

# Model- and Hardware-in-the-Loop Testing in a Model-Based Design Workflow

David Bergström  
Robert Göransson



LUND  
UNIVERSITY

Department of Automatic Control

MSc Thesis  
ISRN LUTFD2/TFRT--5999--SE  
ISSN 0280-5316

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

© 2016 by David Bergström and Robert Göransson. All rights reserved.  
Printed in Sweden by Tryckeriet i E-huset  
Lund 2016

# Abstract

Model-Based Design is a development method that is becoming popular to use when creating control systems. In this thesis a demonstration of the advantages of using this method is made for Combine Control Systems AB. The 3D simulation software IndustrialPhysics is used to represent a real process in form of a gantry crane. A controller for this crane is developed in Simulink and Model-in-the-Loop (MiL) testing is done together with the 3D model. C code is then generated from the controller and transferred to a PLC. A control panel with buttons is connected to the PLC and Hardware-in-the-Loop (HiL) testing is done together with the 3D model. The result of the thesis is a working HiL rig ready to be used on technical fairs to demonstrate the capabilities of the Model-Based Design method.



# Acknowledgments

We would like to thank Combine Control Systems AB for giving us a place to work at and tools to work with. Our supervisor Simon Yngve and the whole staff of Combine in Lund have been very helpful during this project. We would also like to thank our supervisor Anton Cervin at the Department of Automatic Control for the fast responses and help during our thesis. We also want to thank Georg Wünsch at Machineering GmbH & Co. KG for answering our questions. Finally, we would like to thank Madeleine Svensson at B&R Industriautomation AB for helping us choosing a PLC and their customer service for helping us with the problems that occurred.



# Contents

<b>List of Tables</b>	<b>9</b>
<b>List of Figures</b>	<b>9</b>
<b>1. Introduction</b>	<b>11</b>
1.1 Background . . . . .	11
1.2 Goals . . . . .	11
1.3 Limitations . . . . .	12
1.4 Methodology . . . . .	12
1.5 Individual contributions . . . . .	12
1.6 Disposition . . . . .	12
<b>2. Background</b>	<b>14</b>
2.1 Model-Based Design . . . . .	14
2.2 Model-in-the-Loop . . . . .	16
2.3 Software-in-the-Loop . . . . .	16
2.4 Hardware-in-the-Loop . . . . .	16
2.5 Software . . . . .	17
2.6 Programmable Logic Controllers . . . . .	18
<b>3. Design</b>	<b>19</b>
3.1 Choosing a demonstration model . . . . .	19
3.2 CAD models . . . . .	19
3.3 Choosing hardware . . . . .	20
3.4 IndustrialPhysics model . . . . .	21
3.5 Simulink model . . . . .	24
3.6 Model-in-the-Loop testing . . . . .	30
3.7 Software-in-the-Loop testing . . . . .	32
3.8 Hardware-in-the-Loop testing . . . . .	34
3.9 Testing . . . . .	35
<b>4. Results and Discussion</b>	<b>36</b>
4.1 Chosen hardware . . . . .	36
4.2 Models . . . . .	37

## *Contents*

4.3	Difference between HiL and MiL . . . . .	40
4.4	Problems in the workflow . . . . .	42
<b>5.</b>	<b>Conclusions</b>	<b>44</b>
5.1	Future work . . . . .	44
	<b>Bibliography</b>	<b>45</b>
<b>A.</b>	<b>Developer’s Manual</b>	<b>48</b>
A.1	Connecting IndustrialPhysics and Simulink . . . . .	48
A.2	Connecting IndustrialPhysics and the PLC . . . . .	50
A.3	Automatic code generation and implementation . . . . .	52

# List of Tables

3.1	Comparison between different PLC manufacturers models. . . . .	20
3.2	Variables used by the <i>control</i> model. . . . .	26
4.1	The ping times from IndustrialPhysics to Simulink and the PLC. . . .	42
A.1	How the vector of data received from IndustrialPhysics is organized. .	49

# List of Figures

2.1	An overview of the V-model. The process starts in the upper left side and follows the V shape while tests are made along the way. Adapted from [20]. . . . .	15
3.1	An overview of IndustrialPhysics. . . . .	22
3.2	An overview of the <i>mil</i> file in Simulink. . . . .	24
3.3	The Stateflow graph in the <i>control</i> model in Simulink. . . . .	28
3.4	A simple model in Simulink. . . . .	29
3.5	The MiL part of the project where the IndustrialPhysics model runs together with the Simulink model on the same computer. . . . .	30
3.6	The control panel in Simulink used for MiL testing. . . . .	31

## List of Figures

3.7	The SiL part of the project where the IndustrialPhysics model runs together with the simulated PLC in Automation Studio on the same computer. . . . .	32
3.8	The HiL part of the project where the IndustrialPhysics model runs on a computer connected with an Ethernet cable to a PLC, which is connected to the control panel. . . . .	34
4.1	The X20CP1381 PLC and power supply from B&R Automation [4] [3].	36
4.2	The buttons and joystick used in the control panel. . . . .	37
4.3	The final HiL setup. . . . .	38
4.4	The complete model in IndustrialPhysics. . . . .	38
4.5	The complete model in IndustrialPhysics. . . . .	39
4.6	Difference between HiL and MiL without a P-controller. The figure shows that Simulink (dotted line) has no oscillations while the PLC has.	40
4.7	Difference with and without a P-controller during HiL testing. The figure shows that with an P-controller makes the system more stable and smooth. . . . .	40
4.8	Difference between HiL and MiL with a P-controller. The figure shows that there are hardly any difference between HiL and MiL with an P-controller . . . . .	41
4.9	Difference with and without a P-controller during MiL testing. The figure shows that the P-controller makes the control smoother. . . . .	41
A.1	The window for setting block parameters for the TCP/IP Receive block.	49
A.2	The button for generate a HiL configuration. . . . .	51
A.3	The file contents to be moved. . . . .	51
A.4	The Online Settings menu and the Build button. . . . .	52
A.5	The Code Generation menu. . . . .	53
A.6	Location of <i>Existing File</i> in Automation Studio. . . . .	53
A.7	Choosing files from the generated C code. . . . .	54
A.8	The I/O Mapping menu. . . . .	56

# 1

# Introduction

## 1.1 Background

In the creation of complex control systems the Model-Based Design method is becoming increasingly popular to use. It is a method used for developing embedded software and is used in fields such as industrial automation, aerospace, and automotive industry. *Combine Control Systems AB* is a Swedish company that is promoting this development method [6]. They work for companies in different technical fields, often including control systems. This thesis was carried out at their Lund office. Combine wants to have a way of showing the advantages of working with the Model-Based Design method when visiting technical fairs. The idea is that visitors would be able to interact with a model created using this method to show how testing is made very easy during development.

## 1.2 Goals

The goal of this thesis is to show the advantages of using the Model-Based Design to create software. To do this a model of a physical system is created using IndustrialPhysics, a software used to simulate mechatronic models in 3D. The model chosen should be suitable for user interaction. A controller to this model is then created in Simulink. The model and the controller should then be tested together using the Model-in-the-Loop (MiL) method.

The next step is to automatically generate C code from the Simulink controller and import it to a Programmable Logic Controller (PLC). The PLC should be able to communicate with IndustrialPhysics and through the generated code be able to control the model. This will be the Hardware-in-the-Loop test and it will also include having a control panel with buttons connected to the PLC that gives users the ability to interact with the model and see changes in real time. An appropriate PLC and buttons should be chosen for this. At the end of the project there should

be a working 3D model that is controllable both with a Simulink model and with a control panel connected to a PLC.

## **1.3 Limitations**

The thesis does not include creation of the CAD models that is imported to IndustrialPhysics.

## **1.4 Methodology**

The first part of the project was to learn the software. This was done by reading manuals and using samples of other models as a guideline. The next part was to create the 3D model as well as the controller in Simulink. The modeling and the control of the system were done in parallel and took most of the time in the project. Since there was no available information about how to communicate between IndustrialPhysics and Simulink this had to be figured out by creating a TCP/IP connection and send data from IndustrialPhysics. The data was then analyzed and the structure of the communication protocol figured out.

The next part was getting to know the software and the hardware of the PLC. Learning the hardware of the PLC was done by reading the datasheet. The information of the software was, however, limited. There was a help documentation on the software, but many times help came from calling the customer service.

The goals of the project were accomplished but the synchronization between IndustrialPhysics and Simulink did not become perfect. In order to fix these changes the software of IndustrialPhysics had to be altered, which was not possible. The final models worked as they should and even had additional functionality that was not planned originally.

## **1.5 Individual contributions**

Both of the authors contributed to all parts of the project. During development of the models David spent more time developing the Simulink model and Robert spent more time developing the model in IndustrialPhysics.

## **1.6 Disposition**

### **Introduction**

This chapter covers the background, goals, limitations and the methodology used in the project.

## **Background**

Here all the background theory is presented. It covers Model-Based Design and its workflow, the different stages of the project such as MiL, HiL and SiL. Software such as IndustrialPhysics and Simulink are explained. Programmable Logic Controllers are also explained.

## **Design**

This chapter explains how the model and controller were created and why different components were chosen and designed.

## **Results and Discussion**

Here the chosen hardware is presented as well as the model designed in Industrial-Physics. The results from the MiL and HiL stage are presented and discussed.

## **Conclusion**

In this chapter conclusions are drawn from the results. It is also explained how well the Model-Based Design method worked.

# 2

## Background

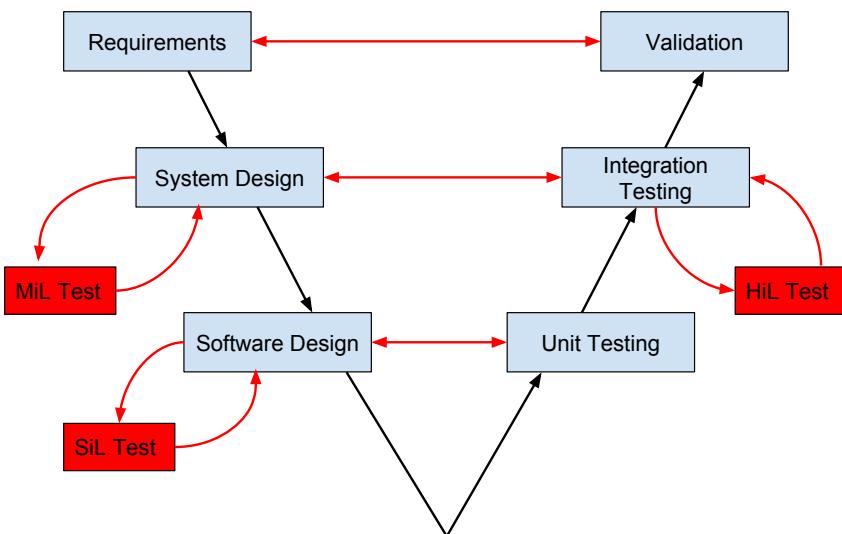
### 2.1 Model-Based Design

In traditional design methodology, there would usually be a system design engineer to grasp the overall system specifications who would hand it over to the software development engineer to implement the system in the software language that is required. The main problem with this solution is that the software developer often has a hard time interpreting the solution given from the system design developer. This means that specifications and data run the risk of being misunderstood when implemented [1].

Using Model-Based Design (MBD) developers can visually create systems using mathematical models that represent different components and the interaction between them. Simulink is a well known tool used in MBD for modeling, analyzing, and simulating a very wide variety of physical and mathematical systems [1]. It enables testing of different scenarios with the virtual model and makes it easy to find bugs in the early stages of development. The instant feedback and the interaction that one is able to have with the model make it possible to verify and validate the testing of the control system, which leads to reduced development time. This also means that software can be tested without building any hardware. This is an important factor because it is much more costly to repair a fault in the field than in the development stage [22]. Costs are also reduced by removing the need to build a physical prototype for testing. This also makes it a more environmentally friendly method. When encountering similar problems during another project, the possibility to reuse systems and code is an advantage when working with MBD. This leads to even greater time savings in the development stage.

