Mittuniversitetet
MID SWEDEN UNIVERSITY

Modelling a Scalable, Reusable and Realistic Digital Twin for Virtual Commissioning

Investigating possibilities with custom SmartComponents in ABB RobotStudio

Maxim Riabichev

Mid Sweden University
Department of Electronics Design (EKS)
Author: Maxim Riabichev, maxim.riabichev@gmail.com
Supervisor: Mazhar Hussain, mazhar.hussain@miun.se
Examiner: Johan Sidén, johan.siden@miun.se
Main field of study: Automation Engineering
Degree programme: Electrical Engineering, 180 credits

Sammanfattning

Avdelningen Advanced Manufacturing på ÅF Pöyry AB utforskar möjligheter med ”Virtual Commissioning” och ”Digital Twins”. Som ett led i detta projekt har syftet med detta examensarbete varit att visa ett sätt att utveckla en skalbar, återanvändbar och realistisk digital mekatronisk modell för den virtuella miljön i RobotStudio. Tidigare forskning har visat, i linje med ÅF Pöyry AB:s satsning, att ett av de stora hinder för att Virtual Commissioning ska kunna implementeras som standard i industrin idag är bristen av skalbara och återanvändbara digitala tvillingar. Efter en genomgång av den befintliga och föreslagna metoden för att utveckla digitala tvillingar presenteras de nödvändiga stegen för att utveckla en SmartComponent för RobotStudio, med programmeringsspråket C#. Resultaten av utvecklingen och testen har visat att den utvecklade SmartComponent är skalbar och återanvändbar: den fungerar med gripare oavsett antal fingrar och den tillåter gripning både genom att applicera tryck på plockobjektet från utsidan och insidan. Den är också realistiskt på så vis att interaktionen mellan griparen och objekten som ska plockas i den virtuella miljön beter sig och ser ut som i verkligheten. Implementeringen av den utvecklade SmartComponent är också mycket effektivare och mindre komplex jämfört med den metod som används idag på ÅF Pöyry AB. Nackdelarna med den föreslagna metoden är de extra kompetenskraven för automationsingenjörer och risken att den digitala tvillingen inte är framtidssäker.

Nyckelord: Virtual Commissioning, Digital Twin, RobotStudio, SmartComponent, CodeBehind, SDK, Visual Studio, .NET, C#, API

Abstract

The Advanced Manufacturing section at ÅF Pöyry AB is exploring the possibilities of virtual commissioning and digital twins. As part of this exploration, this thesis sets out to demonstrate a method of developing scalable, reusable and realistic digital mechatronic models – the heart of a digital twin – for the virtual environment in RobotStudio. Research has shown that one of the major obstacles to implementing virtual commissioning as a standard in industry today is the lack of scalable and reusable digital twins. This is also the experience of ÅF Pöyry AB. After reviewing existing and proposed methods for developing digital twins, this thesis explains the necessary steps for developing a SmartComponent in RobotStudio, using the programming language C#. The results show that the SmartComponent developed is scalable and thus reusable. It works with grippers with any number of fingers and allows gripping by applying pressure to the target object from both the outside and the inside. It is also realistic in the sense that the interaction between the grippers and the objects to be picked in the virtual environment behaves and looks like it does in reality. The implementation of the SmartComponent developed is much faster and less complex than the method used today at ÅF Pöyry AB. The downsides of the developed method are the added competence required of the automation engineer and the risk that the digital twin may not be future-proof.

Keywords: Virtual Commissioning, Digital Twin, RobotStudio, SmartComponent, CodeBehind, SDK, Visual Studio, .NET, C#, API

Acknowledgements

First, I would like to thank Magnus Seger, who works at ABB Robotics, for taking time from his schedule to guide me on the correct path with my thesis.

I would also like to thank Andreas Buhlin at ÅF Pöyry AB for giving me an opportunity to do my thesis at the company and to learn about the concepts 'virtual commissioning' and 'digital twin', both of which instantly captured my attention. I would also like to extend my thanks to two of the automation engineers working with virtual commissioning at ÅF Pöyry AB, namely, Erik Hagman and Atle Zvantesson. Both have been liberal with their time and support for this thesis. It is much appreciated, guys!

Finally, I would like to thank my brother Pavel Riabichev, who works at ABB Robotics, for sharing his expertise in RobotStudio and C# development with me. I truly believe that without his support and never-ending patience, this thesis would not have come to fruition.

Thank you all for your time and support!

Table of Contents

Sammanfattning ii
Abstract iii
Acknowledgements iv
Table of Contents v
Terminology vii
Definition of abbreviations and terms vii
1 Introduction 1
1.1 Background 1
1.2 Problem statement and purpose 2
1.3 Scope 3
1.4 Concrete and verifiable goals 4
1.5 Outline 4
2 Theory 5
2.1 Virtual commissioning 5
2.2 Digital twin 5
3 Existing and proposed implementation methods 7
3.1 The gripper and its function in reality 7
3.2 Mechanism 8
3.3 Interacting with virtual objects in RobotStudio 8
3.4 Existing implementation method 10
3.4.1 Motion control: SC_GripperMovement 10
3.4.2 Gripper control: SC_GripperControl 11
3.4.3 Collision detection: SC_Touchy 12
3.5 Proposed implementation method 13
4 Design 15
4.1 The basics of creating a SmartComponent 15
4.2 The logic of the developed SmartComponent 16
4.2.1 OnPropertyValueChanged() 17

4.2.2 OnSimulationStep() 19
4.2.3 A flowchart of the SC 19
5 Results 20
5.1 Theoretical functionality 20
5.2 Results from testing 23
6 Discussion 27
6.1 First goal: Identify the necessary steps for creating a digital mechatronic model 27
6.1.1 Was it possible to make the model scalable, reusable and realistic? 27
6.2 Second goal: Compare the new and existing models 29
6.3 Third goal: Evaluate whether the final result justifies the additional engineering time required. 30
6.4 Work-arounds implemented 30
6.5 Drawbacks of this method 31
6.6 Ethical and social considerations 31
7 Conclusion 32
7.1 Future work 32
References 33

Terminology

This section covers the relevant definitions of abbreviations and terms.

Definition of abbreviations and terms

Abbreviation	*Definition*
VC	Virtual Commissioning
DT	Digital Twin
RS	ABB RobotStudio
SC	SmartComponent
PLC	Programmable Logic Controller
SDK	Software Development Kit
I/O	Input / Output

Term	*Definition*
SmartComponent	A SmartComponent is a function block in RobotStudio which can be programmed to deliver different functionality to the simulation. SmartComponents can also be nested together in networks within a SmartComponent in order to perform more complex functions. SmartComponents can be used to enable/disable the simulation; jog joints of a robot; react to signals; and create 3D objects in the simulation environment, and much more.
Rapid	The programming language for the robot controller in RobotStudio.

1 Introduction

The Advanced Manufacturing section at ÅF Pöyry AB in Gothenburg is seeking to implement the virtual commissioning method when developing new solutions for its customers. At this time the company is trying to refine a standard that it can build on. This thesis addresses one piece of the puzzle, namely the simulation environment delivered by ABB Robotics with their RobotStudio (RS) software.

1.1 Background

Five years ago, ÅF Pöyry AB received an assignment to commission an upgrade to an existing production plant which included an open-hearth furnace. This plant could not be taken offline at any time prior to or during the commissioning. In order to deliver a solution that fulfilled this requirement, the engineering team at ÅF Pöyry AB were forced to find a way to verify and validate their code without access to the running production plant. They found that this could be done by creating a digital twin (DT) of the production plant and then using virtual commissioning (VC). After they delivered the solution to the customer, the section manager at ÅF Pöyry AB, Andreas Buhlin, realised that VC is the way of the future, not only for production plants which cannot be taken offline but for all plants.

In their 2014 article outlining the necessity of VC during the creation and testing of an industrial production process (see section 2.1), Mathias Oppelt and Leon Urbas stated: 'Today the testing usually takes place during the on-site commissioning within the real plant and therefore influences directly the critical path of the overall engineering process.' [1][1] If the on-site commissioning does not succeed within the set schedule – due, for example, to an unforeseen mismatch of the communication protocols between the various machine interfaces – this can cause long and costly production downtime. With VC, however, such issues can be detected and solved before on-site commissioning begins.

Since the early 1990s, the classic engineering process of creating a manufacturing plant [2] has made use of VC, but not throughout the whole engineering process. It has usually been employed during the installation/construction phase and has ended halfway through the commissioning phase [1]. The next step is, then, to integrate virtual commissioning into the whole engineering process, as illustrated in Fig. 1.

