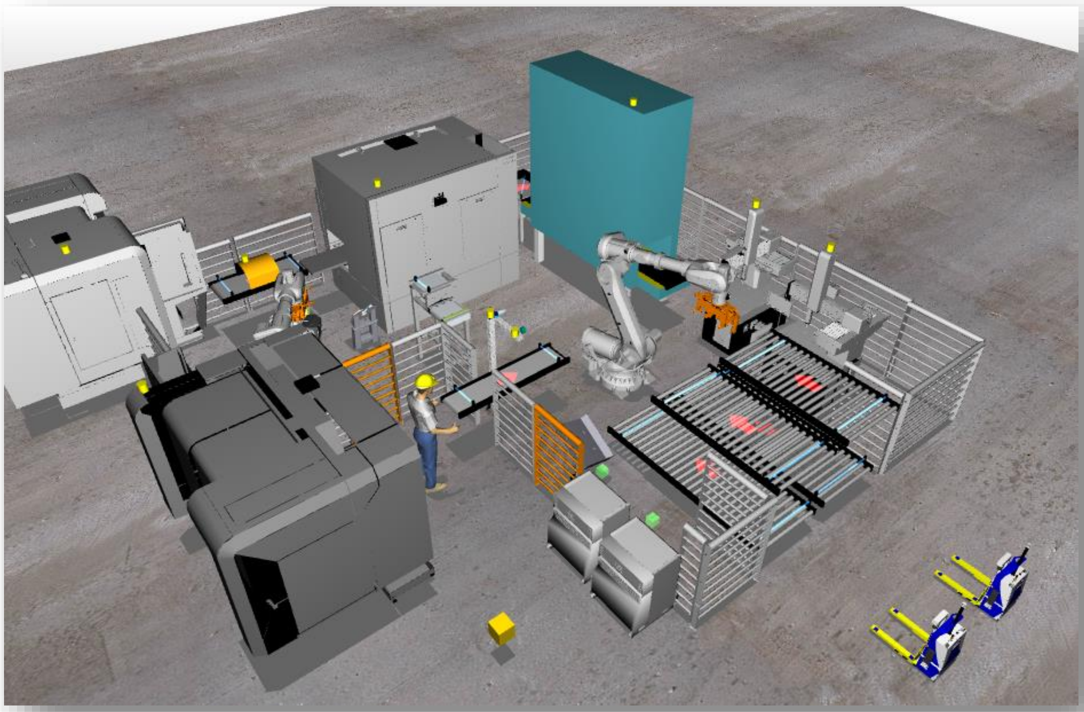
Emulation of a production cell

Developing a Virtual Commissioning model in a concurrent environment



In cooperation with ÅF, ABB, SKF, and Xcelgo.

JESPER HALMSJÖ
JONAS FÄLT

Department of Signals & Systems
Chalmers University of Technology
Gothenburg, Sweden 2016

MASTER'S THESIS EXxxx/2016

Emulation of a production cell

Developing a Virtual Commissioning model in a concurrent environment

Jesper Halmsjö
Jonas Fält

Department of Signals & Systems
Automation research group
Chalmers University of Technology
Gothenburg, Sweden 2016

Emulation of a production cell
Developing a Virtual Commissioning model in a concurrent environment.

JESPER HALMSJÖ
JONAS FÄLT

Supervisor: Fabian Fasth, ÅF
Examiner: Petter Falkman, Department of Signals & Systems

Master´s Thesis: EXXXX/2016
Department of Signals and Systems
Chalmers University of Technology
SE-412 96 Gothenburg
Sweden
Telephone: + 46 (0)31-772 1000

Cover:
The cover picture shows the Virtual Commissioning model created during this Master thesis. The view is a screen shot from Xcelgo Experior environment.

Gothenburg, Sweden 2016

Abstract

As industry demands faster and more secure ramp-up processes of new production systems, the need to verify and detect errors of the system in an early phase increases. Virtual Commissioning (VC) is a concept that allows verification in a virtual environment where engineers can verify and test their programs and ideas. The purpose of this master thesis is to create a VC model that represent a production cell in a new production system. Various methodologies and prior work have been studied and benchmarking has been conducted at state-of-the-art companies within VC. One aim has been to realize a methodology that enables engineers to build up a VC model in a concurrent environment where parallel work is encouraged. The VC model was created in two different software (RobotStudio and Experior) and a comparison has been made. The main difference between the software was the possibility to integrate robot simulation and layout in RobotStudio since the programmers in the project utilized that software. Furthermore, this master thesis has experienced complications in software communications which should have been initiated before starting the emulation modelling. Another finding of this master thesis has been that complex CAD models causes overloaded computer powers that reduces the accuracy of the VC. In addition, the importance of a centralized database and continuous communication with project members are essential to allow for concurrent environment.

Keywords: Virtual Commissioning, Programmable Logic Controller, Automation, Software-in-the-Loop, 3D-modeling, Emulation, Simulation, Experior, RobotStudio.

Acknowledgements

Firstly, we would like to thank our supervisor Fabian Fasth at ÅF that has been of great help since he has been doing a virtual commissioning project before in Experior. He advised us throughout the project regarding pitfalls, programming strategies and also provided us with components and functionalities that he earlier has developed.

Pär Ström and Andreas Buhlin at ÅF have also been of great help to build up a network of people and experts within different kind of fields. In addition, Andreas has been well engaged in the SKF project and provided us with updated information and changes of the concept.

We would also like to thank Aksel Jørgensen and Michael Pedersen at Xcelgo for supporting us in Experior and providing us with new software versions of plugins when it was needed. They rapidly answered our questions and were very helpful.

Björn Magnusson, Anders Spaak and Magnus Seger at ABB have been very helpful regarding supervision in the software RobotStudio. They have been providing us with plugins and applications to enable PLC communications and answered specific questions regarding technical issues in RobotStudio.

Lastly, we would like to thank our supervisor at Chalmers Petter Falkman who has been of great help regarding the project structure and given us input to what to focus on from an academic perspective.

Jesper Halmsjö [illegible]

Jesper Halmsjö Jonas Fält

Contents

1 Introduction ... 1
1.1 Background ... 1
1.2 Purpose ... 1
1.3 Brief description of the production cell ... 2
1.4 Research Questions ... 3
1.5 Delimitations ... 3
2 Literature study ... 4
2.1 Virtual commissioning ... 4
2.2 Approaches of commissioning ... 4
2.3 Simulation vs Emulation ... 5
2.4 Verification vs Validation ... 7
2.5 Digital Factory ... 8
2.6 Physics engine approach ... 11
2.7 Virtual Reality ... 12
2.8 PLC & communication in industrial automation ... 12
2.8.1 PLC theory ... 12
2.8.2 Communication protocol theory ... 13
2.8.3 OPC ... 13
2.9 Prior work ... 15
3 Benchmarking ... 18
3.1 ÅF ... 18
3.2 KUKA ... 19
3.3 Virtual Manufacturing ... 20
4 Virtual Commissioning Framework ... 22
4.1 Characterize the production system ... 22
4.2 Software establishment ... 23
4.3 Emulation Model Building ... 23
4.3.1 Low level modeling ... 24
4.3.2 High level modeling ... 24
4.3.3 Verification and Validation ... 24
4.4 Testing of the emulation model ... 25
4.5 Virtual commissioning ... 25
4.6 Evaluation ... 26

5 Building the SKF Virtual Commissioning model 27
5.1 Characterize the production system 27
5.2 Software establishment 27
5.2.1 Modelling architecture 27
5.2.2 Virtual commissioning architecture 28
5.3 Emulation Model Building (Experior) 29
5.3.1 Low level modelling 29
5.3.2 High level modeling 32
5.3.3 Verification and Validation 34
5.4 Emulation Model Building (RobotStudio) 35
5.4.1 Low level modelling 35
5.4.2 High level modelling 39
5.4.3 Verification and Validation 40
5.5 Testing emulation model 40
5.5.1 Model functionality 41
5.5.2 Test programs 43
6 Discussion 46
6.1 Methodology review 46
6.2 Comparison between VC in Experior and RobotStudio 48
6.3 Increasing the efficiency of VC modelling 50
7 Conclusion 53
8 Future work 55
9 References 57

Appendix A – Example C# Code in Experior for AGVs 1
Appendix B – Excel macro that auto generate C# code for I/O signals 13
Appendix C – Experior problems and suggestions 20
Appendix D – Problems and suggestions for improvements in Robotstudio 21

List of Figures

Figure 1: A simplified layout of the production cell. Green squares represents processes, blue squares represents conveyors and orange square other equipment. 2
Figure 2: Methodology for simulation described by Rabe et al. (2008). 7
Figure 3: Effects of store and utilize standardized components (Kiefer et al.,2008) 10
Figure 4: Procedure for working with Virtual Commissioning in a concurrent environment (Ko et al.,2013) 10
Figure 5: Methodology of plant modelling with consideration of physical behavior developed by Strahilov et al. (2012). 11
Figure 6: Basic arrangements of a PLC (Reddy, 2015) 13
Figure 7: The architecture of OPC communication (Reddy, Y.J 2015). 14
Figure 8: Dzinic and Yao (2013) process of performing a VC. 15
Figure 9: Dzinic and Yao (2013) description of CAD model's work low. 16
Figure 10 - The VC architecture used at ÅF 19
Figure 11: Software architecture used by KUKA 20
Figure 12: Process for creating a Virtual commissioning model. 22
Figure 13: Process for building the emulation model 23
Figure 14: Architecture for converting and importing files into RobotStudio and Experior. 27
Figure 15: VC Architecture when using Experior. 28
Figure 16: VC Architecture when using RobotStudio. 28
Figure 17: An extraction of the Visual Studio project for creating an assembly and add geometrical, functional and electrical properties. 30
Figure 18: Auto generated coding. 31
Figure 19: Gripper configurations. 32
Figure 20: 2D drawing imported into Experior's environment. 33
Figure 21: I/O mapping using Excel sheet. 33
Figure 22: Low level verification with signals, logics, and animation. 34
Figure 23: Components are adjusted with respect of the robot position targets. 35
Figure 24: Interface of Smart Component creation. 36
Figure 25: Connecting I/O signals, relations and child components in RobotStudio. 36
Figure 26: Connecting I/O signals, relations and child components with block diagrams in RobotStudio. 37
Figure 27: Separating CAD-geometry. 38
Figure 28: Configuration of joint movements. 38
Figure 29: The figure sequence shows how a gripping operation is performed. 39
Figure 30: Part of the logic where a part is attached to the gripper. 39
Figure 31: PLC-logic for the gipping tool. 40
Figure 32: Virtual plant in RobotStudio. 41
Figure 33: Virtual plant in Experior. 41
Figure 34: A machine that is currently blocked. 42
Figure 35: HMI for testing the model's functionalities show different states. 42
Figure 36: Effects of delay in the data exchange between the PLC and RobotStudio 44
Figure 37: Extraction of the PLC program used for testing. 44
Figure 38: Extraction of the robot program used for testing. 45

List of Tables

Table 1: Differences between simulation and emulation 6
Table 2: Similarities between simulation and emulation.. 6
Table 3: Description of the three categories of VR systems (Mujber at al., 2004). 12

Abbreviations

3D	3 Dimensional
AGV	Automated Guided Vehicle
C#	C-Sharp
CAD	Computer Aided Design
COLLADA	COLLAborative Design Activity
CPU	Central Processing Unit
FAT	Factory Acceptance Test
HIL	Hardware-in-the-Loop
HMI	Human Machine Interface
I/O	Input/Output
OPC	Object Linking and Embedding for Process Control
PLC	Programmable Logic Controller
RIL	Reality-in-the-Loop
SIL	Software-in-the-Loop
SKF	Svenska KullagerFabriken
SoftPLC	Software PLC
TCP	Tool Center Point
TCP/IP	Transmission Control Protocol/ Internet Protocol
VC	Virtual Commissioning
VR	Virtual Reality
ÅF	Ångpanneföreningen

1 Introduction

This chapter provides an introduction to the project where background, purpose, research questions and delimitations are stated.

1.1 Background

SKF, a bearing manufacturer located in Gothenburg, will invest in a complete new production line including a number of robot stations. The stations include robots and different equipment such as operating machines, measuring machines and conveyors. Some cells in the line will, except for machine processing, also perform some assembly of the products. Automated Guided Vehicles (AGV) will be used for transporting pallets with products to different stations in the production line.

A challenge within the industry today is to verify and validate control programs for Programmable Logic Controllers (PLC), robot and other control programs for a production system before commissioning. In the commissioning phase, several problems might occur that can lead to longer installation times and may in turn result in longer production stops and lead to economic losses for the company. A delay of the production start may also result in economical fines for the company responsible for the installation of the production system.

Virtual Commissioning (VC) is a concept where a representation of the real production system is created in a virtual model. By having this model, control programs could be verified in a virtual representation of the production system before the commissioning. The early verification enables engineers to identify problems in the virtual environment that otherwise might been identified when commissioning the solution.

The consultant company Ångpanneföreningen (ÅF), that are responsible for the design of a part of the new production system, wants to investigate the possibilities within VC. ÅF has therefore decided to create a VC model that should represent one cell in the new production system. The PLC and robot programs will be created in parallel with the building of the VC model which results in various challenges and are a part of this thesis work.

1.2 Purpose

The purpose of this master's thesis is to create a VC model that represent one of the cells in the new production line at SKF. The model can be used for verification and validation of control programs (e.g. PLC and robot programs) before commissioning. This project will also contribute to increased knowledge in VC theory and broaden the library of mechatronic components for future projects within the ÅF Cooperation. Furthermore the master thesis will contribute with a work method applicable for VC projects in a concurrent work environment. The model will be created in two different software which also contribute to a comparison between the two software.

1.3 Brief description of the production cell

The cell that will be modeled will follow the simplified layout in Figure 1. Below follows a brief description of the production flow of the cell. Due to secrecy, no machine specific details are mentioned.

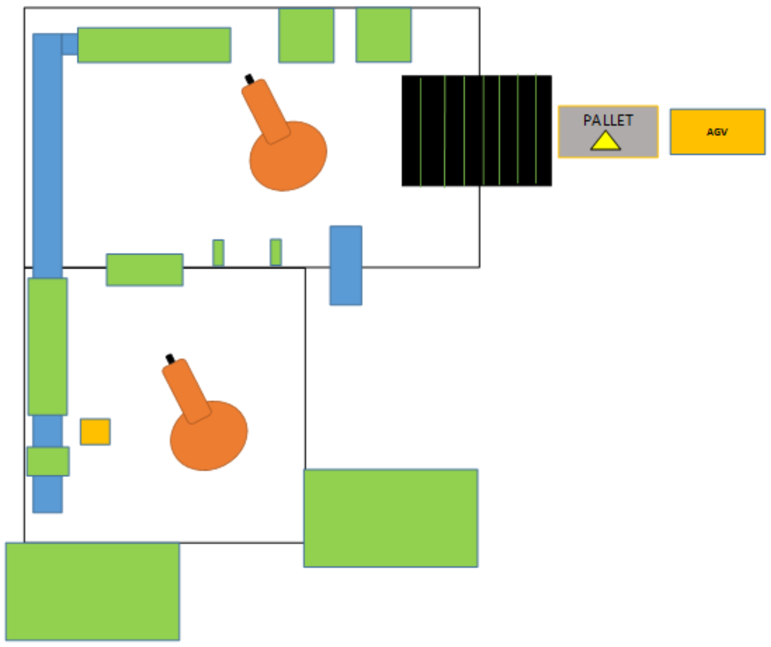


Figure 1: A simplified layout of the production cell. Green squares represents processes, blue squares represents conveyors and orange square other equipment.

The production station consists of two robots that will serve a number of different processes with bearings. The material is delivered to and from the cell by an AGV-System. Each pallet will be equipped with a Radio Frequency Identification (RFID) tag that the RFID-System will be able to read and write necessary production data. The first robot, located in the upper part of the layout in Figure 1 will serve a washing process, three different measuring processes, a printer and a scanner. This robot will also be responsible for the in- and outgoing material for the cell. Interleaving papers are located between each layer of the bearings on the pallet. This papers need to be taken of incoming pallets and be placed on outgoing pallets and is also done by the first robot. To do this the robot will be equipped with both a vacuum and a servo gripper tool.

The second robot in the lower part of the cell will serve four other processes. This part of the cell will also be equipped with a turning station where the rings can be turned before operations if necessary. One of the processes will be reachable by both robots and enables the bearings to be exchanges between the two robots. The cell will also be equipped with an inspection station where the operators can request to inspection a bearing that is processed in the cell. The first robot will then put a bearing at a conveyor that will take it outside the cell.

1.4 Research Questions

In order to narrow the perspective and increase the focus of this project, research questions have been developed. These will be answered in the conclusions and discussed in the discussion part.

- What work methods and structure should be used for creating a VC model in a concurrent environment?

- What are the main difference when modelling a VC in Experior and RobotStudio?

- How to increase the efficiency and speed of VC modelling to make it more beneficial?

1.5 Delimitations

This chapter provides a summary of this project’s delimitations which are described in bullet points below:

- The timeframe of this project will be 20 weeks and the project team consists of two students.
- The VC will only be made to handle one product model with a defined size. The real system will handle multiple product models with different sizes.
- The AGV-system will be simplified. State signals of the AGV will be connected to the master-PLC of the cell for enable communication. When load or unloading of the cell is done, a simplified motion of the loading is simulated.
- The material on the incoming pallet will follow a specific pattern with predefined locations.
- The graphical appearance of the equipment will depend on the availability of Computer Aided Design (CAD) models from the vendors. If no CAD-models exists a simplified representation of the equipment will be modelled.
- The Input/Outputs (I/O) signals for equipment in the cell will depend on the availability of the actual equipment I/O signals. If no technical specification of the equipment exists, assumptions will be made with consolation from supervisors.
- The safety system will be modeled with the respect to available time in the project. The priority will be to model the sensor-curtains for the fence doors.
- Some motions of the product and equipment will be limited, such as spinning motion inside the machines.
- The HMI function will be limited to only show the states for the cell and for individual and for some chosen processes.
- This project does not include writing a complete PLC program to validate the model. Instead smaller test PLC-programs or a will be used. If time allows, a simplified program that will take the ring through all processes will be done.
- The model will be tested using a Codesys softPLC (Codesys WinSys V.3). No hardware PLC will be used.
- No Virtual commissioning verification of the real PLC or robot program will be performed since the program will not be completed within the time frame of this thesis.

2 Literature study

This chapter presents the theoretical framework where different concepts are discussed and prior work within the area of Virtual commissioning are presented. This is the foundation for the creation of the adapted methodology model used for this master thesis as well as identification of necessary activities and procedures needed to perform a Virtual commissioning.

2.1 Virtual commissioning

Virtual commissioning (VC) is a concept where commissioning of a manufacturing system is simulated in a virtual environment in order to detect and correct errors generated during planning, programming and design before installation (Hoffmann and Schumann, 2010). Markis et al. (2012) states that VC includes three important aspects:

1. A mechanical design including actuators, sensors, and behavioral description of a system-related model
2. Machine control, including its input and output signals and
3. Signal connections between sensors/actuators and the control.

A mechatronic design integrates electrics, mechanics and control engineering which forms a holistic view of the three aspects compared to a pure 3D-oriented data model that only represent one aspect. (Kiefer et al., 2008). By utilizing simulation models that includes kinematics, geometric, electric, and control-technical aspects, mechatronic behaviors of the production system can be illustrated which gives a more accurate approach of validation and verification (Markis et al, 2012). Hence, mechanical behaviors with connection to a PLC program can be considered in a virtual environment.

Hoffmann and Schumann (2010) discuss a procedure for how to create a mechatronic model using three steps. Geometrical modelling is the first step where geometry data is imported from the CAD system to provide a geometrical model. Secondly, functional modelling is used and it includes manually allocating actuator functions such as gripping, rotation, translation and sensor functions. Electrical modelling is the final step where electrical inputs/outputs are manually connected to the functional models.

2.2 Approaches of commissioning

Auinger et al. (1999) mention four different approaches that can be used for validate a commissioning:

1. *Real plant and real control system* - The conventional way.
2. *Simulated plant and real control system* - Referred to as “Hardware-In-Loop” (HIL).
3. *Real plant and simulated control system* - Referred to as “Reality-In-Loop” (RIL).
4. *Simulated plant and simulated control system* - Referred to as “Software-In-Loop” (SIL).

In the conventional way, testing is performed during a real commissioning. One conventional method used in industry today is a so called Mock-up. A Mock-up is a method where a controller is connected to a standalone version of the device or system (prototype) before the real system is in place (Schludermann et al., 2000). By testing the PLC program in a real system, errors can be detected and adjusted in by the programmers. However, Schludermann et al. (2000) states that this type of method is expensive and it is hard to design a Mock-up that correspond to the reality. Another disadvantage is that the interaction between the different systems are often ignored that results in an inaccurate model of the reality (Schludermann et.al, 2000). The weaknesses of this method often requires that testing, debugging and verification are done on site during the actual commissioning. This leads to additional costs and the environment of a system that is still not verified and validated can be dangerous for humans. In able to improve verification of a system in the design phase there is a need of another method. One method that can be used is to verify the commissioning in a virtual environment (VC).

In the HIL approach, the mechatronic objects in the simulation are connected to a real control hardware via fieldbus control (Markis et al., 2012). The hardware controller is necessary in advance, but a VC before building the plant is possible (Auinger et al., 1999). If a real plant is available but the control system is simulated, the approach is referred to as RIL (Auinger et al., 1999). This could be useful in those cases where different alternatives of control systems is tested or if the real control system is not available. In the SIL approach, the PLC program is downloaded to a virtual controller where IP/TCP connections enables communications between mechatronic objects and software-emulation controller. (Markis et al., 2012). For example, a software PLC (SoftPLC) can be used in order to simulate the behavior of the physical PLC in a virtual environment. The SIL approach also allows the simulation to take place without access to physical equipment (Markis et al., 2012).

According to Damrath et al. (2014), the HIL approach needs to be used in order to perform a VC. In contrast, Auinger et al. (1999) argues that SIL is enough for performing a complete VC. On the other hand, Makis et al. (2012) mention that a VC project can be built either by HIL or SIL approach. Hence, there seems to be a fragmented perception of which approach to be best suitable for VC.

2.3 Simulation vs Emulation

It could be confusing to differentiate between simulation and emulation. McGregor (2002) discusses the difference between the concepts and a summary of his conclusion is presented in Table 1 and Table 2 where differences and similarities are presented. Emulation seems to be more accurate and operates in real time with focus on representing the reality. Simulation, on the other hand, consists of more approximations than emulation and has the purpose of achieving output results in a cost effective way or to demonstrate functionalities. One functionality can be to test the sequence of welding operations in a cell with multiple robots in able to optimize the sequence. However, the movements and signal handling might not be represented with enough accuracy in the simulation model, or not even at all, to verify that a specific robot program will work in the reality. According to McGregor (2002), it is not recommended to speed up emulation since control systems are designed to work in real time.

Table 1: Differences between simulation and emulation

Simulation	**Emulation**
Decision making happens instantly. E.g. movement of a conveyor in a model occurs directly after the calculations are performed.	Decision making happens after a series of steps of which each step takes a certain amount of time. E.g. movement of a conveyor in a model occurs after: a bar code has been scanned, the information has been sent to system, control system has verified the movements, and an electric motor has received a signal to move conveyor.
The model is repeatable. The logic is built in the model and the model has its own simulation clock.	The model is not repeatable. The logic and clock of the control system are separate from the model itself.
Used to test and develop different solutions and to demonstrate functionality and results in a cost-effective and flexible environment.	Used to test the operation of the control system under different system conditions and for training operators and maintenance staff.
Focus on high speed to get output results quickly. Faster than emulation.	Focus on imitating the real world by using real time in model.
More approximations	Model and calculations are more accurate.

Table 2: Similarities between simulation and emulation

Similarities
3D environment (2D environment is not recommended)
Should be accurate enough to fulfill its purpose
Leave nothing for interpretation and should be easy to understand and model. E.g. A conveyor should behave and look like a conveyor.

2.4 Verification vs Validation

Model verification is defined as "*substantiating that the model is transformed from one form into another, as intended, with sufficient accuracy. Model verification deals with building the model right.*" (Balci, 1998, p.215). Converting a model representation into an executable computer program is an example of building the model right according to Balci (1998). However, in able to decide if the model behaves as intended in relation to the task and the purchaser's wants, the model needs to be validated. Validation is defined as "*substantiating that the model, within its domain of applicability, behaves with satisfactory accuracy consistent with the study objectives. Model validation deals with building the right model.*" (Balci, 1998, p.215).

One example can be the verification and validation of a piece of code. The verification can be seen as checking that the code is working according to the programmer's intention with no bugs in it, by for example debug the code. Validation is rather to analyze that the code is doing the right thing in relation to the given specification from the purchaser. Rabe et al. (2008) states that verification and validation should not be seen just as a task at the end of the modeling phase. When a comprehensive change has been done to the model, the result should be validated immediately. This can detect faults early in the project and can limit the risk of doing the same faults during the further efforts of the modeling. Rabe et al. (2008) illustrate a general methodology for modelling that allow validation and verification during the whole project. As soon as sponsor needs are identified and the project has started, data collection and preparation continues in parallel with the modelling which can be seen in Figure 2.