There are several different kinds of workflow approaches. As the integration level changes during development, the individual test execution environment changes as well. A workflow approach that takes this into consideration is the **V-model**, which links early development activities to their corresponding testing activities later on. The V-model is used as a design process of a project and is illustrated

in Figure 2.1. Conventionally, the left side of the V-model represents the embedded system design phases, while the right side represents the validation and verification phases of the embedded system. The first step is the **requirement analysis**. This phase is about establishing what the ideal system has to perform, without determining how the software will be built or designed. This is done in the section where goal specifications and requirements are made. The second phase of the V-model is the **system design**. This is the phase where the developers analyze and understand the business of the proposed system by studying the user requirements documents and try to enable the requirements and specifications. This is done by testing the model continuously using MiL and coming up with solutions to the requirements and specifications.



**Figure 2.1** An overview of the V-model. The process starts in the upper left side and follows the V shape while tests are made along the way. Adapted from [20].

The third stage is where the code is written or generated. This is the **software design** stage of the V-model. To make sure that the code is working correctly it is tested using Software-in-the-Loop (SiL). On the right side of the V-model is **unit testing**, **integration testing** and **validation**. The unit testing stage is about testing the processor. This will not be a large part in this project since a PLC that is guaranteed to work by the manufacturer is used. In other cases, where the construction of the processor is part of development, Processor-in-the-Loop (PiL) testing can be made in this stage. The integration testing part is where the generated code is inte-

grated with the hardware and HiL testing is made. In this phase tests are made to see if the controller that was made in the system design stage works in the system. This links System Design together with the Integration Testing part. The last step is to validate if the results fulfills the requirements.

## **2.2 Model-in-the-Loop**

Model-in-the-Loop (MiL) testing is done in the early stages of development. During this phase the dynamics of the virtual model is captured and a controller is created based on the inputs from the virtual model. The controller and the environment are then simulated to verify the functionality in the modeling framework without any physical hardware components [24].

## **2.3 Software-in-the-Loop**

Software-in-the-Loop (SiL) testing begins with code being generated from the controller model. This code is then tested in a virtual environment, without any hardware, to test how well the software handles the simulated system. Tests are made to make sure the code works identical to the model when using different types of input conditions, functions, and mathematical algorithms. SiL testing is a good approach when simulating a real-time system that requires fast iterations, to make sure that the software is able to handle the requirements.

## **2.4 Hardware-in-the-Loop**

Once the generated code is verified to work the next step is Hardware-in-the-Loop (HiL) testing. The code is now implemented in the final hardware setup [24]. HiL simulation has to have some sort of actuators or sensors, real or simulated, that represent the hardware of the plant. By simulating a plant or a machine the tests that can be made are infinite, tests that could easily break or harm a real machine. By performing these tests developers may find errors and problems early in the developing stage instead of finding them when the control system and the plant are integrated. Not only would it take time to repair the hardware but it could also be very expensive to replace the broken parts. The safety of HiL simulation is very high compared to a test with real machines doing heavy lifting, which can be very dangerous and need high safety procedures [5].

## 2.5 Software

### IndustrialPhysics

IndustrialPhysics is a physics simulation software made by the German company Machineering GmbH & Co. KG [10, 11]. Its purpose is to realistically simulate mechatronic systems in 3D. It has support for importing CAD models, defining physical parameters and visually simulating the models in real time. Users are able to define which parts should be able to move in which direction, define collisions and write control scripts in the Structured Text PLC programming language. With its HiL interface it is also possible to control the mechatronic models using external software. This is done by sending and receiving data using TCP/IP. That means the control logic can run in a different software or a PLC, while the visual simulation takes place in IndustrialPhysics. With this software, hardware CAD models and controller software can be tested and optimized without building any expensive prototype [25]. This reduces development costs and gives the ability to try out different model versions and analyzing parameter data in an efficient way.

### Simulink

Simulink is a graphical programming environment used for creating models with block diagrams [14]. It is capable of simulating models using different solvers and automatically generating code. It has a large library of function blocks and users can also create their own customized blocks. By creating systems and subsystems in different blocks it is easy to organize a system and to get a clear overview of the different parts of the system in a way that is often not available when using text based editors. Simulink is often used in MBD. The software is created by Mathworks and can share data and functionality with Matlab. By adding certain function blocks, communication with other software is also possible, for example network communication through TCP/IP. Through different add-ons more specialized functionality can be added.

#### *Automatic code generation*

One feature that is very important to the MBD workflow is the automatic code generation. In Simulink this is possible with the Simulink Coder add-on [15]. This gives developers the ability to generate C or C++ code from their Simulink models with a click of a button. It starts by generating a *.rtw* file that contains a representation of the model. This file is then used together with a Target Language Compiler (*.tlc* file) to generate C or C++ code. By changing the *.tlc* file the code can be optimized for different target systems. The generated code is created in appropriate files that is ready to be compiled or included in another program.

In this project the Embedded Coder add-on for Simulink was used [12]. It creates code that is designed to be run on embedded processors. The code can be optimized for parameters such as execution efficiency and RAM usage.

Automatically generated code has the advantage that the human error factor is removed in the writing of the code. All the tests can be made with the model and when all features have been added and the bugs have been sorted out the code is generated. Mistakes can still be made when making the model but these are easier to detect than when writing the code manually. This can reduce errors in the final product significantly which is something important as more and more software is used in different products. Coding errors are very common. As an example Microsoft estimates that 1000 lines of code usually contains 10-20 errors and it is estimated that a car nowadays can contain 150 000 software bugs [23]. Since car manufacturers sometimes have to recall millions of cars because of these bugs, reducing code errors is something that could greatly reduce expenses.

## **2.6 Programmable Logic Controllers**

Programmable Logic Controllers (PLCs) are dedicated control computers that are often used in industrial automation. They can have both analog and digital inputs and outputs and are used to control equipment like electric motors, valves and lights [19]. They are typically used in industrial environments and are built to handle heat, humidity, unreliable power supply and vibrations.

A PLC should be able to react quickly to external events and must therefore run in real time. The PLC continuously reads the inputs, execute the code, and then writes to the outputs. All modern PLCs are capable of running software written in the programming languages defined by the IEC 61131-3 standard. These include Ladder diagrams, Function block diagrams and Structured text. Some manufacturers also include the ability to run C code. PLCs are usually programmed using an integrated software development environment running on a personal computer.

# 3

## Design

### 3.1 Choosing a demonstration model

IndustrialPhysics is mainly used in the field of industrial automation, especially in the packaging industry. Since those types of applications are designed to run automatically with little or no manual interaction they were not suitable for this project. A model was needed where the user could move something with a set of buttons to show that the model is interactive, while also demonstrating the rigid body physics simulation in the software.

It was decided that the model was going to include a gantry crane, usually used for moving shipping containers. The crane should be able to move backwards and forwards, move the crane sideways and go up and down to pick up an object. To make the model more entertaining to use, boxes and a truck were added. The idea was that boxes would be created at a random location on the ground. The user would then steer the crane to the position of the box, pick it up and then drop it in the truck. When the truck was loaded, it would drive away, empty itself and then return.

To control the model the user would have a set of buttons on a control panel. There would be buttons for turning the crane on and off, steering the crane in each direction, move it up and down and for releasing the box from the hook. There would also be a button that would switch the crane into a mode where it would automatically find boxes, pick them up and put them in the truck.

### 3.2 CAD models

The CAD models of the crane and the truck that are used in IndustrialPhysics were found online [8, 18] and edited in IronCAD [9]. The truck was manipulated in IronCAD to remove all of the small parts that were not needed in the simulated model. The imported truck model only had a flat trailer, so the walls of the trailer were created in IndustrialPhysics. The crane, however, did not have to be manipulated

in IronCAD. There were only fine adjustments made in IndustrialPhysics, such as removing the existing hook and creating a new hook.

### **3.3 Choosing hardware**

Part of this thesis was the selection of the hardware. First a PLC with our specifications had to be ordered. Then the control panel had to be designed and the appropriate buttons ordered.

#### **PLC**

The PLC to be chosen had to have certain requirements to work in the project. It had to:

- be compatible with IndustrialPhysics' communication interface,
- be able to run C code,
- have enough inputs and outputs,
- have an Ethernet connection,
- be cost efficient.

IndustrialPhysics has a communication interface that has support for several PLC manufacturers. These are: Siemens, Beckhoff, Schneider Electric, B&R Automation and Rockwell. Through discussion with people with experience with PLCs it was decided not to use Rockwell or Siemens. Schneider Electric did not have any PLCs that could run C code so they were out of the discussion. The two remaining manufacturers, B&R Automation and Beckhoff, were contacted and gave a suggestion for appropriate models.

<i>Manufacturer</i>	<b>Beckhoff</b>	<b>B&amp;R Automation</b>
<i>Model name</i>	CX5130-0120	X20CP1381
<i>Runs C code</i>	Yes	Yes
<i>Ethernet</i>	Yes	Yes
<i>Number of I/O</i>	4x4	30
<i>Performance</i>	1.75 GHz CPU, 4 GB RAM	200 MHz CPU, 128 MB RAM
<i>Software</i>	1 400 SEK/license	Free trial
<i>Price</i>	9 890 SEK	2 900 SEK

**Table 3.1** Comparison between different PLC manufacturers models.

In Table 3.1 the difference between the two suggested models from the manufacturers can be seen. Both meet the requirements, but the Beckhoff model has better performance and a higher price. Since the application that would run on the PLC did not require too much processor power, B&R's model was deemed to have sufficient performance. Since this model also had a much lower price it became the obvious choice.

## **Buttons**

The control panel had to be easy to understand and easy to use. The user should quickly understand how to interact with the model. To control the crane back- and forward and sideways it was decided to use a joystick with a button on top, the type usually used in arcade game machines. To make the crane go up and down, buttons with arrows in each directions were to be used to make it easy to understand. The on and off buttons were to be green and red as they usually are and the "auto-mode" button was going to be orange. These three buttons were all to have a light in them that could indicate which state the machine was in. The control panel would also have a big, red emergency button and a blue lamp indicating whether or not the crane was able to drop the box. It was preferred that the buttons were the same size and looked similar to each other.

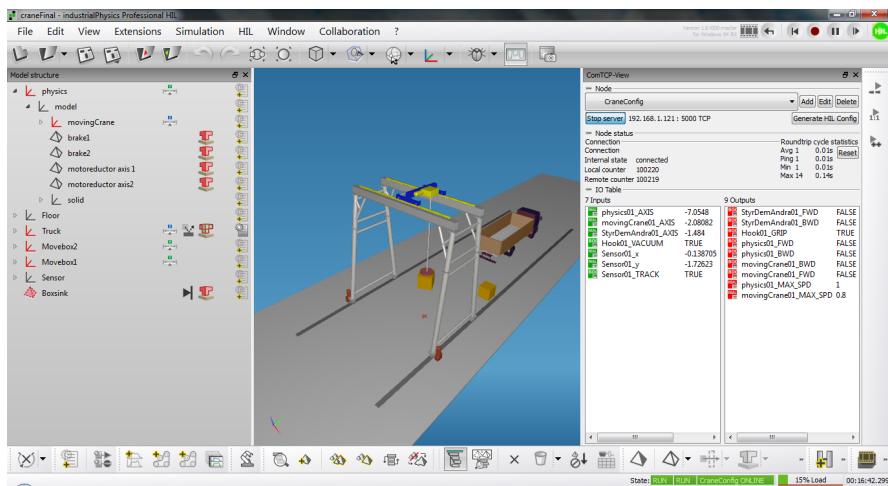
## **3.4 IndustrialPhysics model**

The graphical user interface of IndustrialPhysics can be seen in Figure 3.1. The left side of the window shows the model structure and the right side shows the ComTCP interface. In the middle, the 3D view of the model can be seen.

