2021•2022

Faculteit Industriële Ingenieurswetenschappen

master in de industriële wetenschappen: elektromechanica

Masterthesis

Creating a virtual laboratory with a digital twin

PROMOTOR :

Prof. dr. ir. Michael DAENEN

PROMOTOR :

Mr. Jussi Horelli

COPROMOTOR :

ing. Geert LEEN

COPROMOTOR :

Mr. Jan-Peter NOWAK

Kamiel Cresens

Scriptie ingediend tot het behalen van de graad van master in de industriële wetenschappen: elektromechanica

Gezamenlijke opleiding UHasselt en KU Leuven

2021•2022

Faculteit Industriële Ingenieurswetenschappen

master in de industriële wetenschappen: elektromechanica

Masterthesis

Creating a virtual laboratory with a digital twin

PROMOTOR :
Prof. dr. ir. Michael DAENEN

PROMOTOR :
Mr. Jussi Horelli

COPROMOTOR :
ing. Geert LEEN

COPROMOTOR :
Mr. Jan-Peter NOWAK

Kamiel Cresens

Scriptie ingediend tot het behalen van de graad van master in de industriële wetenschappen: elektromechanica

UHASSELT KU LEUVEN

Preface

When talking at present about visualisation and simulation, the first term that comes to mind is Digital Twin. This thesis is about creating 3D models with the software package: Visual Components to show different aspects of the automation industry and the communication between a digital twin and a PLC (programmable logic controller). The project started at the end of February and ended at the beginning of June. The project took place at the Häme University of applied sciences (HAMK) in Valkeakoski, Finland.

The desire for this project comes from my interest in PLC programming. During my professional bachelor's degree, I worked with PLCs and an industrial hardware process for my internship. I find this subject fascinating because digital twins are becoming more and more popular, and now I have experienced both sides.

After this thesis, I will finish my master's degree in Electromechanical Engineering, specialising in automation technology. This degree is a joint study programme from UHasselt & KULeuven at Diepenbeek, Belgium.

Throughout this project's journey, I had the opportunity to expand my experience communication-wise with Tia Portal (Siemens). I had a short introduction to working with Twincat (Beckhoff), which I had never used before. The most knowledge/ experience I gained is working with a digital twin, and this was done with the software Visual Components where I created my 3D models and controlled them with a PLC. I completed the requirements that HAMK provided me, but I could design even more complex models with more time. The thesis took place on Erasmus, and I was here for only four months. Because of the different universities, much communication was required to come to an agreed subject for the project. This meant the start was a bit slower, but I only had two courses in Belgium left, so I had much time to work on the thesis.

Being an exchange student is not only about doing your thesis at a different university; it teaches you so much more. Before I came here, I could write and understand English at a decent level, but now I am way more confident in speaking it. I met people from all over the world and made some friends for life. I explored the country in my free time and discovered many lovely places in Finland. Aurora Borealis (Northern light) in Lapland is one I will never forget.

First, I would like to thank my supervisors from HAMK, ir. J. -P. Nowak, ir. J.Horelli and ir. J. Sarkula for the continuous support and feedback they provided. I would also like to thank my supervisors from UHasselt/KULeuven, Prof. dr. ir. M. Daenen and ir. G. Leen for their professional guidance and support throughout this project.

Table of contents

List of figures

List of tables

Glossary

AGV	Automated guided vehicle
API	Application programming interface
DT	Digital twin
HIL	Hardware in the loop
LIN	Linear movement, the trajectory is a straight-lined path
MIL	Model in the loop
OPC UA	Open Platform Communication Unified Architecture
PHM	Prognostics and Health Management
PIL	Processor in the loop
PLC	Programmable logic controller
PTP	Point to point movement, the trajectory is the fastest
SIL	Software in the loop
TCP	Tool centre point
TCP/IP	Transmission control protocol/ Internet protocol
VC	Visual Components

Abstract

HAMK (Häme University of Applied Sciences) in Finland wants the possibility to give practical laboratories about PLC programming online and the option for students to practice at home with digital twins. This thesis investigates the possibilities of recreating hardware set-ups related to industrial applications in a 3D simulation and controlling them with a PLC. The main requirement was to examine if the software package could recreate existing hardware models at HAMK. The models exist of optical, material and colour sensors, different actuators, and conveyor belts.

The first part of this master's thesis was investigating which software was the most suitable for the requirements of HAMK. This was determined by doing a literature study about digital twins and different software packages. The first step was to get familiar with the chosen software package, Visual Components. A significant part of this thesis was creating the 3D models and adding Python scripts to be able to control them with a PLC.

With the created models, it was possible to recreate the existing hardware models and simulate them with a PLC. The models have a physics feature, which means they can be influenced by the environment, making them even more realistic. Working with the software showed endless possibilities but also some flaws. Depending which functionality of the software was used, it was impossible to change the course of a robotic arm and AGV to avoid collisions.

Abstract in Dutch

HAMK (Häme University of Applied Sciences) in Finland wil de mogelijkheid om praktische lessen (PLC-programmeren) online te geven alsook de optie voor studenten om thuis te oefenen met digital twins. Deze thesis onderzoekt de mogelijkheden om hardware producten met betrekking tot industriële toepassingen te ontwikkelen en deze te besturen met een PLC. Het hoofddoel was onderzoeken of het mogelijk is om bestaande hardware modellen aanwezig op HAMK te recreëren met het softwarepakket. De modellen bestaan uit optische, materiaal en kleur sensoren, verschillende actuatoren en transportbanden.

Het eerste gedeelte van deze masterproef was het bepalen van de meeste geschikte software voor de vereisten van HAMK. Dit is bepaald doormiddel van een literatuurstudie over digital twins en de verschillende softwarepakketten hiervoor. De eerste stap was vertrouwd raken met gekozen de software, namelijk Visual Components. Het belangrijkste onderdeel van deze thesis was het creëren van de 3D-modellen met bijhorende Python scripts, zodat de PLC deze modellen kon manipuleren.

De gecreëerde modellen geven de mogelijkheid om de hardware modellen te recreëren in de software en deze te simuleren met een PLC. De modellen hebben ook fysieke eigenschappen waardoor de omgeving ze kan beïnvloeden en de simulatie nog realistischer wordt. Werken met de software liet vele mogelijkheden zien, maar ook gebreken. Afhankelijk met welk deel van de software werd gewerkt, was het onmoglijk om het pad van een robot aan te passen en dus botsingen te vermijden.

1 Introduction

1.1 Context

The project occurred at HAMK (Häme University of Applied Sciences) in Valkeakoski, Finland. HAMK has seven campuses spread in the Helsinki metropolitan area of southern Finland. The first roots of HAMK go back to 1840, when their first campus taught agricultural education. Currently, the degrees are focused on bioeconomy, wellbeing, technology, entrepreneurship, and business. Next to their degree programmes, four research units work on assignments for companies and the public sector.

HAMK
HÄMEEN AMMATTIKORKEAKOULU
HÄME UNIVERSITY OF APPLIED SCIENCES

Figure 1 HAMK logo

1.2 Problem statement

The campus in Valkeakoski teaches electrical and automation engineering technologies. In the laboratory at the campus, students work with PLCs and miniature industrial processes. During the pandemic, this was not possible, which is one reason HAMK wants the option to teach practical classes online; this is possible with digital twins. A second motivation is that digital twins are becoming more and more popular, and through this way, students have a first experience with them. Figure 2 shows an example of a digital twin.

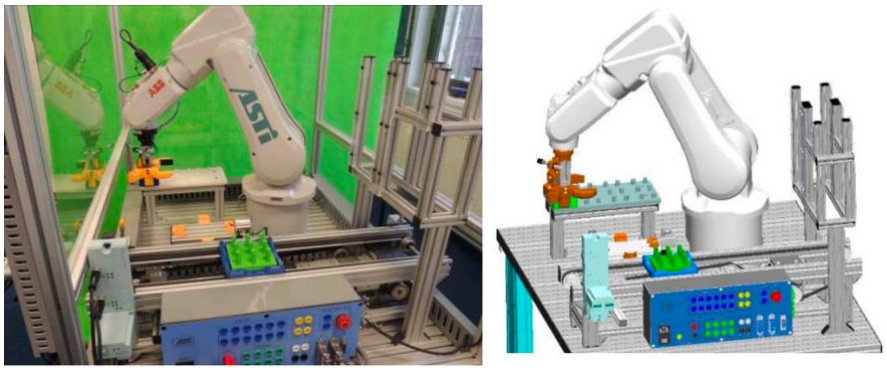

Figure 2 Example digital twin [1, p.607]

Digital twins will be used more and more in the industry because of their many applications. HAMK has one software package, CIROS from Festo, containing pre-designed models. But they want to take it a step further and be able to create models from scratch.

1.3 Goals

The primary purpose of this thesis is to investigate if it is possible to recreate the existing miniature industrial processes in the laboratory with the digital twin software. The created 3D model must have the option to be controlled by a PLC (Siemens and Beckhoff). Figure 3 shows an example model in the laboratory. It consists of a few sensors, a conveyor belt and some actuators.

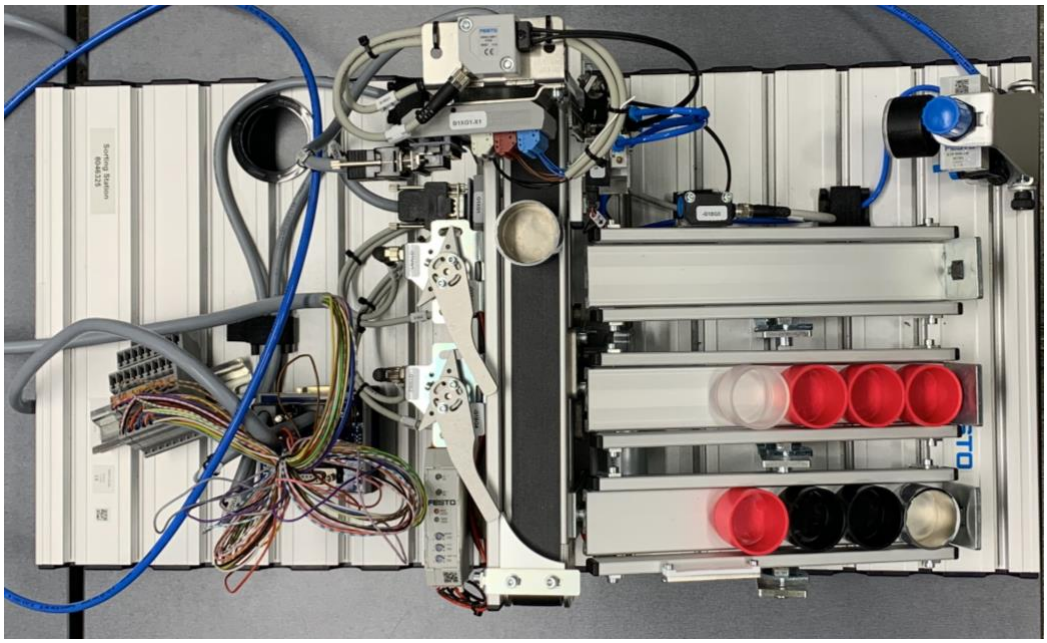

Figure 3 Example of a model in the laboratory

The goal is to create these components separately and then combine them until a similar model is created. With this model, students can practice the PLC programming and the connectivity between the software and the PLC.

Besides creating a model during this thesis, writing a manual on making the components from scratch was also required. With this manual, students can create their ideas/designs and control them with a PLC.

1.4 Method

- Doing a literature study to learn more about digital twins and determine the most optimal software package for the specific requirements.
- Designing models with existing components from the library to show different aspects of automation like production feed, idle time, …
- Creating components from scratch that possess physical behaviour.
- Combine the created components into a miniature industrial process.
- Control the model with the PLC.

1.5 Structure of the scription

The scription starts with a literature study, which explains the evolution of digital twins and Open Platform Communication Unified Architecture (OPC UA). The second part of the literature study compares different software packages and based on this, the most optimal is chosen. The next chapter explains the used system structure and how the PLC communicates with the software. The chapter about the functionalities of the software presents the possibilities and applications of each functionality. The first part of the fifth chapter discusses cases and examples created with the different software functionalities. The second part of this chapter illustrates a setup that combines the cases and examples from the first part. The final chapters discuss future work and the general conclusion.

2 Literature study

The thesis is about creating /designing a 3D simulation model which can be used for a virtual laboratory assignment. When mentioning a 3D simulation model, the term digital twin (DT) comes first to mind. The first part of this literature study is about the research/review of a digital twin's current state of the art. It will start with what precisely a DT is and in which areas it can be used, and some main challenges a DT faces today. Most of the software to create/simulate a DT can use Open Platform Communication Unified Architecture (OPC UA) as the communication protocol for the connection between the DT and example, a simulated PLC. A few of them have the option to communicate with PLCSIM-advanced from Siemens directly, but OPC UA can use other manufacturers as well. In the second part of this study, some research was done about the background of OPC and the system. Thirdly, some examples are discussed created with different software. Lastly, based on existing review/comparison papers and the research from the third part of this study, a conclusion is drawn about which software is further used in the thesis. Figure 4 shows a visual representation of the discussed paragraphs.

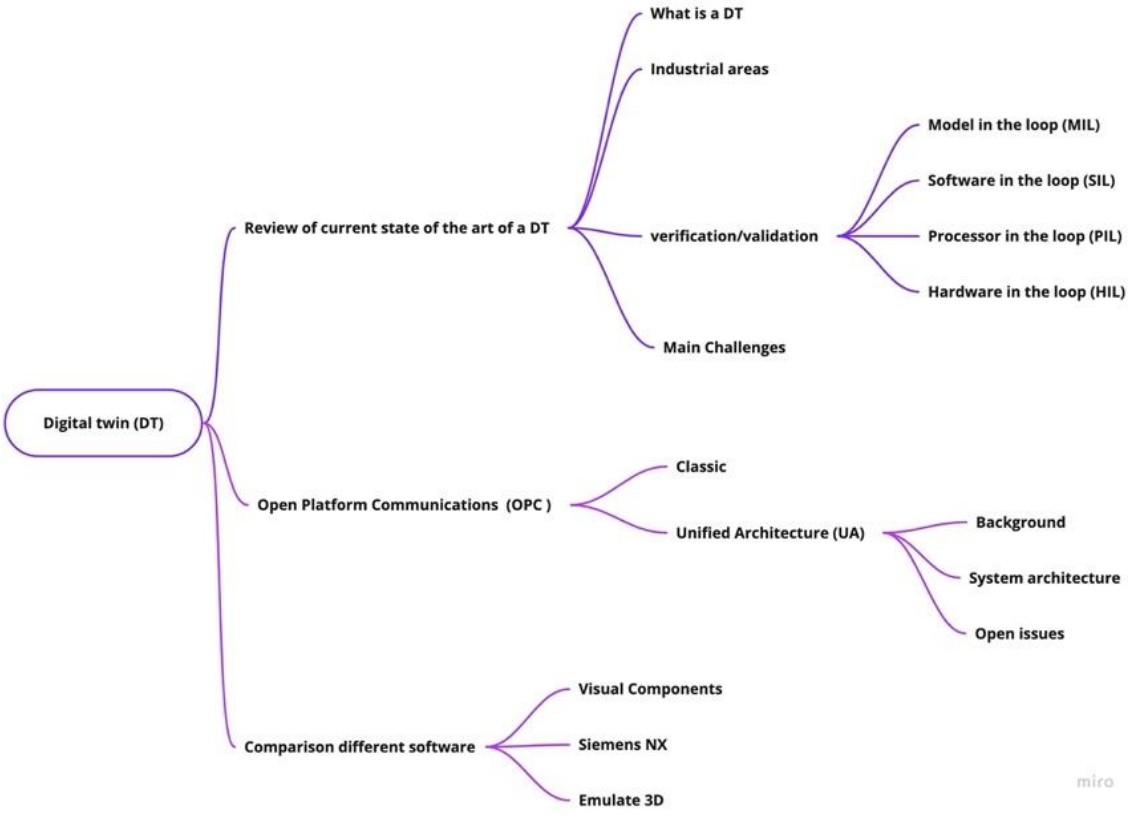

Figure 4 Mind map from the main structure of the study

2.1 Digital twin state of the art

2.1.1 What is a digital twin?

A digital twin (DT) is a popular research topic, and it made its first appearance in 2003, when it was defined as a virtual model of an already existing physical model [2], [3]. There are still different understandings of DTs and where the focus should be for the continued research. One group believes the research direction should be purely on the simulation part [4]. Another group argues that a DT consists of 3 parts: physical, virtual and connection parts and that the combination of those three should be researched [4], [5]. The connection part is responsible for exchanging data and information between the physical and virtual models. The essential representation of this 3-part model is shown in Figure 5.

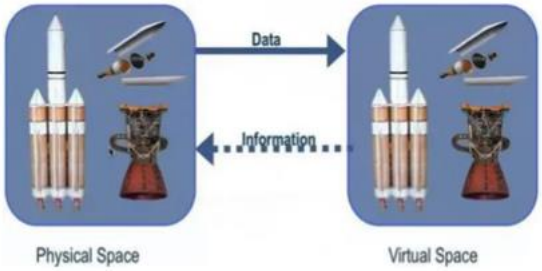

Figure 5 Three-dimension DT model [4, p.2406]

Digital twins can simulate events, and they do this not solely based on expert knowledge but also using data collected from the field [3],[5], and [6]. The virtual representation can compare the DT with the physical model and detect differences, which can point to failures or wear of a product. For example, the data collected from the field can also be used as feedback for companies on how their products are used in real-time and show them possible flaws and where they can improve [7].

In the beginning, there weren't many enthusiasts of a DT because there was not a clear view of what would happen in the long term. After NASA showed what the possibilities were with a DT in 2012, there was more interest in the technology [4]. Figure 6 illustrates the growth in papers about DTs and considering this growth; it will only become more and more used by the industry.

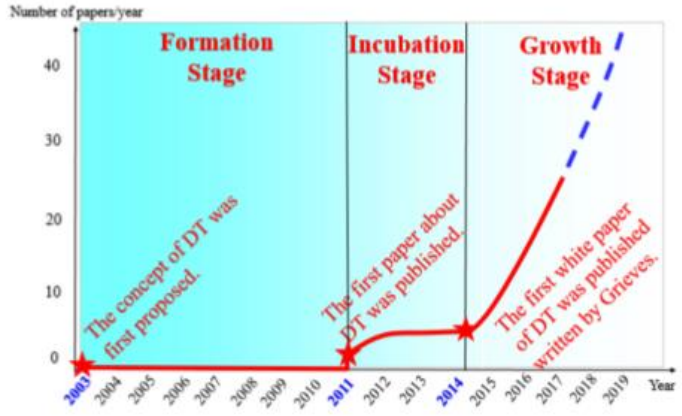

Figure 6 Development of the DT research [4, p.2407]

2.1.2 Industrial applications

This paragraph explains the four main applications of DTs that companies or researchers have officially publicised. Figure 7 gives a visual representation of the different applications where DTs are used.

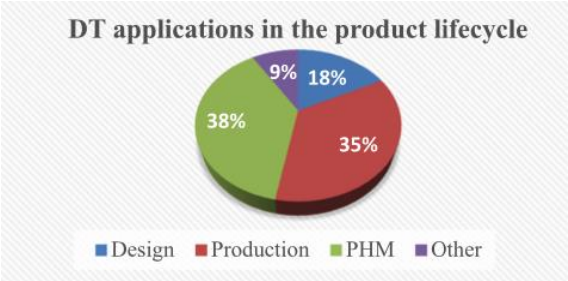

Figure 7 Distribution of DT publications [4, p. 2409]

2.1.3 Product design

DTs can be used for a variety of applications. The first one that comes to mind is product design. The DT can show flaws in the early designing phases before producing the physical product. It gives the user a 3D preview of an industrial line. For example, the DT can verify if the parts/machines fit together before the actual implementation happens. It is even possible to use multiple DTs for designing the layout of an entire factory [3], [4], [6].

2.1.4 Production

A DT used in the run time of a production line has many benefits. The first thing that comes to mind is running the DT simultaneously besides the physical model and updating it with real-time sensor data. This opens the possibility of visualising the production process anywhere. One step further is to predict the physical model's behaviour based on real-time sensor data. The DT can be simulated faster than the actual runtime of the physical model [4], [5]. Combining the two mentioned benefits can be used to optimise production/performance. Based on the monitoring and behaviour predictions, adjustments can be made to prevent failures before they occur [4]. These adjustments could be the intervention by an operator or adjusting the planning of an autonomous system [7].

2.1.5 Prognostics and Health Management (PHM)

At present, DTs are used mainly for PHM. PHM means predicting the current state of a model and predicting the life duration. The prediction of the life of an aircraft was the first time a DT was used for PHM based on damage modelling, structural finite-element analysis, … The predictions went from the life of the wings to the probability of failure of the tires at touchdown. PHM with DTs is not limited to aircraft but can also be used on cyber-physical systems or manufacturing processes [4]. The DT used for PHM shows significant advantages over the traditional PHM. An important difference is that the conventional PHM only uses historical and current sensor data but cannot merge the current and historical data and apply it to a simulation model to predict an outcome. Because of the connection between the virtual and physical models, the virtual can constantly be updated based on real-time data to improve the accuracy even more for predictions [4].

2.1.6 Other

Another application is the combination of a DT and an OPC UA server to remotely control a physical model from any computer. But as [7] mentions, this specific application is not on point because the communication with the used software was not continuous, and the physical model experienced lag. Virtual commissioning is also an option where a DT is useful. It is a combination of product design and production because it is not purely to test upfront before manufacturing a product or to simulate the production [8]. For example, when a new machine is added to an existing production line and must be fine-tuned. The DT can search the optimal parameters so that the physical model can be taken almost instantly in production.

2.2 Verification and validation

Verification and validation are used during product design and production itself. The two phases were mentioned before, but this paragraph describes a few methods to accomplish this. There are again a few definitions for what precisely verification and validation are. Still, the main conclusion is that it verifies if the designer/operator is going in the right direction or checks if a product meets a customer's requirements. These requirements can be about the physical product or how good the visual model is representing the physical model. Figure 8 gives a visual representation of the difference between verification and validation [3].

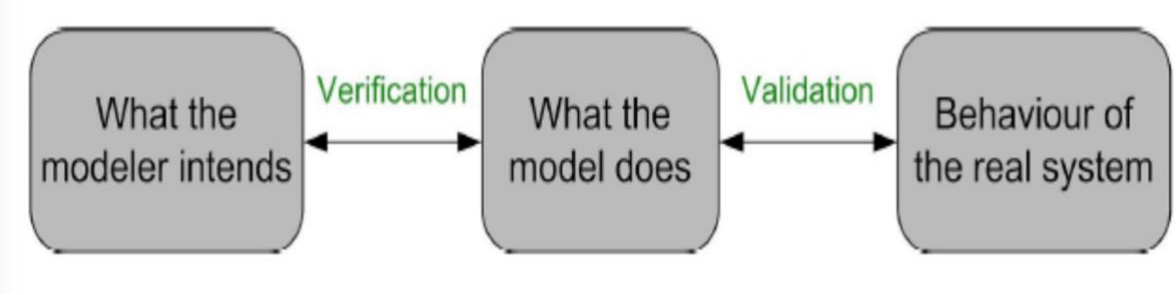

Figure 8 Verification and validation principle [3, p.12]

A few methods are discussed in the following subparagraphs, which are used the most in the industry for verification and validation.

2.2.1 Model in the loop (MIL)

For this approach, the first step is designing a model of the physical plant, which has the essential plant features in simulation software. Simulink is one of the most used software packages. Once the model is created, the next step is creating/designing a controller block to verify if it can run the simulated plant. The part where the logic of the controller model is tested on the simulated plant is called the model in the loop (MIL) [3].

2.2.2 Software in the loop (SIL)

This approach uses a virtual controller with code. It is connected to the virtual model through a communication protocol (OPC UA, TCP/IP, etc.) which replaces the controller block from MIL. The approach tests if the controller logic, once converted to code, can run on a hardware controller. Figure 9 shows a schematic of a possible SIL setup [3].

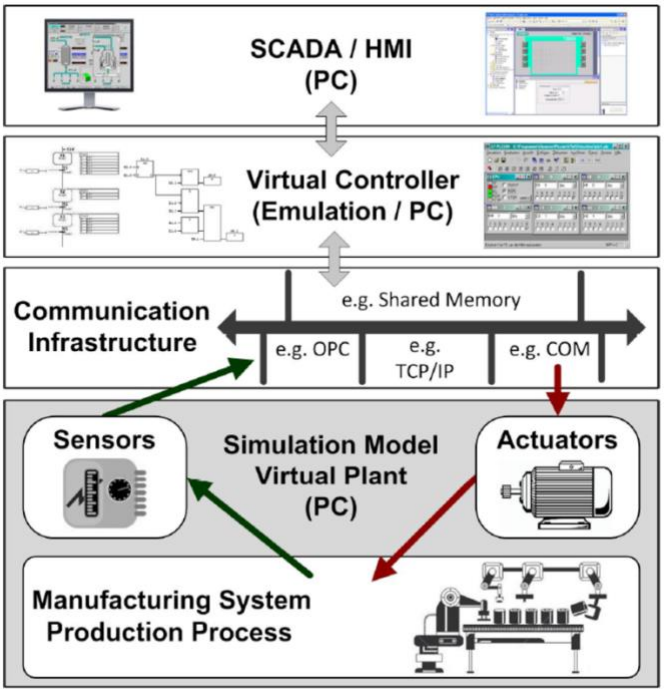

Figure 9 Visual representation of SIL [3, p.14]

2.2.3 Processor in the loop (PIL)

This approach is used for testing the code on the hardware controller but still with the virtual model. This step will show if there are any bugs/errors in the code or problems with the hardware controller. According to [3], this approach is not used often but is mainly applied in the automotive industry.

2.2.4 Hardware in the loop (HIL)

Now that the controller and plant are verified with the previous approaches, the virtual model can be switched with the physical model or run them both simultaneously, depending on the application of the DT [3].

2.3 Main challenges

Digital twins have received a great deal of attention in the last years, and so the technology is still developing and not on point yet. A main drawback of the technology is that creating a model consumes much time, and special training is needed [3]. Especially if the DT is used for virtual commissioning only, the model is not useful anymore after the commissioning. So the consumed time is not worth it at this stage of the technology. Because of the cyber-physical fusion in a DT, security is a field that needs to be studied more in-depth. Thinking about the OPC UA communication and remotely controlling a physical object, the security should be strict [4]. Another challenge for companies is deciding to use a DT based on the return of investment. It is difficult to quantify the value the DT delivers at present, and the high price of the licenses is also a big argument in the decision [8].

2.4 Open Platform Communication (OPC)

2.4.1 Background

OPC is a communication protocol that allows automation/industrial data in an IT context. Microsoft and a few automation companies developed it. It has three main functions: exchanging data, exchanging alarms and events, and exchanging historical data. The first version of OPC is called OPC classic and is based on Microsoft DCOM (communication protocol developed by Microsoft) [3], [7], [9]. OPC UA ”is the data exchange standard for a safe, reliable, manufacturer and platform-independent industrial communication” [9, p.2]. The main difference with the classic version is that the UA version is not based anymore on Microsoft DCOM technology and doesn’t need a Microsoft operating system. Another significant advantage is that OPC UA allows communication between devices of different manufacturers [3], [7].

OPC has a client-server architecture, and the client is connected to the server so that data can be exchanged between devices. These devices can be physical or virtual models. Figure 10 shows the basic principle of an OPC client-server setup. It is possible to have multiple servers in a single network, and these servers will process the client’s requests [3].

