in co-simulation and model exchange mode. The co-simulation mode comes with a choice of five different embedded fixed-step solver variants: Implicit and Explicit Euler, and Runge-Kutta order 2, 3, and 4. The properties of these solvers will not be discussed in this thesis, but details can be found in [Ascher and Petzold, 1997]. Note that the option to export an FMU using variable step-size solvers is not available in MapleSim.

Simulink

Utilizing much of the functionality in Matlab, Simulink is a graphical programming environment for modeling, simulating, and analysing multi-domain dynamic systems created by Mathworks. Many add-ons are available from both Mathworks and other companies which gives the software leverage functionality in different areas such as multibody modeling, automatic code generation, and FMI export/import.

Automation Studio target for Simulink

Automation Studio Target for Simulink is an add-on toolbox for Matlab and Simulink created by B&R Automation that allows the user to export either C or C++ code directly to Automation Studio. With this toolbox, controller design or modeling, for instance, could be done in Simulink and then the system could be exported to Automation Studio for further use.

2.4 FlexPicker robot dynamics

All robot dynamics calculations were done by Adam Bäckström in his thesis [Bäckström, 2014], which in turn builds on Kristofer Rosquist's thesis [Rosquist, 2013]. A summary of the resulting calculations are presented here to provide an overview for the reader. All details can be found in Adam's report.

The FlexPicker Robot

The FlexPicker (ABB IRB340) robot has three identical arms located equiangular from each other at a 120 degree angle. The whole arm consist of an upper arm, an elbow joint and a lower arm made up from two parallel arms. The underarm connects to a travelling plate, where tools and gripping modules can be attached. In addition to the three identical arms, a fourth optional middle arm can be attached to allow rotation of the tool attached to the travelling plate. This arm will however not be used in this project. No tool will be attached, so the Tool Center-Point (TCP) reference will be assumed to be in the middle of the travelling plate, even though the option to vary this variable will be available in the implementation. Figure 2.11 shows a sketch of an individual robot arm and all of the sub-part names.

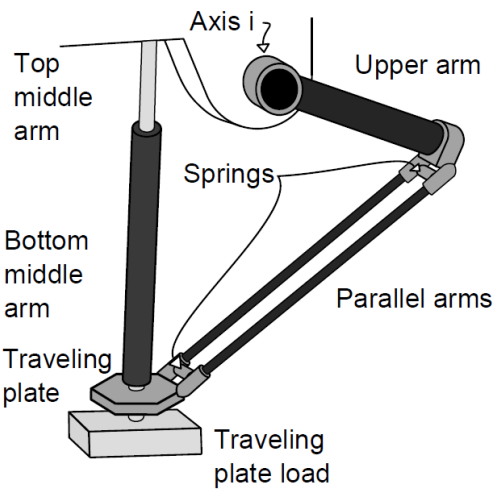

Figure 2.11 A sketch of one of the three robot arms, showing the name of each component [Bäckström, 2014].

Inverse Kinematics

The inverse kinematics of a parallel kinematic robot are a set of equations relating the TCP position in Cartesian coordinate space to the angle of each motor. Contrary to a serial robot, the motor angles of a parallel robot can be calculated independently. Specifically for a delta robot (3 arms) the functions become three separate mappings $\mathbb{R}^3 \rightarrow \mathbb{R}^1 : \phi_i = f(p_x, p_y, p_z)$, where i is the index of an axis and (p_x, p_y, p_z) are the Cartesian coordinates of the TCP.

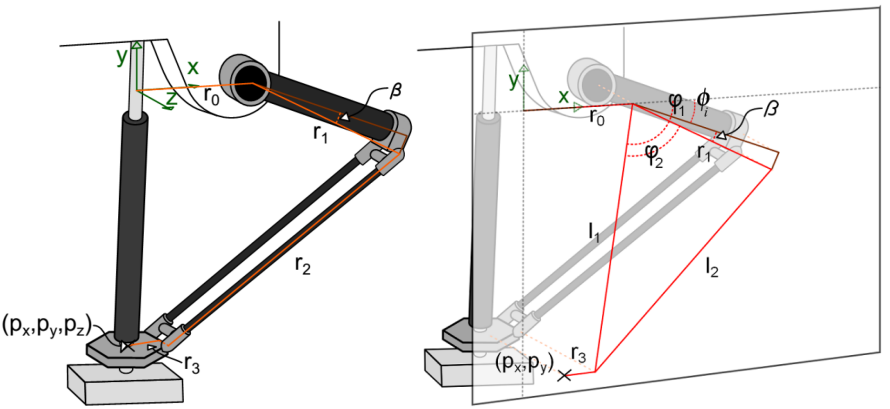

Figure 2.12 Constant parameters and coordinate system (left), projection plane and additional variables (right) [Bäckström, 2014].

A coordinate system is defined with the x-y plane being the plane of motion for a chosen motor axis, the x-axis being parallel to the zero position of the motor axis ($\phi_i = 0$). The z-axis is then well defined by the right-hand rule as it is perpendicular to the other two axes. The origin is chosen in the middle of the robot (so the distance to each motor axis is the same, r_0) and at the y-coordinate of the axes. The motor angle (ϕ_i) can then be unambiguously calculated after projecting the TCP position on the x-y plane. Figure 2.12 illustrates the plane and the projection as well as the measurements used to calculate the inverse kinematics. The final equation can be seen in Equation (2.1) with help-variables defined in Equation (2.2). This equation will result in an angle that is positive when below the x-axis and negative above, which is inverse to a regular polar coordinate system definition. A flip of the sign of Equation (2.1) will change the rotational direction to that of such a system [Bäckström, 2014].

$$\phi_i = \arccos \frac{p_x + r_3 - r_0}{l_1} - \arccos \frac{l_2^2 - l_1^2 - r_1^2}{2 r_1 l_1} - \beta \tag{2.1}$$

$$\begin{aligned} l_1 &= \sqrt{(r_0 - p_x - r_3)^2 + p_y^2} \\ l_2 &= \sqrt{r_2^2 - p_z^2} \end{aligned} \tag{2.2}$$

To do the same calculations for the other two axes, the coordinate system needs to be rotated around the y-axis so that the x-axis aligns with another motor axis. This is done by multiplying the coordinate vector with a rotational matrix dependent on the angle α between the axes, shown in Equation (2.3).

$$\begin{bmatrix} p'_x \\ p'_y \\ p'_z \end{bmatrix} = \begin{bmatrix} \cos\alpha & 0 & -\sin\alpha \\ 0 & 1 & 0 \\ \sin\alpha & 0 & \cos\alpha \end{bmatrix} \begin{bmatrix} p_x \\ p_y \\ p_z \end{bmatrix} \tag{2.3}$$

2.5 Software-in-the-Loop

Software-in-the-Loop is a way of testing used in the development process when for instance designing a new plant or a controller for a plant. The main purpose is to verify the performance of the whole setup and to make sure that the controller is able to control the plant as intended, and that the plant acts as intended under the influence of the controller. In this analysis, all interaction between the different processes happen inside a software environment, for example B&R's Automation Studio. The plant is represented by a virtual model of some sort, which can be built in MapleSim or Simulink for instance. The controller is then simulated together with the virtual model of the plant inside some suitable software on the same hardware where the results are collected and analysed.

2.6 Hardware-in-the-Loop

Hardware-in-the-loop is, just like SiL, a verification step in the testing process. Instead of running both the plant and the controller virtually on shared hardware, as in SiL, the virtual model of the plant is typically placed on a computer or PLC while all other parts of the complete chain are run on the real hardware. The computer or PLC that hosts the virtual model of the plant is then connected physically to the real hardware, and a simulation of the whole system is performed. The goal with this way of simulating is to come as close as possible to the real process. Unlike SiL, a HiL simulation implicitly takes additional physical quantities into account such as currents, temperature and delays, thus making it a more accurate representation of reality than a SiL simulation.

3

Method

3.1 Workflow

The essence of this thesis is the workflow of model-based design using FMI. This section aims to generally define the components that constitutes this workflow. Figure 3.1 illustrates the steps that will now be described in some detail.

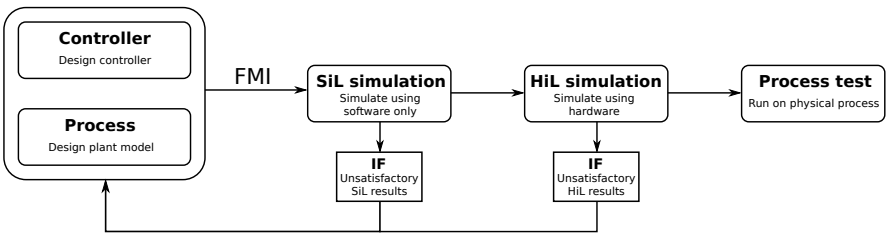

Figure 3.1 The proposed workflow when creating a HiL simulation in Automation Studio using an FMU as virtual model.

Modeling The first step in the workflow is the creation of a model of the process in question, within a software that supports FMI export such as Dymola or MapleSim. The model's inputs and outputs should be chosen so they correspond as well as possible to the inputs and outputs of the process with regard to units, value ranges and data types. With the model complete it should be exported with settings compatible with the target software, e.g., co-simulation or model exchange.

When importing an FMU the implementation may differ among different software although the interface of FMI guarantees that the functionality is the same. After importing the FMU into the target software, declaration of variables for inputs and outputs and corresponding variable mapping may be required for—or simply to facilitate—the coming verification steps.

SiL simulation Before testing any hardware, one should make sure that the desired setup works as intended and meets the requirements in an all-software simulation. Here, the different sub-parts of the whole process such as motors, drives, controllers and FMUs should be connected and simulated together in a suitable software environment such as Automation Studio. It is important to always strive towards getting as close as possible to the real world setup.

The SiL simulation consists of some sort of performance test—for instance a step response—where the controller acts on the FMU. The controller is then modified based on data obtained from such tests and the testing is iterated until the behavior is satisfactory. The testing could also be used the other way around, to develop a model using an already well-performing controller. This could be useful in the scenario described in Section 2.1, where it is suggested that an FMU could be used as a dynamic load model in a feed-forward control architecture.