## **Components**

When importing a CAD model to IndustrialPhysics the hierarchical structure of the model is kept. It is possible to edit certain components or sub-components and also to add new parts to the model. When creating a new component the user can choose between a couple of shapes. This includes square, sphere, cone and cylindrical shapes. The size, mass and other physical parameters of these shapes can be chosen. The components can have different type of characteristics such as kinematic, static, immaterial and dynamic. The parts of the model that was going to move were chosen to be kinematic components. This means they are able to move forward, which is the positive direction of the selected axis and backwards, which is the negative direction. It is also possible to adjust the speed of the movement.

The floor is made of a static material that makes the parts above the floor stay still instead of falling down into infinity. IndustrialPhysics also has several invisible components that can add functionality to the model. The live statistics component counts the number of dynamic components that is in touch with it. This is used



**Figure 3.1** An overview of IndustrialPhysics.

in the truck to count the amount of boxes placed in the truck. Once enough boxes are loaded, the truck drives away. A transient sink component is also used. This is a component that swallows components that are generated from a transient source which is used to generate the boxes.

## Inputs and Outputs

The components used in the model may have several variables that control their properties. They can be viewed and modified in the I/O table in the component properties menu. The inputs and outputs can be used to send and receive data to an external program using the ComTCP feature. On the left side of the I/O table shown on the right side of Figure 3.1 are the inputs that are sent to the external program and the right side are the outputs that are sent to IndustrialPhysics from the external program. They represent coordinates, boolean signals that makes a component move in a direction or representing that a box is attached to the hook and the speed of the crane. The I/O can be used in scripts. With the help of a script one can create a movement based on an input or output. The script for the truck makes it move when there are two boxes placed in the truck and the hook has not got a box attached to it. There are also limitations of how far the truck can move and when it should go back to its original state. The script for the truck can be seen below.

```

MODEL_SCRIPT(1.5)
DEFINITIONS:
// define variables and constants
CONST boxcount := CONNECT("./Counter?TransientsCount");
VAR Truck_FWD := CONNECT(".?KinForward");

```

```

VAR Truck_BWD := CONNECT(".?KinBack");
CONST Truck_AXIS := CONNECT(".?KinAxis");
CONST VACUUM := CONNECT("../physics/model/movingCrane/New_hook
/Hook?Vacuum");
STATEMENTS:
// move the truck if it has two boxes
IF boxcount = 2 AND Truck_AXIS < 24 THEN
    IF NOT VACUUM THEN
        Truck_FWD := TRUE;
    END_IF
    ELSIF boxcount = 0 AND Truck_AXIS > 0 THEN
        Truck_FWD := FALSE;
        Truck_BWD := TRUE;
    ELSE
        Truck_FWD := FALSE;
        Truck_BWD := FALSE;
    END_IF
END;

```

## HiL Mode

To establish a connection between IndustrialPhysics and other control units, the HiL mode has to be running. This makes it possible to exchange data through the ComTCP interface. In the interface, nodes are created that contains a name, a TCP/IP port and a list with input signals and output signals. Each ComTCP node in the model creates a TCP/IP server. As soon as the server is running, a control unit is able to communicate with IndustrialPhysics through this node. The ComTCP nodes are managed in the ComTCP view as shown in the right side of Figure 3.1.

## Tracing

Every component in IndustrialPhysics has x-, y- and z-axis coordinates. When making a component movable, an input is also created where the position can be set. This makes it possible to spawn boxes at exact positions or spawn the boxes randomly within limitations. To make the crane automatically find boxes, the usage of a tracking component was needed. This static component is placed very close to the floor and is able to track the location of a dynamic component that is in contact with it. Knowing the location of the crane and the location of the box, makes it possible to pick up the boxes and placing them in the truck automatically.

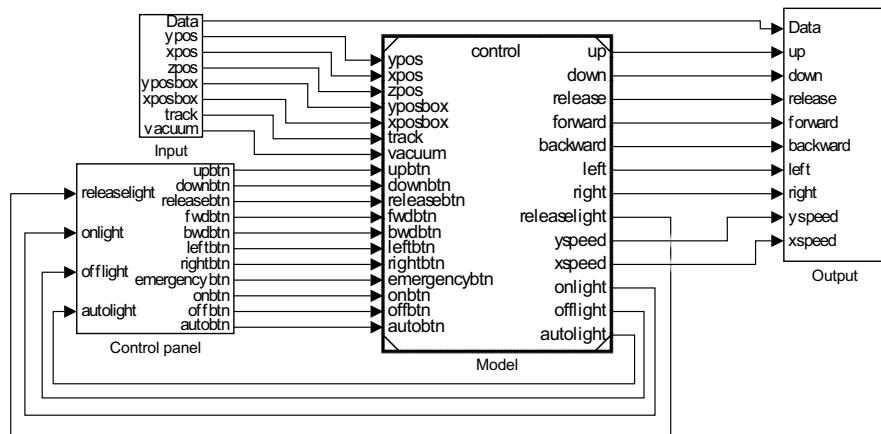
## The Rope and Hook

To make the crane look realistic a rope had to be made, that had the same characteristics as a swinging rope. It also needed the appearance of a crane lowering and raising the rope. Unfortunately there is currently no function in IndustrialPhysics to create such a component, so a compromise had to be made. The rope was divided into two parts. The upper part consists of several kinematic cylinder components, which makes it possible to lower and rise the rope. It has no dynamic properties and

is therefore a rigid construction. The lower part is made of dynamic components and is capable to swing like a real rope. At the end of the lower part there is a gripper that makes it possible to grip any other dynamic component, for example a dynamic box.

### 3.5 Simulink model

The Simulink model that was used to control the IndustrialPhysics model was divided into two files: *mil* and *control*. The *mil* file was the model used in MiL testing and can be seen in Figure 3.2. It consists of an *Input* block that through TCP/IP communication receives data from IndustrialPhysics. It also has a *Control panel* block with a graphical user interface that uses Simulink's Dashboard components to create a virtual control panel with buttons and lights. The outputs from these two blocks are connected to a *Model* block which uses Simulink's Model Reference function to include the *control* file. This makes developers able to have different parts of a model in separate files. Any edit made in a referenced file will then automatically be updated in the main file. The outputs from the *Model* block is connected to a *Output* block which sends back data to IndustrialPhysics through TCP/IP. Some outputs that control the lights also go back to the control panel.



**Figure 3.2** An overview of the *mil* file in Simulink.

The *control* file only includes the actual control and was designed to work no matter if the data is received through the MiL setup in Simulink or the HiL setup running on the PLC. It is this file that was going to be used for code generation. This file was designed to handle both the data going to and from IndustrialPhysics and

the signals going to and from the control panel. The data received from Industrial-Physics is the horizontal x- and y-position of the crane, the vertical z-position of the hook, a boolean that showed if a box was picked up and a boolean showing if the sensor was tracking the position of any box. The data that is sent back is boolean signals that determines if the crane should move in a certain direction or not and if the hook should move up or down. The horizontal speed of the crane is also sent. The data received from the control panel is boolean values that indicate if a button is pushed or not and the data sent is boolean values that control whether the lights should be turned on or off. All on/off values in the model have the boolean data type and all numeric values are in the single-precision floating-point format, that is an approximation of a real number but takes less memory than a double data type. All the input and output variables can be seen in in Table 3.2.

*Signals from IndustrialPhysics*

<b>Name</b>	<b>Type</b>	<b>Description</b>
<i>xpos</i>	single	Backwards/forwards position of the crane
<i>ypos</i>	single	Sideways position of the crane
<i>zpos</i>	single	Vertical position of the hook
<i>yposbox</i>	single	Backwards/forwards position of the latest box
<i>xposbox</i>	single	Sideways position of the latest box
<i>track</i>	boolean	Is the sensor tracking boxes?
<i>vacuum</i>	boolean	Is a box attached to the hook?

*Signals from the control panel*

<b>Name</b>	<b>Type</b>	<b>Description</b>
<i>upbtn</i>	boolean	Up button
<i>downbtn</i>	boolean	Down button
<i>releasebtn</i>	boolean	Button for releasing a box
<i>fwdbtn</i>	boolean	Joystick forward
<i>bwdbtn</i>	boolean	Joystick backward
<i>leftbtn</i>	boolean	Joystick left
<i>rightbtn</i>	boolean	Joystick right
<i>emergencybtn</i>	boolean	Emergency button
<i>onbtn</i>	boolean	Button for turning the crane on
<i>offbtn</i>	boolean	Button for turning the crane off
<i>autobtn</i>	boolean	Button for activating the automatic mode

*Signals to IndustrialPhysics*

<b>Name</b>	<b>Type</b>	<b>Description</b>
<i>up</i>	boolean	Move the hook up
<i>down</i>	boolean	Move the hook down
<i>release</i>	boolean	Release the box
<i>forward</i>	boolean	Move crane forward
<i>backward</i>	boolean	Move crane backward
<i>left</i>	boolean	Move crane to the left
<i>right</i>	boolean	Move crane to the right
<i>yspeed</i>	single	Set forward/backward speed
<i>xspeed</i>	single	Set sideways speed

*Signals to the control panel*

<b>Name</b>	<b>Type</b>	<b>Description</b>
<i>releaselight</i>	boolean	Turn on realease light
<i>onlight</i>	boolean	Turn on light in on button
<i>offlight</i>	boolean	Turn on light in off button
<i>autolight</i>	boolean	Turn on light in auto button

Table 3.2 Variables used by the *control* model.

## Discrete states

To make the model realistic it was decided to let the crane have different discrete states just like a real machine would have. The switching between these states is made possible by Simulink's Stateflow add-on that makes it easy to simulate decision logic using state machines and flow charts. Users can graphically design state machines and write Matlab or C code directly in the different states [17]. The different states of the *control* file are:

### *Off*

The first state the crane is in when the model is run is the *off* state. In this state the model will not react to any button except the *on* button. The red light in the *off* button will be on to indicate the crane is turned off.

### *Button control*

When the *on* button is pushed the *button control* state becomes active and the green light is turned on while the red light is turned off. Now the crane is controllable and can be moved in the different directions through the buttons. When the user places the hook on top of a box, the box becomes attached to the hook and the *release light* is turned on until the *release button* is pushed. Limits in x- and y-axis are defined so the crane cannot be moved too far away. There is also controller logic that makes sure the box cannot be released and the hook cannot move up and down while the crane is moving.

### *Emergency*

When the emergency button is pushed down, all movement stops and no buttons are active. Since the emergency button does not return to its original position once it is pushed, the user has to drag the button out. When that happens the *Off* state becomes active.

