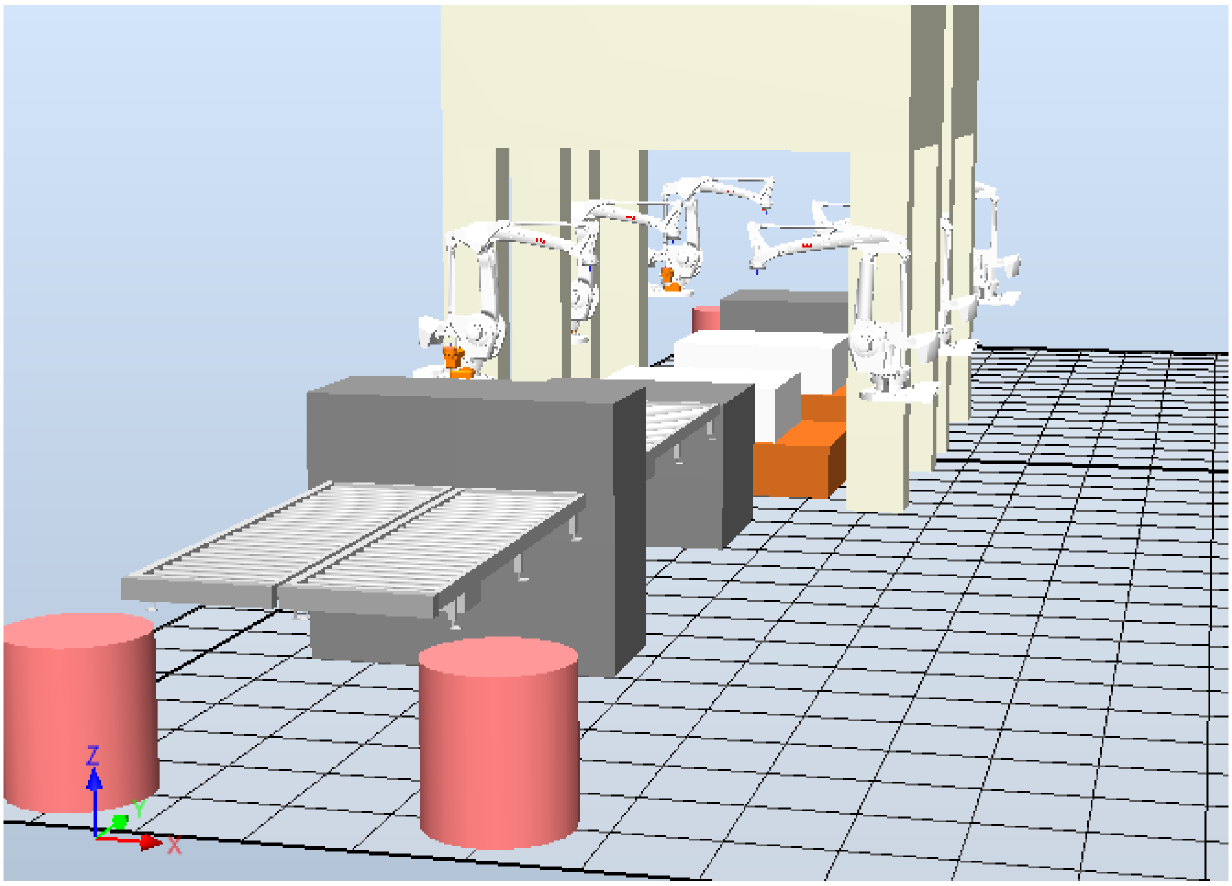
CHALMERS



Productization and validation of PressTending concept into a RobotStudio Addin

Master's Thesis in Systems, Control and Mechatronichs

NILS BLOMQVIST
JENS KULLBERG

Department of Signals and Systems
Division of Automation
CHALMERS UNIVERSITY OF TECHNOLOGY
Göteborg, Sweden 2013

Master's Thesis 2013:10

MASTER'S THESIS 2013:10

Productization and validation of PressTending concept into a RobotStudio Addin

Master's Thesis in Systems, Control and Mechatronichs

NILS BLOMQVIST
JENS KULLBERG

Department of Signals and Systems
Division of Automation
CHALMERS UNIVERSITY OF TECHNOLOGY

Göteborg, Sweden 2013

Productization and validation of PressTending concept into a RobotStudio Addin
NILS BLOMQVIST
JENS KULLBERG

Master's Thesis 2013:10
Number EX051/2013
Department of Signals and Systems
Division of Automation
Chalmers University of Technology
SE-412 96 Göteborg
Sweden
Telephone: + 46 (0)31-772 1000

Cover:
Auto generated station after user input.

Göteborg, Sweden 2013

Abstract

Simulation and off-line programming is today two essential concepts in the automation industry. With the advances made in simulation tools and robot systems (and the control of them) more and more scenarios can be simulated. One area where this is can be implemented is in Press Lines. In this complex environment much can be gained by having tools to build up, simulate and optimize the line in a virtual environment. This is called Virtual Commissioning (VC) and is the latest advances made in digital factoring. The thesis involves the Layout configuration of a Press Line. It is part of a greater tool for optimizing Press Lines. Using the Objective Oriented Programming Language (OOPL) C# (CSharp) a software for organizing, placing and connecting virtual components is created. In a Press Line a constant flow of parts needs to be fulfilled where every component in the line needs to be timed and exactly handled. Every movement being critical and few buffers allowed. Changes in the line directly effect other parts of the station. By having a tool that automatically can place and rearrange components from simple user inputs facilitates the build up of the station considerably.

In this thesis an add-in has been created to ABBs simulation software RobotStudio. The add-in is created in Visual Studio. A User Interface (UI) is created to let the user easily take control of the first stage, the layout configuration. The development of the add-in is done using ABBs Software Development Kits (SDKs) and therefore well compatible for further expansions. Relationship algorithms have been derived, which loops through the existing components in the virtual station in order to determine what relationships should be established. A relationship is defined by determining which device a robot should interact with, e.g. picking up a part from a load table or leaving to a press. After the relationships has been calculated placements are done. Locking segments of the line have been made possible in the add-in. By adding this implementation it is easy for the user to make small changes, whilst still having the possibility to auto correct the rest of the line. Managers for relations, placements and locked objects has been created to keep track of the state the station. The resulting software is a fully functioning add-in with UI for easy control of the different scenarios. The overall project successfully fulfils the outset objectives.

Keywords: Press Line, RobotStudio, Digital Factory, Virtual Commissioning, C#, Automated Layout, Robot, ABB, Relations, Programming

Sammanfattning

Simulering och offline programmering är idag två viktiga koncept inom automaitonsindustrin. Med de framsteg som har gjorts med simuleringsverktyg och robotsystem (och kontrollen av dem) fler och fler scenarier kan simuleras. Ett område där detta kan utnyttjas är presslinjer. Det är ett komplicerat område som kräver avancerade verktyg för att bygga, simulera och optimera virtuellt. Detta kallas virtuell drifttagning och representerar de senaste framstegen som gjorts inom digital fabriker. Tesen avhandlar den virtuella drifttagningen av en presslinje. Den är en del av ett större projekt för att optimera dessa linjer. Med det objektorienterade programmeringsspråket C# (CSharp) har en mjukvara utvecklats för att organisera, placera och koppla virtuella komponenter. Ett konstant flöde genom presslinjen krävs och varje moment behöver vara tids- och välkontrollerat. Varje rörelse är kritisk och få buffrar är tillåtna. Ändringar i linjen påverkar direkt andra delar av linjen. Genom att ha ett verktyg som automatiskt placerar och ordnar komponenter i linjen genom enkla användarinställningar förenklar uppbyggnad och minskar ramp-up tiden.

En addin har utvecklats till ABBs simulationsmjukvara RobotStudio. Addin:en är skapad i Visual Studio. Ett användargrännssnitt är framtaget som låter användaren ta kontroll över de första stegen för att simulera och optimera en presslinje. Utveckling har skett med hjälp av ABBs Software Development Kits (SDKs) och är därför väl kompatibelt för vidare utveckling. Relationsalgoritmer är framtagna, vilka går igenom befintliga komponenter i den virtuella stationen för att avgöra vilka relationer som behöver skapas. En relation är definierad genom att avgöra vilken enhet som roboten ska interagera med, t.ex. plocka en komponent från ett laddningsbord eller lämna till en press. Efter att realationerna är satta så görs placeringarna. Fixering av segment i linjen har gjorts möjlig i addin:en. Genom att lägga till denna funktion gör det lätt för användaren att ändra saker, utan att förlora möjligheten att autokorrigera resten av linjen. Hanterare för relationer, placeringar och låsta object har utvecklats för att hålla koll på tillståndet i linjen. Resulterande mjukvara är en fullt fungerande addin med användargränssnitt för enkel kontrol av olika scenarier. Projektet uppfyller utsatta mål.

Nyckelord: Press Line, RobotStudio, Digital Fabrik, Virtuell Driftsättning, C#, Automatisk Layout, Robot, ABB, Relationer, Programmering

Contents

List of Acronyms

PP	PowerPac
HSPL	High Speed Press Line
BTP	Bhoruka Tech Park
API	Application Programming Interface
SDK	Software Development Kit
OOPL	Object Oriented Programming Language
BLL	Business Logic Layer
DAL	Data Access Layer
UI	User Interface
WPF	Windows Presentation Foundation
XML	Extensible Markup Language
GUID	Globally Unique Identifier
VC	Virtual Commissioning
SIL	Software In Loop
HIL	Hardware In Loop

Preface

The work on this thesis took place from the 15^{th} of February until 15^{th} of July 2013 at *ABB Global Industries & Services Ltd* as a part of the *ABB India Student Intern Program.* The office is located at Bhoruka Tech Park (BTP) in Bangalore, India. The internship has been planned under the guidance of Mr. Sudarshan M-V, from Discrete Automation & LV Products Department. An add-in for ABBs simulation tool RobotStudio has been developed. The add-in with implemented user interface automatically places and connects the components and robots needed in a High Speed Press Line (HSPL). Furthermore, additional options such as saving layout segments and making changes after the auto generation is available.

Aknowledgements

Firstly we would like to thank Magnus Larsson at ABB Bangalore for finding this project and making it possible for writing it India. We felt very welcome and generously treated by everyone. Also a big thanks to our supervisor Sudarshan and his team for guidance throughout our stay in India. Finally, Petter Falkman for believing in this project and helping us with the report.

Göteborg June 2013
Nils Blomqvist, Jens Kullberg

1 Introduction

Today's industries are constantly forced to find new and more optimal solutions in their production lines. Since the birth of off-line programming it is no longer a question of "if" rather then "when" a company should optimize their line. Off-line programming is nowadays widely used in industries all over the globe. It is believed that since the introduction of industrial robots in the late 1960, over 2,3 million of them have been sold. An estimated 1.1-1.4 million were still operational by the end of 2011. This accumulated number varies with the estimated average service life of a industrial robot system, 11 respectively 15 years. In a market estimated to be worth over 25 billion dollars and showing a steady growth since the global financial crisis in 2009, it is clear that keeping up to date is more important than ever [1].