HiL simulation This step is a direct continuation of the SiL simulation, but now including hardware instead of an all-software simulation. With most of the development done in SiL simulation, this is a final testing and verification step before running a controller with a real process. A rule of thumb is to include as much hardware as possible in order to come as close as possible to the real setup. The success of a HiL simulation depends largely on how well the FMU and controller perform, but there may be additional configurations that may need to be tweaked in this step.

The main difference from a SiL simulation is that the FMU and the controller should be executed on independent hardware, e.g., two separate PLCs. The controller should run on the same hardware using the same software that is to be used in the final step. The hardware for the FMU should simply be chosen such that it executes without problems. Repeating the same performance tests as in the previous step the controller (or model depending on what is tested) should be modified and tuned until the HiL simulation provides results vsimilar to those from the SiL simulation.

Process test The final step in the proposed workflow is to test the real process with all hardware involved. The FMU is basically replaced by the corresponding hardware and the controller should be acting on the real process. The purpose of the workflow thus far is to make sure that the controller will run smoothly on the target hardware, not damage any process components and perform close to what is expected, but some final tuning is most likely needed as a simulation never fully includes all details of the real world setup.

3.2 B&R reaction wheel pendulum

This section will describe the hardware and software setup as well as implementation of the tests for the pendulum. The physical setup for all the tests of the pendulum can be seen in Figure 3.2.

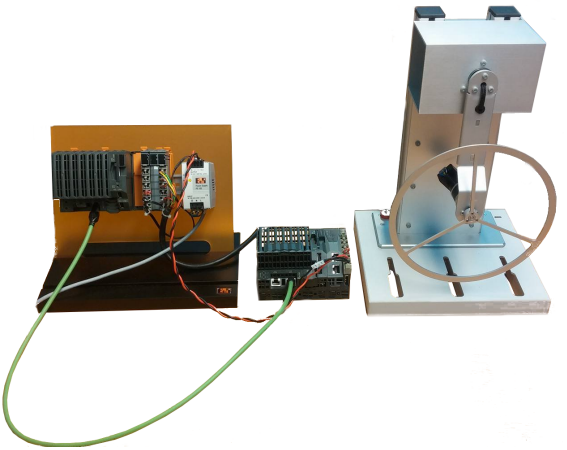

Figure 3.2 The setup for HiL and process test. A PLC can be seen to the far left attached to an orange plate, with a power supply unit to the left of the PLC. The pendulum can be seen to the far right.

Modeling

A model of the B&R reaction wheel pendulum was made by Sara Gunnarsson in her master's thesis project [Gunnarsson, 2016]. The model, seen in Figure 3.3, was made in the Modelica-based software Dymola. It consists of a damped revolute joint, *revolutePsi*, anchored to a world component *world1* and also connected to a body shape *Pendulum* representing the mass and inertia of the pendulum arm. Another revolute joint, *revolutePhi*, connects the other end of the arm to an additional body shape representing the mass of the wheel and rotor as well as the inertia of the wheel. This revolute joint is driven by the motor and its frame of reference is the arm. The input to the system is the motor voltage $u \in [-24, 24]$ volts. The outputs of the system are the angles in radians of the two revolute joints in the reference frame of the world component. The joint at the arm attachment is already in that reference frame and the motor-driven joint is converted to it simply by adding the two joint angles. The angles are allowed to overflow and are thus unbounded, but within one full revolution of each joint the angles are naturally $\varphi, \psi \in [0, 2\pi]$ radians.

Since AS only supports source-code generated co-simulation FMUs, this setting was used to export the FMU from Dymola. The only solver that is supported with

these settings in Dymola is CVODE [Hindmarsh et al., 2005], which was therefore used. The step size was set to $5 \cdot 10^{-3}$ (5 ms) and simulation time to 20 s, although these settings can easily be changed later in AS.

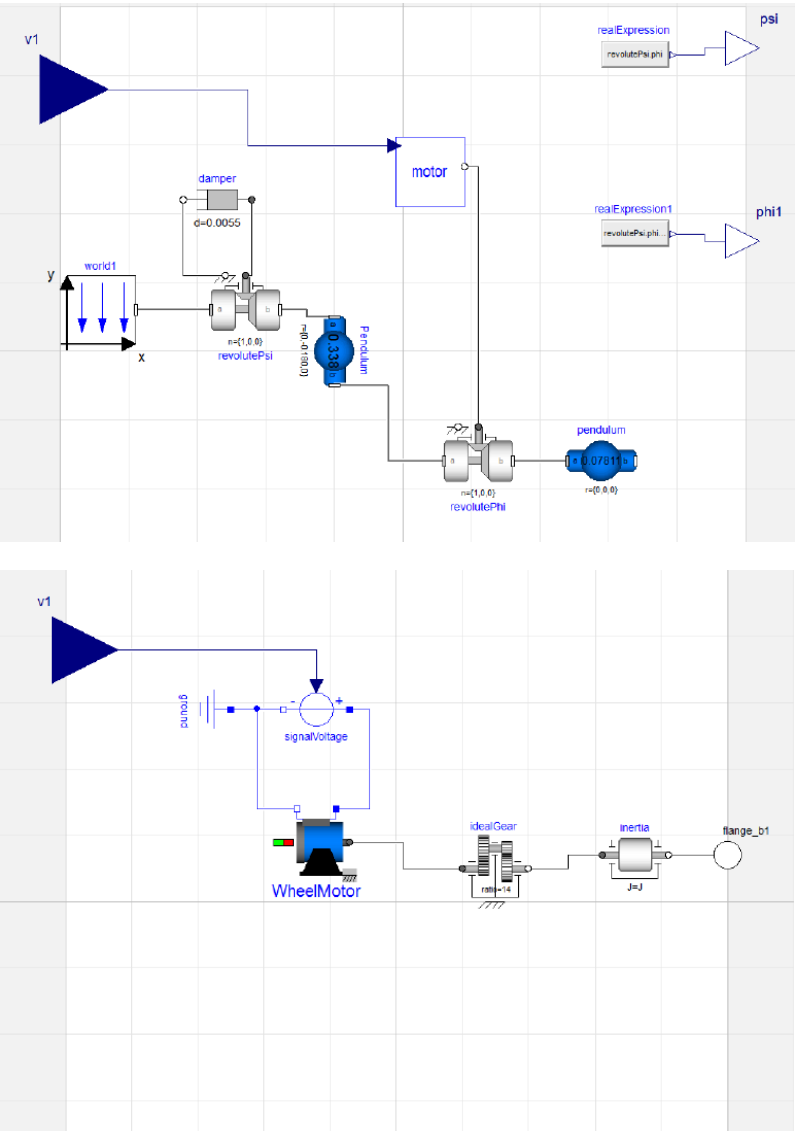

Figure 3.3 Pendulum model in Dymola. The whole system (upper) and the motor block (lower).

In order to be able to export a pendulum FMU with a different solver than CVODE, a model of the Pendulum was also made in MapleSim (Figure 3.4). This model was made to be as an exact copy of Sara's Dymola model as possible. Since almost all of the blocks used in the Dymola model were Modelica standard blocks, these could easily be found in MapleSim as well. In addition, the geometric and electrical parameters of the blocks were directly copied from the Dymola model and used in the MapleSim model.

The MapleSim model was exported as a source code FMU in co-simulation mode. The model was exported with a Backward Euler solver [Ascher and Petzold, 1997] with step sizes $1 \cdot 10^{-3}$ (1 ms) and $5 \cdot 10^{-3}$ (5 ms).

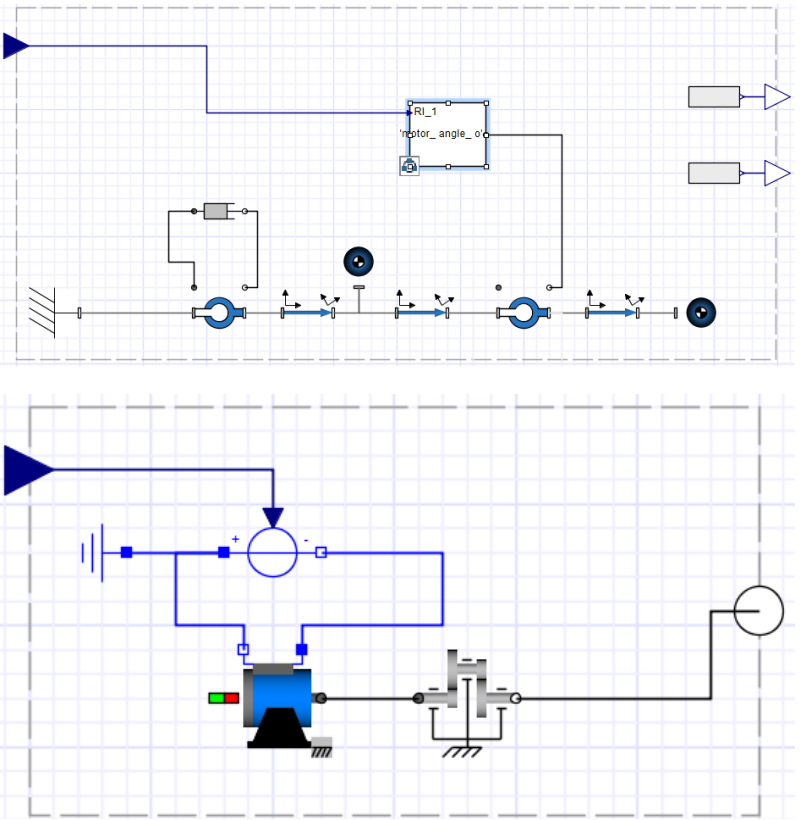

Figure 3.4 Pendulum model in MapleSim. The whole system (upper) and the motor block (lower). The inertia block missing in the motor subsystem compared with Figure 3.3 was removed simply because the value was zero and therefore useless to include.

SiL simulation

The SiL simulation for the pendulum was the main objective of Sara Gunnarsson's master's thesis [Gunnarsson, 2016] and was successful therein. The test was performed again following the steps in the thesis to try and reproduce the results. The general approach was the same, although with a few implementational differences described next.