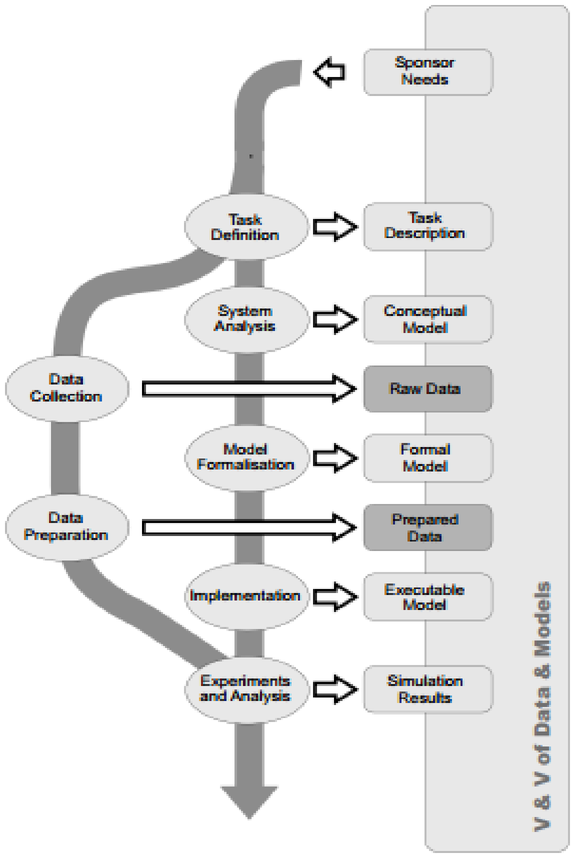


Figure 2: Methodology for simulation described by Rabe et al. (2008).

Sargent (2010) describes some validation and verification techniques that are summarized below:

- *Animation* - Examine the operational behavior of the model graphically.
- *Comparison to other models* - Results (e.g. output) of a model is compared to other valid models.
- *Degenerate tests* - Simplify the examination of the model by narrowing the selected parameters to examine.
- *Event validity* - Compare events in the model to the same type of event in reality.
- *Extreme condition test* - Parameters and structures in the model are set to extreme conditions to validate that the behavior of the model reflect the wanted behavior.
- *Face validity* - Interviews and discussions among experts regarding the behavior of the model.
- *Internal validity* - Determine the variability of occasions by replicate several runs of a stochastic model.
- *Operational graphics* - Visualize graphically selected performance measures as the model runs through time to examine the dynamic behaviors of the model.
- *Parameter variability and sensitivity analysis* - Changing parameters and conditions in the model and examine the behavior/output of the model. That should be in accordance to the real system.
- *Predictive validation* - Predict the system's behavior and then compare it to the model's actual behavior to determine if they are the same.
- *Traces* - Specific entities and logics in the model are traced/followed to examine if the logic of the model is valid.
- *Turing Tests* - Experts validate whether the output results of the model would represent the real system's output.

2.5 Digital Factory

Digital Factory is a concept where different engineering tools are integrated with each other to create a more efficient development of the product and production system (Silva et al., 2015). Sharing information, data and models in a united database enables concurrent engineering and reuse of already created models and information. The Digital Factory methodology often begins with a 3D-model that will be the input when designing the manufacturing process, factory layout, logistic planning and other necessary development tasks. A collection of digital shared data also enables more efficient robotic, production flow and ergonomic simulations. Adapting the Digital Factory can give advantages for the project such as shorter time to market, higher product quality, shorter launch and ramp up time of the production and the ability to be more creative in the development phase (Silva et al., (2015).

Kühn (2006) describes the benefits of Digital Factory compared to conventional engineering. Even though the concept Digital Factory requires more effort in the development phase he argues that the benefits and the reduction of errors in an early stage will reduce total time of the project. Silva et al. (2015) proposes that VC should be added to the model of Digital Factory by Kühn (2006) and states that he does not considered that errors in PLC- and robot programs could slow down the installation and ramp-up of the production system. However, creating a mechatronic model that is needed in able to perform a VC is associated with high effort and expertise which is one reason of why companies still use conventional methods for these aspects (Hoffmann and Schumann, 2010).

Westkämper et al. (2012) claims that there are two main approaches that can be used to reduce the manual effort when creating a VC model: "The library focus" and "Modeling in parallel within the engineering process". The library approach focus on centralized modelling components such as 3D-models, logical behaviors, kinematics and collision objects. Suppliers or different departments within the company provides information and data to the library. The latter approach suggest that a functional description on an abstract level is defined in the initial phase. As the project progress, the functional descriptions are enriched with more information from different disciplines in order to generate the VC.

To further reduce the manual effort of creating a VC model, Westkämper et al. (2012) describe a methodology that focus on automatically generate a VC model with specialized machines. Mechanical engineers provides geometrical parts and the kinematics between them. Electrical engineers provides wiring diagrams that contains connections between parts and to all PLC signals. By implementing an Object Linking and Embedding for Process Control (OPC) that allows connection to the VC model, an automatic generation can be made. Logics are added to the wiring diagram using standard components from suppliers such as actuators, sensors and other electrical components. Hence based on the wiring diagram, logic behaviors could be generated and added to the VC model. The interface between the mechanics and the generated behavior model is also exactly defined. The position of the actuators are defined in the mechanical model and sensor-signals are written to the mechanical model. As a result, the effort of creating the VC model is considerably reduced as well as increased flexibility.

Kiefer et al. (2008) focus on how to standardize mechatronic components and utilize the library approach when building-up a mechatronic cells. They argue that it will be more efficient and profitable for companies in the long term perspective to increase standardization of components used for modelling a VC. Figure 3 illustrates the effects over time of using of a procedure where customized components are stored in a library with standardized components. In the beginning, the creation of mechatronic models are costly and time consuming. However, every time a new object is created, the library of standardized components expands which decrease the time and cost for building a new mechatronic cell. Kiefer et al. (2008) emphasize that this library needs to continuously be updated with reusable components and knowledge in order to increase the efficiency. Rossmann et al. (2007) studied the effects of using a library of reusable controller programs. They developed a library where they imported components and programs when they did their modelling. That resulted in 30 % less effort when building the second model compared to the first model.

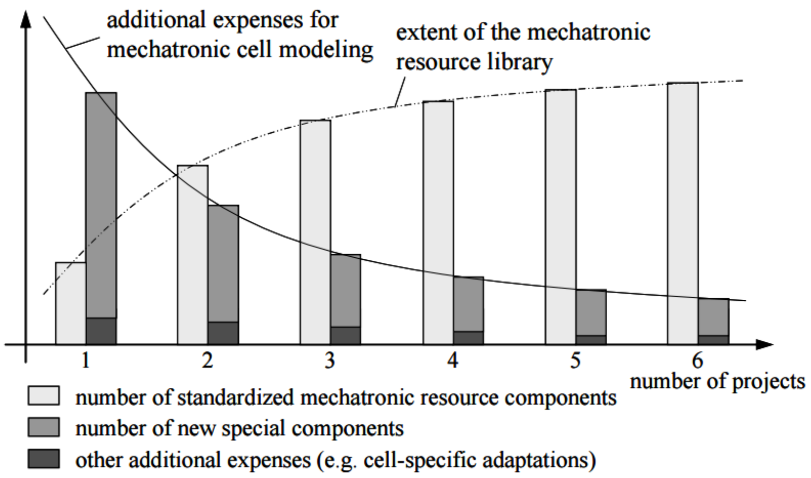


Figure 3: Effects of store and utilize standardized components (Kiefer et al.,2008) .

Another aspect is the managing of the library. The field of data exchange formats has a lot of improvement potentials according to Drath et al. (2008). Since different software are used in the process of creating a VC model, the use of standardized files seems to be essential in order to allow different functions to access/utilize the same data from a centralized library.

Ko et al. (2013) have developed a procedure that allows concurrent engineering of production development including Virtual Commissioning. Traditionally, mechanical and electrical design has been done in sequence. By splitting the virtual model into two sub-models, mechanical and electrical design can work concurrent. The design of the procedure is divided in four major steps. (1) Process design, (2) physical device modeling (geometry and kinematics), (3) Logical device modeling (functional and electrical design) and (4) system control modeling. The output of the first step will be a sequence of operations (SOP) and will be the input to the last three steps that can be performed concurrently without interfering with each other. The two sub-models are then joined together and creates the virtual plant. The procedure is illustrated in Figure 4 below.

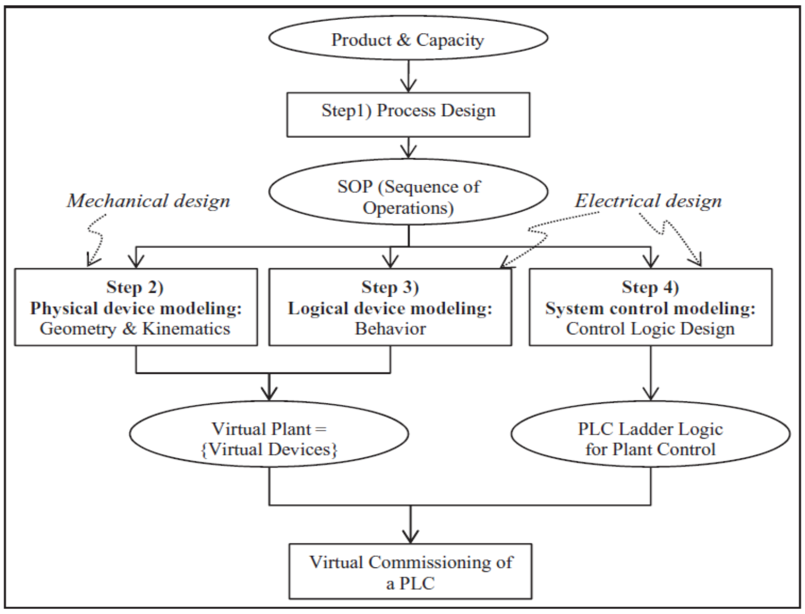

Product & Capacity
Step1) Process Design
SOP (Sequence of Operations)
Mechanical design
Electrical design
Step 2) Physical device modeling: Geometry & Kinematics
Step 3) Logical device modeling: Behavior
Step 4) System control modeling: Control Logic Design
Virtual Plant = {Virtual Devices}
PLC Ladder Logic for Plant Control
Virtual Commissioning of a PLC

Figure 4: Procedure for working with Virtual Commissioning in a concurrent environment (Ko et al.,2013).

2.6 Physics engine approach

“Strength of material” components are geometrical forms in a simulation environment that are affected by external forces (Strahilov et al., 2012). These can be divided into to two different categories: Rigid components and flexible components (Strahilov et al., 2012). Rigid components does not deform when force is applied. Flexible components, on the other hand, undergo elastic deformation when force is applied with the use of finite element calculations.

In order to simulate physical behaviors, physics engines are needed. Physics engines consists of algorithms and software libraries that simulate physical behaviors based on the components’ characteristics (Strahilov et al., 2012). Even though the model provides a more accurate simulation environment with more considerations, Strahilov et al. (2012) argues that the complexity of each component is limited as well as the number of components in the plant modelling. Flexible components, in particular, requires complex meshing and enormous computing time to be included in the simulation environment.

Strahilov et al. (2012) describes a methodology of building a model where the physical behavior of components are taken into account by utilizing a physics engine from the computer game industry. This methodology is illustrated in Figure 5 below.

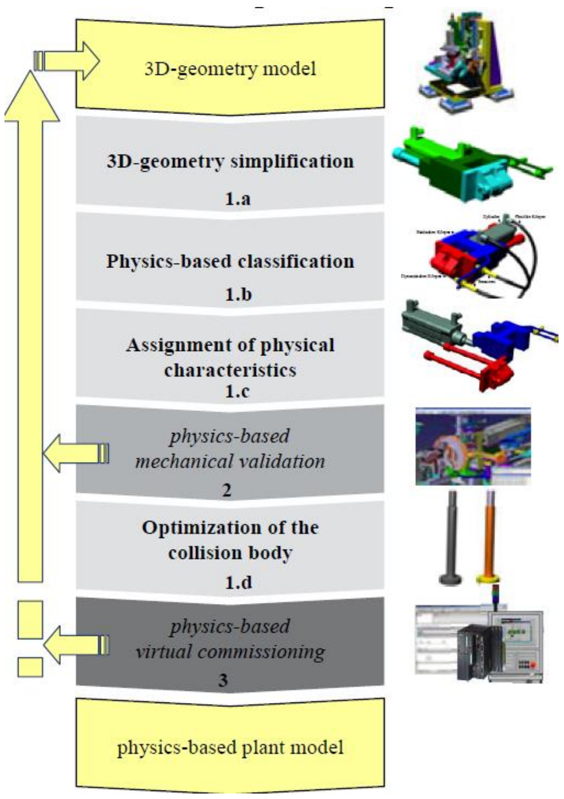
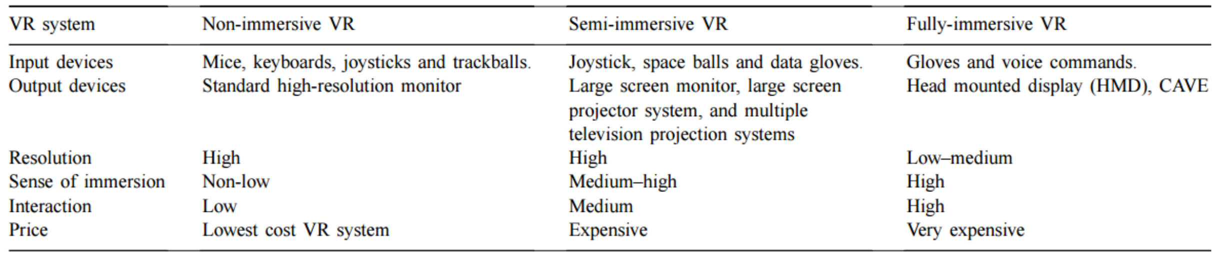

3D-geometry model
3D-geometry simplification
1.a
Physics-based classification
1.b
Assignment of physical characteristics
1.c
physics-based mechanical validation
2
Optimization of the collision body
1.d
physics-based virtual commissioning
3
physics-based plant model

Figure 5: Methodology of plant modelling with consideration of physical behavior developed by Strahilov et al. (2012).

2.7 Virtual Reality

Virtual Reality (VR) is a concept that focus on interacting the human with the computer screen (three-dimensional world) in order for the user to perceive the computer world as it was the real world (Mujber at al., 2004). According to Mujber at al. (2004), VR can contribute to decreased time to market and product cost since it is a powerful tool for testing and evaluate new products and ideas. Mujber at al. (2004) categorize the VR systems into three main areas. These are described in Table 3.

Table 3: Description of the three categories of VR systems (Mujber at al., 2004).



VR system	Non-immersive VR	Semi-immersive VR	Fully-immersive VR
Input devices	Mice, keyboards, joysticks and trackballs.	Joystick, space balls and data gloves.	Gloves and voice commands.
Output devices	Standard high-resolution monitor	Large screen monitor, large screen projector system, and multiple television projection systems	Head mounted display (HMD), CAVE
Resolution	High	High	Low–medium
Sense of immersion	Non-low	Medium–high	High
Interaction	Low	Medium	High
Price	Lowest cost VR system	Expensive	Very expensive

A big application field for this technology is the game industry. There are several studies of how to apply VR in industrial applications. Björklund et al. (2015) study the possibility to combine the concept of VR with VC. The goal was to build the VC model in Process Simulate and utilizing VR techniques from Oculus Rift to integrate these solutions. However, the integration with Process Simulate did not succeed because of the lack of a head tracking feature. Though, they could connect the VR with the simulation environment where the robot movements were simulated. That resulted in a more realistic experience than watching a screen.

2.8 PLC & communication in industrial automation

This chapter explains the basic theory of PLC and communication protocols.

2.8.1 PLC theory

A PLC is a microprocessor-based device that is used to program and control machines and automation equipment (Reddy, 2015). The basic arrangements of a PLC can be seen in Figure 6. The memory is the place where the program is located and must be written in one of the languages included in the IEC 61131-3 explained below. The processor assignment is to interpret the input signals, execute the program in the memory and setting the correct output signals depending on the logic in the program. Signals received to the PLC are called inputs. Signals from a button or sensor values are some example of input signals. Signals that are created by the PLC and sent out to external devices are called outputs. Examples of output signals can be the signal to start a motor or the signal to turn on a lamp on a control panel.

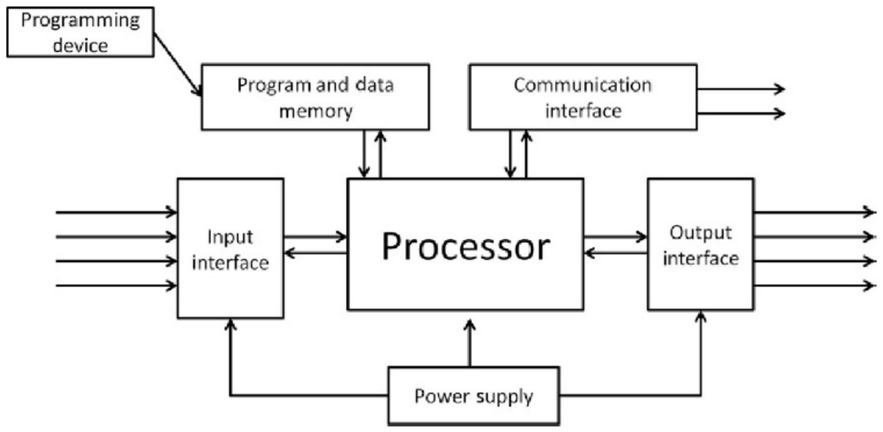


Figure 6: Basic arrangements of a PLC (Reddy, 2015)

Input and output signals are often referred as I/O signals. I/O signals can be either of the digital (discrete) type or analog. Digital I/O signals can only hold the state ON or OFF meanwhile the size of the analog signal is proportional to the size of the variable being controlled (Reddy, 2015). An example of an analog output signal can be the signal to controlling a value. By changing the analog output signal from the PLC the value position can be everything between 0-100 percent open meanwhile a digital output signal only can close or open the value to its maximum.

Programming of PLC follows the IEC 61131-3 standard and include five different programming languages (Erlandsson and Rahaman, 2013). Structured Text and Instruction List are textual based languages meanwhile Ladder Diagram and Functional Block Diagram are graphical programming languages. The fifth language Sequential Function Chart is also a graphical method where a sequence of the program is created including steps (where executions are preformed) and transitions (guard for stepping forward in the sequence).

2.8.2 Communication protocol theory

Transmission Control Protocol (TCP) is the communication protocol used to transfer data between applications on different devices while Internet Protocol (IP) is needed to transfer data between two devices (Dostálek and Kabelová, 2006). The IP address is a numerical label specifying the identification of a device which enables connections to other devices (Dostálek and Kabelová, 2006). To clarify the definitions, one could relate the IP as the way to connect to a specific building and the TCP is then used to access a specific resident in the building. The IP address would be the name and location of the building.

The TCP uses a concept called windowing which is a way of the client to send fractional packages of data to the server application (Dostálek and Kabelová, 2006). After the server has received the package, an acknowledgement is send back to the client which allows the client to continue send data packages to the server. After each acknowledgement, the packages increases in size.

2.8.3 OPC

OPC (OLE for Process Control, OLE = Object Linking and Embedding) was developed in the mid-90s by a group of automation companies (later called the OPC foundation) with the mission to simplifying exchange of process control data (OPC Foundation, 2016). During these times, the vendors for automation equipment often used their own protocol for communication between their different devices in the automation system. These protocols made it hard to establish communication between devices from different vendors and resulted in high implementation costs (Reddy, 2015). The purpose with OPC is to provide a standard infrastructure for enabling exchange of process control data irrespective of its vendor.

OPC has two main components, the OPC server and the OPC client (Reddy, 2015). The OPC server is used to read or write data to a device. The device can for example be the PLC as shown in Figure 7. The OPC client is used to access the data from the OPC server and is embedded or used as a plug-in in the system that needs access to the data. In Figure 7 the HMI (Human Machine Interface) application has an embedded OPC client and can make a request to the OPC server for accessing the latest value of a specific variable. In this way the value of a variable in the PLC can be displayed on the HMI.

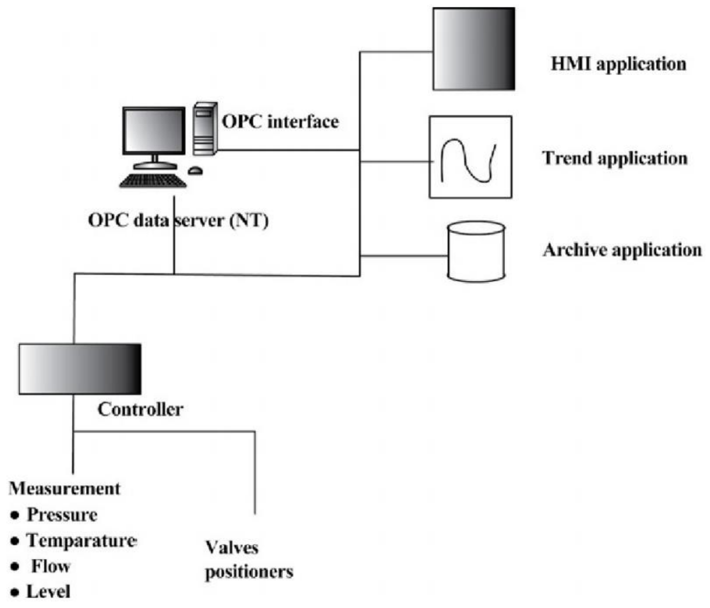


Figure 7: The architecture *of OPC communication (Reddy, Y.J 2015).*

The benefits with OPC-Communication has made it popular to use in simulations software where a real or a soft-PLC is used to control the process simulation with the mission to verify for example the control program (Carlsson et al., 2012). An OPC-client is then used in the simulation software tool and is connected to a real or a soft-PLC through an OPC-server. However, Carlsson et al. (2012) discuss the weaknesses of using an OPC communication when using a simulation software for control program verification. Carlsson et al. (2012) points out four major problems that can result in unreliable verification due to the OPC-connection with complications such as faulty sensor signals and missed signals. Bellow follows a brief description of the different problems stated by Carlsson et al. (2012).

Jitter: Time uncertainty in the microprocessor of the computer used and results in randomly time delays.

Time delay: If the delay is constant the delay is referred to as time delay. The total time uncertainty contains both the jitter and the time delay.

Race condition: If two or more signals that is connected to the PLC through the OPC connection becomes true at the same time but is transferred to the receiving equipment at different times, the problem is categorized as a race condition.

Slow sampling: If the sampling (update rate) of the OPC does not match the sampling time of the PLC or the simulation, slow sampling can occur. Changes in the PLC that are faster than the OPC update rate will not be registered by the client and the signal is never detected in the simulation.

Carlsson et al. (2012) states that the purpose of OPC has never been to be used as a data exchange standard that could support real time exchange protocol for simulation tools. However, Carlsson et al. (2012) suggest that OPC could be used together with simulation based PLC verification. Though, this requires that the PLC vendors, simulation software vendors and the OPC foundation together could develop an extension to OPC where a synchronization mechanism could be specified in the OPC-standard with the purpose to eliminating the problems states above.

2.9 Prior work

Skoogh and Johansson (2008) describe a methodology to increase precision and rapidity of the input data management. They mentioned that data can be divided into three categories: Available, not available but collectible, not available and not collectible. Available data should be gathered as much as possible. Not available data but collectible can be gathered through time-studies, interviews etc. Data that is not available and not collectible might need estimations and approximations.

Dzinic and Yao (2013) made a VC in Experior which was made through five steps: Data collection, 3D model development, PLC programming and I/O signals, simulation model development, communication between simulation model and the PLC program. This process is illustrated in Figure 8. The workflow of the CAD models can be identified in Figure 9 where they started with simplifying the CAD drawings for the VC environment. Thereafter, they did the modelling of components and implemented them in a library. Before the VC could be performed, the creation of the virtual plant was made and components were implemented.

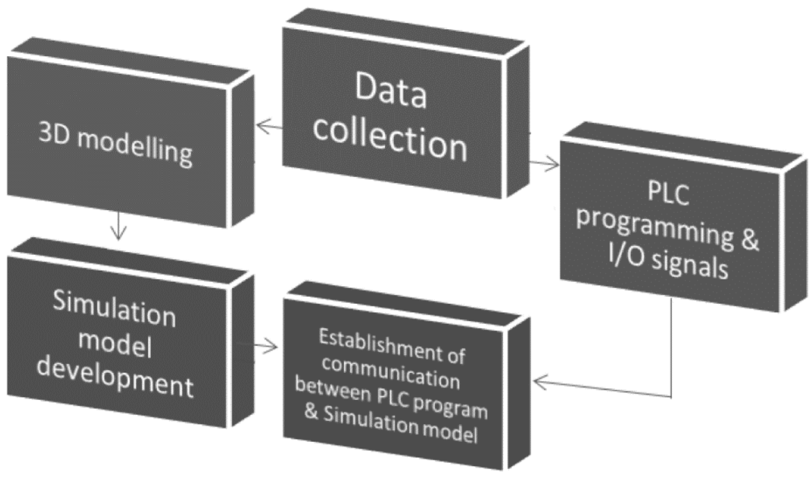

Data collection
3D modelling
PLC programming & I/O signals
Simulation model development
Establishment of communication between PLC program & Simulation model

Figure 8: Dzinic and Yao (2013) process of performing a VC.

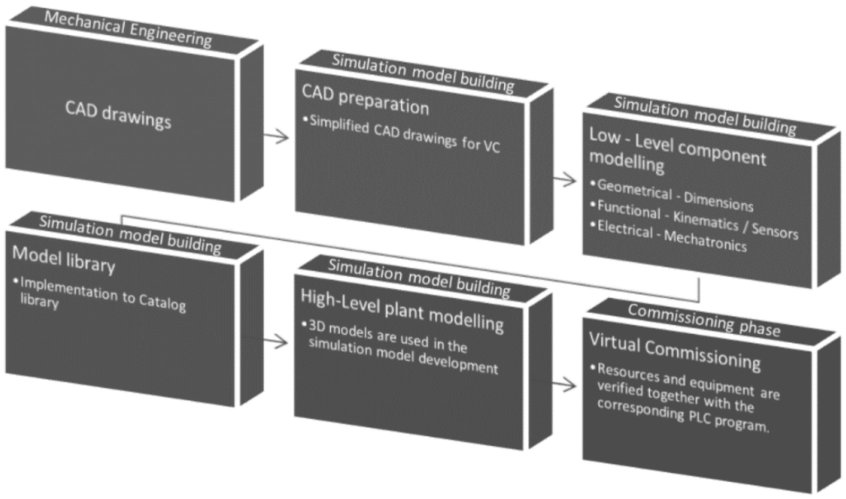

Mechanical Engineering
CAD drawings
Simulation model building
CAD preparation
•Simplified CAD drawings for VC
Simulation model building
Low - Level component modelling
•Geometrical - Dimensions
•Functional - Kinematics / Sensors
•Electrical - Mechatronics
Simulation model building
Model library
•Implementation to Catalog library
Simulation model building
High-Level plant modelling
•3D models are used in the simulation model development
Commissioning phase
Virtual Commissioning
•Resources and equipment are verified together with the corresponding PLC program.