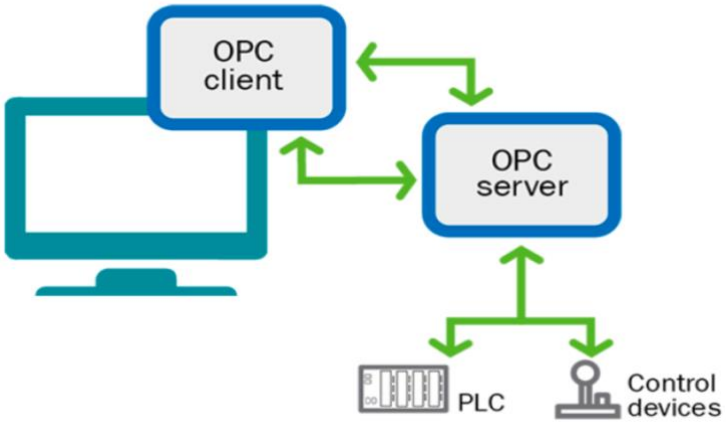

Figure 10 Basic principle OPC client-server [3, p.25]

2.4.2 System architecture

Figure 11 illustrates the main components of an OPC UA client-server setup. The communication is achieved by Application Programming Interface (API), a software interface between devices. The client can send requests through the API, and the server sends the responses. The OPC UA server has an address space consisting of nodes [7]. Nodes are similar to a folder structure on a computer, and each node has its unique identity [7], [9]. The clients can access these nodes and create a reference (Monitored Item in figure 11). Whenever there is a change in a Monitored Item, the data will change, or an alarm is triggered.

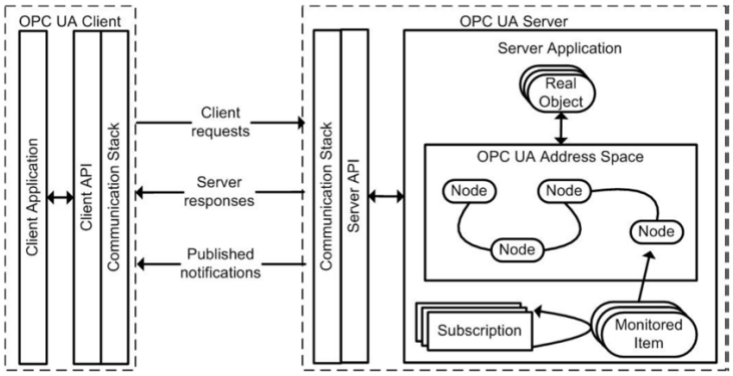

Figure 11 Overview components of an OPC UA architecture [7, p.13]

2.5 Review different software packages

2.5.1 Visual components

Visual Components (VC) can simulate material flow and robotics. The software uses the NVIDIA PhysX engine. Because of this, the software can visualise the effect of physical forces applied to the model depending on the material type. Another great feature of VC is that an existing library already consists of different components from different manufacturers. It is a network library, which means it is constantly updated. The software can import CAD files from a long list of various software packages. The platform is also used for simulation software from other manufacturers such as Kuka Sim and Octopuz. Another important fact is that VC can be connected with a programmable logic controller (PLC) from different manufacturers, the hardware or software version [3], [7], [11]. The available PLC interfaces are Beckhoff ADS, OPC UA and Siemens S7 communication protocol. VC uses Python as a script development language, opening another option for communication interfaces [7], [11].

The website from VC provides a lot of free tutorials, and the software is very user-friendly but still can implement complex layouts [11]. Visual components has the ability during the configuration of the paring of variables to switch between “simulation to server” or “server to simulation”. This option allows to receive information from the OPC-server or send it to the OPC-server [7]. Figure 12 shows the user interface of VC with an example simulation model loaded.

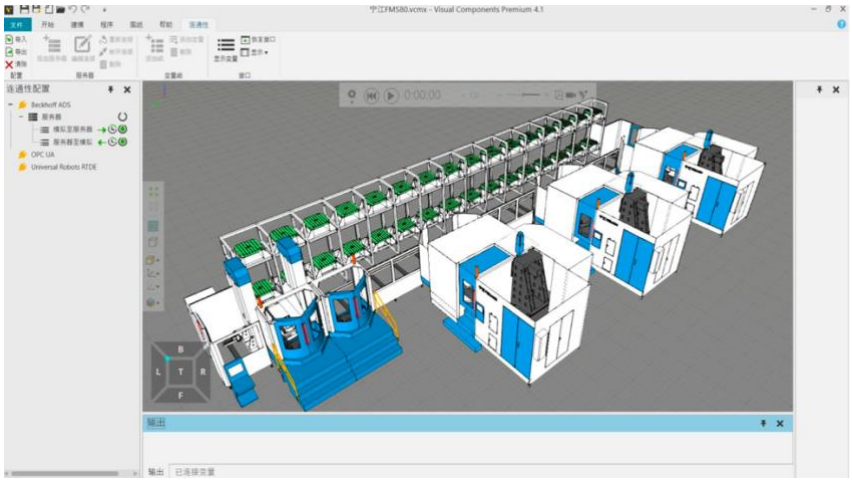

Figure 12 Interface of VC software [11, p.3]

2.5.2 Siemens NX

Siemens NX can design and simulate designs and this all-in-one program, whereas other programs mostly require multiple applications. The software package also uses the NVIDIA PhysX engine, an excellent attribute for testing the behaviour when physical forces are applied. NX can also import models created in other software packages, and these existing models can be tested for their kinematic behaviour. Siemens NX can communicate through an OPC UA-server or connect to a hardware controller. The disadvantage of using NX is that the software has a lot of different applications, which makes it very complex, and there is not much information or tutorials to find [3], [12].
Figure 13 illustrates an imported FESTO model in Siemens NX MCD. MCD is the Mechatronics Concept Designer part of NX, which is the part that is responsible for the actual simulation and kinematic behaviour of the model.

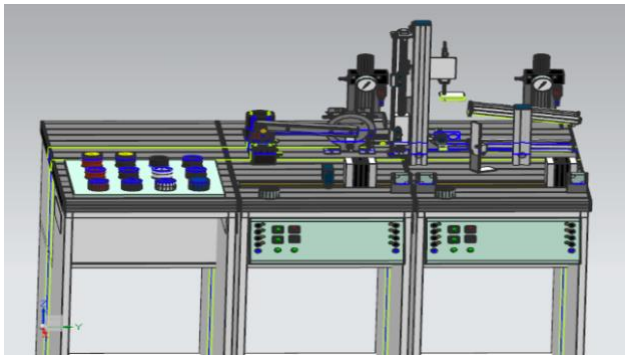

Figure 13 Example of a model in Siemens NX [3, p.19]

2.5.3 Emulate3D

Emulate3D is another simulation software for industrial applications. The primary purpose is to control the simulation model and do virtual measurements to see beforehand if there are any problems with the size of components. The model can be simulated by a hardware or software PLC, and this can be done with Fetch/Write protocols such as Mitsubishi, Siemens, Beckhoff and Rockwell. The possible communication interfaces are Siemens PLC, OPC UA and Beckhoff Soft PLC. The development languages are C# and Jscript, which compared to Python with Visual Components, are harder to learn. The software itself cannot create models, but with the supporting Demo3D modelling, software models can be made from scratch. Emulate3D allows to import CAD files from different file types, and the list is a bit shorter than VC or NX [11]. Figure 14 shows the user interface of Emulate3D.

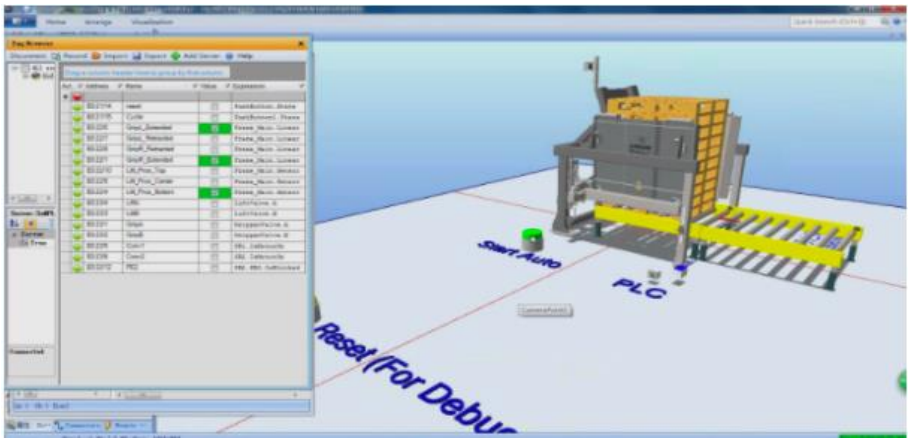

Figure 14 Interface of Emulate3D [[11, p.4]

2.6 Experience with digital twins for educational purposes

According to [13] a big advantage of using digital twins for educational purposes is that it creates extra motivation for students and it lets them expand their knowledge of the automation sector. During the research the main obstacles were IT related, specifically with the OPC communication protocol and connecting it to a PLC. Another factor is that the software for creating DT is very complex, and the teacher does not have enough expertise to answer every question [13], [14]. In [14], the fact that there needs to be a balance between improving learning and avoiding unnecessary obstacles is brought up. Students need to learn with the aid of the digital twin what an actual industrial process is. A problem/obstacle that the research of [14] revealed with Visual Components is that there can only be one PLC connected, whilst in a factory, every station mostly has its own PLC.

2.7 Conclusion

Digital twins have many applications, from product designing to virtual commissioning to predict the behaviour of a physical model. Since their first release, DTs have come a long way and are still evolving today. Most software packages can use the communication PLC interface from Siemens and a few Beckhoff, but almost all can use OPC UA. OPC UA is a universal communication interface independent of the device's manufacturer and can run on most operating systems, unlike the OPC classic. The three different software packages have advantages and disadvantages, but some are more important than others. For example, Visual Components has a lot of free tutorials and guidelines, and this is an immense advantage, as well as the possibility to use a direct connection to a Beckhoff PLC and the network library. Although Emulate3D can connect to Beckhoff, it needs Demo3D software for digital modelling, and it is more inconvenient than VC [11]. Siemens NX has the most possibilities, but it is a complex software package beyond simulating and modelling. The fact that there is not much information or tutorials available is a significant disadvantage. Considering all these factors and the requirements from HAMK, Visual Components is the chosen software for this thesis.

3 System structure

Figure 15 illustrates the used system structure for this project. The programming of the software PLC for this project is achieved with Tia Portal. The communication between Visual Components (VC) and the software PLC is done through OPC UA.

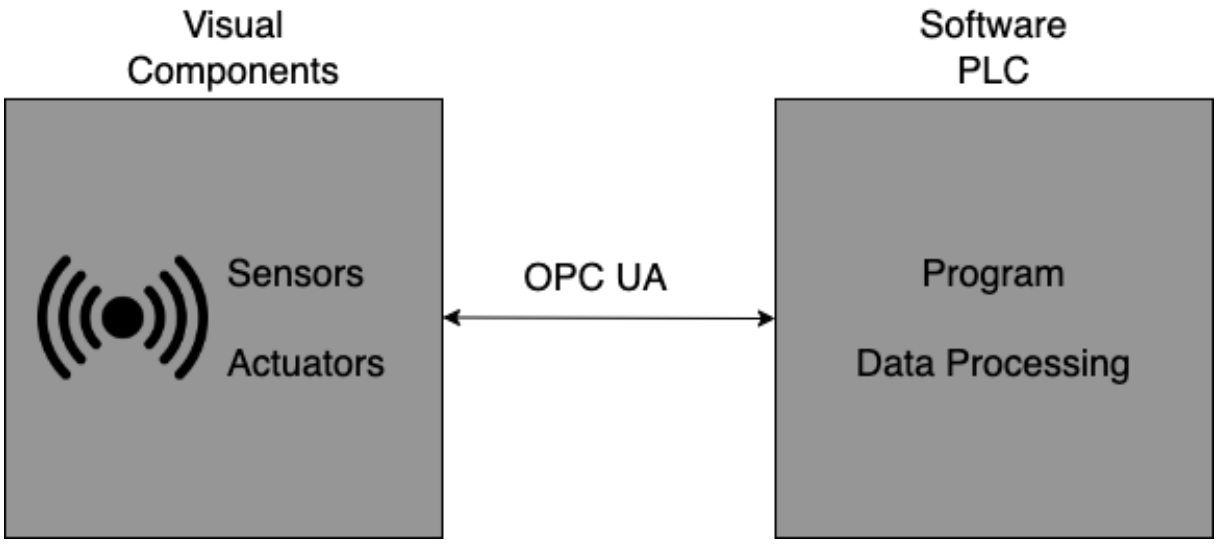

Figure 15 System structure

3.1 OPC UA communication

VC has the option to connect with PLCSIM or TwinCAT directly. The communication protocol OPC UA is used because it is becoming a more popular technology, independent of the software/hardware PLCs manufacturer.

3.1.1 S7-PLCSIM Advanced

To run the OPC UA server on a virtual PLC, PLCSIM Advanced is required. When S7-PLCSIM Advanced is installed, a Siemens Virtual Ethernet adapter is added to the network connections, as illustrated in figure 16.

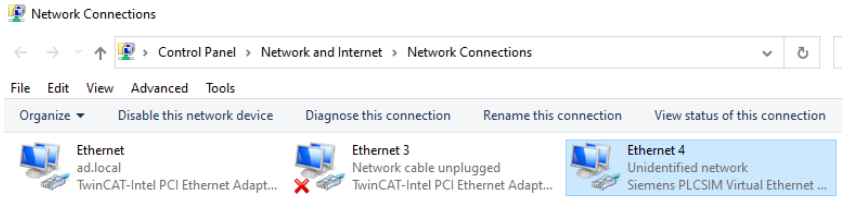

Figure 16 Ethernet adapter settings

To run a virtual PLC, the IP address of the PLC and the virtual ethernet adapter must be in the same range for the connection to work.

Figure 17 shows an example of a virtual PLC with IP address 192.168.0.1. This IP address must be the same as the IP address in the TIA Portal project. The address of the virtual ethernet adapter can be 192.168.0.2, for example [[15].

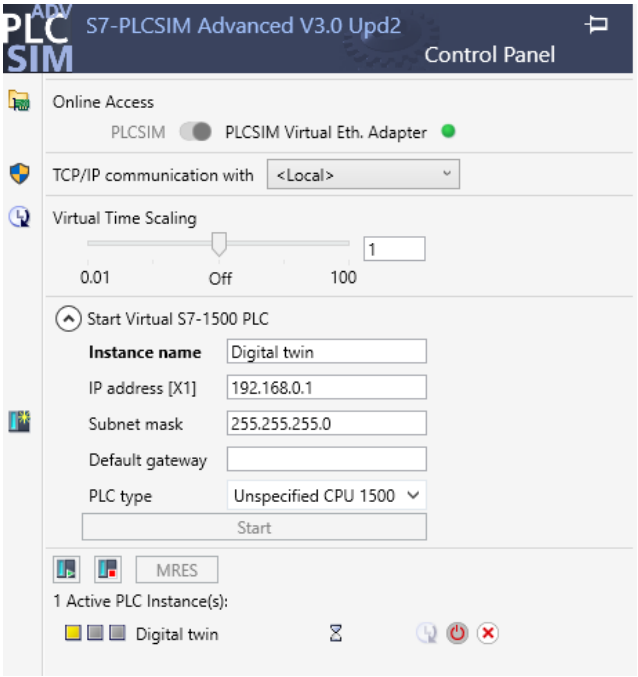

Figure 17 PLCSIM Advanced setup

3.2 Connectivity in Visual Components

In this case, the software PLC is the OPC UA server, and visual components (VC) is the client. This is important to understand for connecting the variables. Once the server is added and the connection is tested, the variables from the PLC and software can be linked.

Figure 18 shows the parameters to connect to the server run by the PLC. Because this was done on a local network, no authentication was necessary. Noteworthy is that it is possible with visual components to connect multiple PLCs through different OPC UA servers.

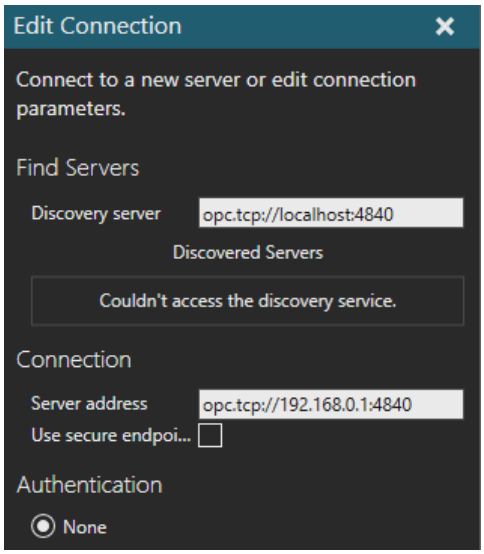

Figure 18 OPC UA server connection in VC

Once the OPC UA server is running, and the connection is made, all the PLC tags are visible in the software and ready to be paired with the variables from the simulation. Figure 19 illustrates an example of connected variables between the PLC and VC. The difference between “simulation to server” and “server to simulation” is important to notice. For example, a sensor in the digital twin is an input for the PLC, and this pair is added in the “simulation to server” part. The exact opposite is happening for the outputs from the PLC.

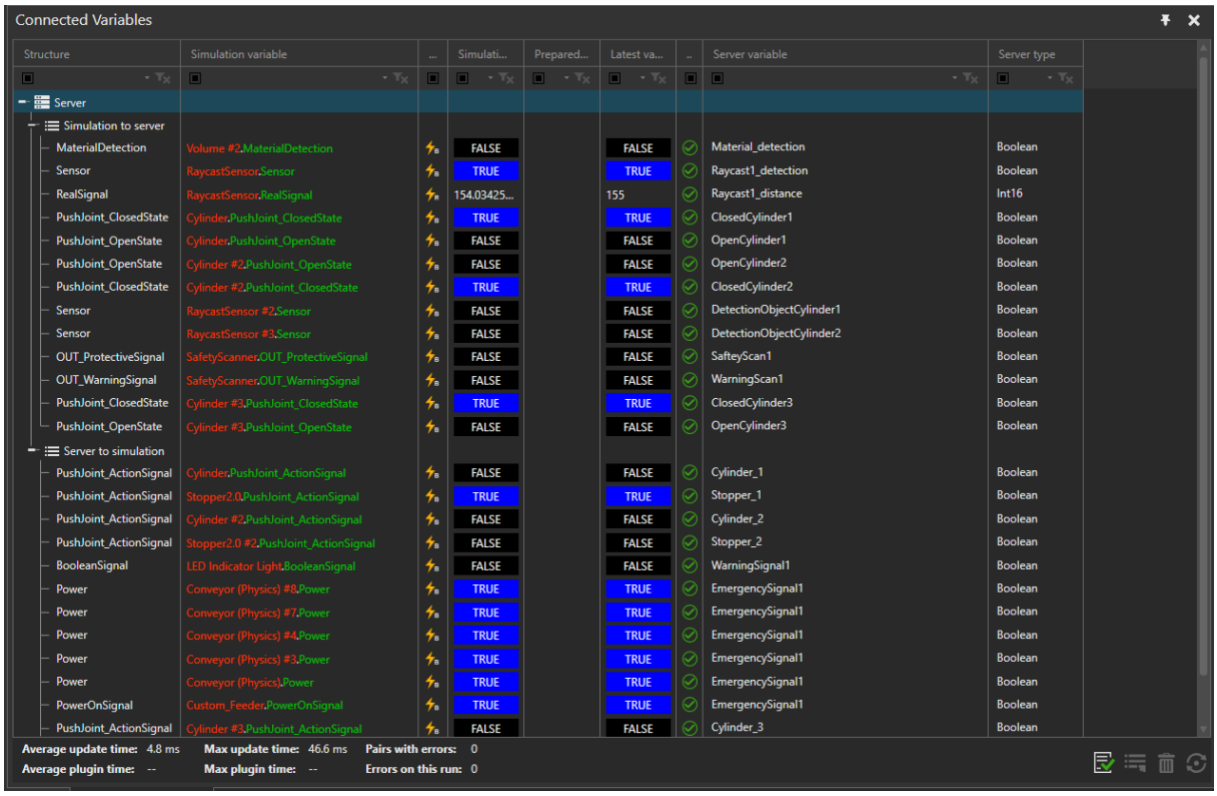

Figure 19 Connected variables in VC

4 Software functionalities

4.1 Process flow

Process Modelling is a simple and visual way of simulating the flow of products and processes. The content of process modelling allows to:

- Create and edit product types
- Create and edit flow groups
- Visualise processes and edit their routines

4.1.1 Product types

The product type editor shows the used product types in the process flow, sorted according to the flow group. Figure 20 shows the layout of the product type editor. A flow group controls the flow of the added product types from one process to another. For example, a flow group could be an industrial process where objects are placed into boxes, but it is possible to work with two different kinds of boxes, then the two boxes and the object are 3 product types.

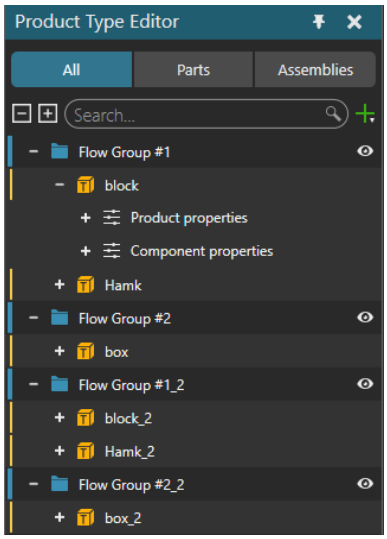

Figure 20 Product type editor

4.1.2 Process flow

In the process flow editor, process sequences associated to flow groups can be created and edited. It allows for creating the transport route in between the different process steps of a flow group. Figure 21 illustrates an example of an industrial process with two flow groups. The first flow group starts at the left conveyor and goes to the machine and then to the conveyor at the right. The process steps in the figure are the blue circles, and their names are below in the editor. In between the steps, different transport methods are used. For example, the robot is used from the conveyor to the machine, as shown in the figure.

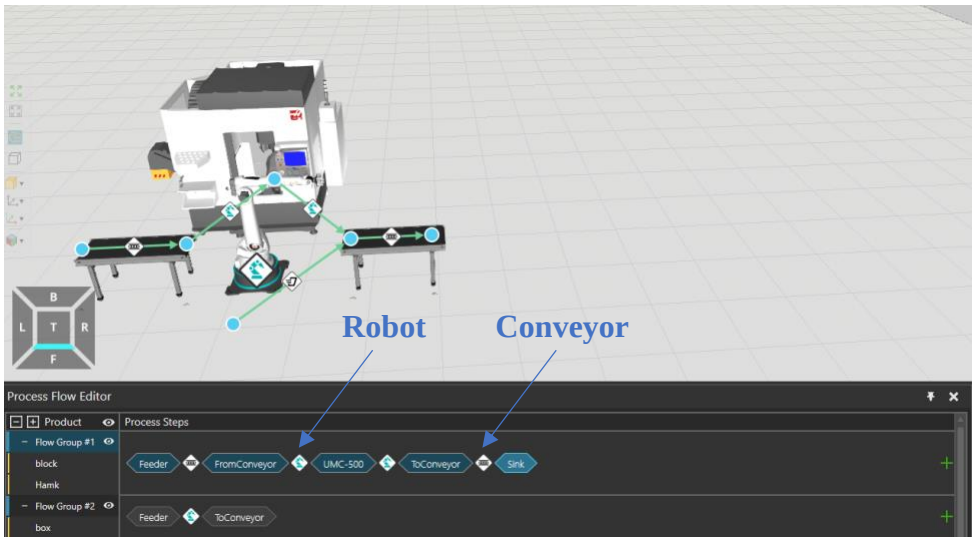

Figure 21 Process flow editor

4.1.3 Process step

Every process step has a routine which exists of different statements. Figure 22 shows the statements of the machine and the conveyor shown in figure 21. The machine represents a milling machine where a solid block enters, and a different shape leaves the machine. There are statements to open and close the machine's doors, change the object, …

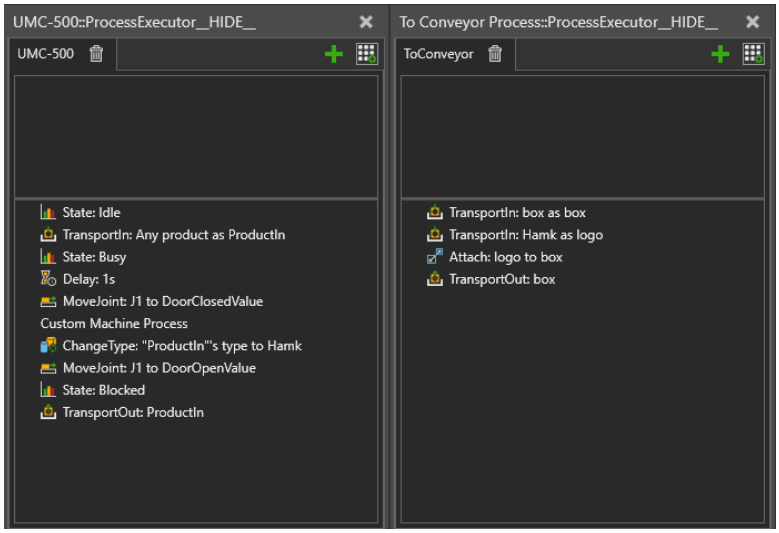

Figure 22 Process step editor

4.2 Robot programming

The robot programming function of the software's role is to create factory simulations with robots and test robotic arms. The program is designed by adding statements that provide the logic for the robot. Figure 23 shows an example of a program. It exists of linear motions, path motions, binary in-and outputs… Figure 23 shows the statements of the subprogram "Welding" the subprograms sequences are called in the "Main" program.

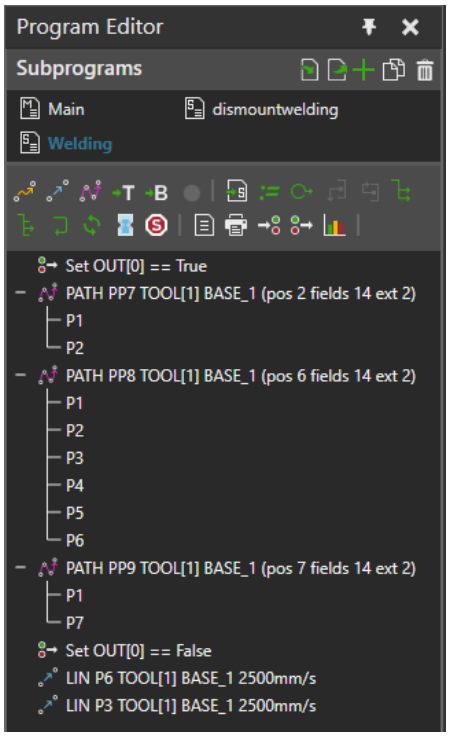

Figure 23 Robot program with statements

The binary in- and outputs can be used for signal actions of the robot. There are a few standard options, but there is also the possibility of creating custom signals. Figure 24 shows an example of the output with address 0, which enables the tracing of the TCP.