In production lines today one of the time consuming aspects, when dealing with control systems especially, is the ramp up time of the real assembly line. One way to shorten the ramp up time is to first simulate the factory and thereby acknowledge faults early in the design phase. Great advantages has been made over the last years in digital factory/manufacturing techniques. The latest trend in digital manufacturing is Virtual Commissioning (VC) which goes one step further than the "Digital Factory" concept. VC incorporates the mechatronic behaviour of the components in the line using the actual PLCs in a virtual environment. The idea is to early implement engineering data from the design phase needed for the simulation, such as CAD components, material flow, detailed I/O connections for the control system and IT infrastructure [2].

A Press Line is a series of presses in a linear pattern. They have long been used in the industry and their goal is to deliver high quality and high output. Press Lines is used to form metal sheets by passing them through the line and consequently pressing them until they have the correct form [3]. Increasing the output of the press line whilst still keeping the quality is a complex procedure with many parameters to be considered. Today many press lines uses on-line tuning together with manual settings to reduce the cycle time. This is often done with empirical testing by individual operators [4]. The Press Line considered in this thesis is a press-to-press line. This means that no intermediate station exists between the presses, the work object is instead transferred directly to the next press [5]. In this case by a robot system, which acts as an extractor/unloader for the previous press and a feeder/loader to the next press. Simulating this procedure is a highly complex task since the whole system is dependent on exact timings and precision throughout the line[4]. Achieving this by off-line programming and making simulations available will give higher efficiency and increased quality.

ABBs own trademarked simulation tool RobotStudio enables both simulation and off-line programming for robot systems. This type of programming is essential both when implementing new robot systems or when updating existing. In order to keep RobotStudio well suited for different applications, add-ins to the basic program are available. These add-ins are called PowerPac (PP)s and are constantly developed to attend customer demands. Developing Addins for RobotStudio is done in different programming languages, ABB's support covers Visual Basic and CSharp(C#) [6].

Several advantages can be made by switching to simulation testing instead of the more traditional way of shop-floor testing. Simulation leads to good decisions, which leads to reduced costs and higher efficiency [7]. Today Press Lines needs advanced calculations in order to determine the most efficient cycle time, offset between presses, necessary parameters etc. A new Addin capable of simulating the entire press line would greatly benefit the industry and also save both time and money. This thesis concerns the start of this new PP, i.e developing an add-in handling the placement and connections of the components needed to represent the press line in a virtual environment. By reducing the time needed to place all the presses, robot systems and other needed components more time can be spent optimizing the line. In Figure 1.1 the primary components in the line is shown, the virtual objects represent a loading table, a robot system and a press.

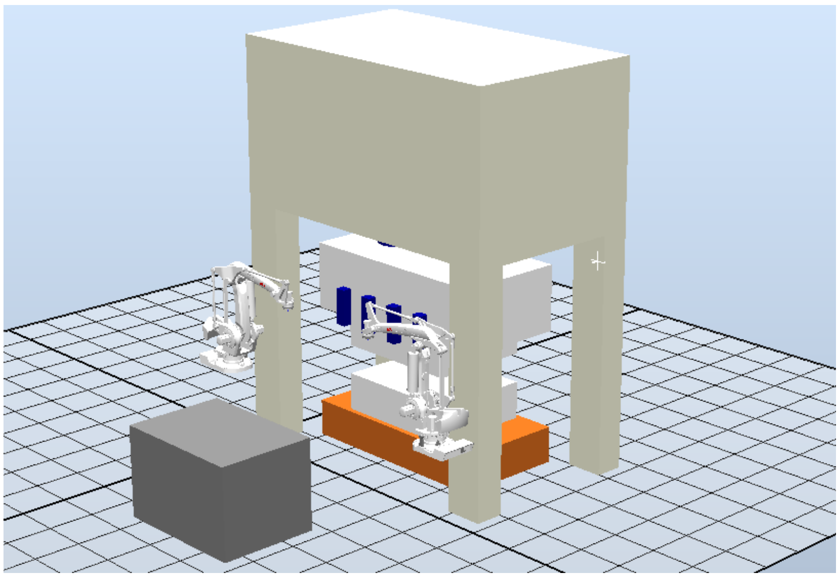


Figure 1.1: A Robot System, Load Table and Press in RobotStudio

1.1 Purpose and aim

This master thesis concerns the development and validation of a subsystem to a new Addin. Fig. 1.2 illustrates the projects development plan and this thesis contribution.

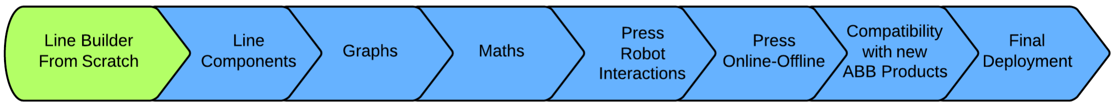


Figure 1.2: Overlook of the development plan. Note; this thesis contribution is coloured green.

The add-ins main purpose is to control the automatic placement of the components in a virtual robot station as well as connecting them automatically. The user should, with help of the add-in, be able to set up a virtual representation of a Press Line.

The add-in solves an otherwise time consuming process of placing and connecting components before the actual simulation of the cell is possible. The optimization process uses newly developed formulas that are depending on where components are placed in the robot cell. Possibilities to continuously make changes such as moving, adding or removing components in the station is a necessary feature for later optimization.

1.2 Specifications

This thesis revolves around a layout challenge where the user with help of a User Interface (UI) should be able to set the number of available components, such as robots, presses, tables and import them into RobotStudio. Automatically a layout of a Press Line should be created in a controlled way. The add-in should uphold the specifications presented 1.2.1 in Tab. 1.1 and Tab. 1.2.

1.2.1 Specifications of the add-in

The specification on the add-in was presented in the beginning by ABB, [8] and developed accordingly to their varying priority, noted on a scale of Low, Medium, High.

Specification	Priority
Automatic layout generation from single Step wizard	High
Automatic interconnections with standard I/O signals	High
Solution based on Smart Components and standard ABB libraries	High
Possibility to import new components in a controlled way	Medium

Table 1.1: Specifications of the add-in

1.2.2 Added specifications to the add-in

During discussions in connection to the planning of the master thesis it became clear that a feature for locking down components was needed. This would enable rearranging the complete line automatically with out changing the locked down components. This feature was not in the original specifications.

Added specification	Priority
Possibility to lock a components position and attributes	Medium

Table 1.2: Added specifications of the add-in

1.3 Limitations

With focus lying on implementing the objectives and making the program stable, design of the UI has been limited. No specific test program is used, more than what is available in Visual Studio and RobotStudio. Collision detection will not be added. Developed algorithms will not be optimized.

2 Theory

This section describes the theory behind the implemented methods and software. It starts by explaining Virtual Commissioning (VC) with its requirements and advantages. Continuing with theory behind the chosen programming language and code structure. The section also covers used applications and interfaces such as RobotStudio, development kits for said application and continues with underlying math for component placement/movements. Persistence in RobotStudio and the XML structure used is also handled in this section.

2.1 Virtual Commissioning

VC was in the beginning called "soft commissioning" and introduced the possibility of connecting real systems with virtual models. The testing and validation could now be done before implementing a real system. VC extends this concept with added simulation analysis, engineering requirements and a more complete vision of the life cycle of the technical system [2]. It facilitates the transition between the real and digital factory. The goal is to test and validate the control in the virtual environment and then, without changes, connect the same control to the real system. This enables the development and configuration of control systems in parallel with mechanical and electrical development. The possibility for different groups to work on the same model enables earlier testing and validation of the plant [9]. As can be seen in [2] a reduction of the ramp up time by 15-25%, using VC on a real system, is achieved.

There are two ways of implementing VC. The first way is Software In Loop (SIL), which means that the control and system is completely represented virtually. One big advantage is that no hardware is needed and that the simulation and control can be done on a PC. The second way, Hardware In Loop (HIL) is when equipment of the real system is tested in conjunction with the virtual environment. As explained in [10], dividing up the hardware and software gives four different scenarios explained below and can be seen in Figure 2.1.

1. Real control system and real system: traditional way of commissioning
2. Real control and virtual system
3. Virtual control and real system
4. Virtual control and virtual system: complete VC

Connecting software with hardware in the second and third example can be solved with OPC. One of the advantages of the forth example and VC is that the simulation can be run in the same tool [10]. Tools that make this possible is e.g. CIROS Planner and ABBs RobotStudio.

2.1.1 Data requirements for VC

To achieve full VC and make use of all its advantages engineering data is needed. These data is often available early in the design phase and VC takes advantage of this fact. Collecting these data and implement them in a structured form lays way for the VC. In accordance to [2] the data requirements are as follow.

- Simulation models of components covering kinematics, geometries, controller programs and electronics.

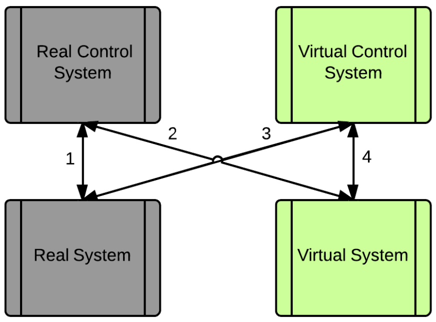


Figure 2.1: Combinations of hardware and software solutions

- Design of the production line regarding placement of resources and equipment.
- Material flow in the line, sequence of operations and relations of processes in the production.
- Control systems, either real or virtual.
- I/O signals in the control system and the connections of the resource components.
- Available functionalities such as e.g safety systems and the signals needed to implement them in the commissioning process.
- Communication protocols and software drivers for connections between the control system and the virtual environment (usually TCP/IP).

With these requirements fulfilled the implementation of VC has come a long way. Using VC facilitates the decisions for the design engineers regarding type and number of components, which type of communications and connections/interfaces between the resources in the line. It is a valuable tool and provides a foundation for easier, time- and cost efficient setup for production [2].

2.2 Object Oriented Programming Language

The chosen programming language C# is an Object Oriented Programming Language (OOPL) developed by Microsoft from C++ and is a part of Microsoft's .NET initiative. It is designed to be a simple and modern programming language and to be used for developing software components [11]. With OOPL objects are created and relationships between these objects are formed. Each object is self-contained and have its own methods/operations and data/attributes. Complicated systems and entities in real life can be implemented in a direct way. This is made possible by the availability of data abstraction, inheritance and encapsulation. This way of coding is tolerant to changes and has high flexibility, reusability and extensibility [12].
Each object stems from a class, which, as mentioned above has methods and attributes. The inheritance possibilities gives the option to easily create subclasses. An illustration of the class structure can be seen in Fig. 2.2.

