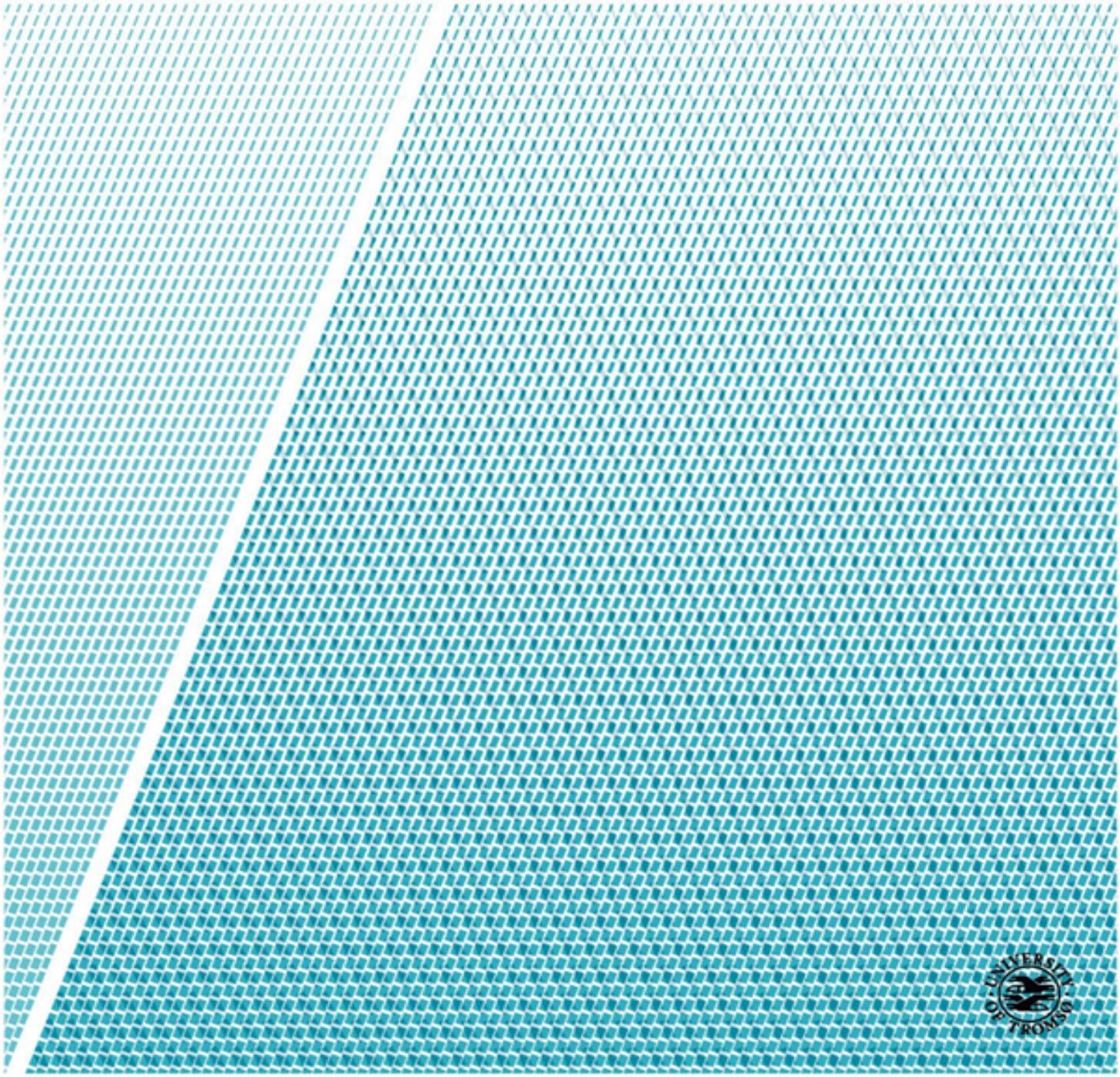
UiT
THE ARCTIC
UNIVERSITY
OF NORWAY

Faculty of Engineering Science and Technology (IVT)
Department of Industrial Engineering

Digital twin simulation with Visual Components

A study based on Digital twin simulation

Stud. Techn. Halldor Arnarson



Abstract

A digital twin is a new and emerging tool of the fourth industrial revolution. It is an extensive part of connecting the manufacturing equipment together to develop a smart factory. Investigation of simulating manufacturing system to bind the digital world with the physical, allows for analysis of data and monitoring of systems to head off problems before they even occur, preventing downtime, develop new opportunities and even plan for the future by using simulations. In this study, a KUKA KR 30-3 Robot was used as a main vessel of motors to mirror the programmed movement initiated in the visualizing software Visual Components. The investigation of the connection between the robot and simulation was therefore conducted with an old version of the robot controller [KR C2], where an OPC UA server was utilized to share the data throughout the work. Additional to the existing digital twin an AVG was planned to interfere conjointly with the system. In purpose to illustrate the capability of the OPC UA to store information for a common system. Findings of the research-study resulted in a stable digital twin of the KUKA robot using RSI communication. Using the OPC UA server provides the possibility to connect future manufacturing equipment to the digital twin.

Keywords: Digital twin, OPC UA, KUKA, AGV, Visual Components

Acknowledgement

I would like to thank my supervisors Wei Deng Solvang and Gabor Sziebig for giving me the topic of my master thesis and thank Wei Deng Solvang for helping me expand the topic of my master thesis's.

I would also like to thank Gabor Sziebig for helping me developing the digital twin and finding good ways to control the robot.

The last part of the project was done at BIT in Beijing, China. I would like to thank Hao Yu for handling the exchange trip and supporting us during the master thesis. I also want to thank my fellow students who went on exchange with me, Mathias Sæterbø and Hans Ivar Arumairase for supporting and motivating me during the exchange trip. From BIT i want to thank our exchange coordinator Summer in Summer in BIT helping to organizing our stay in Beijing China.

Finally, I want to thank Martin Adresen and Lazar Sibul for helping me preform tests over the internet on the KUKA controller i Narvik while i was on exchange at BIT.

Table of Contents

List of Figures

List of Tables

Terms and Abbreviations

OPC - Open Platform Communications

OPC UA - OPC Unified Architecture

PLM - Product Lifecycle Management

EC - Energy Consumption

PHM - Prognostics and Health Management

RAMI 4.0 - Reference Architecture Model for Industrie 4.0

IoT - Internet of Things

API - Application Programming Interface

XML - eXtensible Markup Language

SOAP - Simple Object Access Protocol

KRL - KUKA Robot Language

TCP/IP - Transmission Control Protocol/Internet Protocol

CSV - Comma Separated Value

PHM - Prognostic and Health Management

SPS - Submit interrupter

RSI - Robot Sensor Interface

AVG - Automated/Automatic Guided Vehicle

HIRIT - Human Idustrial Robot Interaction Tool

FEM - Finite Element Model

1 Introduction

We are living in a world where everything becomes more and more connected. Smart homes where you can control everything with your smart phone, is becoming increasingly popular. It is for that reason inevitable that we are going to get the smart. The smart factory is a part of the fourth digital revolution, where you have concepts like Industry 4.0, internet of things(IoT) and digital twin.

A digital twin is a virtual software representation of a physical object or system [1]. This is done through collecting sensor data to be able to recreate the system. It can also make old and outdated equipment smart and connected through the implementation of a digital twin. The digital twin can be used in many different areas. Examples of using a digital twin is predicting when a system will fail, monitoring and improving on the current system. Another field it can be used in is human robot collaboration. The digital twin acts like a fence to allow for more collaboration between humans and robots.

One of Norway's biggest companies Equinor (formerly called Statoil), are working on accelerating the digitising efforts in the company. The oil field Johan Sverdrup accounts for between 20 to 25 percent of Equinors total offshore production and the company mentioned "Thanks to the digital twin Statoil will also have full control of things happening at Johan Sverdrup at all times, and the analytics tools make it easy to generate data on a consecutive basis." [2]. There has also been a growing interest in digital twin from other big companies like General Electric, Siemens, ABB and IBM.

1.1 Background

The digital twin has been growing in popularity for the last years. From Gartner [3], the company states that digital twin will be in the forefront as a disruptive trend that will have significant impact in the five years to come. *Gartner predicts that by 2021, half of the large industrial companies will use digital twins, resulting in those organizations gaining a 10% improvement in effectiveness* [3].

The research vice president at Gartner says *"Digital twins drive the business impact of the Internet of Things (IoT) by offering a powerful way to monitor and control assets and processes"*. It is an important part that connects the manufacturing equipment and gets an overview of how the system.

A previous master thesis from NTNU created a digital twin of a robot cell at the department of MTP Valgrinda. There are four KUKA robots in the cell that are connected to a KR C4 robot controller. However, in the machine laboratory in Narvik, there is a KUKA robot connected to an older robot controller the KR C2. A digital twin hasn't been created for this robot controller, and therefore, the main objective will be to identify if it is possible to create a digital twin for this system.

1.2 Thesis objectives

The first part of the master thesis is to conduct a literature review of the digital twin. This includes finding out what digital twin is, where it is used, and how far we have come with the development of digital twin. The first part does also includes finding out how to communicate with the KUKA controller in the laboratory and how to connect Visual Components to hardware.

- Conduct a literature review on digital twin methods and techniques.
- Do a review on how to communicate with the KUKA controller (KR C2) and how to control the robot.
- Find a method to connect Visual Components with the KUKA controller.

The second part is the largest part of the master thesis and accounts for two-thirds of the master thesis. This part contains creating the digital twin with the KUKA robot using Visual Components and the OPC UA server. This section will find out if an OPC UA server is a good solution to create a digital twin and how the robot can be controlled with Visual Components. The scope of the main part is listed below:

- Create a digital twin of the KUKA robot in the laboratory at UiT using Visual Components for simulation and OPC UA server to establish communication. OPC UA will be used as middleware between Visual Components and the computer connected to the KUKA controller.
- See if it is possible to create a digital twin with an older KUKA controller (KR C2), then has been done before (NTNU digital twin with KR C4).
- See if OPC UA is an excellent solution to connect the simulation model with the KUKA robot.
- Find ways to control the robot from Visual Components.

2 Literature review

This chapter goes through relevant content when it comes to digital twin and different possibilities of connecting the equipment at UIT Narvik. In addition, look for similar work done with Visual Components and digital twin.

2.1 Industrial Revolutions

The manufacturing industry has gone through big changes since the 18th century. These can be divided into four stages, as shown in figure 1.

First industrial revolution occurred towards the end of the 18th century, which introduced water and steam powered mechanical manufacturing. In the second revolution the world was introduced to conveyor belts and mass production at the start of the 20th century. The third revolution employed electronics and information technology to increase the automation of manufacturing. Machines took over manual labour but also some of the "brainwork" [4][5]. Today the manufacturing industry is going through a transformation with the rise of the "*autonomous robots, contemporary automation, cyber-physical systems, the internet of things, the internet of services, and so on.*" [5]. These new changes and developments are called industry 4.0.

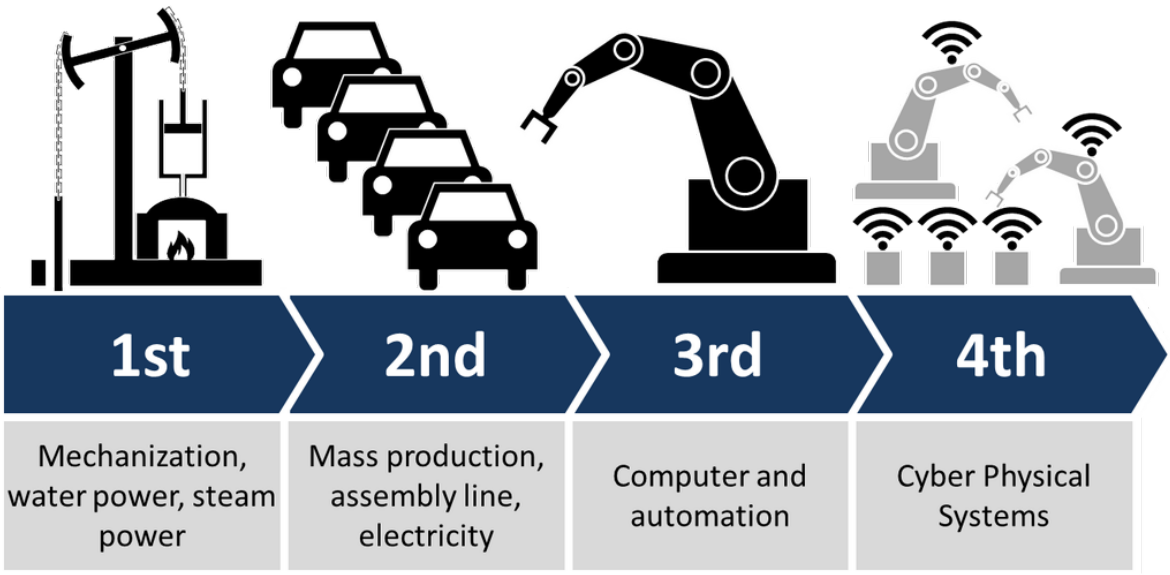


Figure 1: The figure show each industrial revolution from industry 1.0 to 4.0 and the figure is taken from [6].

2.2 Digital Twin

The concept behind "twins" can be found in NASA's Apollo program. NASA made two identical space vehicles to be able to mirror the condition of the mission on the space vehicle. The vehicle that remained on earth was called the twin. The twin was also used in the training

and preparation. In the mission, the twin was used to assist the astronauts in orbit and critical situations [7].

Simulation for technology has been used mainly by engineers in the last decades to answer specific design and engineering questions. It has been limited to computer and numeric experts. Present day simulation is used as a tool for design decisions, validate and test for both components and complete systems. Today you can also see ”communication by simulation,” which is the core concept of model-based engineering. Digital twin approach is the next wave in modell simulation and optimization of technology [8], as shown in figure 2.

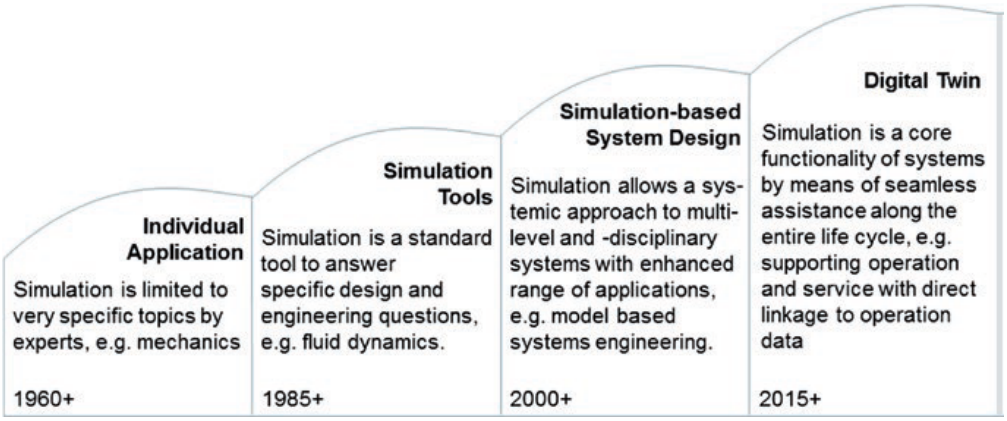


Figure 2: The figure illustrates the development of simulation [6].

Digital twin was first introduced in 2003 by professor Michael Grieves in Product Lifecycle Management (PLM) classes [9]. It can be described as a ”*digital mirror of the physical world and maps the performance of physical world.*” [1].
The digital twin can be divided in to three main parts [9]:

- A physical product in a real space
- virtual model in a virtual space
- Connections of data information from the physical product to the virtual model and back

In decade after the model was introduced, there has been a tremendous increase in the data collected from physical models. This has allowed us to not only visualize the product but also be able to test for performance capabilities [9].

The purpose of a digital twin is to build effective communication between the physical world and the information world. It means using a large amount of data collected from the physical world in an sufficient manner [1].

2.2.1 Defining digital twin

It is important to note that digital twin hasn’t been clearly defined. In some cases, it refers to a virtual representation of a physical object or a system that captures all the manufacturing

defect's, and that can catch the wear and tear while an object is in use. In other cases, it referees to the collection of digital and historical profile. The data collected from sensors is used to help engineers understand how the products are used in real time by the customers and how the product will perform in different circumstances and future scenarios [10].

The digital twin has been growing in popularity the last years, and large corporations like General Electric, Siemens, ABB, Equinor and IBM have taken an interest in the subject. Their definition of what digital twin is listed below.