During her thesis Gunnarsson modified an existing controller for the pendulum available in Simulink made by B&R such that the inputs and output matched those of the FMU. The original, unmodified controller's inputs are encoder increments, $\varphi \in [0, 28000]$ for the wheel and $\psi \in [0, 4096]$ for the pendulum arm and the output is a signed 16-bit integer $u \in [-32768, 32767]$ units. These values were used in the original controller since they are the input and outputs of the process. Another difference is that the reference frame for the wheel angle in the original controller is that of the pendulum arm. To easily be able to change between the process and the FMU the original controller was used and the changes made by Gunnarsson were implemented in the controller's main script instead. A macro was used to only include the changes for the HiL simulation and not the process. To convert the wheel angle back to the correct reference frame the arm angle ψ was subtracted from the wheel angle φ. The different intervals were simply a matter of scaling the inputs and outputs, mapping 28000 and 4096 increments to 2π radians for the wheel and arm respectively and finally mapping 32767 units to 24 volts for the control signal. The implementation of this can be seen in the controller's cyclic function in Section 8.2.

In the SiL simulation done by Gunnarsson the FMU was initialized at $\psi_0 = \pi/2$ radians. The process is usually started at rest, which is around the downright position i.e., $\psi_0 \approx 0$. However the controller will not output a control signal if it is started when the pendulum is in the exact downright position i.e., $\psi_0 = 0$ so the initial angle of the FMU was chosen as $\psi_0 \approx 2\pi/4096$ radians, which corresponds to 1 encoder increment.

The SiL test was simulated entirely in Automation Studio on the ARsim kernel.

Settling time

A quantity that can be studied with regard to performance of the controller is settling time. The settling time (t_s) can be defined as the time required for the response curve to reach and stay within a range of certain percentage (usually 5% or 2%) of the final value [Tay et al., 2012]. The final value of the pendulum is $|\psi| = \pi$, and the range (or error band) was chosen as $\pm 2\%$ of this value.

The end time—i.e., the last time the response curve enters the error band—was found by taking the last of all intersections between the error band bounds and the response curve. A function found on the MATLAB Central File Exchange was used

to find these intersections [NS, 2008].

HiL simulation

In this section the setup for the HiL test for the reaction wheel pendulum will be described in detail. The controller used was, as mentioned in the previous section, the original controller supplied by B&R for the reaction wheel pendulum with some slight modifications after code export.

For the HiL test two PLCs were required, one for the controller and one for the FMU. The PLCs available at the Automatic Control department were an X20CP1484 and an X20CP1486, so these were used. The first was used for the FMU and the latter for the controller and a visualization. The controller cycle time was set to 1 ms regardless of the cycle time of the PLC for the FMU. Only one task class was used on each PLC. The communication between the PLCs was done using EPL with the regulator PLC as managing node and the FMU PLC as controlled node. The PLC tasks were set to synchronize on the EPL, which was set to communicate with a cycle time of 1 ms since this is the cycle time of the controller.

The controller PLC was monitored during the execution. Since an FMU in Automation Studio—which is a function block—needs to be enabled an initialization command was added to the EPL, along with an "enabled" status variable. A reset command was also added, which simply sets the *Enable* input variable of the FMU function block to false. The init and reset commands also turn the controller on and off, respectively. It should be noted that the rest only worked for the MapleSim FMUs, whereas the Dymola FMUs crashed when being reset. To reset the Dymola FMUs the PLC was simply powered off. All the variables on the EPL can be seen in Table 3.1.

Name	Data type	Direction*	Description
psi	REAL	In	FMU arm angle
phi	REAL	In	FMU wheel angle
u	REAL	Out	Control signal
init	BOOL	Out	Enable FMU command
enabled	BOOL	In	FMU enabled status
reset	BOOL	Out	Reset FMU command

*From the managing node's perspective.

Table 3.1 EPL communication variables used in the pendulum HiL simulation.

The quantities that were studied were the same as in the SiL simulation, using the same tools.

Process test

Close to no effort had to be put in to get the real pendulum setup to work, since the B&R PLC included with the pendulum on delivery already had the controller presented in Section 2.2 on it. What remained to be done was just to run the setup, which worked great right away.

Graphical User Interface

To view the swing-up and balancing of the pendulum arm, a Graphical User Interface (GUI) was created in Automation Studio from the visualization tool that the software provides. The GUI consists of two different plot windows, one for the angle of the pendulum arm, and one for the control signal. There is also a button for starting the controller and for resetting the process under test. The GUI can be used for the SiL, HiL and process test. The GUI can be connected to over Ethernet with VNC viewer [RealVNC Ltd, 2015], where the IP adress of the target PLC hosting the visualization is needed. The angle part of the GUI can be seen in Figure 3.5 and the control signal part in Figure 3.6.

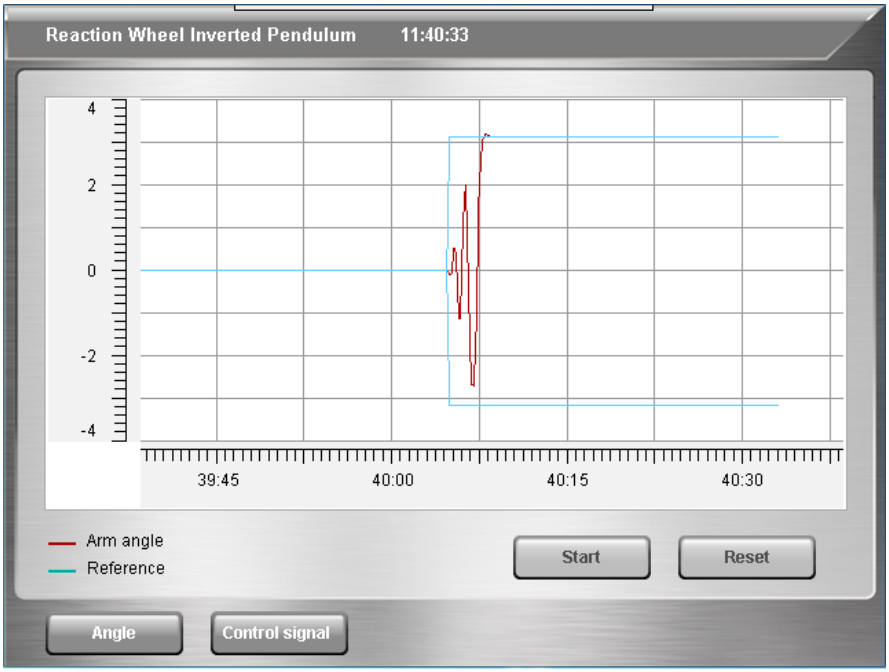

Figure 3.5 The angle part of the GUI with an example of the pendulum arm angle (red) seen during the swing-up and balancing phase. The reference angle values of the pendulum arm, $|\psi| = \pi$, can also be seen as a light blue curve.

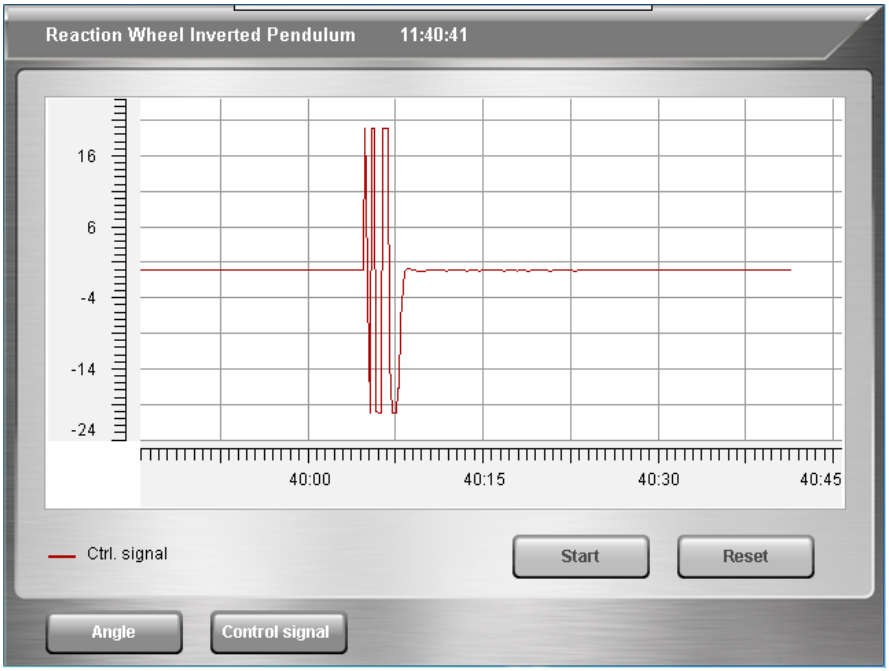

Figure 3.6 The control signal plot corresponding with the pendulum arm angle seen in Figure 3.5.

3.3 ABB IRB340 FlexPicker

This section describes the hardware and software setup as well as implementation of the tests for the FlexPicker. The main difference from the pendulum setup is the presence of the B&R servo drives. These were not included in the MapleSim model of the FlexPicker since AS contains support for simulating these drives as described in Section 2.2.

Modeling

A multibody dynamics software model of the ABB IRB340 FlexPicker robot was created in MapleSim by Kristofer Rosquist in his master's thesis [Rosquist, 2013] and improved by Adam Bäckström in his master's thesis [Bäckström, 2014]. This model was further modified in MapleSim 2016 during the course of the project so that it could be exported as an FMU. The resulting modified model can be seen in Figure 3.7 with its corresponding 3D model in Figure 3.8. An important simplification of the real robot model that was made in the original model is that the parallel

underarms are modelled as a single arm, but with the same moment of inertia and mass as the two parallel rods have.

Examples of changes that were made are the removal of ACOPOS and motor models that existed in the original FlexPicker model. These were removed since only the robot model was needed. Another change that was made was the addition of *Real Input* and *Real Output* ports, which are a requirement in MapleSim to be able to export the model as an FMU [Maplesoft, 2014]. This can be seen in Figure 3.7 with the real inputs characterized with blue triangle arrows, and the real outputs with white triangle arrows. The decision was made to feed the torque from the motor directly into the robot model. The main reason for this is that a real world motor output usually is torque, and that it is easy to work with since torque in electric motors is proportional to current according to the general equation $\tau = \alpha \cdot i$, where τ is the torque, α is some constant related to the motor parameters and i is the current. In addition, a *Real Signal* is easily converted to a torque in Maplesim with the *Torque* block, seen to the right of the input in Figure 3.7.