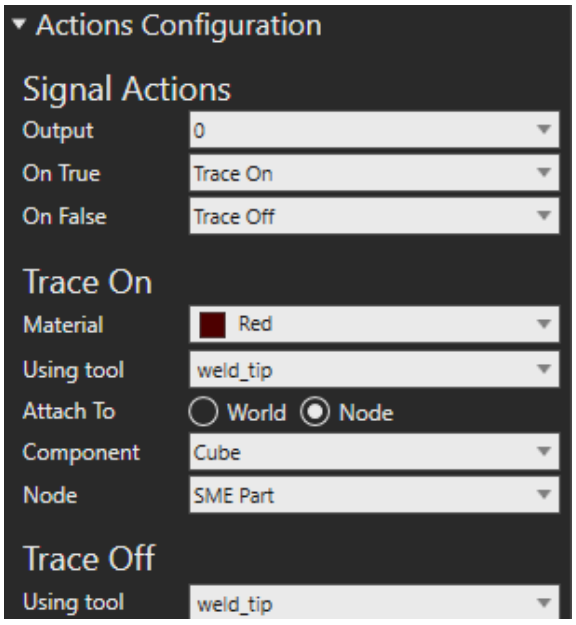

Figure 24 Actions configuration

The path statement allows the robot TCP (tool centre point) to follow a selected curve of an object. This does not automatically mean that it is the most optimal way of approaching that curve. It does not consider possible collisions with other objects or protrusions. A path statement exists of multiple mapped points on that curve, and with the jog window, the joints can be edited and then saved to the path statement. This can be done by changing the value separately for every joint, as shown in figure 25, or by dragging the robot by its TCP.

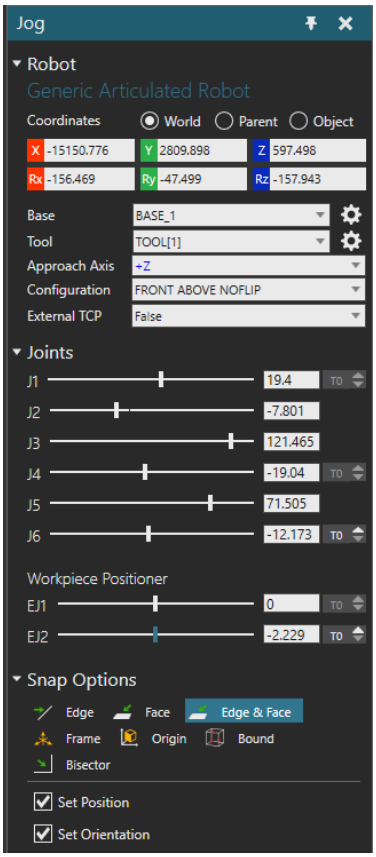

Figure 25 Joint values

4.3 Modelling

The modelling part of the software was the main functionality used from the software during this thesis. The purpose of modelling is to create objects/components from scratch and give them all the necessary attributes to use in the simulation.

4.3.1 3D models

It is possible to create simple shapes in visual components, as illustrated in figure 26. With these shapes, only limited, non-detailed components can be made. For this reason, it is possible to import over 30 different CAD file extension types.

Figure 26 Simple shapes for modelling

4.3.2 Movement/joints

Once a model is imported, it appears as one firm object. If the goal is to design an actuator with moving parts, the first step is to “explode” the model. This will split the model into separate parts, which can be selected to create a joint for a translational or rotational movement. Figure 27 shows the firm object on the left and the exploded model on the right.

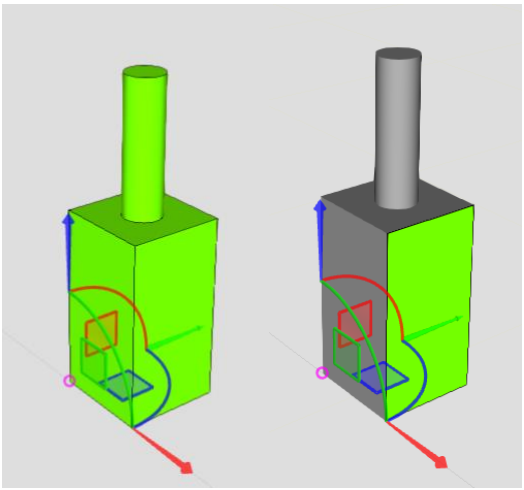

Figure 27 One firm object vs exploded object

4.3.3 Properties

Quick changes can be made in the component properties when a component is finished and used in a model. For example, figure 28 illustrates the properties of a raycast sensor. When a component is created, it has some standard properties, but the detection threshold of the sensor is manually added during the modelling. Properties can be added while modelling and are used in the Python code. This is to make the software more user friendly, and because of this, it is not necessary for a small change to open the Python script.

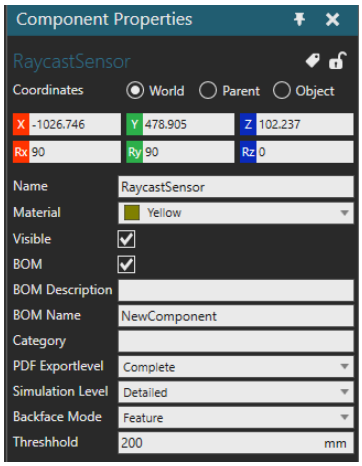

Figure 28 Raycast component properties

4.3.4 Behaviours

A behaviour is an action or a set of activities that a component can perform before or during a simulation. Every purpose of a component requires different combinations of behaviours. Figure 29 shows the list of all the available behaviours. The most important ones for this thesis are the signals (variables to connect to the PLC, signals used in the Python script,..), the sensors and the physics.

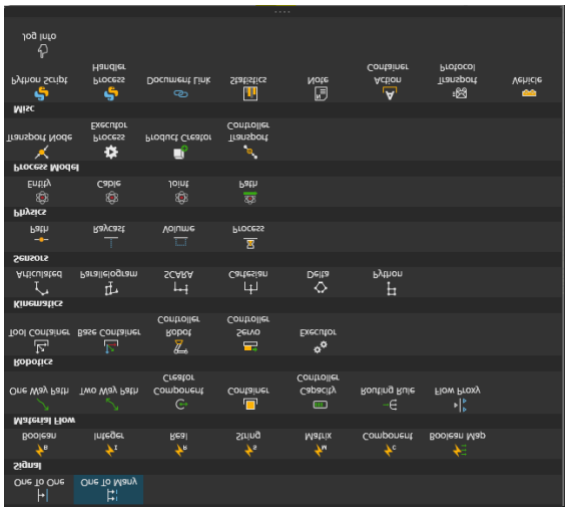

Figure 29 List of behaviours

4.3.5 Python script

Once the properties, behaviours and joints are added to the component, the next step is writing the logic in the Python script. The script's purpose is to manipulate components, commands and the application. A new Python script exists of 2 standard functions: OnSignal and OnRun, as shown in figure 30.

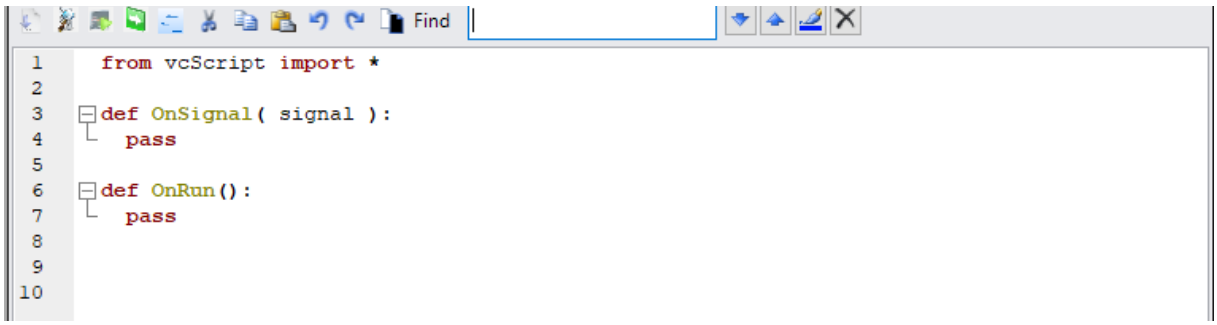

Figure 30 Python script in modelling

OnRun: this is the primary function of the script, and it is executed at the start of the simulation.

OnSignal: when a signal (behaviour) is connected to a script, it can trigger the OnSignal function when its value changes

4.3.6 Physics

To make the model as realistic as possible, it is essential that objects/components can interact with each other during the simulation. A physical entity is a behaviour which can be added, so the component is affected by the physics of the 3D world. Figure 31 shows the parameters that can be adjusted for the component.

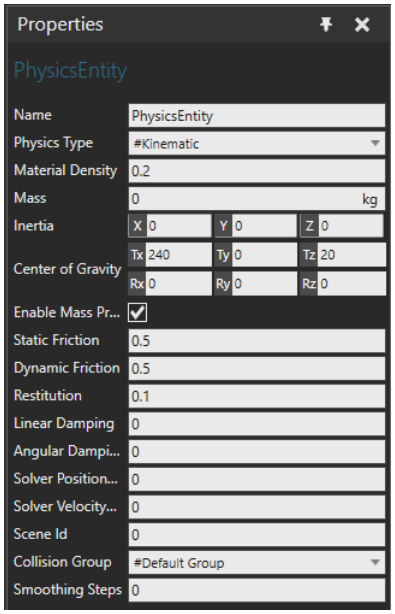

Figure 31 Physics entity properties

The physics type is an important parameter. It determines how the components are affected by physics during the simulation. Table 1 explains shortly what the options are.

Table 1 Explanation of physics entity types

Type	Meaning
#In Physics	Component is affected by gravity and other forces
#Out of Physics	Component is not affected by physics
#Kinematic	Component is not affected by gravity, but can be driven by other forces (conveyor, servo,…)
#In container	Similar to #In Physics but used for tracking the containment of the entity, for example from one path to another

A practical example is an abject on a conveyor which interacts with a pusher/pneumatic cylinder. The object on the conveyor has the “#In Physics” type because it undergoes all forces. If the goal is to simulate the process flow and not become too complex, it is better to give the conveyor the “#Kinematic” type. For example, when an incline conveyor belt is used, but no legs are below it to support it, then with the “#In Physics” type, the conveyor will drop, and the simulation will fail. The goal was to test the process flow, so the conveyor only needs to transport the physical objects, so there is no need for gravity to affect the conveyor.

4.4 Works library

The works library is a simple way to simulate a complex process based on creating tasks, and those tasks require the use of controllers, processes, resources… Whenever a single component is used from the library, it is necessary to add a works task control to the layout, as shown in figure 32. It holds route definitions and global variables used by the library components.

Figure 32 Works task control

The works process component is used to create tasks and can use different machines or resources to complete them. Figure 33 illustrates the component, but a nice feature is an option to change the looks to a conveyor belt so it can be connected in an industrial process. An example will be given in paragraph 5.4.

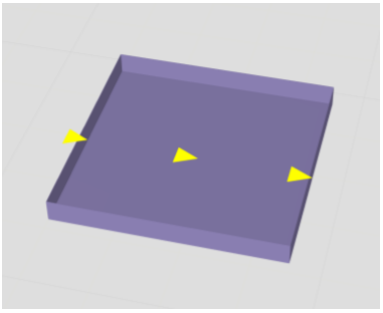

Figure 33 Works process

5 Results

5.1 Case 1: analysing production feed with process flow

The process flow part of Visual Components is not only for trying configurations of hardware setups or visualising a process but also to illustrate the logistics part in automation. Figure 34 shows a setup consisting of 2 conveyor belts, a robotic arm with a vacuum suction cup and a milling machine. The used components are imported from the eCatalog, and the design consists of 2 flow groups. The first flow group starts with spawning blue cubes and transporting them by the left conveyor belt to the pickup point for the robot.

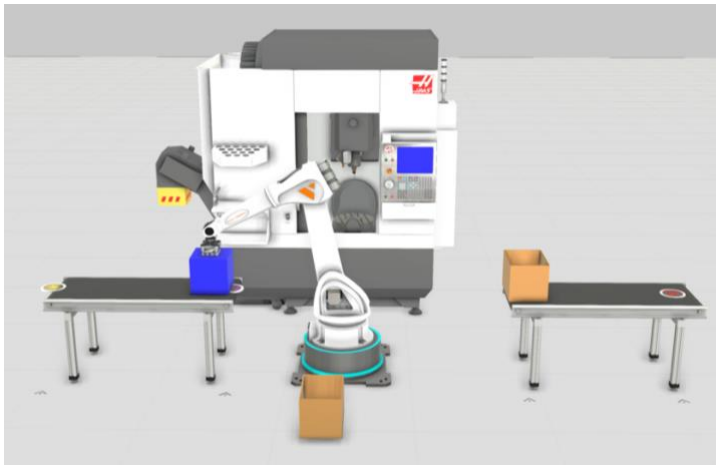

Figure 34 Supply of raw products

The robot places the cube inside the milling machine, the door closes, and the milling starts. Once finished after a specific process time, the robotic arm picks up the logo cut out of the cube and places it in a box, as illustrated in figure 35. The second flow group is for the supply of boxes, taken from the floor and placed on the conveyor by the robot.

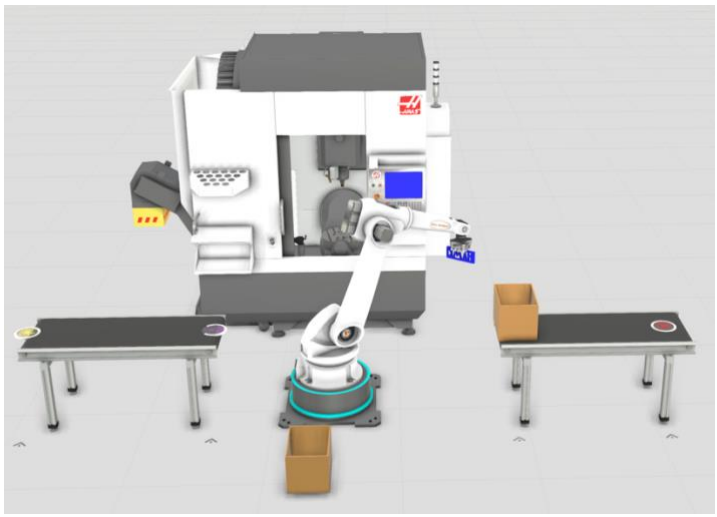

Figure 35 Placing a finished product in a box.

Figure 36 shows a finished product inside the carton box moving to the end of the process, where it will despawn. The point where the robot first places the box and after the finished product is called “to conveyor”, this is a process step. As explained in paragraph 4.1.3, statements can be added, and an important statement here is that the 2 product types (finished product and carton box) are attached. Because of this statement, the box and product will now be one object and move together, with the box as the “parent”.

Figure 36 Finished assembly

To illustrate some logistics with the software, two identical setups were created with the only difference in the supply of raw material (blue cubes). Figure 37 looks similar to figure 36 because in both pictures, a box with a finished product moves to the end of the process, but in figure 36, the supply feed is slower, and the first thing the robot does is pick a new carton box. In figure 37, there is a blue cube ready, and the robot places this first in the milling machine.

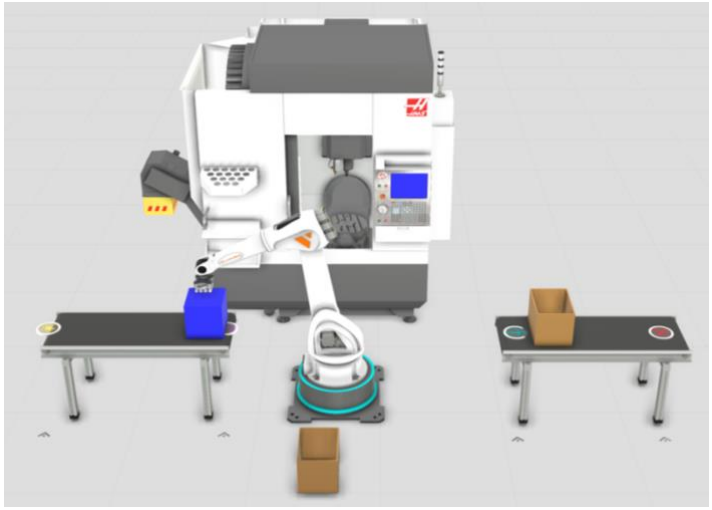

Figure 37 Difference in supply

The difference is that the milling machine in the setup with a faster supply feed is milling continuously, and there are no idle moments. This results in a higher production rate, as illustrated in figure 38. The number of created products is dependent on the speed of the conveyors, the processing time of the milling machine, the speed of the robot and the supply feed. The supply rate was 16 seconds slower at the first setup, resulting in a difference of 8 finished products after only 5 minutes.

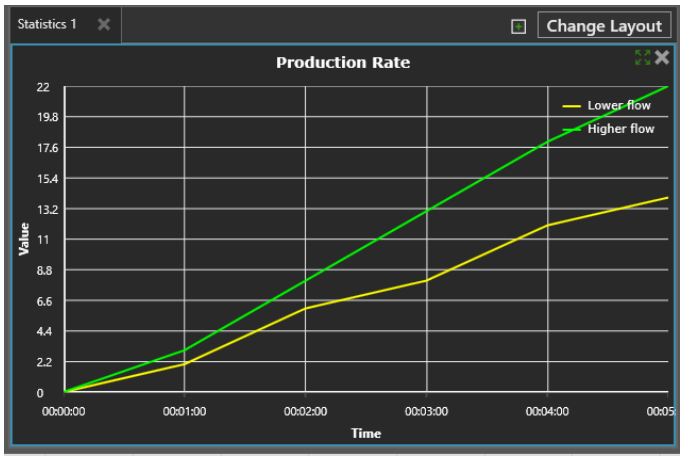

Figure 38 Production rate

There are six process steps in this setup which are explained more in detail in table 2.

Table 2 Process steps

Type	Symbol	Function
Feeder		Creates products with an adjustable interval.
Sink		Deletes/despawns products which arrive on this point.
From conveyor		Placed at the end of a conveyor, products wait on this point to be transported to the next step.
To conveyor		Placed at the beginning of a conveyor, products can be transported to this point.

Figure 39 shows the top view of the setup for a more precise idea of the used process steps, explained in table 2.

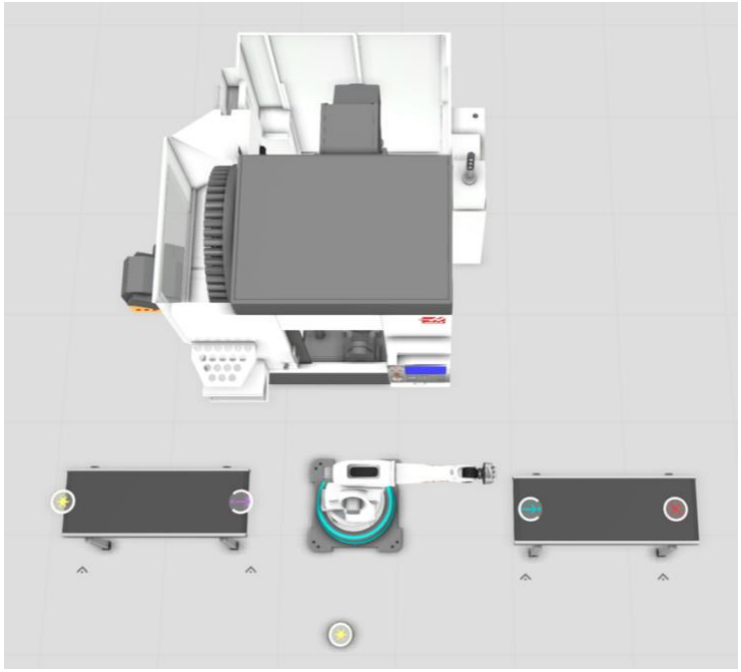

Figure 39 Used process components

5.2 Case 2: programming a welding robot

A welding cycle is chosen to demonstrate the functionalities of the robot programming part of the software. The setup exists of a robotic arm equipped with a welding torch and a workpiece positioner. The workpiece positioner helps the robot reach positions more easily or positions unavailable/unreachable before, like singularities. The robot and workpiece positioner start from a chosen initial state, and the welding torch moves to the starting point of the weld, as illustrated in figure 40.

Figure 40 Starting point of the weld

In paragraph 4.2 was explained that it was possible to use some pre-programmed actions. Figure 41 shows a close-up of the product and the welding tip. A boolean signal was set to true at the start point of the weld; this boolean is linked with the action “tracing”. Tracing marks the followed path of the TCP in a chosen colour. Figure 41 shows a pink line at the inner edge of the product; this is where the welding torch (TCP) has already passed, and the trace here represents the weld.

Figure 41 Tracing a path

The robot's path is determined by statements as mentioned in paragraph 4.2. In every position where a motion statement (PTP, LIN or curve) is added, the software will place a frame, as illustrated in figure 42. The TCP of the robot will go from frame to frame with the correct movement (PTP, LIN or curve). The red, blue and green axle systems in figure 42 represent the frames used for this simulation. The frames are not only points where the TCP passes through but also have an orientation which resembles the orientation of the welding torch.

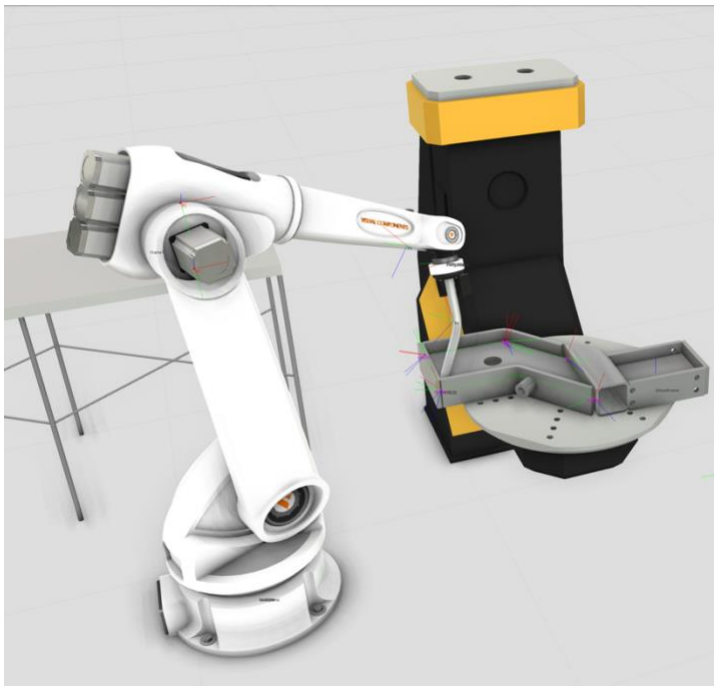

Figure 42 Frames of a path

Two essential properties of any statement are the defined base and the robot's tool. Figure 43 shows the properties of one of the curve statements. In this case, “Tool[1]” is the used TCP, and this was defined in the robot as the end of the welding torch. “Base_1” is located at the foot of the robot, and everything is oriented based on this reference point.

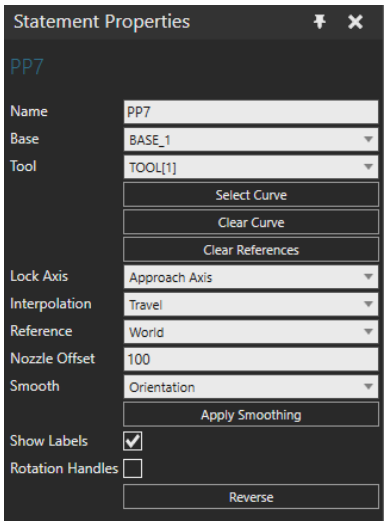

Figure 43 Properties curve statement

Figure 44 shows the end position of the products. At the end of the welding cycle, the robot moves to the table and detaches the welding tool. This action was done again by setting a bool at the end of the program.

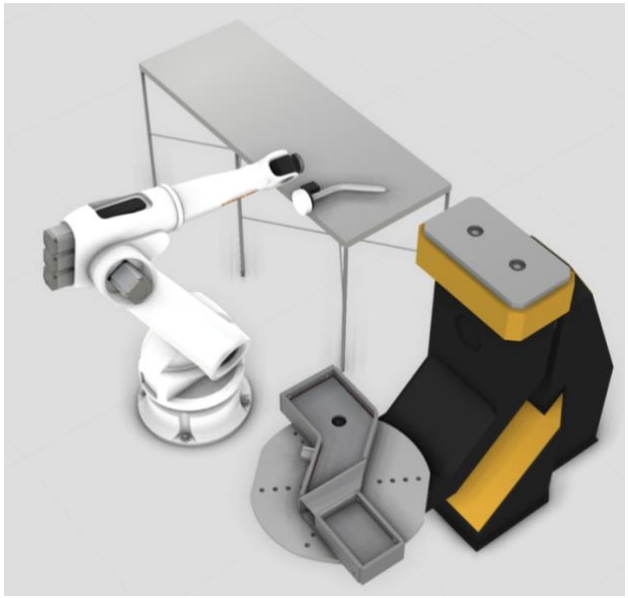

Figure 44 End of the welding cycle

5.3 Examples of modelled components

Modelling was the primary part and goal of this thesis. To investigate if it was possible to recreate the laboratory hardware setups, a list was made of the used components in these models. After this, component by component was created with the software until a combination could be created with the result of a possible laboratory setup.

Table 3 List of components

Type	Function
Feeder	Responsible for creating/spawning products
Colour/material sensor	Returns a boolean signal based on the colour or material of a product
Optical sensor	Returns a boolean signal based on the interference between transmitter and receiver
Pneumatic linear actuator	Push/manipulate a product
Linear vacuum actuator	Ability to pick up products
Pneumatic rotary actuator	Push/manipulate a product
Conveyor belt	Transports the products

5.3.1 Product feeder

The product feeder is a component that spawns in the products processed by the rest of the model. In the eCatalog library from Visual components, few feeders exist, physical feeder, standard feeder, and feeder with batch… As mentioned before, physical properties were added to make the models as realistic as possible. This means that the products created also need physical properties. For this reason, a custom product feeder was designed with the possibility of creating physical products and the option of making products from a batch in random order. Figure 45 shows the custom product feeder, which created a red physical cylinder. The products appear in the centre of the feeder and move to the edge, where the second frame is placed (blue arrow on figure 45).

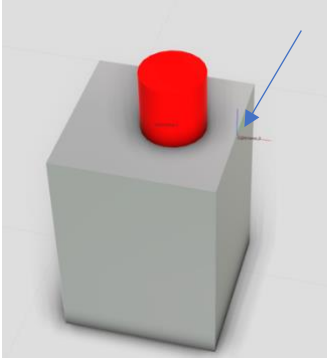

Figure 45 Custom product feeder

The location of the second frame in figure 45 is also the place where a “OneToOneInterface” is created. This interface makes it possible to connect the feeder with other components with a “OneToOneInterface”. Figure 46 shows all the behaviours of the custom feeder. The path makes it possible to move the object from the centre to the edge based on frames. It moves an object in a linear movement from frame to frame.

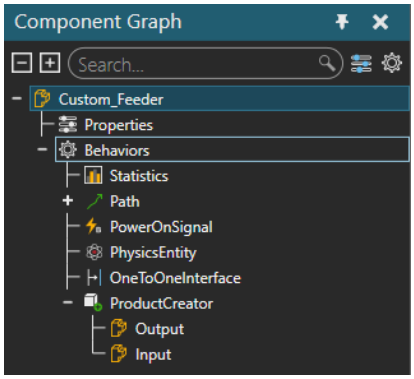

Figure 46 Component graph of custom feeder

Another behaviour that is added in figure 46 is “ProductCreator”. This is what spawns the products and has a property to customise the way of creating products or, in other words, to choose the “FeedMode” (single, batch, distribution).

Once the model is created and used to create a setup, products need to be added, so it knows which ones to create. To create a laboratory setup similar to the existing hardware setups at HAMK, the feeder required the ability to create multiple objects in random order. The first step is adding the different product types in a flow group, as mentioned in paragraph 4.1.1. The FeedMode has to be set on distribution mode to create different products in random order, as shown in figure 47. The figure also shows that 3 product types are added with a customisable probability of spawning.