- **General Electric** - *Digital twins are software representations of assets and processes that are used to understand, predict, and optimize performance in order to achieve improved business outcomes. Digital twins consist of three components: a data model, a set of analytics or algorithms, and knowledge.* [11]
- **Siemens** - *A digital twin is a virtual representation of a physical product or process, used to understand and predict the physical counterpart's performance characteristics. Digital twins are used throughout the product lifecycle to simulate, predict, and optimize the product and production system before investing in physical prototypes and assets.* [12]
- **ABB** - *A digital twin is a complete and operational virtual representation of an asset, subsystem or system, combining digital aspects of how the equipment is built (PLM data, design models, manufacturing data) with real-time aspects of how it is operated and maintained. The capability to refer to data stored in different places from one common digital twin directory enables simulation, diagnostics, prediction and other advanced use cases.* [13]
- **IBM** - *A digital twin is a virtual representation of a physical object or system across its lifecycle, using real-time data to enable understanding, learning and reasoning*. [14]

From the definition above a digital twin is a virtual software representation of a physical object or system. It is used to understand the system, improving the system, and being able to make predictions.

2.2.2 Benefits of digital twin

The digital twin is an important concept in manufacturing and industrial internet because it uses big data to find a recommendation that can be tested in the digital twin simulation before being put into production [15].

Mentioned in chapter 2.1 since we are in the fourth industrial revolution, the benefits of digital twin aren't 100% clear. A master thesis that looked at what potential benefits could be gained from introducing digital twin noted that "*it is not clearly defined what the fourth industrial revolution is, as this revolution is a prediction done apriori. It is therefore not clear exactly what value a DTw will have in the revolutionized industry.*" [16]. It is easier to define the benefits and methods that will revolutionize the manufacturing industry after the revolution.

The benefit from implementing a digital twin at a robot cell in MTP Valgrinda was [16]:

- **Visibility:** The digital twin can improve visibility and can be beneficial in carrying out multi-robot planning and control.
- **Statistical and predictive analysis:** When using Visual Components, predictions and scenarios can be tested for different states. Sensor data from the physical components can be used for analysis to optimize characteristic parameters in a robot program. This data could also be used for error detection and used to predict the need for maintenance.
- **Energy Consumption (EC):** The advantage of having a simulation model is that programs can be tested in the simulation model, without ruining the robot movements. When new robot programs are being developed a lot of testing will be done. Running the robots and the controllers require a lot of energy, and it would, therefore, save a lot of energy to run the robot in the simulation when testing. The simulation can also be used to optimize the robot program and make a more efficient and energy saving program.
- **What if analysis:** Having a digital twin can be used to test the limits of the robot.
- **Documentation and communication mechanism to understand and explain behavior:** Using a digital twin will give a better foundation for organizing, documenting, and explaining the behavior in the robot cell. Comparing this data to historical data could be beneficial for optimizing the robots.

2.2.3 Uses of digital twin

The application of digital twin is currently primarily applied in the field of aeronautics and astronautics for failure predictions. It is mainly used in product service and maintenance phases [17].

As mentioned in chapter 2.2.2, a digital twin can be used to get visibility over a system, and the data collected can be analyzed to improve the system. To be able to manage the increased complexity of larger and larger automated production networks in the manufacturing sector, a digital twin has been proposed as a solution [9][10]. Two significant areas where digital twin can be applied is in the human-robot-collaboration and for autonomous systems.

Digital twin with human-robot-collaboration The robots that are operating in manufacturing's systems today are tightly controlled in isolated environments. The next step in robotics is to move the robots from isolated environments and enable them to cooperate with humans. These systems require substantial engineering effort with careful planning to make efficient utilization of robot recourses. The problem today is that when we remove the robot from a controlled environment, the robots have difficulty performing at their full potential [18].

Another problem is the fluctuating order quantity. One solution to handle this problem is to use a flexible production system with the use of human-robot-collaboration. There are risks with humans working to close to robots, such as pushing, crushing, or clamping during operation. To reduce these risk standards and safety regulations have been created to protect

humans and robots from injury or damage. These regulations lead to significant production losses, and because of the law, human-robot-collaboration cannot be implemented [19].

State of human-robot-collaboration When human-robot-collaboration is applied, the robots are usually surrounded by a fence. In many systems to improve accessibility, optical sensors are used instead of fences. There are also systems that are camera-based or have light sensors [19] as shown in figure 3. Similarly, laser scanners can be used for plane monitoring. However, the problem is that monitoring with high enough resolution and high sampling rate in the range of milliseconds is currently a challenge today.

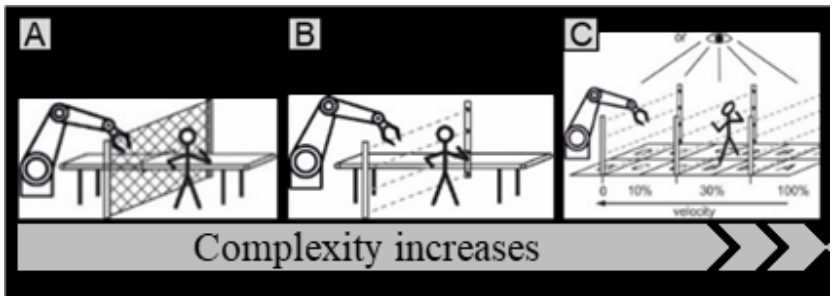


Figure 3: The picture shows the complexity of human robot system where A is with a fence, B uses an optical surveillance and C combines different methods of surveillance [19].

In a human-robot-collaboration system, the robot should be able to react or change its path if there is a human in the way. It should also use online planning to find the most optimal route. A system that uses multi-sensor-based path planning with optical and tactile measurements, that also has intuitive, natural communication would create and allow for a barrier-free human-robot-collaboration system [19].

A solution to solve the problem with human-robot-collaboration is to use a digital twin. The digital twin can use the sensor of the manufacturing system to recreate the system in a virtual space. This will allow monitoring of the robot and allow the human and feces to be avoided, which leads to fewer limitations when it comes to interactions between the robot and human.

The digital twin in human-robot-collaboration is composed of interconnected environments, in other words, the physical and virtual space. The physical environment is the production system containing humans, robots, and the equipment to the production system. The virtual environment, on the other hand, includes the computer simulation of the production system.

When a digital twin is created of a system, it is recommended that the virtual system is created as early as possible in the idea generation phase to be able to create the most optimal design of the system. While the digital twin is created, it is essential that it is continuously updated with the evolutionary changes and modifications of the physical system. The digital twin will then develop during the ideation in its information content and connectedness with the physical system [20]. A digital twin with a human-robot-collaboration system which enables efficient design is shown in figure 4.

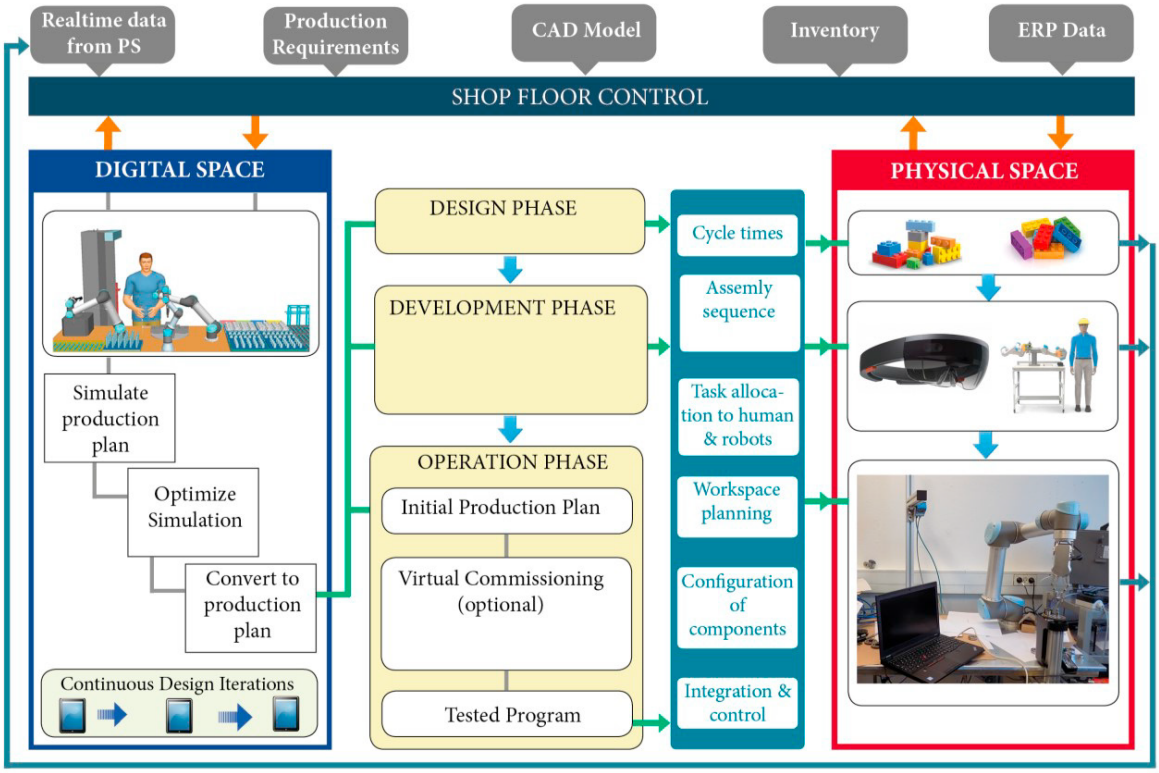


Figure 4: A digital twin framework of a human robot work cell, taken from [20].

When a digital twin is created for a human-robot-collaboration system, some requirements have to be satisfied [21]:

- Enable communication to be able to monitor the robot
- Be able to collect data from the sensors and be able to adjust the robot's movement with the date that has been collected
- To enable communication, proper protocols, and commands
- Data storage should be used in the form of Big Data

Communication is an fundamental part of the digital twin, and in [21], an OPC server can be applied to enable the active connection between the virtual and physical world. The OPC will be further reviewed in chapter 2.3.2.

Case study of digital twin with human-robot-collaboration A case study of an assembly workstation in a manufacturing company. The workstation used plastic and metallic parts in its products that are in the dimensions range of 20mm to 70mm. The company is looking to replace the fully manual work with human-robot-collaboration to be able to increase production at a reduced cost while maintaining the product variety. The human-robot-collaboration system used lean automation by dividing the tasks between human and robots based on repetition, intelligence required, and complexity [20].

The digital twin of the production system without the real-time connectivity made it possible to do what-if analysis for faster and safer human-robot-collaboration design. When the design is finished, the virtual model is connected to the system to detect potential errors. Finally, the virtual model is connected in real-time to the physical system for optimization and performance improvement [20].

The benefits of using a digital twin in a human-robot-collaboration are "*with each change in production parameters the behavior can be visualized and results are assessed without the risk of any financial loss or human injury in real production. The digital twin can help make what-if experiments and estimate results even without a real-time connectivity to the physical system. With advancement in information and communication technologies, DT can continuously be evolved in real time offering greater usefulness at system level*" [20].

2.2.4 Digital twin and autonomous system

Autonomous systems are machines that carry out a task without being programmed. These machines are the hallmark of flexible automation because they can adapt to different environmental conditions, production volume, and the product being produced. To be able to develop these capabilities, the system uses sensors to perceive their environment and find the current state of the situation. An autonomous system requires as much information as possible to be able to re-create the stat of the world. This includes "*the products to be manufactured, the geometry and affordances of the parts and tools to be used, as well as their own capabilities and configuration*" [8].

All the information is stored in a digital twin and is used to represent the full environment and the process state. The information in the digital twin will be used for forward simulation for action planning in the autonomous system. The simulation is used to predict the consequences of actions of the autonomous system. These capabilities are essential so that the system can make autonomous decisions over different action alternatives [8].

The difference between an autonomous system and automated systems is that automated systems perform a fixed sequence of actions. While an autonomous system knows how to perform its tasks based on information from their task, machine, and environment. This gives the autonomous systems the ability to handle variations in products being produced as well as production volume, without input from the supervisor or reconfiguration of the system [8].

2.2.5 The status of digital twin

The digital twin has in the last few years been applied to more eras of industries. This can be seen by the increasing number of publications on digital twin and patents the past years. There have been no efforts on reviewing the digital twin application in the industry [22].

A paper looked at all the relevant journal and conference papers released between January of 2003 and April of 2018 because digital twin was first introduced in 2003. More than 100 papers where found and these papers were then filtered out to find the papers that were related

to digital twin in industry. Finally, there where a total of 50 papers and eight patents. They found that the most relevant theories are [22]:

1. **DT modeling, simulation, verification, validation, and accreditation (VV&A):** Includes physical and virtual modeling, connection, and data modeling for maintaining a steady connection with the physical world.
2. **data fusion:** The digital twin needs a large amount of data. It is, therefore, necessary to include data cleaning, data conversion, and data filtering. The data goes afterward through rule-based reasoning and intelligent algorithm for analysis of the data. Then the theories for data optimizing are used.
3. **interaction and collaboration:** There where only two papers on interaction and collaboration. The digital twin can be used to react to dynamic changes on the physical model and at the same time. The can simulation be used to validate procedures in the virtual space.
4. **service:** Involves structure monitoring, lifetime forecasting, in-time maintenance, etc. It is used to predict when service is needed.

The application areas of digital twin are mainly in [22]:

- **Digital twins in the Product Design:** Be able to design new products by using a digital twin with a more responsive, efficient, and informed manner.
- **Digital twins in the Production:** The digital twin is used to make the production process more reliable, flexible, and predictable.
- **Digital twins in the Prognostics and Health Management(PHM):** The digital twin was first used to predict the structural life of aircraft's "*through multiphysics modeling, multiscale damage modeling, integration of the structural finite-element model (FEM) and damage models, uncertainty quantification, and high-resolution structural analysis*" [22].
- **Digital twins in Other Areas:** Other areas then mentioned above.

As seen in figure 5 the industrial application of digital twin which have been reported through publications have for the most part been in (PHM) and production.

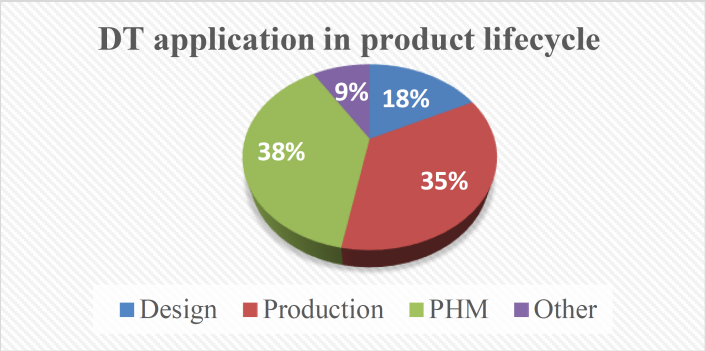


Figure 5: Distribution of digital twin publications in different areas [22].

The digital twin is still growing in popularity as can be seen in figure 6. Gartner which is a *"leading research and advisory company"*[23] mentions in a post that digital twin *"is beginning to gain adoption in maintenance, and Gartner estimates hundreds of millions of things will have digital twins within five years."*[24].

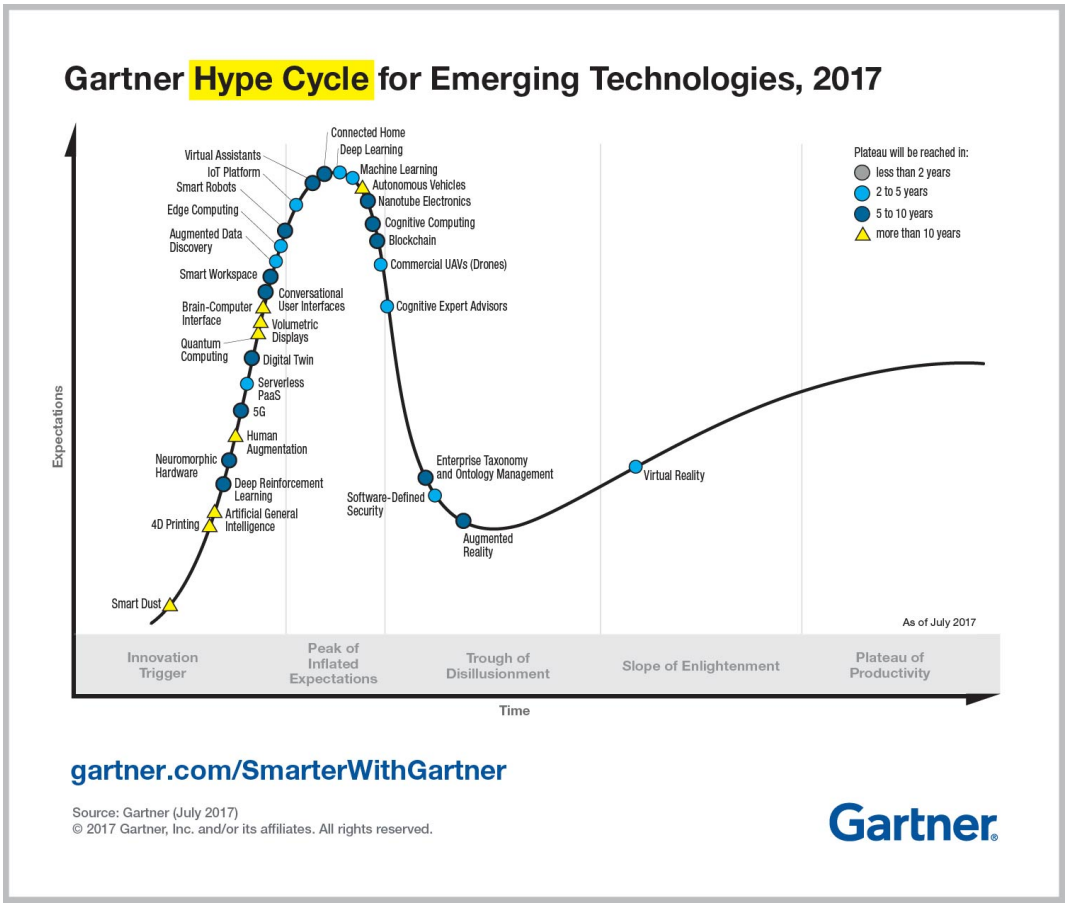


Figure 6: The graph shows new emerging technologies and how the expectations of these technologies develop over time [24].