The output signals consist of the x, y and z coordinates of the TCP and the motor angles of the three motors.

The FMU was exported as source code in co-simulation mode with an Explicit Euler solver. The step size was initially set to $5 \cdot 10^{-3}$ (5 ms).

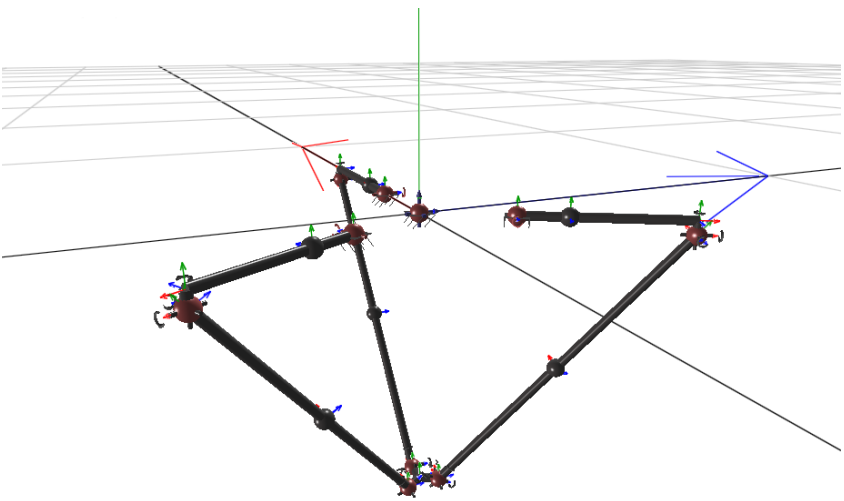

Figure 3.8 The MapleSim model of the FlexPicker robot seen in the 3D workspace of MapleSim.

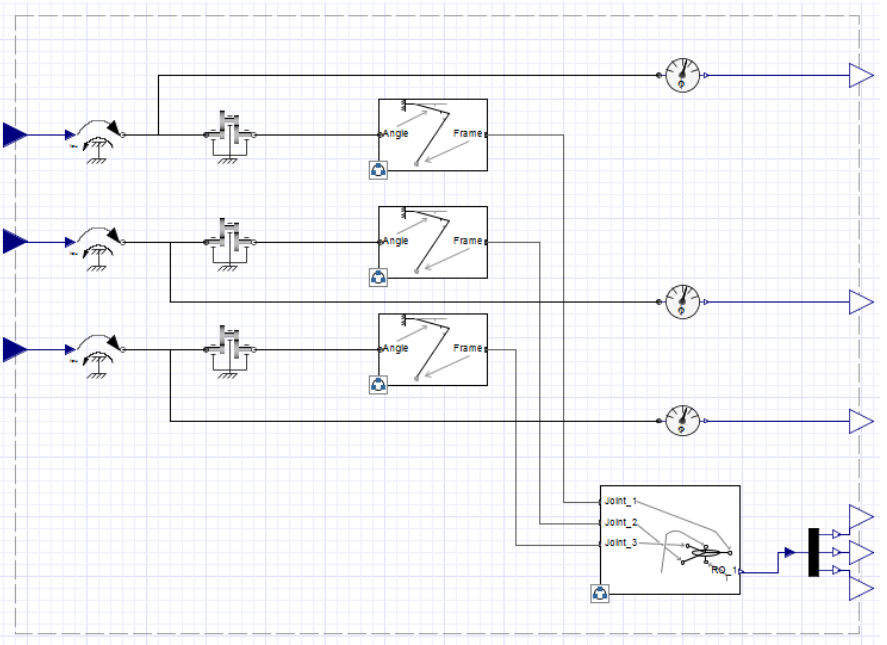

Figure 3.7 The MapleSim model of the FlexPicker robot. The robot has three torque inputs (left side of model) and six outputs (right side of model): the x, y and z coordinates of the TCP and the motor angle of the three motors.

SiL

The implementational goal with the SiL simulation for the FlexPicker was to successfully interface the FlexPicker FMU with the ACOPOS drives and motors in complete simulation mode. The idea was to use the torques generated by the motors in Automation Studio as inputs to the FlexPicker FMU, and feedback the motor angles FMU provided by the FMU. It is important to synchronize the motor angle of the FMU with the motors in Automation Studio so that there is no mismatch between them, since the position of the FMU should drive the control system in each ACOPOS.

Below follows a quick overview of the different parts of the Automation Studio project used in the SiL implementation.

Servo drives

The low level control of the motors in the FlexPicker was done using the B&R ACOPOS servo drives. To interface with these drives AS supports—among other methods—the standardized motion control library PLCOpen [Wal, 2009]. AS also includes an application example for two axes implemented in C code called

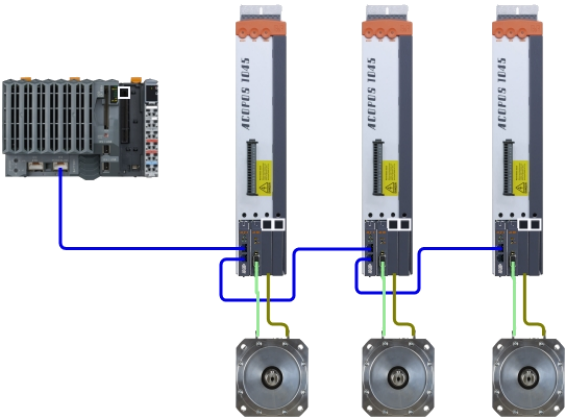

Figure 3.9 The hardware configuration for the SiL simulation which includes a PLC used to host the FMU, inverse kinematics and other programs. The ACOPOS servo drives and motors are also seen connected to the PLC.

LibACP10MC_MotionControl_C utilizing the PLCOpen library to control the axes, which was used with some minor modifications. The connection between the PLCOpen function blocks and the servo drives is done with axis objects of the type *ACP10AXIS_typ*, which are created along with the setup of the servo drives. For the SiL test, the drives were set to complete simulation mode which is covered in more detail in Section 2.2. Simulation mode for the target was also activated (requires AR version A4.24 or later). This allows for a PLC to be used in the hardware configuration instead of the Standard PC otherwise used for simulation in AS, which decreases the amount of additional work needed for the HiL simulation.

Motors

Since the ACOPOS drives in Automation Studio cannot be run in complete simulation mode without being connected to motors, these were necessary to add. The motors chosen were B&R 8JSA24.E9080D000-0 standard synchronous motors with 3 pole pairs, since these are similar to the actual motors of the FlexPicker. Each ACOPOS and motor pair was set in complete simulation mode by setting the mode in the configuration of the ACOPOS in the Physical View. The simulation was initiated automatically by setting the *CMD_SIMULATION* parameter in the ACOPOS parameter tables to the value *ncSWITCH_ON*.

A picture of the hardware configuration in Automation Studio consisting of the motors and ACOPOS drives connected as described can be seen in Figure 3.9.

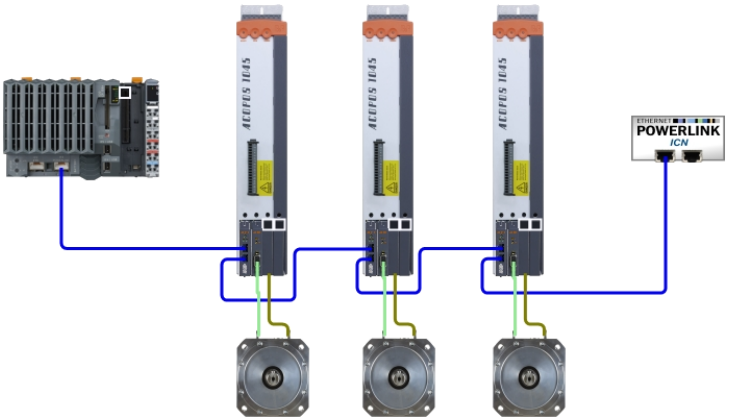

Figure 3.10 The hardware configuration for the HiL simulation which is identical to the SiL setup except for the added ICN hosting the FMU.

Inverse Kinematics

The inverse kinematics for the robot proposed in Section 2.4 was implemented as a cyclic program in Automation Studio directly in C. The program code can be seen in Section 8.3. In addition to the desired motor positions being calculated, they are also written to the status variables used by the axis programs described in Section 3.3.

Main program

The purpose of the main program was to tie together all of the parts mentioned above. The program initiates the drives and runs the homing procedure. The actual motor positions coming from the feedback of the FMU are cyclically written to the ACOPOS drives which in turn passes this on to the motors. Error handling and parsing is also done cyclically in order to make sure that everything is in order each cycle.

HiL

Similarly to the pendulum setup, the HiL simulation setup consisted of an additional component compared to the SiL simulation, namely an ICN. An X20CP1585 was used as the ICN—the same model as the controller PLC which was set to be the managing node. The controlled node was connected to the managing node via the same EPL network as the ACOPOS drives. The hardware setup is illustrated in Figure 3.10. The difference in software configuration from the SiL simulation is only the relocation of the FMU from the managing node to the controlled node.

Name	Data type	Direction*	Description
torque0	REAL	Out	Motor 0 torque
torque1	REAL	Out	Motor 1 torque
torque2	REAL	Out	Motor 2 torque
pos0	DINT	In	FMU axis 0 position
pos1	DINT	In	FMU axis 1 position
pos2	DINT	In	FMU axis 2 position
tcpX	REAL	In	TCP x-coordinate
tcpY	REAL	In	TCP y-coordinate
tcpZ	REAL	In	TCP z-coordinate

*From the managing node's perspective.

Table 3.2 EPL communication variables used in the FlexPicker HiL simulation.

The data flow in the HiL simulation was limited by the fact that only the managing node can communicate with the ACOPOS drives. Because of this, the position feedback from the FMU had to be communicated from the controlled node to the managing node and then written to the servo drives by the main program. Similarly, the actual motor torques could only be read from the drives by the managing node and then communicated to the controlled node. The TCP position was also transmitted from the controlled node to the managing node. All variables communicated between managing and controlled node are presented in Table 3.2.

Process test

The process test for the FlexPicker was never performed.