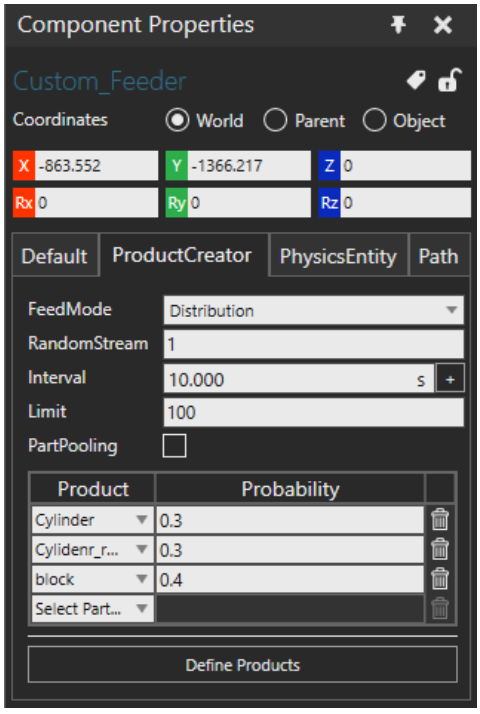

Figure 47 ProductCreator properties

5.3.2 Sensors

The setups at HAMK are equipped with two types of sensors, as mentioned in table 3. The colour/material sensors detect specific products, and the optical sensors detect any product passing through.

5.3.2.1 Optical sensors

To recreate an optical sensor in VC, the first step is to design/import a CAD design. During this thesis, the focus was not on creating the most complex 3D drawings but on creating functional models. Figure 48 illustrates a 3D model where a raycast behaviour is added. A raycast behaviour is a frame added to the model where the red beam points via the positive Z-axis. The red beam represents the laser beam from the hardware version.

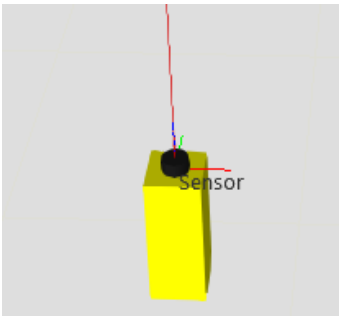

Figure 48 Raycast sensor

The "RaycastSensor" behaviour requires a boolean signal which triggers whenever a component passes through the beam. A real signal is optimal, detecting the distance to the object. These two signals connect with the Python script so that whenever a product triggers the sensor, the "OnSignal" function in the script triggers. More information about the Python script of a raycast sensor can be found in annexe: A.

Figure 49 shows the necessary behaviours for a raycast sensor. The "Sensor" Boolean is added and gets set/reset in the Python script based on the real signal's value, and its purpose is to be used as an input for the PLC. If the goal was just to detect any object, the boolean signal could also be used as a connected variable for the PLC, but the goal was to trigger the sensor depending on the size of an object. Noteworthy is that the Sensor boolean signal cannot be connected to the Python script because this would trigger the "OnSignal" function again.

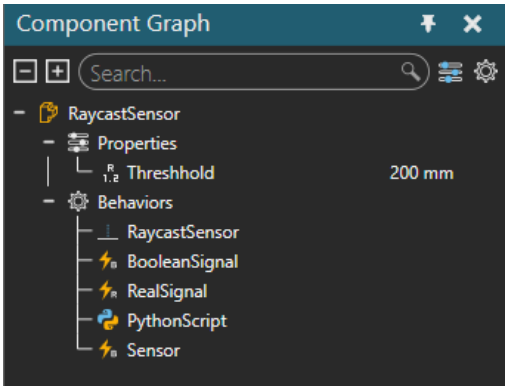

Figure 49 Component graph of a raycast sensor

The “RaycastSensor” behaviour has a few critical properties. The max range determines the length of the beam and the maximum detection range. The detection threshold is the distance where the sensor triggers the boolean signal. Figure 50 also shows the linked bool and real signal mentioned before.

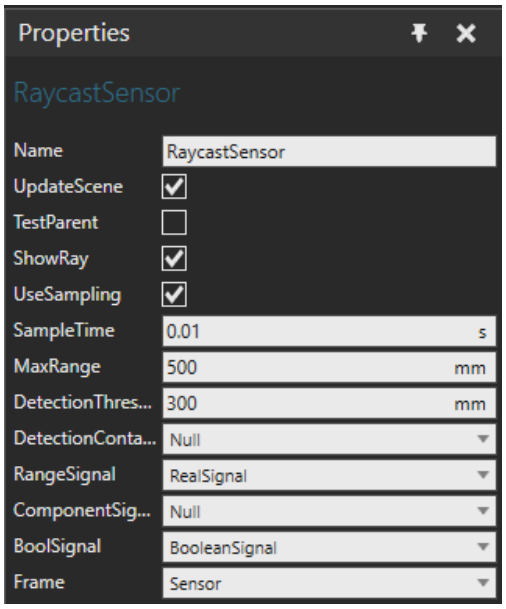

Figure 50 Properties of a raycast sensor behaviour

5.3.2.2 Material/colour sensor

To recreate a sensor that measures a specific attribute of a product is a bit more complicated than just detecting any product or any distance to a product. The solution is to create a volume sensor. As the name says, it measures in a specific customisable volume. The measured volume is determined by two frames, as shown in figure 51.

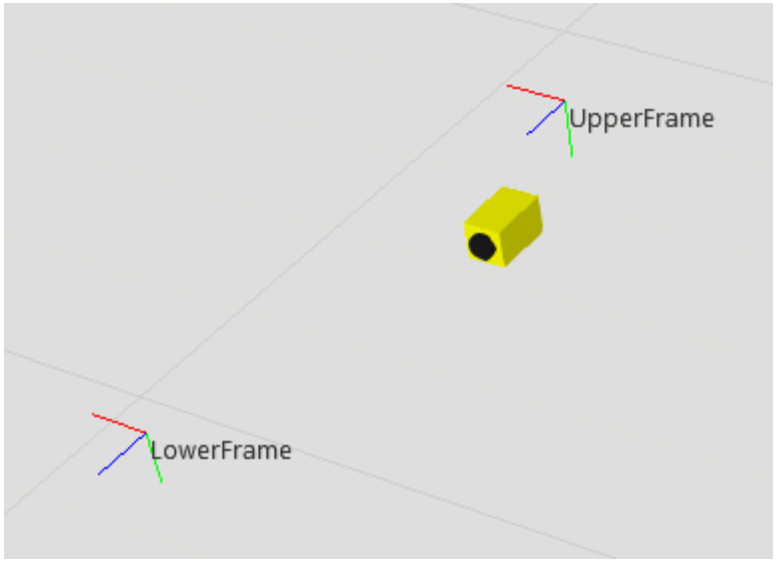

Figure 51 Principle volume sensor

Figure 52 shows the measured volume of the sensor (orange volume). When a product comes inside the volume, different signals can be triggered.

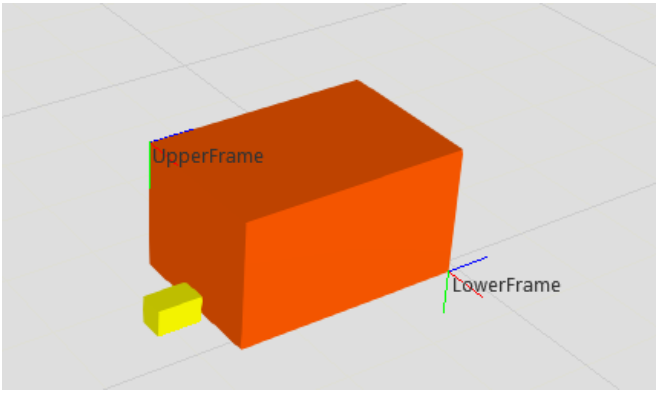

Figure 52 Measured volume

Different trigger points can be selected. Figure 53 shows the properties, and the parameter "TestMethod" is set on "Center inside", which means the centre point of the product must be inside the volume before it triggers. Another possibility would be that the entire product had to be in the volume. The big difference with the raycast sensor is that there is no real signal but instead a "ComponentSignal". A component signal consists of all the attributes/properties of the product passing through the volume. This could be the name of the product, the material, product ID…

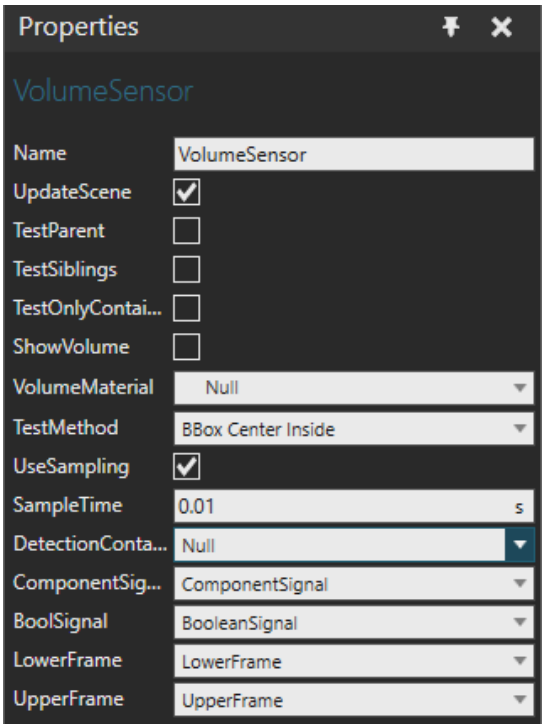

Figure 53 Properties of a volume sensor behaviour

The necessary behaviours are shown in figure 54 below. Instead of setting a threshold distance like the raycast sensor, this volume sensor has the material as a parameter. The setup in figure 54 will trigger the boolean “MaterialDetection” if the product passing the sensor has white as its material. The “BoolenSignal” is again triggered whenever a product passes through the measuring volume and will start the Python “OnSignal” function. More information about the Python script can be found in annexe: A.

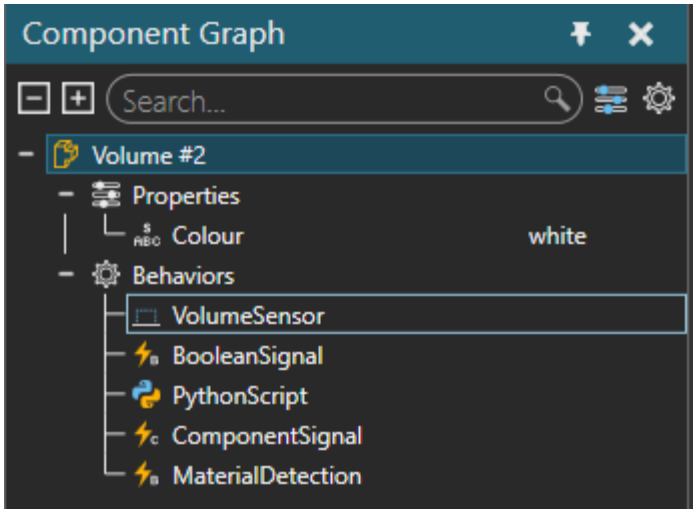

Figure 54 Component graph of a volume sensor

5.3.3 Actuators

As mentioned in table 3, there are three types of actuators in the setups at HAMK. The pneumatic cylinder is used to redirect the path of products. The rotary pneumatic actuator is used to hold/block products moving on a conveyor. The cylinder with vacuum is used to pick up products and place them at another location. The modelling of actuators is more complex than sensors because there are moving parts, and there has to be a physical interaction between the actuator and the product.

5.3.3.1 Pneumatic cylinder

The modelling of an actuator starts as well from a CAD file. Figure 55 below is the created 3D drawing for the pneumatic cylinder. Together with the grey push head, the orange cylinder will move with a linear movement. This is done by exploding the model and extracting a joint, as mentioned in paragraph 4.3.2.

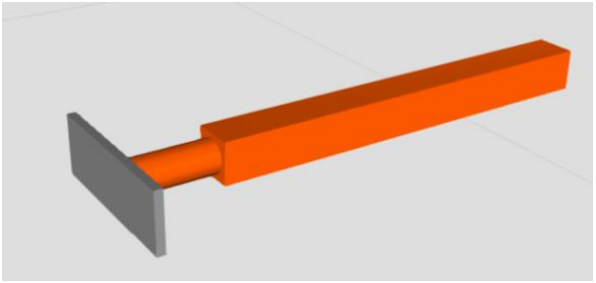

Figure 55 Pneumatic cylinder

Once the joint/link is created, it will also appear in the component graph of the actuator, as illustrated in figure 56 (“Link_1”). The products created by the feeder were physical products, and for the actuator to interact with them, a physics entity has to be added to the joint. The “Servo Controller” is a new behaviour that allows the configuration or control of a joint/mechanism using forward kinematics. The properties are all parameters for the movement, like the speed, acceleration and the length of the movement (“PushJoint_Open”).

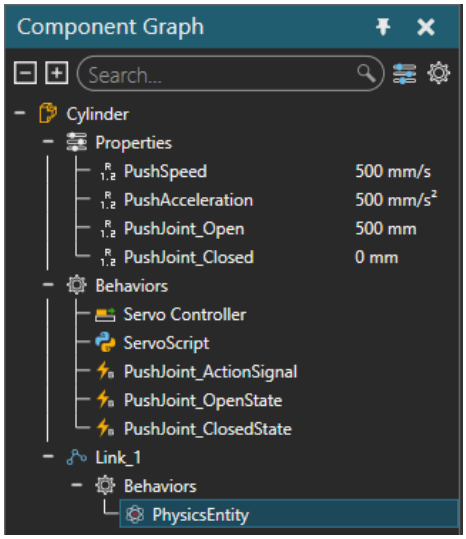

Figure 56 Component graph of a pneumatic cylinder

The booleans: “ActionSignal, OpenState and ClosedState” are linked to the Python script as shown in figure 57 and connected to the PLC. The action signal is connected to an output from the PLC, and when this output is set to true, the “OnSignal” function in the Python script will start because of the connection to the script. The open and close states are used in the Python script, but they are connected to inputs from the PLC and resemble the end of range contacts.

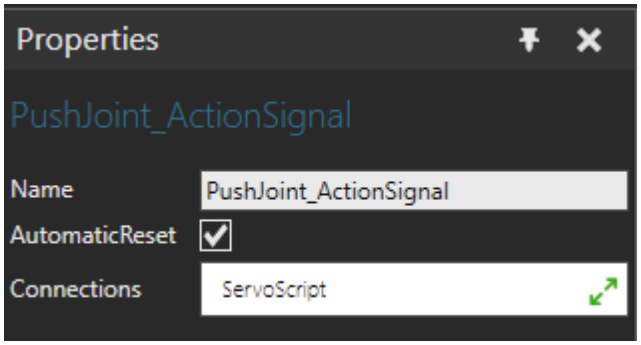

Figure 57 Connection between boolean and Python script

Figure 58 shows the properties once a link is extracted. Automatically a servo controller behaviour gets added to the component. Noteworthy for the Python script is that the name of the joint ("PushJoint" in the figure) must be the same as the first part of the names of the properties and signals in figure 56. For more information about the Python script and the reason for this can be found in annexe: A.

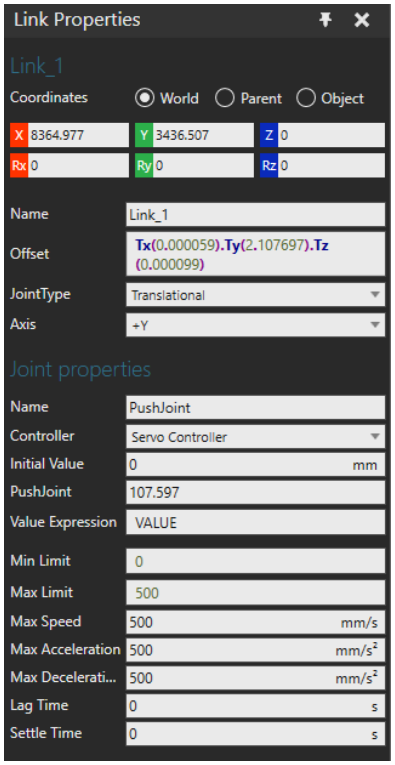

Figure 58 Properties of a joint/link

5.3.3.2 Pneumatic rotary actuator

The goal of the rotary actuator in figure 59 is to stop/block products based on logic from the PLC. The grey 3D shape will make a rotational movement around the black cylinder to resemble a rotary actuator. This movement is created the same way as paragraph 4.3.2 explained. The only difference in the joint/link properties is that the "JointType" is rotational and not translational.

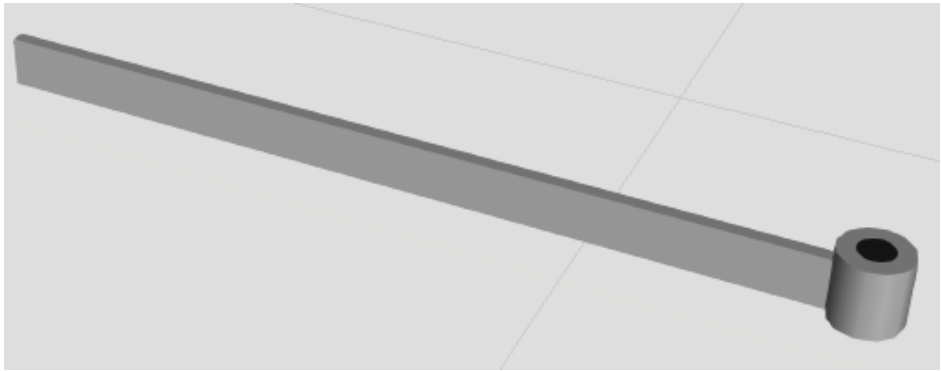
Figure 59 Pneumatic rotary actuator

Figure 60 shows the component graph of the rotary actuator. It is very similar to the linear actuator (pneumatic cylinder). The only difference is that the properties do not have the same units. Here the size of the movement is defined in degrees, and the speed and acceleration are angular.

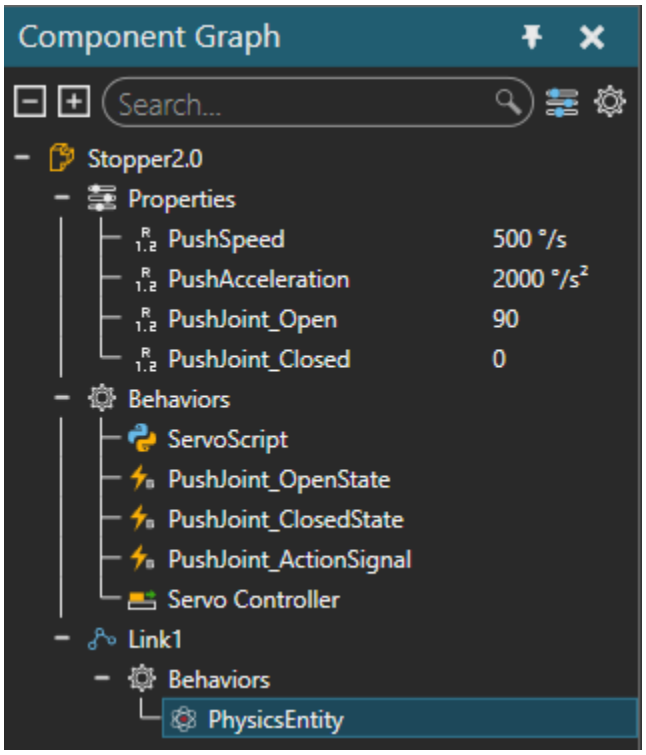

Figure 60 Component graph of a rotary actuator

5.3.3.3 Linear vacuum actuator

This actuator its purpose is to do a linear movement and then grab a product with the white suction cup, as illustrated in figure 61. These actions are the recreation of a vacuum actuator from the HAMK models. Whenever a product is grabbed from above, it can only be moved vertically; for that reason, the combination of a pneumatic linear actuator and vacuum actuator is necessary to reposition it horizontally as well. This version will be used in the combination of all models in paragraph 5.4.

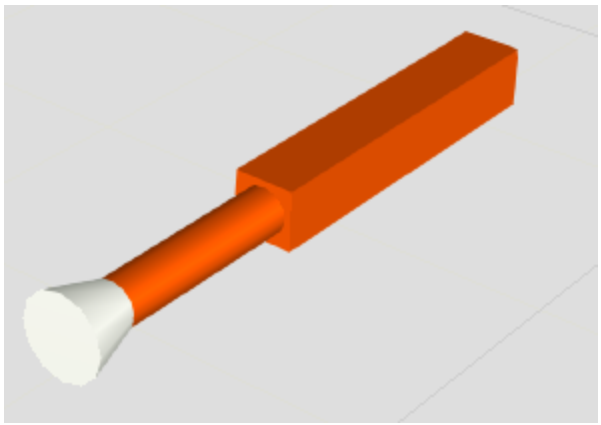
Figure 61 Linear vacuum actuator

The vacuum actuator's component graph shown in figure 62 is similar to the standard linear actuator except for the grabbing function. The modelling part of the software has some wizards available to create standard behaviours and properties for specific components (conveyor, end effector, workpiece positioner…), but none for the grabbing of a product. With the help of a community member on the forum of VC [16], who created a grasp action wizard, and a minor modification, the actuator could now pick up products. The "PhysicsEntity_2" has to be added manually to the "GraspDetection" link, and it must be of type "#Kinematic" to grab products that are moving on a conveyor belt.

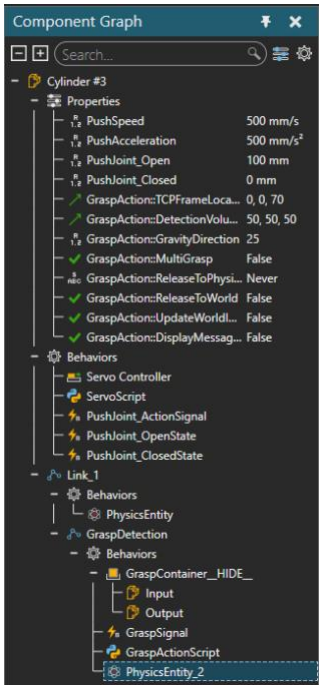

Figure 62 Component graph of a vacuum actuator

5.3.4 Conveyor belts

For the conveyor belts, there are standard models available in the eCatalog. There was only one physical conveyor available, but converting a regular conveyor to a physical one with a few modifications is possible. Physical objects will fall through a standard conveyor because they do not have any reaction forces which hold the physical products on the belt. Figure 63 shows the used model out the eCatalog.


Figure 63 Conveyor

Modifications to make:

- Change the normal path to a physics path
- Add physics entity with the #Kinematic as type
- Change the container from the interfaces to the physics path as shown in figure 64

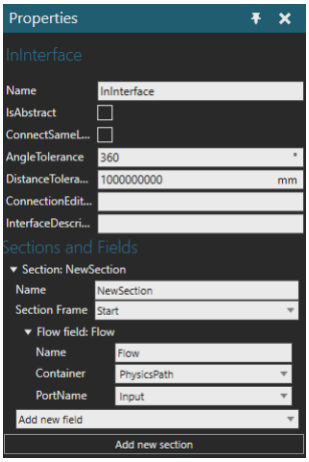

Figure 64 Interface properties

- Select the root and change the physics collider to #box as shown in figure 65

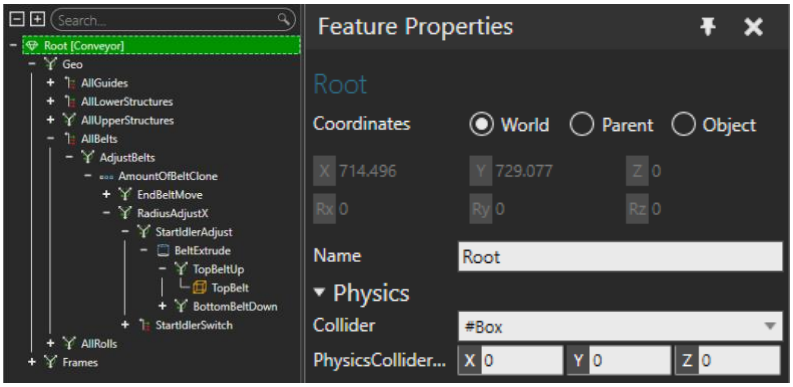

Figure 65 Root and physics collider

Colliders are objects that can react/interact with physical objects/products during a simulation. This will automatically be created once a collider type is added and no physics entity exists. Depending on the type, an entity defines the physical properties (to which forces it reacts).

5.4 Combination to a laboratory setup

In this paragraph, a setup is created and explained. It exists of all the models from paragraph 5.3 and some functionalities from paragraph 4. Figure 66 shows the top view from the setup. The setup idea is that three types of physical products are created and sorted based on size and colour. Instead of just despawning the products after they are sorted, the setup uses the works library to make it a whole process.

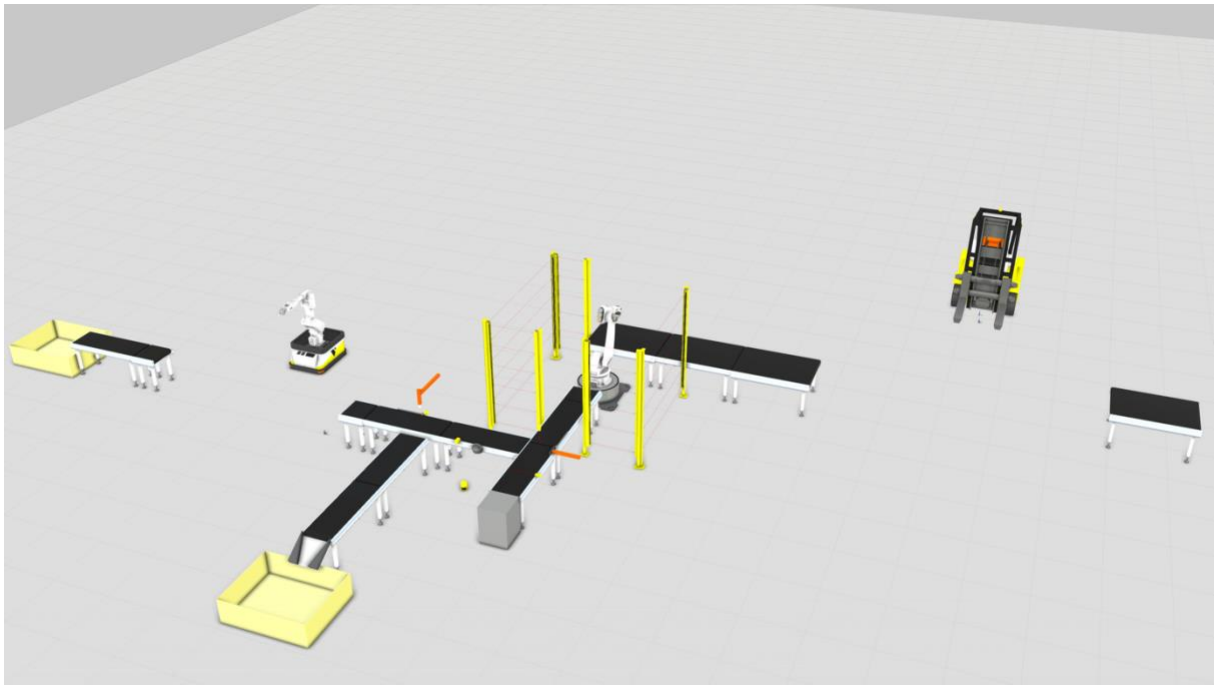

Figure 66 Total view of the setup

Figure 67 illustrates the used product types in the setup, and the product feeder will create these products in random order. The blocks will be separated from the cylinders based on size. The two cylinders have the exact same shape, so they can only be separated based on the material/colour. All of them have physical properties so that they will interact with the actuators and each other.