2.3 Communication

One of the objectives for this master thesis is to use an OPC UA server as middleware to establish communication. Therefore this chapter will look at what OPC and OPC UA is.

Today you can connect physical things up to the internet, which makes it possible to get remote access to sensor data and be able to control the physical world from a distance. The Internet of Things (IoT) is a concept that wants to mash-up captured data with data retrieved from other sources [25]. IoT is an essential part of industry 4.0. It allows employees, machines, and products to communicate with each other. On approach to establish communication is OPC UA [26].

2.3.1 OPC classic

Open Platform Communications (OPC) classic is built on top of Microsofts OLE/DCOM technology for the windows operation system. It was made as a standard for accessing real time data [27]. The standard as three main functionalists [28]:

- **OPC Data Access (OPC DA):** Exchange of data (reading and writing) which include values, time and quality information
- **OPC Alarms & Events (OPC AE):** Exchange of alarms and events through message information. There is also variable states and state management.
- **OPC Historical Data Access (OPC HDA):** defines analytical methods to be applied to historical data like time-stamped data, and query methods.

Later as technology evolved and the needs for OPC has changed, the OPC Foundation released OPC Unified Architecture (OPC UA) in 2008. This platform has integrated all of the OPC Classic specifications and is backward compatible with OPC Classic [28].

2.3.2 OPC UA

OPC UA "*is the data exchange standard for safe, reliable, manufacturer and platform-independent industrial communication*" [29]. It is an IEC standard and enables communication and data exchange between different manufacturing products.

The main difference between OPC Classic and UPC UA is that UPC UA isn't based on Microsoft's DCOM technology. It does not depend on Microsoft's operating system can be adapted to be used in embedded systems and on Linux, macOS, and Windows.

There are also security concepts which offer protection against unauthorized access, sabotage, and modification. OPC UA has integrated encryption mechanisms when sending data and has recognized standard like SSL, TLS, and AES, which are used to secure data on the internet [29].

In April of 2015 the Reference Architecture Model for industry 4.0 (RAMI 4.0), recommended only the OPC UA standard for implementing communication layer. In November of 2016, a checklist for classifying and advertising product as industry 4.0 was made by the Industry 4.0 Platform. To be able to comply with the Industrie 4.0 communication criteria at the lowest level a product has to be addressable over a network using TCP/UDP or IP and have the OPC UA information model [30].

OPC UA model The OPC UA model defines how information has to be represented so that it is integrable with data from other systems. It allows OPC UA to be scalable for already existing standards to be used with application or devices that attempt to communicate through OPC UA [31].

Figure 7 shows the major elements of OPC UA client/server architecture. The server communicates with the clients through the Application Programming Interface (API). The API

exchanges service requests and responses, for example, messages and services. OPC UA provides an internal interface which isolates the application code form the OPC UA communication stack [26].

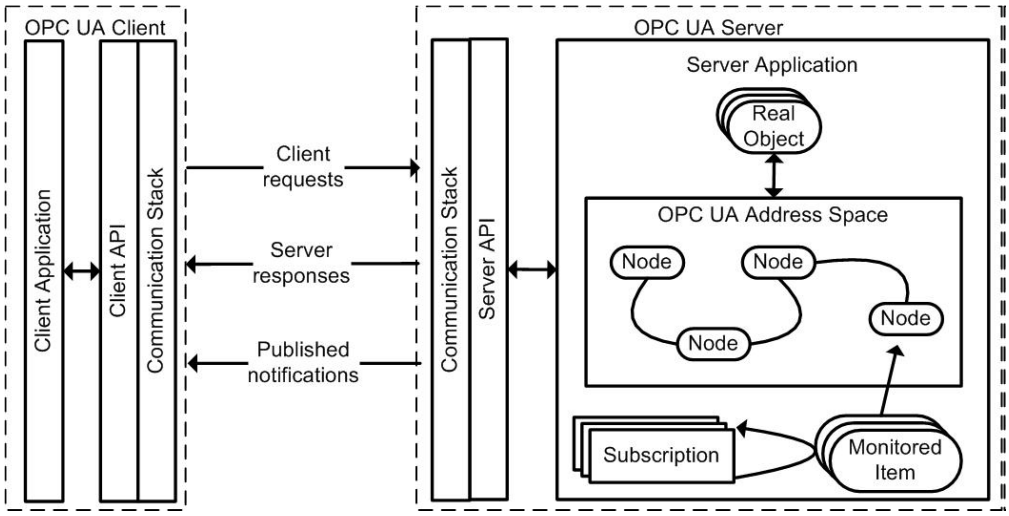


Figure 7: The major elements of OPC UA client/server architecture [26].

In the server, you have OPC UA Address Space which contains nodes. These nodes have a fixed set of attributes where some are mandatory, and others are optional. An example is that each node class has a unique identity, while the description of the node is optional [32]. The nodes are accessible by the clients, where the clients create references that are called ”Monitored Items”. These items show the changes in attributes and behavior. When a change is detected the data changes, or an event/alarm occurs. The client can choose the rate at which an update is received, and the data is only sent out to the clients who have subscribed to the item. [26].

The encoding of data can be done in two ways [27][32]:

- Extensible Markup Language (XML)
- UA Binary

The UA Binary sends a serialization of data in a byte string. It is faster than the XML encoding because the message size of XML is bigger than for UA Binary. However, XML is used to create communication between web services and is typically only used for UA Web Service mapping [32]. It can be used to represent both tabular data (data from a database or spreadsheet) and semi-structured data (such as a web page or business document). Using pre-existing formats such as comma separated value (CSV) work well for tabular data, however, handles semi-structured data poorly. That is why XML has gained widespread adoption [33].

XML allows for Simple Object Access Protocol (SOAP) clients to interpret data in SOAP messages [32]. In SOAP messages specification, the message does not have to go from the

initial sender to the ultimate receiver, but rather additional processing nodes can be integrated into the message path [34].

In an XML document the SPAO message contain the following elements [35]:

1. **Envelope:** Is used to define the start and end of the message and is a mandatory element.
2. **Header:** Has the attributes of the message and is used for processing the message. It is an optional element.
3. **Body:** Contains the information in the message which is being sent and is a mandatory element.
4. **Fault:** Its an optional element which provides information about errors and faults.

2.4 Simulation Software

When a digital twin is applied, there is often a 3D CAD model, to give a visual representation of the physical object/system. As mentioned in the scope of the thesis, this project will use the simulation software Visual Components to make a visual representation of the robot. This chapter will, therefore, take a closer look at the program Visual Components.

2.4.1 Visual Components

Visual Components is a leading developer in 3D manufacturing simulation software. It was founded in 1999 in Finland and had the goal to make manufacturing design and simulation easy and more accessible to a manufacturing organization. Today it is recognized as the global leader in the manufacturing industry [36].

In Visual Components there is a library with more than 1200 components from over 30 of the most significant brands in industrial automation. It is possible to import CAD files into the 3D world. It also features the ability to connect the simulation with your control system easily. You can either use the industry standard OPC UA or supported vendor-specific interfaces. It can be used to test out logical programs and gives you the ability to collect and analyze data from the PLC and test out improvements [37].

The architecture of Visual Components is open and makes it easy to customize the platform. Visual Components is built on .NET, giving developers using the program a familiar framework. There is also a Python API, to be able to customize everything from the UI to simulation behavior [38]. In other words, Visual Components is built with the ability to customize and change the simulation as needed.

2.5 KUKA robot and controller

In the lab, there is a KUKA KR 30-3 robot which is used for robotic welding as can be seen in figure 8. The robot can lift 30 kilograms and has a reach of up to 3,102 millimeters with six-axis [39].

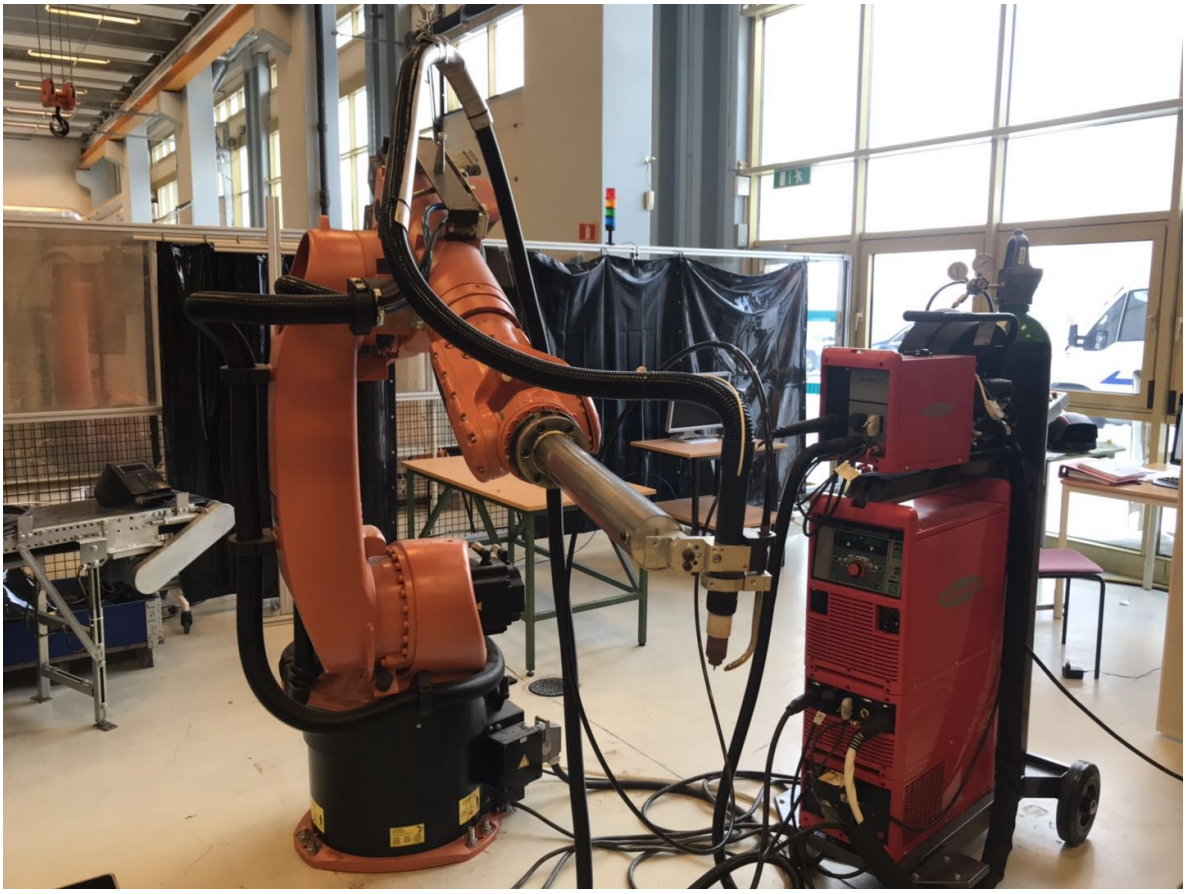


Figure 8: A picture of the KUKA KR 30-3 robot which is in the machine laboratory in Narvik.

The robot is connected to a KR C2 controller 9. The KUKA KR C2 controller has already been connected to a computer with serial port and Ethernet.

Figure 9: A picture of the KUKA KR C2 controller that is used to control the KUKA KR 30-3 robot.

The KUKA robot uses the KUKA Robot Language (KRL), and is text based. KRL allows declaration of variables and simple motions as well as interactions with tools and sensors via input/output. The KRL programs run on the controller. However, the interface is quite limited when it comes to research purposes. There is no native way to include third-party libraries. To expand the capability of the robot, software packages from KUKA have to be used [40].

2.5.1 JOpenShowVar and KUKAVARPROXY

To be able to create a digital twin of the KUKA robot, there has to be communication between the KUKA controller and the external computer. Many manufacturers are unwilling to share intimate details regarding their systems architecture, because of the high level of competition in the market, and the manufacturers doesn't want to share technologies. It is therefore difficult to exploit robotic platforms in a scientific context. There are only a few industrial manipulators that have an open platform interface [40]. There does, however, already exists an open-source communication interface for KUKA robots called JOpenShowVar.

JOpenShowVar is compatible with KUKA robot controller version 4 and KUKA robot controller version 2 and therefore should be consistent with the KUKA robot controller at UIT Narvik machine laboratory. JOpenShowVar is a Java open-source cross-platform communication interface that allows reading and writing of all controlled manipulators variables [40].

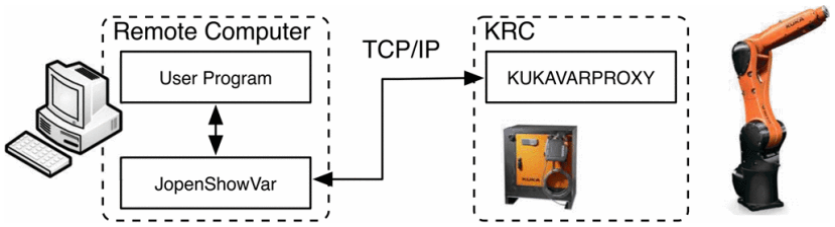


Figure 10: How proposed communication architecture for JOpenShowVar by [41].

The idea is to run KUKAVARPROXY on the KUKA controller and have JOpenShowVar running on a computer, as shown in figure 10. The communication is established by using Transmission Control Protocol/Internet Protocol (TCP/IP) [41]. TCP/IP "*specifies how data is exchanged over the internet by providing end-to-end communications that identify how it should be broken into packets, addressed, transmitted, routed and received at the destination*" [42].

KUKAVARPROXY is a multi-client server that can serve up to 10 clients simultaneously. It implements the KUKA CrossComm class that allows for "*selection or cancellation of a specific program, errors and faults detection, renaming program files, saving programs, resetting I/O drivers, reading variables and writing variables.*" [41]. The KUKA CrossComm class can only be remotely accessed by TCP/IP. The problem is that TCP/IP communication causes delays, which means that JOpenShowVar cannot provide real-time access to robot data. KUKA does not provide any information or documentation on communication speed, and therefore, there have been done several experiments to test the speed. From investigations to test the performance of the communication, the average time to access data was about 5 ms [40].

When reading variables with JOpenShowVar, the client which is the external computer must specify two parameters, the desired function which is "0" when reading and the name of the variable that you want to read. One example is reading the variable $OV_PRO, which is the to override the speed of the robot will have the format that is shown in figure 1. Then the client has to send 0009007$OV_PRO to read the override speed of the robot. The first to characters of the string is the identifier (ID) with an increasing integer number between 00 to 99, and the answer will contain the same ID so that it is possible to associate the right response to each request. The next to characters of the string specify the length of the segment in hexadecimal units. In the example shown in figure 1, 09 accounts for one character that defines the function, two characters indicates the length of the next segment and the seven is the number of characters variable to be read. The "type of desired function" is used to choose between reading and writing variables, in this example, it is reading. Next character indicates the variable length in hexadecimal units, and finally, you have the variable that is going to be read [43].

On the other hand, to change/write variables, the function will be changed "1" instead of "0" and the variable that changes have to be specified. At the end of the message the new value has to be defined as can be seen in figure 2

Field	Description
00	message ID
09	length the next segment
0	type of desired function
07	length of the next segment
$OV_PRO	Variable to read

Table 1: Reading variables in the KUKA controller with the JOpenShowVar program, taken from [43]

Field	Description
00	message ID
09	length the next segment
0	type of desired function
07	length of the next segment
$OV_PRO	Variable to read
50	value to be written

Table 2: Writing variables in the KUKA controller with the JOpenShowVar program, taken from [43]

2.6 Related work

The last part of the literature review is to look at related work that has been done. Looking at related work can give inspiration on how to solve the problem.