### *Reset*

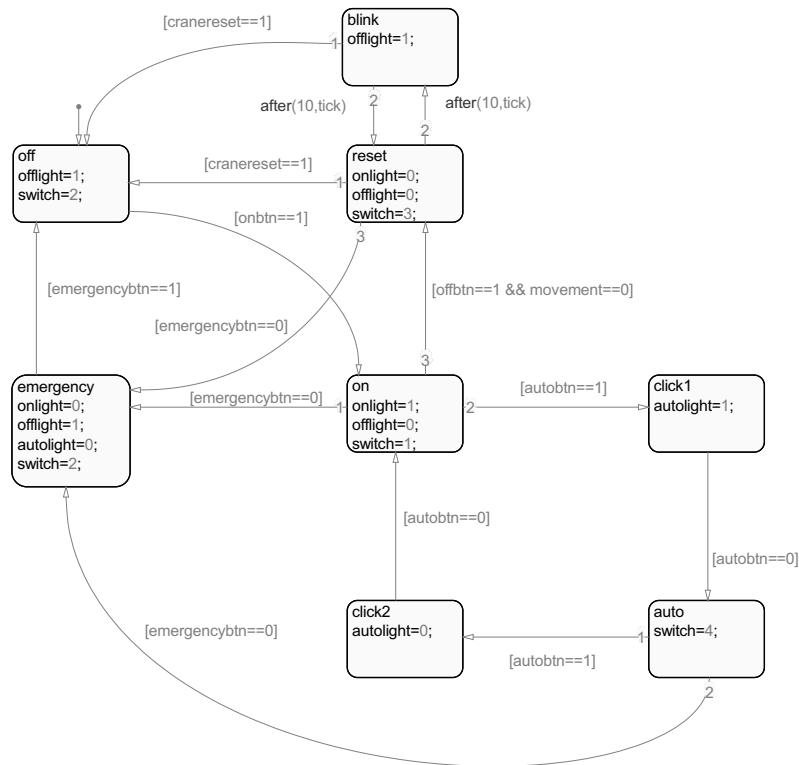
If no box is attached to the hook while the crane is turned on the user can push the *off* button to turn it off. This will also make the crane automatically drive itself to its original position. The red light will blink during the reset and then becomes constantly turned on when the crane stops moving.

### *Auto control*

If the crane is turned on and the user pushes the *auto* button, its light is turned on and the *Auto control* state becomes active. In this state the controller reads the coordinates of the latest spawned box from the sensor and moves the crane to that position. The box is picked up and the crane is driven to the position of the truck where the box is released. The coordinates for the next box is read and the process repeated until the user pushes the *auto* button again.

Which state is active is controlled by the *switch* variable in the flowchart visible in Figure 3.3. The values of *switch* represent the following states:

1. *Button control*
2. *Off/Emergency*
3. *Reset*
4. *Auto control*



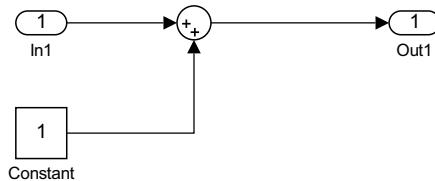
**Figure 3.3** The Stateflow graph in the *control* model in Simulink.

The Simulink model was developed using the standard building blocks available in the Simulink library. These include different math and logic operators, switches, data type converters, Matlab function blocks and TCP/IP communication blocks. During HiL testing it was discovered that the delay in the PLC made the automatic control unstable. This was fixed by letting a P-controller change the speed of the crane.

## Code generation

When generating C code from the Simulink model *control*, seven different files are generated. *Control.c* contains entry points for code implementing the model algorithm. It includes an initialization function, a step function and a terminate function that are called in the main file. The *ert\_main.c* file that is generated is not used. Instead, the step function is called from the *Cyclic.c* file used in the PLC implementation. *Control.h* declares model data structure and a public interface to the model entry points and data structures. *control\_types.h* is a macro guard, which is used to avoid the problem of double inclusion. That occurs when having one or more data type definitions to a generated header file, making sure that there will be no identifier clashes. *Control\_private.h* contains local macros and local data that are required by the model and subsystems. The *rtwtypes.h* defines data types, structures and macro guards required by the generated code [12].

As a simple example, Figure 3.4 shows a Simulink model called *add*. It has one input named *In1* and one output named *Out1*. The output is calculated as the input added with a constant value of one.



**Figure 3.4** A simple model in Simulink.

If code is generated from this model and the *add.c* file is examined, the functions can be observed. First there is the initialize function that is meant to run in the beginning of execution:

```

/* Model initialize function */
void add_initialize(void)
{
    /* Registration code */

    /* initialize error status */
    rtmSetErrorStatus(add_M, (NULL));

    /* external inputs */
    add_U.In1 = 0.0;

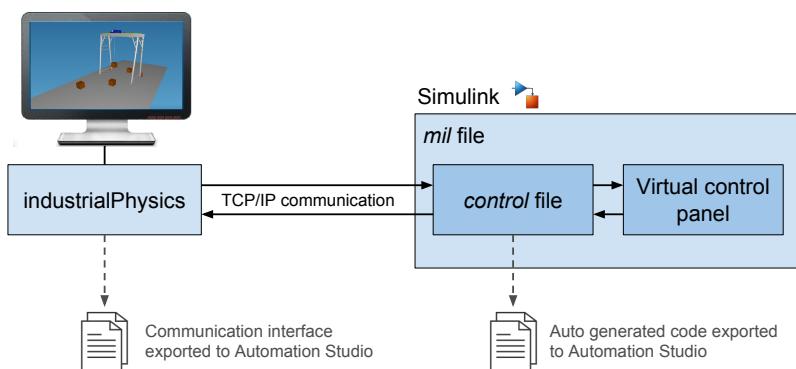
    /* external outputs */
    add_Y.Out1 = 0.0;
}
  
```

Here it can be seen that both the external inputs and outputs are set to 0.0 as the original value. Then there is the step function that runs every simulation step:

```
/* Model step function */
void add_step(void)
{
    /* Outport: '<Root>/Out1' incorporates:
     * Constant: '<Root>/Constant'
     * Inport: '<Root>/In1'
     * Sum: '<Root>/Sum'
     */
    add_Y.Out1 = add_U.In1 + add_P.Constant_Value;
}
```

Here it can be seen that the output value *Out1* equals the sum of the input value *In1* and the *Constant\_Value* of 1. Since this code does the same calculation as the Simulink model, the code generation has worked well. A termination function is also created but is empty. In this project code generation is done from the *control* file and the generated code is similar to the example above. When the code is implemented in the C program on the PLC, both the *initialize* and the *step* function are called. This is further explained in Appendix A.3.

## 3.6 Model-in-the-Loop testing

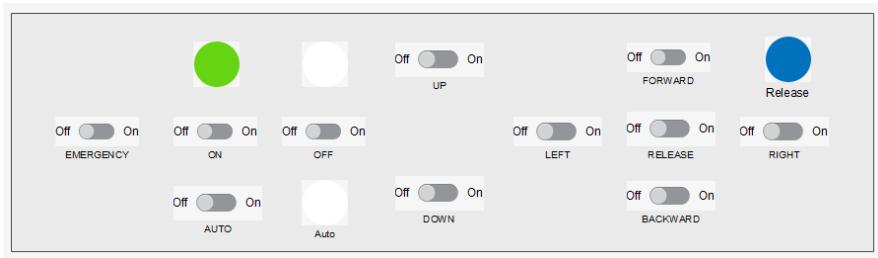


**Figure 3.5** The MiL part of the project where the IndustrialPhysics model runs together with the Simulink model on the same computer.

For MiL testing the *mil* file was designed. Its purpose is to handle communication with IndustrialPhysics and to simulate the control panel. Since there was no

available documentation on how the data from IndustrialPhysics was formatted the first step was to find out what data was sent. This was made by running the model in IndustrialPhysics with the TCP/IP server running. A connection to the server was then made from the Matlab command line and data was received. By changing the values and number of variables sent and analysing the data, the format of the sent data could soon be figured out. It was discovered that IndustrialPhysics always sends a number that counts upwards for each simulation step. It also sends the number of variables and the size of the data that is sent. Since the variables were sent in different data types and the TCP/IP blocks in Simulink are only capable of outputting one data type, data type conversion had to be made. The data of the variables was converted from the *uint32* data output to *boolean* and *single* values. Before data was sent back it is converted back to *uint32*. In Appendix A.1 the details of the connection between IndustrialPhysics and Simulink are explained.

Simulink has multiple Dashboard blocks which made it possible to create a virtual control panel. Switches were used to represent the buttons and lamps were used to represent the lights in the buttons. The user can click on these switches while the model is running and thereby control the crane from Simulink. The virtual control panel is shown in Figure 3.6.



**Figure 3.6** The control panel in Simulink used for MiL testing.

## Synchronization

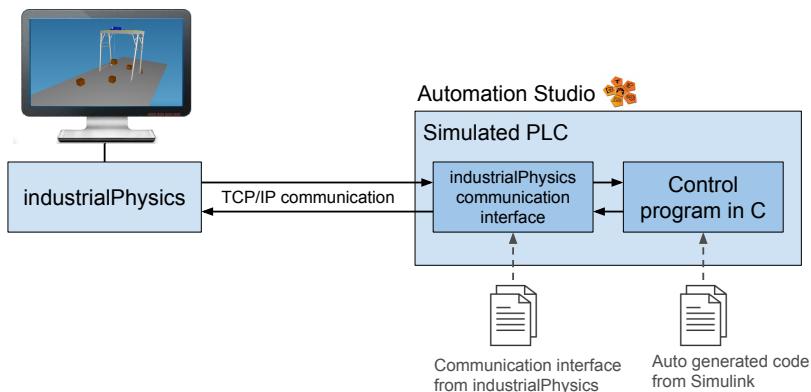
Time synchronization between the Simulink model and the IndustrialPhysics model is something that could be necessary in some projects. That means Simulink runs one time step and then waits for IndustrialPhysics to run a time step before it continues and vice versa. This makes the two software wait for each other so they are synchronized. Although perfect synchronization was not needed in the model for this project, an investigation was made to see if it could work. The time step in both programs was chosen to be 10 ms. The TCP/IP receive block in Simulink was set to blocking mode, which means the simulation is blocked while it is waiting for the requested data to be available [13]. This makes Simulink wait for new data so it

never runs faster than the IndustrialPhysics model. The problem was that there was no way to make IndustrialPhysics wait for the Simulink model. The model could be manually controlled to step forward one time step at a time but it was not possible to perform this action automatically. Machineering was contacted to solve this problem but unfortunately they did not have the time to add this feature.

## Real-time in Simulink

Another test that was made was to run the Simulink model in real time. This was made possible with Simulink Desktop Real-Time [16]. This software compiles the model to a binary file and runs it using a real-time kernel. This is something that could be useful during MiL testing and tests were made to see if it would work and if so, what difference in performance there would be compared to normal mode. Since the TCP/IP protocol includes a check if all data has been received correctly it is not suitable for real-time applications. Instead the UDP protocol had to be used. The *mil* model was updated to use an UDP block and a registry entry had to be changed to enable IndustrialPhysics to use UDP. Since the difference in performance between real-time mode and the normal Simulink mode turned out to be negligible, the normal mode with TCP/IP was used. How to use UDP in Simulink and IndustrialPhysics is explained in Appendix A.1.

## 3.7 Software-in-the-Loop testing



**Figure 3.7** The SiL part of the project where the IndustrialPhysics model runs together with the simulated PLC in Automation Studio on the same computer.

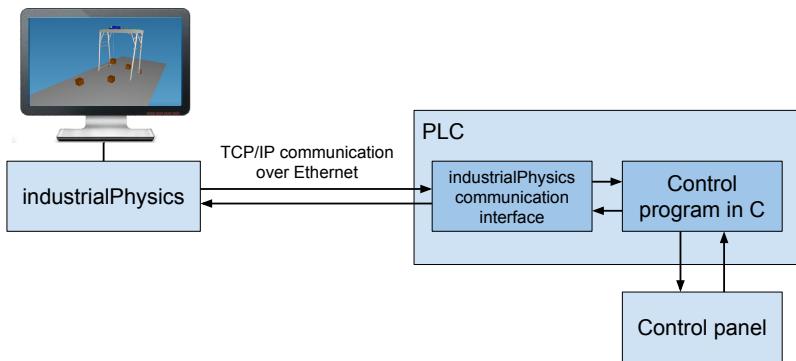
To implement the code on the PLC, Automation Studio was used. It is a software developed by B&R and is used for programming their PLCs. The software is used to design, simulate and monitor processes. Programs can be made in any of the languages in the IEC 61131-3 standard as well as ANSI C & C++, which can also be combined as required and by the modern architecture and structuring of the programming environment. All languages can access the same data types and use the same libraries and variables [2].