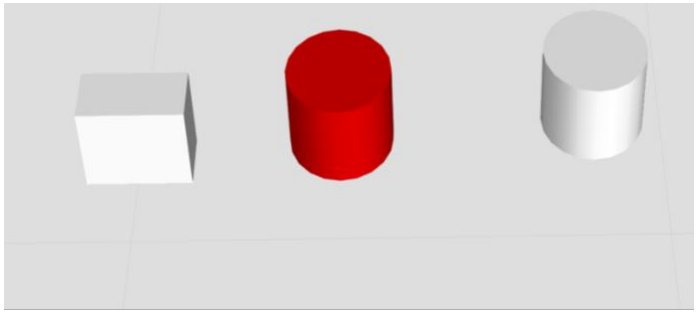

Figure 67 Used product types

The process starts with creating products by the product feeder, located at the right of figure 68, and the first sensor they pass is a raycast sensor that will measure the size. In the Python script, the real signal's value gets compared with a constant value (threshold property). The cylinders have a bigger size, which will set a boolean signal to true, connected to an input to the PLC. If the white block would pass, the real signal's value would be bigger because the block is smaller, and the boolean would not be set to true.

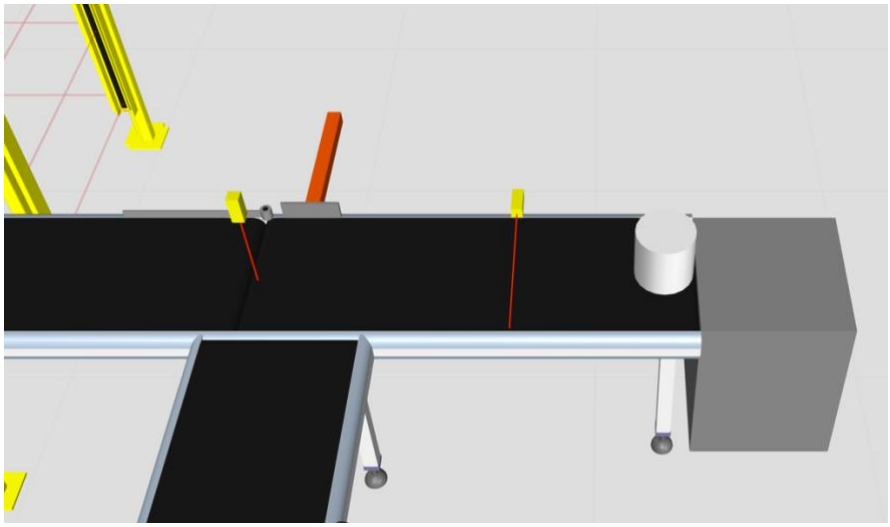

Figure 68 Step 1: sorting based on size

When the first raycast sensor detects a cylinder, the pneumatic rotary actuator will hold the cylinder, and the second raycast sensor will detect an object waiting to be pushed. Based on these conditions, the linear actuator will now push the cylinder on the perpendicular conveyor belt, as shown in figure 69.

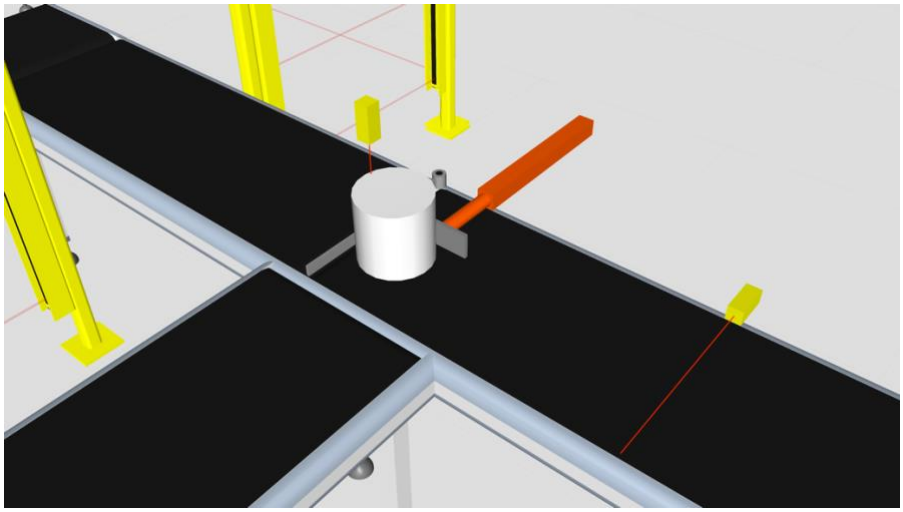

Figure 69 Step 2: cylinders get separated from blocks

Figure 70 illustrates the sorting process of the cylinders. The first sensor (yellow with a black lens) is a volume sensor and can detect the product's material/colour of the products. Whenever the product has the colour "white", it will activate the pneumatic rotary actuator to stop the cylinder. The raycast sensor will detect a product waiting, and then a combination of a linear actuator and a linear vacuum actuator will pick up the white cylinder and place it on the perpendicular conveyor belt.

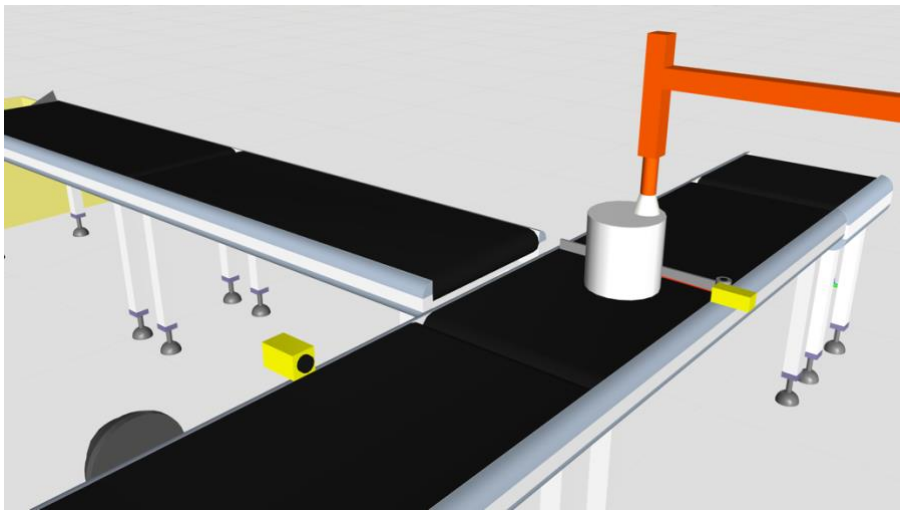

Figure 70 Step 3: sorting cylinders based on material/colour

Once the white cylinder is placed on the conveyor, as shown in figure 71, it will be transported into the yellow container. The container is a physical object, so the cylinders cannot roll out of the container.

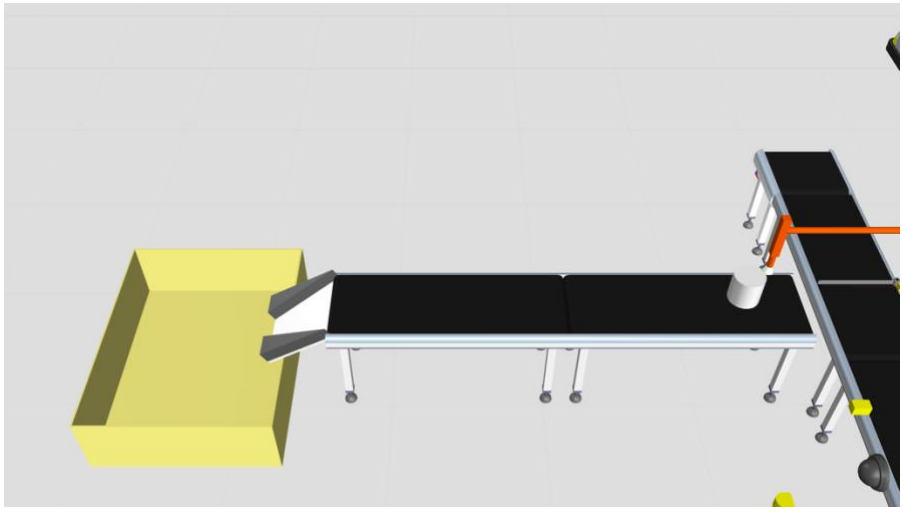

Figure 71 Step 4: end of process for the white cylinders

The first part of the model separates products based on size. The rotary actuator stops the cylinders, but the blocks keep going straight on the conveyor to the robotic arm. The robotic arm will place the blocks on a euro pallet in a fixed pattern.

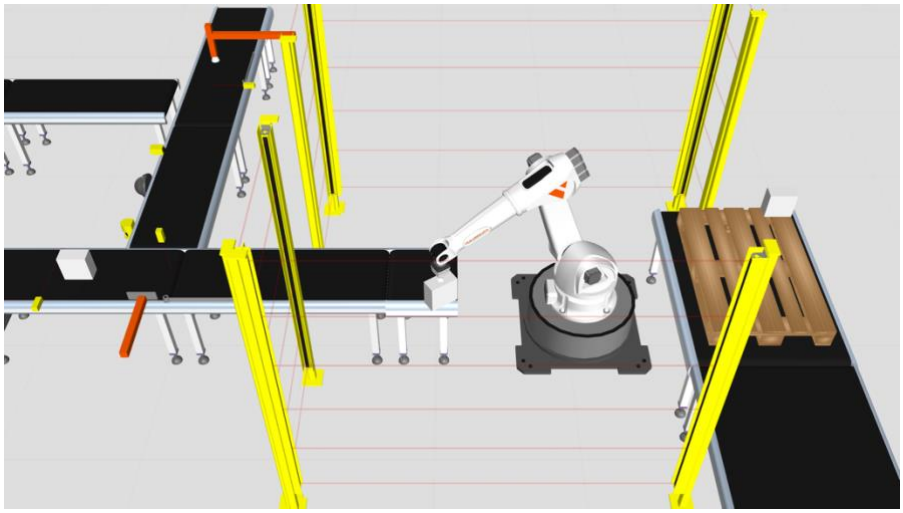

Figure 72 Step 5: processing white blocks

As mentioned before in paragraph 4.4, a works process can be visually changed to a conveyor belt, as illustrated in figure 73. At the right side of the figure, there is a list of tasks added to this works process. The first task is transporting in a product (white block), and then there is a feed task. The feed task has a name and a parameter which is the name of the tool used.

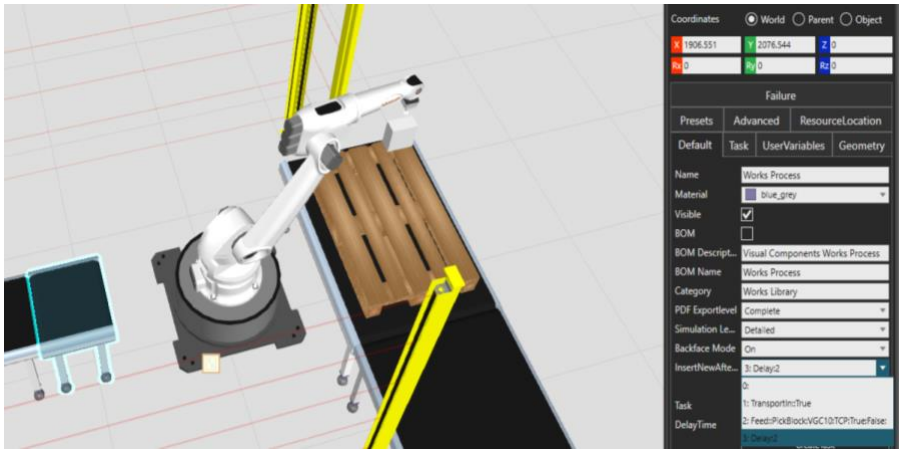

Figure 73 Work process feed task

As shown in figure 74, the task's name is “PickBlock”, and the used tool is VGC10; this is the tool's name attached to the robotic arm.

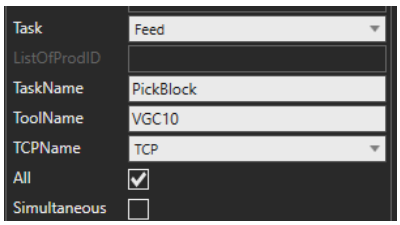

Figure 74 Feed task parameters

When using a robotic arm together with the works library, it has to be placed on a works robot controller; this is the blue selected object in figure 75 beneath the robotic arm. In the right side of the picture, there is a parameter “Tasklist”, and here is the same name as the TaskName of the Feed task. This is necessary, and the robotic arm will now pick up a white block whenever the feed task is executed.

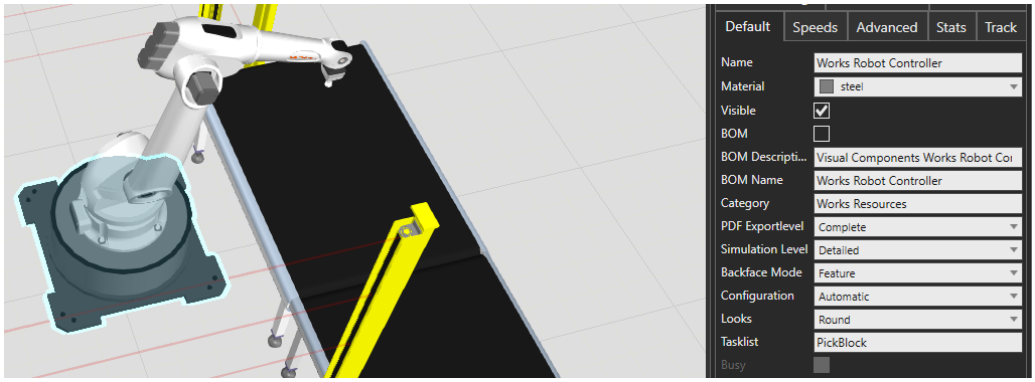

Figure 75 Robot controller parameters

At this point, there is a feed task and a machine/resource to execute the task, but there is still no end position. Figure 76 illustrates another works process with the visual looks of a conveyor, and on the right, the task list is visible. The first step is creating a euro pallet, and task 2 is a NeedPattern task. This asks specifically for white blocks, and the pattern means it is asking for more than one block. A custom amount and placement pattern can be made. After this, the pallet and blocks are merged to continue as one object and transported out once the pattern is filled with white blocks.

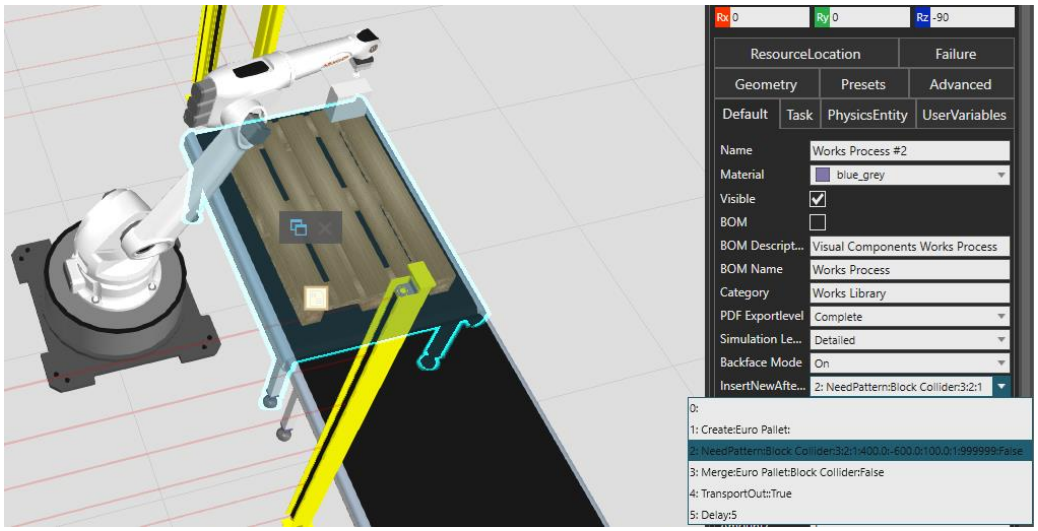

Figure 76 Works process need task

For this setup, a pattern of 6 white blocks was chosen. The filled euro pallet will move to another works process, and a forklift will pick it up and move it to a position where it will despawn, as shown in figure 77. The principle of the forklift is the same as the robotic arm, only is the forklift now the machine/resource that is executing the tasks.

Figure 77 Step 6: end of the process for the white blocks

The volume sensor detects a red colour and will not activate the boolean connected to the PLC, so the rotary actuator and vacuum actuator will not be activated. Instead, the red cylinders move to the end of the conveyor, where an AGV (automated guided vehicle) with a robotic arm will pick them up. The AGV is again the machine/resource executing the feed task, but the tool name is the name of the tool attached to the robot. The robotic arm and controller are attached to the AGV, so the software automatically knows that the tool has to pick up the cylinder and the AGV is transporting it. The AGV normally functions as a conveyor and will transport the cylinder on him, but now the robotic arm is placing the cylinder onto the AGV, as shown in figure 78.

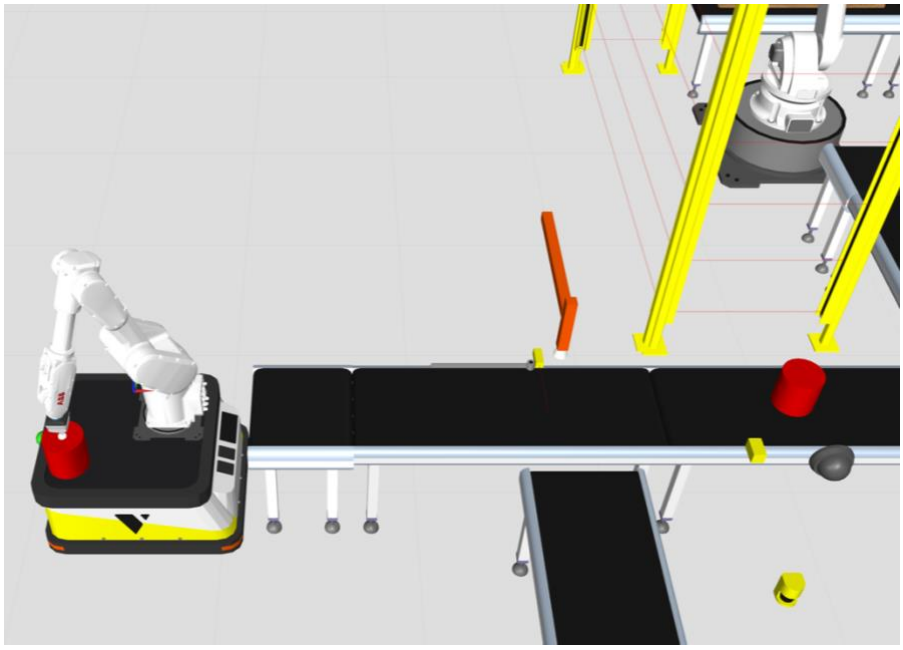

Figure 78 Step 7: picking up red cylinders

Figure 79 also shows a container into which the red cylinders will fall. If the red cylinders did not have physical properties, they would all fall in the exact same place in the exact position, and it would show as one cylinder. The figure clearly shows that the cylinders have physical properties and have fallen “randomly” based on physical forces (gravity, friction…)

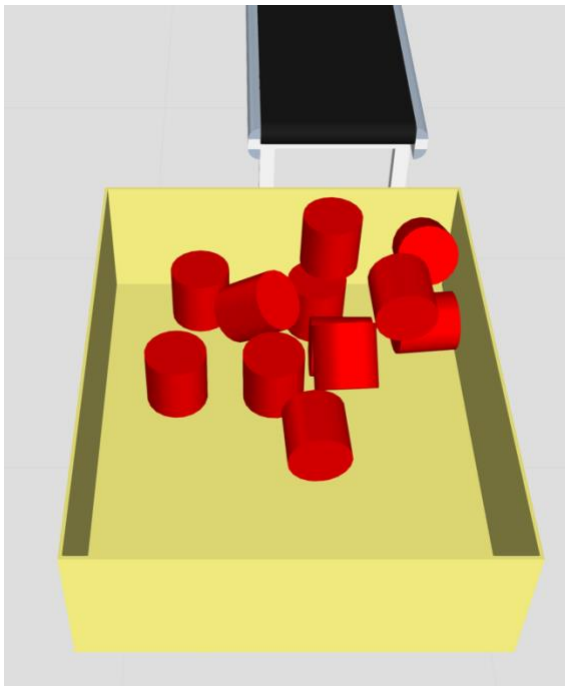

Figure 79 Step 8: end of process of the red cylinders

The last feature that was added is a safety scanner. Figure 80 shows a cylinder lying in the warning zone (yellow), and a yellow light has turned on to give a warning signal. When any object enters the critical area (red), the conveyors and product feeder will shut off as a safety measure. This was added to illustrate that safety measures can also be implemented with the software.

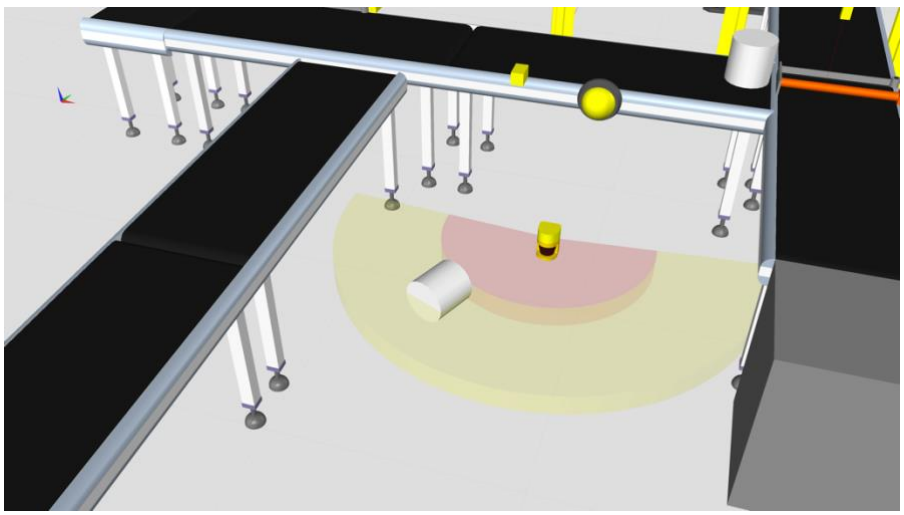

Figure 80 Safety scanner

6 Conclusion

This thesis starts by explaining what a digital twin is and what the benefits/applications are. It is a technology that is becoming more and more popular, and for this reason, HAMK wants to teach students to work with DTs.

The literature study explains what a digital twin is and discusses the leading applications. HAMK works with Beckhoff and Siemens, and for this reason, OPC UA was chosen as the communication protocol. The review of different software packages made it clear that for the requirements of HAMK, Visual Components was the obvious choice. The software has a lot of tutorials and a good community available, which is perfect for students.

The system structure exists of a software PLC running an OPC UA server, and Visual Components is the OPC UA client. Once the connection was established, it worked without any problems, unlike [13] claimed in their research. According to [14], it was possible to connect only 1 PLC with Visual Components, but the connection with multiple PLCs was successful during this thesis. Visual Components allows to connect to multiple OPC UA servers, and PLC SIM-advanced allows to run multiple virtual software PLCs and so multiple OPC UA servers.

The main objective was modelling components, but the software had so much more to offer. Therefore, in paragraph 5, two cases were created to give a short introduction to the functionalities of the software besides modelling. A first case intends to illustrate the logistics side of the automation sector by generating charts of production parameters of a small process. The second case was designed to illustrate the possibilities with the robot programming part of the software. This was done by creating a welding example with a product on a workpiece positioner.

Modelling of components was the main part of this thesis. The main objective was to model components and, with these components, have the possibility to recreate the hardware setups at HAMK. This was done successfully, and to illustrate this, a combined setup was designed where all the models and other functionalities, like the works library, were used.

HAMK was provided with all the files of the created models, the laboratory setup, and the corresponding PLC code. To allow students to develop their own models with Visual Components, a manual which describes step by step how to create the models used during this thesis and to set up the communication was added as an annexe in this thesis.

Future work

Although a working setup was delivered and the models for all the components present in the hardware versions were created, more time would have only improved the delivered results. Because of the short period of time that was available for this thesis, due to the fact it was done on Erasmus, the design of the 3D models is purely functional. The models would have a more aesthetic look with more time available instead of just being practical.

Bibliography

[1] V. Mihai and D. Popescu, ‘Digital Twin Model for Assembly Robotic Cell’, *Int. J. Progress. Sci. Technol.*, vol. 24, no. 2, Art. no. 2, Jan. 2021, doi: 10.52155/ijpsat.v24.2.2612.

[2] ‘IEEE Xplore Full Text PDF’. Accessed: Mar. 11, 2022. [Online]. Available: https://ieeexplore.ieee.org/stampPDF/getPDF.jsp?tp=&arnumber=8477101&ref=aHR0cHM6Ly9pZWVleHBsb3JlLmllZWUub3JnL3N0YW1wL3N0YW1wLmpzcD90cD0mYXJudW1iZXI9ODQ3NzEwMQ==

[3] S. M. A. Kazmi, ‘METHODOLOGY FOR VALIDATING MECHATRONIC DIGITAL TWIN’, p. 74.

[4] F. Tao, H. Zhang, A. Liu, and A. Y. C. Nee, ‘Digital Twin in Industry: State-of-the-Art’, *IEEE Trans. Ind. Inform.*, vol. 15, no. 4, pp. 2405–2415, Apr. 2019, doi: 10.1109/TII.2018.2873186.

[5] T. Gabor, L. Belzner, M. Kiermeier, M. T. Beck, and A. Neitz, ‘A Simulation-Based Architecture for Smart Cyber-Physical Systems’, in *2016 IEEE International Conference on Autonomic Computing (ICAC)*, Jul. 2016, pp. 374–379. doi: 10.1109/ICAC.2016.29.

[6] J. Eyre, T. Dodd, C. Freeman, R. Lanyon-Hogg, A. Lockwood, and R. Scott, ‘Demonstration of an Industrial Framework for an Implementation of a Process Digital Twin’, Nov. 2018, p. V002T02A070. doi: 10.1115/IMECE2018-87361.

[7] H. Arnarson, ‘Digital twin simulation with Visual Components’, Jun. 2019, Accessed: Mar. 11, 2022. [Online]. Available: https://munin.uit.no/handle/10037/18109

[8] M. Ugarte, L. Etxeberria, G. Unamuno, J. L. Bellanco, and E. Ugalde, ‘Implementation of Digital Twin-based Virtual Commissioning in Machine Tool Manufacturing’, *Procedia Comput. Sci.*, vol. 200, pp. 527–536, 2022, doi: 10.1016/j.procs.2022.01.250.

[9] D. Bruckner *et al.*, ‘An Introduction to OPC UA TSN for Industrial Communication Systems’, *Proc. IEEE*, vol. 107, no. 6, pp. 1121–1131, Jun. 2019, doi: 10.1109/JPROC.2018.2888703.