2.6.1 Digital twin with Visual Components and OPC UA

A master thesis from NTNU in 2018 [16], analyzed the potential benefits of using digital twins. It also investigated why OPC UA communication architecture is beneficial when developing digital twin.

To investigate the benefits, a digital twin was created of the robot cell at the Department of MTP Valgrinda. The robot cell contained four KUKA robots that were connected individually to a KUKA KR C4 controller w/smartPAD. The controller was connected to a network with a Simens PLC, figure 11 shows the setup of the robot cell.

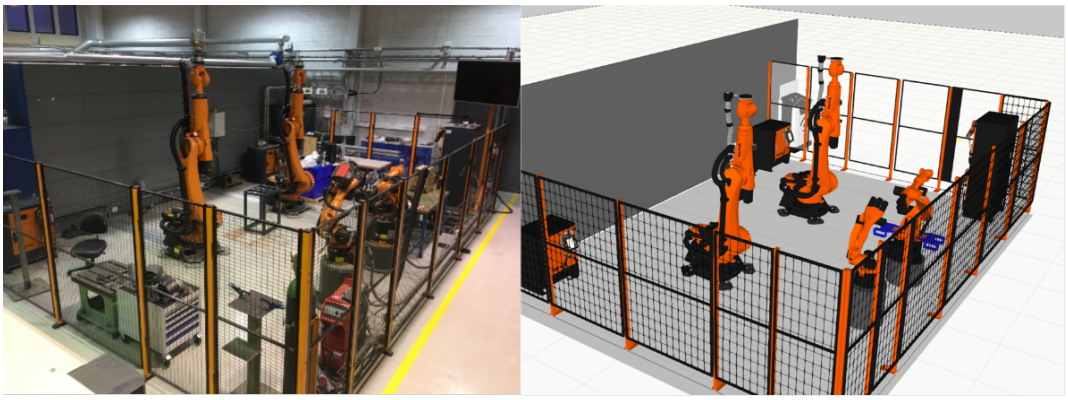


Figure 11: A physical model on the left and digital model on the right, of the robot cell at MTP Valgrinda [16].

There where three simulations and visualization software which were considered Visual components 4.0, KUKA.Sim and Simens SIMATIC WinCC. The software used for this project was Visual components 4.0 because Simens SIMATIC WinCC didn't have pre-defined components and KUKA.Sim wasn't available for use. It's important to note that KUKA has acquired Visual Components in December of 2017 [44].

The biggest challenge developing the digital twin was *"the development of a robust communication system, complying with OPC UA standards and including the functionalities that could be useful in a DTw."*. Therefore most of the work done in the project was creating communication modules to connect Visual Components to the physical robot. From the project, a communication library was created, which is available on Github.

Figure 13 shows how the communication between the different entities works. There was already a server made for KUKA, which supported the OPC Classic, however, the software wasn't available under the project. There had already been made an open source communication interface called "JOpenShowVar" and KUKAVARPROXY, as mentioned in chapter 2.5.1. The JOpenShowVar communication interface had been written in Java. To avoid cross programming language communication between Java(JOpenShowVar) and python (OPC UA server), a translated version of JOpenShowVar in python made by Ahmad Saeed from his GitHub Repository was used [45].

To implement the OPC UA standard, a free open source library called "FreeOpcUa" was used as a foundation for the OPC UA server. A windows computer was used as middleware between the computer running Visual Components and the KR C4 robot controller, as shown in figure 12.

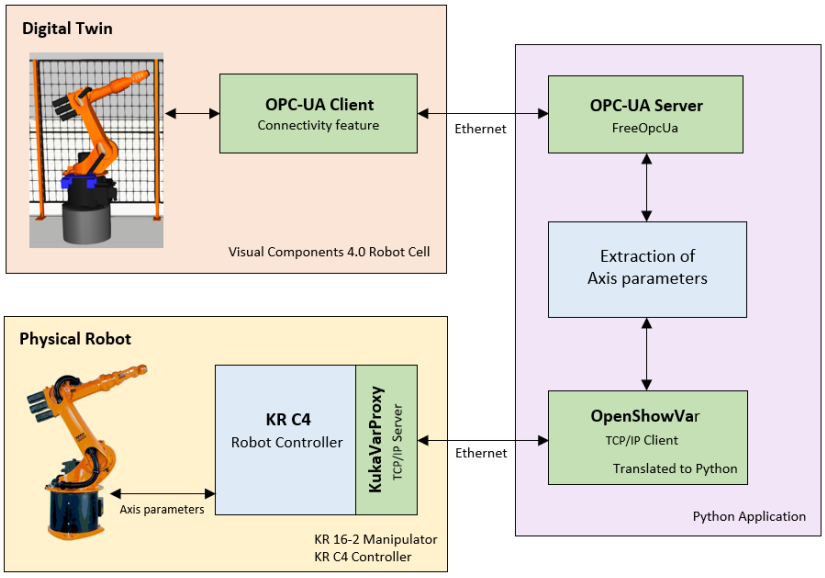


Figure 12: How the communication works for the digital twin at MTP Valgrinda [16].

Further functionality of the digital twin was implemented. Four servers were created to share the value of velocity, torque, current and temperature of the motor controlling the x-axis in real time. Four more servers were designed to save the data collected in real time to .CSV files for later analysis. Since there where so many servers, a graphical user interface (GUI) was created to have a structured environment to run the different servers. Figure 13 show the result of how the digital twin works.

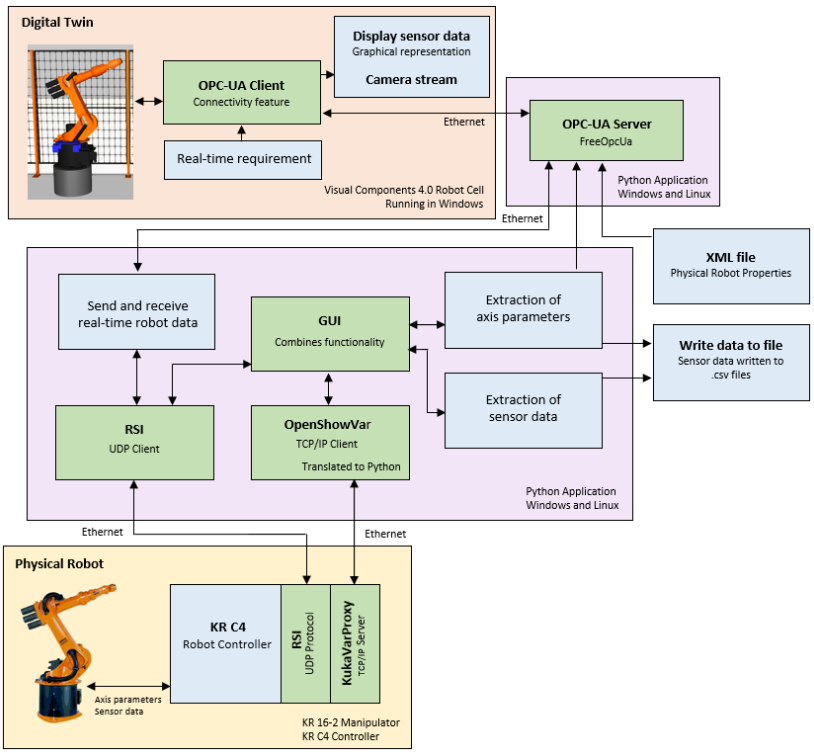


Figure 13: How the communication works for the digital twin at MTP Valgrinda [16].

In the ”FreeOpcUa” there is an XML-modeler used to construct XML-files with information about the object in the communication network. The XML file contained to Boole values called KUKAVARPROXY and RSI, as well as six Double values containing each axis of the robot. There was also a string containing the IP of the KR C4 robot controller. To be able to extract the relevant information from the XML-file a separate OPC UA server was created with ”FreeOpcUa” as the base.

To increase the communication speed and get a cycle rate of ca 4ms, Robot Sensor Interface (RSI) communication was used. There had already been developed a KUKA RSI-3 Communicator by Eren Sezener from BCCN - TU Berlin (https://github.com/erensezener). However, the RSI communication model was made by using a run.py file as a base made by Torstein Anderssen Myhre. It was found that the RSI communication had to run on Linux because the communication broke down after 100 iterations on a windows computer. The reason the communication broke down was likely ”*caused by the received data packages arriving too late from the robot controller, and after a buffer size of 100 late packages the communication shut down. This is a part of the RSI's more strict real-time requirements.*” [16].

After testing the KUKAVARPROXY and RSI communication in parallel, it was found that the RSI server had a higher possibility of breaking down. KUKAVARPROXY was a more stable solution. It did, however, have a more significant time delay. The RSI module had to be run as a separate program on the robot controller, which means that it was not possible to run other programs. KUKAVARPROXY was running in the background on the robot controller, and the robot could, therefore, execute programs. There was also some unwanted behavior where

the robot moved in unintended paths and ejections of tools during an operation, especially when RIS controller servers was developed.

Further work In the end of the master thesis in the chapter called ”Further work,” it mentions that further work on this project is ”*Develop RSI server controlling robots from VC 4.0*”. The digital twin did not have the ability to control the robot with RSI communication. It did only have the ability to control the robot from using OpenShowVar and KUKAVARPROXY.

3 One-way digital twin

The master thesis has been divided into three parts. The first part, is called ”One-way digital twin”, consists of visualizing the robot movement in Visual Components. This includes making a model of the laboratory in Visual Components, creating an OPC UA server in python and connecting it to Visual Components and being able to retrieve the rotation or position of the robot joints and send them to Visual Components.

3.1 Visual Components model

The first task will be to create a model of the laboratory in Visual Components. The Visual model has been limited to the area of the robot and not the whole laboratory.

Measurements were taken with a measuring tape and an AR app, called ”Mesaure” [46] was used in areas that were long and hard to measure. The app is developed by Google and can be used with most phones that support ARCore.

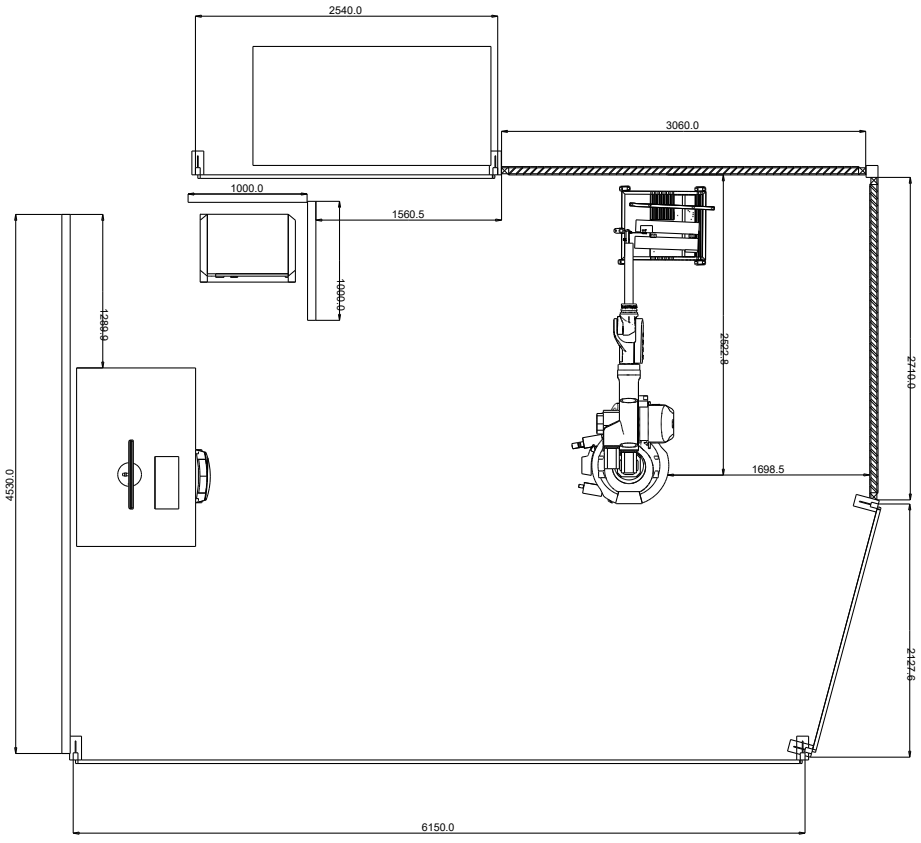


Figure 14: 2D drawing of the machine laboratory in Narvik, with measurements.

When all the measurements had been taken, a model was made in Visual Components. In Visual Components, there is already a cad-model of the KUKA KR 30-3 robot and the KUKA KR C2 controller in the default library. The rest of the components like fences, tables, and walls were found in the default library by finding similar parts.

As mentioned before the KUKA robot at the laboratory was used for robotic welding, and the end of the robot is therefore extended with a welding head. Another student had already created a CAD model of the welding head and the table that is used to weld on. These parts were imported and used to save time and to be able to make a more accurate model of the laboratory. In figure 15 you can see the real laboratory on the left and the Visual Components model on the right side.

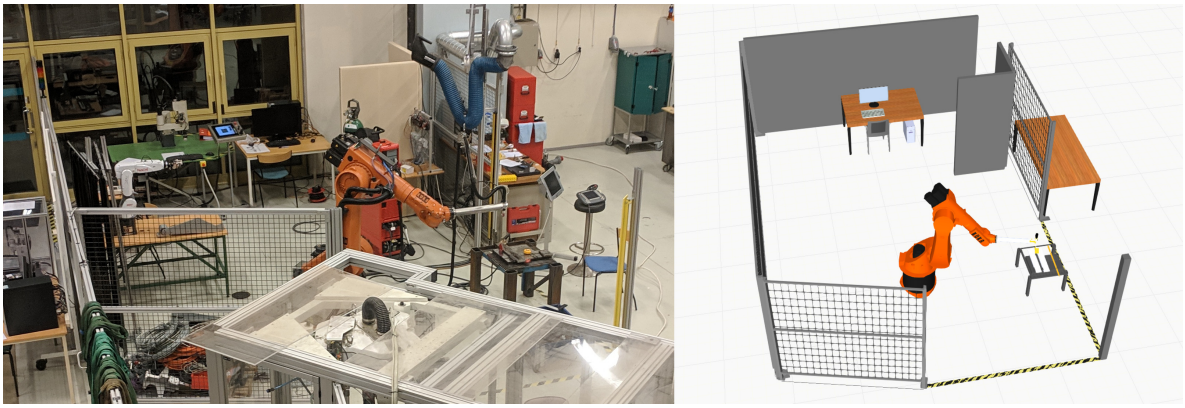


Figure 15: The physical model of the laboratory on the left side and the Visual Components model of the laboratory on the right side.

From the figure, you can see that the digital model is simplified compared to the physical model. This is done because the robot is used for robot welding, and the robot never turns 180 degrees around since the welding equipment limits its movement. Equipment like hoses and tubes is challenging to model in, and they have unique features when the robot moves. They are, therefore, not included in this model.

3.2 Connecting Visual Components to OPC UA

As mentioned Visual Components has a Python API, it can, therefore, be beneficial to use a python to create the server to avoid cross programming languages.

3.2.1 OPC UA server in python

An OPC UA server can easily be created using the ”FreeOpcUa” library [47] as mentioned in chapter 2.6.1. The library can be installed by writing ”pip install opcua” in the command prompt on windows. In the library, there are examples of codes on how to set up the OPC UA server. There is a file called ”server-minimal.py” that shows how to create a minimal server. It was used as a template to create the server. In the server, you set the URL which the clients can connect to, and you can create folders with variables that can be int, double, or Boolean.

The server that was created has a folder which contains six variables, one for each of the rotations. The variables are set to be writable so that clients that connect to the server can change them.

3.2.2 Connecting Visual Components to the OPC UA server

AS mentioned above, you can connect the simulation to the OPC UA server. On Visual Components website [48], you can find a brief YouTube tutorial, that shows you how you can easily connect the simulation to OPC UA server. It is done first by adding the connectivity tap to Visual Components and then connecting to the OPC UA server with the same URL as the server has. When you are connected, you can pair variables from the OPC UA server with variables in the simulation, as shown in figure 16.