The first step in the SiL testing was to establish a connection between IndustrialPhysics and Automation Studio. To do this a HiL configuration was generated from the IndustrialPhysics model. This creates project files that can be imported to Automation Studio, which gives details about the number of I/O, what kind of I/O it is and a Structured Text program to establish a connection to the specific IP-address the IndustrialPhysics server has. The HiL configuration is further explained in Appendix A.2.

The next step was to import the C code from the automatic code generation of the *control* model in Simulink. First, a new C program was created in the Automation Studio project. This creates three .c files. There is *Init.c* for initialization, which is executed in the beginning, and *Cyclic.c* that runs every  $x$  ms, where  $x$  is the cycle time set by the user. Finally there is *Exit.c* that is used when the program stops. The files from the code generation were imported to the C program and the initialization function was called from the *Init.c* file. In the *Cyclic.c* file the step function is called and the inputs and outputs of the C program are connected to the outputs and inputs from the IndustrialPhysics interface.

Testing of the code was done by running the simulation mode in Automation Studio, which runs the code on simulated hardware. The cycle time in the simulation mode was set to 100 ms since it was not possible to set a smaller cycle time in Automation Studio. Since the connection between IndustrialPhysics and Automation Studio was established, the code was working and the SiL stage was complete.

### 3.8 Hardware-in-the-Loop testing



**Figure 3.8** The HiL part of the project where the IndustrialPhysics model runs on a computer connected with an Ethernet cable to a PLC, which is connected to the control panel.

Once the SiL stage was done, it was possible to export the code made in Automation Studio to the PLC knowing that the HiL configuration made from IndustrialPhysics was working. To be able to use the inputs and outputs of the PLC they had to be given a variable name which had to also be included in the *Cyclic.c* file. Here is an excerpt from that file:

```

void _CYCLIC_ProgramCyclic(void)
{
    // Get inputs to control from IndustrialPhysics
    control_U.ypos = SimInputs.physics01_AXIS;
    control_U.xpos = SimInputs.movingCrane01_AXIS;
    ...

    // Get inputs to control from the control panel
    control_U.rightbtn = rightbtn;
    control_U.leftbtn = leftbtn;
    ...

    // Step forward
    control_step();

    // Outputs from control to IndustrialPhysics
    SimOutputs.movingCrane01_FWD = control_Y.right;
    SimOutputs.movingCrane01_BWD = control_Y.left;
}

```

```

    ...
    // Outputs from control to the control panel
    releaselight = control_Y.releaselight;
    onlight = control_Y.onlight;
    ...
}

```

As can be seen in the code, the variables from IndustrialPhysics and the control panel are set as input variables to the C program. After the step function is called, the output variables from the C program are then sent back to the IndustrialPhysics interface and the control panel. This is possible because different programs on the PLC can share variables.

The cycle time of both the IndustrialPhysics interface and the C program was originally 100 ms but was set to 10 ms to reduce delay. After the code was transferred to the PLC, communication between the PLC and the computer running IndustrialPhysics was made with TCP/IP through an Ethernet cable. A test rig was made with a box where the buttons were mounted. Cables connected the buttons to the PLC inputs and the outputs of the PLC were connected to the lights.

## 3.9 Testing

Multiple tests were made to see the difference in performance between the MiL and the HiL stage. During HiL testing it was noticed that the original on-off controller for the crane was a bit unstable when using the auto control and the reset mode. This meant that it took a long time for the crane to stop. This problem had not occurred during MiL testing since the delay was smaller. To solve this problem a P-controller was implemented to control the speed of the crane. The controller slowed down the speed of the crane when the distance to the chosen position was small enough and thereby ensuring the crane did not move past the set point before receiving the position data.

To observe the difference, data was collected from IndustrialPhysics. The crane was moved as far left as possible and then the reset button was pushed, moving the crane from -5.5 m to -2.75 m in the x-axis. This test was done with and without the P-controller both using Simulink (MiL) and the PLC (HiL). The position data was then plotted and compared. The ping times from IndustrialPhysics to Simulink and the PLC were also collected during the different tests. The results of these tests are presented in the next chapter.

# 4

## Results and Discussion

The aim of the thesis was to show the advantages with Model-Based Design by controlling a virtual model in IndustrialPhysics using a PLC. The goal was to be achieved by using a controller made in Simulink, auto generate C code from that model and then implement the code on the PLC. In this chapter, we show that the goals for this thesis were all achieved successfully.

### 4.1 Chosen hardware

The PLC that was chosen was the *X20CP1381* model from B&R Automation [4] which can be seen on the left side of Figure 4.1. It has a 200 MHz CPU, 128 GB RAM, 30 digital inputs/outputs, two USB ports and one Ethernet connection. The PLC required 24 V DC so a power supply was also bought from B&R [3] and can be seen on the right side of Figure 4.1.



**Figure 4.1** The *X20CP1381* PLC and power supply from B&R Automation [4] [3].

The buttons chosen came from Schneider Electric's Harmony XB4 series [21] and is shown in Figure 4.2. They are 22 mm wide plastic buttons surrounded by chromium plated metal. When pushed, they close the circuit that is normally open. When released, a spring return it to the original position. A joystick was bought from another manufacturer [7]. It has five switches, one for each direction and one button on top of the joystick that acts like a release button. The whole setup of the hardware is shown in Figure 4.3.

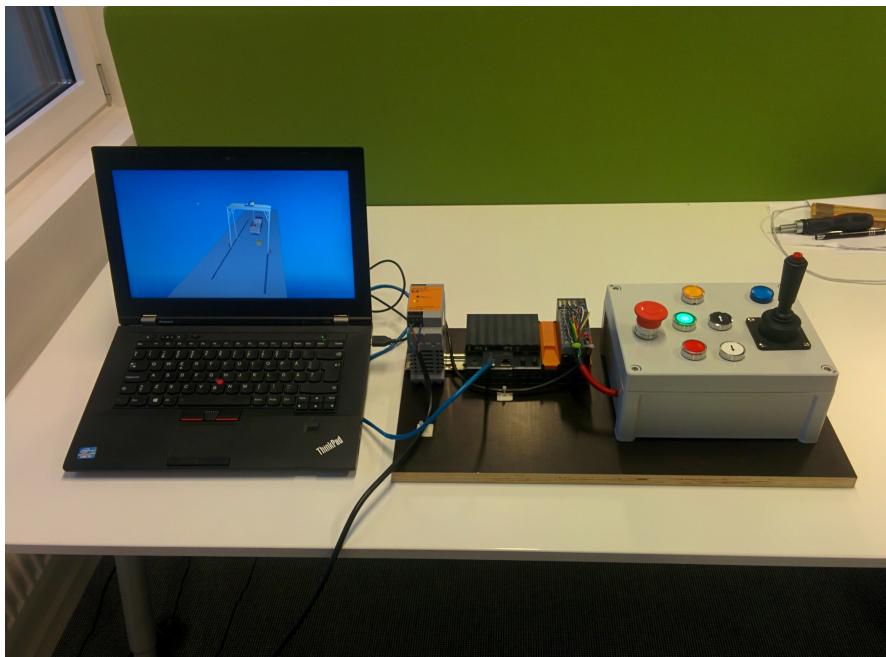


**Figure 4.2** The buttons and joystick used in the control panel.

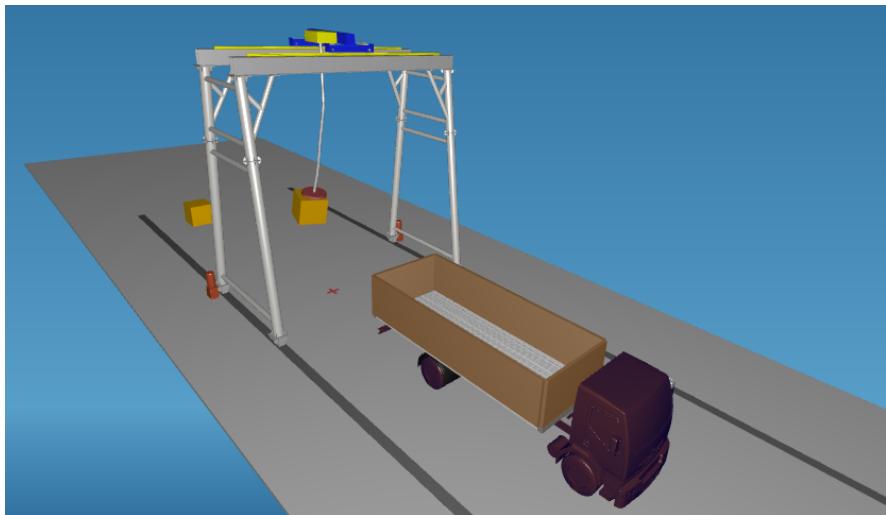
## 4.2 Models

### IndustrialPhysics

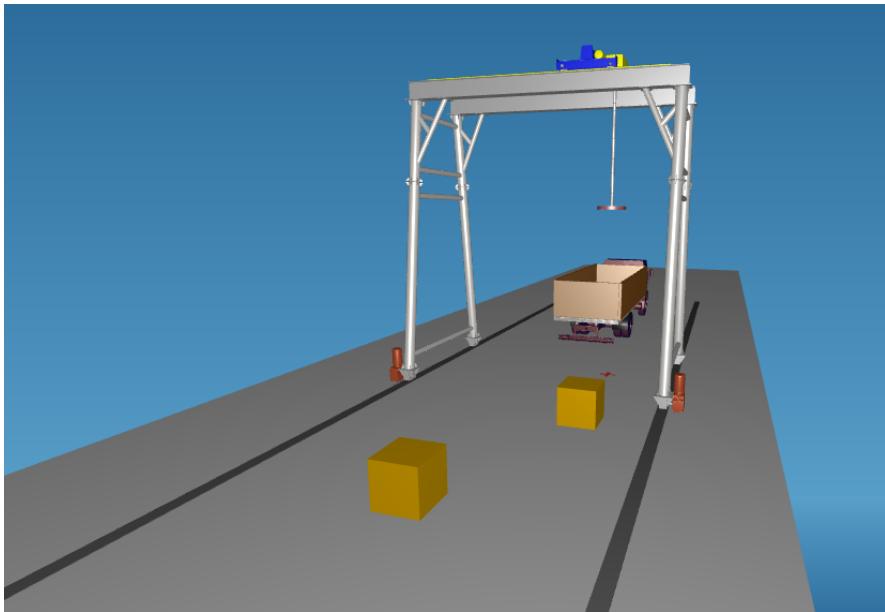
Much time was spent on making the virtual model in IndustrialPhysics e.g. creating the boxes at a random location and calibrating the damping of the system so that the rope would not swing to much. The aim with the rope was to have the rope move vertically and having the physics of an actual rope. Since this was not possible, a compromise was made by dividing the rope into two parts. One that has the physics of a rope and the other making the rope move vertically. Apart from the rope, the model looked realistic and worked as it should. In Figure 4.4 and 4.5 the final model can be seen with the crane, truck, ground and boxes.