[10] ‘OPC-UA-Interoperability-For-Industrie4-and-IoT-EN.pdf’. Accessed: Mar. 22, 2022. [Online]. Available: https://opcfoundation.org/wp-content/uploads/2017/11/OPC-UA-Interoperability-For-Industrie4-and-IoT-EN.pdf

[11] W. Sun, J. Wu, G. Xiao, and Z. Jin, ‘Research on selection of commercial industrial simulation software oriented to virtual commissioning’, *J. Phys. Conf. Ser.*, vol. 1906, no. 1, p. 012052, May 2021, doi: 10.1088/1742-6596/1906/1/012052.

[12] J. Taipalus, ‘3D-Virtualization of a Conveyor Machine’, p. 32.

[13] A. Liljaniemi and H. Paavilainen, ‘Using Digital Twin Technology in Engineering Education – Course Concept to Explore Benefits and Barriers’, *Open Eng.*, vol. 10, no. 1, pp. 377–385, Jan. 2020, doi: 10.1515/eng-2020-0040.

[14] K. Eriksson, A. Alsaleh, S. Behzad Far, and D. Stjern, ‘Applying Digital Twin Technology in Higher Education: An Automation Line Case Study’, in *Advances in Transdisciplinary Engineering*, A. H. C. Ng, A. Syberfeldt, D. Högberg, and M. Holm, Eds. IOS Press, 2022. doi: 10.3233/ATDE220165.

[15] ‘Siemens TIA and PLCSIM Advanced - User Manual’. Accessed: May 17, 2022. [Online]. Available: https://simumatik.com/learn/ThirdParty/PLC/siemens_TIA/

[16] ‘Grasp Action Wizard (Professional) - Extensions and Python Add-ons / Visual Components Add-on Samples’, *Visual Components - The Simulation Community*, May 16, 2019. https://forum.visualcomponents.com/t/grasp-action-wizard-professional/675 (accessed Jun. 01, 2022).

Annexes

A. Manual

This manual will explain how the communication is set up and how the components are modelled. If a particular term is unclear or not enough explained, look at the thesis above, where more information is available. A reference is sometimes made to the thesis above to make the manual not too large and complicated.

1 Communication between PLC and Visual Components

1.1 Tia portal

The first step is to create a new project in Tia Portal and add the hardware configuration. This manual will use OPC UA as the communication protocol, so adding a PLC that supports this protocol is essential.

1.1.1 Setting up the PLC

1.1.2 Assign an IP address to the PLC (192.168.0.1, for example)

1. Make sure the subnet mask is 255.255.255.0
2. Enable the OPC UA server as shown in figure 1 below

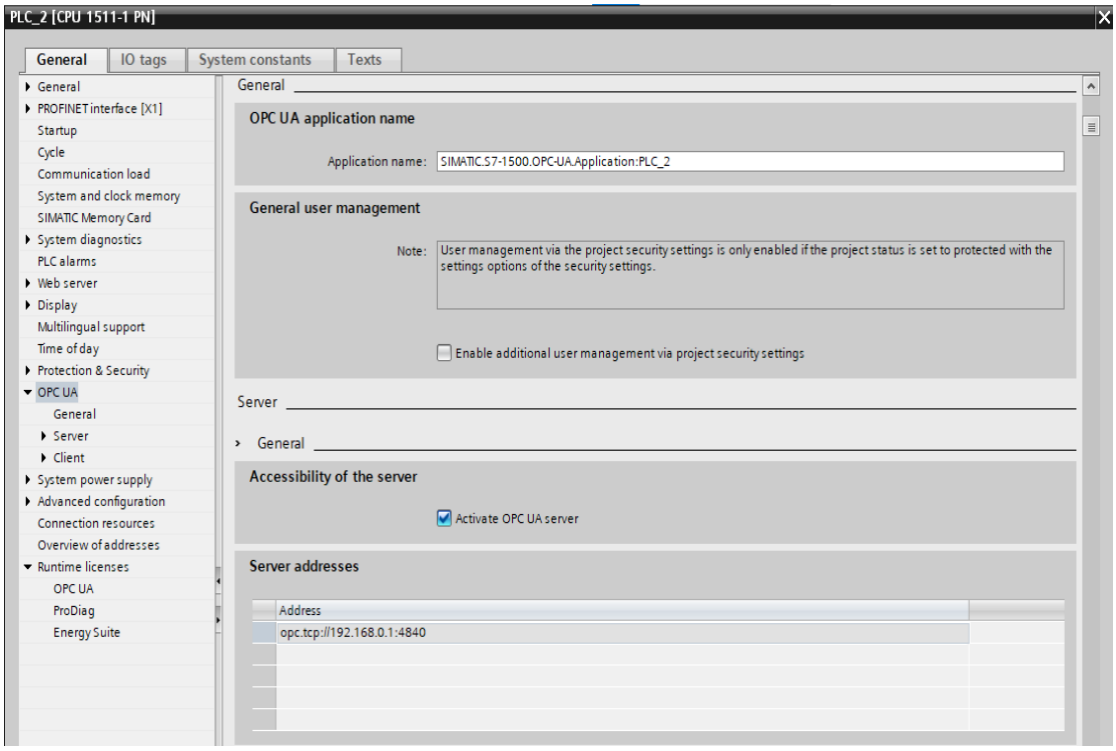

Figure 1 Activate OPC UA

3. Check the “support simulation during block compilation”, as shown in figure 2. Otherwise, the download will fail to the virtual PLC. This setting is found in the properties of the entire project.

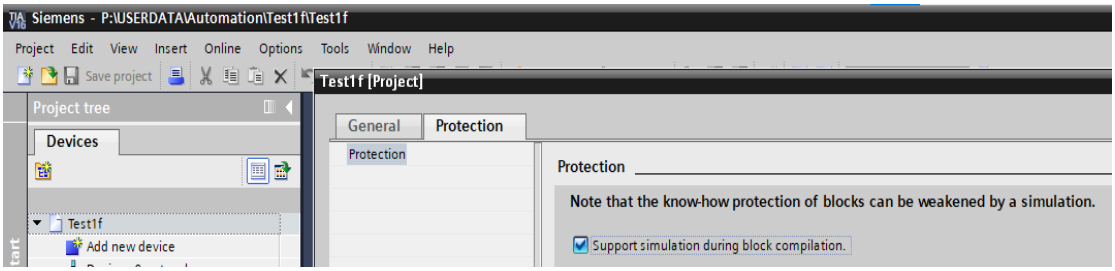

Figure 2 Block compilation

4. Whenever using an input card (digital or analogue), the Process Image has to be changed from “Automatic” to “None”. Otherwise, the state of the inputs will not update coming from the digital twin software. This setting can be found in the properties of the input card, as illustrated in figure 3.

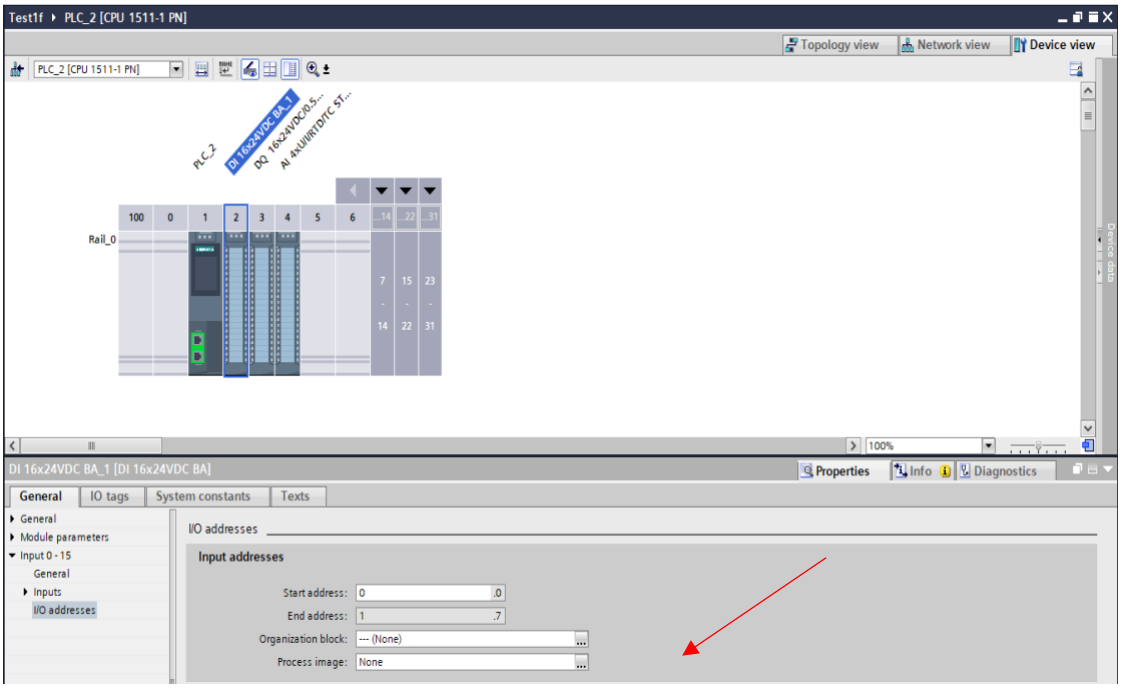

Figure 3 Process image of input cards

1.1.3 Creating a virtual ethernet adapter

A simulated version of a PLC has to be created to download the program. For a standard PLC simulation, PLCSIM is used, but to run an OPC UA server, a PLCSIM Virtual Ethernet Adapter is necessary. The program “S7-PLCSIM Advanced V3.0 Upd2” has to be installed. An ethernet adapter is automatically added to the PC's settings when this program is installed. Figure 4 shows the added adapter: “Siemens PLCSIM Virtual Ethernet Adapter”.

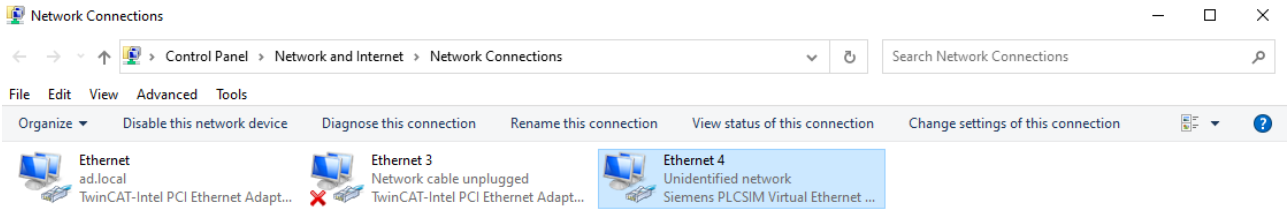

Figure 4 Ethernet adapter in windows

Right-click on the adapter, open the properties as shown in figure 5, and open the “Internet Protocol Version 4 (TCP/IPv4)” properties.

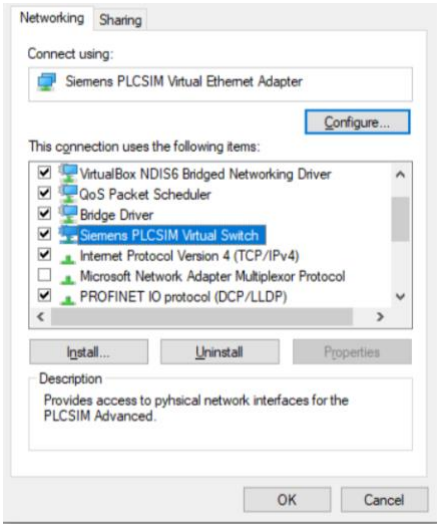

Figure 5 Ethernet adapter properties

The following window will open, as shown in figure 6. Remember that as an example, the IP address of the PLC in Tia was set on 192.168.0.1; it is now necessary to give this connection an unused IP address in the same range. For example, 192.168.0.2 or 192.168.0.10 would work; the subnet mask is also 255.255.255.0.

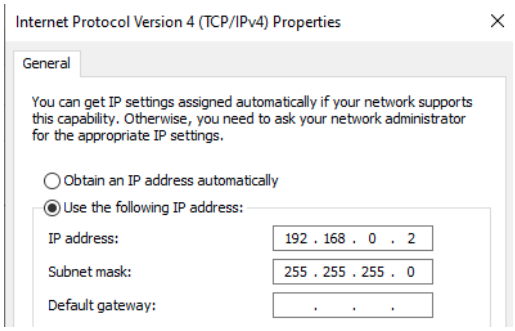

Figure 6 Properties of the TCP/IP connection

The next step is opening PLCSIM Advanced and creating an active PLC instance. Figure 7 shows the configuration window. Important to notice is that the switch above is set on a virtual ethernet adapter and not PLCSIM. The instance name can be chosen to preference, but the IP address must be the **same** as the one configured for the PLC in Tia. Once the instance is started, a yellow light will appear until the hardware and software are downloaded from Tia. Noteworthy, the next time PLCSIM advanced is started, typing in the same instance name is enough to use the same configuration.

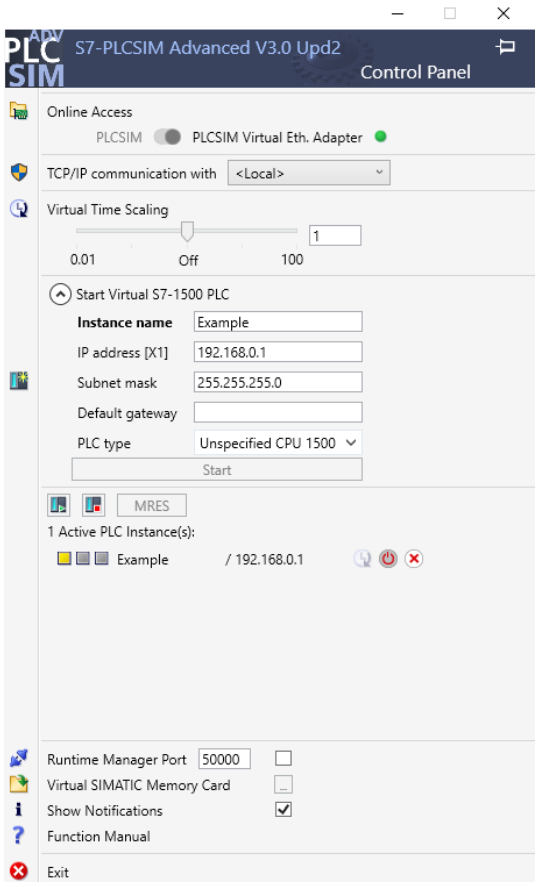

Figure 7 PLCSIM advanced control panel

The last step is now to download to the simulated PLC. Figure 8 illustrates the downloading window, and it essential is to set the PG/PC interface to Siemens PLCSIM Virtual Ethernet Adapter and search.

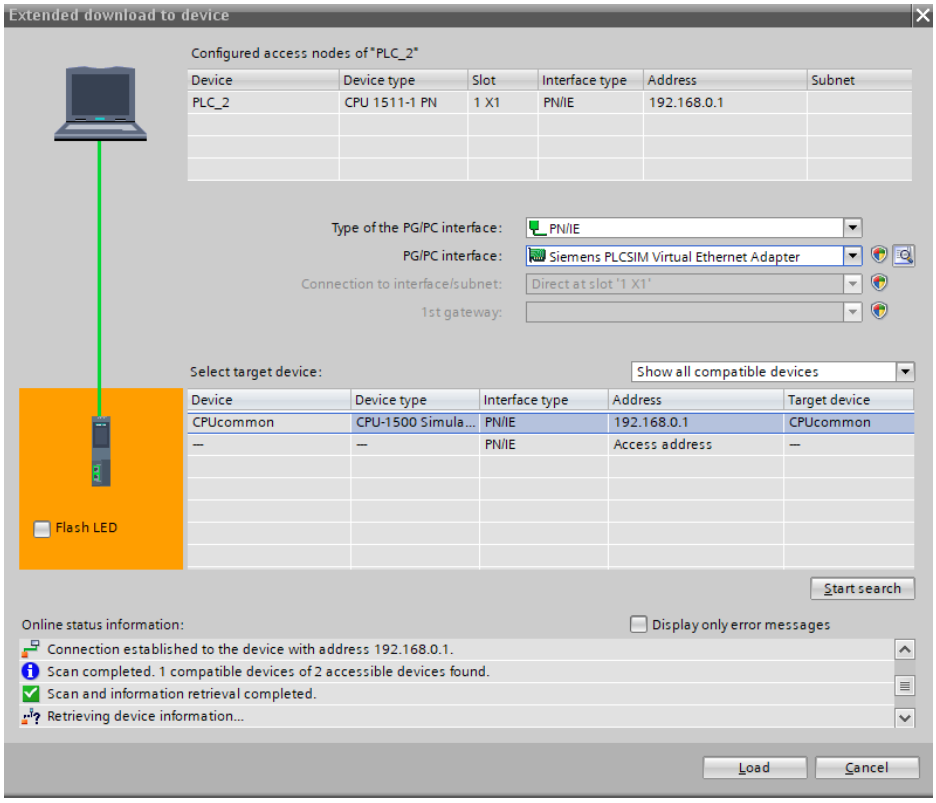

Figure 8 Downloading in Tia

Once downloaded, the light that was yellow before will be green now, the PLC is in RUN mode and the OPC UA server is active.

1.2 Visual Components (VC)

The PLC is running the software PLC, and Visual Components will be the client. The first step is to launch Visual components, create a new project, and go to the connectivity tab, as shown in figure 9. Click on OPC UA and add a new server.

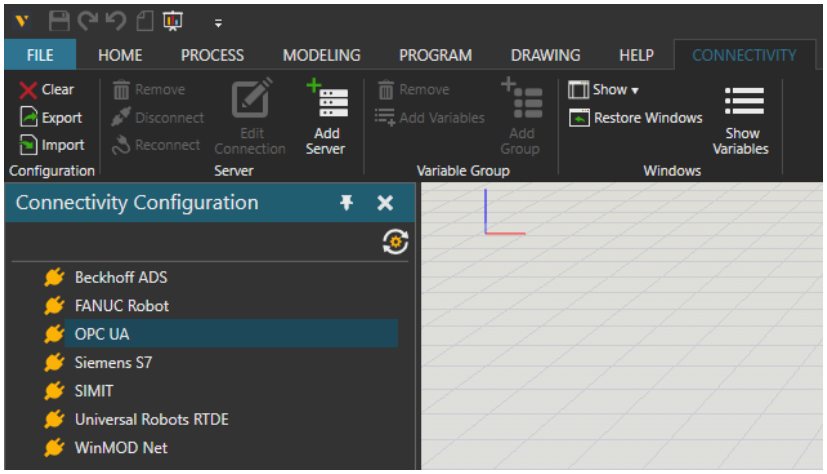

Figure 9 Adding an OPC UA server to VC

The following window will open, as shown in figure 10. Change the “localhost” to the IP address from the OPC UA server; in this case, it was 192.168.0.1 and test the connection.

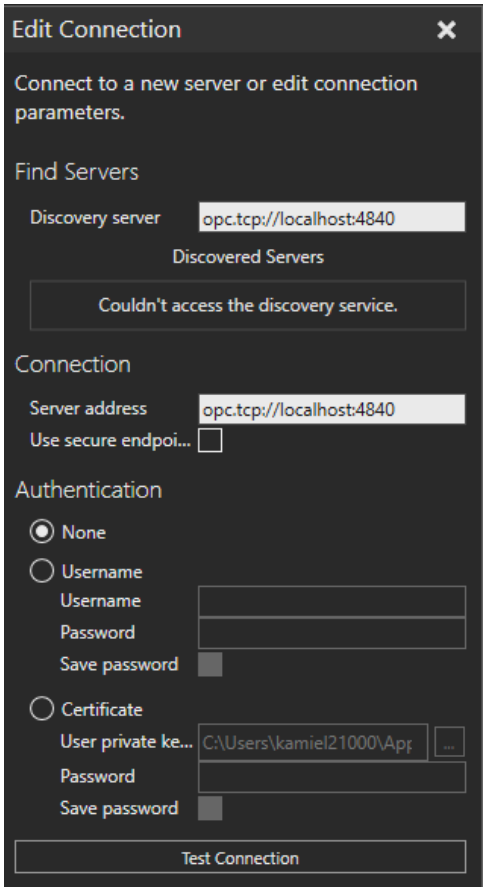

Figure 10 OPC UA properties

A window with the notification that the connection succeeded will appear, as shown in figure 11. Now press apply at the bottom of the window to create the server.

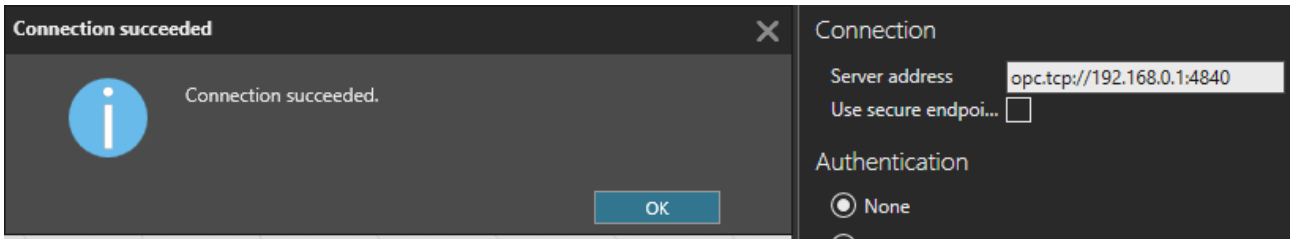

Figure 11 Testing connection

In the left menu, the server will appear, as shown in figure 12. To make the connection active, click the circle, and it will turn green. The output window will print that the connection has been made.

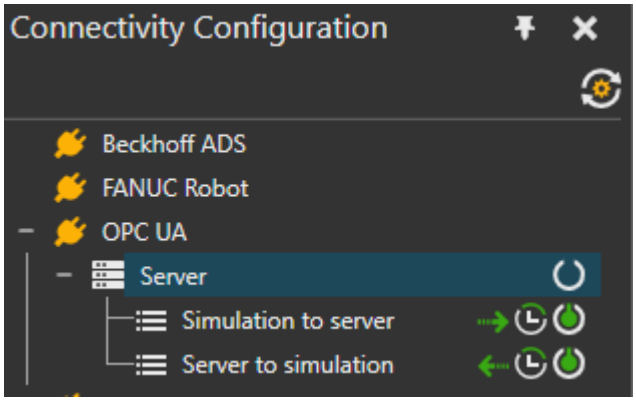

Figure 12 Enable/disable server

There are two options to connect variables. The first is “simulation to server”, which means the digital twin (VC) will send information to the PLC (inputs). The “server to simulation” is the opposite and are the outputs from the PLC that are sending signals to the digital twin.
A raycast sensor and linear actuator are used to demonstrate how to connect variables, as shown in figure 13.

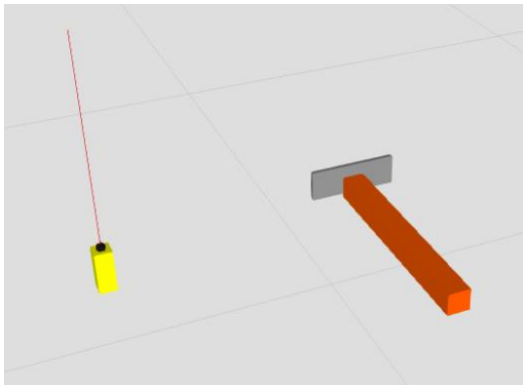

Figure 13 Sensor and actuator

Right-click on the “simulation to server” and click on add variables. This will open a window with the variables from the software on the left side and on the right side the PLCs. Right-click and press “reload simulation structure” when nothing appears, as shown in figure 14.

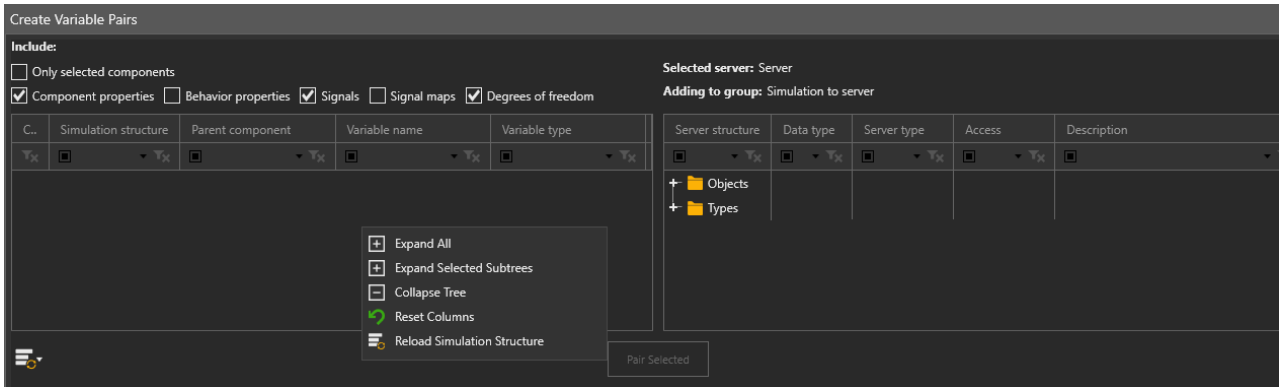

Figure 14 Adding variable pairs

“Simulation to server” is used for the inputs; in this case, this will be the sensor. Select the preferred variable to connect to the PLC’s variable. To find the PLC’s variables, open the object map as shown in figure 14 and then open the folder with the PLC's name. Now all the folders from the PLC will be visible: Inputs, memory, outputs… In these folders, only variables will appear that have been made in a PLC-tag table. For example, if in Tia a PLC-tag is made for I0.0 with the name raycast sensor, it will appear here. When a variable has been selected on both sides, click on the pair selected, and they will appear in the paired variables at the bottom of the window. The same procedure is done for the outputs, but this is done in the “server to simulation” tab.

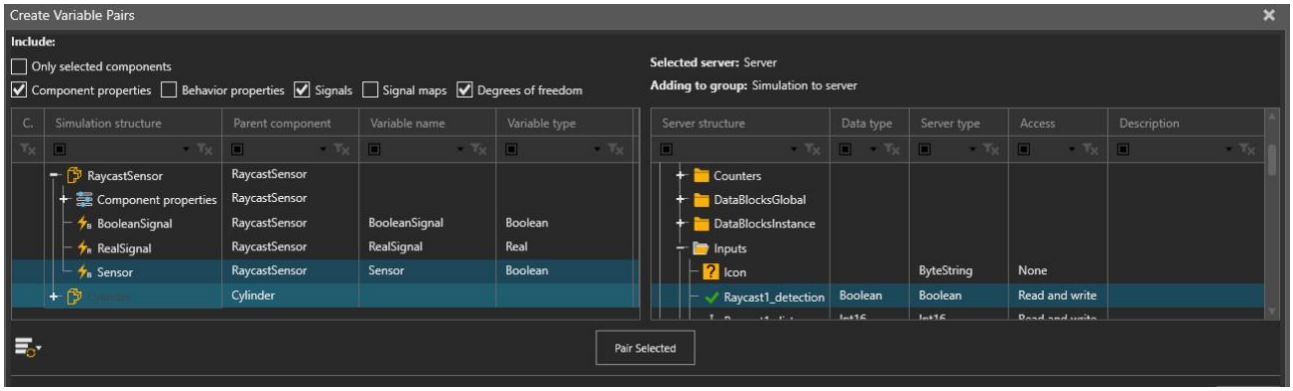

Figure 15 Pairing variables

When connecting all the variables and running the simulation, figure 16 illustrates what the software shows. Sometimes when downloading the Tia program again, the OPC UA server restarts and the green checks in the figure will appear red. The solution is to disable and enable the server by clicking on the white circle, as shown in figure 12.