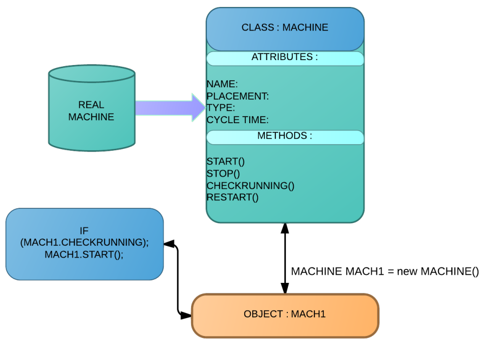


Figure 2.2: Class structure example

2.3 Three Layer code architecture

This section presents the theory behind the multi tier architecture, used to achieve a three layer framework. This is done to simplify the programming process and facilitate changes and increase the development efficiency. Each layer should only communicate with its closest neighbour. Applications constantly change due to user specifications, new standards or other factors. By applying the three layer architecture changes in the code are only necessary at certain places.

The three layers consist of from top to bottom: Presentation layer, also know as UI, explained in 2.3.1, Business Logic Layer (BLL), explained in 2.3.2, and Data Access Layer (DAL), explained in 2.3.3. The first layer is the one visible to the user. This layer handles input and displays data. The input/output is then transferred to the BLL, where the necessary calculations and calls are done. The BLL is the core of the system and handles the operations. If access to data is needed it is the BLL that calls the DAL. It is then the DALs task to retrieve or write the data in the database implementation [13]. The different layers and the structure can be seen in Fig. 2.3

2.3.1 Presentation Layer

The presentation layer is also called the front layer. It is what the user sees and interacts with. The components use to make this possible is all included in the presentation layer. It handles all the inputs from the user and displays the necessary information [14].

By using Windows Forms adding components to the presentation layer is convenient. It has been apart of the .NET Framework since the beginning. Window forms is a library that builds up standard components such as buttons and drop down lists. It is well suited for business like applications. If there is no need for rich media content in the UI it is equally suited for applications as the newer more graphic Windows Presentation Foundation (WPF) [15].

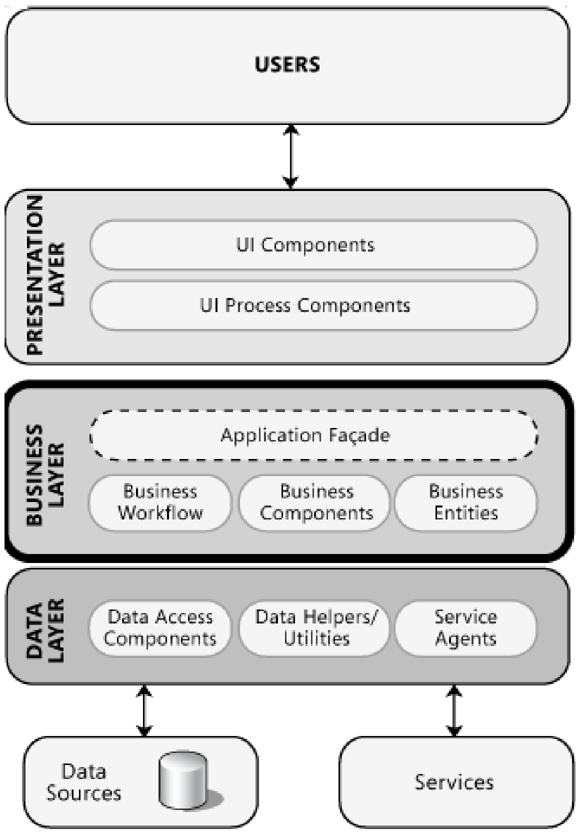


Figure 2.3: Three Layer Structure: UI, BLL and DAL

2.3.2 Business Logic Layer

BLL occupies the middle section of the three-tier planning framework and coordinates the application. It handles and processes commands, calculation operations and logical decisions. The BLL is the middle-man between the three layers and is the core of the system [16],[13].

2.3.3 Data Access Layer

In computer software the DAL, is a part of the application which provides an easy access to stored data. In object-oriented programming the layer returns a reference to an object rather than tables of data, though it can be used both ways. Data access components are created specifically to make the application easier to configure and maintain in terms of functionality [17].

2.4 RobotStudio Add-ins

ABB's simulation tool RobotStudio implements several different functions. Writing a add-in where the application lets the user control certain functions demands some knowledge by the programmer. To access different functions in RobotStudio by the application the right Software Development Kit (SDK), explained in 2.5.1, has to be used. With the PC-SDK e.g. the application programmer can ask for mastership, explained in 2.5.2, over the virtual IRC5 Controller, explained in 2.5.3.A graphical object in the simulation enviroment contains it position and orientation represented in a 4×4 matrix, explained in 2.5.4. Extensible Markup Language (XML) provides the ability to exchange and store information in RobotStudio, explained in 2.5.5, and how this is done is explained in 2.5.6.

(a) RobotStudio-SDK for developing applicationss for the RobotStudio environment.

(b) PC-SDK is used for writing programs accessing and editing either the real or the virtual IRC5 Controller.

(c) When constructing interfaces and applications for the FlexPendant the Flexpendant-SDK i used.

Figure 2.4: The three SDKs

2.5 Developing applications for RobotStudio

Developing applications for RobotStudio is done with three different SDK with corresponding Application Programming Interface (API). The three SDKs are PC SDK, RobotStudio SDK and the FlexPendant SDK. With these SDKs programmers can develop customized interfaces suiting user specific needs [18],[19].

2.5.1 Software Development Kits

The SDKs comes with the RobotStudio installation. They make it possible to develop applications on the RobotStudio platform and customize it to the users specifications. Such as added scripts, extra user interfaces or entire add-ins. These packages/libraries are essential when developing PPs for RobotStudio. The three SDKs are RobotStudio-SDK, PC-SDK and FlexPendant-SDK and are represented in Figure 2.4.

RobotStudio-SDK is the basic SDK needed for developing software applications for RobotStudio. It contains the API crucial for developing, extending and customizing RobotStudio to the user specifications. RobotStudio SDK is also used when creating custom Smart Components with Code behind [19], further explained in 2.5.9.

PC-SDK enables programmers to build applications on the PC and connect them to the ABB Robot controller (Virtual, Real). The customized applications, be it for third parties, end users or system integrators, can be implemented as stand alone PC applications. They can also operate the IRC5 controller via the network.

FlexPendant-SDK gives the possibility to write custom programs for this device and therefore extend the functionalities available [18]. FlexPendent is a hand held device attached to the robot controller. It can handle several of the functions needed to operate the robot, such as: running, creating and editing specific programs, jogging the robot etc.

Another advantage of the PC-SDK is the ability to keep track of multiple IRC5 controllers in the same application, this is not possible with the FlexPendant. The PC application is however a remote client and will not have the same privileges as the FlexPendent which is a local client. Remote clients also need to request Mastership over the Rapid domain, before they get write access [6].

2.5.2 Mastership

In order to control the robots on the shop floor mastership is needed. This is implemented so that only one person or program has control the robot at any given point. Several clients can still be logged on with read only access, which is a default access right, but only one with write access. This is for security reasons as well as protection from data being accidentally overwritten. The same applies for the virtual IRC5 controller in RobotStudio, when a user via an add-in in RobotStudio wants to write to the controller access must first be given. This is done by taking mastership over the controller [6]

2.5.3 The Virtual IRC5 Controller

The IRC5 is ABB's fifth generation robot controller. RobotStudio provides a perfect digital copy of the robot system together with strong programming and simulation features [20]. In other words, the same code can be executed both in the RobotStudio simulation and on the shop floor without any difference in behaviour or physical appearance. To write code accessing the virtual controller PC-SDK is used [18].

2.5.4 Homogeneous Transformation and Matrix4

The following is a summary of Homogeneous Transformation as explained in [21]. ABB uses via RobotStudio-SDK a 4×4 matrix to describe the orientation of an object in the workspace, the used notation is *Matrix4*. Every object in the workspace has a *Basefram* defined by a *Matrix4*. The workspace is the platform where the actual robot cell is placed. The workspace has one defined world coordinate system, it acts as a base reference point and can be placed anywhere in the workspace, usually at $(x, y, z) = (0, 0, 0)$. Each objects baseframe is referenced to this coordinate.

An orientation in a three dimensional euclidean space can be represented as three unit vectors, forming a 3×3 matrix. To explain a rotation of the orientation a rotation matrix can be used. Given two orientations O_0 and O^1 in the same origin and with unit vectors x_0, y_0, z_0 and x_1, y_1, z_1 respectively, the orientation of O_1 with rotation matrix $\mathbf{R}$ is calculated by:

$$O_1 = RO_0 \tag{2.1}$$

Introducing a third orientation O_3 and the notation R^i_j representing going from orientation i to j. With $i, j = 0, 1, 2$ the rotation from O_0 to O_2 is written as $\mathbf{R}^0_2 = \mathbf{R}^0_1\mathbf{R}^1_2$. Worth noting is that the rotations are not commutative, $\mathbf{R}^2_1\mathbf{R}^1_0 \neq \mathbf{R}^1_0\mathbf{R}^2_1$. The rotation matrix for an angle α around the x-axis is given by:

$$\mathbf{R}_x(\alpha) = \begin{pmatrix} 1 & 0 & 0 \\ 0 & \cos\alpha & -\sin\alpha \\ 0 & \sin\alpha & \cos\alpha \end{pmatrix} \tag{2.2}$$

In the same way the rotations about the y- and z-axis with an angle of β and γ is given by

$$\mathbf{R}_y(\beta) = \begin{pmatrix} \cos\beta & 0 & \sin\beta \\ 0 & 1 & 0 \\ -\sin\beta & 0 & \cos\beta \end{pmatrix} \tag{2.3}$$

$$\mathbf{R}_z(\gamma) = \begin{pmatrix} \cos\gamma & -\sin\gamma & 0 \\ \sin\gamma & \cos\gamma & 0 \\ 0 & 0 & 1 \end{pmatrix} \tag{2.4}$$

As mentioned before representing both the rotation and location from one coordinate system to another is made possible by the homogeneous transformation matrix. Given two orientations in space, O_0 and O_1 with a distance $\mathbf{d}_1^0$ between them. With vector points $\mathbf{p}_0$ and $\mathbf{p}_1$ for the different orientations and the rotation matrix $\mathbf{R}_1^0$ describing the relative location of O_0 to O_1. The vector points can now be given by

$$\mathbf{p}_0 = \mathbf{d}_1^0 + \mathbf{R}_1^0\mathbf{p}_1. \tag{2.5}$$

It can also be shown that the inverse transformation holds, which can be written as

$$\mathbf{p}_1 = -\mathbf{R}_0^1\mathbf{d}_1^0 - \mathbf{R}_0^1\mathbf{p}_0. \tag{2.6}$$