Figure 9: Dzinic and Yao (2013) description of CAD model's work low.

The conclusion made by Dzinic and Yao (2013) was that VC has potential to decrease the overall project lead time for automation development by verifying PLC-programs in an early stage of the commissioning process. Another result was that the model showed no difference in either delay, interruption or behavior when both a hardware PLC (HIL) and virtual PLC (SIL) was used to control the model. However, they discuss that only simple I/O signals was used and no function such as servo motors and frequency converters had been used. A more complex model could thus show more difference using HIL or SIL.

Dzinic and Yao (2013) also mentions some challenges and obstacle that arose during the project. One was that Experior and SketchUp does not used the same coordinate system rotation. This created problems during the modeling and is something to consider when using Experior and SketchUp. Another challenge and something to keep in mind was that when a 3D model was imported to Experior, an invisible block was surrounding the model. This resulted in that features in the graphical representation such as whole was not detected by sensors in Experior. One solution according to Dzinic and Yao (2013) is to separate the 3D-model into separate parts but instead they used a solution where the sensors was moved.

Erlandsson and Rahaman (2013) conducted a master thesis within virtual commissioning. The purpose was to analyze if it was possible to get a real time communication between a virtual model made in Experior and a hardware PLC. The model and connection was tested by running PLC-programs on the PLC and analyze the behavior in the model. One finding was that there sometimes was a delay between exchanging the signals and resulted that the model stop after long runs. By changing the time that the sensors signal was high from 0,2 seconds to 0,6 seconds, was one countermeasure that was done. Another finding was that large CAD-files required a high speed CPU in able to drive the simulation model. One challenge during this thesis was to develop the virtual servo axis drive and get it to work with the PLC. Due to this complexity, electrical motors without servo axis control were used.

Makris et al. (2012) present a case study where an industrial cell with two cooperating robots were validated and tested through VC. The methodology they used are described below:

1. *Mechanical model development*
    - Included positioning and modeling of sensors and components, definition of the degrees of freedom and constraints, and configuration of moving axes.
2. *I/O signals definition*
    - Defined all input and output signals in a list.
3. *Material flow definition*
    - This step included sequence of operations and signal assignment for I/O signals to the robots, the sensors, the operators, and any other entity. Dummy signals were also defined to allow control of the cell through a HMI. Behavior of the model was then verified by executing programs and verify the resources (PLC, robot). Interlocks in the control simulation software and simulation activities were identified.
4. *Human Machine Interface (HMI) definition*
    - Created a HMI to simulate control panels or ongoing activities e.g. an operator loading parts. HMI was an essential part in the model since SIL was used.

Guerrero et al. (2015) describe their study of how to build up a customized machine using CAD software and implement these models in Process Simulate to perform a VC. The machine existed physically but needed to be implemented into a virtual environment. Their procedure consisted of five steps described below:

1. *Characterizing the system*
    - This step consisted of identifying the functionality of the system and what activities that needed to be done.
2. *Computer Aided Design*
    - This step included drafting of all components in a 1:1 scale.
3. *Virtual environments*
    - The purpose of this step was to simplify some steps in the kinematics representation in the Process Simulate program. The Kinematics Editor and Cyclic Event Evaluator were used to connect the links and joints and to test the virtual environment respectively. Then internal logic block and actions could be implemented into Process Simulate.
4. *Testing the virtual environments*
    - This step consisted of two test methods. Firstly, validation of joint movements and sensor detection was performed by forcing signals to different states. Thereafter, verification of the PLC program was made by connecting the PLC program to the model using a HIL approach.
5. *Virtual environments as a monitoring system*
    - In this step, the program was downloaded to the physical system and tested.

3 Benchmarking

Interviews have been conducted at different companies that are frontier within VC. The interviews was focusing on what methods that are used in industry today in order to verify and validate their PLC/control programs before the physical system is in place and what visions that exists. A summary of the interviews conducted at the different companies are provided in the subsections below. Semi-structured interviews were conducted which means that the questions consisted partly of open questions as well as prepared questions in order to let the interviewees talk freely combined with answering specific questions (Barriball & While, 1994).

3.1 ÅF

At ÅF, Fabian Fasth[1] and Andreas Buhlin[2] was interviewed through face-to-face conversations. Fabian has worked with VC in prior projects and built up a VC model for testing and verification. Andreas was responsible for the division of automation where he constantly strived of an increase in VC within Industry.

ÅF is an abbreviation for Ångpanneföreningen and is one of the largest technical consultant companies in Sweden. One area of expertise is automation where ÅF delivers solutions and provides knowledge and consultants to different industries. Andreas Buhlin is head of one automation division in Gothenburg and sees the potentials of VC. He states that: "*The question is not if Virtual Commissioning will be frequently used in Industry, the question is when companies dare to invest in the technology*." Though, he considers the initial costs to be too great at the moment and the technology is not yet standardized and developed to the extent that it is profitable to include in automation projects. Therefore, most of the automation projects are verified through Factory Acceptance Test (FAT) where a prototype of the production system is built up and programmers can verify PLC and robot programs on physical equipment.

However, ÅF has experience within VC and has used the concept in prior automation projects with the goal of being competitive on the market when the technology will blossom. The software that has been used at ÅF is Experior for the emulation environment with connections to Siemens Step 7 where the PLC programs are created. The programs from Siemens Step 7 was uploaded to a Siemens softPLC that was connected to the emulation environment. For robot movements, RobotStudio was connected to Experior using servers with TCP/IP protocol that reflected the movements from RobotStudio into Experior. In addition, Cognex Vision was used to emulate vision system in the production system and the HMI was connected to the softPLC. Figure 10 illustrates the VC architecture used at ÅF for Siemens PLC.

[1] Fabian Fasth (Simulation engineer, ÅF Gothenburg) interviewed by the authors 27 May 2016.
[2] Andreas Buhlin (Head of automation division, ÅF Gothenburg) interviewed by the authors 31 May 2016.

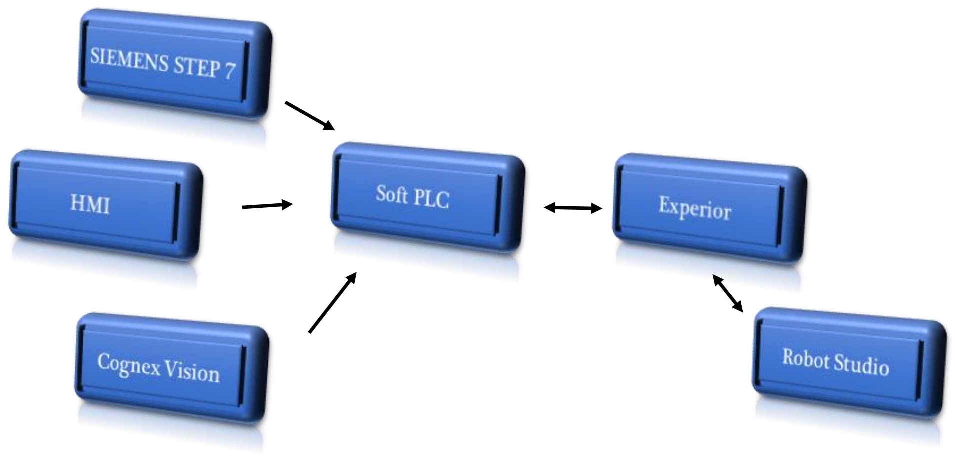


Figure 10 - The VC architecture used at ÅF

3.2 KUKA

Two engineers were interviewed at KUKA, Thomas Kieker[3] and Martina Kammler[4]. Thomas area of expertise was within robotics and robot simulation meanwhile Martina worked with building up logic expressions that later was used for the VC. The interviews were conducted through Skype.

KUKA is a global company based in Germany that focuses on automation and industrial robots. They have been doing VC in industrial automation projects the last five years. Since they started with VC, the commissioning time has been reduced from 4-6 weeks to 2-3 weeks (reduction of 50 %). They have two main perspectives of motivating VC:

- The customer demands VC and pays extra
- To decrease the commissioning time on site.

The architecture that KUKA used for VC is illustrated in Figure 11. HIL was used, that is utilizing a hardware PLC instead of a softPLC. To enable connection between the PLC and the emulation environment, SIMBA-BOX was used so that bus components and bus structure could be emulated from the PLC. WinMod was used to build up and emulate the logics of each component in the VC model. WinMod was connected to RF::Suite that is a platform for working with robotics. RF::RobSim is one part of the RF platform that KUKA used to emulate the robot controller that consisted of the specific robot programs. For visualization of the virtual production plant, RF::SGView was used. When the VC took place, the robot programmer was focusing on the robot programs meanwhile the PLC programmer was responsible for the PLC programs.

[3] Thomas Kieker (Engineer within robotics, KUKA) interviewed by the authors 21 April 2016.
[4] Martina Kammler (Engineer within logics and virtual commissioning, KUKA) interviewed by the authors 21 April 2016.

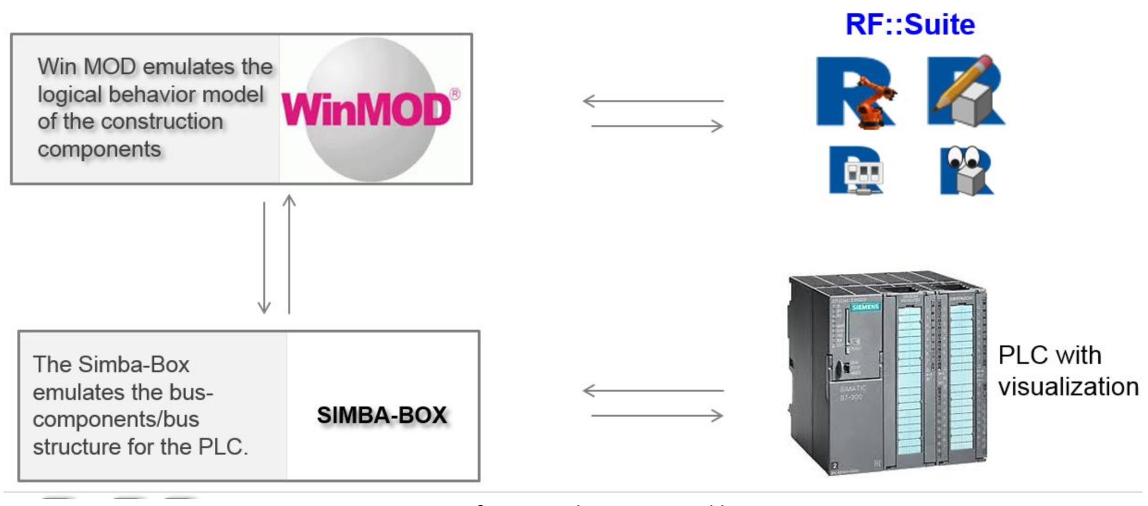

Win MOD emulates the logical behavior model of the construction components
WinMOD
RF::Suite
The Simba-Box emulates the bus-components/bus structure for the PLC.
SIMBA-BOX
PLC with visualization

Figure 11: Software architecture used by KUKA

KUKA divides the terms simulation and emulation in different work disciplines. Simulation includes the robot movements and verifying collisions meanwhile emulation is when the different applications are integrated and combined to verify the concept. In KUKA's working methodology, simulation needs to be done before doing the emulation. In addition, the creation of the VC models is not started before electrical engineers and mechanical engineers are finished in order to reduce unnecessary rework. However, KUKA's architecture and software establishment are standardized and developed during the years which also developed stable interactions.

3.3 Virtual Manufacturing

Kåre Folkeson[5] was interviewed at Virtual Manufacturing by face-to-face conversation. Kåre has a background in 3D Computer Aided Design and simulations. At 2006, he was part of the start-up of the company Virtual Manufacturing together with three coworkers. Virtual Manufacturing is a consultant company that have four main areas of expertise where Robotics and Automation is one of them. Within the department of Robotics and Automation, they do simulations and preparations for automated solutions mainly to automotive industry.

Before they worked with VC, robot controllers or simulated controllers (mimic controller) was used as PLC. Today, Virtual Manufacturing utilizes Process Simulate V.6 as a platform for building the VC model where the PLC logics can be verified and errors can be detected in an early phase. The latest VC project was the implementation of the painting section at Volvo Cars. The VC included fixtures on a conveyor that interacted with painting robots.

There are some challenges of VC that Kåre considers are important to solve in order for a breakthrough within VC. The ability of connecting and interacting with different applications is one challenge. Also, possess the right competence and develop knowledge within the area. In addition, the time to build the model is too long today which makes it unprofitable. Handling CAD models and data could also be a challenge since the complexity of the models utilizes too much computer power. Hence, the importance of optimization of CAD models is essential.

[5] Kåre Folkeson (Group Leader of Robotics and Automation, Virtual Manufacturing) interviewed by the authors 22 April 2016.

To enable building VC in a concurrent environment, Kåre states that a centralized database is a key. He also emphasizes that the technique of a shared database is already developed and used today (e.g. TeamCenter) so it should also be integrated in the VC concept.

4 Virtual Commissioning Framework

This chapter explains the methodology developed by the authors for this thesis work and describes how to build a VC model in a concurrent environment based on the literature studies and interviews. Figure 12 shows an illustration of the methodology.

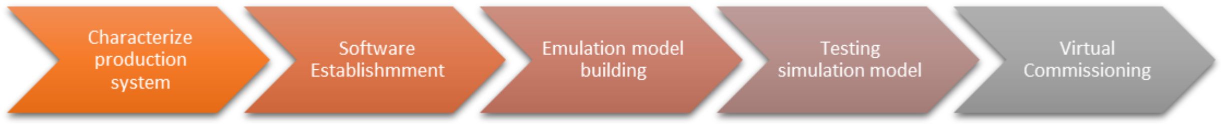


Figure 12: Process for creating a Virtual commissioning model.

4.1 Characterize the production system

The first step in the process for creating a VC model is to characterize the production system that is going to be modeled by collecting necessary data. Makris et al. (2012) mention some of the necessary data that is needed in order to build a VC model and are summarized below:

- Layout of the production cell including equipment and exact placement.
- Material flow in the shop floor including sequence of operation and the relation between production processes.
- Information about PLC and other control equipment.
- Detail definition about I/O signals and the mapping between resource components.
- IT infrastructure including fieldbus hardware, software drivers and communication protocols.
- Information about extra functionalities such as alarm and HMI functions.

In a concurrent environment where e.g. layout, material flow, I/O lists are continuously updated and maybe not yet defined, a general perception is still needed in this phase. As new information and functions are updated during the project, the description from this phase also needs to be updated and accessible to the affected project members.

To develop a VC model for an existing production systems, gathering data might be easier compared to a non-existing production system. The perception of the material flow and the construction of machines might be more concrete as the production could be observed in reality. In addition, data could be gathered by studying the actual information from the current production system. However, modelling a non-existing production system might need another approach since there is no physical production system to observe and study. Therefore, it puts higher demands on accurate specifications and visualization. Interviews might be needed to be conducted where experts within different fields provide a description of the future system and machines. Data collection is essential in both cases, but the accessible data and functionalities might be in a more conceptual phase in a non-existing production system compared to an existing one.

4.2 Software establishment

The purpose of this phase is to define the architecture of the VC and establish communication between software. By realizing the architecture and what software that are needed, OPC and communications between software does not need to be a limitation when modelling the VC model. Drath et al. (2008) claim that one critical issue in the process of building a VC model is to initiate OPC communications between software. Having a working OPC communication in an early state enables verification of the model using a softPLC throughout the modelling phase. In addition, if it is concluded that there might be complications in establishing communication between applications for the VC, these complications can be identified before starting the VC modelling. That could contribute a reduction of rework if, for example, the modelling is needed to be done in another software or to allow software developer to develop plugins and updates in time so communications could be established. Therefore, the architecture of the VC needs to be established before starting the VC modelling and identify possible complications with the architecture.

However, as the concept becomes mature within the company, this procedure might become standardized and more stable with time. As more VC projects are initiated in the company, lessons learned and discussion should be initiated to define what should be changed till next time software connections are established.

4.3 Emulation Model Building

After the Software establishment is initiated, the Emulation model building starts. The processes and procedures of this phase is illustrated in Figure 13 and further explained in this chapter.

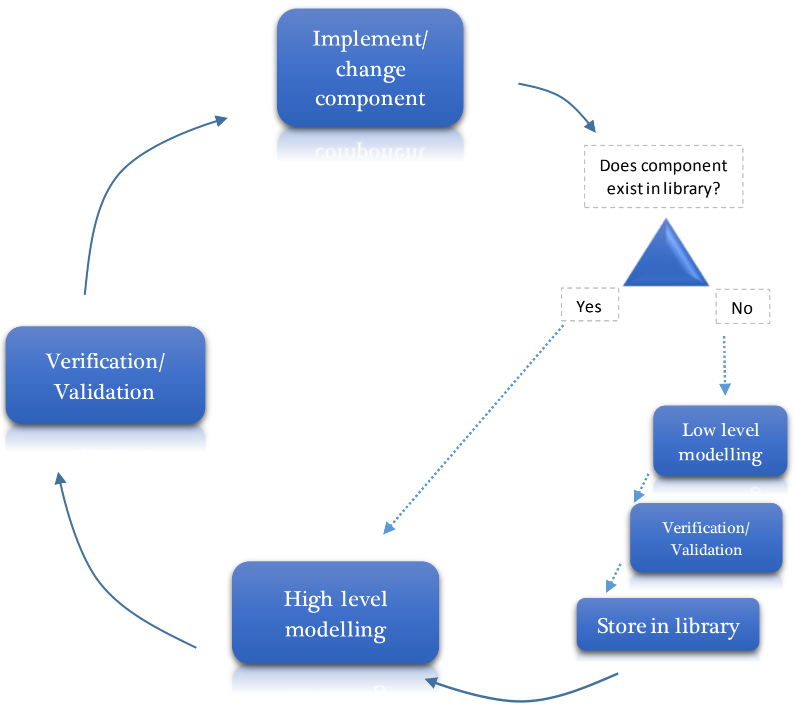


Figure 13: Process for building the emulation model

4.3.1 Low level modeling

If a machine or component is not available in the component library, low level modeling is needed. By utilizing 3D models, functionalities and I/O lists, a mechatronic model could be implemented. In addition, software that provides a physics engine that enables actual movements and behaviors of components need some additional setups.

The first step in low level modeling is to create a 3D model that represents the geometry. In best case scenario, the 3D models are provided by the vendors with a standardized format that can be used when creating the low level component. However, if 3D models are not available, they will have to be manually modeled which could be time consuming. Furthermore, the 3D models could be very detailed and include lots of information which result in high CPU usage when the models are going to be used in the emulation which can affect the quality of the emulation (Dzinic and Yao, 2013). Therefore the models need to be pre-processed before they are imported into the simulation environment. Hidden lines, unnecessary layers, and unnecessary details such as holes and bosses are removed. The number of polygons/triangular for the mesh can also be reduced to simplify the model. Simplifying the models manually can be time-consuming, hence utilizing plugin tools and software that support the process is advisable. Dzinic and Yao (2013) utilizes a plugin tool in Sketchup called “Clean-Up” for automatic simplification of parts.

If the low level component is dynamic, functionality and electrical behaviors need to be implemented. An example of functionality could be friction between surfaces, movements etc. In addition, electric signals that are connected to the component are defined by mapping the signals to the component where Excel could be used to organize the I/O:s. Static components, on the other hand, only utilize the geometrical input since there is no need for electrical and functional definitions. When the components’ definitions are completed, they are aligned into an assembly which later is stored in the library to be used in high level modelling.

In able to keep the CPU usage of the emulation model low, one should separate static parts (fixtures, rails and other non-moving parts) and dynamic parts (grippers, buttons etc.). In Experior, static parts can be imported as .dae files and uses less CPU power than dynamic parts that should be imported using the .x format (Dzinic and Yao, 2013).

4.3.2 High level modeling

If the machine or component is defined in the library, high level modelling (also referred to as plant modelling) can be started. In High level modelling, the components found in the library are composed in the emulation model to represent the actual plant. These components already include mechatronic definitions. The distances between different equipment are important as well as the layout of the plant. In addition, extra effort in configuration and mapping of I/O signals between software might be needed.

4.3.3 Verification and Validation

During the Emulation model building phase, verification is done for each low level component by testing the internal functionality. Verification is also carried out each time a low level component has been added in the High level modeling. To validate the completed low level components, the behavior can be compared to the vendor's technical specification and/or by consulting with involved partners in the project which possess deep knowledge of the production site (Face validity and Turing Tests).

The high level model is verified each time a component has been added to the model or if a change has been made. Small PLC programs and/or script variables can be used to trigger events in the model in order to verify and validate and that the behavior of the virtual plant corresponds to the real physical plant. The interaction between the different components should be the focus when analyzing the high level model.

Beside the explanation above, following techniques can be used to verify and validate the model:

- *Comparison to other models*, *Parameter variability and sensitivity analysis*
  - There is often not an exact same model to compare to but some function can be validated by comparing to a similar model done earlier.
- *Degenerate tests and traces*
  - Each function can be verified separately to ensure that they work before the complete model are verified and validated.
- *Extreme condition test*
  - By using extreme but still valid parameters such as motor speed and number of components, the model can be tested under extreme circumstances to ensure correct behavior.
- *Predictive validation*
  - Compare the behavior of the model to the technical specification

4.4 Testing of the emulation model

When the Emulation model building is completed, there is a need of both end verification and validation to make sure that the model behave as expected. Compared with the verification in the emulation modeling, this verification will include more complex and longer sequences of PLC-programs to check that the interconnection between all equipment behave as expected. In order to validate the complete model the behavior will be compared to the product specification given by the customer. The validation will also be done by letting involved people in the project and experts study the model.

4.5 Virtual commissioning

When the model is considered to be acceptable, the model can be used to verify PLC-programs which allows detection of possible problems or issues that otherwise would have been identified during installation and ramp-up.

4.6 Evaluation

When all the steps in the VC process is completed there is important to perform an evaluation. During the evaluation both strengths, weaknesses and challenges with the VC-models should be discussed. This evaluation can later be used to improve the methodology for the next process of creating a VC-model. It is not just the end result that is important to discuss but rather the process for reaching it. Therefor it is important to continually collect and document problems, challenges and ideas that arises during all steps of the methodology.

5 Building the SKF Virtual Commissioning model

This chapter describes the method and the steps used during the modelling of the production cell at SKF.

5.1 Characterize the production system

Since the production system was nonexistent and in its initial project phase, information was collected by studying technical specifications, interviewing involved engineers, and by continuously request updates from involved parties. The fact that the VC model was developed concurrent with the actual design of the cell, the importance of having continuous updates and verification of data through the whole project increased as well as communication with all parties. Data that was not available or not confirmed was approximated until the data became available. This however led to rework where the VC model needed to be updated continuously.

5.2 Software establishment

This chapter provide a description of how the connections between software were established with schematic illustrations and descriptions.

5.2.1 Modelling architecture

An illustration of the modelling architecture is illustrated in Figure 14. CATIA and Google SketchUp was used to create 3D models. When the 3D model was finished, the file needed to be converted into an acceptable format depending on where it should be imported. Experior supports the 3D file format ".x" (Microsoft DirectX) for dynamic parts while RobotStudio supports the STEP format. As for static parts, COLLADA files were supported by both software. Google SketchUp exported DirectX files with a plugin tool called 3D-rad. AnyCad was used to convert the files from CATIA into COLLADA files so the CAD files could be imported into Google SketchUp.

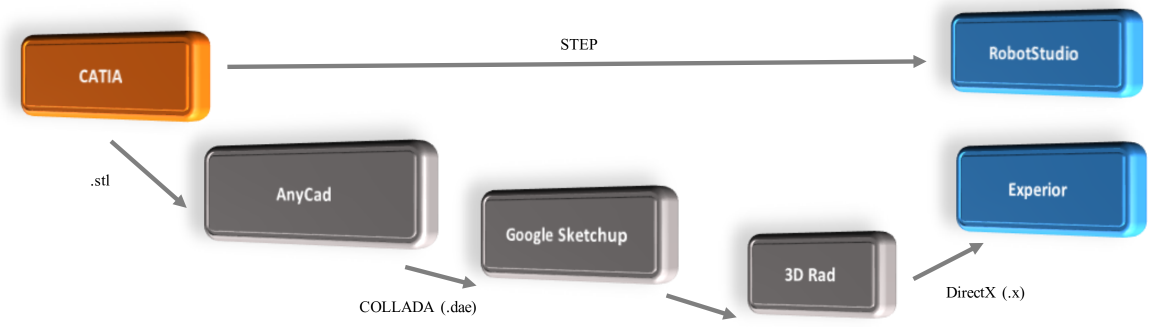


Figure 14: Architecture for converting and importing files into RobotStudio and Experior.

5.2.2 Virtual commissioning architecture

The VC architecture when using Experior is illustrated in Figure 15. The emulation environment was in this case developed in Experior. Codesys, a software for developing controller applications, was used for PLC programming. The softPLC used to execute the program was Codesys WinSys V.3, a softPLC design to follow the IEC 61131-3 PLC-standard. To access the softPLC, a gateway connection needed to be established. The OPC server in Codesys was connected to the gateway where the Codesys WinSys could be accessed through a TCP/IP connection. An OPC client within Experior was used that could be connected to Codesys' OPC server. The symbol list, created within Codesys, contained the all I/O signals that were used in the PLC program and they could be accessible from in Experior when connection was established. To access the motions of the robots, Xcelgo has developed a plugin that enables connections between Experior and RobotStudio by creating two TCP/IP servers that scan for I/O signals and transformations. The plugin scans 25 times per second for transformation changes and sends a message if any changes is found. Hence, the robot motions were reflected in Experior's emulation environment which enabled direct use of the soft robot controller and robot program.