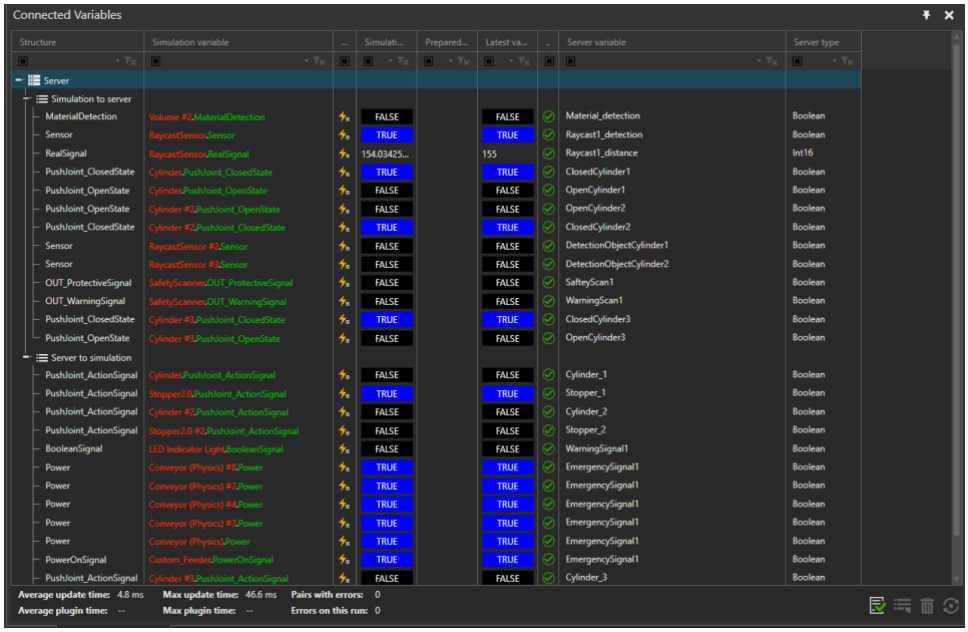

Figure 16 List of connected variables

2 Component modelling

During this chapter, different types of components will be explained step by step on how to create them. The main goal is to be able to control or use them as inputs with a PLC.

2.1 Creating a linear actuator/cylinder

The first step is always to import a 3D model or design one in visual components. Go to the modelling tab and click on “new” to create a new model. Visual components has a few basic shapes, as shown in figure 17, that can be used.

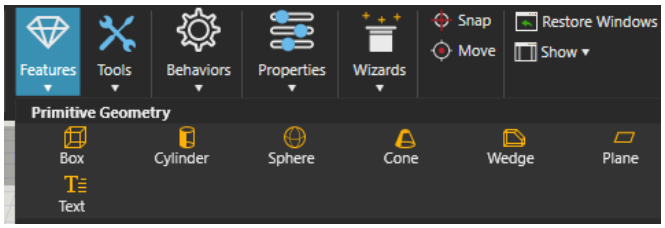

Figure 17 Basic shapes VC

For example, two boxes and a cylinder are drawn and changing their parameters gives the following shapes, as shown in figure 18. On the left of the screen, the three shapes appear under the Root.

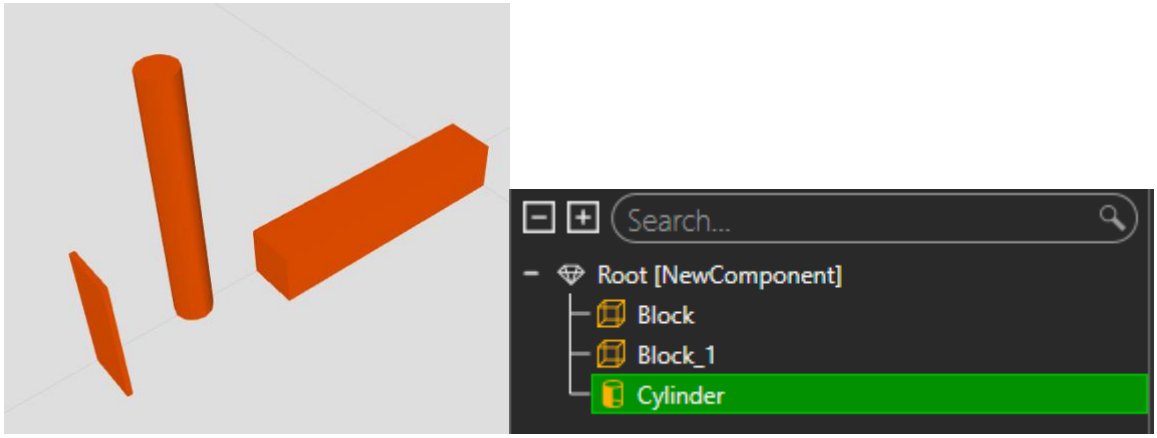

Figure 18 3 basic shapes

The next step is to make the 3D representation of a cylinder. The shapes can be moved with the move commando in the toolbar. Some helpful tools are align and snap to help with this process.

Figure 19 shows the shapes in place. The cylinder has the same length and material right now as the beam, so it might not be visible. As shown in the figure, clicking on the cylinder in the left bottom corner will reveal it. Any selected shape will automatically open the property window at the right, and the material can be changed there. This makes it easier to see the different shapes.

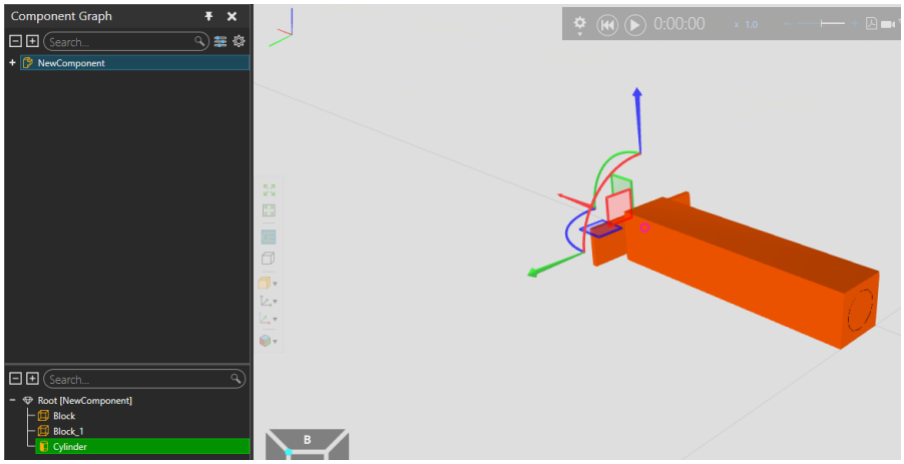

Figure 19 Assembled shapes

The next step is adding the linear movement. In the root menu in the left bottom corner, select the cylinder and the mounted plate (hold ctrl and click on both). Right-click on one of them, and a menu, as shown in figure 20, should pop up. Click on “extract link”, and it opens up a window on the right and select translational as the type of joint. The window will now have some parameters available.
Tip: if a CAD file is imported, it will appear as one shape, right-click on the root and press explode; this will separate all the shapes

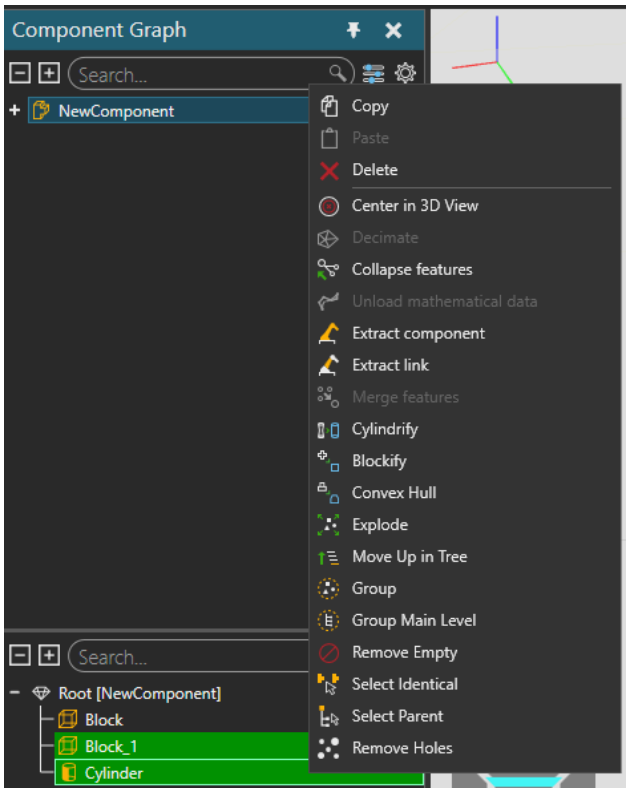

Figure 20 Menu right-click on shapes

Depending on how the shapes were orientated initially, the Axis for the translational movement will differ. In the toolbar at the upper left corner, select interact and now, when pressing on the cylinder or mounted plate, they can be dragged. They will only move via the selected axis in the menu, so it can still be changed if it is the wrong axis. The cylinder should move, as shown in figure 21.
Tip: when the link moves in the wrong direction and another axis is to be tried, first press on the double arrow (shown in the figure with the orange arrow) to go to the initial state.

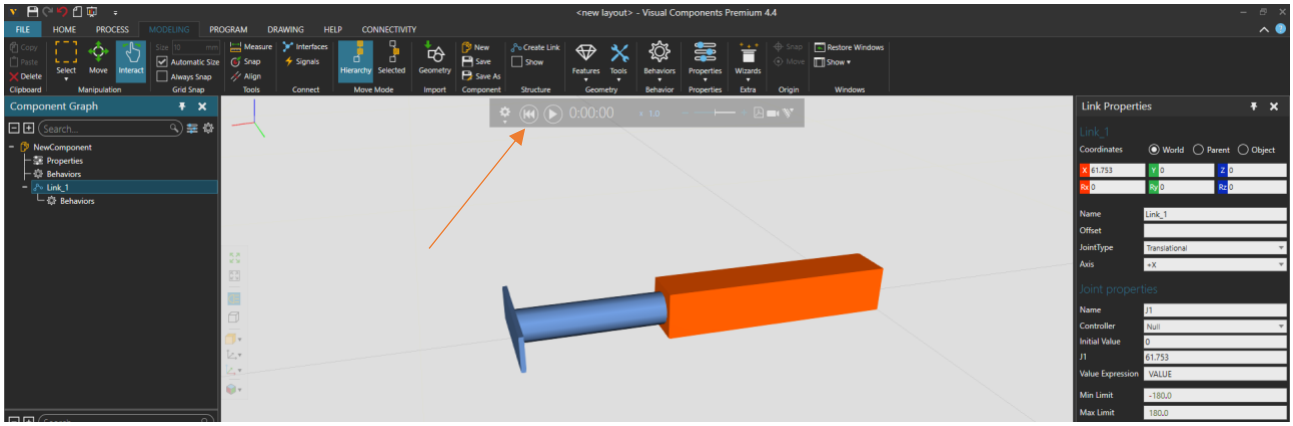

Figure 21 Axis of translational movement

Figure 22 shows the properties of the link. Change the name to “PushJoint” (this can be anything, but for the rest of this tutorial, this is used as well) and for the controller, just add a new one. The shapes used in this example are a beam and cylinder with a length of 100mm, so the min limit is 0, and the maximum is 100mm of the joint.

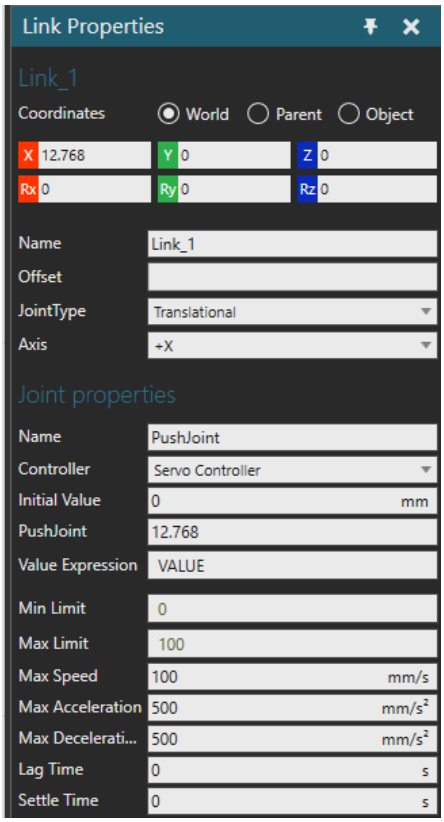

Figure 22 Properties of the link

The next step is adding behaviours and properties. Whenever in the home tab a setup is created, and a component is placed, it opens up a window with customisable parameters. The parameters that appear there are some standard options or properties added during the modelling. For example, a property could be created for how far the cylinder would make a linear movement. Instead of going to modelling and changing it manually, it could be connected to that property, making it more user-friendly.
Create the following properties as shown in figure 23. There is an option "properties" in the toolbar, and the added properties here are 4 of the type "real". Change their name and quantity (speed, acceleration, distance….) Noteworthy is that the name of the last two properties starts with the exact name as the name of the Link ("PushJoint"); this will be explained later why this is necessary.

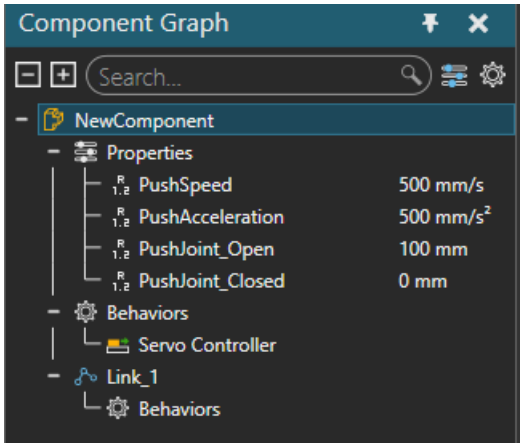

Figure 23 Adding properties

The next step is adding behaviours. Figure 24 shows the added behaviours and their names. Behaviours can be found in the top toolbar. For this model, add three booleans and 1 Python script. Important once again is the name of the booleans. The action signal is for starting the movement (connected to an output from the PLC), and the other two are end of range contacts (inputs for the PLC). Besides that, they also play a significant role in the Python script. It essential is to click on every boolean and connect them to the Python script, as shown in the figure.

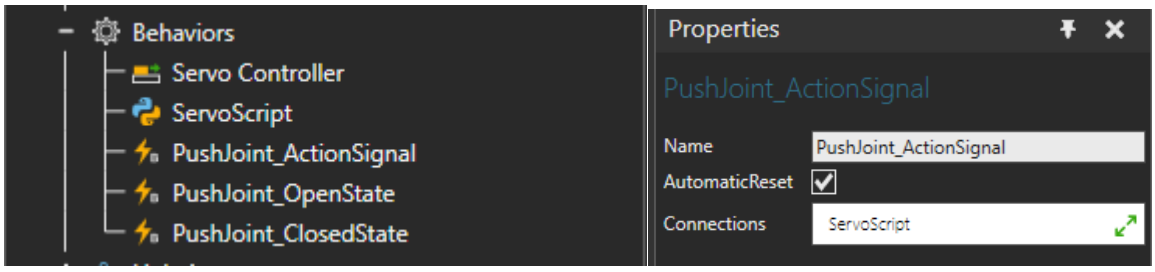

Figure 24 Adding behaviours

In the thesis above, the laboratory setup was created with everything having physical properties so they would interact with each other. To make this cylinder interact with physical objects, to push a block, for example, it needs physical properties. Click on the Link_1 in the component graph at the left and add a physics entity behaviour. Set the physics type to # Kinematic using this actuator on a conveyor belt. More information on the different types can be found in the thesis above in paragraph 4.3.6.

The last step is using the Python script to make the link move automatically. The green text explains a bit what is happening. To explain every line of code would be too complicated for this manual. Following the free Python script academy on the Visual Components website is recommended to understand more about the Python script. The “Trace execution” (orange arrow) is a great feature when the script is not doing what it should do. Enable it and then start the simulation; this will show live which line of code the Python script is running or is stuck at.

Cylinder::ServoScript

Find

```
1  from vcScript import *
2  comp = getComponent()
3  app = getApplication()
4  servo = comp.findBehaviour('Servo Controller')
5
6  servo_speed = comp.getProperty( 'PushSpeed' )
7  # this part is just to be sure there is a property for the speed and acceleration, if not it will be automatically created
8  if not servo_speed:
9     servo_speed = comp.createProperty( VC_REAL, 'PushSpeed' )
10    servo_speed.Value = 500
11 servo_speed.Quantity = app.findQuantity('Speed')
12 servo_acc = comp.getProperty( 'PushAcceleration' )
13 if not servo_acc:
14    servo_acc = comp.createProperty( VC_REAL, 'PushAcceleration' )
15    servo_acc.Value = servo_speed.Value*4
16 servo_acc.Quantity = app.findQuantity('Acceleration')
17
18 def changeSpeed(arg):
19    servo.Joints[0].MaxSpeed = servo_speed.Value
20    servo.Joints[0].MaxAcceleration = servo_speed.Value
21    servo.Joints[0].MaxDeceleration = servo_acc.Value
22
23 servo_speed.OnChanged = changeSpeed
24 servo_acc.OnChanged = changeSpeed
25
26 def OnSignal( signal ): #function gets triggered when PushJoint_ActionSignal becomes High
27    global queue
28    if '_ActionSignal' in signal.Name:
29       joint_name = signal.Name.replace('_ActionSignal','') #Pushjoint_ActionSignal becomes PushJoint
30       queue.append( [joint_name, signal.Value] )    #queue is here[PushJoint, 0 or 1]
31
32 def OnRun():
33    global queue
34    queue = []
35    joint_map = {}
36    for i,j in enumerate(servo.Joints):
37       joint_name = j.Name
38       joint_map[joint_name] = i
39       open_state_signal = comp.findBehaviour(joint_name + '_OpenState')
40       closed_state_signal = comp.findBehaviour(joint_name + '_ClosedState')
41       min_val_prop = comp.getProperty(joint_name+'_Closed')
42       max_val_prop = comp.getProperty(joint_name+'_Open')
43       # set initial values for state signals so the script knows in what position the joint/link is
44       if all([min_val_prop, max_val_prop, closed_state_signal, open_state_signal]):
45          if j.CurrentValue == min_val_prop.Value: #check if the cylinder is closed based on its current position (j.CurrentValue)
46             closed_state_signal.signal(True)
47          else:
48             closed_state_signal.signal(False)
49          if j.CurrentValue == max_val_prop.Value:
50             open_state_signal.signal(True)
51          else:
52             open_state_signal.signal(False)
54
55    while True:
56       condition(lambda: queue)
57       task = queue.pop(0)
58       joint_name, val = task
59       open_state_signal = comp.findBehaviour(joint_name + '_OpenState')
60       closed_state_signal = comp.findBehaviour(joint_name + '_ClosedState')
61       if val == False:
62          # code to close the cylinder
63          min_val = comp.getProperty(joint_name+'_Closed').Value
64          open_state_signal.signal(False)
65          joint_index = joint_map[joint_name]
66          servo.moveJoint(joint_index, min_val)
67          closed_state_signal.signal(True)
68       elif val == True:
69          # code to open the cylinder
70          max_val = comp.getProperty(joint_name+'_Open').Value
71          closed_state_signal.signal(False)
72          joint_index = joint_map[joint_name]
73          servo.moveJoint(joint_index, max_val)
74          open_state_signal.signal(True)
```

Figure 25 Python script

In the code, the joint name gets used a few times (lines 28, 29, 30, 59, 60, 63….). This is why it is essential to name everything the same in the behaviours and properties. Even the purple text in the Python code is the same, so it can have any name, but it should be the same one everywhere.

The last step is now to test if it works. Create 3 PLC tags, 1 output and 2 inputs and connect them as mentioned before in this manual. Download the PLC code and maybe reconnect the server. Run the simulation and change the output, and as shown in figure 26, the cylinder will move outwards.

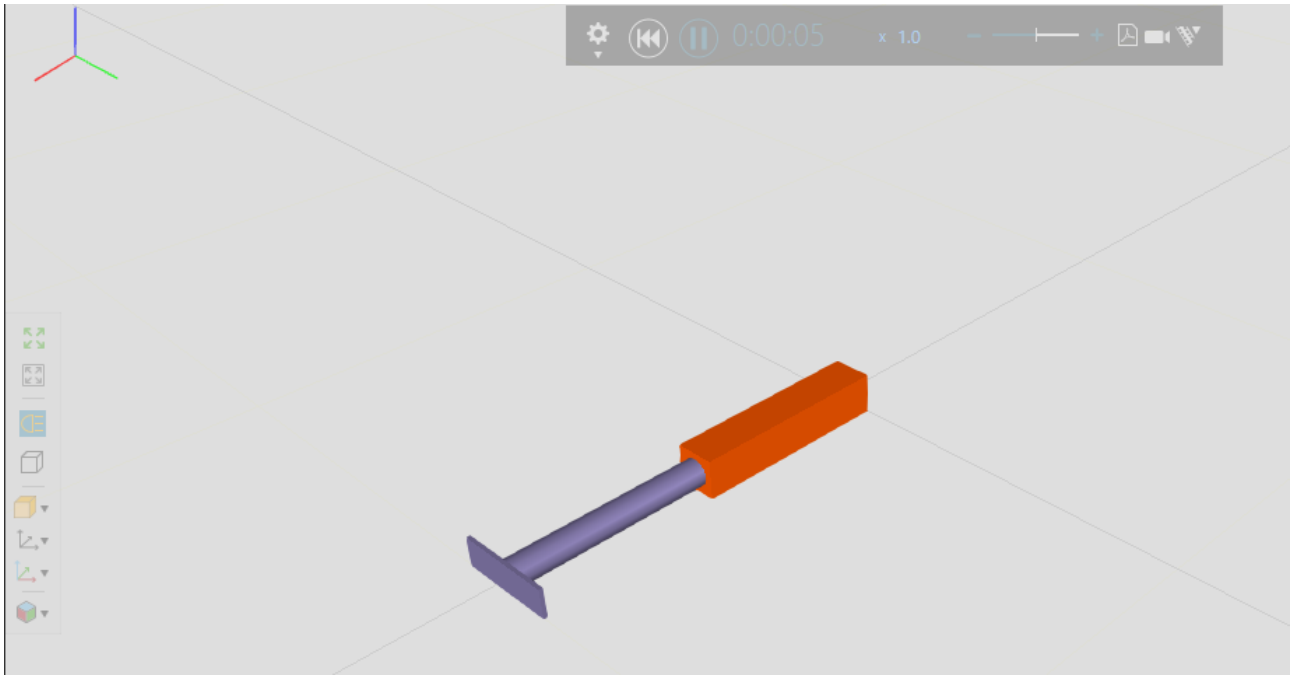

Figure 26 Running the simulation

Once this model is created, this setup can be used for any model of any size with a linear movement and needs to be controlled with the PLC.

2.2 Rotary actuator

The modelling of a rotary actuator is very similar to a linear actuator. For this reason, only the differences will be explained in this part of the manual.
Figure 27 illustrates a rotary actuator which will make a rotational movement. An example of an application can be stopping a product on a conveyor belt.

Figure 27 Rotary actuator

The same procedure is done as with the linear actuator, and in this case, the model exists out of 2 cylinders and 1 beam. The grey beam and cylinder will rotate around the fixed black cylinder. Again, select the beam and cylinder and extract a link. The only difference now is that the type is not translational but rotational. The min and max limits are now in degrees and not in mm.

The booleans and properties are identical except for the quantity of the properties. Instead of speed, it is now angular speed and acceleration, and instead of distance in mm, it is now in degrees. The Python script can remain precisely the same.

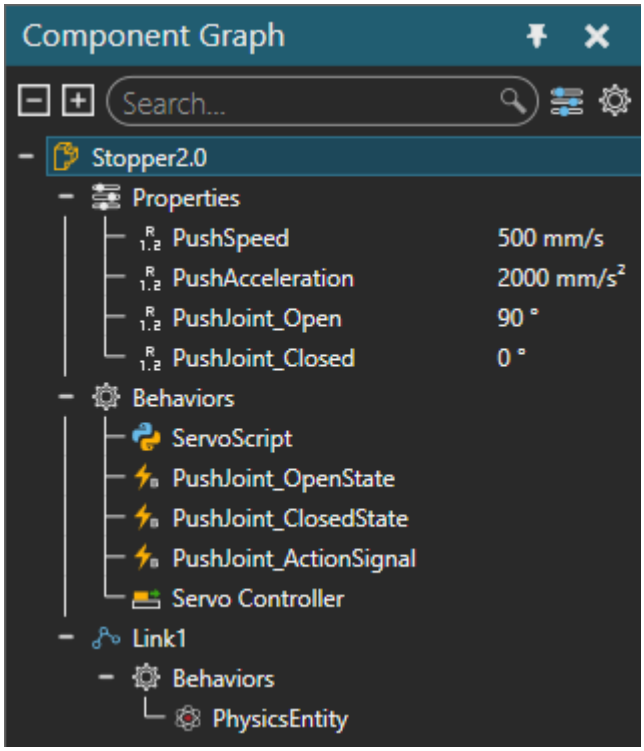

Figure 28 Component graph rotary actuator

2.3 Linear vacuum actuator

The procedure is again identical to the one from the standard linear actuator, except there is something extra added. Figure 29 shows an example of a vacuum actuator. The only difference is that a cone is added instead of a mounted plate, which resembles a vacuum suction cup. The cylinder can also move with a linear movement. To create this, repeat the exact procedure as for the standard actuator.

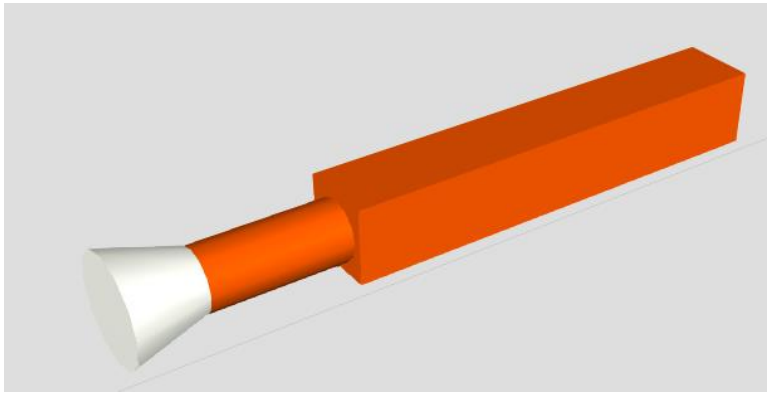

Figure 29 Vacuum actuator

The modelling tab also has some wizards available to jumpstart creating some components (end effector, workpiece positioner, conveyor…). To grab a component with this actuator, a wizard was also used to simulate a vacuum suction cup. This wizard is not available in the regular installed version of Visual Components. Download via the following website or ask the teacher for the .zip file.

Grasp action wizard: https://forum.visualcomponents.com/t/grasp-action-wizard-professional/675
Once downloaded, unzip it, put the folder in "My Commands, " and relaunch VC.

- PC at HAMK: P:\USERDATA\Visual Components\4.4\My Commands
- Own PC: C:\Users%username%\Documents\Visual Components\4.4\My Commands
-

Once relaunched, the wizard will be added. Select the vacuum actuator and click on the wizard. At the right, a window will open where a node should be selected. (choice between Link_1 or full component), choose the link and apply. This will automatically create another node in the component graph and add some properties, as shown in figure 30.
Automatically a boolean behaviour is added to control the "vacuum" (GraspSignal), which can be connected to the PLC. The last important aspect is adding a second physics entity to the grasp detection node, as shown in the figure. If not only components can be picked up when stationary and not from a conveyor belt, that is why a #Kinematic physics entity type has to be added.