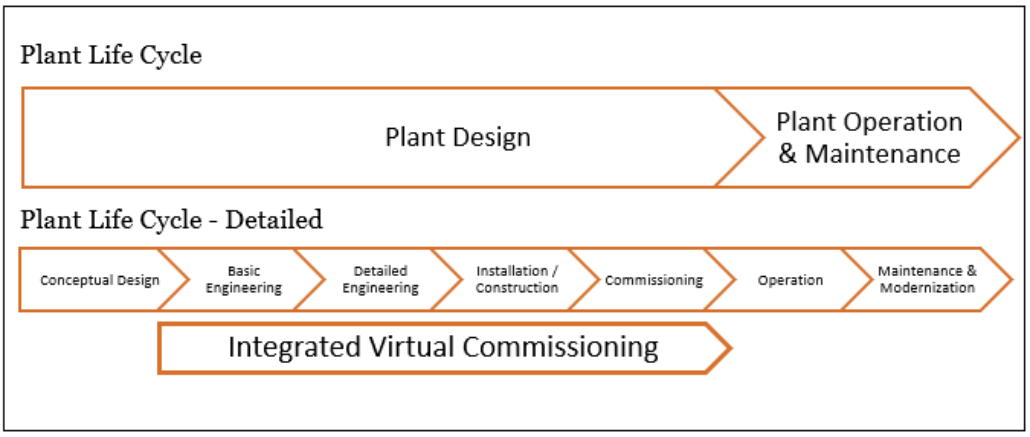

Figure 1. The engineering process of creating a manufacturing plant, with integrated virtual commissioning. [1]

The argument is that if there were a virtual model of the production cell, and if that model were detailed enough, then the communication between all the machinery within the cell could be tested off-site during the development process. This method would minimise the risk of long production stops during on-site commissioning due to potential unforeseen programming errors and would thus be more cost-effective than current commissioning methods.

Having said this, one might wonder why integrated VC is not an industry standard today, given the potential benefits. One reason is the difficulty of creating realistic, reusable and scalable virtual models, an endeavour which can be time-consuming and thus costly in itself. [1][2] This thesis aims to tackle this problem by demonstrating a method of creating such a virtual model.

1.2 Problem statement and purpose

Research has shown that one of the main hurdles to industry's adaption of integrated VC is the lack of a method for creating scalable, reusable

and realistic digital representations of an industrial production plant [1][2]. Realistic digital representations (often called digital twins) are representations of real objects (e.g. robots, conveyor belts and fittings). DTs are classically designed to represent one specific object (unique to that specific production cell), which makes their creation time inefficient and thus costly. According to Oppelt and Urbas,

> [t]he possibility to use virtual commissioning depends on the availability of a sufficient simulation model. Today these models are generally created from scratch for the individual simulation studies. The effort for building up these models can be quite high and is currently seen as one of the major hurdles to use simulation and especially virtual commissioning more broadly. [1]

The general aim of this thesis is, therefore, to investigate the possibility of creating virtual models that can be used on more than one occasion. The purpose is first to demonstrate how a custom SmartComponent (SC) for RS can be created in a .NET environment, and second to evaluate how this method compares to the standard way of implementing the multiple built-in SCs (connected to one another via various I/Os). The comparison must include complexity, scalability, and reusability (all of which affect the lead-time of engineering), and realism.

1.3 Scope

As this thesis is based on an existing project at ÅF Pöyry AB, the scope has largely been set by that project.

The thesis is focused on the tool attached to an industrial robot manufactured and delivered by ABB. The tool is a three-finger gripper (see section 3.1). A digital representation of a real object in a virtual environment (in RS) has three main parts: (1) a three-dimensional CAD model, (2) a mechanism, and (3) a digital mechatronic model. The digital mechatronic model is used to animate the mechanism in the virtual environment by reacting to signals sent to the virtual object. This thesis will focus solely on the digital mechatronic model of the object. The 3D model and the mechanism were created and supplied to this thesis by ÅF Pöyry AB.

The simulation software is RS. RS is coded mainly in C#, which is why the creation of the digital mechatronic model, the SC, is in a .NET environment coded in C#.

1.4 Concrete and verifiable goals

1. What are the necessary steps for creating a digital mechatronic model of a gripper in the .NET environment, and can it be made to be scalable, reusable and realistic?
2. How does the new model compare to the one currently in use in terms of ease of development, complexity, and efficiency during implementation?
3. Does the final result justify the additional engineering time (and cost) required to create such a model?

1.5 Outline

This thesis report has the following outline: The next section covers the relevant theoretical background. Section 3 describes the current method and the proposed method. The design of the SC is described in section 4. Section 5 presents the results, followed by a discussion in section 6. The report ends with a conclusion in section 7.

2 Theory

This section reviews the two core concepts for this thesis based on previous research.

2.1 Virtual commissioning

Virtual commissioning is the ability to commission a plant without direct access to the plant itself. This concept has already been touched on in 1.1. Because VC is still relatively new it can be understood and applied differently depending on the interpreter and the user. In some areas of industry, it may not be used at all, while in others it may be understood as something that is performed just before the real commissioning for validation purposes. In other cases, it may be understood as integrated VC which is performed in parallel with the development of the production process. [1][2]

ÅF Pöyry AB (Industry Division, Advanced Manufacturing Section) has taken the concept one step further so that it is now possible to speak of 'Real Virtual Commissioning'. The company is developing a work flow based on a DT of the whole production process which has such fidelity that all the code for the process control (PLC code and robot controller for example) can be written and constantly validated against the DT. This means that the engineers who wrote the robot controller code can verify that their code works with the PLC program throughout the development phase instead of, as it is usually done today, just before or during the real commissioning. The level of detail of the DT required to reach this state is high but possible, which is why the term 'real' is used. The company aims to have a development process which makes possible writing code which needs no change when going from the virtual environment to the physical.

2.2 Digital twin

Kritzinger et. al. [3] conducted a categorical literature review on the concept of DT in manufacturing. According to them, the first definition of DT dates to 2002 when it was used by Michael Grieves in the context of an industry presentation on product lifecycle management. However,

the more widely recognised definition in an industrial context comes from Glaessgen and Stargel in 2012:

> The digital twin is an integrated multi-physics, multi-scale, probabilistic simulation of a complex product and uses the best available physical models, sensor updates, etc., to mirror the life of its corresponding twin. [3]

While this definition is rather broad, the most relevant aspect for the thesis at hand is that to succeed with real VC, a DT "must mirror the life of its corresponding twin".

The concept of a DT can be used on different abstraction levels. At a high abstraction level, a DT can refer to the whole production cell including equipment such as robots, conveyor belts, buttons and lights. At a low abstraction level, a DT can refer to just one of these items, such as one conveyor belt or a light. This distinction is relevant to this thesis, which focuses on a scalable, reusable and realistic digital twin of a gripper for an industrial robot that is part of a production process. This represents a low level of abstraction. Most previous research has been conducted at higher abstraction levels, looking at DTs of whole production cells, for example, and automatically generating communication interfaces based on available documentation [4]–[6]. The only work which is reasonably similar to the work in this thesis created a DT of a CNC machine tool [7]. However, the complexity of their tool is much greater than the complexity of the gripper in this study.

3 Existing and proposed implementation methods

To control a gripper in a virtual environment, the 3D object must first be converted to a mechanism, and then a digital mechatronic model must be created.

This section covers the method developed by ÅF Pöyry AB (subsection 3.4), and the method developed within this thesis (subsection 3.5).

3.1 The gripper and its function in reality

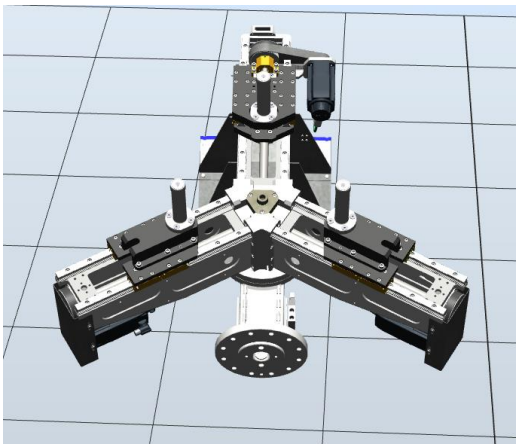

Figure 2. The gripper as viewed from underneath in RS. This 3D model was provided by ÅF Pöyry AB.

The gripper (Fig. 2) in this case is a multi-purpose tool. Its main function, which is the one relevant to this thesis, is to pick up different sized 'ring-shaped' objects (see Fig. 3 for an illustration). It can pick them up by applying pressure either from the inside or the outside. For this task it has three fingers which are controlled by three servo motors. The servo motors are controlled by analogue signals from a PLC.