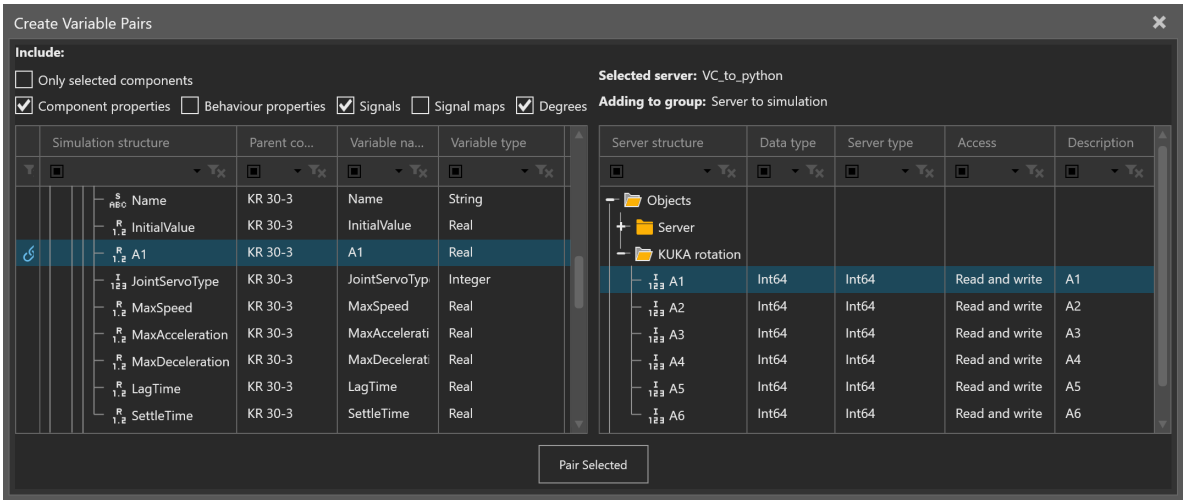


Figure 16: Shows the variables from the OPC UA server in Visual Componenets.

3.3 Communication with the KUKA robot

The KUKA KR C2 controller in the machine laboratory in Narvik is compatible with the OpenShowvar and KUKAVARPROXY software. The KUKA controller in Narvik has also the add-on package for RSI communication to control the robot.

As mentioned in chapter 2.6.1, the master thesis from NTNU [16] found that RSI communication had a higher probability of breaking down and was more unstable compared to OpenShowVar and KUKAVARPROXY. Therefore the OpenShowVar and KUKAVARPROXY communication method were first used to test if it is the right solution for the KUKA KR C2 controller to create a digital twin.

3.3.1 Setting up KUKAVARPROXY

The files for KUKAVARPROXY were obtained from [49] and to install KUKAVARPROXY on the controller, you place the KUKAVARPROXY.exe somewhere in the Windows environment. It might be beneficial to set the program as a start-up program so that it doesn't have to be started each time the controller is restarted. When the program is started, you can see how many computers are connected to the software, and if you press on the "Debug" button, you can also see what information is sent, which can be seen in figure 17.

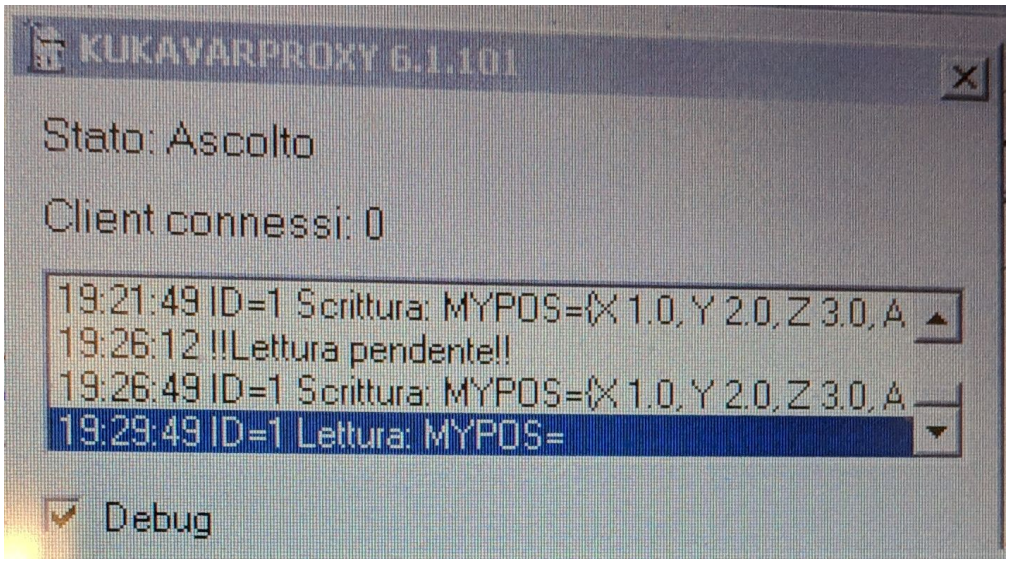


Figure 17: Picture of the KUKAVARPROXY program running on the controller in debug mode.

3.3.2 Communication with KUKAVARPROXY

KUKAVARROXY uses a unique way to receive end send data, as mentioned in chapter 2.5.1. To be able to send and receive data with python the same code was used as in the master thesis from NTNU [16]. This code worked to send and receive data with KUKAVARPROXY running on the controller.

3.4 Assemble One-way digital twin

The last part of enabling the one-way digital twin is to integrate the KUKAVARPROXY communication with the OPC UA server. It is done by putting the code to communicate with KUKAVARPROXY in python file and call the methods in the server to receive and send data with KUKAVARPROXY. Figure 18 shows how the one-way digital twin works.

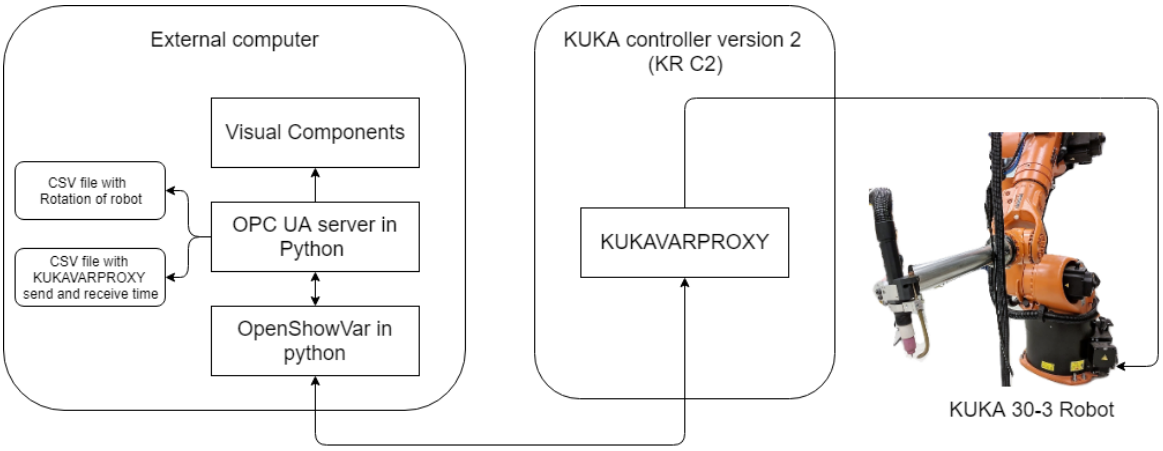


Figure 18: A simple flow diagram of how the one way digital twin works.

When the program runs, two .CVS, files are created as seen in figure 18. One that saves the time it takes to get the rotation information from KUKAVARPROXY, and one that saves the current rotation of the robot with the present time. The graph in figure 19 shows the time it takes to get data from the robot controller using KUKAVARPROXY.

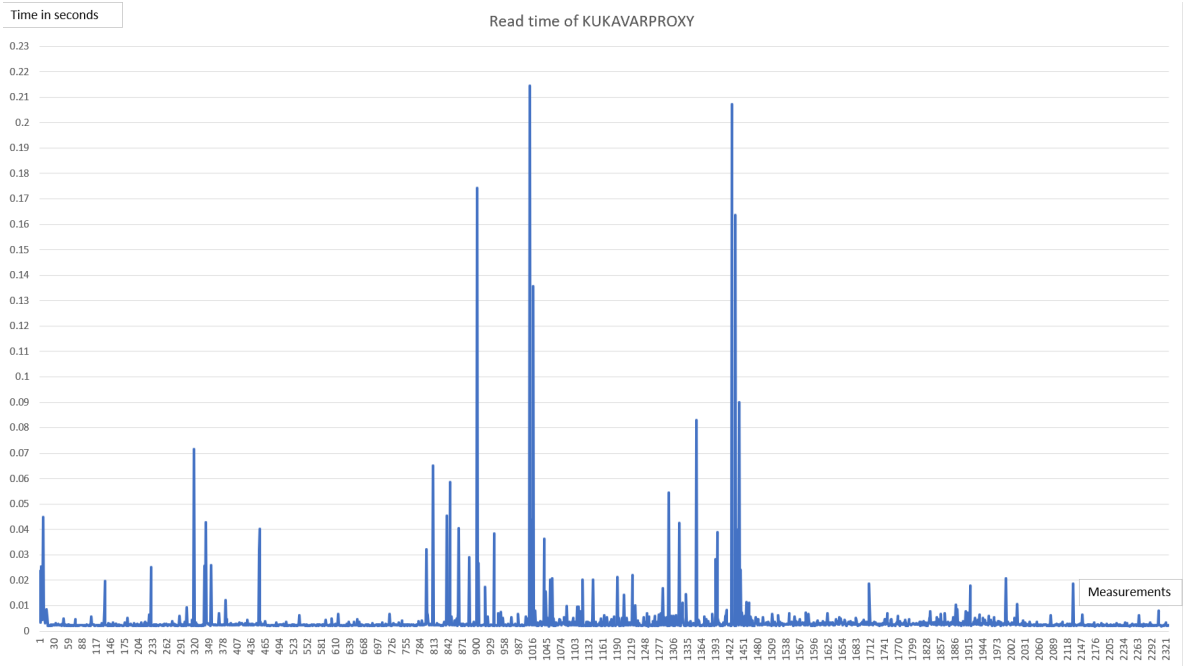


Figure 19: The graph shows how long it takes to read the position of the robot joint when using KUKAVARROXY.

From a test with a sample size of around 2300, the average time it takes to receive the rotation of the robot with KUKAVARPORXY is 3.79 milliseconds. Data is however not sent with a constant speed as we can see from the graph, with a max time of 214.5 milliseconds and min of 1.92 milliseconds.

The code for the one-way digital twin can be found in appendix section B.1.1 and two Youtube videos have been made of the one-way digital twin:

1. `https://www.youtube.com/watch?v=wdgzXG8xIz4`
2. `https://www.youtube.com/watch?v=EWYvDx7DCG8`

The computer that is connected to the KUKA controller is inside the area where the fence is. It would, therefore, be safer to control the computer outside the fences remotely. The program TeamViewer was used to manage the computer connected to the KUKA controller, as shown in figure 20.

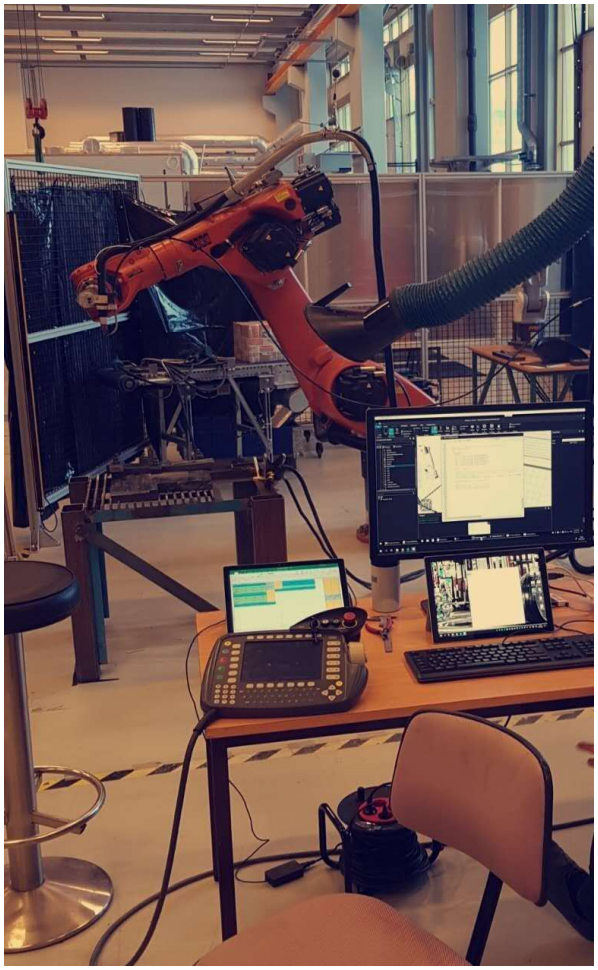


Figure 20: The picture show the setup when testing on the KUKA robot was done.

4 Two way digital twin

After creating the one way digital twin, the next step is to create the two way digital twin, which controls the robot with Visual Components. This part includes sending position data from Visual Components to the OPC UA server and being able to control the robot from a remote computer.

4.1 Send data from Visual Components to the robot

To be able to send rotation data from Visual Components to the OPC UA server, almost the same method was used as in the one-way digital twin. When you're paring variables in Visual Components, you can either choose "Simulation to server" or "Server to simulation" as shown in figure 21.

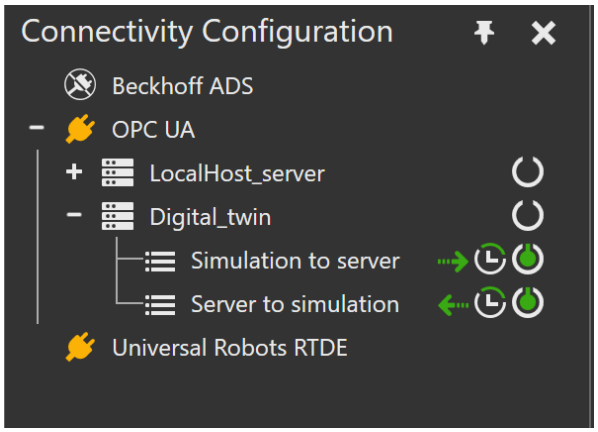


Figure 21: A screenshot of connection options in the connectivity tap in Visual Components.

To switch between "Simulation to server" and "Server to simulation," you can double click on the white circle, to turn it on or off, as seen in figure 21.

4.1.1 Controlling the robot

It is easy to control the robot in Visual Components. There is a tap called "PROGRAM," and it allows the user to create subprogram with sequential movement. It is also possible to use if statement's and call another subprogram. This is, a rather simple tool, and if you have a lot of equipment and if statments, it can be hard to program or re-program the system.
As mentioned before Visual Components uses a Python API. Python scripts can be created to control the robot and also other objects, with object-oriented programming. There are already Python methods that can be used to manage the robot or other equipment in Visual Components such as conveyor belts. You can also read variables of other robots by using python scripts and it gives a much easier way to control the robot in Visual Components. The subprograms created in the "PROGRAM" tab can also be run from the Python script.

However, using a python script requires that the user knows how the Python works and it is harder to use compared to the "PROGRAM" tap in Visual Components.

4.2 Controlling the KUKA robot

KUKAVARPROXY can be used to control the robot. An example from [16] was used to control the robot. It is done by using KUKAVARPROXY to change a variable in the KUKA controller. In the KUKA programming language, you have a type of variable called “E6AXIS”, which contains all the angle values (A1, A2, A3, A4, A5, A6) of the robot and the angle values for the external axes (E1, E2, E3, E4, E5, E6).

To be able to control the rotation of the robot using python, first two files called “openshowvar_KUKAVARPROX.dat” and “openshowvar_KUKAVARPROX.src” which is made from the code created in the NTNU thesis [16], has to be placed in the folder `C:\KRC\ROBOTER\KRC\R1\`. The code can be found in appendix section B.4.1. The files will show up in the programs menu on the controller. Some of the code in the program is shown under:

```
PTP  XHOME

TARGET_AXIS = $AXIS_ACT

LOOP
PTP TARGET_AXIS
ENDLOOP
```

The program works by first setting the robot in its home position which is defined in ”$config.dat” file on the controller. When the robot has reached its home position, the variable TARGET_AXIS paired with the variable $AXIS_ACT. Afterward, the robot goes into an endless loop, where it continuously goes to the position of TARGET_AXIS, which is the same position as $AXIS_ACT. When the robot moves to its destination, it uses the maximum axis specific acceleration and velocity of the leading axis called ”point to point” method [50].

When the program is looping, the OpenShowVar python version is used to change the variable $AXIS_ACT and sending new rotations of the robot. To test and control the robot, a Python program was made that first sets the speed of the robot by changing the variable “$OV_PRO” and then reads the position of the robot. The program increases and decreases the angle of rotation A1 with 0.5 degrees every second, and if the angel is four degrees bigger than the first read of the angle, the robot will start to move in the other direction. The python program was tested and worked.