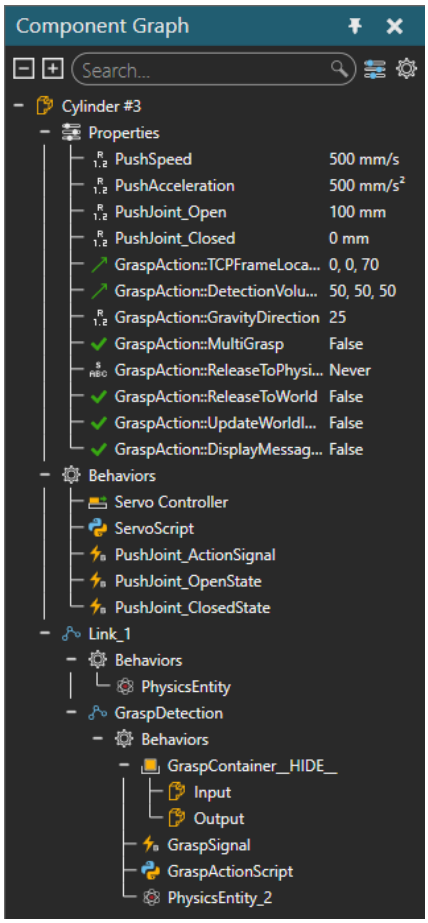

Figure 30 Component graph vacuum actuator

Raycast sensor

A raycast sensor is like an optical sensor. The first step is importing/drawing a 3D model to start from. Figure 31 is an example, but it can have any shape.

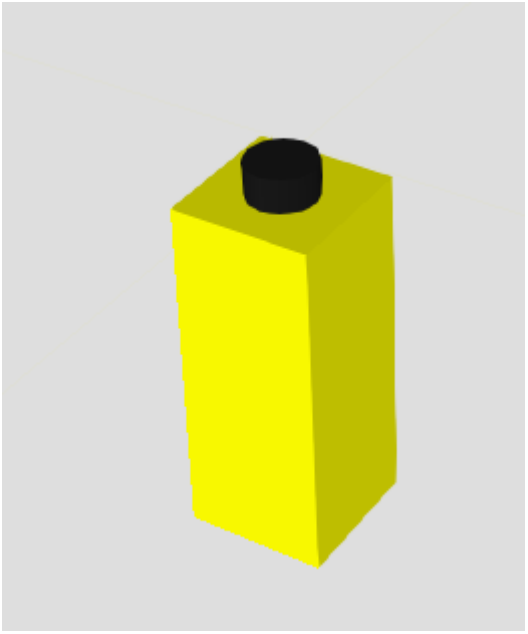

Figure 31 3D model for raycast sensor

The next step is adding behaviours and properties, as shown in figure 32. There are two new types of behaviours added here that did not appear before in this manual, the real signal, which can contain a value and the RaycastSensor behaviour. Connecting the BooleanSignal and the RealSignal to the Python script is crucial but **NOT** the Sensor one.

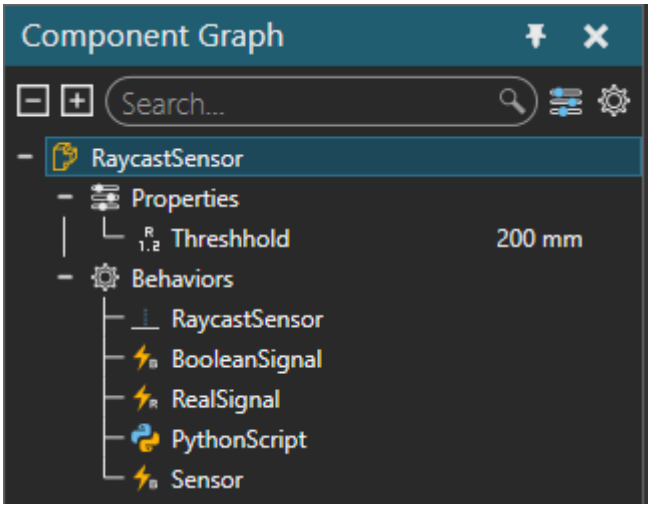

Figure 32 Component graph raycast sensor

Figure 33 shows the properties of the sensor. The behaviour can measure the distance to an object. Therefore the real signal is connected to the range signal. The sensor also detects when a component is closer than the detection threshold, and therefore the boolean signal is connected to the bool signal. The max range of the beam can be changed and the detection threshold as well. The last parameter is the frame; the sensor's beam will go in the direction of the positive Z-axis of the chosen frame. Before choosing a frame, one should be added manually. This can be done by clicking in the toolbar on features and then frame. Position the frame with the snap, align and move commandos to the preferred position on the 3D model and select the frame in the raycast parameters. When the Z-axis is not pointing upwards, just rotate the frame with the move commando.

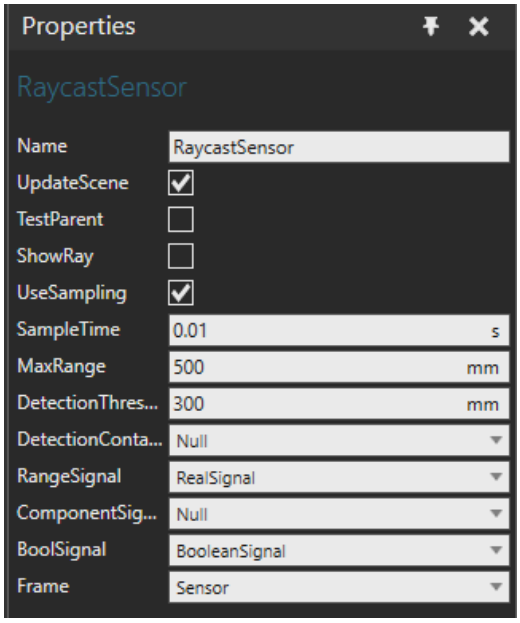

Figure 33 Properties of the raycast sensor

Figure 34 shows the result when everything is done correctly.

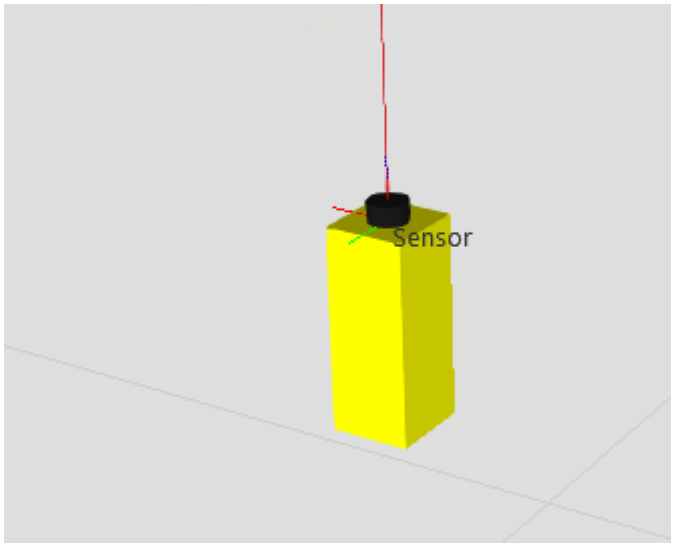

Figure 34 Result of adding the raycastsensor frame

The last step is writing the code in the Python script. Figure 35 shows the used code. The detection threshold, which was a parameter from the raycast sensor, will determine when the sensor detects an object that is close enough, and this will trigger the boolsignal. In the thesis above, the raycast sensor separates products based on size. For this reason, the property is added as well, and the real value, which is measured by the sensor, is compared to this property, and this comparison will determine if the Sensor boolean is set or reset. The sensor boolean is an input for the PLC, but that is just for this specific application of separating sizes. When detecting any object in the threshold zone, connecting the boolean signal directly to the PLC is enough.
The function “OnSignal” gets executed when a signal connected to the Python script becomes high. For this reason, the Sensor signal cannot be connected to the script because this would lead to directly resetting the signal again, and the PLC will not read an input value of 1.

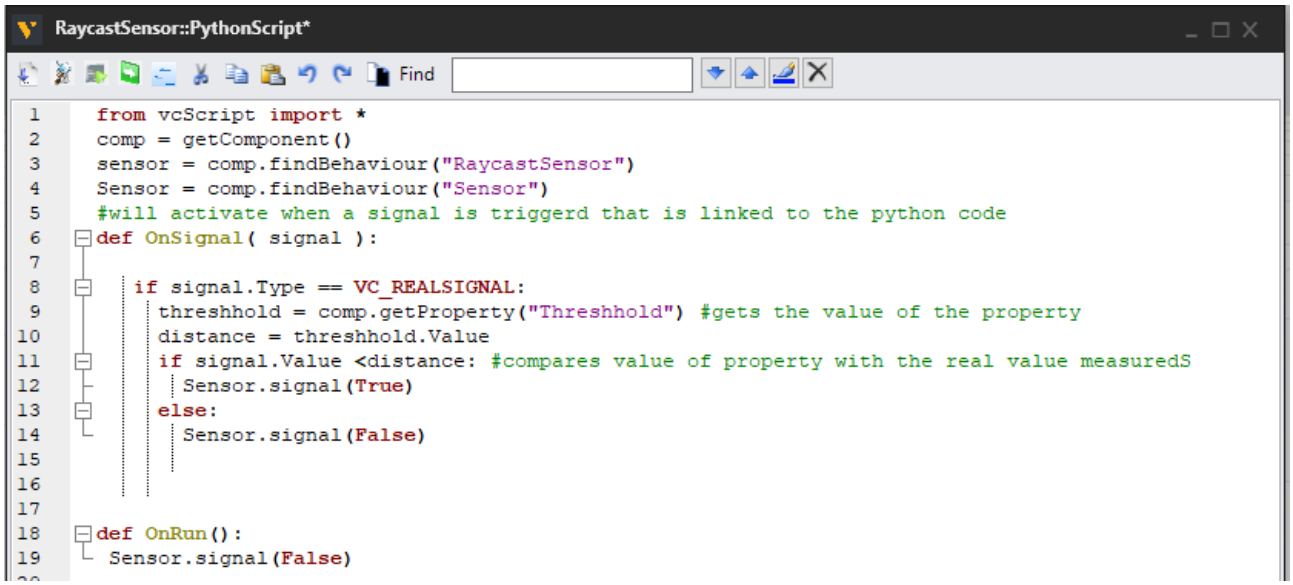

Figure 35 Python script raycast sensor

2.4 Volume sensor

The volume sensor is a digital version of a material/colour version. The same 3D model was used as the raycast sensor to keep this manual as compact as possible. Figure 36 shows the necessary behaviours and properties.

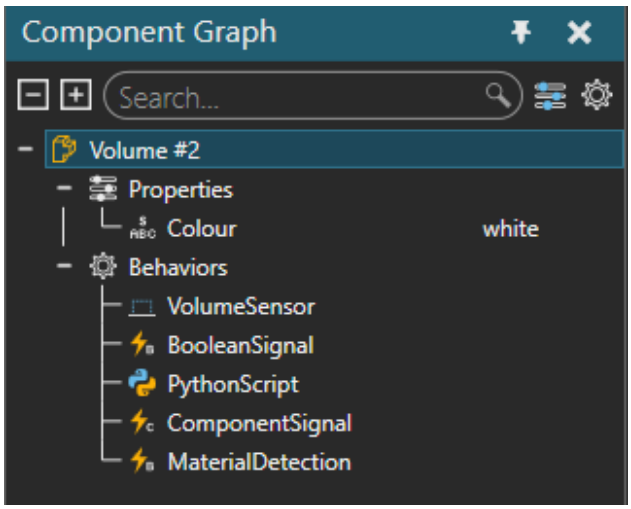

Figure 36 Component graph volume sensor

Instead of adding a real as a property, a string is added as properties. With the raycast sensor, the distance was compared to a real number. With the volume sensor, the material/colour of a product can be measured and compared with the property, and the material detection boolean is set to true, which is connected to the PLC. Again, there are two new behaviours: the component signal and the "VolumeSensor" behaviour. A real signal used for the raycast can contain a value (number), but the component signal contains information about a component/product (material, name, product ID…).
Figure 37 illustrates the properties of the volume sensor behaviour. Again the boolean signal is triggered whenever the sensor detects a component and the component signal contains the information about the detected component. The sensor measures in a contained volume, determined by the lower and upper frame. These two frames have to be added manually, as shown on the right side of the figure. The "TestMethod" determines when the sensor detects a product in the volume (entire object inside, centre inside…).

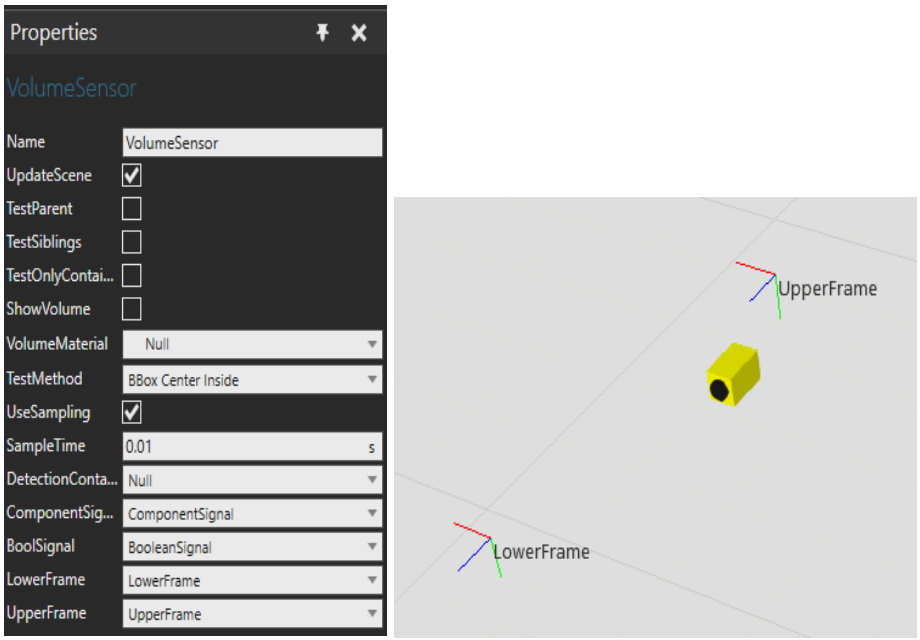

Figure 37 Properties of the volume sensor behaviour

Figure 38 illustrates the measuring volume of the sensor, contained by the upper-and lowerframe.

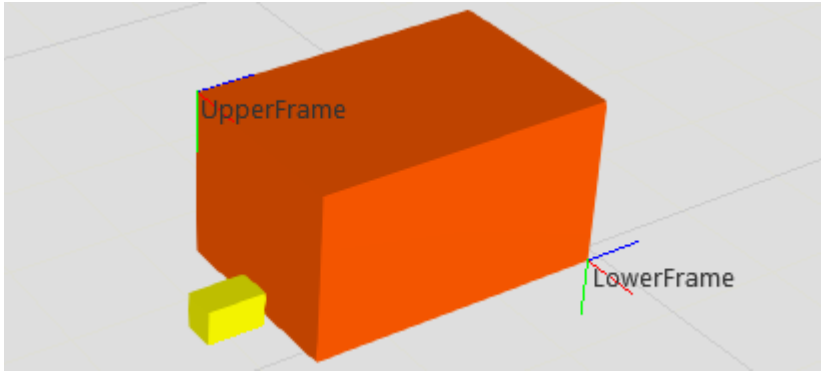

Figure 38 Volume sensor with visible volume

Figure 39 illustrates the Python code of the sensor. In the thesis above, the sensor separates products based on their material/colour. For that reason, this sensor will set the boolean "MaterialDetection" to true if the measured material is the same as the property "Colour".

Volume #2::PythonScript*

Find

```
1
2   from vcScript import *
3   comp = getComponent()
4   sensor = comp.findBehaviour("VolumeSensor")
5   Sensor = comp.findBehaviour("MaterialDetection")
6   app=getApplication()
7
8
9   def OnSignal( signal ): #triggers whenever a component is detected
10
11      if signal.Value == True:
12
13          test = sensor.ComponentSignal.Value.Material.Name #gets the material of the detected component
14          colour = comp.getProperty("Colour") # gets the property
15          value = colour.Value
16          if value == test:  #compares if the property is the same as the material measured by the sensor
17              Sensor.signal(True)
18      else:
19          Sensor.signal(False)
20
21  def OnRun():
22      Sensor.signal(False)
23      pass
```

Figure 39 Python script volume sensor

2.5 Product feeder

To run a simulation, there need to be products created in an interval. This is why a product feeder is necessary. As far as the 3D model for this, a single beam is fine. The feeder is meant to be connected to a conveyor belt, so the easiest part is creating a beam in VC so that later on, the dimensions can be easily changed to the conveyor height. Figure 40 shows what features are added in step1. It exists of 1 beam and 2 frames. Place one frame in the middle; this is where the products will spawn, and place the second one in a straight line on the edge.

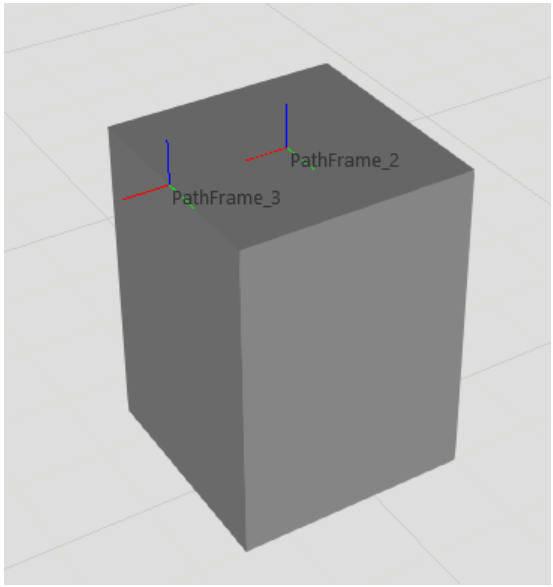

Figure 40 Features for the product feeder

Start by adding the following behaviours to the component graph, as shown in figure 41. The physics entity is again from the #Kinematic type.

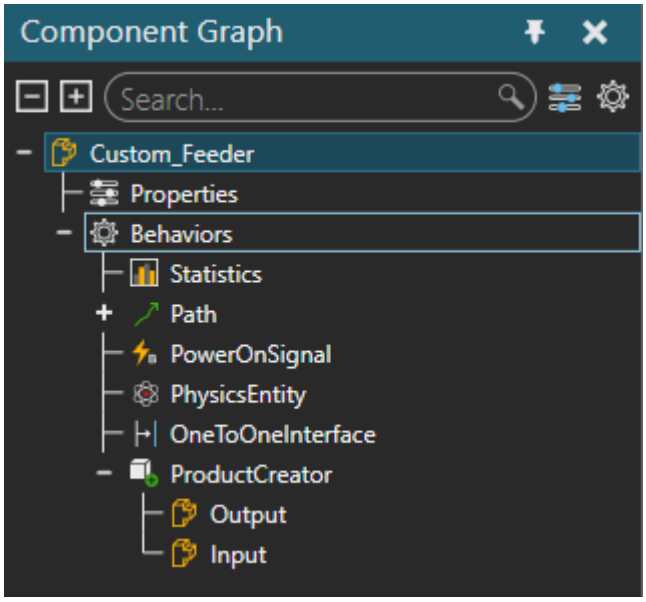

Figure 41 Component graph product feeder

A path is a new behaviour in this manual, and it is to transport the products from the middle to the edge. Figure 42 shows the parameters; the only important thing is the speed (movement speed of products) and the path. Press on the green plus and now select the frames of which the path exists. First, choose the frame from the centre (PathFrame_2 in this example) and then the frame on the edge.

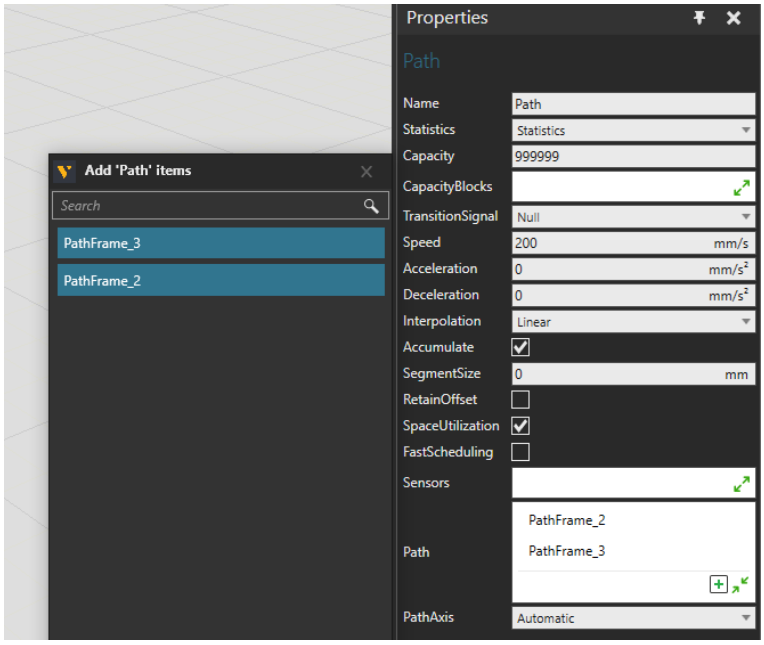

Figure 42 Parameters for a path

The “OneToOneInterface” is a new behaviour, and because of this behaviour, the product feeder can be connected to a conveyor belt. First, add a new section and select the frame's name that is located on the edge. After that, add a new field of the type “Flow”, and the container has to be the name of the path created before, and the “PortName” has to be set to output; this is because products are flowing from the feeder to a conveyor.

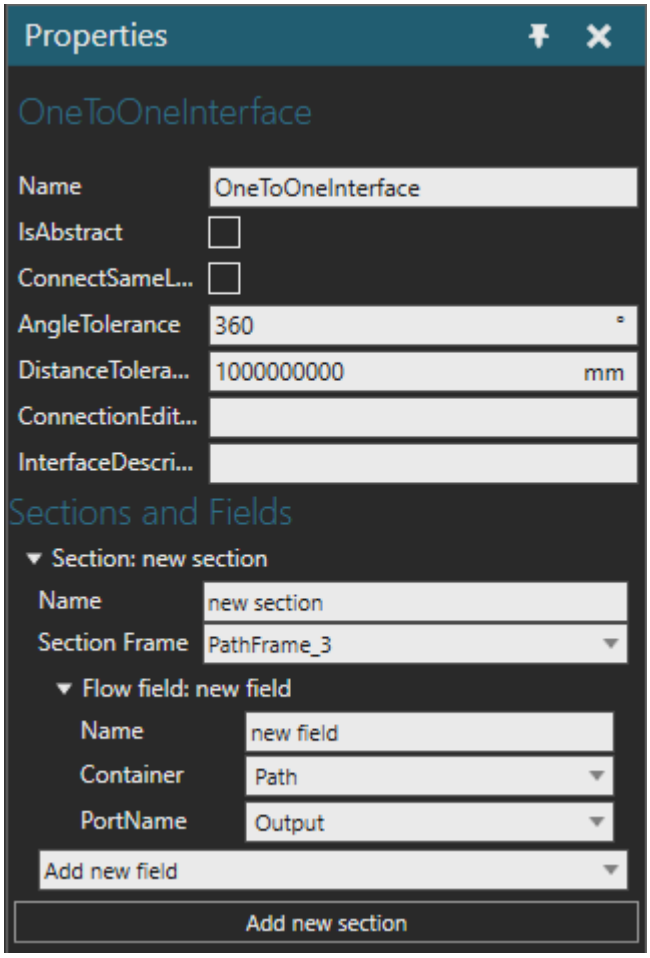

Figure 43 Parameters for the interface

The last new behaviour added was a product creator. Once added, open the output parameters as shown in figure 44 and change the connection to Path and the “Port” to Input.

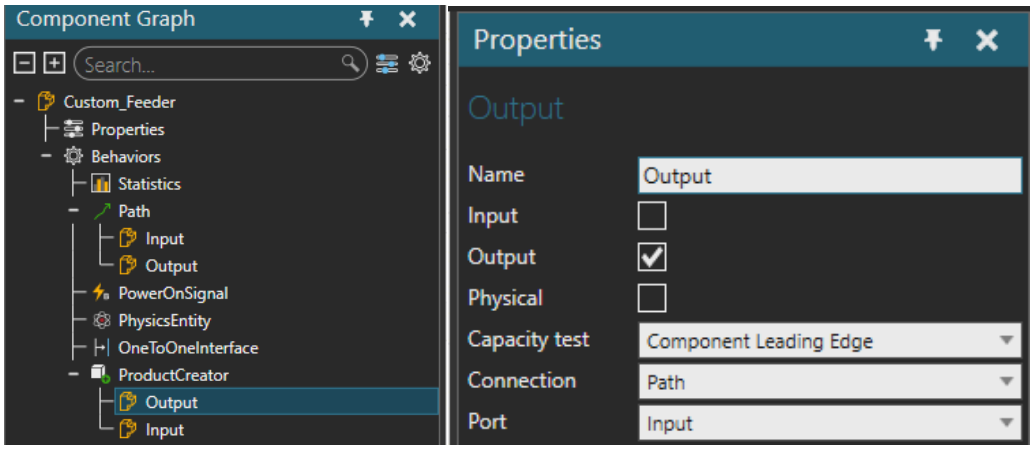

Figure 44 Product creator output parameters

When selecting the feeder in the home tab (not modelling) and selecting product creator, as shown in figure 45, there is now the option to choose which part must be created and with which interval. To create different products in a random order, read paragraph 5.3.1 in the thesis above.

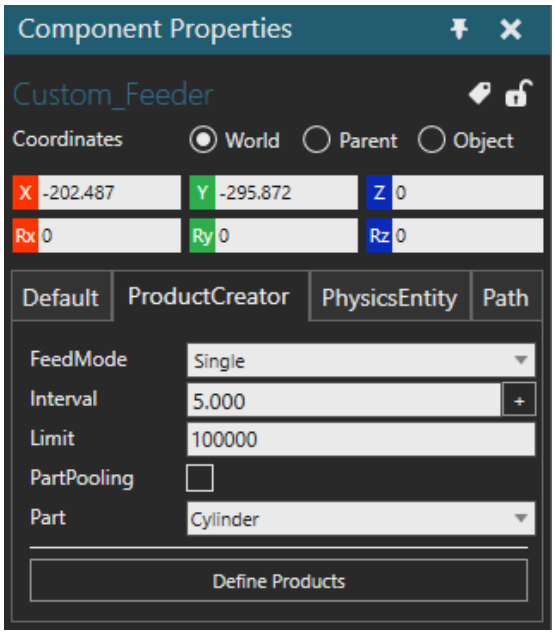

Figure 45 Configuring the feeder

3 General tips coming from experience

- When you can't find a solution, go to the help tab in the software, open the help file, or visit VC's website and search for a tutorial. The last resort is using the forum, it has a great community, and from experience, a solution is presented relatively quick.
- Save a lot when working on a model, project… the program crashed multiple times and has **NO** autosave.
- In the home tab, use the PNP tool in the toolbar to connect components with interfaces.
- Save your own created models and save them in a folder. Add this folder as a source in the eCatalog to quickly drag the components in the layout.
- Running a simulation where a PLC program is involved and where there are timers used in the PLC program does not speed up the simulation. Speed it up to fast, and the communication cannot follow, and the timers in the PLC program still have the same time, so they will run behind.
- When running the simulation and stopping halfway and resetting the simulation can sometimes cause actuators to move when starting the simulation again because the outputs were still high in the PLC code