4

Results

4.1 Pendulum

SiL

The results from the SiL tests of the inverted pendulum are presented in this section. As mentioned in Section 3, the results have been collected by letting the pendulum swing up from a downward position ($\psi \approx 0$ rad) to balance at the reference position $|\psi| = \pi$. The SiL simulation results of the ψ angle for the Dymola model and the MapleSim model can be seen in Figure 4.1, and the control signal results of the SiL simulation can be seen in Figure 4.2. All data has been translated such that the control starts after 5 seconds for the sake of visualization.

Settling time

Figure 4.3 shows the balancing part of the control procedure, with the settling time marked by a solid vertical line.

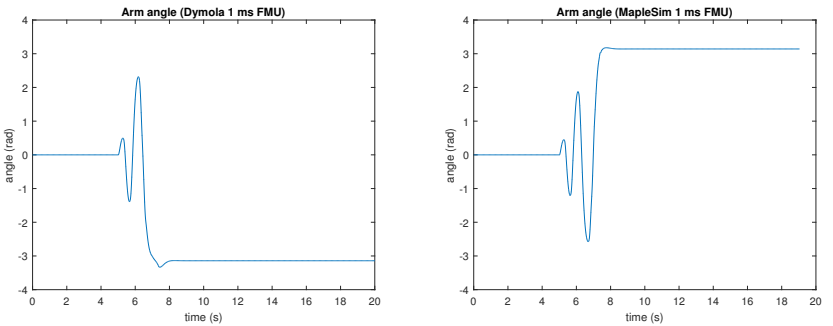

Figure 4.1 ψ angle of the Dymola FMU (left) and the MapleSim FMU (right) of the pendulum. The results are collected for models running on a cycle time of 1 ms.

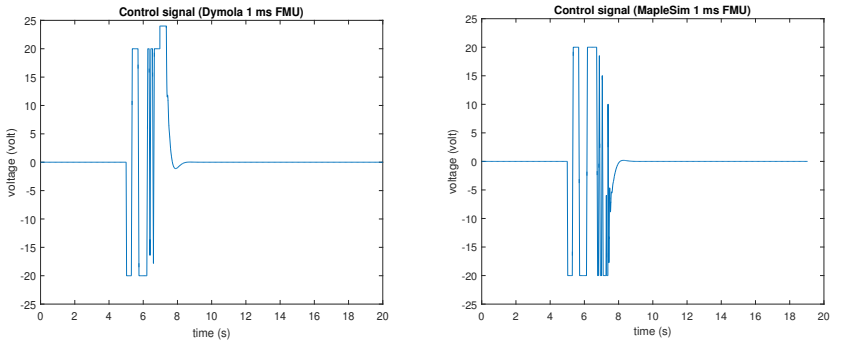

Figure 4.2 The control signal when controlling the Dymola pendumlum FMU (left) and MapleSim pendulum FMU (right). The results are collected for models running on a cycle time of 1 ms.

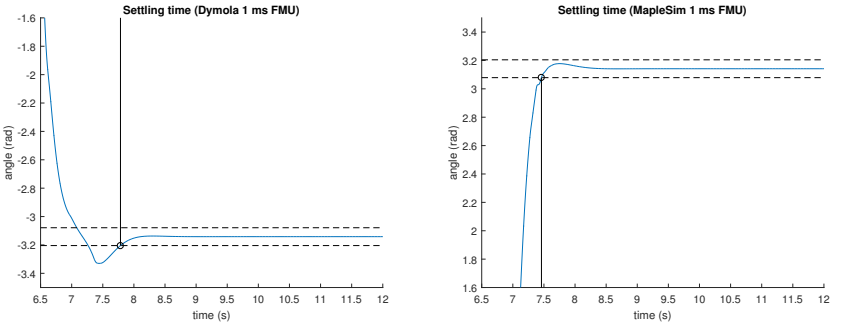

Figure 4.3 The settling time of the Dymola pendulum FMU (left) and the MapleSim pendulum FMU (right). The black dashed line marks the 2% error band bounds of the control goal $|\psi| = \pi$.

HiL and process

The results from the HiL and process tests are presented in this section. Figure 4.4 depicts the arm angle of the pendulum and Figure 4.5 depicts the control signal. All data has been translated such that the control starts after 5 seconds.

Settling time

Figure 4.6 shows the balancing part of the control procedure for both the HiL and process tests, with the settling times marked by solid vertical lines. The settling times for the SiL, HiL and process tests are also presented in numerical form in Table 4.1. From the table it can be seen that the settling time for the MapleSim model 1 ms HiL simulation is 0.263 seconds longer than for the real process.

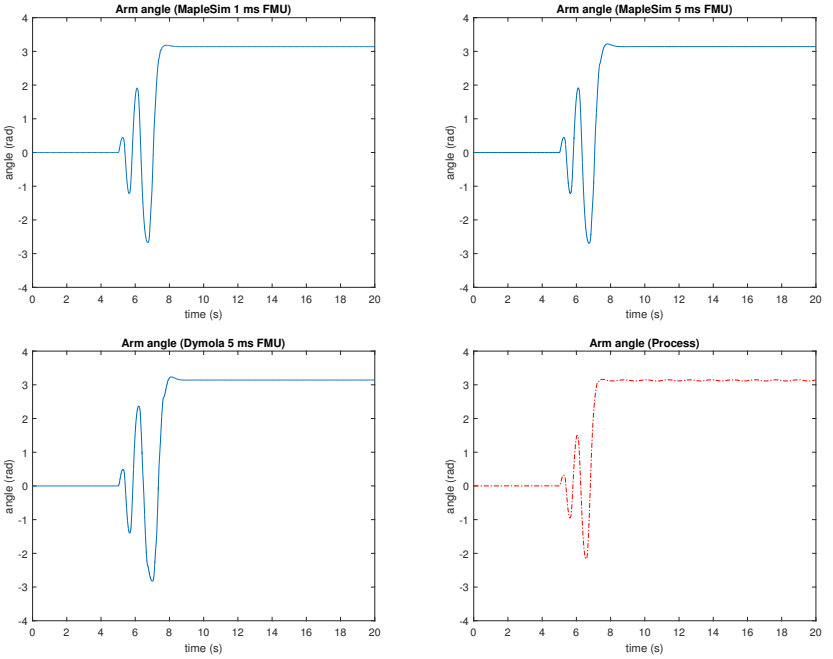

Figure 4.4 ψ angle of the arm when controlling the pendulum. Top left: 1 ms MapleSim FMU. Top right: 5 ms MapleSim FMU. Bottom left: 5 ms Dymola FMU. Bottom right: Process.

	Software	Cycle time (ms)	Start time (s)	End time (s)	t_s (s)
SiL	MapleSim	1	5.000	7.4601	2.460
	Dymola	1	5.000	7.784	2.784
HiL	MapleSim	1	5.000	7.547	2.547
	MapleSim	5	5.000	7.967	2.967
	Dymola	5	5.000	8.292	3.292
Process	-	-	5.000	7.284	2.284

Table 4.1 Settling time for the swing-up of the real pendulum and the FMUs exported with different settings.

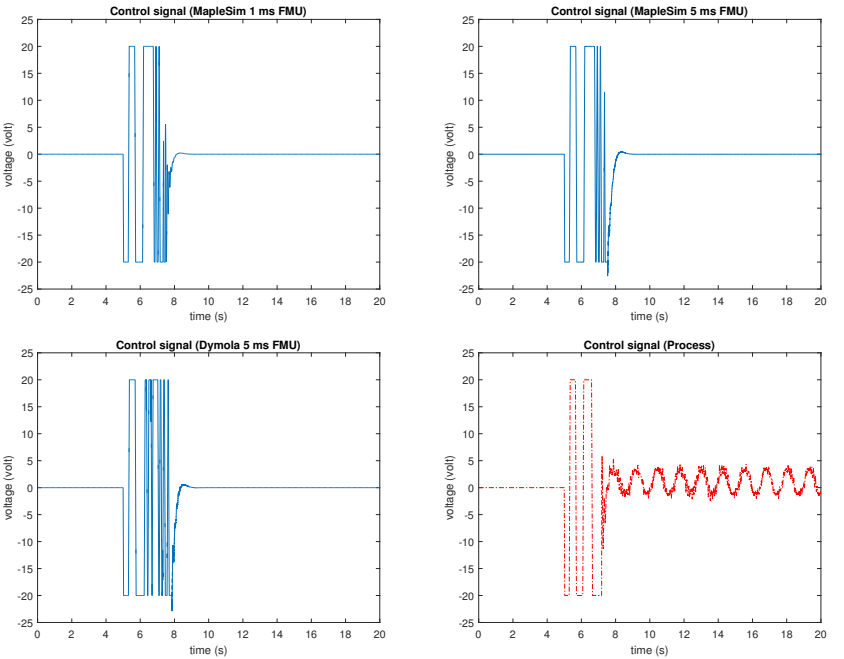

Figure 4.5 Control signal when controlling the pendulum. Top left: 1 ms MapleSim FMU. Top right: 5 ms MapleSim FMU. Bottom left: 5 ms Dymola FMU. Bottom right: Process.

4.2 FlexPicker

An FMU was successfully generated from MapleSim using the explicit Euler as solver. It was imported into Automation Studio and executed there without any control connected. The simulation ran both in MapleSim and Automation Studio until the effect of gravity (in combination with no holding torque applied) moved the robot into a configuration that induced a crash. With an applied constant holding torque to each motor the simulation ran without problems until it finished. This was however the full extent of the testing for the FlexPicker since the control from the ACOPOS simulation could not be integrated with the FMU. No results of neither a SiL nor HiL simulation were successfully obtained despite trying different methods. Instead, tested methods and proposed future ways of working to successfully do the SiL and HiL simulations will be discussed and presented in Sections 5.2 and 7.2.