As analogue signals are sent to the gripper, the PLC waits for a certain amount of torque to build up in one of the axes, which happens when the fingers press against the object. This is the way it discerns whether something has been grasped by the fingers of the gripper. If the fingers reach a certain position without reaching the torque threshold, this indicates that the robot has missed the target object and a new attempt must be made.

Mimicking such behaviour in a virtual environment requires a number of work-arounds because it is not possible to simulate all features. For example, there is no way to simulate torque in RS.

3.2 Mechanism

For a 3D model to be able to move in the virtual environment in RS, a mechanism must first be created. The mechanism dialogue can be found under the '*Modeling*' tab in RS. It is beyond the scope of this report to cover the creation of mechanisms in RS, but it is relevant to mention that it is in this dialogue that the *Joints* of the mechanism are defined. These joints are later used both by the current solution and by the solution presented in this thesis.

For this thesis the 3D model of a gripper was provided by ÅF Pöyry AB with the mechanism that allows it to move.

3.3 Interacting with virtual objects in RobotStudio

Currently in RS version 6.08.01 the gripper is able to pick up an object in the virtual environment in two ways. This can be done using physics, where friction is simulated between two bodies, or with the help of a feature called *Attach*. This feature allows an object to become bound to another object in the virtual environment and move with it as if they were one.

If the objective is to create a realistic simulation, then the gripping should be performed using physics (Fig. 3).

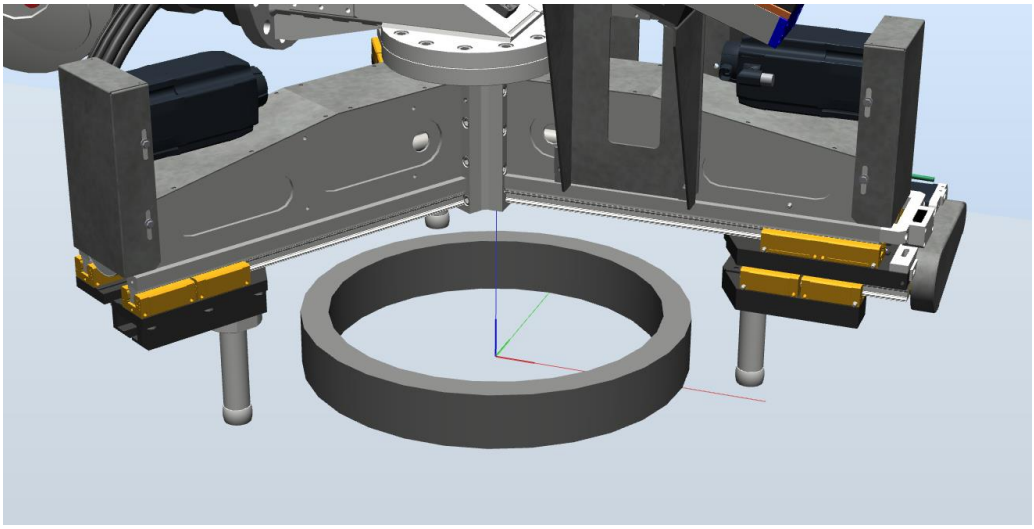

Figure 3. The gripper and an object, in this case a ring.

When a finger of the gripper pushes against a side of the object a collision should occur and the object should move, sliding on the surface it lies on according to the force applied by the finger. If all three fingers come into contact with the object (the ring) and if the fingers are stopped in that position, the gripper should be able to lift the ring from the surface it is resting on. This is possible with the physics simulator built into RS.

However, the simulation of physics in RS is neither realistic nor reliable enough. One of the issues is that, according to information from ÅF Pöyry AB and supported by tests performed within this thesis, the level of repeatability is very low. Sometimes objects react in an unpredictable manner to predictable movements. For example, sometimes objects shoot upwards when they come into contact with one another and sometimes they do not. Another issue is that collisions between objects based on physics register before the bodies actually intersect. For example, a finger can be a few millimetres away from the ring and yet the ring will act as if it is in contact with the finger. This behaviour persists even if one selects '*Use original geometry*' for collision geometry in both the finger and the ring (Fig. 4).

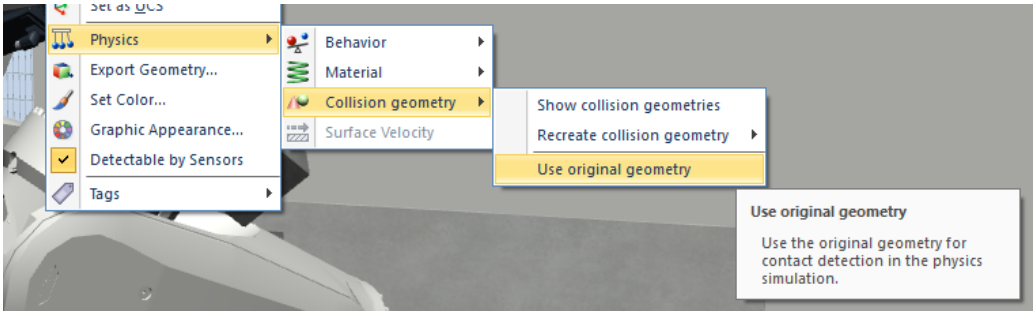

Figure 4. Use original geometry for contact detection in the physics simulation in RS.

A work-around has been found to overcome the deficiencies in the physics simulation in RS and has been implemented by ÅF Pöyry AB in their current solution and by this thesis. More detail is given in the Design section, section 4.

3.4 Existing implementation method

The existing digital mechatronic model developed by ÅF Pöyry AB which controls the gripper mechanism in the virtual environment was created using multiple built-in SCs in RS. There are three overarching SCs, one for movement control, one for gripper control, and one to indicate when a finger is near the object being picked up.

3.4.1 Motion control: SC_GripperMovement

The motion of the four axes of the gripper is controlled by the SC called *SC_GripperMovement*, which itself contains another SC called *JointMover*. As shown in Fig. 5, four analogue inputs are connected to the *JointMover* SC (named Axis in Fig. 5). Axes 1 through 3 (named J1–J3 in the SC) control the three fingers of the gripper. Axis 4 (J4 in the SC) controls the extra movement capability in axis 3. The SC accepts setpoint values in metres. The digital input signal named *diMove* tells the SC when to move the axes. When this signal is high, the SC will attempt to make the current position equal to the setpoint. Finally, the digital output signal *doExecuting* sends feedback back to the PLC whenever the SC is in an executing state, that is, whenever the current position is not equal to the setpoint.

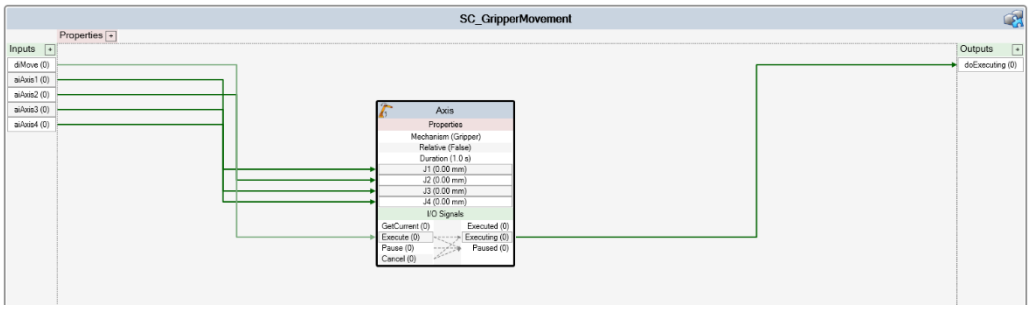

Figure 5. The SC which controls the movement of the four axes of the gripper. Name: SC_GripperMovement

It is noteworthy that the *JointMover* SC automatically creates the correct number of analogue inputs according to the number of movable joints in the mechanism (in this case J1–J4) selected in the SC. It also configures the range of allowed movement per joint in millimetres, according to the input values when the mechanism is created.

3.4.2 Gripper control: SC_GripperControl

This SC, *SC_GripperControl* (Fig. 6), contains multiple sub-SCs. The *RingFinder* is a *LineSensor*, there is a *Detacher* and an *Attacher*, a *LogicSRLatch* and a *PhysicsControl*.