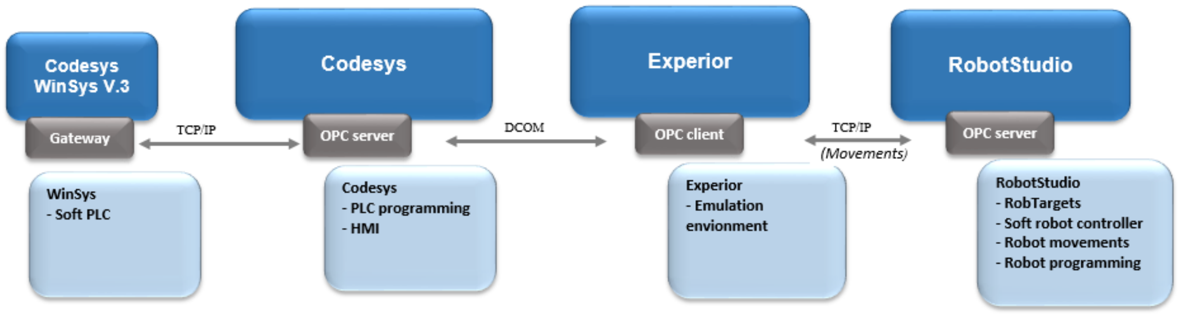


Figure 15: VC Architecture when using Experior.

The architecture of the VC for RobotStudio is illustrated in Figure 16. In this case RobotStudio was used both for simulating the robot movements by having the direct connection to the soft robot controller and as an emulation software where both low and high level modeling was done. RobotStudio used the same architecture for PLC communication as Experior described above, where an OPC client within RobotStudio was used to access the symbol file downloaded from Codesys into the softPLC.

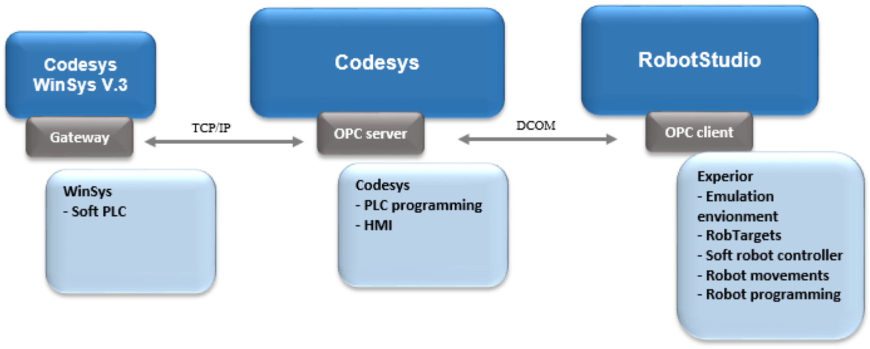


Figure 16: VC Architecture when using RobotStudio.

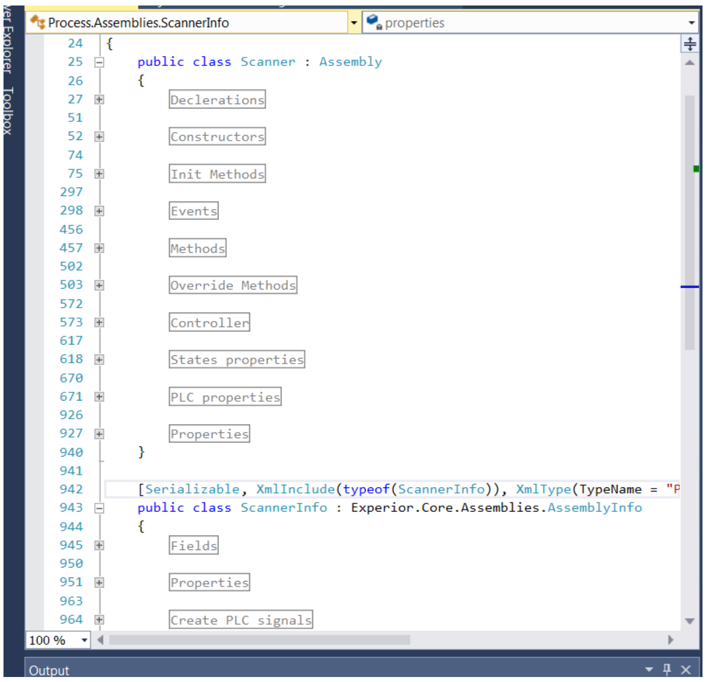
5.3 Emulation Model Building (Experior)

This chapter describes the proceedings of Low level modelling and High level modelling that was used to create the virtual environment that represented the production cell at SKF. The chapter focuses on the emulation model building in Experior.

5.3.1 Low level modelling

Since the developing of the layout and production concept was performed in parallel with the VC modelling, fragments of data and information were available and delivered during the project. Therefore, the geometrical representations was limited at a start where only boxes and simplified developed geometries were implemented. As soon as the actual geometries were defined and what components to include, the temporary geometric representation was exchanged by the real geometric representation. In able to work efficient in the project, existing 3D models from SKF and machine vendors was used as much as possible. These were available through ÅF's department at Laxå where the robot programming was made. They provided updated versions of the current layout that was specified in RobotStudio where robot position targets, CAD models and part of the robot program were included. Hence, the CAD models from RobotStudio needed to be converted and imported into Experior's environment. That required simplification of the CAD files since Experior could not handle the same complexity and CAD data compared to RobotStudio. Since no software for optimization was available, aid was provided by Xcelgo where the developer optimized the required CAD files. For geometries that were not available or included in RobotStudio, additional 3D modelling was required.

Functionality and electrical properties was added to the geometrical representation. In Experior, the properties were defined by programming C# scripts using Visual Studio, see Figure 17. Appendix A includes a C# script for the AGV used in the model. Visual Studio was connected to Experior so that debugging could be accomplished directly in Experior's virtual environment. The logics of each machine and component in the production cell was programmed and stored as an assembly in one defined catalog. The catalog included all customized assemblies used in SKF's production cell. Each of these assemblies could then be gathered from the specified catalog in Experior's emulation environment to be placed in accordance to the layout specifications.

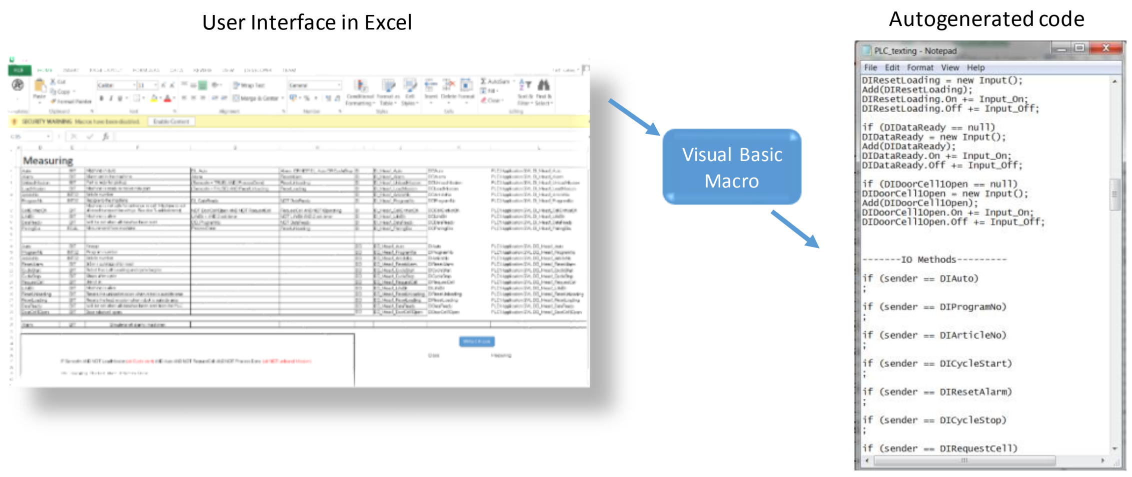


```
{
    public class Scanner : Assembly
    {
        Declerations

        Constructors

        Init Methods

        Events

        Methods

        Override Methods

        Controller

        States properties

        PLC properties

        Properties
    }

    [Serializable, XmlInclude(typeof(ScannerInfo)), XmlType(TypeName = "F
    public class ScannerInfo : Experior.Core.Assemblies.AssemblyInfo
    {
        Fields

        Properties

        Create PLC signals
```

Figure 17: An extraction of the Visual Studio project for creating an assembly and add geometrical, functional and electrical properties.

In addition to the assemblies, customized products that should be included in the physics engine needed to be created. A europallet, bearings and interleaving paper was created in a load class where geometrical representation and physical properties was added. This was also done in Visual Studio by defining the properties using C#.

The structure and coding of the assemblies were similar with each other and some parts identical except for the source names. Therefore, auto generation of code was considered to be a suitable approach. A user interface in Excel was developed where all the defined I/O signals and logic specification were stated for each component in the production cell (see Figure 18). New project inputs was added into the user interface to enable a centralized base for information and data. In addition, macros programmed in Visual Basic was added into the user interface that printed the auto generated code (see Appendix B) for each component that initialized the required PLC signals that were included in the user interface. That contributed to decreased amount of time to initialize the PLC signals for new components and new project updates.



```
DIResetLoading = new Input();
Add(DIResetLoading);
DIResetLoading.On += Input_On;
DIResetLoading.Off += Input_Off;

if (DIDataReady == null)
DIDataReady = new Input();
Add(DIDataReady);
DIDataReady.On += Input_On;
DIDataReady.Off += Input_Off;

if (DIDoorCell1Open == null)
DIDoorCell1Open = new Input();
Add(DIDoorCell1Open);
DIDoorCell1Open.On += Input_On;
DIDoorCell1Open.Off += Input_Off;



-------IO Methods---------

if (sender == DIAuto)
;

if (sender == DIProgramNo)
;

if (sender == DIArticleNo)
;

if (sender == DICycleStart)
;

if (sender == DIResetAlarm)
;

if (sender == DICycleStop)
;

if (sender == DIRequestCell)
```

Figure 18: Auto generated coding.

The connection to RobotStudio was also needed to be configured and stated. By importing the COLLADA file that represented the specific robot from RobotStudio into Experior, the movements of the robot could be mirrored in Experior. This connection was possible by using the plugin that has been developed by Xcelgo. The gripper to the robot also needed to be created and configured. This was done in the following steps:

1. *Optimize the COLLADA file of the gripper*
2. *Import the COLLADA file into Visual Studio project*
3. *Coding the properties (PLC signals, logics)*
4. *Create and position MagnetSensors to the gripper*

The COLLADA file was too large and complex for Experior, hence optimization was required. The coding in Visual Studio was done in a class called IGripper where all properties for the gripper were specified. To enable the gripper to interact with parts of the loadclass within Experior's environment, a MagnetSensor needed to be positioned where the gripper was supposed to grip the load. When a grip signal was received from the PLC, the MagnetSensor activated and attached the load that lied within its boundary (see Figure 19).

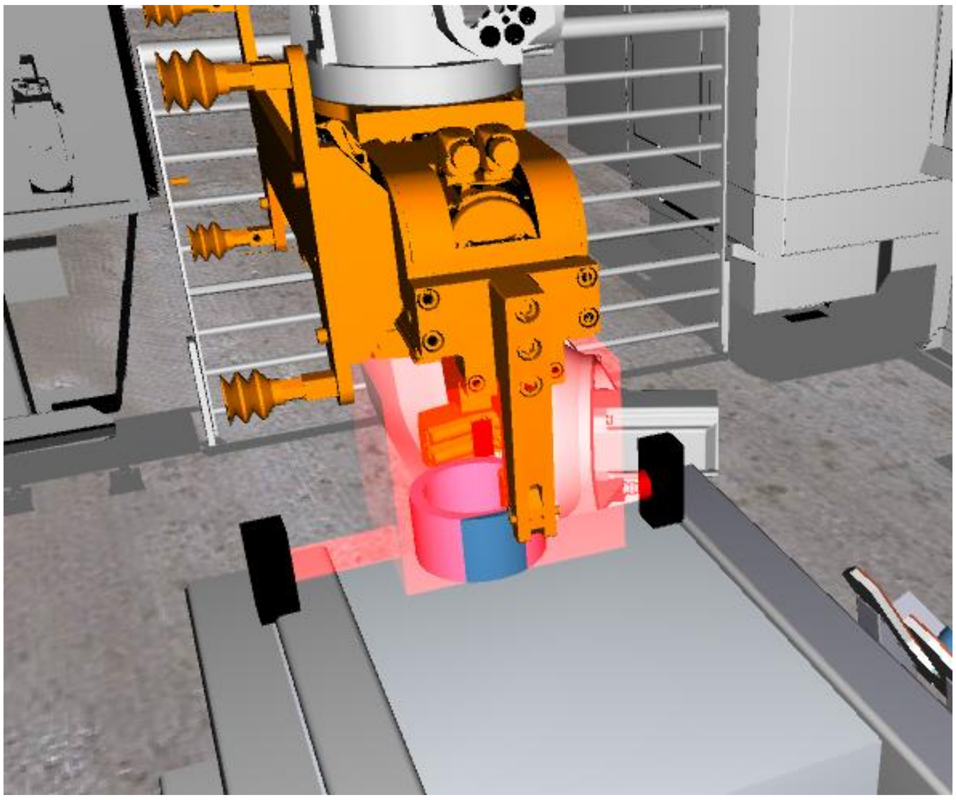
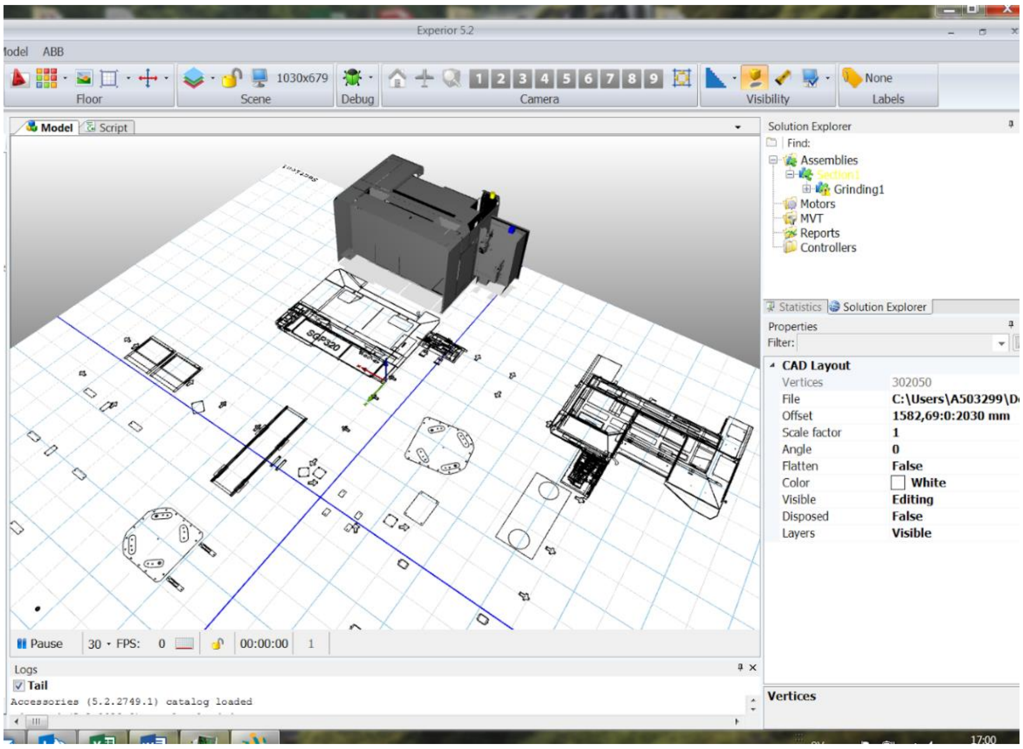


Figure 19: Gripper configurations.

Furthermore, since Experior consisted of a physics engine called PhysX, the components needed some additional setups to include them in the physic engine. The physics engine only considered build-in cubes/cylinders/spheres that were defined within Experior. Therefore, the geometry of a component needed to be represented by the built-in parts in order to include the components' behavior in Experior's physics engine. This was implemented in C# where the physical representation were added and positioned in relation to the mesh. There was also a possibility to let Experior estimate the physical representation of a mesh using triangular and convex calculation methods. However, that resulted in that high utilization of the computer power which jeopardized the accuracy in signal connections and the real time simulation clock.

5.3.2 High level modeling

Since the layout of the production cell was defined in RobotStudio, it needed to be converted and imported into Experior. This was done by exporting the layout as a 2D drawing (.dwg) from RobotStudio and import the file into Experior. The 2D drawing was then illustrated on the floor in Experior's emulation environment (See Figure 20). That enabled an easy drag and drop approach where each component could be placed in accordance to the 2D drawing. When the layout was updated and a new version of RobotStudio was available, then the procedure of exporting and importing a new 2D drawing was repeated.

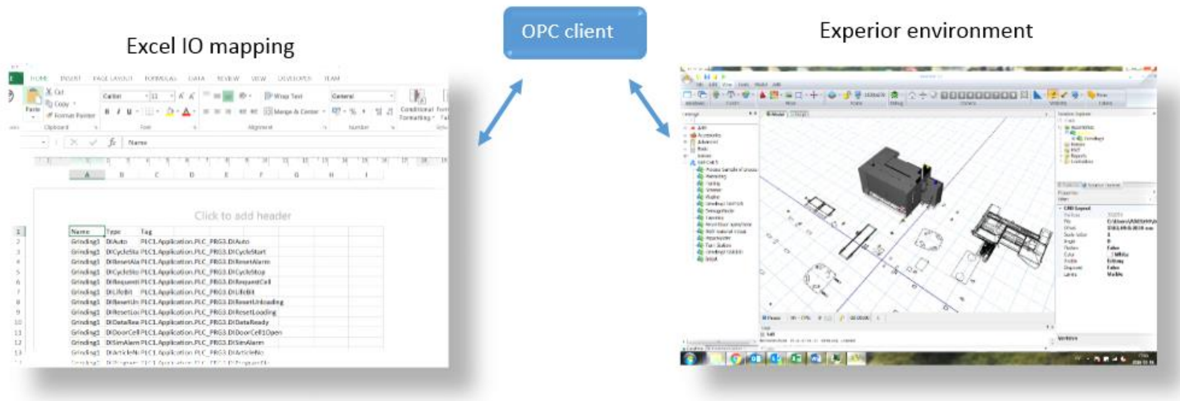


Vertices	302050
File	C:\Users\A503299\D
Offset	1582,69:0:2030 mm
Scale factor	1
Angle	0
Flatten	False
Color	White
Visible	Editing
Disposed	False
Layers	Visible

```
Accessories (5.2.2749.1) catalog loaded
```

Figure 20: 2D drawing imported into Experior's environment.

As the model developed, the I/O signals continued to expand which put high demand on a structured approach to map all I/O signals to the right component in the emulation environment. Experior's OPC client offered a functionality to map all I/O signals using Excel. By structuring the Excel sheet in the right way and provide the required data, the I/O mapping could be done using an Excel document. To generate the I/O mapping in the emulation environment, the OPC client needed to be connected to the right OPC server and the symbol file needed to be generated from Codesys before importing the Excel file. Then, the references in the simulation environment was reconfigured in accordance to the information provided in the Excel document. Figure 21 illustrates the process of interacting with the Excel to generate the I/O mapping.

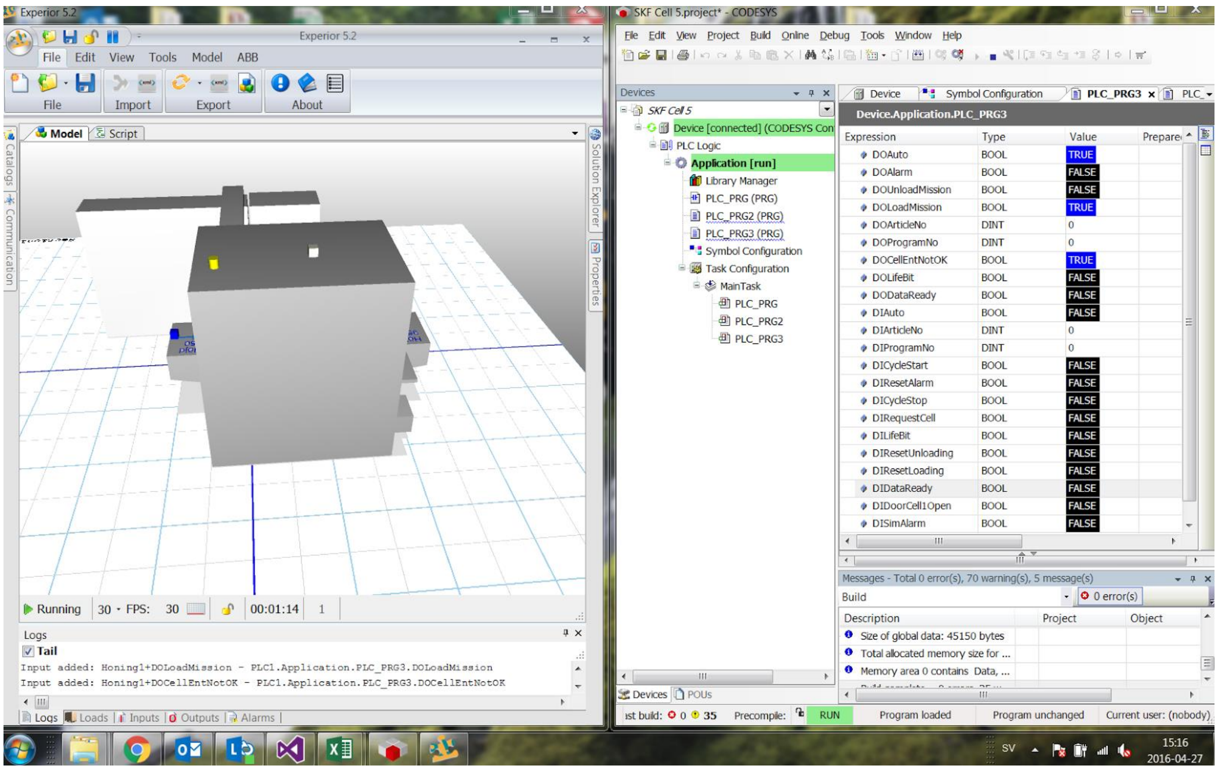


Name	Type	Tag
Grinding1	DIAuto	PLC1.Application.PLC_PRG3.DIAuto
Grinding1	DICycleSta	PLC1.Application.PLC_PRG3.DICycleStart
Grinding1	DIResetAla	PLC1.Application.PLC_PRG3.DIResetAlarm
Grinding1	DICycleSto	PLC1.Application.PLC_PRG3.DICycleStop
Grinding1	DIRequestC	PLC1.Application.PLC_PRG3.DIRequestCell
Grinding1	DILifeBit	PLC1.Application.PLC_PRG3.DILifeBit
Grinding1	DIResetUn	PLC1.Application.PLC_PRG3.DIResetUnloading
Grinding1	DIResetLoa	PLC1.Application.PLC_PRG3.DIResetLoading
Grinding1	DIDataRea	PLC1.Application.PLC_PRG3.DIDataReady
Grinding1	DIDoorCell	PLC1.Application.PLC_PRG3.DIDoorCell1Open
Grinding1	DISimAlarm	PLC1.Application.PLC_PRG3.DISimAlarm
Grinding1	DIArticleNo	PLC1.Application.PLC_PRG3.DIArticleNo

Figure 21: I/O mapping using Excel sheet.

5.3.3 Verification and Validation

The signal handling and functionality of low level components was verified before they were included in Experior's library by debugging the scripts and using small test PLC-programs. Figure 22 shows the verification procedure of the honing machine PLC signals where triggered and logics could be verified. In addition, animation and movements of the machine was also studied and reconfigured to work accordance to the specification.



```
Input added: Honing1+DOLoadMission - PLC1.Application.PLC_PRG3.DOLoadMission
Input added: Honing1+DOCellEntNotOK - PLC1.Application.PLC_PRG3.DOCellEntNotOK
```

Device.Application.PLC_PRG3

Expression	Type	Value	Prepare
DOAuto	BOOL	TRUE	
DOAlarm	BOOL	FALSE	
DOUnloadMission	BOOL	FALSE	
DOLoadMission	BOOL	TRUE	
DOArticleNo	DINT	0	
DOProgramNo	DINT	0	
DOCellEntNotOK	BOOL	TRUE	
DOLifeBit	BOOL	FALSE	
DODataReady	BOOL	FALSE	
DIAuto	BOOL	FALSE	
DIArticleNo	DINT	0	
DIProgramNo	DINT	0	
DICycleStart	BOOL	FALSE	
DIResetAlarm	BOOL	FALSE	
DICycleStop	BOOL	FALSE	
DIRequestCell	BOOL	FALSE	
DILifeBit	BOOL	FALSE	
DIResetUnloading	BOOL	FALSE	
DIResetLoading	BOOL	FALSE	
DIDataReady	BOOL	FALSE	
DIDoorCell1Open	BOOL	FALSE	
DISimAlarm	BOOL	FALSE	

Messages - Total 0 error(s), 70 warning(s), 5 message(s)

Description	Project	Object
Size of global data: 45150 bytes		
Total allocated memory size for ...		
Memory area 0 contains Data, ...		

Figure 22: Low level verification with signals, logics, and animation.