4.3 Assemble the programs

The next step is to put the robot control program together with Visual Components, to be able to control the robot with Visual Components. The structure of this program is similar to the one-way digital twin, as can be seen in figure 22.

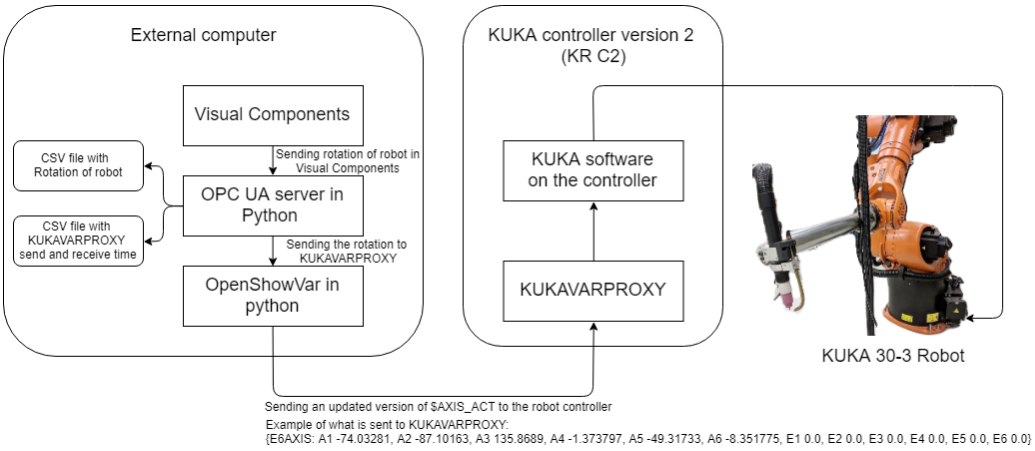


Figure 22: Flow diagram of how the two way digital twin works.

Visual Components sends the rotation of the digital robot to the OPC UA server. The OPC UA server then uses the python version of OpenShowVar to send new position through the variable $AXIS_ACT. As mentioned before the robot has to be looping with a ”point to point” movement on $AXIS_ACT to be able to move.

The same methods for saving the rotation as well as the time of KUKAVARPROXY is also saved in a .CSV file. A test was performed to get information on how long it takes to send new rotation to the robot is shown in figure 23. The average sending time is 2.4715 milliseconds, with a maximum time of 28.5579 milliseconds and minimum of 1.9362 milliseconds.

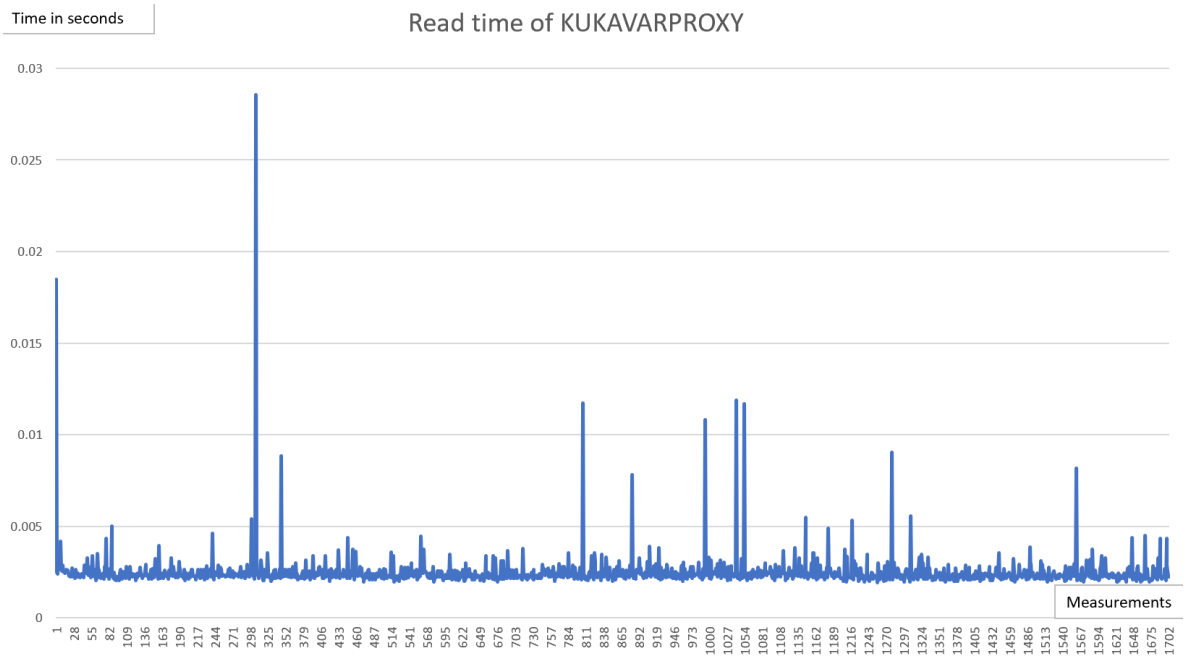


Figure 23: A graph of the time it takes to send down new rotation to the robot controller.

The program was tested, and there were some problems. When the robot moved at high speed, it was able to follow the Visual Components program, but the movement of the robot was

unsmooth. However, if the speed were reduced to 10%, the robot would move much better, but there would b a delay of 6-10 seconds. A Youtube video that shows the robot at 100% and 10% speed can be found at `https://www.youtube.com/watch?v=vQxy7p4qPIU` and the code can be found in appendix B.2.1.

After talking to my supervisor Gabor Sziebig and Beibei Shu who also have worked with Visual Components, it was suggested that Visual Components was to slow. In the master thesis from NTNU [16], it was also mentioned that Visual Components is slow when it comes to sending data and that Visual Components doesn't send data in a continuous phase. The master thesis from NTNU was however done with an older version of Visual Components compared to the one used in this master thesis. It was therefore suggested to make a program that didn't use Visual Components, to see if OpenShowVar-KUKAVARPROXY or Visual Components was the problem.

4.3.1 Robot control without Visual Components

To test and see if Visual Components was the problem, a new python program was created. The program moved the robot from one start position to an end position gradually with a selected amount of steps. When the program was tested, the robot still had an unsmooth movement. A video was taken of the movement and according to the video the robot was 2.14 seconds behind the python program. The video can be found at `https://www.youtube.com/watch?v=I_3kEDNeW5s` and the code can be found in appendix B.2.2.

To further test KUKAVARPROXY a second program was created that used the arrow keys on the computer keyboard to move the robot. The left and right arrow keys moved the A1 axis while the up and down arrow keys moved the A2 axis. From the test with the keyboard control, it turned out to be too much delay in using KUKAVARPROXY. A video was also taken of this program, and can be found at `https://www.youtube.com/watch?v=Oyrc2VMLHGw` and the code can be found in appendix B.2.3. From the video, the robot has a delay of between 0.8 to 1.5 seconds.

As can be seen in the Youtube videos from the tests, the robot either has a delay of 6-10 seconds when the speed of the robot is high or moves in an unsmooth manner when the speed is high.

From the tests, it seems that when the robot moves "point to point" it takes and reads the variable TARGET_AXIS and then moves to that point. If the variable TARGET_AXIS gets updated while the robot is moving the robot first goes to its previous position stops and then goes to the new TARGET_AXIS position. Since the robot stops every time it reaches a position, the robot doesn't move in a smooth manner.

The speed of the robot can be changed by using either the controller or KUKAVARPROXY. A simple up and down program was made that controlled the robot from Visual Components and when the robot is at 10% speed the robot moves smooth, but with a 5-10 seconds delay. When the speed is increased to 100% there is little delay but the robot moves unsmoothly again. A Youtube video has been made that shows when the robot moves at 10% speed and 100% speed `https://www.youtube.com/watch?v=vQxy7p4qPIU`.

The reason for that the robot moves unsmoothly when the speed is high is because the robot never reaches its maximum speed. The robot ends up accelerating, decelerating and stopping to read the next position. When the robot speed is low it is able to reach its maximum speed faster and because the robot is so far behind the Visual Components program the robot only reads positions that are far away. Figure 24 illustrates the problem with using KUKAVARPROXY to control the robot. The first graph shows what happens when the speed is 100% and the second graph with 10% speed.

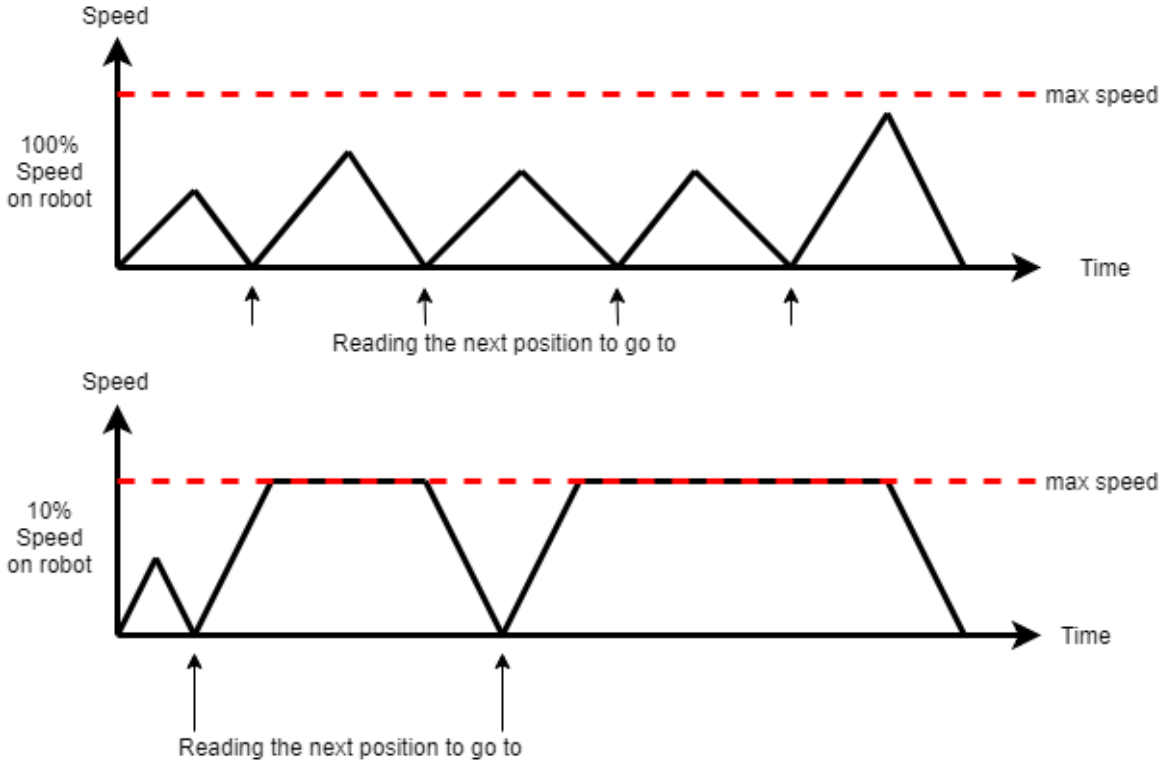


Figure 24: A graphical illustration of the problem with KUKAVARPROXY.

After discussion with Gabor Sziebig, he suggested to try out a Submit interrupter (SPS), and if that did not work out, then he recommended Robot Sensor Interface (RSI) control function.

4.3.2 Submit interpreter

A submit interrupter is a program that runs independently from the selected program that is running [51]. In default the robot has a startup interpreter called sps.sub, but it can easily be replaced with another interrupt program.

The idea is to create a submit interrupter program that updates the variable while the robot is moving. It is done by setting the used variables equal to another variable or the same variable. After testing the submit interpreter, it didn't seem to influence the robot movement. To check if the submit interrupter was running, a console print function was put inside the method. The submit interrupter worked as it should. An interrupt was also made inside the submit interrupter to try to update the position, but it also did not work.

Since there was limited time with the robot, the conclusion is that OpenShowVar-KUKAVARPROXY was not suitable for good and fast control of the KUKA robot. Therefore the work started on controlling the robot with RSI.

4.4 RSI communication

KUKA RSI is an add on package for the KUKA controller and can be used for [52]:

- Influence the motion of the robot with the means of data exchange via Ethernet connection.
- Configuration of the data exchange between the KUKA controller and the external system via Ethernet.

The communication with the external system uses an interpolation cycle of 12 ms parallel to the program being executed. The data exchange can be transmitted with either TCP\IP or UDP\IP with messages as XML strings.

To be able to communicate with an external system, you need the RSI object ST_ETHERNET. Afterward, the object has to be created to enable control of the joints in the robot. When all the objects have been created the command ST_ON is used to start communication as seen in figure 25.

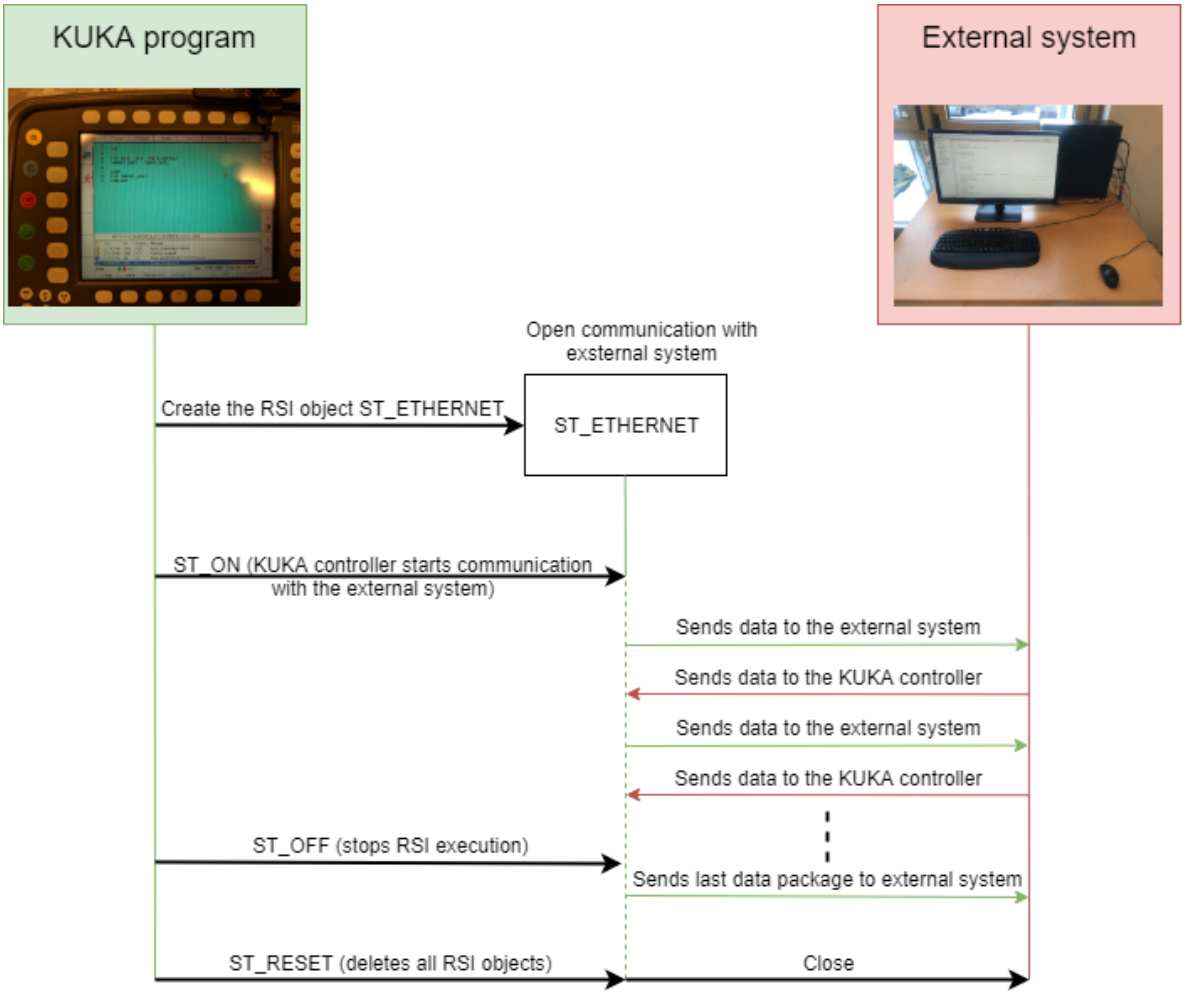


Figure 25: A simple figure of how the RSI system works. It is based on the figure from the KUKA RSI manual [52].

When the data packet is sent to the external system, the external system has to answer the KUKA controller within approximately 12 ms. If the data packages are not answered within the right time, the controller will classify them as late, and if the number of late packages exceeds the maximum number of late packages the data exchange will stop.