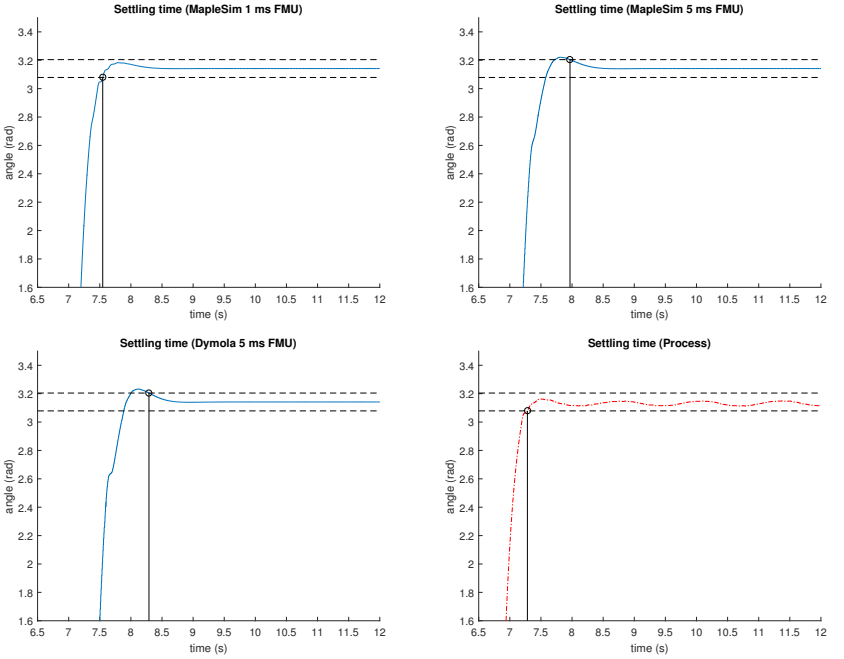

Figure 4.6 The black dashed line marks the 2% error band of the control goal $|\psi| = \pi$. Top left: 1 ms MapleSim FMU. Top right: 5 ms MapleSim FMU. Bottom left: 5 ms Dymola FMU. Bottom right: Process.

5

Discussion

5.1 Pendulum

SiL

The 24 V peak seen at around $t = 7.5$ s in the left plot of Figure 4.2 is the real limit value of the controller, even though it might appear that 20 V is. This is because the swing-up controller is saturated at 20 V, while the balancing controller is saturated at 24 V. The sudden rise to 24 V happens when the controller switches from swing-up to balancing mode. The balancing controller is a state feedback controller driven by the arm angle, arm angular velocity and wheel angular velocity with a much higher gain on the arm angle.

A notable difference between the Dymola pendulum model seen in the left plot of Figure 4.1 and the MapleSim pendulum model seen to the right is that the Dymola model swings up after 4 turns, whereas the MapleSim models swings up after 5. An explanation could be that the implicit Euler solver is better suited for the problem than the CVODE solver used for the Dymola model, since the behavior was more similar to the real process. However, CVODE contains two different formulas suited for different problems. Unfortunately the formula used when exporting from Dymola was not uncovered and as such a qualitative comparison between the solvers is not possible.

When comparing the settling times of the Dymola model and the MapleSim model taken from Table 4.1 there is a 0.324 s difference between the two, where the MapleSim model is faster. This could be an indication that the pendulum model in MapleSim is not an exact replica of the Dymola model, or that the values found in Dymola to be used in the MapleSim model were not correct.

HiL

Looking at the bottom left arm angle plot of the 5 ms Dymola FMU in Figure 4.4 one can notice a few inconsistencies in the movement around the times 6.5 seconds

and 8.5 seconds. Comparing with the control signals in Figure 4.5 it is clear that the control signal for the FMU includes unnecessarily fast switching, which seems to be the cause of the inconsistencies in the angle. The most likely explanation for this oscillatory behavior in the control signal is the slow execution speed of the FMU, compared to the controller. Since the controller is five times faster than the FMU, the FMU will appear to be standing still during the additional four executions of the controller. An unwanted effect of this is that any derivative that the controller calculates will be skewed. The bang-bang control used during the swing-up of the pendulum relies heavily on the direction of the angular velocity of the arm, estimated using the angular measurement.

This behavior can also be seen to some extent for the 5 ms MapleSim model FMU upper right plot in Figure 4.4 at around 7.5 s, which is expected since it runs on the same cycle time as the Dymola FMU. What is notable, however, is that the other inconsistency found at 6.5 s in the Dymola figure cannot be seen. The explanation is probably the same as for the SiL differences — that the MapleSim model was either not a perfect duplicate of the Dymola one, or that the Implicit Euler solver performs better than the CVODE.

When looking at the upper left plot in Figure 4.4 that represent the 1 ms MapleSim model FMU, the inconsistencies found previously in both the 5 ms Dymola model, and to some extent the 5 ms MapleSim model, are gone. This verifies the assumption that the inconsistencies can be solved by executing faster. In addition, the settling time of the 1 ms MapleSim model is much closer to that of the real process when running HiL simulations than with the other FMUs generated, which can be seen by studying the data of Table 4.1. The reason for this is probably that both the FMU and the controller is running on the same execution time, and are therefore better synced than when running with a 5 ms FMU and a 1 ms controller. Despite the 1 ms FMU performing better than the 5 ms FMUs, there are still differences between the FMU and the process. This is to be expected since the real process is much more complex than the model, which of course affects the results.

When comparing both the control signal and the arm angle of the process (lower right plot, Figures 4.6 and 4.5) with the same signals of the FMUs, the steady-state behavior differs significantly. The process exhibits an oscillating behavior around the reference point, whereas the simulations balance perfectly at it. The probable cause for this is that only viscous friction has been modeled for the arm joint, represented by the damper seen in Figure 3.3. The main friction components missing are Coulomb friction and Stribeck friction, which temporarily make the arm stick even though a non-zero torque is applied. If the model include these friction components, the same steady-state behavior would most likely be seen in the HiL simulations as well. This shows the importance of testing the real process in addition to simulation, where a lot can be learned of the final behavior, but not everything.

General

Another method of doing the HiL simulation was tried, where the PLCs communicated via analog I/O modules instead of the POWERLINK interface. The setup was more complicated since each communication line had to be wired individually, whereas the EPL allows for up to 255 channels to be configured in the software. The analog modules have no support for real-time communication, so synchronization of the two PLCs was an issue. Another disadvantage of the analog modules is that they are restricted to handling signed 16-bit variables, and the resolution of the specific module can be lower than that. The EPL channels support all standard datatypes up to 32 bits, including unsigned types. The approach using analog modules was abandoned early in the project because of these reasons.

Additional solvers, specifically explicit Euler and Runge-Kutta 2, 3 and 4, were tried but none of them managed to solve the system equations. The simulation failed in MapleSim since the output variables grew rapidly, likely because the specific solvers mapped the poles of the stable continuous-time system to unstable discrete-time system poles. An attempt was made to extract the continuous-time transfer function for the MapleSim model by exporting the equations to Maple and processing them there, with the purpose of analyzing the mapping of the poles for the solvers. This did not succeed however, since Maple was not able to construct the transfer function. The rapid growth of the solution indicates that the problem was stiff and solved using a non-stiff solver with too large sampling interval. A slightly shorter sampling interval could of course be tried but it is limited by the computation power of the PLCs.

5.2 FlexPicker

As mentioned earlier in Chapter 4, neither the SiL nor HiL simulations were successfully run during this thesis despite trying out different methods. In this section, the outline of the main problem with conducting a HiL test in AS with an FMU will be given. The methods that were tested will be discussed, and pointers to which approaches that might provide successful results in the future will be given.

In order for the ACOPOS drive to provide a control signal from the cascade control structure seen in Figure 2.6, the ACOPOS must be connected with a motor object in AS, see Figure 3.9. Writing internal variables in the ACOPOS, and especially the motor, is not something that is easily done, which is what caused problems when trying to conduct the SiL test. The main reason to why the SiL and HiL test never succeeded was because the data exchange between the FMU and the ACOPOS control system including motor proved harder to implement from a software point of view than initially thought. The motor angle feedback coming from the FMU could not be used as intended, and the ACOPOS and motor had no notion of the current

state of the FMU. Below follows small descriptions of the methods that were tried to write the feedback of the FMU to the ACOPOS, and a discussion around the potential and shortcomings of these methods.

Overwriting *PCTRL_S_ACT* approach

An idea on how to route the feedback motor angles of the FMU to the ACOPOS control structure was to simply overwrite the *PCTRL_S_ACT* parameter in the position loop of the ACOPOS controller, which contains the actual position of the motor. This parameter is normally written to by the motor encoder, but the idea was to simply overwrite this with the motor angle feedback from the FMU, thus connecting the FMU with the ACOPOS and motor.

The problem with this approach is that the *PCTRL_S_ACT* has read-only access. This access is absolute and can in no way be altered by the user in AS.

Virtual encoder approach

To affect the value of the *PCTRL_S_ACT* parameter, the source of its data (the encoder) could potentially be accessed. The control structure of the ACOPOS drives allows for a choice of encoder, specified through the ParID *PCTRL_S_ACT_PARID*, see Figure 5.1. To this parameter a ParID is written which points to the data containing the position value. The ParIDs that can be chosen are limited to a set of encoders and the set-point value data (used for sensorless control), all read-only variables as well. Just like *PCTRL_S_ACT* these read-only variables get their data from other ParIDs. The only choice where the encoder position value can be modified at some level is the virtual encoder, chosen by writing *ENCOD0_S_ACT* to *PCTRL_S_ACT_PARID*.

The virtual encoder can be set in a few different modes, chosen by setting the first 8 bits of the ParID *ENCOD0_MODE*. The relevant mode for this scenario is the network encoder mode, chosen by setting the mode bits to $0x1E$. In this mode it can be used as an incremental encoder (with or without reference track) or absolute encoder. The simplest to set up is the basic incremental encoder, which is chosen by additionally setting bit 17 of the mode ParID, i.e., setting *ENCOD0_MODE* to $0x2001E$.