**Figure 4.3** The final HiL setup.



**Figure 4.4** The complete model in IndustrialPhysics.



**Figure 4.5** The complete model in IndustrialPhysics.

## Simulink

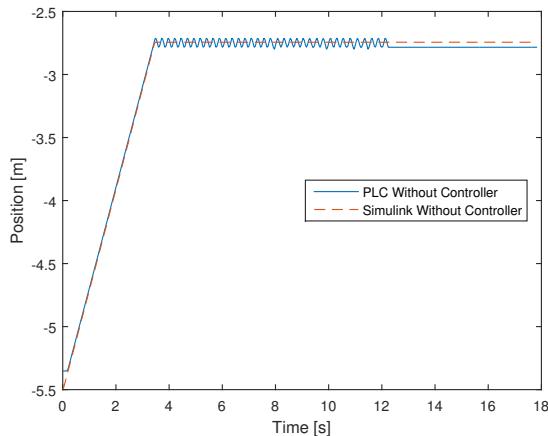
The Simulink environment fulfilled all the specifications. The available TCP/IP blocks made communication easy and since the controller could be tested against the model all the time, fixing bugs and trying new functions could be done very fast. The Stateflow functionality in Simulink was very helpful since this model needed different states which would be hard to implement otherwise. Unfortunately the synchronization between Simulink and IndustrialPhysics was not perfect since there was no way to make IndustrialPhysics wait for feedback from Simulink before it does a time step. However, in this model that was not something that had a major impact on the functionality.

As can be seen in Table 4.1 there was no difference in performance between running Simulink in normal mode and real-time mode. As a result of this, the MiL testing was done using the normal mode. It is possible that having a different Simulink model might affect the performance and thus running Simulink in real-time mode might be a better choice.

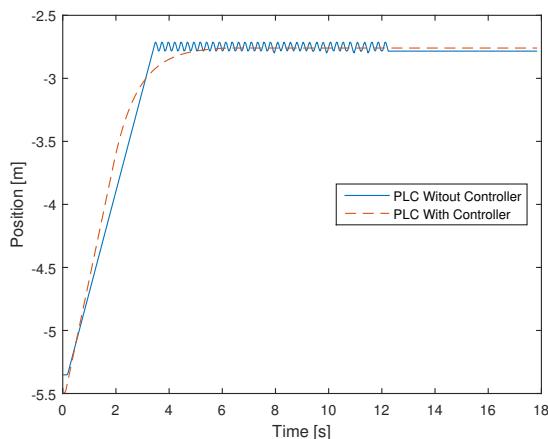
The mode that makes the crane automatically pick up the boxes was not originally planned but was added because this required delays to be low which made it more interesting to test.

### 4.3 Difference between HiL and MiL

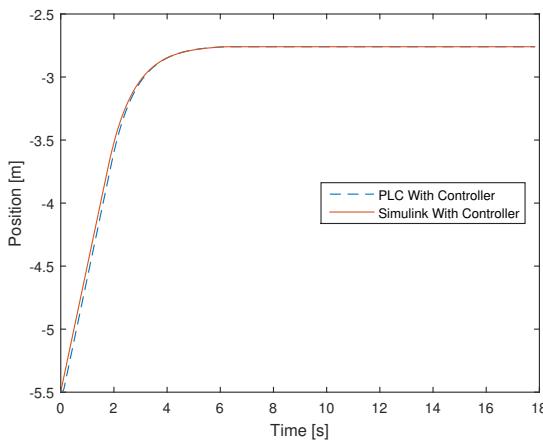
The plots in Figure 4.6, 4.7, 4.8 and 4.9 show the x-axis position of the crane when it is reset from -5.5 m to -2.75 m with and without a P-controller during MiL and HiL testing.



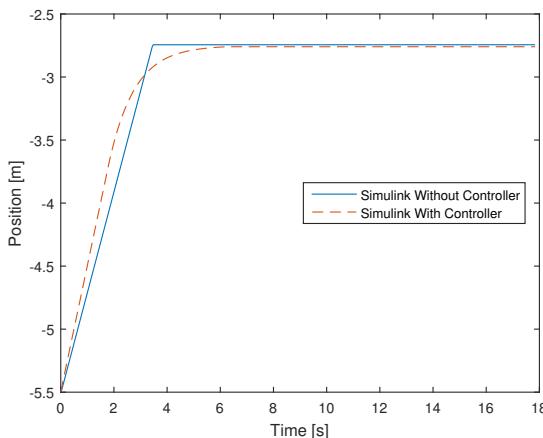
**Figure 4.6** Difference between HiL and MiL without a P-controller. The figure shows that Simulink (dotted line) has no oscillations while the PLC has.



**Figure 4.7** Difference with and without a P-controller during HiL testing. The figure shows that with an P-controller makes the system more stable and smooth.



**Figure 4.8** Difference between HiL and MiL with a P-controller. The figure shows that there are hardly any difference between HiL and MiL with an P-controller



**Figure 4.9** Difference with and without a P-controller during MiL testing. The figure shows that the P-controller makes the control smoother.

Before HiL testing, an on-off controller was used to automatically move the crane. This worked well in MiL testing but as can be seen in Figure 4.6 oscillation occurred when using the PLC. This was because the PLC had a larger delay which meant it took some time before it got the signal that the reference point had been

reached. Before reducing the cycle time in Automation Studio from 100 ms to 10 ms this oscillation was even larger. To fix this problem a P-controller was introduced and as can be seen in Figure 4.7 this worked well since there is no oscillation. Figure 4.8 shows that with the P-controller there is now almost no difference between MiL and HiL. Figure 4.9 also shows that the introduction of the controller also made the Simulink control a bit smoother.

The discovery of the difference between MiL and HiL shows the benefit of using MBD. Since the model did not include the delay discovered in HiL testing, the model had to be updated and new testing had to be done. To be able to continuously execute test and fix bugs early in the development phase are great advantages that MBD has compared to other methods.

	Simulink	Simulink Desktop Real-Time	PLC
Average ping (s)	0.01	0.01	0.02

**Table 4.1** The ping times from IndustrialPhysics to Simulink and the PLC.

In Table 4.1 the ping times from IndustrialPhysics to Simulink and the PLC can be observed. The ping time when the Simulink model is run in real time is also included. As can be seen, running Simulink in normal mode and real-time mode makes no difference for this model. Since the time step in both Simulink and IndustrialPhysics was set to 10 ms a delay of 10 ms means the Simulink model is just one step behind IndustrialPhysics on average, which is as good as it gets. The delay of the PLC varied a bit more but the average was still only 20 ms, which is very good. When developing a real machine, delay is something that is going to be there and thus the delay between the PLC and IndustrialPhysics is similar to what can be expected in the real implementation.

The crane's speed could have been faster in the x- and y-direction. It was, however, decided to keep this speed because of the dimensions of the model. If the model would have been larger the speed would have been greater. This was made to make the model as realistic as possible, which was the goal during the whole project. The velocity of the crane also contributed to the oscillations, shown in Figure 4.7 and 4.6. When the velocity was increased it was noticed that even Simulink got some oscillations when not using a P-controller because Simulink had some delay and could not manage a very high speed.

## 4.4 Problems in the workflow

Overall the workflow in this thesis worked well. There were, however, some functions in the software that could have been better. The best example of this is when a HiL config is generated from IndustrialPhysics for the PLC. First, one zip file is

created that contains an Automation Studio project with the communication interface to IndustrialPhysics. A folder is also created, containing the information about variables specific for the current model. The project files in the zip file had to manually be merged with the files in the folder. When this was imported to Automation Studio it did not work right away. Instead, several variables and much code had to be manually moved to make the program work as it should. This whole process should have worked without any manual changes to make it easier and faster. Another issue occurred when changes to the code on the PLC was uploaded. The program then stopped working. It was found out that this was probably related to the memory not being reset. That meant the PLC often had to be rebooted to reset the memory, which was very time consuming. There was also sometimes problems with both the simulated PLC and the real PLC booting in service mode, which also meant reboots had to be made.

The MiL part of the project worked well and communication between Simulink and IndustrialPhysics was fast and reliable. It would have been helpful to have documentation on how the data from IndustrialPhysics is coded, but since this was figured out after a while it was not a major problem. IndustrialPhysics has support for HiL config generation for several PLC manufacturers, but for connection with external software there is no existing interface. An API (Application Programmable Interface) would have been a great addition to IndustrialPhysics since it would make communication with other software a lot easier. Simulink and the add-ons used worked as they should and the generated C code could easily be imported into Automation Studio without any problems.

# 5

# Conclusions

This thesis has shown the advantages of using the Model-Based Design method when developing software. To develop the same kind of demonstrator with the old fashioned method would have been harder and would probably have taken longer time. Being able to test the model in all the development stages made it easy to discover bugs and to try out new functions. The strong advantage of using MBD is that errors can be discovered early in development. That made this project go smoother and it is easy to see the great advantage this gives in a real development situation with many different developers working on different part of the project. With embedded software becoming increasingly used in technology it is possible that MBD will soon be an important part of many companies development stage. The ability to try out software before any hardware is built can be an advantage in software design similar to the introduction of CAD software in mechanical hardware design.

The goals of the thesis were achieved and functions that were not planned from the beginning were even added to the models. Hopefully the models and control panel that were created will come to use and be able to meet the goal of showing how Model-Based Design can be used to increase efficiency.

## 5.1 Future work

To further test the capabilities of IndustrialPhysics it would be interesting to try a model where the time synchronization is of greater importance. In this thesis, the model and controller did not have to be perfectly synchronized and the number of variables sent was relatively small. In a complex industrial machine the synchronization could be very important and the number of signals sent could be very high. To try a model like that would be a tougher test for the software.

# Bibliography

- [1] M. Ahmadian, Z. Nazari, N. Nakhaee, and Z. Kostic. “Model Based Design and SDR”. In: *DSPenabledRadio, 2005. The 2nd IEE/EURASIP Conference on (Ref. No. 2005/11086)*. 2005. URL: <http://ieeexplore.ieee.org.ludwig.lub.lu.se/stamp/stamp.jsp?tp=&arnumber=1575352&tag=1>.
- [2] B&R Automation. *Automation Studio*. URL: <http://www.br-automation.com/en-us/products/software/automation-studio/> (visited on 01/26/2016).
- [3] B&R Automation. *B&R: OPS1020.0*. URL: <http://www.br-automation.com/en/products/power-supplies/single-phase-power-supplies/ops10200/> (visited on 01/18/2016).
- [4] B&R Automation. *B&R: X20CP1381*. URL: <http://www.br-automation.com/en/products/control-systems/x20-system/x20-cpus/x20cp1381/> (visited on 01/18/2016).
- [5] C. Kleijn. *Introduction to Hardware-in-the-Loop Simulation*. URL: <http://www.hil-simulation.com/images/stories/Documents/Introduction%20to%20Hardware-in-the-Loop%20Simulation.pdf> (visited on 01/21/2016).
- [6] Combine. *Combine website*. URL: <http://www.combine.se/> (visited on 01/22/2016).
- [7] Electrokit. *Köp Joystick arkad till rätt pris @ Electrokit*. URL: <http://www.electrokit.com/joystick-arkad.46838> (visited on 01/26/2016).
- [8] F. García. *Gantry crane - 3D CAD model - GrabCAD*. URL: <https://grabcad.com/library/gantry-crane-1> (visited on 01/18/2016).
- [9] IronCAD. *Ironcad website*. URL: <http://www.ironcad.com/> (visited on 01/25/2016).
- [10] machineering GmbH & Co. KG. *IndustrialPhysics manual*.