In a more compact way the homogeneous transformation 4×4 matrix can be written as

$$\mathbf{A}_1^0 = \begin{bmatrix} \mathbf{R}_1^0 & \mathbf{d}_1^0 \\ \mathbf{0}^T & 1 \end{bmatrix} \tag{2.7}$$

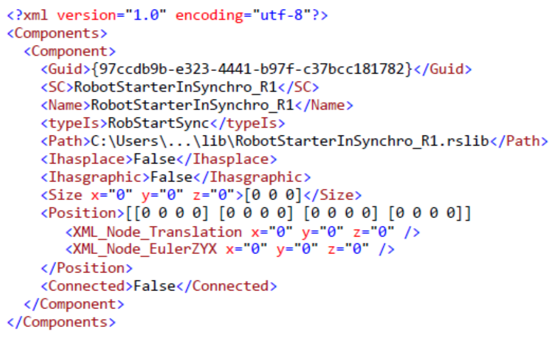
and expresses the coordinate transformation from one frame to another. The final matrix with a slight difference, i.e formulating the matrix is, defined as:

$$\mathbf{A}_1^0 = \begin{bmatrix} \mathbf{R}_1^0 & \mathbf{0} \\ \mathbf{d}_1^{0T} & 1 \end{bmatrix} \tag{2.8}$$

2.5.5 XML

XML was introduced in the 1990s and is a way of exchanging and storing data. XML can be used to share data between almost any application. Even though it is text based and easily understood for a human eye, processing the information can be quite complex. The implementation of the data can be done in different ways and XML leaves much freedom for the user. With just simple rules to follow, it is very flexible and lets the user decide rather freely on how to build up the data storage. For making it possible for different applications or even companies a XML schema can be used for exchanging data in a structured and consistent way. Implementations vary between manipulations of existing XML data, store data easily read by other applications and standards that already are based on XML [22].

XML code can be created and edited in any standard text editor. The guidelines for XML was defined by World Wide Web Consortium (W3C) and is a way of having structured data in plain text. The data structure is formed by creating and organising elements and attributes. The structure is only bound by some simple rules, and then its up to the user to decide which form it will take [22]. In Fig. 2.5 the XML document is used for saving a state of a component with name, position, type etc.



```xml
<?xml version="1.0" encoding="utf-8"?>
<Components>
  <Component>
    <Guid>{97ccdb9b-e323-4441-b97f-c37bcc181782}</Guid>
    <SC>RobotStarterInSynchro_R1</SC>
    <Name>RobotStarterInSynchro_R1</Name>
    <typeIs>RobStartSync</typeIs>
    <Path>C:\Users\...\lib\RobotStarterInSynchro_R1.rslib</Path>
    <Ihasplace>False</Ihasplace>
    <Ihasgraphic>False</Ihasgraphic>
    <Size x="0" y="0" z="0">[0 0 0]</Size>
    <Position>[[0 0 0 0] [0 0 0 0] [0 0 0 0] [0 0 0 0]]
       <XML_Node_Translation x="0" y="0" z="0" />
       <XML_Node_EulerZYX x="0" y="0" z="0" />
    </Position>
    <Connected>False</Connected>
  </Component>
</Components>
```

Figure 2.5: XML document saving a state of a component in a virtual environment

2.5.6 Persistence

When RobotStudio is closed the information of the station state needs to be stored somewhere. An add-in with connections and relations has a lot of information and have to store that information between closing and opening the main program. RobotStudio persists the changes that have been made in the station by providing “Attributes” where information can be stored. Any object in RobotStudio inheriting from *Project Object* (PO), an abstract base class, can be used to store information. Attributes are stored as key/value pair, for safety and stability from conflict with other add-ins it is recommended to associate the keys with a Globally Unique Identifier (GUID) and is only known to the client. The value itself is the XML file and an example of one is shown in Fig. 2.6 [23].

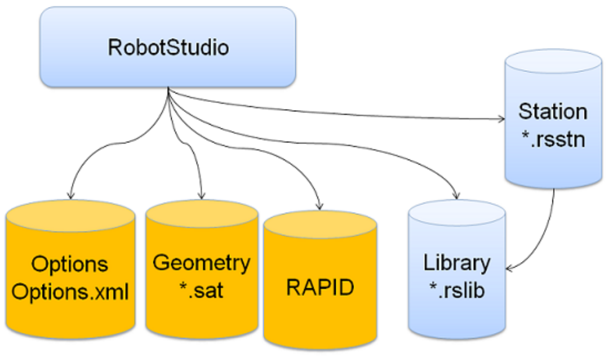


Figure 2.6: XML document saving a state of a component in a virtual environment

2.5.7 I/O Connections

Each SmartComponent has the ability to communicate with another SmartComponent via I/O-signals.

2.5.8 Property bindings

Property bindings allow communication between SmartComponents on a higher level, they connect values of one property to the value of another. This allows for mathematical implementation in the SmartComponents.

2.5.9 Smart Component

Smart Component is a RobotStudio object with or without a 3D graphical representation, that has a added behaviour which can be implemented by code-behind and/or aggregation

Source Object	Specifies the owner of the source property
Source Property	Specifies the source of the binding
Target Object	Specifies the owner of the target property
Target Property	Specifies the target of the binding

Table 2.1: Property binding description

by other Smart Components. The following list describes the different terminologies used when setting up a robot cell and working with Smart Components. The lists is directly cited from [24].

- **Code behind** A .NET class associated with a Smart Component that can implement custom behavior by reacting to certain events, for example simulation time steps and changes in property values.
- **[Dynamic] property** An object attached to a Smart Component that has value, type and certain other characteristics. The property value is used by code behind to control the behavior of the Smart Component.
- **[Property] binding** Connects the value of one property to the value of another property.
- **[Property] attributes** Key-value pairs that contain additional information about a dynamic property, for example value constraints.
- **[I/O] signal** An object attached to a Smart Component that has a value and a direction (input/output), analogous to I/O signals on a robot controller. The signal value is used by code behind to control the behavior of the Smart Component.
- **[I/O] connection** Connects the value of one signal to the value of a different signal.
- **Aggregation** The process of connecting several Smart Components using bindings and/or connections in order to implement a more complex behavior.
- **Asset** Data object contained in a Smart Component

3 Method

This chapter describes the different methods and general approach used in the thesis. In VC the idea is to early on acquire the data needed for commissioning the line in a virtual environment. The data is usually available in the design phase but needs to be implemented. Most of the requirements for VC, as seen in 2.1.1, is available from the start of the thesis and is therefore implemented in the application. The work is concentrated on solving the available specifications, for the addin, given by ABB. These specifications closely resembles the requirements needed to do VC. VC has taken the digital factory design one step further by extending it and making better use of the engineering data already available early on in the design phase. To do a complete VC of the line would need some further data not available at the moment. The chosen methods and approach is mostly planned in conjunction with ABB but also decided by own research and logic.

3.1 Approach

The primary goal was to create relationships between the presses and the robots and then build from there. The relationships core is built up by the device concept, section 3.3. By creating data objects for every component in the station and then establishing relations between them, the placement would come relatively easy. Categorising the components relative the robots as devices facilitates the structure and the placement. A device is defined as an object which the robot interacts with. In order to keep track of the relationships between a robot system and device a relationship manager class is designed and engineered. The positioning and distancing is managed by the distance manager class. All classes are explained in more detail in 3.4. Section 3.5 explains the automatic placement algorithms. Section 3.6 shows how the I/O connections is done. 3.7 contains the added lock-feature algorithm.

There are different ways of testing a control systems. Figure 2.1 shows four ways by combining virtual and real testing. The third way is called emulation and the forth is pure simulation. In the thesis pure simulation is used, VC. The control of the robots for ABB is however the exact same for virtual and real controllers. This gives the possibility to test the control in the virtual environment and then directly transfer it to the real system.

3.2 Defining challenges

The complexity of this project regards the automatic placement of the HSPL. By first establishing the relationship between the chosen components in the station the automatic placement becomes relatively easy. A developed algorithm is used to determine the relationships and works together with a relationship manager class in order to keep track of the initial state and any changes.

- **First layout**
  The first challenge is to establish relationships between robots, presses and tables etc. in the station. When all relationships is determined the placements needs to be calculated. The placement of the components takes input from the user for an initial layout. With help of the established relationships the placement goes through the components in the correct order and determines the position of the objects.

- **Change layout**
  The possibility to add another press, conveyor or other existing component into the station is a one of the basic specifications. The added components needs to create new relationships and also get relative positions. With this implemented, options to automatically recalculate and remodel the station should exist.

- **Lock down layout**
  It is important not to unintentionally overwrite information regarding a components attributes such as position when updating a robot cell. This causes obvious problems. If the user positions robot at a certain position and then updates the robot cell, the software would recalculate and remodel from default settings, which means that the changes in the station will be overwritten and ignored. A possibility to lock down components at their current position and with its other attributes is a wanted but complex feature. This has been solved and is presented in 3.7.

3.3 Device concept

According to the requirements, part **1.4.1 Device concept** every area which might be occupied by robots or machines is identified as a device.

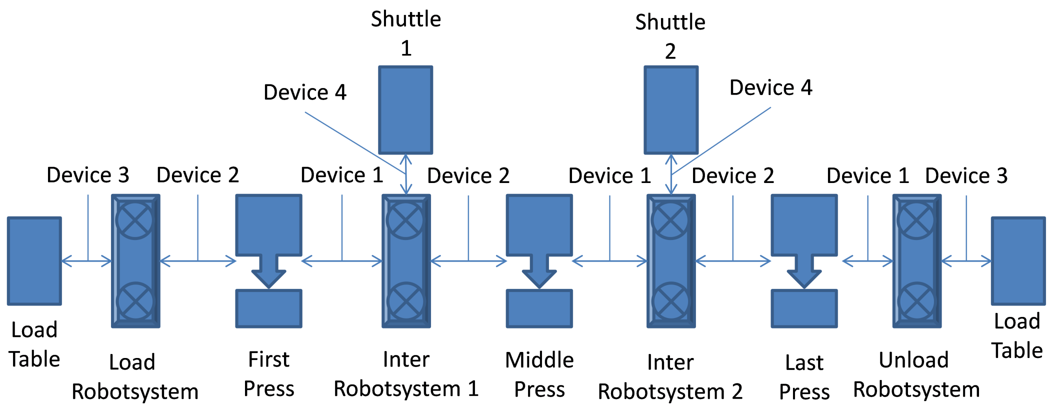


Figure 3.1: Example of a robot cell layout

Every device is located in connection to a robot. The function of each device vary depending on the purpose of the robot it is connected to. The relationship between robot and its attached devices is henceforth known as "Relationship". The possible devices a robot system can have is shown in Tab. 3.1.