The verification of the layout and the positioning of components was made by jogging the robot to the specified robot position targets. The accuracy of the high level modelling using 2D drawing was not as high as required. Several adjustments were needed in order for the robot to place the loads at the right location of each component. Partly because the physics engine included some unexpected events that needed to be handled. In Figure 23, an illustration is shown where the components are adjusted with respect of the robot position targets.

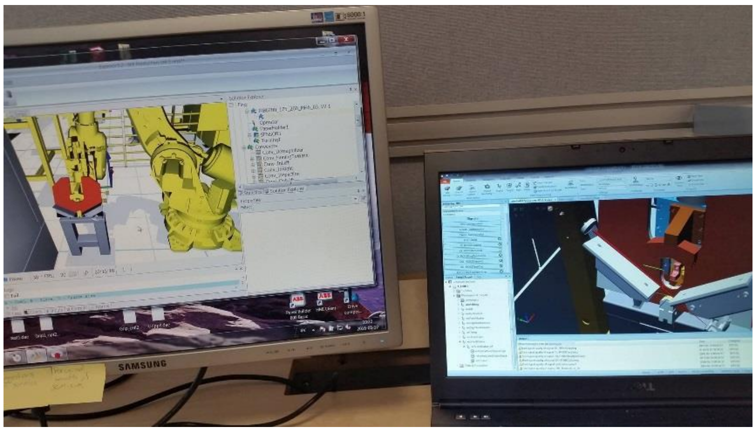


Figure 23: Components are adjusted with respect of the robot position targets.

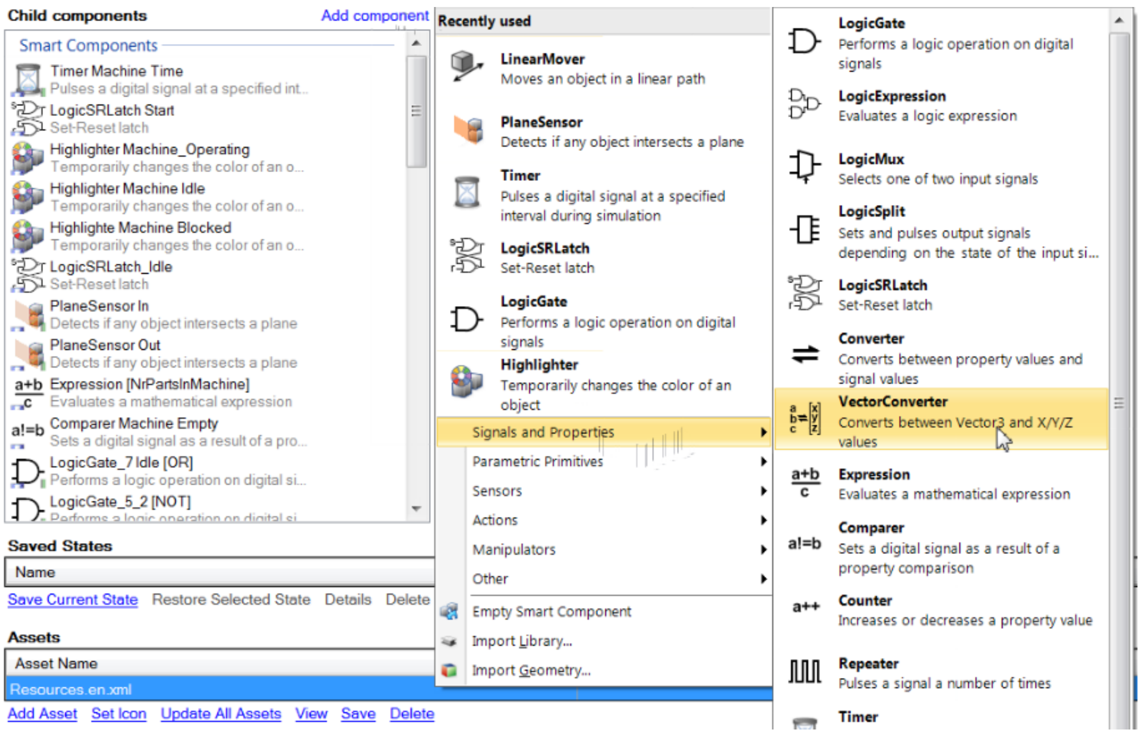
The parts included in the physics engine was verified by testing them in a high level environment. By putting a the specified part on e.g. a conveyor, the behavior could be analyzed when it for example gets stuck on a side wall or falls down to the ground (animation verification technique).

5.4 Emulation Model Building (RobotStudio)

This chapter describes the proceedings of Low level modelling and High level modelling that was used to create the virtual environment that represented the production cell at SKF. The chapter focuses on the emulation model building in RobotStudio.

5.4.1 Low level modelling

RobotStudio used an interface based application called Smart Components that enabled the creation of low level components. By combining different functions in the component (called child component) such as logical gates (AND, OR, NOT etc.), counters, timers, sensors, manipulators, queues (lists) etc. customized components could be created. The logic was used to represent the logical behavior (control system) of the real machines meanwhile the manipulators was used to create motions for mechanical parts. An example of a motion was the movement of parts on a conveyor. Figure 24 shows the library where all child components is collected and can be inserted to the Smart Component.


Child components Add component

Smart Components
- Timer Machine Time — Pulses a digital signal at a specified int...
- LogicSRLatch Start — Set-Reset latch
- Highlighter Machine_Operating — Temporarily changes the color of an o...
- Highlighter Machine Idle — Temporarily changes the color of an o...
- Highlighte Machine Blocked — Temporarily changes the color of an o...
- LogicSRLatch_Idle — Set-Reset latch
- PlaneSensor In — Detects if any object intersects a plane
- PlaneSensor Out — Detects if any object intersects a plane
- Expression [NrPartsInMachine] — Evaluates a mathematical expression
- Comparer Machine Empty — Sets a digital signal as a result of a pro...
- LogicGate_7 Idle [OR] — Performs a logic operation on digital si...
- LogicGate_5_2 [NOT] — Performs a logic operation on digital si

Saved States

Name

Save Current State Restore Selected State Details Delete

Assets

Asset Name

Resources.en.xml

Add Asset Set Icon Update All Assets View Save Delete

Recently used
- **LinearMover** — Moves an object in a linear path
- **PlaneSensor** — Detects if any object intersects a plane
- **Timer** — Pulses a digital signal at a specified interval during simulation
- **LogicSRLatch** — Set-Reset latch
- **LogicGate** — Performs a logic operation on digital signals
- **Highlighter** — Temporarily changes the color of an object

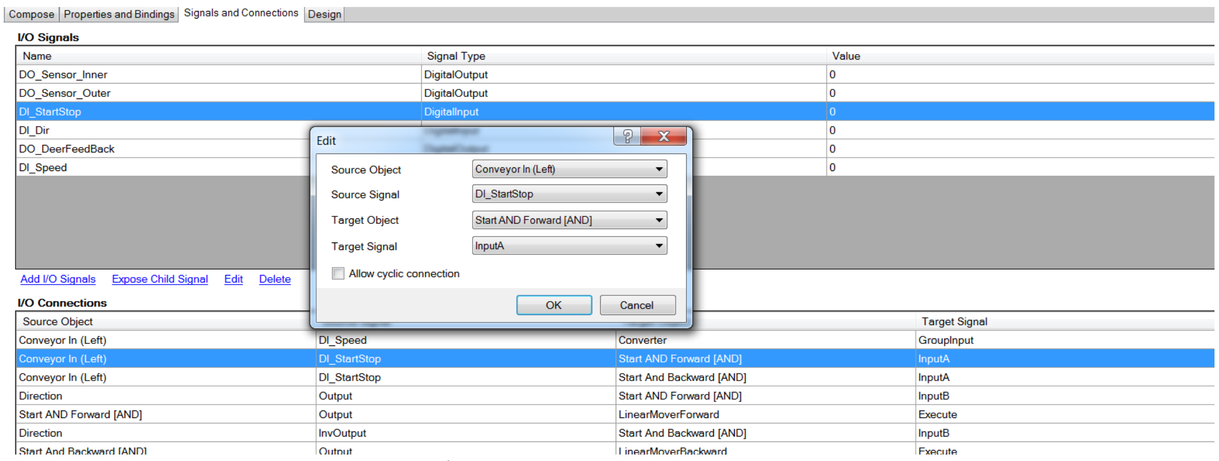
Signals and Properties
Parametric Primitives
Sensors
Actions
Manipulators
Other
Empty Smart Component
Import Library...
Import Geometry...

- **LogicGate** — Performs a logic operation on digital signals
- **LogicExpression** — Evaluates a logic expression
- **LogicMux** — Selects one of two input signals
- **LogicSplit** — Sets and pulses output signals depending on the state of the input si...
- **LogicSRLatch** — Set-Reset latch
- **Converter** — Converts between property values and signal values
- **VectorConverter** — Converts between Vector3 and X/Y/Z values
- **Expression** — Evaluates a mathematical expression
- **Comparer** — Sets a digital signal as a result of a property comparison
- **Counter** — Increases or decreases a property value
- **Repeater** — Pulses a signal a number of times
- **Timer**

Figure 24: Interface of Smart Component creation.

In able to interact with the components, creation of I/O signals was done that depending on the logic inside the Smart Components affected the components in different ways. By adding dynamic properties to the Smart Component, variables could be defined that was used for numerical values, controlling for example speeds of movements or timer values. As the child components could have relations to both I/O-signals, dynamic properties and other child components, they needed to be connected to each other. This could be done in two ways.

The first one is represented in Figure 25 where the digital Input DI_StartStop (source signal) was connected to a logical AND gate (target signal). If later the AND gate becomes true, the gate will set its output signal to true.


Compose | Properties and Bindings | Signals and Connections | Design

I/O Signals

Name	Signal Type	Value
DO_Sensor_Inner	DigitalOutput	0
DO_Sensor_Outer	DigitalOutput	0
DI_StartStop	DigitalInput	0
DI_Dir	[illegible]	0
DO_DeerFeedBack	[illegible]	0
DI_Speed	[illegible]	0

Add I/O Signals Expose Child Signal Edit Delete

Edit
- Source Object: Conveyor In (Left)
- Source Signal: DI_StartStop
- Target Object: Start AND Forward [AND]
- Target Signal: InputA
- Allow cyclic connection

OK Cancel

I/O Connections

Source Object	[illegible]	[illegible]	Target Signal
Conveyor In (Left)	DI_Speed	Converter	GroupInput
Conveyor In (Left)	DI_StartStop	Start AND Forward [AND]	InputA
Conveyor In (Left)	DI_StartStop	Start And Backward [AND]	InputA
Direction	Output	Start AND Forward [AND]	InputB
Start AND Forward [AND]	Output	LinearMoverForward	Execute
Direction	InvOutput	Start And Backward [AND]	InputB
Start And Backward [AND]	Output	LinearMoverBackward	Execute

Figure 25: Connecting I/O signals, relations and child components in RobotStudio.

The second approach was using the block diagram function as can be seen in Figure 26. Figure 26 shows the logic created for controlling of a conveyor belt. Here the different child components are connected by dragging arrows between the blocks. As can be seen in Figure 26, each predefined child component had its own input, output and properties that could be used when connecting all the different functions.

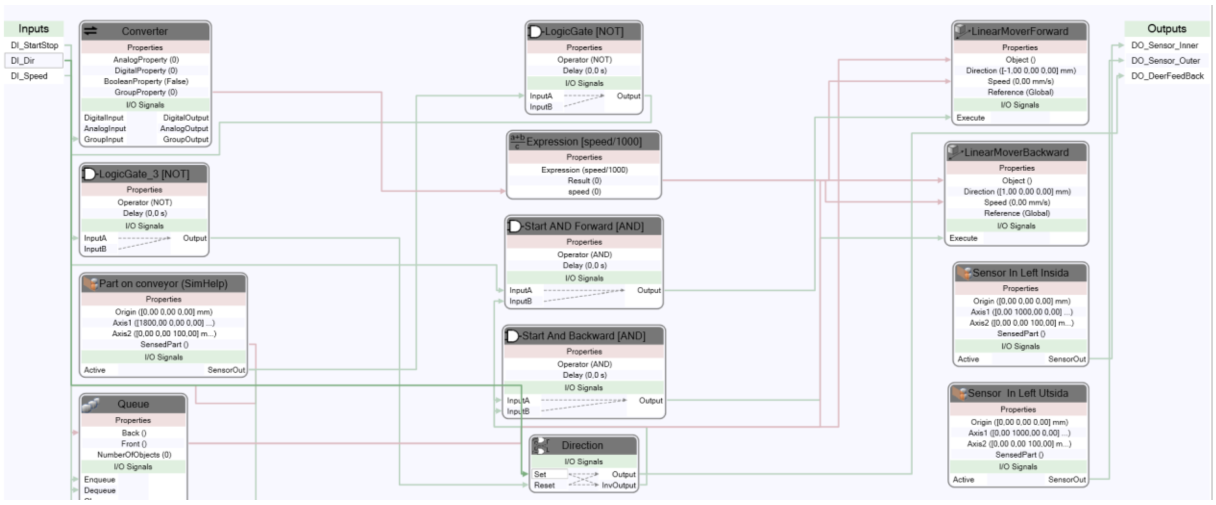


Figure 26: Connecting I/O signals, relations and child components with block diagrams in RobotStudio.

In order to give the reader an understanding of the process of creating a Smart Component follows an example where a gripping tool is modeled. The tool shown in this examples is not the final version of the tool in the SKF-project. The gripping tool consists of two grippers and is controlled by servo drives which opens and closes the grippers. To know when a part has been gripped the "momentum stop" of the servo is used. An increase of the motor momentum triggers a signal and is interpreted as a part is gripped.

Step 1: CAD & Kinematics

The first step was to insert the CAD-geometry of the gripping tool. Since the models provided by the construction department at ÅF-Laxå was detailed they were too heavy to use in a emulation. Therefore they needed to be simplified. RobotStudio provides a built in features that could simplify the geometry of the model.

The next step was to separate the moving parts from the non-moving parts. This was done by selecting all solids of the moving gripping claw and merge it as one part (Figure 27). Figure 28 shows the next step where a mechanism was created as a tool and the different links and the fundament of the tool was defined. Thereafter the joint was configured. Joint type was chosen as prismatic, movement directions was defined as well as the joint limits for how long the gripper could be moved.

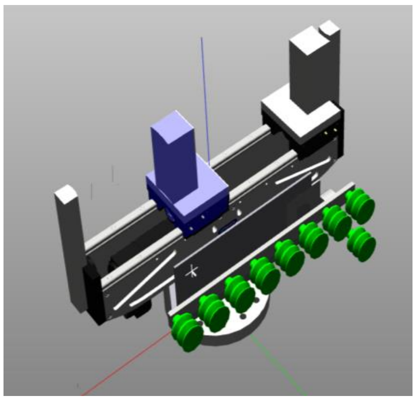
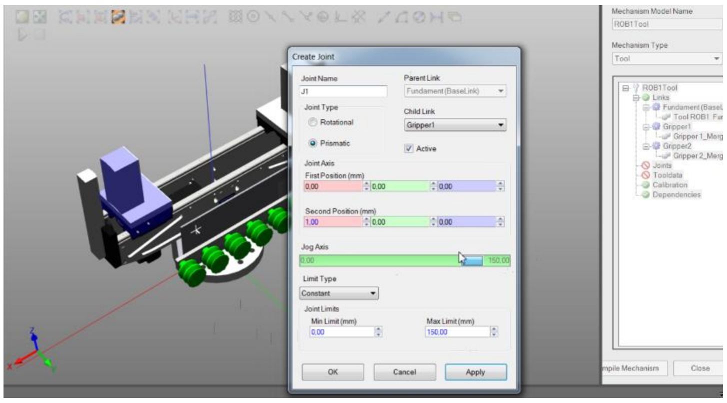


Figure 27: Separating CAD-geometry.


Create Joint

Joint Name: J1

Parent Link: Fundament (BaseLink)

Joint Type: Rotational [✓ Prismatic]

Child Link: Gripper1

[✓] Active

Joint Axis

First Position (mm): 0,00 | 0,00 | 0,00

Second Position (mm): 1,00 | 0,00 | 0,00

Jog Axis: 0,00 – 150,00

Limit Type: Constant

Joint Limits

Min Limit (mm): 0,00

Max Limit (mm): 150,00

Mechanism Model Name: ROB1Tool

Mechanism Type: Tool

```
ROB1Tool
  Links
    Fundament (BaseL
      Tool ROB1_Fu
    Gripper1
      Gripper 1_Merg
    Gripper2
      Gripper 2_Merg
  Joints
  Tooldata
  Calibration
  Dependencies
```

Figure 28: Configuration of joint movements.

Step 2: TCP-definition

As the Smart Component was going to be used as a tool for the robot there was a need to define the different Tool Center Points (TCP) of the grippers and in this case also the vacuum.

Step 3: Creations of I/O signals

In able for the PLC or the robot control system to interact with the Smart Component there was a need to create I/O signals. For this tool, digital inputs in form of Open/Close gripper were defined. As the model only considered one size of the ring, the position of the open and close state was configured directly in the Smart Component. However, the logic could be change to act upon a numeric position input from the PLC instead of a Boolean for open/close the gripper. As it comes to the outputs the Smart Component was equipped with an output signal to notify when a part was gripped (this function is explained in step 4).

Step 4: Placement of Sensors & Logics

As the tool should be able to pick up parts there was a need for sensors that could sense which CAD-geometry that should be attach to the tool. Attachment of a part means that the CAD-geometry will follow another geometry of the model to which it is attached (in this case the tool) until the part is detached. One sensor was attached to the moving gripping claw (yellow plane in Figure 29A. Since it was attached to the moving gripping part it followed every movement it did. When the signal "Close grip" became true, the moving gripping claw started to move towards the ring until the state as can be seen in Figure 29B. In this state the sensor sensed the ring and the logic within the Smart Component was configured to attach the ring to the moving claw if the sensor sensed any object while the gripper was closing. Figure 30 shows a part of the tools logic where the attachment takes place. The gripper continued to move but now with the ring attached and “dragged” the ring towards the non-moving gripping claw to the right of the tool. At the non-moving gripping claw was another sensor located. When that sensor also sensed the ring (Figure 29C) the logic was configured to set the output “material in gripper” to true and the PLC-program could then stop the gripping motion. This solution represented the momentum stop of the servo motors. In this state the ring was attached to the tool and could be moved around in the cell as can be seen in Figure 29D.

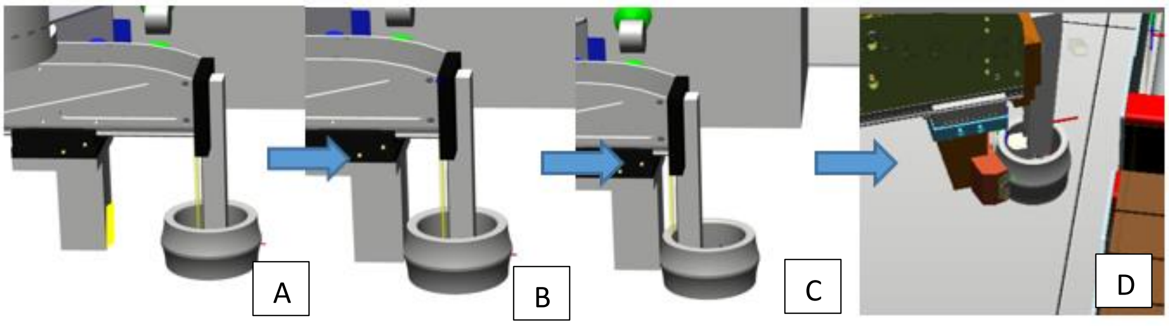


Figure 29: The figure sequence shows how a gripping operation is performed.

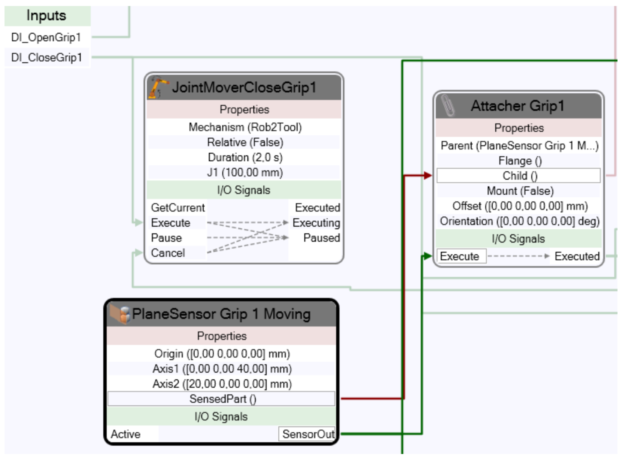


Figure 30: Part of the logic where a part is attached to the gripper.

5.4.2 High level modelling

When a low level component was done it could be inserted in the high level model. By dragging or defining exact position in the world coordinate system, the component could be positioned according to the layout. As the 3D-models from ÅF Laxå came as a RobotStudio file, the models were already in place. When a Smart Component had been added to the high level model its I/O signals had do me mapped to the correct I/O signals in the PLC program. In Codesys, all I/O signals that had relations to the model had to be added in a symbol file. This symbol file was then available through the OPC-client in RobotStudio. The OPC client had to be configured by selecting which I/O signals that should be available in the model and definition of the I/O types were also needed. A Boolean input was defined as digital input in RobotStudio meanwhile an analog or an INT variable was consider as a Digital Group Input.

When the I/O signals had been added to the client they had to be mapped to the right Smart Component. This was done through the Station logic function in RobotStudio.

5.4.3 Verification and Validation

When the low level component was considered done the functions and logic of the component was tested by trigging I/O signals directly in RobotStudio and ensuring that the logic did as intended. The model could also be verified by connecting the components to the PLC program and trigger I/O signals. Figure 31 shows a small test program for the gripping procedure created in Codesys.

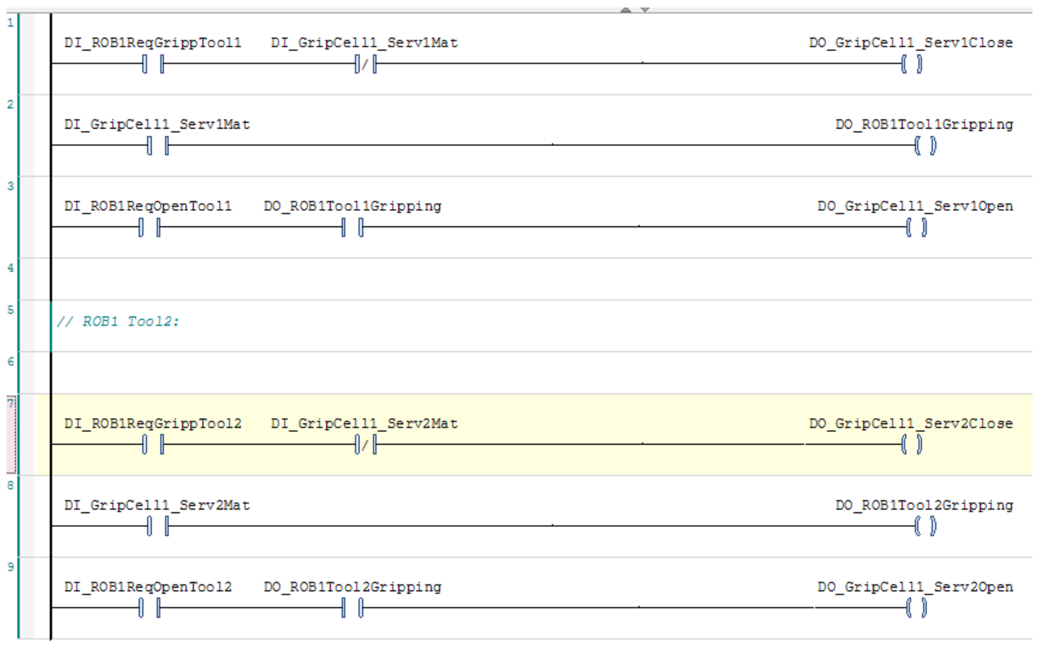


Figure 31: PLC-logic for the gipping tool.

After the verification the Smart Component was saved into the component library of RobotStudio.

The verification of the high level model in RobotStudio was done similar as the way in Experior described in section 5.3.3. The robot was moved to different machines and positions of sensors and positioners of outgoing rings from machines were verified and optimized. Sensor positions was for many machines unknown and had to be positioned after the robot target positions. In order to verify the model in a more real sense, smaller sequences of robot and PLC was created. In this programs, all the necessary data transfers, handshakes and other communication was included.

5.5 Testing emulation model

Until this phase, only fragments and parts of the emulation model had been tested and verified. The purpose of this phase was to test the whole model and consider it functionalities. This chapter describes and illustrates the emulation models and the required test programs.

5.5.1 Model functionality

Figure 32 illustrates an overview of the virtual plant in RobotStudio and Figure 33 illustrates an overview of the virtual plant in Experior. The functionalities were identical in the two emulation models, hence the same test programs and test procedures were similar when evaluating the model functionalities.