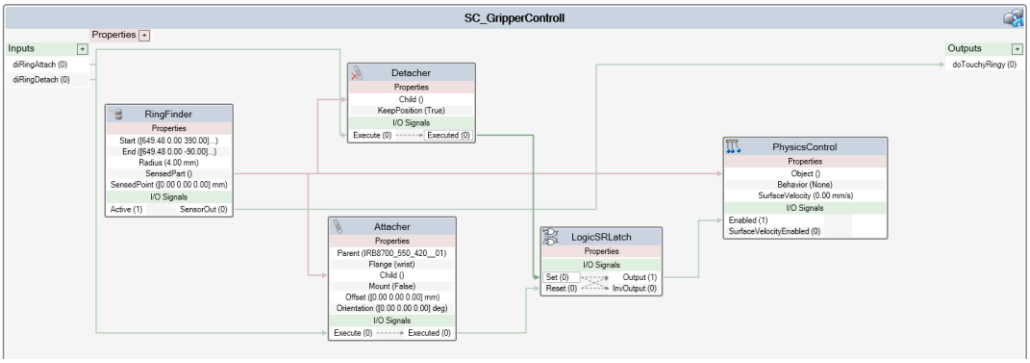

Figure 6. SC_GripperControl. This SmartComponent controls the attachment and detachment of the object picked by the gripper.

The *RingFinder* SC plays a vital role in the simulation. It is located at the centre point of the gripper (Fig. 7) and its purpose is to 1) indicate that the fingers of the gripper surround an object and 2) identify this object and set it as a 'child' in the *Attacher*, *Detacher* and *PhysicsControl* SCs. The digital output signal *doTouchyRingy* is set high when the *RingFinder* touches an object. The PLC will then begin the pick-up procedure in *SC_MovementControl*. Note that there is no such laser sensor in reality.

The RingFinder SC is a work-around implemented to create a working digital twin.

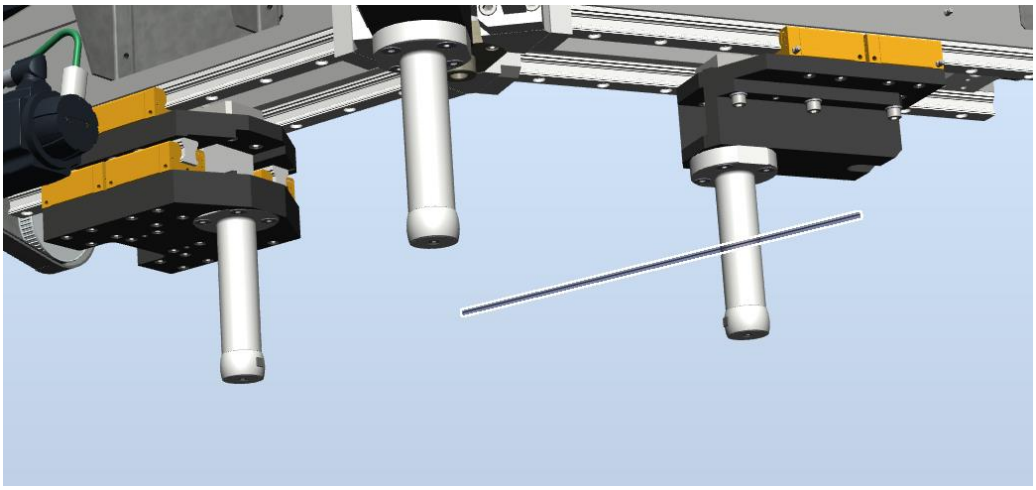

Figure 7. The SC LaserSensor, named RingFinder. It is located at the centre of the gripper and indicates when an object is in position for pick-up.

3.4.3 Collision detection: SC_Touchy

As the fingers move toward the object, there is one more SC, a *VolumeSensor*, placed on the finger with high precision movement capability (grey vertical pillar to the right of the finger in Fig. 8). This SC

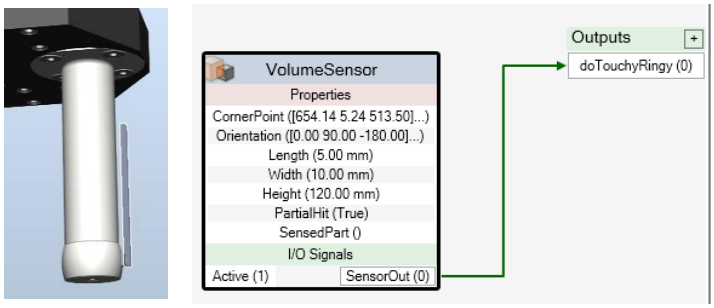

Figure 8. SC_Touchy, a volume sensor placed on the finger with high precision movement capability.

sets the digital output signal *doTouchyRingy* to high when the finger has moved close enough to the object, indicating that the object is ready to be picked up. When this happens, the PLC enables *diRingAttach* in the *SC_GripperControl* and the object is then attached to the flange of the robot itself. When the attachment occurs, the physics are also disabled using the *SC PhysicsControl* in *SC_GripperControl*. This is done to prevent the unpredictability of the physics simulation mentioned in section 3.3. The physics are re-enabled when the object has been detached from the robot, which happens when the object has been placed.

Finally, it is important to note that the current implementation does not allow objects such as rings to be picked up from the inside. This is because the collision between the object and the fingers is actually detected between the object and the small dark grey-coloured bodies attached to the insides of the tips of the fingers (Fig. 7). The fingers themselves do not currently have physics enabled. If a pick-up from the inside were to be attempted, the fingers would be able to pass through the geometry of the ring without consequence.

3.5 Proposed implementation method

The aim of this thesis is to develop a solution for controlling a gripper via an SC which is scalable, can be reused in future projects, and allows for realistic simulation scenarios within RS. The solution should also be less complex to implement than the current one in order to save engineering time and thus lower the cost of virtual commissioning. Figure 9 shows the developed SC, as viewed in RS.

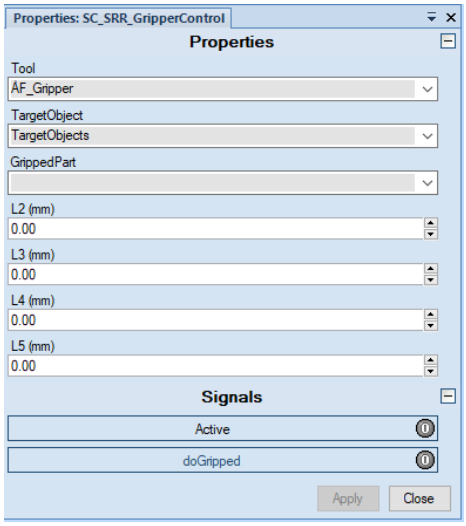

Figure 9. The developed SC: scalable, reusable and realistic gripper control.

To implement the new SC the following steps are required:

- In RS: The physics behaviour of the fingers of the gripper must be set to *Kinematic* (otherwise the fingers will not be part of the physics simulation).
- In RS: A custom physics material must be created and applied to all fingers. Suggested values can be found in Fig. 10 below. The

reason these specific values were chosen is covered in the discussion section.

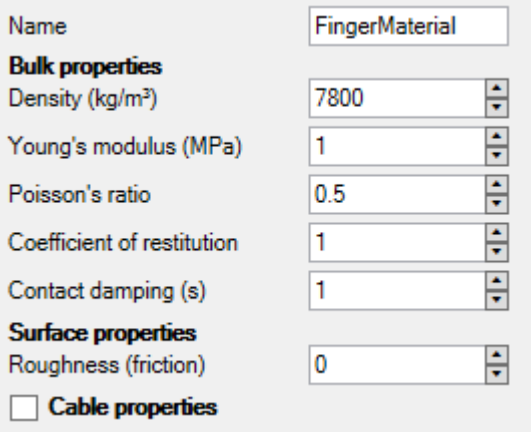

Figure 10. Custom physics material for fingers of the gripper. This is a required step for reliable and realistic physics interaction between objects in RS.

- In the SC: A tool must be selected (this generates the input properties L2–L5).
- In RS: a component group containing all objects that will be picked must be created. Objects that the picker is supposed to pick up can be 'sourced' into this group by setting it to *Parent* in the *Source* SC.
- In RS: the physics behaviour of the component group must be set to *Dynamic*. This behaviour is inherited by all items in this group and allows for all children to be part of the physics simulation.
- In the SC: The component group must be selected in the *TargetObject* dropdown.
- In RS: Under Simulation -> Station Logic, connect the analogue input properties to the output of the controller of your choice.
- In the SC: The digital input signal *Activate* should be connected to a PLC for example, to activate the SC. Manual activation is also possible, that is, the SC will always be enabled and start to function as soon as the simulation starts.

The complete functionality of the SC developed is described in the Results section.

4 Design

In this section the logic behind the SC and the steps taken to develop it will be reviewed. The following software was used:

- Microsoft Visual Studio (VS)
- RobotStudio SDK[1] (RS SDK)
- ILSpy[2]
- ABB RobotStudio

ABB provides instructions on their website[3] on how to set up the development environment and offers a basic guide on how to create a SC.