## Bibliography

- [11] machineering GmbH & Co. KG. *Machineering website*. URL: <http://www.machineering.de/en/> (visited on 01/18/2016).
- [12] MathWorks. *Embedded Coder User's Guide*. 2015. URL: [http://se.mathworks.com/help/releases/R2015a/pdf\\_doc/ecoder/ecoder\\_ug.pdf](http://se.mathworks.com/help/releases/R2015a/pdf_doc/ecoder/ecoder_ug.pdf) (visited on 01/21/2016).
- [13] MathWorks. *Receive data over TCP/IP from specified remote machine - Simulink - MathWorks Nordic*. URL: <http://se.mathworks.com/help/instrument/tcpipreceive.html> (visited on 02/01/2016).
- [14] MathWorks. *Simulink - Simulation and Model-Based Design - MathWorks Nordic*. URL: <http://se.mathworks.com/products/simulink/> (visited on 01/18/2016).
- [15] MathWorks. *Simulink Coder User's Guide*. 2015. URL: [http://se.mathworks.com/help/releases/R2015a/pdf\\_doc/rtw/rtw\\_ug.pdf](http://se.mathworks.com/help/releases/R2015a/pdf_doc/rtw/rtw_ug.pdf) (visited on 01/21/2016).
- [16] MathWorks. *Simulink Desktop Real-Time - MathWorks Nordic*. URL: <http://se.mathworks.com/products/simulink-desktop-real-time/> (visited on 02/01/2016).
- [17] MathWorks. *State Machine - Stateflow - Simulink - MathWorks Nordic*. URL: <http://se.mathworks.com/products/stateflow/> (visited on 01/26/2016).
- [18] E. Nunez. *Renault Midlum - 3D CAD model - GrabCAD*. URL: <https://grabcad.com/library/renault-midlum-280-185300-with-flat-bed> (visited on 01/18/2016).
- [19] G. Olsson and C. Rosen. *Industrial automation applications, structures and systems*. Department of Electrical Engineering and Automation, 2005.
- [20] SAMAA A. ABDEL SAMIE. *Automated Model in the Loop for Embedded Systems Testing*. URL: <http://www.wseas.us/e-library/conferences/2015/Dubai/CEA/CEA-60.pdf> (visited on 02/02/2016).
- [21] Schneider Electric. *Harmony XB4 - Schneider Electric*. URL: <http://www.schneider-electric.com/en/product-range/632-harmony-xb4/?parent-category-id=4800&parent-subcategory-id=4840> (visited on 01/26/2016).
- [22] V. Soccia. "Implementing a model-based design and test workflow". In: *Systems Engineering (ISSE), 2015 IEEE International Symposium on*. 2015, pp. 130–134. URL: <http://ieeexplore.ieee.org.ludwig.lub.lu.se/stamp/stamp.jsp?tp=&arnumber=7302745&tag=1> (visited on 01/18/2016).
- [23] The Economist. *Tech.View: Cars and software bugs*. 2015. URL: [http://www.economist.com/blogs/babbage/2010/05/techview\\_cars\\_and\\_software\\_bugs](http://www.economist.com/blogs/babbage/2010/05/techview_cars_and_software_bugs) (visited on 01/18/2016).

- [24] A. Vidanapathirana, S. Dewasurendra, and S. Abeyaratne. “Model in the loop testing of complex reactive systems”. In: *Industrial and Information Systems (ICIIS), 2013 8th IEEE International Conference on.* 2013, pp. 30–35. URL: <http://ieeexplore.ieee.org.ludwig.lub.lu.se/stamp/stamp.jsp?arnumber=6731950> (visited on 01/20/2016).
- [25] G. Wünsch. “Simulation for optimizing energy consumption in packaging machines”. *Economic Engineering* **2013**:6 (2013), pp. 30–31. URL: [http://www.machineering.de/fileadmin/data/Infos\\_als\\_PDF/Simulation\\_for\\_optimization\\_energy\\_consumption\\_in\\_packaging\\_machines\\_en.pdf](http://www.machineering.de/fileadmin/data/Infos_als_PDF/Simulation_for_optimization_energy_consumption_in_packaging_machines_en.pdf) (visited on 01/21/2016).

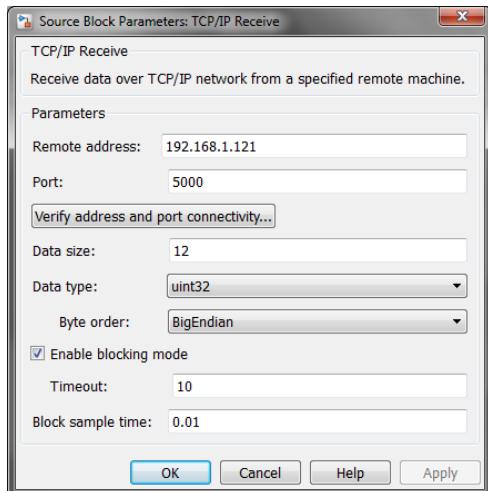
# A

# Developer's Manual

## A.1 Connecting IndustrialPhysics and Simulink

Here it will be shown how to create a connection and send data between IndustrialPhysics and Simulink. This will make it possible to control the model in IndustrialPhysics from the Simulink model.

1. Start by creating a new node in the ComTCP-View in IndustrialPhysics. Choose the computer's IP address, add the inputs and outputs that are going to be used, and activate the HiL mode in the HiL menu. Do not check the *Packed data* box.
2. In Simulink, open up the Library Browser and click on the *Instrument Control Toolbox*. Add the *TCP/IP Receive* block to your model.
3. Double click the block to open up the window shown in Figure A.1 and edit the parameters. Set the IP address and port to the same as in IndustrialPhysics. The *Data size* should be the number of inputs in the IO Table in IndustrialPhysics plus five (example: 7 inputs means the *Data size* should be 12). Set the data type to *uint32* and byte order to *BigEndian*. Blocking mode should be enabled and the sample time should be the same as the time step size in IndustrialPhysics (the default is 0.01 s).
4. The output data from this block will now be a vector that gets new values each time data is received. The first five values of the vector contains information used by IndustrialPhysics and the rest of the values contains the variables sent. How the data in this vector is organized is shown in Table A.1.



**Figure A.1** The window for setting block parameters for the TCP/IP Receive block.

Nr. in vector	Description
1	A counter that increases with one for each time step.
2 and 3	Unknown
4	Number of variables sent multiplied with four plus four. Example: 4 inputs gives $(4 \times 4) + 4 = 20$
5	Number of variables sent.
6 and forward	The variables that are sent. Boolean values are ones and zeroes and numeric values are in the single-precision floating-point format.

**Table A.1** How the vector of data received from IndustrialPhysics is organized.

5. Since the values from IndustrialPhysics have different data types and the *TCP/IP Receive* block only can output one data type, data conversion is needed before using the data. The *Data Type Conversion* block can be used for conversion between *uint32* and *boolean* values. This block is, however, not possible to use for conversion between *uint32* and *single* values. Instead, a *MATLAB Function* block has to be used with the code below.

Conversion from *uint32* to *single*:

```
|| function y = fcn(u)
|| u = typecast(uint32(u), 'single');
```

```
|| y = u;
```

Conversion from *single* to *uint32*:

```
|| function y = fcn(u)
u = typecast(u, 'uint32');
y = u(1);
```

6. When sending back data to IndustrialPhysics the *TCP/IP Send* block should be used. The data has to be converted back to *uint32* in one vector. The 4th and 5th number in the vector have to be changed depending on the number of variables that is going to be sent to IndustrialPhysics as described by Table A.1.

## Simulink Desktop Real-Time

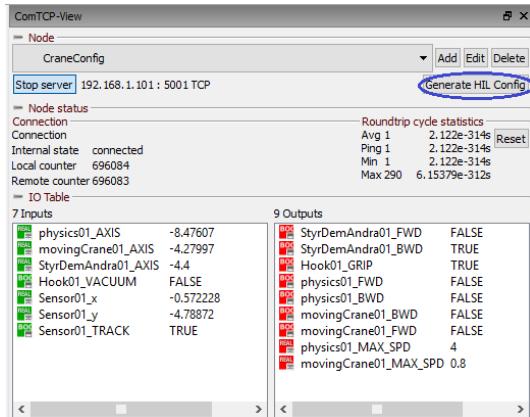
If the model is going to be run using Simulink Desktop Real-Time, UDP communication is needed instead of TCP/IP. The *Packet Input* and *Packet Output* blocks can be used for this. In the block's settings choose *Install new board / Standard Devices / UDP Protocol* and edit the address by clicking *Board setup*.

To use UDP in IndustrialPhysics it has to be manually activated in the Windows registry. Open it up and navigate to *HKEY\_CURRENT\_USER / Software / machineering / IndustrialPhysics / ComTCP* and change *UseUdp* from 0 to 1.

## A.2 Connecting IndustrialPhysics and the PLC

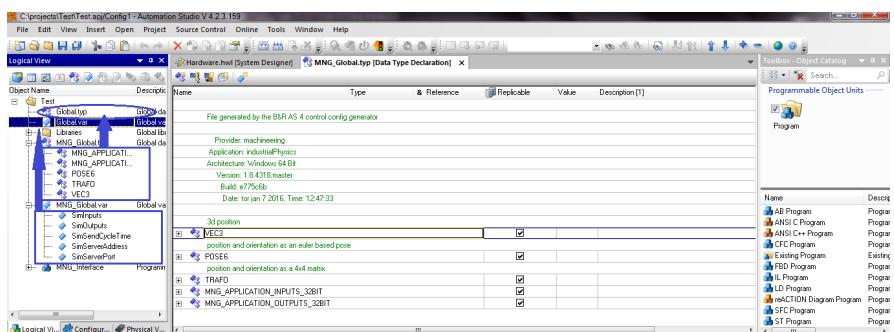
These are the steps to establish a connection between IndustrialPhysics and a B&R PLC:

1. Create a new node in the ComTCP-View in IndustrialPhysics. Choose the computer's IP address, add the inputs and outputs that are going to be used. Activate the HiL mode.
2. Click the *Generate HiL Config* button shown in the Figure A.2. This will open up a window where the user chooses the manufacturer of the PLC, and in what folder the HiL configuration shall be saved.
3. This creates a folder with the HiL configuration interface. In the folder there is a file called *01\_Static.zip* and folder called *Logical*. The *01\_Static.zip* contains two folders called *Physical* and *Logical*. The *Logical* folder inside zip file needs to be replaced by the other *Logical* folder outside zip file. This can be done by unzipping *01\_Static.zip* to a new folder, copy the *Logical* folder to the new folder and replace the files. Then create a zip file containing the *Physical* and the new *Logical* folder.



**Figure A.2** The button for generate a HiL configuration.

4. Create a new project in Automation Studio, click *import* and search for the zipped file created earlier.
5. When the file is imported, two files and one program are created in Automation Studio. For some reason, however, they do not work right away. First the contents of the two files *MNG\_Global.typ* and *MNG\_Global.var* needs to be copied respectively into the existing files *Global.typ* and *Global.var* as shown in Figure A.3. After that they can be deleted. The next step is to create a new ST Program, delete the existing files in this program and then move the contents of the *MNG\_Interface* program into the newly created ST Program. The now empty *MNG\_Interface* folder can be deleted.



**Figure A.3** The file contents to be moved.

- The HiL configuration interface is now implemented and other programs running on the PLC can access and edit the variables in *SimInputs* and *SimOutputs* in the *Global.var* file. Click *Settings...* in the *Online* menu and connect to the PLC. Click the *Build* button as shown in Figure A.4 to build the program and upload it to the PLC.