Robot System	Device 1	Previous Press
	Device 2	Next Press
	Device 3	Load/Unload Table

Table 3.1: Available Relationship

3.3.1 Robot System

The user labels the available robot systems in the add-in. Each robot is set by the user to be either a Loader, Interpress or Unloader robot described in Tab. 3.2. The labels represent the robot systems activity in the add-in and not which type of robot. It is implemented in such a way that there can only exist one or none Loader robot and one or none Unloader robot. Each type of robot comes with different settings and conditions e.g. a robot set to be Loader is given priority over Interpress and Unloader to connect to the press labelled First. The concept of labelling presses is explained further in 3.3.2.

Robot System	Loader Robot System
	Interpress Robot System
	Unloader Robot System

Table 3.2: Available Robot Systems

3.3.2 Press System

Each press can be either device 1, 2 or both at the same time depending on the number and type of robots in the robot cell. Both meaning the press is previous and next press at the same time according to Tab. 3.1. Depending on the purpose of the press it exits a possibility to label the press system as either First, Middle or Last press described in Tab. 3.3. Each label comes with different settings and conditions e.g. a press labelled First is given priority over Middle and Last to connect to a Load table.

Type	Device
First Press	1,2 or both
Middle Press	1,2 or both
Last Press	1,2 or both

Table 3.3: Available Press Systems

3.3.3 Table

A Table can either be Load or Unload table but it is always a device 3 as described in Tab. 3.4.

Type	Device
Load	3
Unload	3

Table 3.4: Available Table System

3.4 Classes

This section is divided into two parts, implemented BLL classes and implemented DAL classes, that represents the core of the software and where the actual “work” is done. There are more classes working in the background but they give little understanding of the whole process, therefore they are not presented to the fullest in this paper. One such class is the “Helper” class, its only purpose is to “assist” the programmer with commonly used methods and functions that are not connected to a specific object. Fig. 3.2 illustrates a overlook of the most important implemented classes in the software.

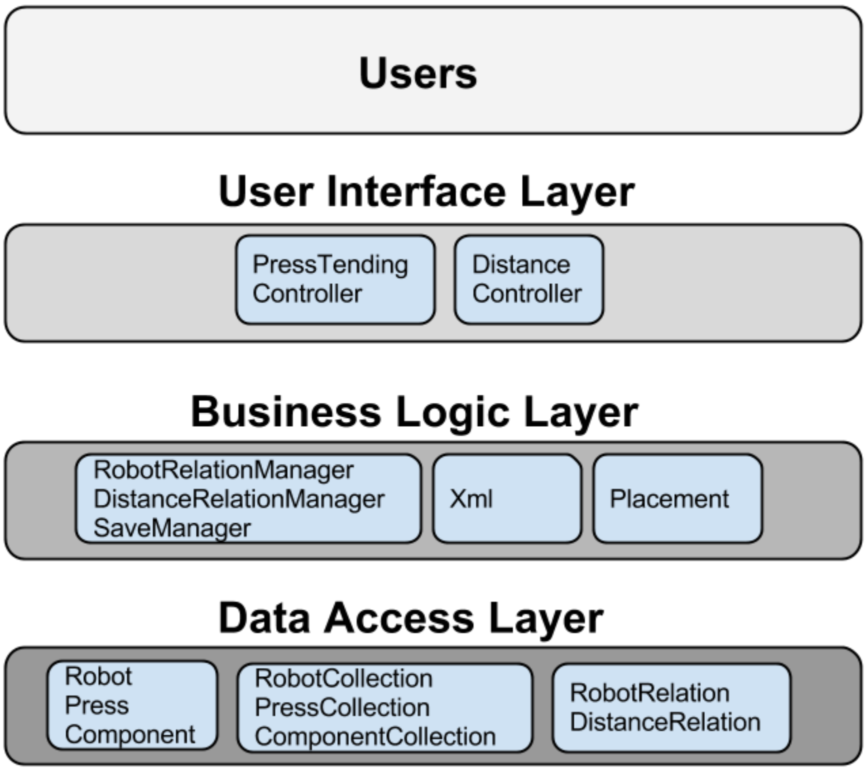


Figure 3.2: Layer

Implemented DAL Classes Implemented DAL Classes consists of the classes programmed under the DAL, e.g. the class **Robot** which when called by the user creates a data object representing a real robot system. The data object only contains the information equal to what a robot "in real life would know", such as position, orientation and name. Every virtual object in RobotStudio is given a unique identification, GUID, that is passed onto the object. This is a crucial process that must work due to the fact that if the GUID does not match the objects unique identification the used algorithm can not separate a component from different component. The same implementation is used with the Press and Component class. A robot object does not contain information regarding its relationship to other devices neither the distance to them. The information regarding both robot and distance relationship for a robot object are created simultaneously and stored in separate collection classes.

Implemented BLL Classes Implemented BLL Classes consists of the classes programmed under the BLL, e.g. the class **RobotRelationManager** which is created only once when the add-in is started. RobotRelationshipManager stores information regarding all RobotRelationship objects. Each robot class is assigned with a relationship the moment it is created. The distance relationship stores information about the desired distance, set by the user in the UI, to any device. Each robot class is assigned with a distance relationship the moment it is created. DistanceRelationManager manages all DistanceRelation objects. The managing involves calculating the distances set by the user between robot and device. The LockManager class is responsible for and handles the locking down process. It involves recalculating distances depending on the new position of the locked object. There are a number of constraints and limits that make up the rules about where/when/how the user is allowed to move and lock an object explained in 3.7.3.

3.5 Placement class

This chapter explains what and how the Placement class is used and implemented. It also sits under the BLL but due to the importance to the software it is explained in more detail. How the used automatic placement algorithm is coded and implemented is explained under 3.5.1.

3.5.1 Auto-Placement Method

In order to auto-place the components in the robot cell a algorithm is developed. The algorithm consists of placing the first robot at any desired position, e.g. at $(x, y, z) = (0, 0, 0)$ in the robot cell and adding on the next robot in the chain of robots until there are no more to place. The chain in this case is the list of robots in descending order from first to last. The next robot in the chain is according to the set relationships always the robot that the current robot is sharing a device with. When all the robots are placed, each robots devices are placed around it. A maximum of two robots can be parent to one particular device, for example the robot on the left side of the device and the robot on the right side.

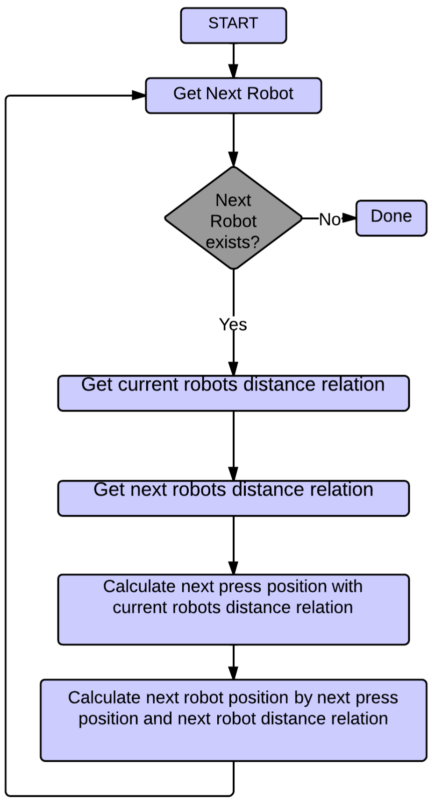

START
Get Next Robot
Next Robot exists?
No
Done
Yes
Get current robots distance relation
Get next robots distance relation
Calculate next press position with current robots distance relation
Calculate next robot position by next press position and next robot distance relation

Figure 3.3: Auto-placement algorithm flowchart

To calculate the position of the next robot, if there exists one, the algorithm takes the current robots position and checks the distance to the device that the two robot must share and calculates a temporarily position at a new point in the coordinate system. From this point the algorithm checks the distance to the device from the next robot, inverts it and calculates the position of the next robot. This creates a zigzag “position”-curve shown in Fig. 3.4.

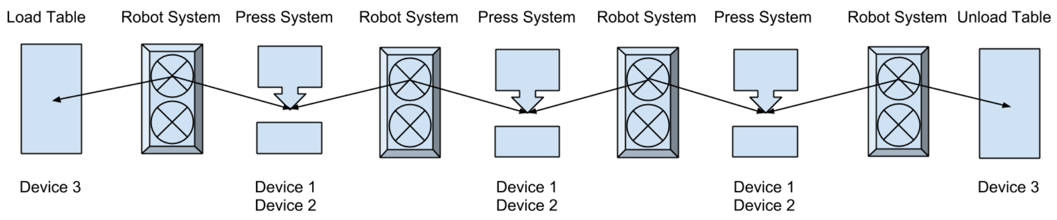


Figure 3.4: Zigzag “position”-curve

3.6 Auto connections

One of the objects is to automatically connect the Smart Components in the Station. This is done by creating property bindings and I/O connections. Each Smart Component needed for the station is added as an object in the program, and its inputs and outputs is known. Depending on the amount of presses and robots in the station, the bindings and connections are then created. This is needed for simulations to run. Algorithms built in to the Press Components and several line-configuration Smart Components need to be connected to control the movement and offset of the presses.

3.7 Lock algorithm

One foreseeable problem is that, for example a robot, has been moved manually instead of with the UI and followed up by updating the robot cell the software recalculates and remodels from “default” settings. Which means that the manual changes made in the station will be overwritten and ignored. A possibility to lock components at their current position is therefore a needed feature. To solve this a new method is invented and is described in two stages in 3.7.1 and 3.7.2. Depending on which component is locked there are different consequences and also a set of conditions are implemented both are described in 3.7.3.

3.7.1 First

When the user selects a component in RobotStudio and presses “Save”, the lock-algorithm finds the corresponding data object of the component and flags it as “Saved”. The algorithm locks the distance relationship in question, making them unchangeable until the robot that owns the relationship is flagged “Unsaved” by the user. The flowchart in Fig 3.7 illustrates how the algorithm functions. Note that the backtracking part of the flowchart is explained in 3.7.2.