ILSpy is open-source software that can decompile dll-files (Dynamic-link library). This software made it possible to learn from and reuse libraries, classes and methods developed by ABB Robotics for the built-in SC's such as *JointMover* and *Attacher*. ABB Robotics were kind enough to pin-point the relevant dll-file for this thesis: 'RobotStudio.Services.Modeling.dll'.

4.1 The basics of creating a SmartComponent

When creating an SC project, a few files are generated automatically by the interaction between RS SDK and VS. The two most important files are CodeBehind.cs and the <ProjectName>.xml.

The CodeBehind.cs file contains the C# code for the SC. The three main methods are called by different events in RS. Figure 11 shows a snippet from this file.

[1] http://developercenter.robotstudio.com/robotstudio
[2] https://github.com/icsharpcode/ILSpy
[3] http://developercenter.robotstudio.com/robotstudio/gettingstarted

```
namespace SC_SRR_GripperControl
{
    /// <summary> Code-behind class for the smartComponentTest Smart Component.
    4 references | Maxim Riabichev, 8 days ago | 1 author, 5 changes
    public partial class CodeBehind : SmartComponentCodeBehind
    {
        /// <summary> Called when the value of a dynamic property value has changed.
        0 references | Maxim Riabichev, 45 days ago | 1 author, 3 changes
        public override void OnPropertyValueChanged(SmartComponent component, DynamicProperty changedProperty, Object oldValue)...

        /// <summary> Called when the value of an I/O signal value has changed.
        0 references | Maxim Riabichev, 49 days ago | 1 author, 1 change
        public override void OnIOSignalValueChanged(SmartComponent component, IOSignal changedSignal)...

        /// <summary> Called during simulation.
        0 references | Maxim Riabichev, 8 days ago | 1 author, 3 changes
        public override void OnSimulationStep(SmartComponent component, double simulationTime, double previousTime)...
```

Figure 11. The CodeBehind.cs file and its main methods. The <summary> contains a description of when each method is called.

The <ProjectName>.xml file contains the definition of the properties and the signals of the SC (as shown in Fig. 12). Comparing the code in the snippet with the illustration of the SC in RS (Fig. 9) shows the location of each of these properties.

```
<SmartComponent name="SC_SRR_GripperControl" icon="SC_SRR_GripperControl.png"
            codeBehind="SC_SRR_GripperControl.CodeBehind,SC_SRR_GripperControl.dll"canBeSimulated="true">

  <Properties>
    <DynamicProperty name="Tool" valueType="ABB.Robotics.RobotStudio.Stations.Mechanism">
      <Attribute key="AutoApply" value="true" />
    </DynamicProperty>
    <DynamicProperty name="TargetObject" valueType="ABB.Robotics.RobotStudio.Stations.GraphicComponentGroup">
      <Attribute key="AllowedTypes" value="ABB.Robotics.RobotStudio.Stations.GraphicComponent"  />
      <Attribute key="AutoApply" value="true" />
    </DynamicProperty>
    <DynamicProperty name="GrippedPart" valueType="ABB.Robotics.RobotStudio.Stations.Part" readOnly="true" />
    <!--<DynamicProperty name="PreventFingerMovementIfGripped" valueType="System.Boolean" value="false" />-->
  </Properties>

   <Signals>
    <IOSignal name="Active" signalType="DigitalInput" />
    <IOSignal name="doGripped" signalType="DigitalOutput" />
  </Signals>
```

Figure 12. A snippet of the .xml file of the SC developed.

A note on the .xml file: In order to simulate the SC in RS, the attribute *canBeSimulated="true"* must be added to the element *SmartComponent*, as in the documentation provided by ABB Robotics online.

Finally, properties and signals defined in the .xml file can be overridden by the code in CodeBehind.cs. This means that properties can be created and removed via CodeBehind.cs, irrespective of whether they exist in the .xml or not.

4.2 The logic of the developed SmartComponent

The inspiration for the logic underlying the digital mechatronic model, the SC, comes from knowledge about the gripper's functionality in reality. The aim is to mimic its function in the virtual environment as closely as possible. As previously mentioned, not everything can be

simulated; work-arounds must be implemented. The flow chart of the SC can be reviewed in Appendix A.

This subsection describes the two main methods of the CodeBehind.cs file used in this thesis. Many of the methods used are available in libraries written by ABB Robotics. These methods will not be discussed in detail because they are available in the ABB documentation.

4.2.1 OnPropertyValueChanged()

```
public override void OnPropertyValueChanged(SmartComponent component, DynamicProperty changedProperty, Object oldValue)
{
    if (!component.StateCache.ContainsKey("BasePropertyCount"))
        component.StateCache.Add("BasePropertyCount", component.Properties.Count); //This is used to keep track of the properties which are defined in the .xml.

    FindMovableLinks(component, changedProperty);

    component.StateCache.TryGetValue("BasePropertyCount", out var basePropertyCount);

    bool isJointPositionChanged = false;
    foreach (DynamicProperty prop in component.Properties)
    {
        if (prop.Name == changedProperty.Name)
        {
            isJointPositionChanged = true;
            break;
        }
    }

    if (isJointPositionChanged)
        IsAllFingersInCollision(component);
}
```

Figure 13. OnPropertyValueChanged method. This is called every time a property is changed in the SC.

When this method is first called (see snippet in Fig. 13) a number is stored in the *StateCache* dictionary under the '*BasePropertyCount*' key. This number represents the number of properties predefined in the <*ProjectName*>.xml file.

The *StateCache* is a collection that must be used as explained by ABB in the remarks for the *CodeBehind* class (Fig. 14).

> The code-behind class should be seen as a service provider used by the Smart Component runtime. Only one instance of the code-behind class is created, regardless of how many instances there are of the associated Smart Component.
> Therefore, the code-behind class should not store any state information. Instead, use the SmartComponent.StateCache collection.

Figure 14. A snippet of the remarks for the CodeBehind class, explaining the need for StateCache.

This step happens only once per runtime and allows the code to ignore the default properties a) when the input parameters are generated, and b) during collision detection. This latter makes the code tighter and faster.

Thereafter the software searches for the movable links of the mechanism selected in the Tool dropdown property of the SC (method

FindMovableLinks). Movable links previously configured in the mechanism are added to the SC as properties with their respective ranges and types (rotation/prismatic). When a movable link is found, it is added to the property list and is intended to be used as an analogue input.

This method also defines the *BaseLink* of the mechanism as *AttachBase* key in the *StateCache,* which is then used as the attachment point of the gripper. It also identifies which of the movable links is a finger by looking at their *PhysicsMotionControl* property. If a link has the physics behaviour *Kinematic* set, then this link will be regarded as a finger of the gripper in the SC.

The way this method is constructed creates the potential for the SC to be dynamic, and thus scalable, because it allows the SC to be used for a gripper with any number of fingers.

The next function to be called is *IsAllFingersInCollision*. This function is called only if the property changed is one of the generated analogue input properties. The ABB-provided method *CheckCollision* checks each finger for collision with any child of the *TargetObject* component group. Only if all fingers are in collision with the same object will the object be attached (via *Attach* method) to the *BaseLink* of the gripper. The digital output signal *doGripped* will be set high, and the object will be available in the *GrippedPart* property as an output. The *GrippedPart* property allows the SC to share information about what it is currently carrying (as its 'child') with other SCs if this is necessary.

If the collision no longer exists, that is, if at least one of the fingers has moved away from the object, the method *Detach* is called and the object is detached, the *GrippedPart* property is set to null, and the digital output signal *doGripped* is set low.

The *Attach* and *Detach* methods were inspired by the code for the built-in SCs with the same names but were rewritten to suit this thesis.

4.2.2 OnSimulationStep()

```
public override void OnSimulationStep(SmartComponent component, double simulationTime, double previousTime)
{
    if ((int)component.IOSignals["Active"].Value == 0) return;

    MoveJoint(component);
}
```

Figure 15. OnSimulationStep method. It is called on every simulation step (refer to RS manual for definition). Each step occurs every 12ms (time can be changed in options in RS).

This method first checks whether the SC is active (Fig. 15). If it is, the *MoveJoint* method is called.

The *MoveJoint* method is based on the code developed by ABB in the SC called *JointMover* but has been partly rewritten to suit this thesis. This method uses the built-in *SetJointValues* method to set current joint values to the values received via the analogue input of the SC.

4.2.3 A flowchart of the SC

The flow chart of the SC can be reviewed in Appendix A.

5 Results

This section first covers the theoretical functionality achieved by the developed SC and then the results from the tests performed in the virtual environment within RS.

5.1 Theoretical functionality

The developed SC described in the Design section allows all the functionality achieved by the current method used at ÅF Pöyry AB and eliminates the need for all other SCs (cf. section 3.4). In addition, it expands functionality by allowing objects to be picked up from the inside without the need of any additional SCs.