The information required cyclically by the virtual encoder in this mode is provided through the ParIDs *ENCOD0_POS1_IN_PARID* and *ENCOD0_TIME_IN_PARID*. The first needs to be connected to a ParID containing the encoder position counter, 16- or 32-bit datatype. The second needs to be connected to a ParID containing the encoder timestamp, 16-bit datatype. The idea would then be to retrieve the position value from the FMU and manufacture the timestamp. The problem with this approach arises here, since these ParIDs need to be supplied with ParIDs pointing to data that can be updated cyclically, which does not seem to be possible in Au-

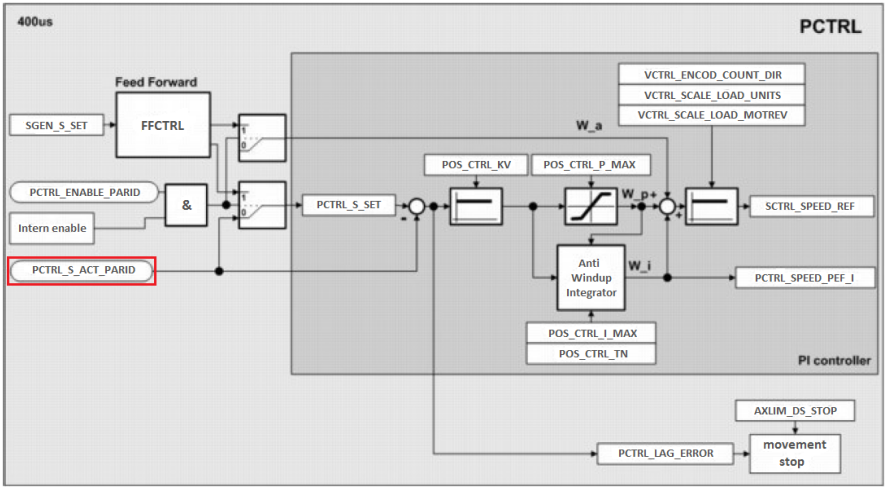

Figure 5.1 A more detailed view of the control structure seen in Figure 2.7. In addition, the ParIDs of the position controller can be seen. The *PCTRL_S_ACT_PARID* parameter can be seen to the left enclosed in a red box [B&R Automation, 2016a].

tomation Studio. There are function blocks that are referenced by ParIDs and can be given custom values (e.g., *USER_I4_VAR1*) and these can be written to cyclically using *MC_BR_CyclicWrite*.

MC_SimIf approach

An approach was tried which aims to affect the actual motor position via a library called *MC_SimIf*. The library can only be used when simulating the ACOPOS drives, and contains a function block called *MC_BR_WriteLoadSimPosition*, used for writing the position of the motor load (Figure 5.2). The main problem encountered with this approach was that the lag (control) error produced between the value written by the function block and the actual motor position instantly became large enough to forcefully deactivate the controller. This happened when a position was manually given to the function block. The reason for this seems to be that the position is compared to the motor position each cycle. This means that the position fed by the function block must follow approximately the same trajectory as the motor for the lag error to stay within acceptable bounds. In turn, this means that it is necessary to feed a dynamic load to the motor simulation to get good following between the motor positions and the corresponding positions in the FMU.

This approach was clearly the simplest and most promising. It was also confirmed by B&R that the purpose of this library is providing an interface between the ACOPOS simulation and an external simulation, which is exactly what was sought. The

main problem left to be resolved is thus to obtain a load model that can supply motor torque loads to the ACOPOS simulation.

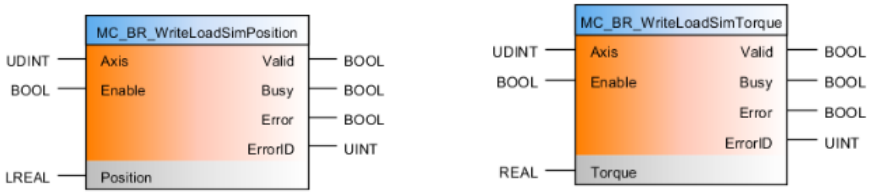

Figure 5.2 Function block for writing the actual position of the motor (left) and for writing the FlexPicker torque load to the motor (right). Only one instance of each function block can be used per axis [B&R Automation, 2016a].

The *MC_SimIf* library contains the function block *MC_BR_WriteLoadSimTorque*, used for writing the load torques acting on the motors (Figure 5.2). This provides an easy way of writing an external load torque to the motor simulation. A dynamic model from Adam Bäckström's master's thesis [Bäckström, 2014] could potentially provide the load torques, but that required a lot of manual processing of equations and is therefore outside the scope of the workflow which this thesis covers. Instead, the load torques could potentially be provided by the FMU. A way of extracting the load torques from the FMU should exist since there is functionality for this within MapleSim. However, this was never explored.

6

Conclusions

6.1 Pendulum

The SiL simulation of Sara Gunnarsson's thesis "Evaluation of FMI-based workflow for simulation and testing of industrial automation applications" [Gunnarsson, 2016] was successfully reconstructed using her Dymola pendulum model and workflow. The MapleSim duplicate of Gunnarsson's model created in this thesis differed more in settling time than expected, which is most likely explained by difference in solver performance.

The HiL tests using FMUs were successfully implemented and run with different solvers and execution times. The assumption that the inconsistencies discussed in Section 5.1 would disappear if the execution time was decreased was verified. Furthermore, the HiL tests show that the models are good representations of the real process. The differences seen between the process test and the HiL simulations are probably mainly due to lack of friction modeling and in the arm joint. The main friction contribution that lacks is Coulomb, but the Stribeck contribution should not be neglected either. In general, differences are to be expected since a model will never be an exact representation of the real world. The differences in the case of the 1 ms MapleSim model HiL simulation versus the real process are small enough to be satisfactory, but could be improved with better friction modeling.

In the end the results show that the proposed workflow is a viable method to model-based design and could potentially be an efficient approach to use in industry.

6.2 Flexpicker

After exploring several approaches to conduct SiL and HiL simulations with the FlexPicker FMU, the main hindrance was evidently the interfacing between the FMU and the servo drives. However, if this seemingly narrow issue is solved, the

proposed workflow could possibly be applied to any practical application that involves the use of the B&R ACOPOS servo drives. In conclusion, the workflow could not be applied to the FlexPicker setup, but the work on the pendulum proves the concept is certainly viable and could — with a bit more work — become a powerful and efficient tool for developing industrial automation solutions.

Although none of the approaches for the FlexPicker tried in this thesis turned out successful, the FMU and ACOPOS data exchange issue could most likely be resolved if more time would be put into the implementation. The recommended approach would be the MC_SimIf because of its simplicity and since it is designed for this exact purpose. The information collected in this thesis regarding the virtual encoder concludes that — although there may be means to use this approach — it requires in-depth knowledge about the ACOPOS controller structure and is far from trivial to set up and is thus not be a viable approach.

A torque load model of the FlexPicker would be a necessity in order to put an accurate load on the motors in simulation. Only having the motor angle feedback coming from the FMU would most likely not be good enough to produce precise results, or any results at all considering the issue where the controller switched off because of the large lag error.

7

Future work

7.1 Pendulum

An interesting investigation to showcase the versatility of FMI would be to export the Simulink controller as an FMU and run with the pendulum model FMU in Automation Studio. Lack of time and access to the FMI export option put this outside the scope of the thesis.

7.2 Flexpicker

Since the SiL or HiL simulations for the FlexPicker robot were never run, further effort could be put in to complete these tests. In addition to the current MapleSim model of the FlexPicker, the setup would have to be completed with a torque load model, cyclically supplying an external torque load to the motor load simulation. The easiest way to obtain a load model would probably be to extract it from the MapleSim model of the FlexPicker in some way. Whether this is possible and if so, how, would have to be investigated.

It would be interesting to use the versatility of the Modelica language to reverse the signals in the FlexPicker MapleSim model in order to get an FMU that could be used for inverse kinematics. With the *Transcribed Translation* and the *Conversion Block*, the desired position of the TCP could be used as FMU input, and the output would then be the motor angles of the FlexPicker.

If a torque load model was available it could also be used for feed-forward control. If the torque load on each motor is available it can be compensated for by the controller, resulting in less reliance on the integral action and faster response. An inaccurate load model could however decrease the performance, which demands a very high accuracy of the load model.

8

Appendix

8.1 Pendulum program variables

Table 8.1 provides an overview of the variables used for the SiL, HiL and process tests in the pendulum AS project, except for internal variables of the controller and FMU. The variables are separated into global variables and local variables by program.

Name	Type	Value	Description
		Global	
PendDym	Pend	(0)	FMU function block (Dymola)
PendMsim	Pendm	(0)	FMU function block (MapleSim)
period_duration_pwm	UINT	1	Duration of PWM period (μs)
ai_phiIncr	INT	0	Arm encoder analog input
ai_psiIncr	INT	0	Motor encoder analog input
phiIncr	LREAL	0.0	Arm angle (increments)
psiIncr	LREAL	0.0	Wheel angle (increments)
u	INT	0	Control signal (increments)
phi_float	REAL	0.0	Arm angle (radians)
psi_float	REAL	0.0	Wheel angle (radians)
u_float	REAL	0.0	Control signal (volts)
on	BOOL	FALSE	Regulator on command
init	BOOL	FALSE	Init FMU and regulator command
reset	BOOL	FALSE	Reset FMU and regulator command
enabled	BOOL	FALSE	FMU enabled status
		command	
counter	USINT	0	Counter for command variable reset
		encoder	
phi_old	LREAL	0.0	Previous wheel angle (radians)
phioffset	LREAL	0.0	Difference in arm angle (radians)

Table 8.1 Program variables for the pendulum Automation Studio project.