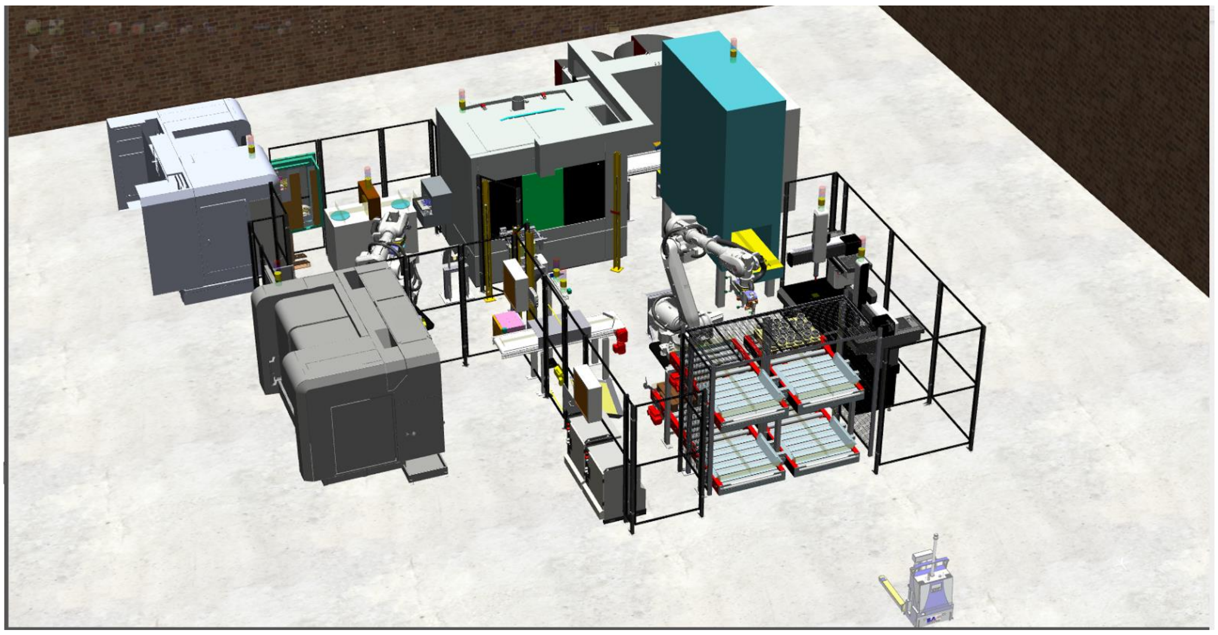


Figure 32: Virtual plant in RobotStudio.

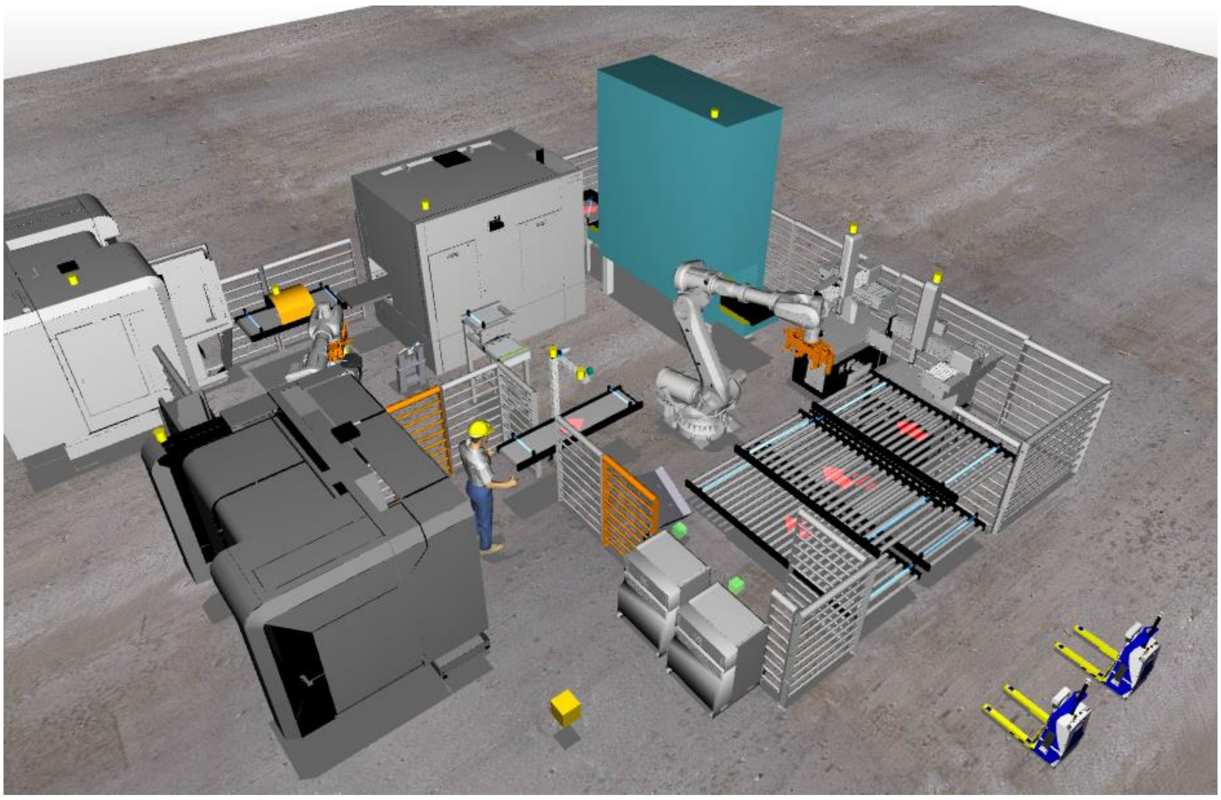


Figure 33: Virtual plant in Experior.

The AGVs were simulated and provided the cell with new materials as well as getting the processed materials from the cell. The robots were used for material handling where the gripper placed the products to the required position. Machines that were supposed to process the product and thereafter repositioning the product to an unloading location consisted of simulated movements to illustrate the functionalities. As the product were processed, it was placed at the top of the machine to illustrate that there is a product in the machine. In addition, lamps were implemented to each machine and component to visualize the current state of the machines. In Figure 34, a machine that currently has processed a product and waiting for the unload location to be empty. In this case, the machine is blocked and a blue lamp visualized. In addition, all the machines in the model constantly received and sent the required data and signals to the PLC and the machines behaved accordingly. Function that stopped or shut the machine down were also included. The conveyors in the model moved the products with a defined speed set by the PLC.

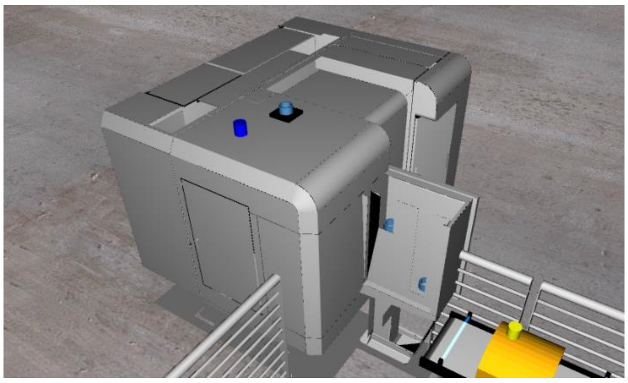


Figure 34: A machine that is currently blocked.

To be able to verify the doors to the cell, a trigger could be initiated and a simulated operator moved into the cell and opened the cell door. That triggered signals to the PLC that indicated that the cell was opened.

To further simplify verifications of functionalities and illustrate different machine states, a simple HMI was developed in Codesys. An illustration of a HMI is shown in Figure 35.

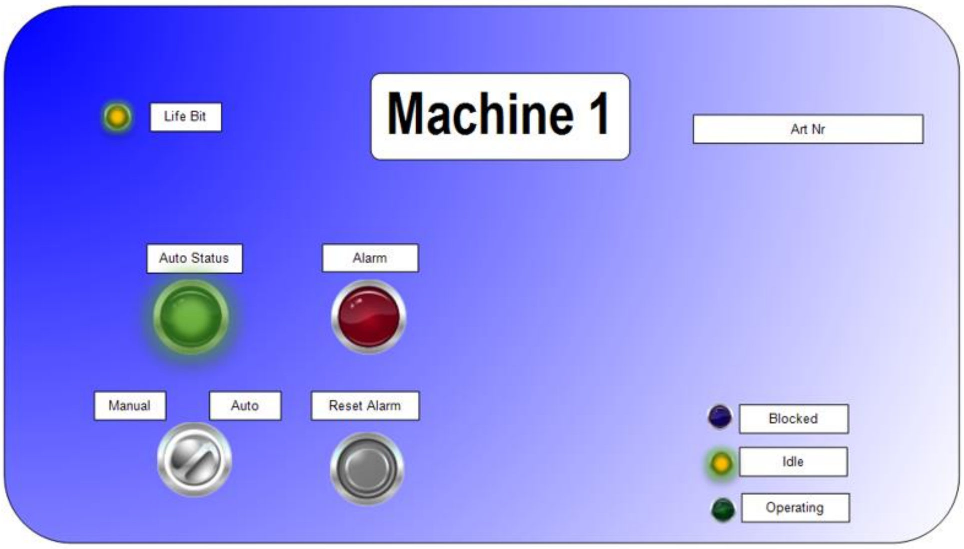


Figure 35: HMI for testing the model's functionalities show different states.

5.5.2 Test programs

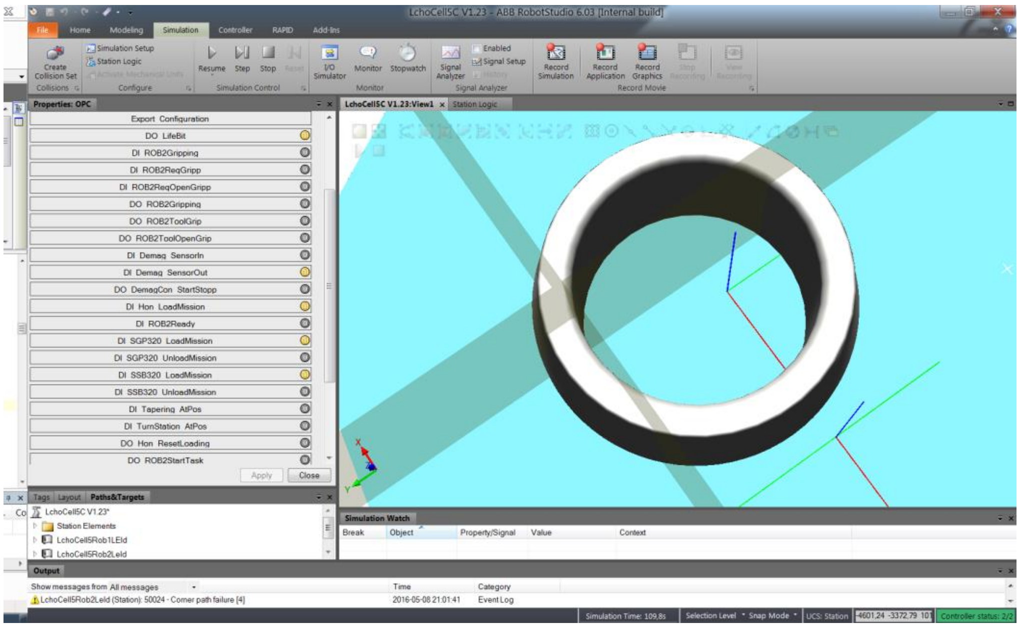
Since there were no completed programs that could be used to test the emulation environment, small PLC and robot programs were developed by the thesis authors. These programs were used to test the emulation model and discover additional errors in the model. During the testing, several unexpected events and errors occurred during the commissioning. Some of the adjustments that were needed are stated in the list below. Additional, more technical problems and suggestions for improvements can be found in Appendix C and Appendix D.

Experior:

- The gripper sometimes dropped the products. The cause was that the physical property “kinematic” was set to true which resulted in unpredictable behavior of the gripper and product.
- Lagging problems in simulation environment. The CAD files were too large and complex and needed additional optimization.

RobotStudio:

- Some problems that was not found during the verification of the low level components were unveiled after repetitive testing of the model.
- The reset function was shown unreliable. Sometime variables from an old simulation was remaining even if the reset function was used. This concerned both variables in the robot controller and variables in Smart Components (especially queues).
- When both robot systems and all Smart Components were run at the same time, the memory usage of the RobotStudio application was at its maximum limit. This made the model both slow and jagged and was also effecting the simulation clock. The 64-bit application of RobotStudio was during the project shown to be the better choice when running big models but was not able to use since the OPC-application did not work in the 64-bit version.
- During the verification with the test program it was discovered that there was some delay in the data exchange from the PLC to Robotstudio. Figure 36 shows the effects of this delay. From the moment that the sensor in RobotStudio sensed a part on a conveyory, to the moment the PLC stopped the conveyor, the ring had time to move a little bit further. This resulted in that the ring was positioned wrong and affected the pick-up procedure done by the robot. As the communication protocol was considered hard to fix in relation to available time, the solution was to change the position of the sensors so that the ring was position right in relation to the robot position target.



```
LchoCell5Rob2Leld (Station): 50024 - Corner path failure (4)    2016-05-08 21:01:41    Event Log
```

Figure 36: Effects of delay in the data exchange between the PLC and RobotStudio

The PLC programs were developed in Codesys where a simplification of one production cycle was programmed to verify that the model behaved as expected during one production cycle. For additional functionalities such as doors, alarms, data transfer, and AGV triggers, complemented PLC programs and functions were programmed. Figure 37 shows an extraction of the PLC programs for the machines.

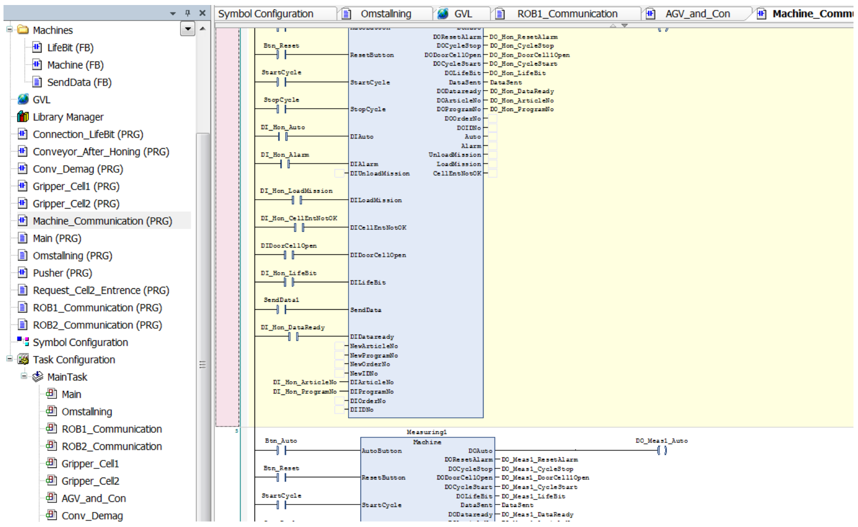
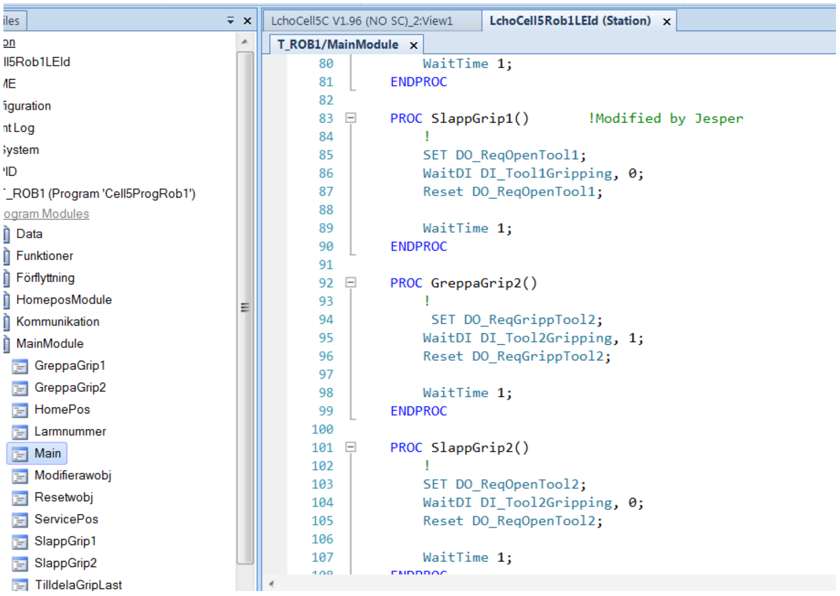


Figure 37: Extraction of the PLC program used for testing.

In addition, a robot program was created in RobotStudio from the routines and robot target positions that the robot programmers had completed. The routines developed by the robot programmers were adjusted so the model could complete one production cycle. Some routines, such as handling various product sizes, were excluded and simplified. Furthermore, robot target positions that were not up to date were adjusted to fit layout of the model. Routines and robot target positions that had not yet been developed were also created by the thesis authors. A screenshot from the robot program can be seen in Figure 38.



```
        WaitTime 1;
    ENDPROC

    PROC SlappGrip1()       !Modified by Jesper
        !
        SET DO_ReqOpenTool1;
        WaitDI DI_Tool1Gripping, 0;
        Reset DO_ReqOpenTool1;

        WaitTime 1;
    ENDPROC

    PROC GreppaGrip2()
        !
         SET DO_ReqGrippTool2;
        WaitDI DI_Tool2Gripping, 1;
        Reset DO_ReqGrippTool2;

        WaitTime 1;
    ENDPROC

    PROC SlappGrip2()
        !
        SET DO_ReqOpenTool2;
        WaitDI DI_Tool2Gripping, 0;
        Reset DO_ReqOpenTool2;

        WaitTime 1;
```

Figure 38: Extraction of the robot program used for testing.

6 Discussion

From the benchmarking studies and the statements of Schludermann et al. (2000) in the literature framework, it could be concluded that the most common way of verifying and validate PLC and robot programs today is to build up a physical representation of the real system (Mock-up). Then programmers can test their programs and it could be proven to the customer that the concept works. However, building up a Mock-up could be expensive and hard to design to correspond to the real system according to Schludermann et al. (2000). In addition, the safety issues around the Mock-up is usually not fully taken into consideration so validation and verification could be dangerous for humans. Hence, a Mock-up should be depreciated and replaced by VC as the technology matures. This chapter discusses the research questions as well as problems that accrued during the project.

6.1 Methodology review

The methodology developed for this master thesis has been working well. The ability to work with the low and high level modelling integrated has turned out to be a successful way of working. By not dividing the work in a sequential flow where all low level components need to be completed before starting the high level modelling, it allows for more freedom and the possibility to work with the data and specifications available at the moment.

Since the creation of VC modelling started as soon as the contract with SKF was signed, electrical engineers, programmers and mechanical engineers worked concurrently. The layout was continuously updated and no defined specifications were available at the start. That put high requirements of interacting with disciplines to access each other's data. However, it could be discussed whether the creation of VC modelling should start before electrical engineers have defined the I/O lists and mechanical engineers are done with e.g. CAD models. By starting the creation of the VC modelling at the same time as electrical and mechanical engineers, it also results in rework and additional work when adding new CAD models and I/O signals to the VC logics and environment. As discussed in the benchmarking with KUKA, they did not start the creation of the VC model before the electrical engineers and mechanical engineers were finished in order to reduce unnecessary rework. In addition to the benchmarking, the literature framework indicated that most of the prior work methodologies suggested that I/O signals and geometries should be defined before starting the VC modelling. Though, starting with the VC model concurrently with electrical and mechanical engineers enables the VC model to be finished earlier in the project. In addition, disciplines could take advantage of each other by exchanging data which allows for detecting problems in an early phase. However, this requires additional development of integrating applications and different disciplines.

Furthermore, since the VC applications was developed in parallel with the model building, the software communication was not able to be established in the phase according to the methodology. ABB's concept of integrating their software with each other was not finished in time. This required temporary solutions during the project to enable PLC communications and software integrations. Experior also had problem of handling the updated versions of RobotStudio. Though, the developers at Xcelgo were quick of coming with Experior updates that handled the new versions of RobotStudio. However, the optimal situation should be to establish the software communications and integration between software in an earlier phase before starting the VC modelling. That allows to focus more on the VC modelling instead of software establishment during the project. In addition, verifications of the model could be done continuously during the project using the connections to e.g. softPLC and robot controller.

One important factor to discuss is the reliability of the model, especially how reliable the connection between the simulation software and the PLC is. During the testing of the emulation model, it was noticed that there were minor delays for the data exchange. One example is the delay shown in Figure 36. Delay in relation to stopping parts at a sensor might not directly be a problem since the sensor positions still needs to be optimized at the commissioning. However, the delays in the emulation model proofs that delays exists and can cause problems since the delay might create differences between the VC model and the real production system. The delay also result in that sensor positions needs to be optimized in the model so that positions of products lies at correct positions when the robot will pick the part up.

The cause of the delay most likely depend on the OPC-solution used for connecting the PLC to the emulation model since OPC has known issues regarding speed and synchronizations and is confirmed by Carlsson et al. (2012). But other factors like how the emulation software are processing the signals might also be a contributing factor. In the process of making VC reliable there is a need for making the data exchange more reliable. Some software has special made protocols for communication with PLC where one example is Experior who uses customized communication. However in able to make VC a non-vendor dependable tool it is important to develop a universal standard protocol that enables a synchronization mechanism and enough speeds. Since OPC communication already follows the strategy to be non-vendor specific, the strategy of develop an OPC extension suggested by Carlsson et al. (2012) would be a suggestible in the mission to make VC more available and reliable.

As mentioned earlier, the programmers were in the process of constructing the programs concurrently to the building of the VC model. Therefore, it was needed for the thesis authors to develop PLC and robot programs in order to test and verify the functions and layout of the virtual plant. One could discuss the delegation of who should build these small PLC programs. Should it be experts in PLC programming or could the system integrators that work with the virtual plant build simplified programs used for testing the virtual plant? If experts within PLC programming build up these simplified programs, the quality might be increased compared to the system integrator as well as less time consuming. Though, the PLC programmers might not have the time to build up test programs since the highest priority is to complete the real programs that should be used for commissioning. But one important discovery of this master thesis work has been that the efficiency and quality of building the test programs has increased considerably due the continuous PLC verification in the VC model. Having a VC model connected to the PLC programming software facilitates the process of building the programs. Hence, one could argue that if the experts within PLC programming take the time of building the test programs for the system integrators, the spent time on building test programs will be recaptured by having a functional VC model connected to the PLC programming software with continuous verification.

Furthermore, no complete VC were able to be initiated during this project since the programmers were not finished with the programs within the time limit of this master thesis. However, the Testing of emulation model was done so a VC using the real PLC and robot programs should be possible with some further adjustments depending on the I/O list and the concept used by the programmers. In addition, some additional adjustments and initializations might be needed to make it possible to connect the VC model to the hardware PLC that will be used in the real plant.

6.2 Comparison between VC in Experior and RobotStudio

Experior allows for more interactions between applications compared to RobotStudio and is therefore more flexible in this aspect. For example, ABB robots could be included in Experior's environment where the robot movements are reflected from RobotStudio. Hence, the concept of Experior is to act as an integrating software that focus on communication between different systems and applications. In contrast to VC in RobotStudio, ABB seams to focus on great software communication among their own applications within their own developed platform.

However, in Experior a Cartesian left handed coordinate system was used in the emulation environment where Y was up. In RobotStudio on the other hand, a Cartesian right handed system was used where Z was up. This made it confusing when integrating the two programs and including geometries from RobotStudio to Experior. The same issue occurred when importing a customized CAD model into Experior from SketchUp since the coordinate systems between SketchUp and Experior did not match. Hence, one could discuss the choice of coordinate system in an application. Since Experior is an application that is designed to interact with other applications and work as an intermediator, the coordinate system should be same as the integrating applications. Same coordinate system would facilitate the positioning of components, modelling procedures and definition of movements and kinematics.

RobotStudio has, compared to Experior, built-in optimizers that can reduce the complexity of CAD models. Therefore, RobotStudio can handle CAD models that not Experior can handle. This has been a big issue during the project since the layout and all CAD models have been provided through RobotStudio, so optimization of CAD files was required for all geometries that needed to be imported into Experior. Complications sometimes appeared depending on that some information was lost during the optimization. For example, when the gripper in Experior was created, the tree structure was erased after optimization. That resulted in complications when trying to reflect the gripper movements from RobotStudio into Experior.

Large and complex CAD models from vendors utilizes large computer power. It might not be needed to import detailed CAD models since the purpose of VC is to verify the PLC programs and handle the signals. However, it should be considered in future projects that the vendor provides two versions: one complex and one simplified. The simplified CAD model can then be used for simulation and selling purposes while the complex CAD file could be used to demonstrate details of the machine. One could further develop the idea of the vendors providing a mechatronic model to the customers with logics and signals included in the model. However, that requires a standardized format of the file to be shared so that the model could be handled by the customer's software. Thought, it would increase the speed and efficiency of creating the VC model since the customized machines requires some extra effort to include in the emulation environment.

As prior mentioned, the layout containing CAD models has been modelled in RobotStudio by the engineers working with the robot programming. Hence, additional work was needed to transfer the layout and CAD models into Experior's emulation environment. In addition, the interactions between the robot system and Experior had to be configured. In conclusion, one could discuss the efficiency of using Experior for projects with a lot of interactions with robots. One factor is that it is time consuming to build up and configure the gripper in Experior. To include gripper movements from RobotStudio and configure the layout in relation to the robot position targets, requires some extra effort. In this project, all CAD models had already been included and positioned in RobotStudio as well as the robot target positions. Instead of building a virtual plant from scratch, the already built-up virtual plant could be used and functions could be added directly to the components. That reduced the modelling time considerably in RobotStudio and might be considered to be a more optimal tool in this case. However, the effort of establish communication and integrations to other software were a bigger challenge for RobotStudio since the vision was to utilize the applications provided by the ABB's Automation Builder platform.

By utilizing RobotStudio as the emulation environment, the verifications of robot programs could be used in the same software. Collision detections are possible and the process of jogging the robot is smooth since it is integrated with the layout. Hence, the low level components could easier be tested in the virtual plant directly after implemented RobotStudio.

Compared to RobotStudio, Experior consists of a physics engine that enables components to behave according the law of physics. Physics properties such as gravity, weight, friction and kinematics are specified and affects the behavior of the product. However, this configuration requires some extra effort in order to make the parts behave as expected and the unpredictability of the model increases. It could be discussed if the unpredictable events that occurs in the model due to the physics engine could be used for verification purpose. Is the accuracy of the physics engine accurate enough to make it worth the extra effort of making it work? Of course, for a selling purpose and to increase the common perception of concepts, the physics engine might contribute great value.

Another perspective is the computer power needed to utilize the physics engine. It could be discussed whether it is worth the high utilization of processing power to include physics into the model. Maybe the computer power could be used for a better purpose than the physics engine, e.g. higher accuracy in PLC connections. But with increased accuracy of the physics engine and continued research in the field, it would be a great complement to VC.