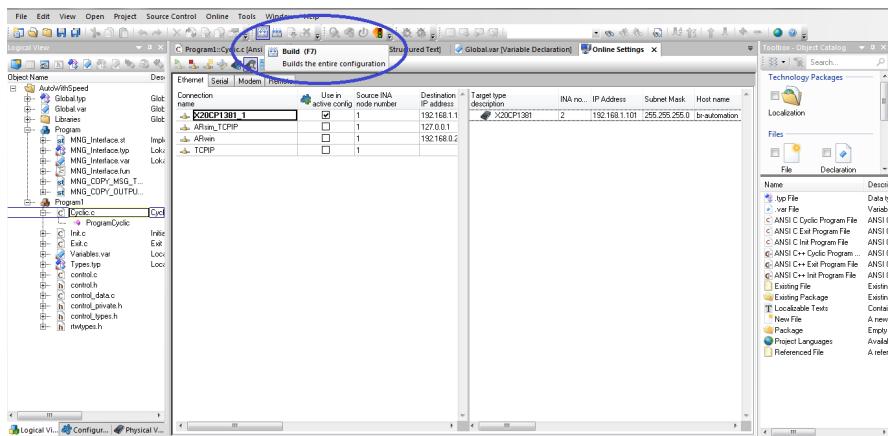


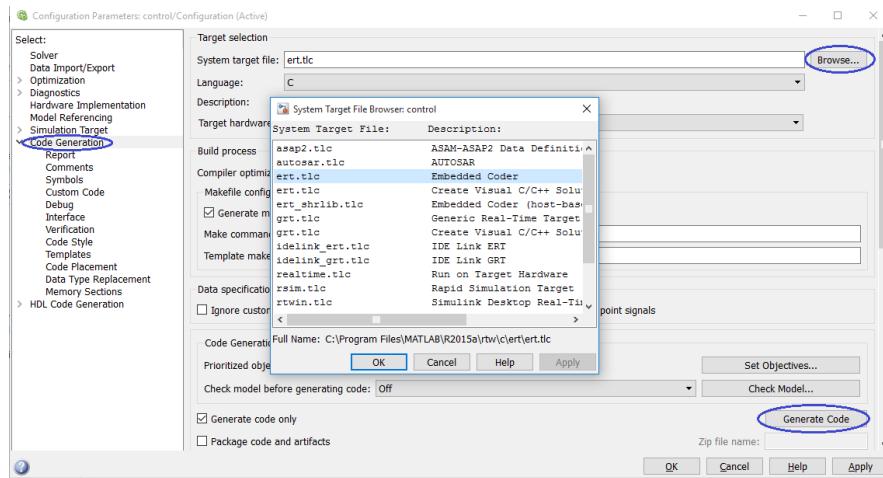
Figure A.4 The Online Settings menu and the Build button.

### A.3 Automatic code generation and implementation

These are the steps to auto generate C code from a Simulink model and importing it to an Automation Studio project. In this guide it is assumed that the Simulink model is called *control*.

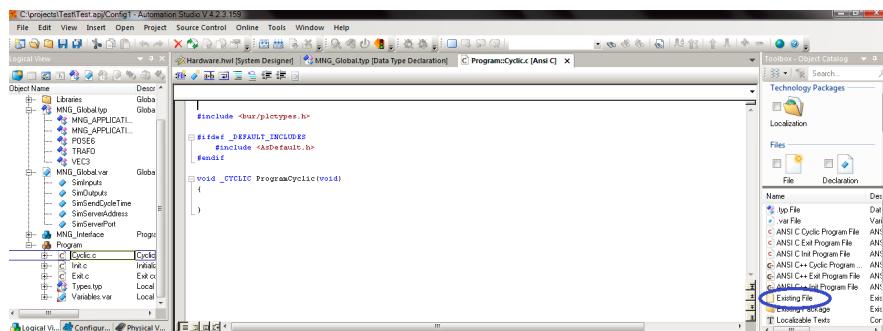
- Open up the Simulink model and click "Model Configuration Parameters" under the *Simulation* menu. First choose the *Solver* submenu and set the stop time to "inf". Choose "Fixed-step" as the solver and choose an appropriate step size.
- Click on the submenu "Code Generation" and then the browse button to choose "ert.tlc" Embedded Coder (The Embedded Coder add-on for Simulink is needed for this. If it is not available choose "grt.tlc" Generic Real-Time Target). Click the "Generate Code" button shown in Figure A.5. This creates a folder called *control\_ert\_rtw* where the files are generated.

### A.3 Automatic code generation and implementation



**Figure A.5** The Code Generation menu.

3. Before importing the generated code in Automation Studio, the user first needs to create a new C program in the project and then click on *Existing File* as shown in Figure A.6.



**Figure A.6** Location of *Existing File* in Automation Studio.

4. The user then needs to navigate to the *control\_ert\_rtw* folder and choose all the .c and .h files except *ert\_main.c* as shown in Figure A.7. Six files should be imported.

Namn	Senast ändrad	Typ	Storlek
buildInfo.mat	2016-01-18 13:41	MATLAB Data	14 kB
codeInfo.mat	2016-01-18 13:41	MATLAB Data	6 kB
compileInfo.mat	2016-01-18 13:41	MATLAB Data	1 kB
control.bat	2016-01-18 13:41	Windows-komma...	1 kB
control.c	2016-01-18 13:41	C-fil	31 kB
control.h	2016-01-18 13:41	H-fil	9 kB
control.mk	2016-01-11 10:25	MK-fil	19 kB
control.obj	2016-01-18 13:41	3D-objekt	13 kB
control_data.c	2016-01-18 13:41	C-fil	2 kB
control_data.obj	2016-01-18 13:41	3D-objekt	1 kB
control_private.h	2016-01-18 13:41	H-fil	1 kB
control_ref.rsp	2016-01-18 13:41	RSP-fil	0 kB
control_types.h	2016-01-18 13:41	H-fil	1 kB
defines.txt	2016-01-18 13:41	Textdokument	1 kB
ert_main.c	2016-01-18 13:41	C-fil	4 kB
ert_main.obj	2016-01-18 13:41	3D-objekt	3 kB
modelsources.txt	2016-01-18 13:41	Textdokument	1 kB
rtw_proj.tmw	2015-12-08 14:38	TMW-fil	1 kB
rtwtypes.h	2016-01-18 13:41	H-fil	6 kB
rtwtypeschksum.mat	2016-01-18 13:41	MATLAB Data	2 kB
setup_mssdk71.bat	2016-01-18 13:41	Windows-komma...	3 kB

Figure A.7 Choosing files from the generated C code.

- Once the generated C code files are imported into Automation Studio the user needs to connect the variables from the HiL configuration and the variables from the generated C code. The basics of this is explained in the sections 3.7 and 3.8. The code for the files used in the project can be seen below.

#### Init.c

```
#include <stddef.h>
#include <stdio.h>
#include "control.h"
#include "rtwtypes.h"

#include <bur/plctypes.h>

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

void _INIT ProgramInit(void)
{
    control_initialize();
}
```

#### Cyclic.c

```
#include <stddef.h>
#include <stdio.h>
#include "control.h"
#include "rtwtypes.h"
```

```

#include <bur/plctypes.h>

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

void _CYCLIC ProgramCyclic(void)
{
    // Get inputs to control from IndustrialPhysics
    control_U.ypos = SimInputs.physics01_AXIS;
    control_U.xpos = SimInputs.movingCrane01_AXIS;
    control_U.zpos = SimInputs.StyrDemAndra01_AXIS;
    control_U.vacuum = SimInputs.Hook01_VACUUM;
    control_U.xposbox = SimInputs.Sensor01_x;
    control_U.yposbox = SimInputs.Sensor01_y;
    control_U.track = SimInputs.Sensor01_TRACK;

    // Get inputs to control from the control panel
    control_U.upbtn = upbtn;
    control_U.downbtn = downbtn;
    control_U.rightbtn = rightbtn;
    control_U.leftbtn = leftbtn;
    control_U.fwdbtn = fwdbtn;
    control_U.bwdbtn = bwdbtn;
    control_U.releasebtn = releasebtn;
    control_U.onbtn = onbtn;
    control_U.offbtn = offbtn;
    control_U.emergencybtn = emergencybtn;
    control_U.autobtn = autobtn;

    // Step forward
    control_step();

    // Outputs from control to IndustrialPhysics
    SimOutputs.StyrDemAndra01_FWD = control_Y.up;
    SimOutputs.StyrDemAndra01_BWD = control_Y.down;
    SimOutputs.Hook01_GRIP = control_Y.release;
    SimOutputs.physics01_FWD = control_Y.forward;
    SimOutputs.physics01_BWD = control_Y.backward;
    SimOutputs.movingCrane01_FWD = control_Y.right;
    SimOutputs.movingCrane01_BWD = control_Y.left;
    SimOutputs.physics01_MAX_SPD = control_Y.yspeed;
    SimOutputs.movingCrane01_MAX_SPD = control_Y.xspeed;

    // Outputs from control to the control panel
    releaselight = control_Y.releaselight;
    onlight = control_Y.onlight;
    offlight = control_Y.offlight;
    autolight = control_Y.autolight;
}

```

6. The inputs and outputs to the control panel can be seen in the code with variable names such as *upbtn*, *downbtn* etc. These variables are connected to

different I/O ports of the PLC and can be chosen by editing the I/O mapping as seen in Figure A.8. This menu is accessed by choosing the *Physical View* and right click on X2 or X3 and choose *I/O Mapping*.

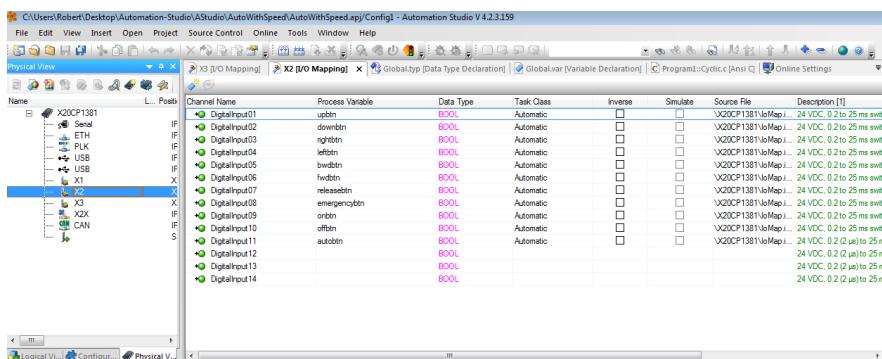


Figure A.8 The I/O Mapping menu.

<b>Lund University</b> <b>Department of Automatic Control</b> <b>Box 118</b> <b>SE-221 00 Lund Sweden</b>		<i>Document name</i> <b>MASTER'S THESIS</b>
		<i>Date of issue</i> <b>February 2016</b>
		<i>Document Number</i> <b>ISRN LUTFD2/TFRT--5999--SE</b>
<i>Author(s)</i> <b>David Bergström</b> <b>Robert Göransson</b>		<i>Supervisor</i> <b>Simon Yngve, Combine AB</b> <b>Anton Cervin, Dept. of Automatic Control, Lund University, Sweden</b> <b>Karl-Erik Årzén, Dept. of Automatic Control, Lund University, Sweden (examiner)</b>
		<i>Sponsoring organization</i>
<i>Title and subtitle</i> <b>Model- and Hardware-in-the-Loop Testing in a Model-Based Design Workflow</b>		
<i>Abstract</i>  <p>Model-Based Design is a development method that is becoming popular to use when creating control systems. In this thesis a demonstration of the advantages of using this method is made for Combine Control Systems AB. The 3D simulation software IndustrialPhysics is used to represent a real process in form of a gantry crane. A controller for this crane is developed in Simulink and Model-in-the-Loop (MiL) testing is done together with the 3D model. C code is then generated from the controller and transferred to a PLC. A control panel with buttons is connected to the PLC and Hardware-in-the-Loop (HiL) testing is done together with the 3D model. The result of the thesis is a working HiL rig ready to be used on technical fairs to demonstrate the capabilities of the Model-Based Design method.</p>		
<i>Keywords</i>		
<i>Classification system and/or index terms (if any)</i>		
<i>Supplementary bibliographical information</i>		
<i>ISSN and key title</i> <b>0280-5316</b>		<i>ISBN</i>
<i>Language</i> <b>English</b>	<i>Number of pages</i> <b>1-56</b>	<i>Recipient's notes</i>
<i>Security classification</i>		