4.4.1 Setting up RSI

The KUKA RSI package included an example program that uses RSI communication. It includes:

- KRL program "RSIEthernet.src" and "RSIEthernet.dat"
- Configuration file "RSIEthernet.xml"
- A server app "ServerApplication.exe"
- sample source code "C#_ServerApplication.cs"

The original files were first placed in the robot controller. The KRL programs "RSIEthernet.src" and "RSIEthernet.dat" are placed in `krc\ROBOTER\KRC\R1\Program`, while the XML text "RSIEthernet.xml" is placed in `krc\ROBOTER\INIT` on the robot controller. These files can be found in appendix section B.4.2 and B.4.2. The server app was then tested to verify that everything works.

The XML file that is inside the KUKA controller is used to define what is sent by the robot controller to the external system and what the KUKA controller expects to revive from the external computer. It is also used to define IP number, port, and what protocol you want to use. The example XML file "RSIEthernet.xml", was used because it makes the KUKA controller send out the current position of the robot in x, y, z, a, b, c (Cartesian coordinates) and the rotation of the axis (A1-A6). With this XML file, the robot expects to receive change in the x, y, z, a, b, c or a difference in the rotation axis.

The KRL program is used to send the XML string with data to the external system. The example code "RSIEthernet.src" uses the ST_ETHERNET to first open communication with the external system and then creates objects that enable control of the robot with x, y, and z coordinates. If these objects are not created, it is not possible to control the robot with RSI. A manual has been created that shows how to create objects for the KUKA robot to control the robot with the A1-A6 axis. The manual can be found in the appendix D.

The KRL is then turned on with the command ST_ON, and the program then waits with the line ST_SKIPSENS() until RSI brakes.

It was included a sample code "C#_ServerApplication.cs", that communicates with the robot through RSI. All the code does is connect to the IP address and port. It then waits for a package from the KUKA controller, and when it receives the packages, it answers and sends a package back to the KUKA controller.

When the RSI program is running on the controller, and the example KRL and XML code is used, the robot sends this XML string:

```
<Rob Type="KUKA">
<RIst X="-64.4407" Y="1924.0547" Z="1668.3203" A="179.9997" B="-1.6011" C="-179.6991"/>
<RSol X="0.0000" Y="0.0000" Z="0.0000" A="0.0000" B="0.0000" C="0.0000"/>
<AIPos A1="-89.9998" A2="-90.0000" A3="90.0014" A4="0.0000" A5="-0.0006" A6="-0.0011" />
<ASPos A1="-89.9998" A2="-90.0000" A3="90.0014" A4="0.0000" A5="-0.0006" A6="-0.0011"/>
<EIPos E1="0.0000" E2="0.0000" E3="0.0000" E4="0.0000" E5="0.0000" E6="0.0000"/>
<ESPos E1="0.0000" E2="0.0000" E3="0.0000" E4="0.0000" E5="0.0000" E6="0.0000"/>
<MACur A1="0.3604" A2="-1.4814" A3="1.7451" A4="0.0227" A5="-0.4526" A6="0.0166"/>
<MECur E1="0.0000" E2="0.0000" E3="0.0000" E4="0.0000" E5="0.0000" E6="0.0000"/>
<Delay D="0" />
<Tech C11="0.0000" C12="0.0000" C13="0.0000" C14="0.0000" C15="0.0000" C16="0.0000"
C17="0.0000" C18="0.0000" C19="0.0000" C110="0.0000" />
<DiL>0</DiL>
<Digout o1="0" o2="0" o3="0" />
<ST_Source>0.0000</ST_Source>
<IPOC>4208611517</IPOC>
</Rob>
```

The external system has to take out the number that is in between IPOC and put it into the XML string which is returned to the KUKA controller. As mentioned before, this has to happen within 12 ms. The external system will then return the XML string:

```
<Sen Type="ImFree">
<EStr>ERX Message! Free config!</EStr>
<RKorr X="0.0000" Y="0.0000" Z="0.0000" A="0.0000" B="0.0000" C="0.0000" />
<AKorr A1="0.0000" A2="0.0000" A3="0.0000" A4="0.0000" A5="0.0000" A6="0.0000" />
<EKorr E1="0.0000" E2="0.0000" E3="0.0000" E4="0.0000" E5="0.0000" E6="0.0000" />
<Tech T21="1.09" T22="2.08" T23="3.07" T24="4.06" T25="5.05" T26="6.04" T27="7.03" T28="
<DiO>125</DiO>
<IPOC>4208611517</IPOC>
</Sen>
```

To be able to move the robot, the variables in RKorr has to be incremented. The robot can be moved with either X, Y and Z or A, B and C. It is not possible to move the robot with the rotation axis using the example code. There has to be created objects for them, as mentioned before.

4.4.2 Controlling the robot with RSI

The example program ”C#_ServerApplication.cs” was expanded, and a new method was created that changes the X, Y and Z values of the robot. This made the robot move, but the movement was limited because the robot has a default limit on how far the robot can move. The default movement limit on X, Y, Z and A, B, C is a lower bound of -5mm and upper bound of 5mm. By using the ST_PATHCORR the limit was expanded to 100mm for upper and lower limits, and a simple program was made to test the movement. The Program moved the robot 0.1mm every 12 ms for 7.2 seconds and then the robot moves in the opposite direction. This movement was tested for X, Y, and Z corrections.

The one-way digital twin used the rotation of the axis to control the robot, therefore the ability to use rotation axis will also be added. As mentioned before the objects for correction of the rotation axis must be added, which is described in the manual (appendix section D). The code

can be found in an appendix in section B.4.4, and the same XML file was used as for x, y, and z control. There are also limitations for this movement as well and are by default ±5 degrees. By using the command ST_AXISCORR the limits were increased to ±100 degrees, which should be efficient for testing.

A new method was created for controlling the robot with the rotation axis. This method takes in the XML string with the right IPOC number and then adds new values in the AKorr section. The test program that was used to see if the X, Y, and Z movement worked, was also used to test and see if the rotation axis method worked. The code can be found in appendix B.2.4.

4.4.3 RSI with Python

The OPC UA server is written in python, and Visual Components, as mentioned before, uses a Python API. It will, therefore, be easier to have the code that communicates with the robot controller in python, to avoid communication between different programming languages in the external computer.

The C# program that could control the robot with both Cartesian coordinates and the joint coordinates was then translated over to python. Two programs were made in python that uses RSI:

- One that uses Visual Components to control the robot
- The second program controls the robot with a python program and while Visual Component constantly imitates the KUKA robot in real time through OPC UA.

4.4.4 Visual Components control with RSI

As mentioned before, when creating the two-way digital twin, the KUKAVARPROXY program is not good enough to control the KUKA robot. Therefore a second two-way digital twin program was made with Visual Components where RSI was used to control the robot. The program still uses KUKAVARPROXY to get the information of the current position of the robot, to be able to calculate the deviation from the Visual Components model and the real model.

The program has one main script that creates an OPC UA server and controls the robot with RSI communication. Two other scripts also have to run to make everything work. The two other scripts connect to the OPC UA server, and the one called ”client” uses KUKAVARPROXY to find the current position of the robot. The second script called ”client_dif” calculates the difference between the current joint position of the robot and the desired position and starts a simple P regulator on the robot. Figure 26 shows how the program works.

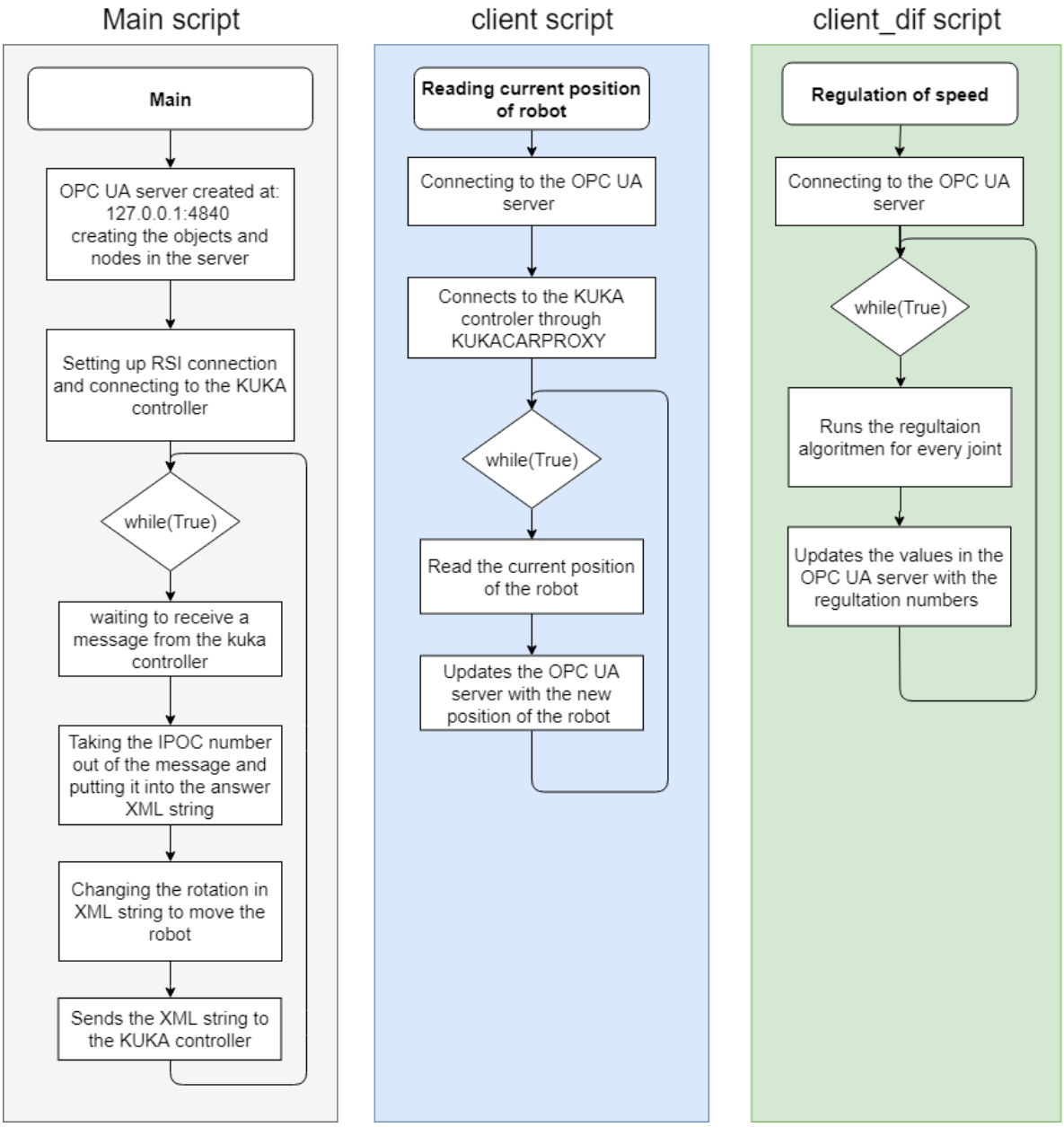


Figure 26: Flow diagram of how the Visual Components RSI program works.

When KUKAVARPROXY is used to control the robot, you specify the coordinates to where the robot is supposed to go. It then uses a set speed that is a percentage of the robot speed, on default it is 100%. However, with RSI control, you specify how much the robot will move every 12ms. To be able to regulate the speed of the robot, a PID regulator will be used.

4.4.5 PID algorithm

To be able to control the robot effectively, a PID regulator is used. The mathematical formula for the PID regulator [53] is shown in equation 1.

$$u(t) = K_p e(t) + \frac{K_p}{T_i}\int_{t_0}^{t} e(\tau)d\tau + K_p T_d \frac{de(t)}{dt} \tag{1}$$

Where $e(t)$ is the deviation(wanted value - real value) and K_p, T_i, T_d are constants that are chosen under the regulation.

The KUKA robot has six joints, and therefore, six PID regulators are used, one for each rotation of the robot. To calculate the deviation, you take the wanted value minus the real value of the robot as shown in equation 2.

$$\begin{aligned} e_1 &= W_1 - R_1 \\ e_2 &= W_2 - R_2 \\ e_3 &= W_3 - R_3 \\ e_4 &= W_4 - R_4 \\ e_5 &= W_5 - R_5 \\ e_6 &= W_6 - R_6 \end{aligned} \tag{2}$$

Where W_1 to W_6 are the wanted values while R_1 to R_6 is the current value of the robot joints and since we don't want to have a difference between the desired and present value of the rotation we will set e equal to 0.

Discrete implementation To be able to use the PID regulator the formula 1 has to be discretized. First, the P part of the regulator is discretized and then the I and D parts.

P-part The P-part produces a value that is proportional to the deviation, where K_p is a proportional amplification constant. The discrete P-part is then:

$$u_p(k) = K_p e(k) \tag{3}$$

I-part I-part is the integral of the deviation with regards to τ. To discretized the I-part, the trapezoidal rule is used [54]. The trapezoidal rule is used to calculate an approximation value for a particular integral, as shown in equation 4.

$$\int_b^a f(x)dx \approx (b-a)\left[\frac{f(a)+f(b)}{2}\right] \tag{4}$$

In the equation 4 the $f(a)$ is set to be the current deviation and $f(b)$ is the previous deviation $e(K-1)$. The time $b-a$ is replaced with T_s which is the sample rate. Since the integral

is integrated from the start to the end, it is important to summarise the previous integral $u_i(k-1)$. Then the I part can be written as:

$$u_i = u_i(k-1) + \frac{K_p T_s}{T_i} \cdot \frac{e(k) + e(k-1)}{2} \tag{5}$$

It is essential to restrict the integral part (anti-windup), to avoid the I-part not to integrate itself up to a significant value. It can be done in the code by using two ”if” statements.

```
if(ui >= umax){
    ui(k) = umax;
}
if(ui <= umin){
    ui(k) = umin;
}
```

Where umax and umin are the limits, to avoid that the I-part becoming too big.

D-part The derivation of the discreet D-part can be approximated with the rate of ascent:

$$\frac{de(t)}{dt} = \frac{\triangle e}{\triangle t} = \frac{e(k) - e(k-1)}{t_2 - t_1} \tag{6}$$

The sample rate T_s mentioned above is every 12ms, it is therefore possible to assume that $t_2 - t_1$ is the same as the sampling rate. Then we can write:

$$u_d(k) = K_p T_d \frac{(e(k) - e(k-1))}{T_s} \tag{7}$$

The whole PID regulator When all the part of the PID regulator have been calculated, the parts are summed together, and we get the formula:

$$u(k) = u_p(k) + u_i(k) + u_d(k) = K_p e(k) + u_i + \frac{K_p T_s}{T_i} \cdot \frac{e(k) + e(k-1)}{2} + K_p T_d \frac{(e(k) - e(k-1))}{T_s} \tag{8}$$

There needs to be six PID regulators. Because the regulators are identical, the regulator will be put into a loop. This regulator runs every 12ms second for all the joints because the communication through RSI uses a 12ms loop.

Testing Visual Components control with RSI When the program was tested, only the P-part of the regulator was used, because, in this project, there was limited time with the robot. K_p was chosen to be 0.005, and the reason it is so low is to avoid that the robot moves to fast. There was also put in a limitation of ±0.055 degrees per 12 ms on how fast the joints could move. It means that the joint can rotate at a maximum speed of 4.58 degrees per second in each direction.

The computer that is currently connected to the KUKA controller is slow. When the program is running on the PC, it used 100% of the CPU. This is because the computer has an old Intel

Pentium CPU. A gaming laptop with an Intel i7-8750H CPU was borrowed from a fellow student, that is more than capable of running my programs.

The program was tested, and two tests where filmed and can be seen on Youtube and the code can be found in appendix B.2.5.:

- `https://www.youtube.com/watch?v=P7q6WmCTeL8`
- `https://www.youtube.com/watch?v=LuittMhbgwc&t=1s`

The robot did not move that smooth. A test was then done to investigate how Visual Components sends data. An OPC UA server was created with one variable. Then Visual Components connects to the server, and Visual Components is used to change the variables every second to see how constant the software is. As seen in figure 27, Visual Components sends data unevenly, with an average time of 0.985 sec, max time of 1.153 sec and min time of 0.823 sec. This delay makes it harder to tune the PID regulator and reduces the reaction time of the robot. Another problem is that by using Visual Components to control the robot, then the robot will always lag behind the Visual Components. It is better to use Visual Components to imitate the robot in real time to avoid the lag. It was, therefore, decided not to use Visual Components to control the robot.