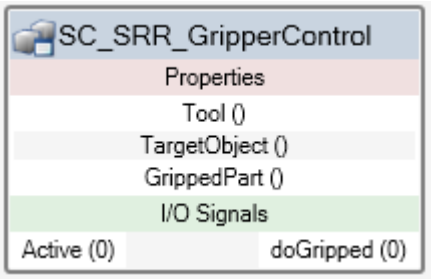

Figure 16. SC_SRR_GripperControl: The developed SC for controlling a gripper with any number of fingers.

The SC developed in this thesis is illustrated in Fig. 16 above and allows for the following features in the virtual environment:

- Automatically adjusts the number of available inputs, as properties with the correct range of motion, according to the number of movable links in the selected mechanism. The inputs are named according to the links (Fig. 17) in the mechanism.

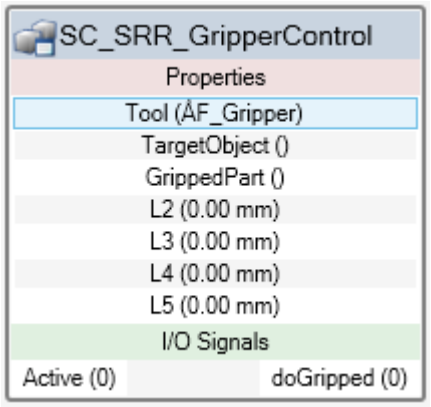

Figure 17. The template gripper supplied by ÅF Pöyry AB has been selected as a Tool. The L2–L5 properties were automatically generated.

- Each input property accepts discrete values, in the SI unit meter. Each link moves without regard to others, all according to the signals received from the PLC, for example. At this time interpolation between two positions A and B is not available.
- Selects a group of objects that the gripper is allowed to pick up (see Fig. 18).

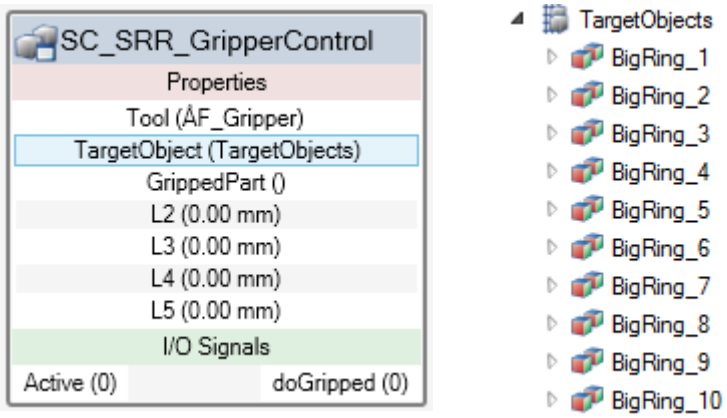

Figure 18. The Component Group 'TargetObjects' was selected in the TargetObject property of the SC.

- Detects collisions between all fingers and an object that is part of the component group selected in *TargetObject* property. Only those links which have been configured as *Kinematic* in the Physics behaviour are included in collision detection (Fig. 19).

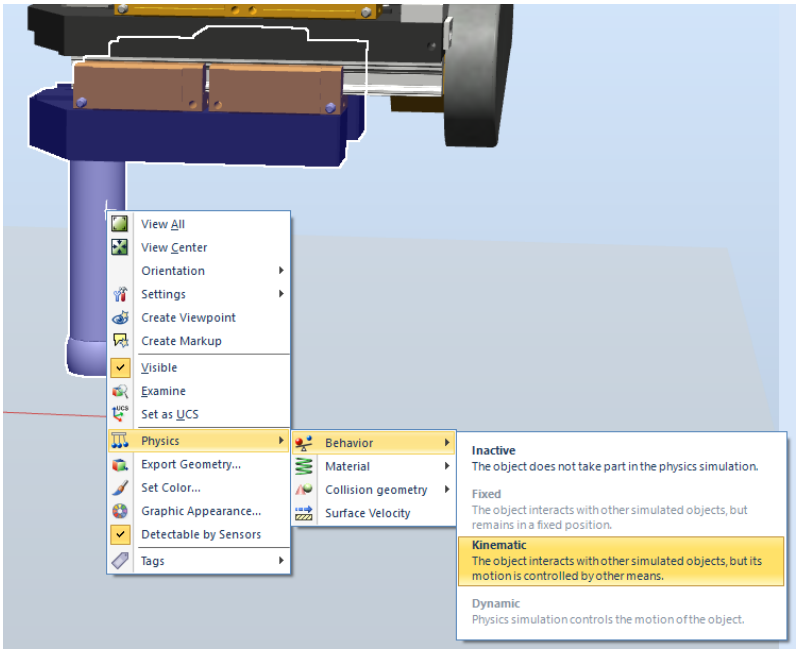

Figure 19. The selected finger is configured to have the kinematic physics behaviour, which includes it in the collision detection built into the developed SC.

- If all fingers are in collision with the same object, that object is shown as *GrippedPart* property, *doGripped* is set to high, and the object is attached to the *BaseLink* (L1 by default) of the gripper.
- If the object being picked up is off-centre, the finger which first comes into contact with it will push on it, using physics, according to the point of impact and direction of the movement.

5.2 Results from testing

Figure 20 shows the simple testing rig created in RS. In this rig the robot is programmed to move the ring from one pallet to the other. When moving the ring from the right pallet to the left, the gripper is supposed to grip the ring from the outside. The same applies when moving from left to right.

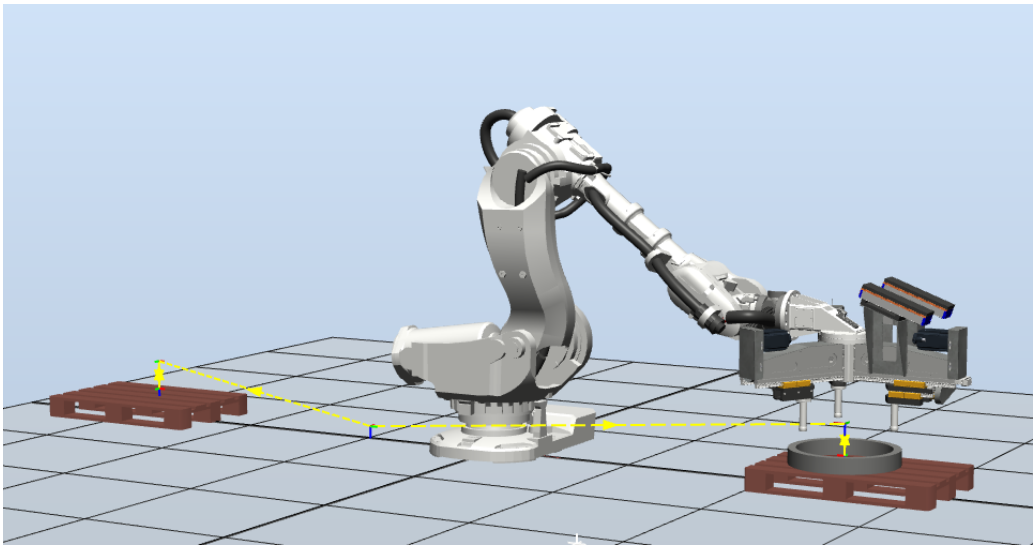

Figure 20. The testing rig: one robot, two pallets, one ring and the gripper. Paths and targets are shown in yellow.

The full movement of the robot, from right to left and back again, is split up into four segments (Fig. 21). Each segment is initiated by setting the

```
PROC Init()
    WHILE TRUE DO
        IF movePickOutside=1 THEN
            Path_PickOutside;
        ELSEIF movePlaceOutside=1 THEN
            Path_PlaceOutside;
        ELSEIF movePickInside=1 THEN
            Path_PickInside;
        ELSEIF movePlaceInside=1 THEN
            Path_PlaceInside;
        ELSEIF fullAutoRun=1 THEN
            WHILE fullAutoRun=1 DO
                Path_FullAutoRun;
            ENDWHILE
        ENDIF

        Main;

    ENDWHILE

    WaitTime 1;
ENDPROC
```

Figure 21. The Init procedure: A continuous loop which is waiting for certain signals to trigger.

digital output signal corresponding to the respective segment to high.

The digital output signals (for example, *movePickOutside*) were created manually in the I/O System of the robot controller.

The signals are controlled manually via the I/O Simulator available in the Simulation tab of RS.

Finally, the signal *fullAutoRun* calls for a procedure that runs through all four segments continuously until the signal is manually set to low or the simulation is stopped.