8.2 Pendulum control program

```
#define _ASMATH_
#define ASSTRING_H_
#include <bur/plctypes.h>
#ifdef _DEFAULT_INCLUDES
#include <AsDefault.h>
#endif

#include "CodeGeneration.h"
#include "rtwtypes.h"

/*  Defines */

/*  Data Types */

/*************************** GLOBAL DATA
    ************************************/
/*  Definitions */

/*  Declarations  */

/*************************** FILE SCOPE DATA
    *******************************/

/*************************** FUNCTIONS
    ***************************************/
void _INIT plant_mainINIT( void )
{
        period_duration_pwm = 1;
        on = 0;
        init = 0;
        enabled = 0;
  RTInfo_typ rt_info;
  rt_info.enable = 1;
  RTInfo(&rt_info);
  if (rt_info.cycle_time != 1000) {
    /* cycle time does not match Simulink fixed-step size */
    ST_tmp_suspend(0);
    ERR_warning(33310, 0);
  }

  /* initialize model */
  CodeGeneration_initialize(1);
}

void _CYCLIC plant_mainCYCLIC( void )
{
#ifdef HIL_TEST
        /* Convert angles to increments */
        phiIncr = (LREAL)((phi_float - psi_float) * 28000 / (2 *
            PI));
        psiIncr = (LREAL)(psi_float * 4096 / (2 * PI));
#endif
```

```
        /* call model step function if regulator is on */
        if (on)
                CodeGeneration_step(0);
        else
                u = 0;

#ifdef HIL_TEST
        /* Convert control signal to float */
        u_float = (REAL)(u * 24.0 / 32767.0);
#endif

}

void _EXIT plant_mainEXIT( void )
{
  /* terminate model */
  CodeGeneration_terminate();
}
```

8.3 Inverse Kinematics program

```
#include <bur/plctypes.h>
#include <math.h>

#ifdef _DEFAULT_INCLUDES
        #include <AsDefault.h>
#endif

void _INIT ProgramInit(void)
{
        r1 = sqrt(pow(gRobPars.UpperArmLength, 2) + pow(gRobPars.
            ElbowJointOffset, 2));
        beta = atan(gRobPars.ElbowJointOffset / gRobPars.
            UpperArmLength);
        /* Calculate sine and cosine used in coordinate system
            base change later */
        for (i = 0; i < 3; ++i) {
                axisVars[i].cosa = cos(gRobPars.
                    MotorAngularSpacing * i);
                axisVars[i].sina = sin(gRobPars.
                    MotorAngularSpacing * i);
        }
}

void _CYCLIC ProgramCyclic(void)
{
        /* Calculate inverse kinematics for all three motors */
        for (i = 0 ; i < 3 ; ++i) {
                /* Coordinate system base change for each motor
                    and add the y-offset for the TCP */
                axisVars[i].px = gDesCartPos.x * axisVars[i].cosa
                     + gDesCartPos.z * axisVars[i].sina;
```

```
                axisVars[i].py = gDesCartPos.y + gRobPars.
                    TCPOffset;
                axisVars[i].pz = -gDesCartPos.x * axisVars[i].
                    sina + gDesCartPos.z * axisVars[i].cosa;
                /* Distances to TCP attachment projected on the
                    plane parallel to motor movement */
                axisVars[i].l1 = sqrt(pow((gRobPars.MotorOffset -
                     axisVars[i].px - gRobPars.PlateCenterOffset)
                    , 2) + pow(axisVars[i].py, 2));
                axisVars[i].l2 = sqrt(pow(gRobPars.LowerArmLength
                    , 2) - pow(axisVars[i].pz, 2));
                /* Angles used for final calculation */
                axisVars[i].phi1 = acos((axisVars[i].px +
                    gRobPars.PlateCenterOffset - gRobPars.
                    MotorOffset) / axisVars[i].l1);
                axisVars[i].phi2 = acos((pow(axisVars[i].l1, 2) +
                     pow(r1, 2) - pow(axisVars[i].l2, 2))/ (2 *
                    r1 * axisVars[i].l1));
                /* Motor angular position [Units] */
                gDesAngPos[i] = (axisVars[i].phi1 - axisVars[i].
                    phi2 - beta) * 1000 * gRobPars.GearRatio /
                    brmTWOPI;
        }

        /* Write inverse kinematic result to ACOPOS */
        Axis[0].Parameter.Position = gDesAngPos[0];
        Axis[1].Parameter.Position = gDesAngPos[1];
        Axis[2].Parameter.Position = gDesAngPos[2];
}

void _EXIT ProgramExit(void)
{
        // Automatically included function not in use.
}
```

Bibliography

ABB Robotics Products AB (2000). *Product Specification IRB 340*. [Accessed 2016-06-27]. URL: https://library.e.abb.com/public/a51536e64dbeff85c12576cb00528f01/Product%5C%20specification%5C%20340%5C%20M98%5C%20BWOS3.2.pdf.

Ascher, U. M. and L. R. Petzold (1997). *Computer methods for ordinary differential equations and differential-algebraic equations*. Vol. 61. Siam, Cambridge.

Bäckström, A. (2014). *Time-Optimal Control by Iterating Forward and Backward in Time*. ISRN LUTFD2/TFRT–5944–SE. Master's Thesis. Department of Automatic Control, LTH, Lund University, Lund, Sweden.

Bernecker & Rainer Industrie-Elektronik GMBH (2016). *Automation studio*. Version 4.2.5.388. URL: http://www.br-automation.com/en/products/software/automation-studio/.

Blochwitz, T., M. Otter, J. Åkesson, M. Arnold, C. Clauss, H. Elmqvist, M. Friedrich, A. Junghanns, J. Mauss, D. Neumerkel, H. Olsson, and A. Viel (2012). "Functional mockup interface 2.0: the standard for tool independent exchange of simulation models". eng. In: *Proceedings of the 9th International Modelica Conference*. The Modelica Association, Munich, Germany, pp. 173–184. ISBN: 978-91-7519-826-2. URL: http://dx.doi.org/10.3384/ecp12076173.

B&R Automation (2009). *B&R Reaction Wheel Pendulum - Operating Manual*. [Accessed 2016-06-10].

B&R Automation (2015). *ACOPOS User's Manual*. [Accessed 2016-06-23]. URL: http://www.br-automation.com/downloads_br_productcatalogue/BRP4440000000000000332806/MAACP2-ENG_V2.01.pdf.

B&R Automation (2016a). *Automation Studio - B&R Help Explorer*. Version: 4.1.2.13334. [Accessed 2016-07-25].

B&R Automation (2016b). *Data sheet X20(c)CPx58x*. [Accessed 2016-07-15]. URL: http://www.br-automation.com/downloads_br_productcatalogue/BRP4440000000000000428404/X20CPx58x-ENG.pdf.

Dassault Systèmes (2016). *Dymola*. Version 2017. URL: http://www.3ds.com/products-services/catia/products/dymola.

Ethernet POWERLINK Standardisation Group (2013). *Ethernet POWERLINK - Communication Profile Specification*. Version 1.2.0. URL: http://www.ethernet-powerlink.org/en/downloads/technical-documents/.

FMI-Standard (2016). https://www.fmi-standard.org. [Accessed 2016-03-01].

Gunnarsson, S. (2016). *Evaluation of FMI-based workflow for simulation and testing of industrial automation applications*. ISRN LUTFD2/TFRT–6002–SE. Master's Thesis. Department of Automatic Control, LTH, Lund University, Lund, Sweden.

Hindmarsh, A. C., P. N. Brown, K. E. Grant, S. L. Lee, R. Serban, D. E. Shumaker, and C. S. Woodward (2005). *SUNDIALS: Suite of nonlinear and differential/algebraic equation solvers*. Vol. 31. 3. ACM, pp. 363–396.

Maplesoft (2014). *Getting Started with the MapleSim FMIConnector*. [Accessed 2016-07-18]. URL: https://www.maplesoft.com/documentation_center/toolboxes/MapleSimFMIConnectorGS.pdf.

Maplesoft (2016). *Maplesim*. Version 2016.1. URL: http://www.maplesoft.com/products/maplesim/.

NS (2008). *Curve intersections*. [Accessed 2016-07-19]. URL: http://www.mathworks.com/matlabcentral/fileexchange/22441-curve-intersections.

RealVNC Ltd (2015). *VNC Viewer*. Version 5.3.0. URL: https://www.realvnc.com/.

Rosquist, K. (2013). *Modelling and Control of a Parallel Kinematic Robot*. ISRN LUTFD2/TFRT–5929–SE. Master's Thesis. Department of Automatic Control, LTH, Lund University, Lund, Sweden.

Tay, T.-T., I. Mareels, and J. B. Moore (2012). *High performance control*. Springer Science & Business Media, New York.

The Modelica Association (2012). *Modelica - a unified object-oriented language for systems modeling*. Version 3.3. URL: https://www.modelica.org/documents/ModelicaSpec33.pdf.

Wal, E. van der (2009). "PLCopen". *IEEE Industrial Electronics Magazine* **3**:4, p. 25.

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

Document name
MASTER´S THESIS

Date of issue
August 2016

Document Number
ISRN LUTFD2/TFRT--6016--SE

Author(s)
Charlie Erwall
Oscar Mårtensson

Supervisor
Christian Tallner, B&R Industrial Automation
Kurt Zehetleitner, B&R Industrial Automation
Christoph Neukamp, B&R Industrial Automation
Anders Robertsson, Dept. of Automatic Control, Lund University, Sweden
Rolf Johansson, Dept. of Automatic Control, Lund University, Sweden (examiner)

Sponsoring organization

Title and subtitle
Model-based design of industrial automation solutions using FMI

Abstract
This thesis defined and investigated a general workflow based on model-based design using the Functional Mock-up Interface (FMI), involving Hardware-in-the-Loop (HiL) simulation. The thesis was a direct continuation of Sara Gunnarsson's master's thesis "Evaluation of FMI-based Workflow for Simulation and Testing of Industrial Automation Applications", where a Software-in-the-Loop (SiL) simulation of the B&R Reaction Wheel Pendulum was conducted in Automation Studio using a model imported with FMI. A HiL simulation of the pendulum was performed to complete the work done by Sara, thus showcasing the strength and possibilities of using FMI in testing. The performance of the HiL results were evaluated by comparing the settling time with the SiL test and the real process swing-up.

In addition to the pendulum work, this thesis also aimed to perform model-based tests of the ABB IRB340 FlexPicker robot, including SiL and HiL simulations. This was done in order to define a general workflow for conducting tests using FMI, and to verify the approach on a more complex process than the pendulum. A MapleSim model of the robot was exported as a Functional Mock-up Unit and imported in Automation Studio, where the testing was done.

The results of the pendulum test showed that a HiL simulation with an FMU can be performed. The HiL simulation produced a settling time of 2.55 s at best, compared to 2.46 s of the SiL simulation and 2.28 s of the process. For the FlexPicker, the SiL and HiL tests were never run due to a lack of time. Instead, a recommended approach for implementing the SiL and HiL test—along with two less promising approaches tested—were discussed and evaluated. The conclusion is that the workflow and model-based design using FMI is a promising way of conducting tests, but that there is more implementational work needed before SiL and HiL results of the FlexPicker can be successfully collected.

Keywords

Classification system and/or index terms (if any)

Supplementary bibliographical information

ISSN and key title
0280-5316

ISBN

Language
English

Number of pages
1-59

Recipient's notes

Security classification