3.7.2 Second

The first part solves the problem with overriding and ignoring the manual changes but the locked component will be moved due to the chosen placement method describe in 3.5.1. The distances between the locked robot and its devices will still be the same but e.g. if a distance between a robot and press is changed and the robot is placed before the locked down robot, the locked robot will be moved. The solution is called backtracking and how it works is illustrated in Fig. 3.6. When the algorithm encounters a robot system in a locked state,

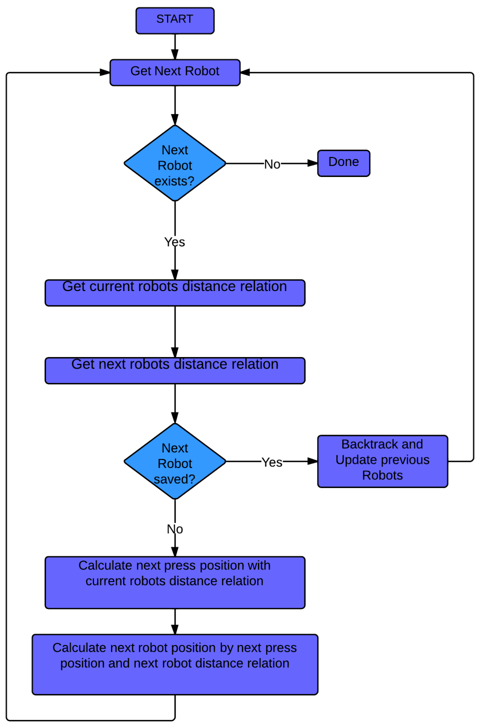

START
Get Next Robot
Next Robot exists?
No
Done
Yes
Get current robots distance relation
Get next robots distance relation
Next Robot saved?
Yes
Backtrack and Update previous Robots
No
Calculate next press position with current robots distance relation
Calculate next robot position by next press position and next robot distance relation

Figure 3.5: Lock algorithm flowchart, including backtracking algorithm

it places the system at the locked position regardless of the newly calculated position of the system. Afterwards the backtracking recalculates the positions of the previous systems using a algorithm similar to the one showed in Fig. 3.3 but backwards.

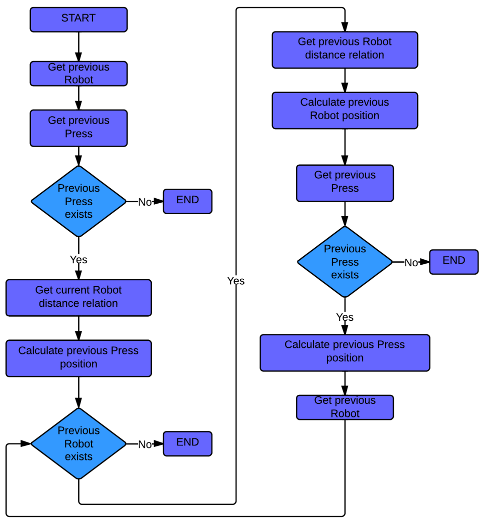

START
Get previous Robot
Get previous Press
Previous Press exists
No
END
Yes
Get current Robot distance relation
Calculate previous Press position
Previous Robot exists
No
END
Yes
Get previous Robot distance relation
Calculate previous Robot position
Get previous Press
Previous Press exists
No
END
Yes
Calculate previous Press position
Get previous Robot

Figure 3.6: Backtracking algorithm flowchart

3.7.3 Consequences and constraints

The conditions and constraints are set up to be intuitive and logical.

Conditions Different conditions apply regarding on which component is locked. When chosen component is selected and locked the following will happen:

- *Robot System* All distance relationships owned by the robot are locked.
- *Press System* The distance relationship owned by the robot which the press was moved towards is locked.
- *Load/Unload Table* The distance relationship owned by the robot which the load/unload table was moved towards is locked.
- When an unlocked robot is moved outside the base frame of any attached device the robot is moved and no distances will be changed (the whole chain follows).
- When an unlocked press is moved closer to a robot and locked down the rest of the chain behind the press follows.
- When an unlocked press is moved further away from a robot and locked down the rest of the chain behind the press follows.

Constraint This constraint is based on logics, e.g. if a robot is located between two locked down robots, the distances to its devices can not be changed since the locked robots are set a certain positions. If the robots distances to its devices were to be changed, the locked robot would be moved which defeats the purpose of the lock-feature.

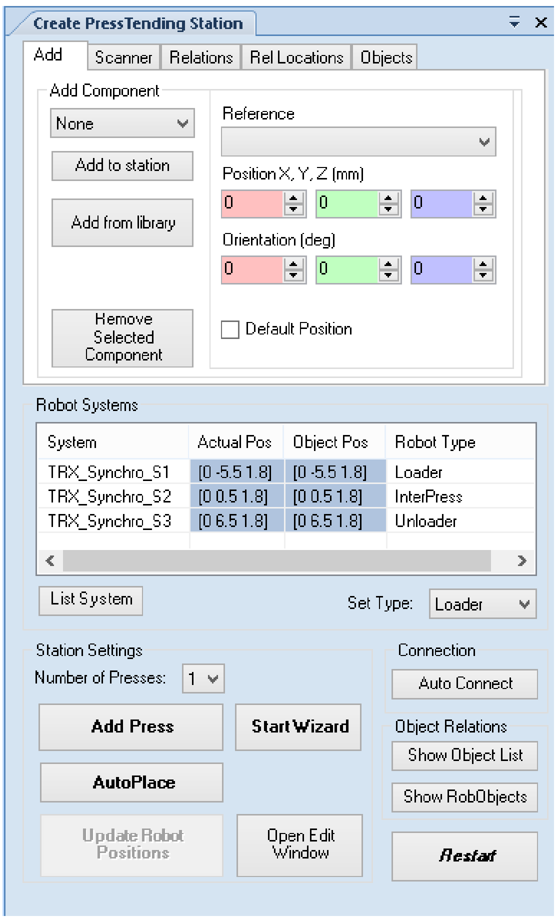
- It is impossible to move a robots distances, if the robot is located between two other locked robots.

4 Results

This section presents the software that has been programmed according to specifications described in Section “Specifications” and implemented using “CSharp” in “Visual Studio” for “RobotStudio”. Ideally the results of this paper would be best presented in a physical demonstration; however, this is not possible in this format. The results are therefore presented in scenarios during different circumstances.

4.1 User interface

In this scenario there are three robots in the robot cell, one of each type. Fig. 4.1a shows the main window of the add-in. The middle window named “Robot Systems” is where the user can see the added robot systems in the cell. The actual position of the virtual robot in the cell is shown next to its corresponding data objects position. It is implemented in this prototype as a check point but is not needed in the finished product. The top part handles the adding of components from other ABB libraries e.g. conveyor belts as in 4.2, Fig. 4.2a and Fig. 4.2b



Add Component: None

Position X, Y, Z (mm): 0 | 0 | 0

Orientation (deg): 0 | 0 | 0

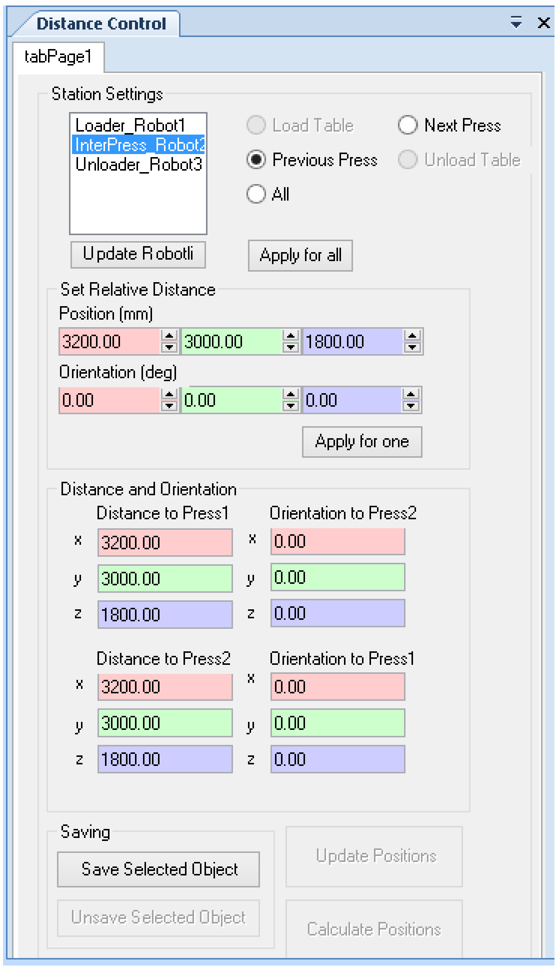
Robot Systems

System	Actual Pos	Object Pos	Robot Type
TRX_Synchro_S1	[0 -5.5 1.8]	[0 -5.5 1.8]	Loader
TRX_Synchro_S2	[0 0.5 1.8]	[0 0.5 1.8]	InterPress
TRX_Synchro_S3	[0 6.5 1.8]	[0 6.5 1.8]	Unloader

Set Type: Loader

Number of Presses: 1

(a) Main window of the add-in



Loader_Robot1
InterPress_Robot2
Unloader_Robot3

[✓ Previous Press]

Set Relative Distance

Position (mm): 3200.00 | 3000.00 | 1800.00

Orientation (deg): 0.00 | 0.00 | 0.00

	Distance to Press1		Orientation to Press2
x	3200.00	x	0.00
y	3000.00	y	0.00
z	1800.00	z	0.00

	Distance to Press2		Orientation to Press1
x	3200.00	x	0.00
y	3000.00	y	0.00
z	1800.00	z	0.00

(b) Distance controller in the add-in

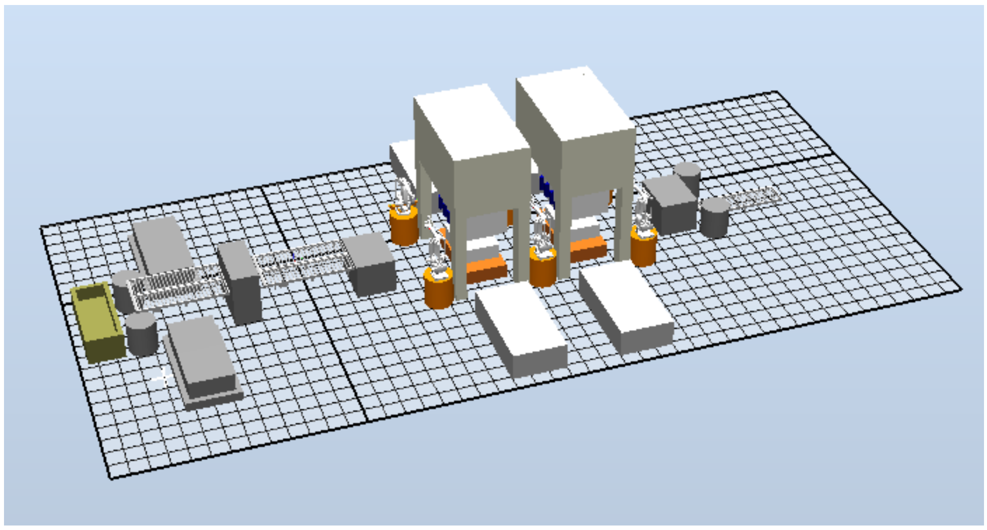
Figure 4.1: Pictures of user interface, need new ones