Figure 22 shows the path which represents the position of the robot in Fig. 20. Besides moving the robot to the various targets, it sets the position of the fingers of the gripper to 330 mm. This specific value sets the positions of the fingers slightly beyond the outer diameter of the ring and is only here for testing purposes. In reality the position can be calculated and set by other means, for instance, from known data or laser sensor values.

```
PROC Path_PickOutside()
    SetDO movePickOutside,0;
    MoveJ Target_AppPickOutside,vmax,fine,tcpGripper\WObj:=wobj0;
    SetAO aoAxis1,330;
    SetAO aoAxis2,330;
    SetAO aoAxis3,330;
    MoveL Target_PickOutside,v1000,fine,tcpGripper\WObj:=wobj0;
    DecrementJointValueOutside;
    MoveJ Target_AppPickOutside,vmax,fine,tcpGripper\WObj:=wobj0;
ENDPROC
```

Figure 22. An example of one of the Path procedures.

The movement of the fingers of the gripper is controlled by code in Rapid (Fig. 23), which simulates signals sent from a PLC, for example. As long as the *diGripped* is high (indicating that the gripper is carrying something, in this case the ring) the position of the fingers will increment by 1 mm every 200 ms, widening the grasp.

```
PROC IncrementJointValueOutside()
    VAR num Axis1;
    VAR num Axis2;
    VAR num Axis3;
    WHILE diGripped=1 AND aoAxis1<=440 AND aoAxis2<=440 AND aoAxis3<=440 DO
        Axis1:=aoAxis1;
        Axis2:=aoAxis2;
        Axis3:=aoAxis3;
        Incr Axis1;
        Incr Axis2;
        Incr Axis3;
        SetAO aoAxis1,Axis1;
        SetAO aoAxis2,Axis2;
        SetAO aoAxis3,Axis3;
        WaitTime 0.2;
    ENDWHILE
ENDPROC
```

Figure 23. The Rapid code for moving the fingers of the gripper away from the centre.

As soon as the SC indicates that the collision between the fingers and the ring no longer exists, the robot continues its pathing routine. Similar procedures are created for all paths.

A view of the station logic (Fig. 24) shows all the SCs used to obtain the functionality mentioned in section 5.1. The expression SCs are there only to convert the analogue signal from metres to millimetres by dividing the signal by 1000. The range of the mechanism is 0 to 441mm.

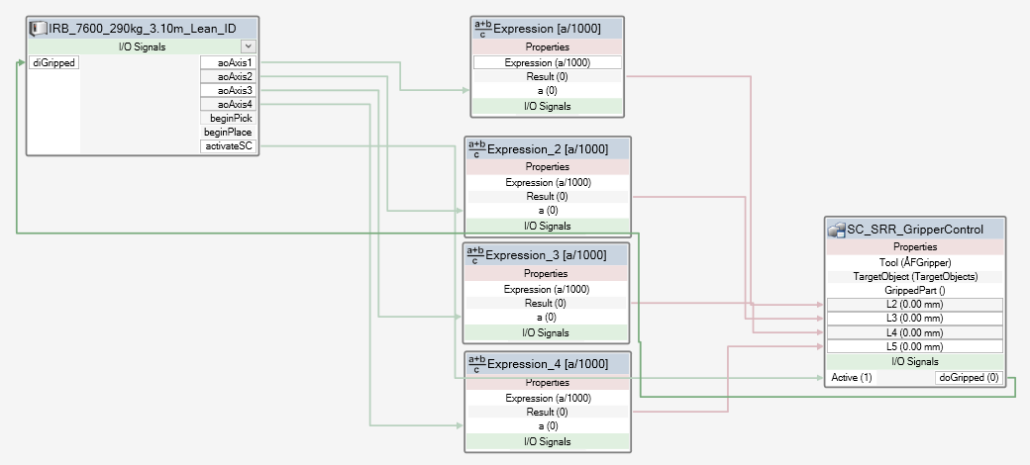

Figure 24. Station Logic: The robot controller is connected to the developed SC via a number of expression SCs.

Figure 25 shows a view of the Layout tab in RS. No other SCs than the developed one and the four expression blocks are used.

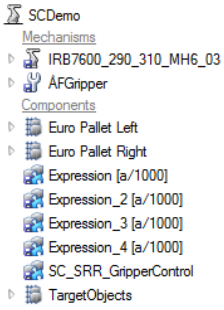

Figure 25. A view of the Layout tab in RS.

The results from the test rig show that the required functionality of the SC has been achieved. The gripper is able to pick and place the ring from the inside and outside; it is able to use the fingers to push the ring sideways if it is offset from the centre of the gripper; and it works as intended with grippers with different numbers of fingers.

However, the unreliability of the physics simulation in RS did manifest itself during testing. Sometimes the ring did not move according to the collision with the fingers, and sometimes it did. This topic will be discussed in the next section.

Finally, a test for reusability was conducted. The test was performed with an ABB tool available with all RS installations called '*ABB Smart Gripper*'. As can be seen in Fig. 26 the SC automatically generated the proper number of analogue inputs. When 11 mm was set on the input, the gripper opened up accordingly.

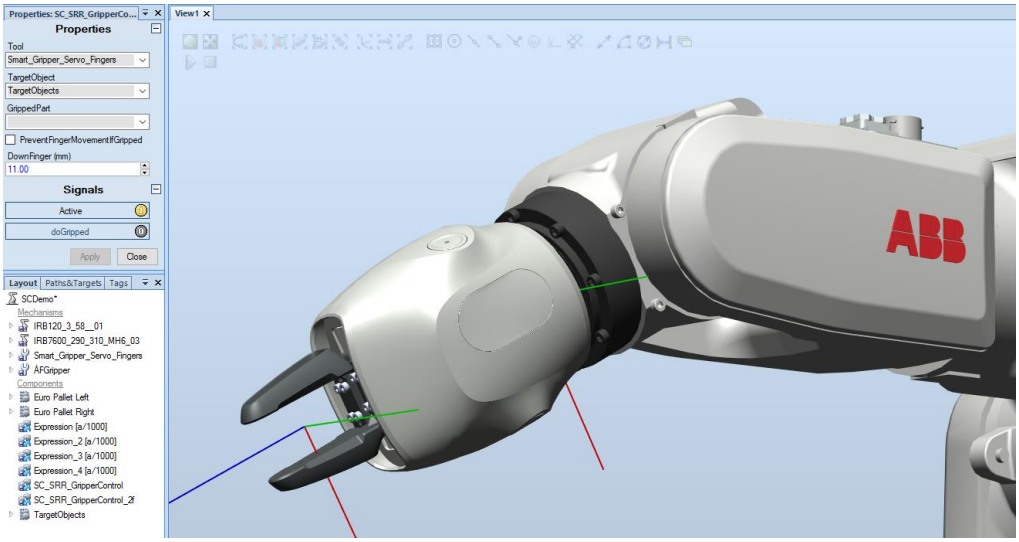

Figure 26. Test for scalability and reusability.

6 Discussion

The goals of this thesis were (1) to identify the necessary steps for creating a digital mechatronic model of a gripper in a .NET environment and discern if it can be made scalable, reusable and realistic; (2) to compare the new model to the one currently in use in terms of ease of development, complexity and efficiency during implementation; and (3) try to discern whether the final result can justify the additional engineering time (and cost) required to create such a model.

A number of decisions were made regarding the design of the SC in order to be able to deliver a working and reliable alternative to the current method. These decisions are also presented below.

Finally, some drawbacks of this method are mentioned.

6.1 First goal: Identify the necessary steps for creating a digital mechatronic model

The information in this report enables anyone with only a basic understanding of .NET development to create a SC via the 'CodeBehind-method' using the C#-language in VS. Section 4 covers all the needed software, relevant dll files, and where to access the built-in code by ABB for inspiration. The developer can choose whether or not the created SC is to be scalable, reusable and realistic.

6.1.1 Was it possible to make the model scalable, reusable and realistic?

The model can be considered scalable and reusable because it has been created to work with any gripper, no matter the number of fingers. It works by picking up objects using pressure on the objects by its fingers either in a pinching motion or a pushing out motion. The SC automatically generates a number of analogue input properties corresponding to the number of links in the mechanism and their physical behaviour. This property was fully tested and verified with one gripper mechanism. It was only tested to some extent with another gripper due to lack of time. A test to determine whether a gripper with

two fingers and only one analogue input is able to grip an object was not performed.

The object is picked up by the tool only when all the fingers of the gripper are in contact with the object being picked up (Fig. 27). This means that the object will always be at the centre of the gripper when it is being picked up. It is possible to adjust this feature, to allow further scalability and reusability of the SC, by allowing an individual finger to indicate collisions and stop the movement of that finger. It can be argued that this would open up new options for the way the gripper can interact with objects. However, this comes at the cost of added complexity both in terms of the code and the implementation.