One important factor to discuss is the maturity, more specific the stability of the simulation software. An unstable software results in bugs and software related problems which are time consuming. Using RobotStudio as a VC tool where the mechanical models are created is a fairly new concept and that bugs appear in the software is understandable. Since creating a VC model often is time consuming it is important that the software used is stable and that the company providing the software also strives for continuous improvements and listens to the customer’s suggestions for improvements.

It was decided that the interface in RobotStudio was going to be used for making the Smart Components in this project. The interface was user-friendly and easy to use when creating components with simple logics. However, as the components became more advanced the interface became unstructured and debugging of the logic was hard to perform. This resulted in longer debugging times and an unstructured visualization. RobotStudio supports Smart Components to be written in C# using Visual Studio. This would allow both debugging and more freedom to create customized functions and logic. Therefore it could be argued that this strategy should be tested when creating more advanced components in future projects.

6.3 Increasing the efficiency of VC modelling

It has been a challenge to gather all the required information since the information has been spread and stored at different parties in the project. New decisions were taken throughout the project as the specifications became more detailed, but they were not directly communicated to all parties within the project. To avoid misunderstanding of concepts and encourage a unified perception, new decisions and updates should be directly communicated to all involved parties. One way of facilitate the communication and sharing of files is a centralized database where e.g. CAD files (compatible and optimized), new project information and decisions, contact information, specifications of machines and logics, I/O lists, layout with positions of every component and robot target positions could be shared. This has also been discussed in the benchmarking where Kåre Folkeson emphasize that the technique of sharing data is already available on the market. So the next step should be to integrate that technique into the concept of VC. However, as stated in the literature framework, it is important to consider the format of the files used in the database. If the file formats are not standardized, there could be complications of converting to other formats so that the software can handle the files.

To further develop this concept, one could argue that the files should not only be accessible from a shared database. There should also be an integration to the software so that an update in one software is automatically updated in the other software without any manual effort. For example, if a machine is moved in the layout in Experior, the robot target positions in RobotStudio should automatically be adjusted. Another integration that could increase the efficiency is a connection between the software used of the electrical construction department and the VC model. During electrical construction, I/O signals are created and changes. I/O signals are also a major part in VC modeling and a connection that could automatically change I/O signals in the VC-model if I/O signals are change in the electrical construction software would result in time savings and a more efficient process. In addition, a VC model includes a lot of data which creates opportunities of using the model in other ways than verification purposes. The Digital Factory concept include many virtual engineering tools that can interact and aggrandize each other. A VC model should contribute well to the concept of the Digital Factory and maybe can data needed in electrical construction be generated from the VC model.

To encourage parallel work when building the VC model, one could divide the work into different disciplines. For example, one engineer could work with functional and logical properties, another department focus on the geometrical properties with e.g. CAD. Furthermore, signals and electrical properties should also be defined and could be done by another department or engineer. The animations and kinematics in the VC model is an area that also takes time to complete and could be done by a separate engineer. The concept of dividing the work into different disciplines is also discussed in the literature framework where it has shown to increase the speed and efficiency of the VC modelling. However, the system integration of these disciplines needs to be considered and well developed with software and file formats that enables integration between each other.

As discussed in the literature framework, it takes time to build up knowledge and a library with components that can be reused. The first VC projects might not become profitable for a company since a lot of components need to be modelled and added into the library in order to create a VC model. But with time, the library expands and knowledge of procedures increases which lead to less effort of building the VC model hence become more profitable. Therefore, the question of increasing the efficiency in building the VC model would be relevant in this context. Standardization of procedures and software connections should be a foundation when building up knowledge about VC. By using standardized components that exist in the library, the time of building the VC model decrease since the low level modelling is no longer needed to a high extent.

When building up a library consisting of mechatronic models, there should also be a focus on building flexibility into the models. By having a parametrized model, it could easily be adjusted and reused for other projects without the need of building a new component from scratch. Therefore, the flexibility aspect should not be forgotten when building the mechatronic models and include them in the library.

In the literature framework, some approaches of reducing the manual effort when building the VC model is discussed. This might reduce the modelling effort drastically. In addition, after have built the macro that auto generated the required code for Experior in this project, the manual effort reduced considerably. Of course, it took time to initiate the macro code but when that was done it facilitate the handling of new updates and building new customized components.

An important factor for building a successful VC has turned out to be communication to the software developers. A continuous communication to the developers at ABB and Xcelgo has facilitated bug problems and reduced the modelling time. In addition, the developers has provided programming strategies and tips when building the model which increased the modelling efficiency.

During this master´s thesis, all the control systems of the machines in the cells was created for enabling communication between the PLC and the machine. This control systems was then simplified so only signals and logic that would affect the PLC- and robot communication was included. One question that effects the time creating the VC-models is how simplified these control system should be. Simpler models require less time but how does a simplification affects the reliability of a VC-model and how much more value can a more complex representation of the control system give to the model? In this project the level of complexity of the control system is not as complex as the real control system but is neither in is simplest from. In able to know the answer of this question further research is required and a suggestion to this is explained in the section Future work.

7 Conclusion

What work methods and structure should be used for creating a VC model in a concurrent environment?

The methodology used for this master thesis has been working well and should be used and further developed for future VC projects. Working concurrently with the VC model as electrical engineers, mechanical engineers and programmers experienced to be a challenge though. Continuous updates resulted in rework and access of data was a challenge. However, working concurrently enables the VC model to be finished earlier and different disciplines can exchange information and detect errors in an early phase. But this requires additional development of integrating applications and different disciplines.

By allow work with low and high level modelling integrated, the currently available data could be used for creating components and verifying the layout constantly. The importance of an early establishment of software communication and architecture should not underestimated since complications appeared in this project when not being able of that establishment in an early phase. Standardized data exchange and a standardized architecture should be strived for. Considerations should also be taken to which discipline that construct the test programs for verifying the VC model. PLC programmers should take the time to develop the programs to increase quality and efficiency.

To encourage parallel work with the VC modelling, electrical, geometrical, logical, and animation properties should be divided into different disciplines. Though, this put requirements of a smooth system integration of the disciplines with supportive file formats.

What are the main difference when modelling a VC in Experior and RobotStudio?

Experior allows for interaction between different applications compared to RobotStudio. Though, the coordinate system in Experior could be confusing since other integrating applications use a twisted coordinate system. RobotStudio consists of a user-friendly interface when creating simplified mechatronic components. However, for complicated mechatronic components the interface becomes unstructured and hard to debug. Structuring complex logics and mechatronic components are done more systematically using Experior’s concept.

Another main difference is that Experior enables the use of a physics engine and allows more realistic animation and the movement of mechanical parts. However, having a physics engine also increase the workload, complexity and computer power needed to create and run the emulation.

Compared to Experior, RobotStudio supports collision detection with the robot and also direct verification of the robot program. In addition, RobotStudio consists of built-in optimizers which enable more complicated geometries without any external optimizer where potential data is lost. However, considerations should be taken to which level of detail and complexity of CAD models that are necessary in a VC model.

How to increase the efficiency and speed of VC modelling to make it more beneficial?

In RobotStudio, the layout was already predefined which facilitated the process of the VC modeling in RobotStudio compared to Experior. RobotStudio was also the platform for the robot virtual controller. The conclusion of this is to have all information and functions in one platform because it facilitates the creation of the VC modeling and the process when working in a concurrent environment.

To increase the efficiency of VC modelling, auto generating should be considered. By auto generate code or electrical properties, the manual effort is reduced which decrease the time of VC modelling. Hence, different disciplines should somehow integrate and auto generate their solutions to the VC model. In addition to auto generation, a close relationship to the software developer is important to increase modelling efficiency. The software developer can provide quick solutions to bug problems and provide tips and strategies of how to solve specific issues that occurs.

Since gathering of data and information has been time consuming, the importance of having a centralized database should be emphasized. The data included in this database should then be compatible to the concerned applications. As the library extents with more mechatronic models, next VC projects can utilize the components in the library and therefore reduce increase the efficiency. In addition, the components should be flexible with a parameterized approach to allow component adjustments in future projects. A further development to the library approach would be to integrate the applications into the database so that components are linked to each other where updates in one software automatically are generated in other software.

However, to be able to build up the knowledge and library required to be profitable, companies have to have the courage to invest in the concept and build up the knowledge of the concept. As stated by Andreas Buhlin at ÅF: "*The question is not if Virtual Commissioning will be frequently used in Industry, the question is when companies dare to invest in the technology*".

One factor that affects the time needed in able to create a VC model is how simplified a mechanical components control system should be in relation to the real control system. In this project only signals and logic that would affect the PLC- and robot communication was included in the machines together with some additional functions to simulate for example alarms. In able to answer this question further research is required and a suggestion is explained in the section Future work.

8 Future work

During this thesis work a number of interesting topics within VC has arisen that due to the time limit and the delimitation not been included. Therefore this chapter will give some suggestions of future work that can further improve the VC-concept.

Combining Virtual Reality and Virtual Commissioning

A complement to VC could be to add Virtual Reality to the emulation environment. As discussed in the literature framework, Virtual Reality is a powerful tool for testing new concepts and for verifications. Hence, adding Virtual Reality to VC could add great value and contribute to VC becoming a more powerful tool for verifications. In addition, training of operators could be done in a Virtual Reality environment which might decrease the ramp-up time in production as well as include the operators' ideas in an early phase.

Code Behind in RobotStudio

During this thesis only the interface of RobotStudio was used to create mechanical components, a method that works well for simpler logic but not when the components became too complex. Therefore a suggestion to a future work is to investigate the possibilities and challenges of creating this components using C# script, a concept RobotStudio refer to as "Code Behind".

Extension of the OPC-communication

One of the findings of this thesis was that using an OPC-connection between the PLC and the emulation software can have negative effects of the data exchange and can result in an unreliable model. There exists vendor specific protocols for more reliable data exchange, but in able to make VC more available and non-vendor specific, an extension of the today existing OPC-protocol might be preferable. A future work would therefore be to investigate the downside of using OPC for VC and what that is needed in an eventual extension of the OPC-protocol.

Real Virtual Controllers

During this project, virtual controllers have been simplified and only signal and logic needed for PLC communication have been added. As mentioned in the discussion there is a question of how complex the virtual controllers should be and how it effects the VC-model. An interesting future work would therefore be to investigate if vendors of machines also have developed virtual controllers of their real control system. If this is the case, it would be interesting to develop a VC-model including a number of this virtual controllers and compare the reliability and usefulness compared to a model with simplified virtual controllers.

Further development in Experior

As the robot tool was expected complex to integrate in Experior's environment because of the different coordinate systems and loosing necessary data after CAD optimization, this area is something to look into. A more user-friendly strategy should be developed to allow a smooth and accurate integration with the gripper. In addition, the area of CAD optimization should be considered as well. The environment should be able to handle more complex CAD models and allow for optimization features within Experior's environment that does not exclude necessary data that is needed for Experior's functionalities.

Centralized database

One big challenge during this thesis was the handling of data. The data was constantly changed and the information came from different stakeholders. A necessary future work would be to investigate the possibilities of using a centralized database together with either Experior or Robotstudio.

Standardized format for mechatronic model

The creation of the mechatronic models (Low Level Components) was shown time consuming. One idea that arose was therefore to letting the machine to vendors provide a mechatronic models to the customers with logics and signals included in the model arose. This because they already have the knowledge of the system and can make it a business. This would however require a standardized file-format that could be handled of different applications. One future work could therefore be to investigate how this file-format could be developed and how it could be used in different applications.

Auto generation and Virtual Commissioning

One aspect of this thesis has been to investigate what that is needed to increase the efficiency for the creation of a VC-model. Another aspect that has been discovered during the project is if the VC-model itself can be used generate data to other instances working with the development of automation system. One proposal for a future work would be to develop the tools needed for auto generation of data both to and from the VC-model.

Use the VC-model to perform the VC of the real control programs

Since the PLC and the robot program for the SKF project was not completed within the timeframe of this thesis the model was not used to verify the real control programs. A future work would be to test the models with these programs in able to evaluate the model and collect feedback regarding its functions and suggestions for improvements.

9 References

- Auinger, F., Vorderwinkler, M. and Buchtela, G. (1999) Interface driven domain-independent modeling architecture for 'soft-commissioning' and 'reality in the loop'. *Proceedings of the 1999 Winter Simulation Conference*. p.798-805 [Online] Available from: http://ieeexplore.ieee.org/xpl/login.jsp?tp=&arnumber=823265&url=http%3A%2F%2Fieeexplore.ieee.org%2Fxpls%2Fabs_all.jsp%3Farnumber%3D823265. [Accessed: 2016-02-05]

- Barriball, K.L., & While, A. (1994). Collecting Data using a semi-structured interview: a discussion paper. *Journal of advanced nursing,* 19(2), 328-335. Available from: http://onlinelibrary.wiley.com/doi/10.1111/j.1365-2648.1994.tb01088.x/pdf [Accessed: 2016-05-29]

- Balci, O. (1998) Validation, verification, and testing techniques throughout the lifecycle of a simulation study. *Proceedings of the 1994 Winter Simulation Conference*. p.215-220. [Online] Available from: http://dl.acm.org.proxy.lib.chalmers.se/citation.cfm?id=194018 [Accessed: 2016-02-04]

- Björklund, M. et al., (2015) Virtual Commissioning with Oculus Rift. Chalmers University of Technology. Bachelor thesis. Gothenburg. [Online] Available from: http://publications.lib.chalmers.se/records/fulltext/219266/219266.pdf [Accessed: 2016-02-04]

- Carlsson, H., Svensson, B., Danielsson, F. & Lennartson, B., (2012), Methods for Reliable Simulation-Based PLC Code Verification. *IEEE TRANSACTIONS ON INDUSTRIAL INFORMATICS, VOL. 8, NO. 2, MAY 2012* [Online] Available from: http://publications.lib.chalmers.se/publication/157984-methods-for-reliable-simulation-based-plc-code-verification. [Accessed: 2016-05-24]

- Damrath, F., Strahilov, A., Bär, T., and Vielhaber, M., (2014). Establishing Energy Efficiency as Criterion for Virtual Commissioning of Automated Assembly Systems. *Conference on Assembly Technologies and System* 23(2014) p.137-142. [Online] Available from: http://www.sciencedirect.com/science/article/pii/S2212827114011391. [Accessed: 2016-02-04]

- Dostálek, L., Kabelová, A. (2006) *Understanding TCP/IP*. Great Britain: Packt Publishing.

- Drath R., Weber P. (2008) An evolutionary approach for the industrial introduction of virtual commissioning. [Online] Available from: http://ieeexplore.ieee.org.proxy.lib.chalmers.se/stamp/stamp.jsp?tp=&arnumber=4638359 [Accessed: 2016-02-05]

- Dzinic, J., Yao, C. (2013) *Simulation-based verification of PLC programs.* M.Sc. thesis, Chalmers University of Technology. [Online] Available from: http://publications.lib.chalmers.se/records/fulltext/195493/195493.pdf. [Accessed: 2016-02-05]

- Guerrero, J.V., López V.V., and Mejía J.E., (2015). Virtual Commissioning with Process Simulation (Tecnomatix). *Computer-Aided Design and Applications.* [Online] 11(1) p.11-19. Available from: http://www.tandfonline.com/doi/abs/10.1080/16864360.2014.914400. [Accessed: 2016-02-04]

- Hoffmann, P., Schumann, R. (2010) Virtual commissioning of manufacturing systems a review and new approaches for simplification. *Proceedings of the 2010 European Conference on Modelling and Simulation.* p.175-181 [Online] Available from: https://www.researchgate.net/profile/Andrzej_Bargiela/publication/235991499_Proceedings_of_the_24th_European_Conference_on_Modelling_and_Simulation/links/0deec516006eaa13e0000000.pdf#page=194. [Accessed: 2016-02-05]

- Kiefer, J., Baer, T., and Bley, H. (2008). Mechatronic-oriented design of automated manufacturing systems in the automotive body shop. *International design conference* [Online] 19 May. p.1295-1302. Available from: https://www.designsociety.org/publication/26717/mechatronic-oriented_design_of_automated_manufacturing_systems_in_the_automotive_body_shop. [Accessed: 2016-02-04]

- Ko, M., Ahn, E. & Park, S.C. (2013) A concurrent design methodology of a production system for virtual commissioning., *Concurrent Engineering,* vol. 21, no. 2, p. 129-140. [Online] Available from: http://cer.sagepub.com.proxy.lib.chalmers.se/content/21/2/129. [Accessed: 2016-03-22]

- Kühn, W. (2006), Digital factory: simulation enhancing the product and production engineering process. *Proceedings of the 2006 Winter Simulation Conference*, p. 1899-1906. [Online] Available from: http://dl.acm.org.proxy.lib.chalmers.se/citation.cfm?id=1218458#. [Accessed: 2016-03-22]

- Makris, S., Michalos, G., and Chryssolouris, G. (2012). Virtual Commissioning of an Assembly Cell with Cooperating Robots. *Advances in Decision Sciences.* [Online] Available from: http://www.hindawi.com/journals/ads/2012/428060/cta/. [Accessed: 2016-02-04]

- McGregor, I. (2002). The relationship between simulation and emulation. *Proceedings of the 2002 Winter Simulation Conference.* p.1683-1688 [Online] Available from: http://ieeexplore.ieee.org/xpl/login.jsp?tp=&arnumber=1166451&url=http%3A%2F%2Fieeexplore.ieee.org%2Fxpls%2Fabs_all.jsp%3Farnumber%3D1166451. [Accessed: 2016-02-05]

- OPC Foundation. 2016. History - OPC Foundation. [ONLINE] Available at: https://opcfoundation.org/about/opc-foundation/history/. [Accessed 30 May 2016].

- Rabe, M., Spickermann, S. and Wenzel, S. (2008) A new procedure model for verification and validation in production and logistics simulation. *Proceedings of the 2010 Winter Simulation Conference.* p.1717-1726 [Online] Available from: http://dl.acm.org/citation.cfm?id=1517045. [Accessed: 2016-02-05]

- Reddy, Y.J., (2015), *Industrial process automation systems: design and implementation*, Butterworth-Heinemann, Oxford, England; Waltham, Massachusetts. [Online] Available from: http://www.sciencedirect.com.proxy.lib.chalmers.se/science/book/9780128009390. [Accessed: 2016-05-24]

- Rossmann, J., Stern, O. and Wischnewski, R. (2007) Eine Systematik mit einem darauf abgestimmten Softwarewerkzeug zur durchgängigen Virtuellen Inbetriebnahme von Fertigungsanlagen von der Planung über die Simulation zum Betrieb. *apt Automatisierungstechnische Praxis.* 49(7). p.52-56.

- Sargent, R.G. (2010) Verification and validation of simulation models. *Proceedings of the 2010 Winter Simulation Conference.* p.166-183 [Online] Available from: http://dl.acm.org/citation.cfm?id=1162736. [Accessed: 2016-02-05]

- Schludermann, H., Kirchmair, T., & Vorderwinkler, M. (2000). Soft-commissioning: Hardware-in-the-loop-based verification of controller software. *Proceedings of the 2000 Winter Simulation Conference*. p.893-899. [Online] Available from: http://ieeexplore.ieee.org/xpl/login.jsp?tp=&arnumber=899889&url=http%3A%2F%2Fieeexplore.ieee.org%2Fxpls%2Fabs_all.jsp%3Farnumber%3D899889. [Accesses: 2016-02-05]

- Silva, F., Gamarra, C. J., Araujo Jr, A. H., & Leonardo, J. (2015). Product lifecycle management, digital factory and virtual commissioning: Analysis of these concepts as a new tool of lean thinking. *Proceedings of the 2015 International Conference on Industrial Engineering and Operations Management.* Available from: http://ieomsociety.org/ieom_2015/papers/199.pdf. [Accessed: 2016-03-22]

- Skoogh, A., Johansson, B. (2008) A methodology for input data management in discrete event simulation project data management. *Proceedings of the 2008 Winter Simulation Conference.* p.1727-1735 [Online] Available from: http://dl.acm.org/citation.cfm?id=1517046. [Accessed: 2016-02-05]

- Strahilov, A., Ovtcharova. A., Bär, T. (2012) Development of the physics-based assembly system model for the mechatronic validation of automated assembly systems. *Proceedings of the 2012 Winter Simulation Conference* [Online] Available from: http://ieeexplore.ieee.org.proxy.lib.chalmers.se/xpls/abs_all.jsp?arnumber=6465049&tag=1 [Accessed: 2016-02-05]

- Mujber, T.S., Szecsi, T., Hashm, M.S.J. (2004) Virtual reality applications in manufacturing process simulatin. Journal of Materials Processing Technology V.155–156 (2004) p.1834–1838. [Online] Available from: https://www.researchgate.net/profile/Msj_Hashmi/publication/222756230_Virtual_reality_applications_in_manufacturing_process_simulation/links/550b24010cf290bdc111d918.pdf [Accessed: 2016-02-04]

- Westkämper, E., Baudisch, T., Schlögl, W., Frank, G. (2012) Automatic model generation for virtual commissioning of specialized production machines, Softwaretechnik-Trends 32. No.2, 2 pp. Available from: http://pi.informatik.uni-siegen.de/stt/32_2/03_Technische_Beitraege/MAT12/mat2012_submission_5.pdf [Accessed: 2016-03-21]