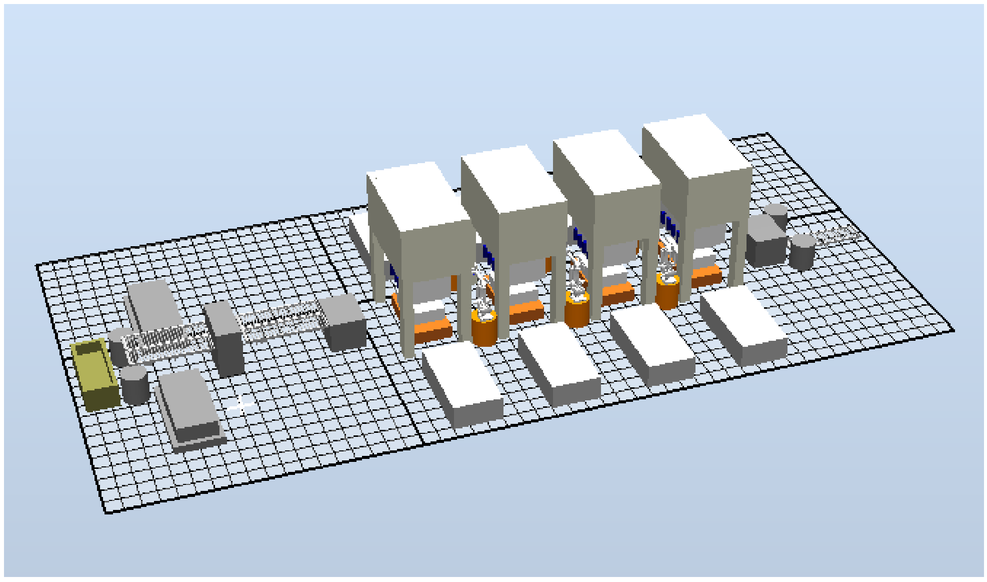
Fig. 4.1b shows the distance controller, reached by the “Open Edit Window” button in the main window. Firstly, this is where all robots distance can be changed by selecting a robot from the list and then which device can be selected right next to the list. Secondly, at the bottom of the windows is where the lock function is implemented, consisting of a

"Save" and a "Unsave" button. The object to be locked can either be selected from the list or from the virtual environment in RobotStudio by clicking on it. The middle part of this window shows the current distance to the selected robots devices.

4.2 Auto-layout

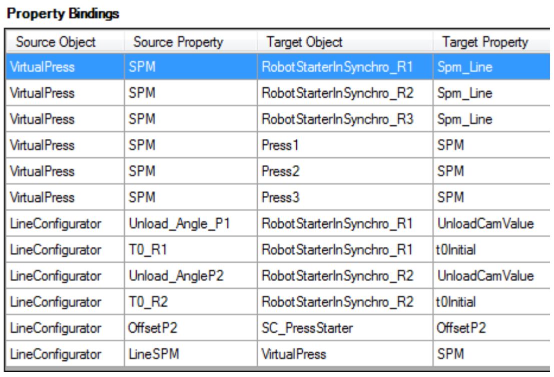
Because of the flexibility of the add-in it is always possible to add another component, if available, to the cell. To avoid redundancy two scenarios of different layouts is presented.


(a) This station consists of three robot systems, two presses, one load table and one unload table. The cell is also equipped with other components imported from other ABB libraries.


(b) This station consists of three robot systems, four presses, one load table and one unload table. The cell is also equipped with other components imported from other ABB libraries.

4.3 Auto-connections

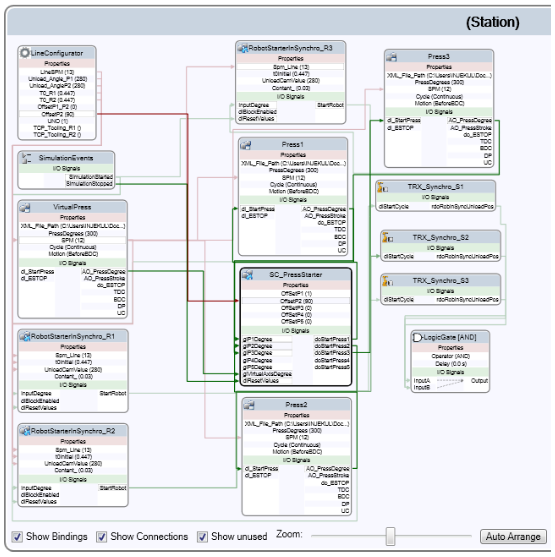
These two figures illustrates the same scenario. Fig. 4.2c shows the connected property bindings in the cell while in Fig. 4.2d both property bindings, red lines, and I/O connections, green lines.



Property Bindings

Source Object	Source Property	Target Object	Target Property
VirtualPress	SPM	RobotStarterInSynchro_R1	Spm_Line
VirtualPress	SPM	RobotStarterInSynchro_R2	Spm_Line
VirtualPress	SPM	RobotStarterInSynchro_R3	Spm_Line
VirtualPress	SPM	Press1	SPM
VirtualPress	SPM	Press2	SPM
VirtualPress	SPM	Press3	SPM
LineConfigurator	Unload_Angle_P1	RobotStarterInSynchro_R1	UnloadCamValue
LineConfigurator	T0_R1	RobotStarterInSynchro_R1	t0Initial
LineConfigurator	Unload_AngleP2	RobotStarterInSynchro_R2	UnloadCamValue
LineConfigurator	T0_R2	RobotStarterInSynchro_R2	t0Initial
LineConfigurator	OffsetP2	SC_PressStarter	OffsetP2
LineConfigurator	LineSPM	VirtualPress	SPM

(c) The property bindings in the robot cell.



(d) All connections in the robot cell represented.

4.4 The whole add-in

Fig. 4.2 shows how the add-in look like while running in RobotStudio.

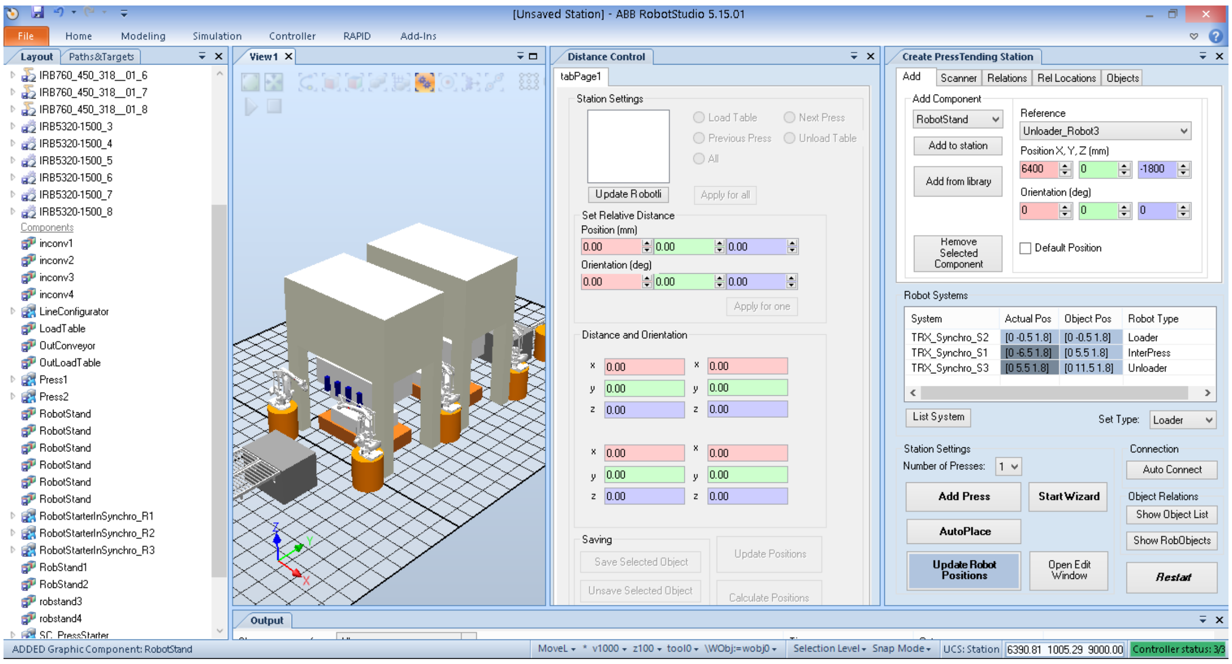


Figure 4.2: The whole add-in in RobotStudio

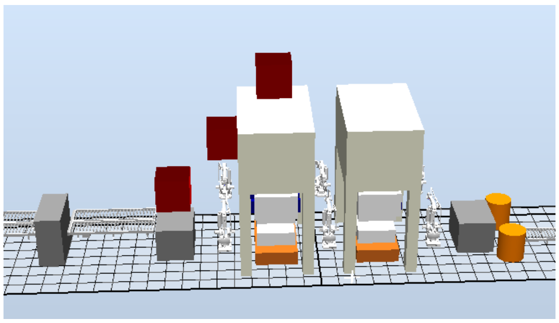
4.5 Locking down

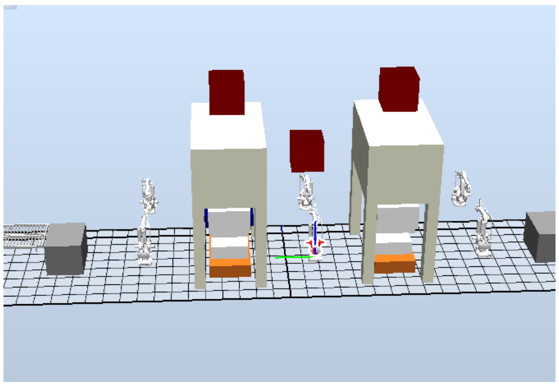
- Condition 1. In the scenario, shown in Fig. 4.3a, the robot system is locked down, the distance relationships of the locked robot are fixed and therefore its attached devices are also fixed. The sole purpose of the red boxes in the figures are to indicate if an object is in a locked state or not. A red box means that the object beneath it is in a locked state.

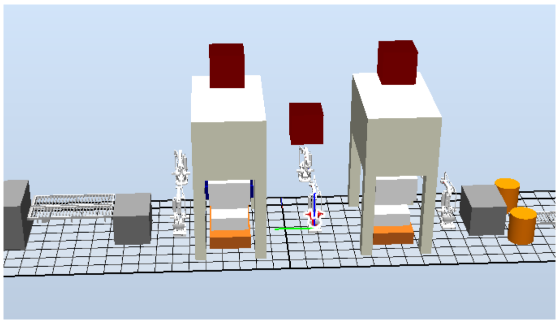
- Condition 2. In Fig. 4.3b a robot in the end of the robot chain is locked. In this scenario distance relationships regarding a robot before the locked robot in the chain are changed. In a straight build order this would mean that the locked robot would be moved according to the new distances. This would severely undermine the lock-function since one of the conditions were that a locked object should not move under any circumstances. Due to the backtracking function the problem is avoided and instead of moving the locked robot the entire chain of robots before it is moved.