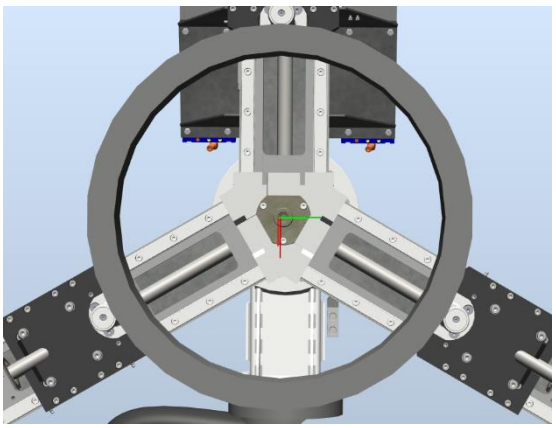

Figure 27. All fingers in contact with the ring. The ring is pushed to the centre of the gripper. This view is from the underneath the gripper.

In terms of realism, the aim has been to replicate the functionality of the real gripper. In particular this means the ability to push objects along a surface with its fingers and pick them up using the friction between the fingers and the object between them. In its current form, the SC is able to push an object along a surface, but it is unable to pick up the object by friction alone. The reason for this outcome is the unreliable physics simulation in RS. There are many factors at play, and their interaction is not clear and not well documented. For example, a custom material for the fingers had to be created to achieve the desired functionality of the gripper. The properties of this material are not yet documented in RS, and there is no way to know how these properties interact with the virtual objects in RS.

To a large extent it is a guessing game. The factors I have found to work best for the SC are the ones shown in Fig. 28. These values solved the problem of the object getting caught on the fingers while sliding down when released, even though the roughness (friction) was set to 0. With these parameters, the fingers are still able to push steel around (the material chosen for the ring) along a virtual floor. It is possible that a custom material for the objects, with some specific physical material factors, could lead to a more realistic physics simulation.

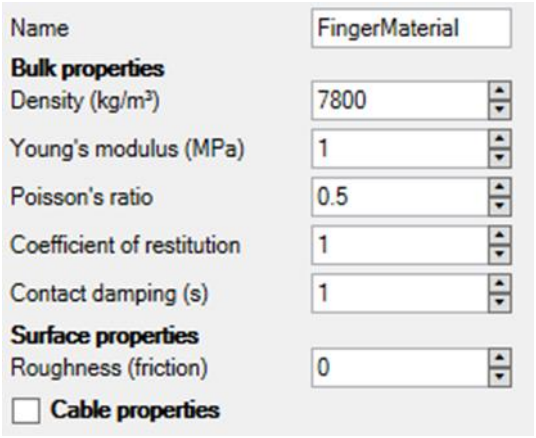

Figure 28. The physics material creation dialogue. The factors are guesstimated due to lack of documentation.

Even though work-arounds were required to achieve similar functionality in the virtual world to what exists in the real world, I believe the end result fulfils the aim of 'realism'.

6.2 Second goal: Compare the new and existing models

The existing method for controlling the gripper involved multiple, sometimes nested, SCs to solve the rather mundane task of picking up an object with a gripper. Even for an experienced RS engineer, these SC networks take time to set up and create a complex troubleshooting situation. It can be argued that it would be as easy to create a new solution for the next similar project as it would be to reuse the current solution. This existing method thus only works for a specific gripper, and only when picking up an object from the outside.

The new method requires only one SC and allows the same range of usability of the gripper as in reality. It works with any gripper with any number of fingers. There is no need for the extra complexity that extra

sensors add to fulfil the task at hand. The SC effectively transforms the fingers into sensors, which eliminates the need for the other SCs which are used in the current method. The implementation is fast and simple, and there are no time-consuming additional steps required during implementation.

In terms of efficiency of implementation and in terms of complexity, the new method must be regarded as superior to the older method.

As regards the ease of development, it is my perception that developing a custom SC is relatively simple. For a .NET developer with some RS experience, together with the information provided in this report, it should not take more than a week or two to create an SC of the same complexity as this one. However, someone with no previous .NET development experience would have to put in considerable effort to learn the functions of the different built-in namespaces, classes and methods.

6.3 Third goal: Evaluate whether the final result justifies the additional engineering time required.

For this specific task, I believe that creating a custom SC that could be reused for similar projects in future was worth the effort and time. An additional benefit is that the knowledge and experience gained from creating one SC considerably reduces the effort required to create the next one, particularly if code can be reused. Thus there is a trade-off between the value of time and the value of experience gained.

However, I do not think that it is feasible to create custom SCs for all tasks and projects. Many built-in SCs are often sufficient to solve the problem at hand.

6.4 Work-arounds implemented

Two major work-arounds are worthy of mention.

- The functionality where the object is picked up once a collision is detected between all fingers and the object between them is a work-around due to the inability of RS to simulate torque.

- The functionality where the object is picked up by attaching it to the gripper is a work-around to overcome the unreliable physics simulation in RS.

These work-arounds had to be implemented in order to achieve reliable functionality of the SC.

6.5 Drawbacks of this method

As this method of creating and implementing SCs relies heavily on the built-in code developed by ABB, there is a risk that the product will not function as intended after a future update released by ABB. It is impossible to assess the likelihood of this happening, but it could lead to additional development time to troubleshoot the broken SCs.

The other potential difficulty with taking this route, which is also corroborated by previous research, is the need for engineers who are knowledgeable not only in robot programming but also in .NET development.

6.6 Ethical and social considerations

The SC developed in this thesis has little ethical or social relevancy. Its impact is mainly on the valuable engineering time spent on redundant tasks, which can be categorised as re-inventing the wheel when setting up new projects that are similar to old projects.

As regards the social and ethical implications of VC as a whole, other authors have already covered this topic. As an example, Sarah Winther mentions in her work that the benefits of the implementation of VC as a standard in industry could reduce the workload and thus the stress on the developers while increasing the profits for the companies. [2]

7 Conclusion

For virtual commissioning to become an adopted industry standard, the creation of reusable virtual models is a must. This thesis has been a demonstration of how such a model can be created for the RS environment. The developed scalable, reusable and realistic SC decreased the complexity of implementation and improved the available functionality of the gripper compared to the current method at ÅF Pöyry AB. The difficulty of developing similar reusable models is not high in terms of engineering time savings when compared to the benefits such models provide. The major drawback of this method is the inability to guarantee that the SCs developed will be future-proof.

7.1 Future work

I would recommend investigating how more functionality could be implemented in the SC for grippers. It would be beneficial if the functionality could be turned on or off according to the needs of the gripper. For example, it would be useful to be able to stop the fingers as soon as they individually or all together contact the same object, irrespective of the value received at the analogue input. This could prevent objects from being malformed or displaced by the physics engine if the latency between the controller and the PLC is too large.

References

[1] Oppelt, Mathias, and Leon Urbas. 'Integrated virtual commissioning an essential activity in the automation engineering process: From virtual commissioning to simulation supported engineering'. *Industrial Electronics Society IECON 2014-40th Annual Conference of the IEEE*, 2014, pp. 2564-70.

[2] Winther, S. 'Virtual commissioning of production process'. [Master thesis on the Internet]. Gothenburg; Chalmers University of Technology / Department of Signals and Systems; 2016 [cited 2019 May 24]. Available from: https://hdl.handle.net/20.500.12380/250446

[3] Kritzinger, W., et al. 'Digital Twin in Manufacturing: A Categorical Literature Review and Classification.' *IFAC PapersOnLine*, vol. 51, no. 11, 2018, pp. 1016–22.

[4] Dahl, M., et al. 'Automatic Modeling and Simulation of Robot Program Behavior in Integrated Virtual Preparation and Commissioning.' *Procedia Manufacturing*, vol. 11, no. C, 2017, pp. 284–91.

[5] Ko, Minsuk, and Sang C Park. 'Template-Based Modeling Methodology of a Virtual Plant for Virtual Commissioning.' *Concurrent Engineering*, vol. 22, no. 3, 2014, pp. 197–205.

[6] Luo, Weichao, et al. 'Digital Twin for CNC Machine Tool: Modeling and Using Strategy.' *Journal of Ambient Intelligence and Humanized Computing*, vol. 10, no. 3, 2019, pp. 1129–40.

[7] Scaglioni, Bruno, and Gianni Ferretti. 'Towards Digital Twins through Object-Oriented Modelling: a Machine Tool Case Study.' *IFAC PapersOnLine*, vol. 51, no. 2, 2018, pp. 613–8.

Appendix A: Flowchart of SC