Appendix A – Example C# Code in Experior for AGVs

```
namespace Process.Assemblies
{
    /// <summary>
    /// This simulate the AGV movements. The bearing size is defined in properties by
the user.
    ///
    /// Instructions:
    /// Z - Loads a full pallet with bearings and papers
    /// X - Loads an empty pallet
    /// C - Unloads the full pallet
    /// V - Unloads the empty pallet
    /// </summary>

    public class AGV : Assembly
    {

        #region Declerations

        private System.Timers.Timer timer;
        Vector vectormotor;

        public enum AGVstates { Idle, LoadHigh, LoadLow, UnLoadHigh, UnLoadLow }
        public AGVstates state = AGVstates.Idle;

        public int seq = 0; // Wait state in controller

        public Experior.Core.Parts.Mesh agvBody;
        public Experior.Core.Parts.Mesh agvBody2;
        public Experior.Core.Parts.Mesh agvLift;

        public Experior.Core.Parts.Sensors.RigidSensorPart AtHome;
        public Experior.Core.Parts.Sensors.RigidSensorPart AtConv;
        public Experior.Core.Parts.Sensors.RigidSensorPart AtPallet;

        // private Experior.Catalog.Load.EuroPallet EuroPallet;

        float conveyorOver = 0.05f; // hight over conveyor when lifting
        private bool liftDone = false;

        int manual = 0; // To enable manual triggers of load/unload of AGV and
materials.

        // --------------------------------------------
        bool regroup = false;
        bool regroup2 = false;
        private int grouptrigger = 0;
        private Experior.Core.Loads.Load[] list_grouped;
        private Experior.Core.Loads.Load[] list_grouped2;
        private Vector3[] list_Pos;
        private Experior.Catalog.Load.EuroPallet2 europallet;

        #endregion

        #region Constructors
        public AGV(AGVInfo info)
            : base(info)
        {

            // INIT METHODS
            initPLC();
            initTimer();
            CreateAgvMesh();
            CreateVectormotor();
            createSensors();
            InitInfoValues(); // Initate propertey values


        }



        #endregion

        #region Init Methods

        private void InitInfoValues()
        {
            // Init Bearing properties
            if (((AGVInfo)Info).OD == 0)
                ((AGVInfo)Info).OD = 0.118f;
            if (((AGVInfo)Info).Hight == 0)
                ((AGVInfo)Info).Hight = 0.063f;

            // Init conveyor hight
            if (((AGVInfo)Info).conveyorHight == 0)
                ((AGVInfo)Info).conveyorHight = 0.5f;
            if (((AGVInfo)Info).conveyorHightLow == 0)
                ((AGVInfo)Info).conveyorHightLow = 0.2f;
        }

        private void createSensors()
        {
            // At Home
            AtHome = new Experior.Core.Parts.Sensors.Cube(Color.Blue, 0.1f, 0.1f,
0.1f);
            Add(AtHome);
            AtHome.LocalPosition = ((AGVInfo)Info).AtHomePos;
            AtHome.Collision = Experior.Core.Environment.Collisions.Equipment;
            AtHome.Visible = false;

            //At Conveyor
            AtConv = new Experior.Core.Parts.Sensors.Cube(Color.Green, 0.1f, 0.1f,
0.1f);
            Add(AtConv);
            AtConv.LocalPosition = ((AGVInfo)Info).AtConvPos;
            AtConv.Collision = Experior.Core.Environment.Collisions.Equipment;
            AtConv.Visible = false;

            //At Pallet
            AtPallet = new Experior.Core.Parts.Sensors.Cube(Color.Black, 0.1f, 0.1f,
0.1f);
            Add(AtPallet);
            AtPallet.LocalPosition = ((AGVInfo)Info).AtPalletPos;
            AtPallet.Collision = Experior.Core.Environment.Collisions.Equipment;
            AtPallet.Visible = false;
}

public void initPLC()
{

    // ----DOloaddone----
    if (DOloaddone == null)
    {
        ((AGVInfo)Info).DOloaddone = new Output();
        DOloaddone.Description = "DOloaddone";
    }
    Add(DOloaddone);


    // ----DOunloaddone----
    if (DOunloaddone == null)
    {
        ((AGVInfo)Info).DOunloaddone = new Output();
        DOunloaddone.Description = "DOunloaddone";
    }
    Add(DOunloaddone);

    // ---Start LoadHigh---------
    if (LoadHigh == null)
    {
        ((AGVInfo)Info).In_loadmaterialHigh = new Input();
        LoadHigh.Description = "LoadHigh";
    }
    Add(LoadHigh);


    // ---Start LoadHigh---------
    if (LoadLow == null)
    {
        ((AGVInfo)Info).In_loadmaterialLow = new Input();
        LoadLow.Description = "LoadLow";
    }
    Add(LoadLow);

    // ---Start UnLoadLow---------
    if (UnLoadLow == null)
    {
        ((AGVInfo)Info).In_unloadmaterialLow = new Input();
        UnLoadLow.Description = "UnLoadLow";
        UnLoadLow.ByteNo = 1;
        UnLoadLow.BitNo = 3;
    }
    Add(UnLoadLow);

    // ---Start UnLoadHigh---------
    if (UnLoadHigh == null)
    {
        ((AGVInfo)Info).In_unloadmaterialHigh = new Input();
        UnLoadHigh.Description = "UnLoadHigh";
        UnLoadHigh.ByteNo = 1;
        UnLoadHigh.BitNo = 4;
    }
    Add(UnLoadHigh);

}
```

```
public void initTimer()
{
    timer = new System.Timers.Timer(4000);  // 4 sec
    timer.AutoReset = false;               // So timer does not restart
}

private void CreateAgvMesh()
{
    agvBody = new Experior.Core.Parts.Mesh(Common.Meshes.Get("AGVbody1.x"));
    agvBody2 = new Experior.Core.Parts.Mesh(Common.Meshes.Get("AGVbody2.x"));
    agvLift = new Experior.Core.Parts.Mesh(Common.Meshes.Get("AGVLift.x"));

    Add(agvBody);
    Add(agvBody2);
    Add(agvLift);

    agvBody.Rigid = RigidPart.Rigids.Convex;
    agvLift.Rigid = RigidPart.Rigids.Triangle;
    agvLift.Friction.Coefficient = Friction.Coefficients.Sticky;

    agvLift.Kinematic = true;
    agvBody.Kinematic = true;

    agvLift.OnLocalMovingFinished += agvLift_OnLocalMovingFinished;

}

private void CreateVectormotor()
{
    vectormotor = new Vector(new VectorInfo());
    vectormotor.Parts.Add(agvLift);
    vectormotor.Parts.Add(agvBody);
    vectormotor.Parts.Add(agvBody2);
    vectormotor.Name = "Motor Forward/backward";
    vectormotor.VectorDirection = new Vector3(0, 0, 0.8f);
    Add(vectormotor);

}

#endregion

#region Event Methods

void AGV_OnSelected(Assembly sender)
{
}

void agvLift_OnLocalMovingFinished(RigidPart part)
{
    liftDone = true;
}

#endregion

#region Other Methods

private void UpdatePos()
{
    AtHome.LocalPosition = ((AGVInfo)Info).AtHomePos;
    AtConv.LocalPosition = ((AGVInfo)Info).AtConvPos;
    AtPallet.LocalPosition = ((AGVInfo)Info).AtPalletPos;
```

```
        }

        private void addPallet()
        {
            Vector3 current = (AtPallet.Position) + (new Vector3(-0.3f, 0.06f, 0));

            // -----Adding only Europallet for lower conveyor-----
            if (state == AGVstates.LoadLow)
            {
                Experior.Catalog.Load.PureEuroPallet PureEuroPallet = new
Experior.Catalog.Load.PureEuroPallet(new Experior.Catalog.Load.PureEuroPalletInfo());
                Experior.Core.Loads.Load.Items.Add(PureEuroPallet);

                PureEuroPallet.Position = current;
            }

            // -----Adding full Europallet (Europallet + Bearing + Paper) for upper
conveyor------
            else
            {
                europallet = new Experior.Catalog.Load.EuroPallet2(new
Experior.Catalog.Load.EuroPallet2Info(), ((AGVInfo)Info).OD, ((AGVInfo)Info).Hight,
current);
                Experior.Core.Loads.Load.Items.Add(europallet);

                //regroup = true;
            }



        }

        public override void Dispose()
        {
            agvLift.OnLocalMovingFinished -= agvLift_OnLocalMovingFinished;

            base.Dispose();
        }

        public override void Step(float deltatime)
        {
            vectormotor.Step(deltatime);
            base.Step(deltatime);

            Controller();
        }

        public override void KeyDown(System.Windows.Forms.KeyEventArgs e)
        {

            if (e.KeyCode == System.Windows.Forms.Keys.Z) // LoadHigh
                manual = 1;
            if (e.KeyCode == System.Windows.Forms.Keys.X) // LoadLow
                manual = 2;
            if (e.KeyCode == System.Windows.Forms.Keys.C) // UnLoadHigh
                manual = 3;
            if (e.KeyCode == System.Windows.Forms.Keys.V) // UnLoadLow
                manual = 4;

            base.KeyDown(e);
        }
```

```
public override void Reset()
{
     DOunloaddone.Off();
     DOloaddone.Off();
     agvLift.OnLocalMovingFinished -= agvLift_OnLocalMovingFinished;

     agvBody.Dispose();
     agvBody2.Dispose();
     agvLift.Dispose();
     vectormotor.Stop();

     CreateAgvMesh();
     vectormotor.Parts.Add(agvLift);
     vectormotor.Parts.Add(agvBody);
     vectormotor.Parts.Add(agvBody2);

     seq = 0;
     manual = 0;
     state = AGVstates.Idle;
     liftDone = false;

     base.Reset();

}

#endregion



#region Controller

private void Controller()
{


    // ---------IDLE/INITIERING--------
    if (seq == 0 && (state == AGVstates.Idle)) // Wait for assignment
    {
        if (LoadHigh.Active == true || manual == 1)
        {
            seq = 1;
            state = AGVstates.LoadHigh;
            manual = 0;
        }
        if (LoadLow.Active == true || manual == 2)
        {
            seq = 1;
            state = AGVstates.LoadLow;
            manual = 0;
        }
        if (UnLoadHigh.Active == true || manual == 3)
        {
            seq = 1;
            state = AGVstates.UnLoadHigh;
            manual = 0;
        }
        if (UnLoadLow.Active == true || manual == 4)
        {
            seq = 1;
            state = AGVstates.UnLoadLow;
```

```
                manual = 0;
            }
        }

        // --- MISSIONS---------
        if (state == AGVstates.LoadHigh)
            Load(ConveyorHight);
        if (state == AGVstates.LoadLow)
            Load(ConveyorHightLow);
        if (state == AGVstates.UnLoadHigh)
            Unload(ConveyorHight);
        if (state == AGVstates.UnLoadLow)
            Unload(ConveyorHightLow);



    }

    private void Unload(float convhight)
    {
        if (seq == 1)   // Lift up
        {
            agvLift.LocalMovement(new Vector3(0, 0.5f, 0), convhight);
            seq = 2;
        }

        if (seq == 2 && liftDone == true) // Forward
        {
            liftDone = false;
            vectormotor.Forward();
            vectormotor.Start();
            seq = 3;
        }

        if (seq == 3 && AtConv.Active) // Stopping and lift
        {
            vectormotor.Stop();
            agvLift.LocalMovement(new Vector3(0, 0.1f, 0), conveyorOver);
            seq = 4;
        }

        if (seq == 4 && liftDone == true) // Moving home
        {
            liftDone = false;
            vectormotor.Backward();
            vectormotor.Start();
            DOunloaddone.On();
            seq = 5;
        }

        if (seq == 5 && AtHome.Active)   // Stopping and lower
        {
            vectormotor.Stop();
            agvLift.LocalMovement(new Vector3(0, -0.5f, 0), convhight +
conveyorOver);
            seq = 6;
        }

        if (seq == 6 && liftDone == true)   // Done
        {
            liftDone = false;
            state = AGVstates.Idle;
```

```
                DOunloaddone.Off();
                seq = 0;
            }
        }

        private void Load(float convhight)
        {
            if (seq == 1 && AtHome.Active) //To pallet
            {
                addPallet();
                DOloaddone.Off();
                vectormotor.Forward();
                vectormotor.Start();
                seq = 2;
            }

            if (seq == 2 && AtPallet.Active) //lift pallet
            {
                vectormotor.Stop();
                agvLift.LocalMovement(new Vector3(0, 0.3f, 0), (convhight + 
conveyorOver));
                seq = 3;
            }

            if (seq == 3 && liftDone == true) //going to Conveyor
            {
                liftDone = false;
                vectormotor.Forward();
                vectormotor.Start();
                seq = 4;
            }

            if (seq == 4 && AtConv.Active) //Place at conveyor
            {
                vectormotor.Stop();
                agvLift.LocalMovement(new Vector3(0, -0.2f, 0), conveyorOver);
                seq = 5;
            }

            if (seq == 5 && liftDone == true) //going home
            {
                liftDone = false;
                vectormotor.Backward();
                vectormotor.Start();
                seq = 6;
                DOloaddone.On();
            }

            if (seq == 6 && AtHome.Active) //At Home
            {
                vectormotor.Stop();
                agvLift.LocalMovement(new Vector3(0, -0.5f, 0), (convhight));
                seq = 7;
            }

            if (seq == 7 && liftDone == true)   //Done
            {
                liftDone = false;
                state = AGVstates.Idle;
                DOloaddone.Off();
                seq = 0;
            }
```

```
}


#endregion

#region Properties

[CategoryAttribute("Outputs")]
[DisplayName("DOloaddone")]
[DefaultFilter("I/O")]
public Output DOloaddone
{
    get { return ((AGVInfo)Info).DOloaddone; }
    set { ((AGVInfo)Info).DOloaddone = value; }
}

[CategoryAttribute("Outputs")]
[DisplayName("DOunloaddone")]
[DefaultFilter("I/O")]
public Output DOunloaddone
{
    get { return ((AGVInfo)Info).DOunloaddone; }
    set { ((AGVInfo)Info).DOunloaddone = value; }
}

[CategoryAttribute("Inputs")]
[DisplayName("LoadHigh")]
[DefaultFilter("I/O")]
public Input LoadHigh
{
    get { return ((AGVInfo)Info).In_loadmaterialHigh; }
    set { ((AGVInfo)Info).In_loadmaterialHigh = value; }
}

[CategoryAttribute("Inputs")]
[DisplayName("LoadLow")]
[DefaultFilter("I/O")]
public Input LoadLow
{
    get { return ((AGVInfo)Info).In_loadmaterialLow; }
    set { ((AGVInfo)Info).In_loadmaterialLow = value; }
}

[CategoryAttribute("Inputs")]
[DisplayName("UnLoadHigh")]
[DefaultFilter("I/O")]
public Input UnLoadHigh
{
    get { return ((AGVInfo)Info).In_unloadmaterialHigh; }
    set { ((AGVInfo)Info).In_unloadmaterialHigh = value; }
}

[CategoryAttribute("Inputs")]
[DisplayName("UnLoadLow")]
[DefaultFilter("I/O")]
public Input UnLoadLow
{
    get { return ((AGVInfo)Info).In_unloadmaterialLow; }
    set { ((AGVInfo)Info).In_unloadmaterialLow = value; }
}
```

```
public override string Category
{
    get { return "Assembly"; }
}

public override Image Image
{
    get { return Common.Icons.Get("MyAssembly"); }
}



[CategoryAttribute("Part Positions")]
[DisplayName("At Conveyor")]
[TypeConverter(typeof(Vector3Converter))]
public Vector3 AtConvPos
{
    get { return ((AGVInfo)Info).AtConvPos; }
    set { ((AGVInfo)Info).AtConvPos = value; UpdatePos(); }
}

[CategoryAttribute("Part Positions")]
[DisplayName("At home")]
[TypeConverter(typeof(Vector3Converter))]
public Vector3 AtHomePos
{
    get { return ((AGVInfo)Info).AtHomePos; }
    set { ((AGVInfo)Info).AtHomePos = value; UpdatePos(); }
}

[CategoryAttribute("Part Positions")]
[DisplayName("At pallet")]
[TypeConverter(typeof(Vector3Converter))]
public Vector3 AtPalletPos
{
    get { return ((AGVInfo)Info).AtPalletPos; }
    set { ((AGVInfo)Info).AtPalletPos = value; UpdatePos(); }
}

[CategoryAttribute("Lifting properties")]
[DisplayName("Conveyor hight - High")]
public float ConveyorHight
{
    get { return ((AGVInfo)Info).conveyorHight; }
    set { ((AGVInfo)Info).conveyorHight = value; }
}

[CategoryAttribute("Lifting properties")]
[DisplayName("Conveyor hight - Low")]
public float ConveyorHightLow
{
    get { return ((AGVInfo)Info).conveyorHightLow; }
    set { ((AGVInfo)Info).conveyorHightLow = value; }
}


//----- Bearing properties --------

[CategoryAttribute("Bearing initiation")]
[DisplayName("OD")]
public float OD
{
```

```
            get { return ((AGVInfo)Info).OD; }
            set { ((AGVInfo)Info).OD = value; }
        }

        [CategoryAttribute("Bearing initiation")]
        [DisplayName("Hight")]
        public float Hight
        {
            get { return ((AGVInfo)Info).Hight; }
            set { ((AGVInfo)Info).Hight = value; }
        }

        #endregion

    }

    [Serializable, XmlInclude(typeof(AGVInfo)), XmlType(TypeName =
"Process.Assemblies.AGV2Info")]
    public class AGVInfo : Experior.Core.Assemblies.AssemblyInfo
    {

        #region Fields

        private readonly static ProcessInfo properties = new ProcessInfo();
        public Vector3 AtPalletPos = new Vector3(0.18f, 0, 1);
        public Vector3 AtHomePos = new Vector3(0.18f, 0, 0);
        public Vector3 AtConvPos = new Vector3(0.18f, 0, 2);
        public Vector3 AGVPos;
        public float conveyorHight;
        public float conveyorHightLow;

        #endregion

        #region Properties

        public static object Properties
        {
            get
            {
                properties.color = Experior.Core.Environment.Scene.DefaultColor;
                return properties;
            }
        }

        #endregion

        #region Create PLC signals

        public Input In_loadmaterialHigh;
        public Input In_loadmaterialLow;
        public Input In_unloadmaterialHigh;
        public Input In_unloadmaterialLow;
        public Output DOloaddone;
        public Output DOunloaddone;

        #endregion

        #region Bearing

        public float OD;
        public float Hight;
        #endregion

        #region Motors

        public Experior.Core.Motors.VectorInfo vectormotorinfo = new VectorInfo();
        public Experior.Core.Motors.VectorInfo liftmotorinfo = new VectorInfo();

        #endregion
    }
}
```

Appendix B – Excel macro that auto generate C# code for I/O signals

```
Sub WriteCode()

Dim StartColumn As Integer
Dim StartRow As Integer
Dim C As Integer
Dim R As Integer

Dim Name As String
Dim IO As String
Dim DataType As String
Dim Description As String
Dim Class As String

' Autoformat: ctrl + K + F

StartColumn = 3
StartRow = 6
Class = Cells(36, 12).Value 'Plats K36

Set fs = CreateObject("Scripting.FileSystemObject")
Set a = fs.CreateTextFile("C:\Users\A503299\Desktop\PLC_texting.txt", True)

' Start writing code:

WriteText a, Cells(2, 4).Value 'write headline

C = StartColumn
R = StartRow

WriteText a, "------Initiation------"
Init R, C, a

C = StartColumn
R = StartRow

WriteText a, "-------IO Methods---------"
IOMethods R, C, a

C = StartColumn
R = StartRow

WriteText a, "---------Dispose-------"
Dispose R, C, a

C = StartColumn
```

```
R = StartRow

WriteText a, "---------Properties-------"
Properties R, C, a, Class

C = StartColumn
R = StartRow

WriteText a, "-------Info---------"
Info R, C, a

C = StartColumn
R = StartRow



a.Close

OpenFile


End Sub

Sub WriteText(a, Head)

a.WriteLine ("")
a.WriteLine ("")
a.WriteLine (Head)
a.WriteLine ("")

End Sub


Sub Init(R, C, a)

WriteText a, "// Outputs"

Do Until Cells(R, C).Value = ""

   IO = Cells(R, C).Value
   Name1 = Cells(R, C + 1).Value
   DataType = Cells(R, C + 2).Value
   Description = Cells(R, C + 3).Value
   Name = (IO & Name1)

   If IO = "DI" Then
   IO = "Input"
   End If
   If IO = "DO" Then
   IO = "Output"
   End If
   a.WriteLine ("if (" & Name & " == null)")
   a.WriteLine (Name & " = new " & IO & "();")
   a.WriteLine ("Add(" & Name & ");")

   If DataType <> "BIT" Then
    a.WriteLine (Name & ".Size = DataSize." & DataType & ";")
   End If

   a.WriteLine ("")

   R = R + 1
Loop

If Cells(R + 2, C) = "DI" Then

R = R + 2

WriteText a, "// Inputs"

Do Until Cells(R, C).Value = ""

   IO = Cells(R, C).Value
   Name1 = Cells(R, C + 1).Value
   DataType = Cells(R, C + 2).Value
   Description = Cells(R, C + 3).Value
   Name = (IO & Name1)

   If IO = "DI" Then
   IO = "Input"
   End If
   If IO = "DO" Then
   IO = "Output"
   End If

   a.WriteLine ("if (" & Name & " == null)")
   a.WriteLine (Name & " = new " & IO & "();")
   a.WriteLine ("Add(" & Name & ");")

   If DataType <> "BIT" Then
    a.WriteLine (Name & ".Size = DataSize." & DataType & ";")
   End If

   a.WriteLine (Name & ".On += Input_On;")
   a.WriteLine (Name & ".Off += Input_Off;")

   a.WriteLine ("")

   R = R + 1
Loop

End If
End Sub

Sub Properties(R, C, a, Class)

WriteText a, "// Outputs"

Do Until Cells(R, C).Value = ""

   IO = Cells(R, C).Value
   Name1 = Cells(R, C + 1).Value
   DataType = Cells(R, C + 2).Value
   Description = Cells(R, C + 3).Value
   Name = (IO & Name1)

   If IO = "DI" Then
   IO = "Input"
   End If
   If IO = "DO" Then
   IO = "Output"
   End If

      a.WriteLine ("[CategoryAttribute(""PLC " & IO & """)]")
      a.WriteLine ("[DisplayName(""" & Name & """)]")
      a.WriteLine ("[DefaultFilter(""I/O"")]")
      a.WriteLine ("public " & IO & " " & Name)
      a.WriteLine ("{")
      a.WriteLine ("get { return ((" & Class & "Info)Info)." & Name & "; }")
      a.WriteLine ("set { ((" & Class & "Info)Info)." & Name & " = value; }")
      a.WriteLine ("}")
      a.WriteLine ("")

   R = R + 1
Loop

If Cells(R + 2, C) = "DI" Then

WriteText a, "// Inputs"
R = R + 2

Do Until Cells(R, C).Value = ""

   IO = Cells(R, C).Value
   Name1 = Cells(R, C + 1).Value
   DataType = Cells(R, C + 2).Value
   Description = Cells(R, C + 3).Value
   Name = (IO & Name1)

   If IO = "DI" Then
   IO = "Input"
   End If
   If IO = "DO" Then
```

```
    IO = "Output"
    End If

        a.WriteLine ("[CategoryAttribute(""PLC " & IO & """)]")
        a.WriteLine ("[DisplayName(""" & Name & """)]")
        a.WriteLine ("[DefaultFilter(""I/O"")]")
        a.WriteLine ("public " & IO & " " & Name)
        a.WriteLine ("{")
        a.WriteLine ("get { return ((" & Class & "Info)Info)." & Name & "; }")
        a.WriteLine ("set { ((" & Class & "Info)Info)." & Name & " = value; }")
        a.WriteLine ("}")
        a.WriteLine ("")

    R = R + 1
Loop

End If

End Sub

Sub Info(R, C, a)

WriteText a, "// Outputs"

Do Until Cells(R, C).Value = ""

    IO = Cells(R, C).Value
    Name1 = Cells(R, C + 1).Value
    DataType = Cells(R, C + 2).Value
    Description = Cells(R, C + 3).Value
    Name = (IO & Name1)

    If IO = "DI" Then
    IO = "Input"
    End If
    If IO = "DO" Then
    IO = "Output"
    End If

    a.WriteLine ("public " & IO & " " & Name & ";")

    R = R + 1
Loop

If Cells(R + 2, C) = "DI" Then

R = R + 2

    WriteText a, "// Inputs"

Do Until Cells(R, C).Value = ""
```

```
    IO = Cells(R, C).Value
    Name1 = Cells(R, C + 1).Value
    DataType = Cells(R, C + 2).Value
    Description = Cells(R, C + 3).Value
    Name = (IO & Name1)

    If IO = "DI" Then
    IO = "Input"
    End If
    If IO = "DO" Then
    IO = "Output"
    End If

    a.WriteLine ("public " & IO & " " & Name & ";")

    R = R + 1
Loop

End If

End Sub

Sub Dispose(R, C, a)


Do Until Cells(R, C).Value = "DI"
    R = R + 1
Loop

WriteText a, "// PLC Dispose"

Do Until Cells(R, C).Value = ""

    IO = Cells(R, C).Value
    Name1 = Cells(R, C + 1).Value
    DataType = Cells(R, C + 2).Value
    Description = Cells(R, C + 3).Value
    Name = (IO & Name1)

    a.WriteLine (Name & ".On -= Input_On;")
    a.WriteLine (Name & ".Off -= Input_Off;")

    R = R + 1
Loop


End Sub


Sub IOMethods(R, C, a)

Do Until Cells(R, C).Value = "DI"
```

```
R = R + 1
Loop

Do Until Cells(R, C).Value = ""

   IO = Cells(R, C).Value
   Name1 = Cells(R, C + 1).Value
   DataType = Cells(R, C + 2).Value
   Description = Cells(R, C + 3).Value
   Name = (IO & Name1)

   If IO = "DI" Then
   IO = "Input"
   End If
   If IO = "DO" Then
   IO = "Output"
   End If

   a.WriteLine ("if (sender == " & Name & ")")
   a.WriteLine (";")
   a.WriteLine ("")

   R = R + 1
Loop


End Sub


Sub OpenFile()

   Dim myShell As Object
   Set myShell = CreateObject("WScript.Shell")
   myShell.Run "C:\Users\A503299\Desktop\PLC_texting.txt"

End Sub
```

Appendix C – Experior problems and suggestions

- Facilitate measuring in emulation environment
- Better structure and documentation of handling physics engine
- A twisted coordinate system compared to the integrated applications
- Yellow color in "Solution Explorer" results in bad visualization of text
- Develop an Experior version for 64bits to increase the capacity
- When grouping a load with another in the model, the physical representation changes rotation if MeshPart.Cylinder[] is used.
- Make it possible for a load or mesh to move along a path to simulate movements that are not included in the controller.
- Enable collision detection of robots in Experior. Priority including the gripper and the two nearest joints in collision detection. However, real verification of robot programs should be done in e.g. RobotStudio.
- Introduce an internal controller that can simulate the movements of the machines using PLC code or similar. In this project it was solved by using "override Step()". Could maybe be done in the interface instead.
- Large CAD files cannot be handled. This has caused following problems:
  - Gripper (tree-structure disappear after optimization and connections to RS are lost)
  - Layout (integrate the layout and CAD models from RS)
  - Införa CAD filer till simuleringsmiljön
  - Develop optimization programs within Experior
- When grouping loads, the original physical representation is not attached to the mesh if the grouping occurs before the emulation is running. The emulation needs to run when grouping in order for the physical representation to follow the mesh.
- IGripper – The coordinate system for the offset function of the LoadMagnet is confusing. Hard to position the sensor. In addition, have to establish connection to RS to see the relation between MagnetSensor and the gripper.
- IGripper not compatible with SmartComponents, only mechanisms in RS.
- Extend the information in Wikin with more detailed information regarding e.g. IGripper, physics engine, pitfalls, meshpart. More example codes and templates.
- Enable triggering signals in the emulation environment through an integrated interface.
- Introduce standardized components for machines as a template for customized components. Library should be extended with laths, grinders, milling machine etc.
- Include the possibility to turn off the physics engine and handle the loads without it. For example, when ridig är false for a load, it is not possible to be detected by sensors.
- Enable the possibility to graphically decide and define positions and rotations of components concurrently with coding in C#. E.g. place components as you want them in environment, then press "save positions" button, then the component's positions are stored in C# or info file.

Appendix D – Problems and suggestions for improvements in Robotstudio

- Edit Signals – Make it possible to change name and type of signal after its creation.
- Include categories for station signals.
- Include parametric solids that holds properties that are possible to change after the creation of the solid.
- Include a Set/Reset latch that toggle stats after an incoming pulse.
- Make a function that update all targets/signals in the beginning of the simulation, today there exist a function that does this but only for a specific logic block inside a SC.
- Make it possible to reach signal in SC that is located in a group component when they are connected through Station logics. This also includes the function when creating a mechanism, in this case only parts located directly under the browser is selectable.
- OPC Client (6.03.6906.1004): Today the Excel function for exporting and importing I/Os and connections does not work.
- OPC Client (6.03.6906.1004):: Make it compatible with the 64 bits version of RobotStudio.
- OPC Client (6.03.6906.1004):: The place where the name of the OPC-server is put in is initial read only.
- Make it possible to sort IOs in SC
- Make it possible to change the position frame for a tools, when a tool is created the position of the local coordinate system is located in the wob0 origin.
- Station Logic: When there is too many reset states the menu compose is not reachable.

- Add a sensor that can sense all the objects in its area. Today only 1 object can be sensed.

- When components are removed in the compose windows there should be an alternative to also remove connected signals related to that component.

- Add a “tear of function” for the rapid editor window and the simulation window.

- When the child SC “logical expression the size of the box is getting bigger when the expression of the logic block becomes longer. The box eventually too big.

- In the child SC “Logical expression” a function that marks which signals that is selected as “NOT” in the logical expression should be marked in the signal view in able to ease the debugging.

- The reset function is today unstable. Sometimes you have to activate it several times before components are replaced. Further Signals in both Rapid and varibles in SC is sometime not reset.

- After using the reset function queues is shown empty but some code is still left. This is shown when parts follows its old queues I a simulation after a reset.

- Today it is not possible to set when a variable should be transfer to a Analog output (group output) A smart component including an “execute” input that sets variable needs to be developed.