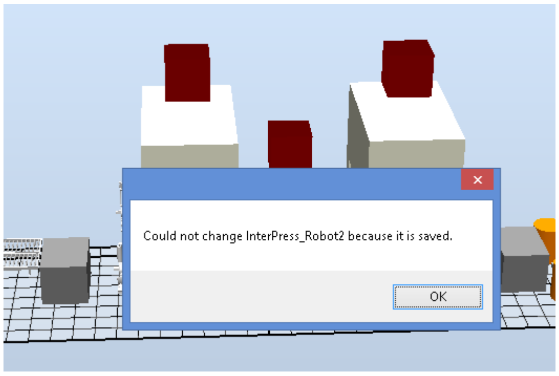
- Condition 3. Fig. 4.3c shows a scenario where a device, a press system, is moved towards a robot and then locked down. According to the lock-function specifications the robot system the device is moved towards is put in a locked state. Note; when a robot system is locked as a consequence of moving a device, not all distance relationships owned by the robot are locked down as in Fig. 4.3a only the one belonging to the moved device.

- Condition 4. Fig. 4.3d shows the message box, indicating that the Robot System is locked down and thus it is not allowed to move it.


(a) Condition 1. When the robot system is locked down its attached devices are also locked. The distance relationships to the locked devices are fixed.


(b) Condition 2. In Fig. 4.3a a robot is locked. In this scenario distances regarding robots before in the robot chain are changed.


(c) Condition 3. A device, a press system in this scenario, is moved towards to a robot and then locked down.


Could not change InterPress_Robot2 because it is saved.

(d) Condition 4. Locked and not allowed to be moved

5 Conclusions

An add-in to ABBs RobotStudio has been developed. The add-in lets the user set-up a Press Line from scratch with existing standard components. Using these components, such as known Smart Components, I/O connections and Property Bindings are done automatically. This facilitates for further implementation and simulation. The user has the ability to add and remove components from the station in a controlled way. The save function lets the user edit the station and still have the possibility to auto-rearrange other parts of the line. The add-in has an important role , where it saves time in the set-up of a Press Line and lays the way for simulation and optimization. The add-in is programmed in a OOPL and in modules where its clear what does what. The structure of the code therefore lets future developers add methods and classes easily.

The important results of this paper can be divided into the five objective set out in the beginning.

Automatic layout generation from single Step Wizard A fully functional automatic layout has been developed and implemented. The decision was made to change the Step Wizard into a UI instead. This was done due to the fact that graphical design was not prioritized. Another reason was that the many settings was better implemented on a UI instead of a Single Step Wizard.

Automatic interconnections with standard I/O The automatic interconnections with standard I/O feature has been implemented in the UI and is functioning. Both deleting components and their connections manually and with UI has been implemented. The current solution is based on known components and their connections. Should they have different in- or outputs, the connection has to be done manually. This is something that could have been fixed if more time was available.

Solution based on Smart Components and standard ABB libraries The solution has been implemented based on Smart Components and standard ABB libraries. More knowledge of the logics behind the Smart Components and the algorithms for running the simulation would have been interesting.

Possibility to import new components in a controlled way The solution has been with the possibility to add new components to each already existing components in the robot cell. The new components can have any existing component as a reference point and much like the automatic layout, it is based on defined relationship regarding purpose and distance. A custom graphical interface for choosing the different standard components and libraries was something wanted. It is possible to import components through the standard folder way.

Possibility to lock down components The possibility to lock a component at its current position and distance relationships has been implemented. This was an added feature by us and solved problems that would otherwise be present.

The outcome of the thesis is a well thought out and working add-in, with that said there are a couple of things that could have made it even better. A more planned out structure of working would possibly have made the work easier, but was hard to adapt because of the simultaneous learning curve that was present. Early information and data is as mentioned in the report essential when doing VC. This is something that needs to be considered in future developments and is more discussed in next section.

6 Recommendations and future work

6.1 Implement Automation Markup Language

Automation Markup Language, AutomationML, is a data format based on XML used for storage and exchange of information mostly concerning engineering. Its purpose is to create and maintain a standard which greatly simplifies the process of information storage and exchange in projects. Having this data in a structured way would make it possible to automate the procedure even further, which is one of the important goals for efficiency in VC

6.2 Implement dynamic properties for Smart Components

As previously mentioned in Sec.2.5.9 every Smart Component has dynamic properties. This property value is used by Code Behind to control the behaviour of the Smart Component. It can be used to change the physical lengths of a component, such as width. This would greatly benefit the simulation and modelling of press lines, when the user can change the size of for example the die parts in a press to get different sized output. The die is a specialized tool used in manufacturing to cut or shape materials using a press.

6.3 Remodel UI into Single Step Wizard

As previously mentioned in Sec.5 the end result of the add-in is a UI instead of a single step wizard due to the fact that the many settings and functions created was better implemented on a UI instead of a Single Step Wizard for this project. Nevertheless a Single Step Wizard was in the requirements. The amount of work needed to create a wizard is rather limited since the necessary functions already exists and have been implemented. There is also an aesthetic matter when designing the wizard which will need work.

References

[1] Executive summary. World Robotics 2012 Industrial Robots.
Available through: International Federation of Robotics. [Accessed 28 May 2013]
www.ifr.orguploadsmedia WR_ Industrial_ Robots_ 2012_ Executive_ Summary.pdf

[2] S. Makris, G. Michalos, G. Chryssolouris, (2012) *Virtual Commissioning of an Assembly Cell with Cooperating Robots.* Advances in Decision Sciences.

[3] N.K.Nia, (20122 *Efficient Simulation and Optimization for Tandem Press Lines* . Thesis for the Degree of Licentiate of Engineering. Department of Signals and Systems. Automation Research Group. Chalmers University of Technology.

[4] B. Svensson, F. Danielsson, B. Lennartson, (2009) *Simulation Based Optimization of a Sheet-Metal Press Line* 978-1-4244-2728-4.IEEE

[5] Tweedy K (1994) *Automating the press line.* Production vol. 106, no. 2, pp. 36.

[6] ABB AB. Application Manual, (2010) *PC SDK Application Manual.* Document id 3HAC036957-001. ABB Robotics Products. Västerås. Sweden.

[7] A. Pfeiffer, B. Kádár, L. Monostori, (2003) *Evaluating and improving production control systems by using emulation*, Twelfth IASTED International Conference on Applied Simulation and Modelling, ASM 2003, September 3-5, 2003, Marbella, Spain, pp.:261-267.

[8] D. Vilacoba, ABB Spain. *Developing requirements*

[9] Z. Liu, N. Suchold, C. Diedrich, (2012) *Virtual Commissioning of Automated Systems.* ISBN: 978-953-51-0685-2, InTech, DOI: 10.577245730. [Accessed 20 September 2013]
http:www.intechopen.combooksautomationvirtual-commissioning-of-automated-systems.

[10] P. Hoffmann et al, (2010) *Virtual Commissioning of Manufactoring Systems a Review and New Approaches for Simplification.* Proceedings 24th European Conference on Modelling and Simulation. [Accessed 25 May 2013]
http:www.scs-europe.netconf ecms2010 2010%20accepted%20papersibs_ ECMS2010_ 0041.pdf.

[11] F. Rasheed, (2006) *C# School.* [e-book]. Fuengirola Spain. Synchron Data. Available through: Programmers Heaven website. [Accessed 4 April 2013]
www.programmersheaven.comebookscsharp_ ebook.pdf

[12] BB. Wu, (1995) *Object Oriented systems analysis and definition of manufacturing operations*, International Journal Of Production Research, vol. 33, no. 4, pp. 956, Business Source Premier, EBSCOhost.

[13] X. Hu, J. Xue, H. Liu, (2011) *Research of Architecture Pattern Based on .NET Distributed System.* 2011 International Conference on Mechatronic Science, Electric Engineering and Computer August 19-22, 2011, Jilin, China

[14] Microsoft Developer Network, Microsoft Application Architecture Guide, 2nd Edition - 2009. Design Fundamentals - *Chapter 6: Presentation Layer Guidelines.* [Accessed 15 July 2013] *http:msdn.microsoft.comen-uslibraryee658081.aspx.*

[15] Microsoft Developer Network, Microsoft Application Architecture Guide, 2nd Edition - 2009. Design Fundamentals *Chapter 11: Designing Presentation Components.* Accessed 15 July 2013]. *http:msdn.microsoft.comen-uslibraryee658100.aspx*

[16] Microsoft Developer Network, Microsoft Application Architecture Guide, 2nd Edition - 2009. Design Fundamentals - *Chapter 7:Business Layer Guidelines.* [Accessed 15 August 2013]. *http:msdn.microsoft.comen-uslibraryee658103.aspx.*

[17] Microsoft Developer Network, Microsoft Application Architecture Guide, 2nd Edition - 2009. Design Fundamentals - *Chapter 8: Data Layer Guidelines.* [Accessed 15 Juli 2013]. *http:msdn.microsoft.comen-uslibraryee658127.aspx.*

[18] ABB AB. Application Manual, (2010) *Flexpendant SDK 5.13 Application Manual.* Document id 3HAC036958-001. ABB Robotics Products. Västerås. Sweden.

[19] ABB AB. Application Manual, (2011) *RobotStudio SDK Reference Manual.* ABB Robotics Products. Västerås. Sweden.

[20] ABB AB. Robotics. IRC5 Datasheet *IRC5 Industrial Robot Controller.* [Accessed 15 September 2013]
http:www05.abb.comglobalscotscot241.nsfveritydisplayc13e1c5490c61230c125796000 515137$fileIRC5%20datasheet%20PR10258%20EN_R13.pdf.

[21] B. Siciliano, L. Sciavicco, L. Villani, G. Oriolo, (2009) *Robotics Modelling, Planning and Control.* Series: Advanced Textbooks in Control and Signal Processing. XXIV, 632p.

[22] M. MacDonald, D. Mabbutt, A. Freeman, (2010) *Pro ASP.NET 4 in VB 2010.* XML Chapter. pp 621-680.

[23] ABB AB. RobotStudio Development, Persistence. [Accessed 15 July 2013] *http:developercenter.robotstudio.comIndex.aspx?DevCenter=RobotStudio&OpenDocume nt&Title=Persistent.*

[24] ABB AB. Application Manual, (2008-2010) *Operating manual RobotStudio 5.13 2008-2010, pp 238.* ABB AB Robotics Products SE-721 68 Västerås Sweden