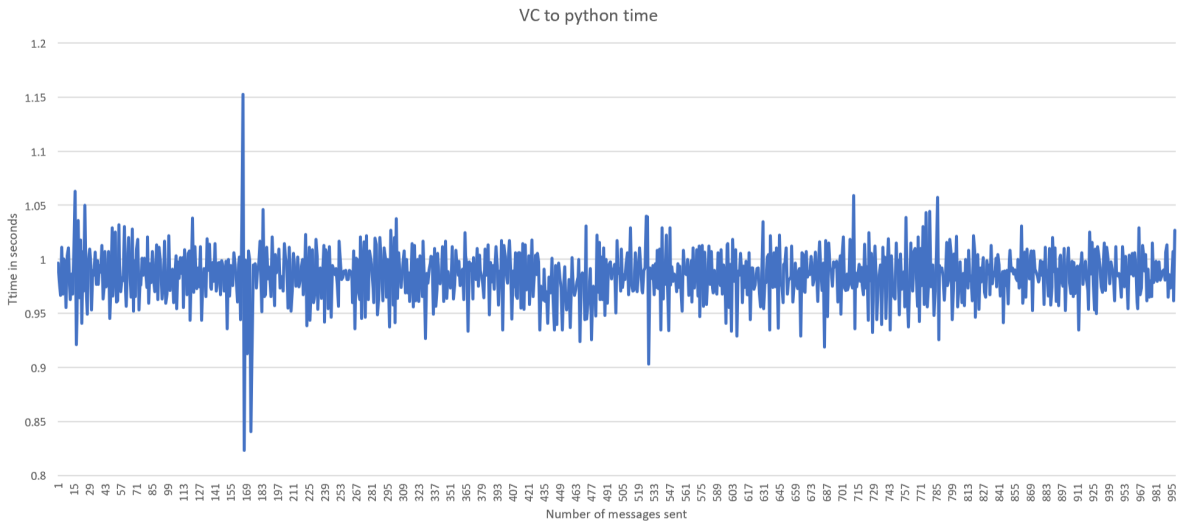


Figure 27: A graph of how Visual Components sends data

The next plan is to create a program that uses a python script to control the KUKA robot and also sends the current position of the robot to Visual Components, to get a visual representation of the robot while it is moving.

4.4.6 Python control with RSI

A program was made that used a python script to control the robot while Visual Components imitates the robot. This is done because Visual Components sends data to slow and not in a constant manner. Figure 28 illustrates how the program works.

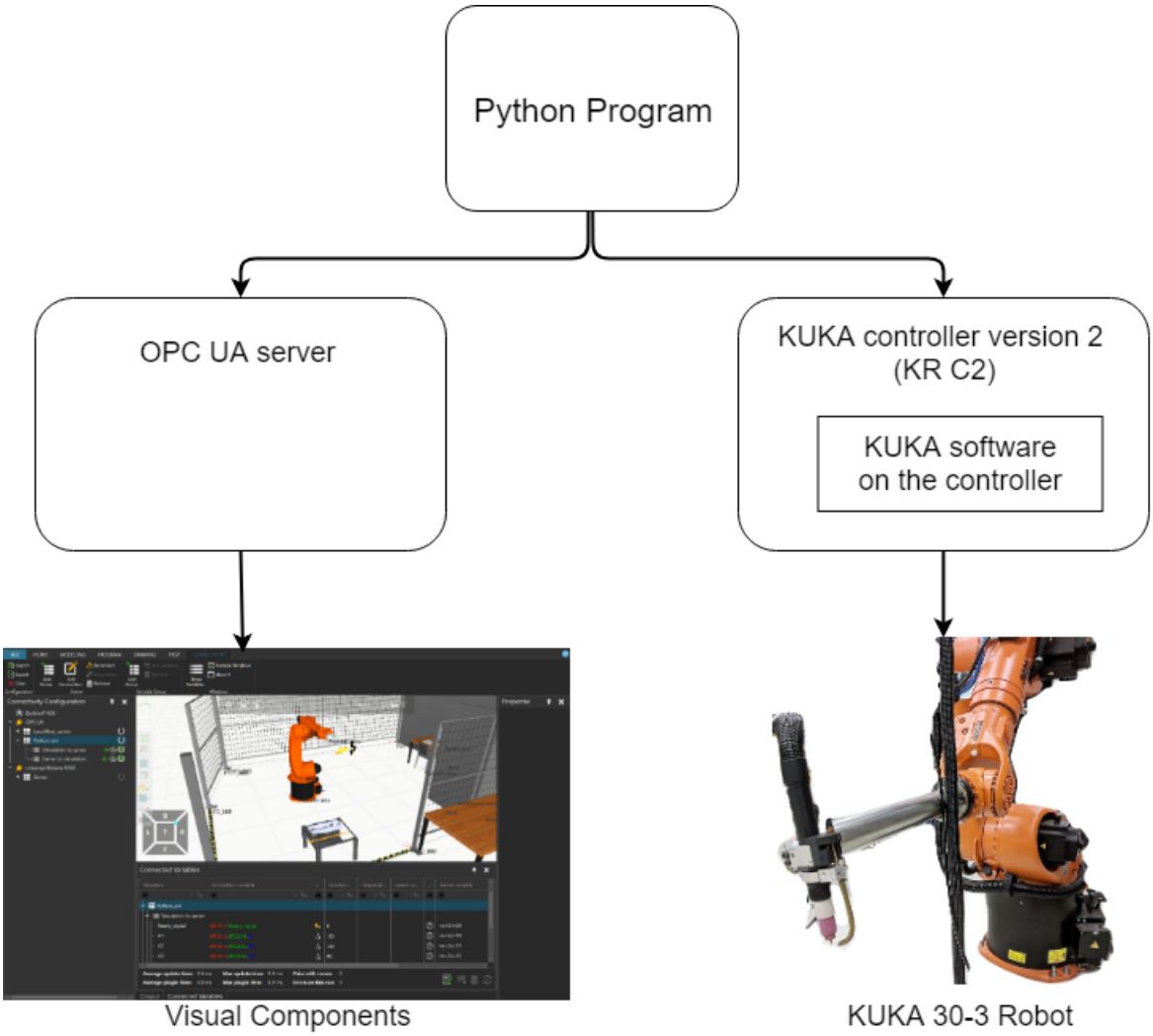


Figure 28: The figure illustrates how the program program works. Where A python script is used to send position data to Visual Components and the KUKA robot.

In the previous program there where three scripts that had to run simultaneously. It would be easier only to have one script to run instead of three. As mentioned before, the message received by the external computer has to be answered within 12ms. If the code uses a long time to run, it might not answer the XML message within 12ms and then the RSI communication would shut down. To make the code more stable while only having one script to run, the function multiprocessor is used. This method allows the user allocates a separate processor for a task/method and can make more than one method to run simultaneously.

To be able to share data between the processes, shared memory is used to share data. Either a single variable can be shared or an array with either a double or int data variables. In this program, three variables where created, one that had the current rotation of the robot, the wanted rotation of the robot and the difference between the wanted and current rotation.

There were created four separated processors, one for creating and updating the OPC UA server, RSI communication, PID-regulation, and one to read the current position of the robot.

A flowchart of how the program works can be seen in figure 29.

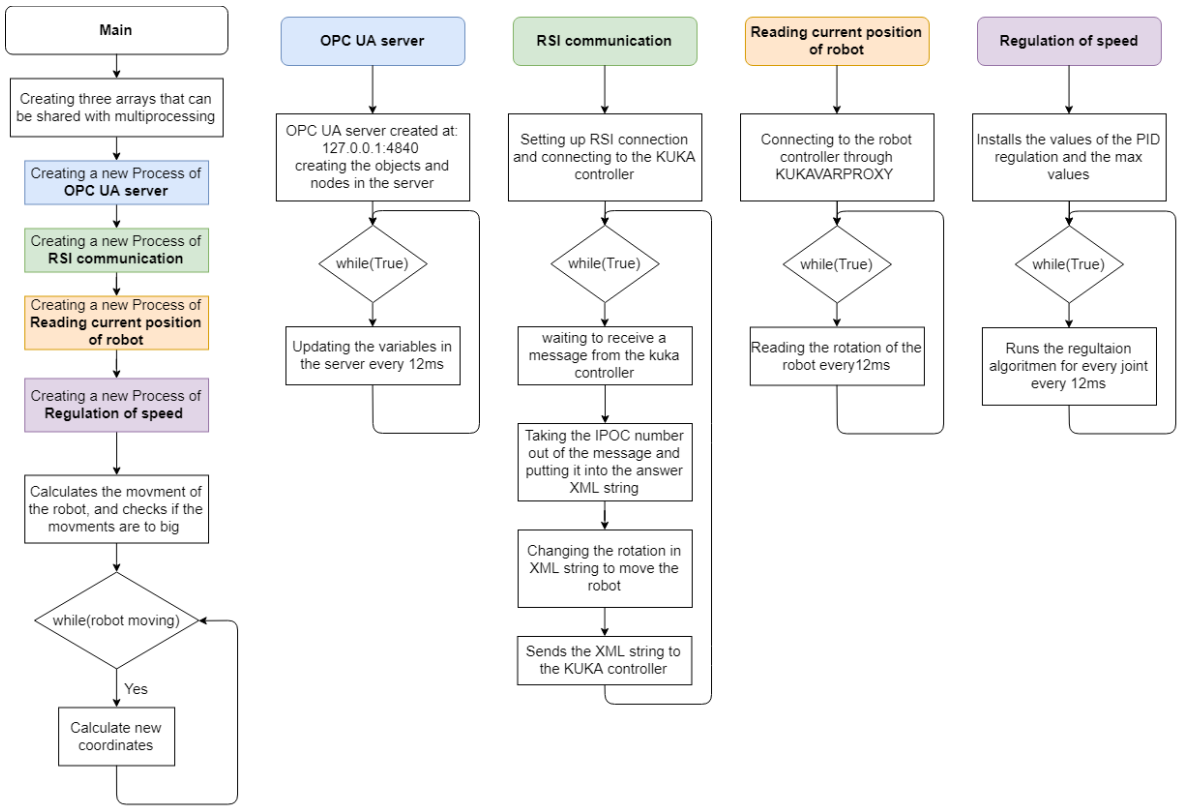


Figure 29: A simplified flowchart of how the python control program works.

The processor for RSI communication connects to the robot controller through RSI and then waits to receive a message from the KUKA controller. When a message is received from the controller, a message is sent back with the right IPOC number and change in the rotation.

As can be seen in the flowchart in figure 29, the regulation, reading current rotation of the robot, and updating the OPC UA server happens every 12ms. This is because the RSI uses the 12ms cycle for updates on where the robot should go.

When the robot is moving it starts and ends at a specific point which is specified in the code, as can be seen in figure 30. The number of steps it uses to get to the position can be edited by changing the variable num_of_updates. The code runs with the 12ms cycle, and new coordinates get calculated every 12ms. If the variable num_of_updates is sett to be 1000 the robot will use 12 seconds to move.

```
18
19      #-------------------------------------------------------------------------------
20      #Calculating the movment of the robot
21      start = [-89.99522, -90.00285, 89.99929, -0.0002927956, 0.0002801316, -0.001443268]
22      end = [-74.03281, -87.10163, 135.8689, -1.373797, -49.31733, -8.351775]
23      num_of_updates = 1000
```

Figure 30: Screen shot of the code in the python control program. The screen shot shows where you can edit the start point and endpoint of the robot when it moves.

The program was tested, and the code can be found in appendix B.2.6 and two Youtube videos were created of the tests:

- `https://www.youtube.com/watch?v=2fQ7irRJt2g`
- `https://www.youtube.com/watch?v=yvbaAGOpaHc`

From the test, the robot moved smoothly. Using python to control the robot while Visual Components is used to imitate the robot in real time is the best solution for creating a digital twin of the robot.

4.5 Improvements on Python control with RSI

When a stable and smooth control of the robot had been established. The next step is to improve the digital twin.

In the previous program, the data of the joint coordinates are collected with OpenShowVar-KUKAVARPROXY. As seen in figure 23, there are some long delays when OpenShowVar-KUKAVARPROXY is used, with a maximum time of 28.5579ms. The average time of 2.4715 is less than 12ms of RSI, but RSI is constant. Since new coordinates get updated every 12ms. Therefore the joint coordinates are instead taken from the RSI message. The idea is, when the computer has responded to the message with RSI, it takes the XML string received from the KUKA controller and saves the current rotation of the joints in the shared memory.

The OpenShowVar-KUKAVARPROXY was, however, used to collect information about the robot because the KUKAVARPROXY can be used to read variables. There is a manual [55] for the KUKA controller which contains all the variables of the robot and these can be read by using OpenShowVar-KUKAVARPROXY.

When making a digital twin, it is essential to collect as much information as possible to be able to create a replica of the physical model. OpenShowVar-KUKAVARPROXY was used to read the speed of the joints, temperature, speed of the fan in the controller and if the robot has a power failure. These values are then added to the OPC UA server and can be monitored by other computers that connect to the server. The program UaExpert [56] can be used to connect to the OPC UA server as a client and monitor the variables in the server. A screenshot of this program, when it is monitoring the OPC UA server can be seen in figure 31.

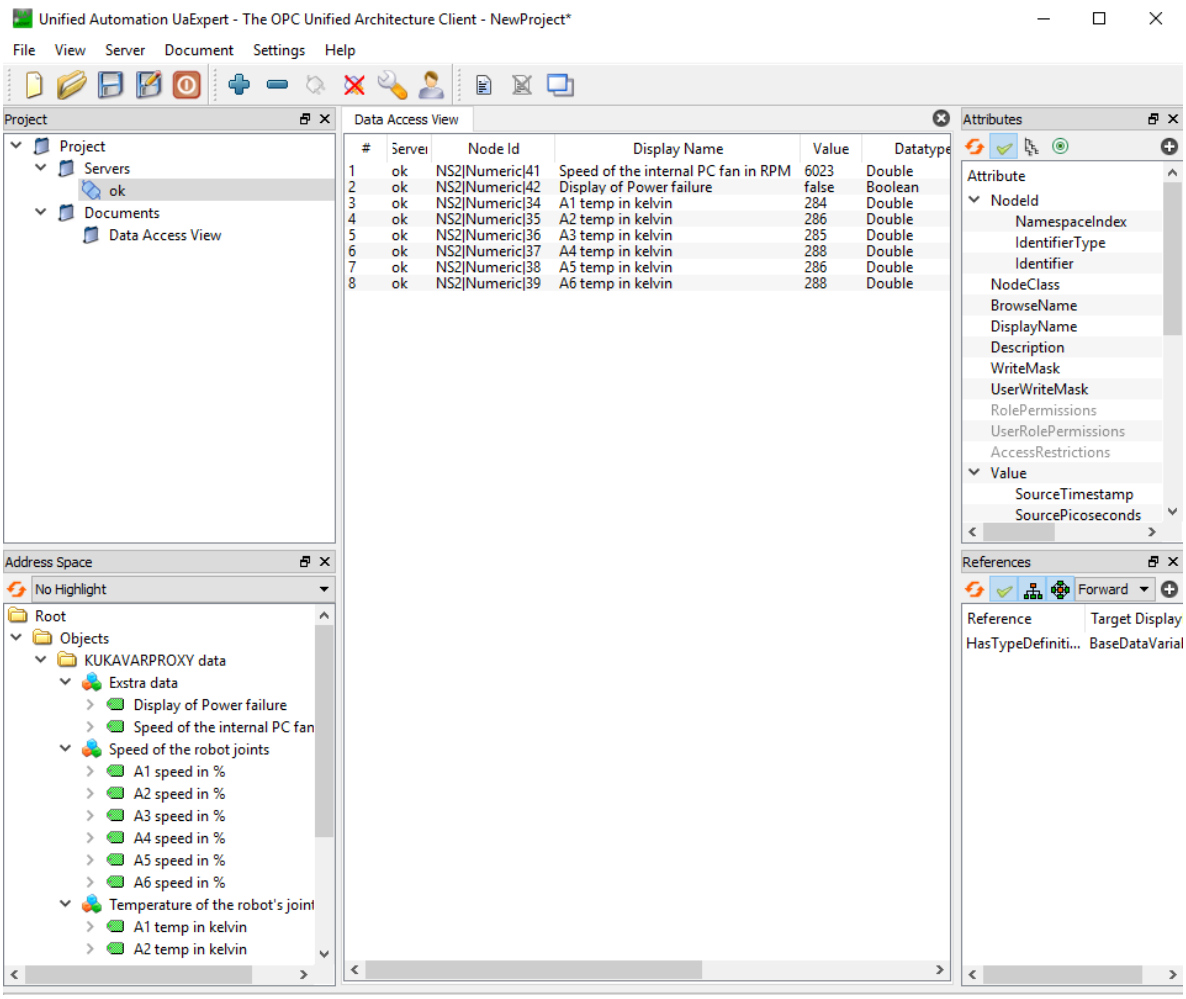


Figure 31: Screen shot of the UaExpert program that is monitoring the variables in the OPC UA server.

Even more, data could be added to the server, for this project, the speed, temperature, and speed of the fan was chosen as an example. The information in the digital twin should be dependent on what you're going to use the digital twin for.

To make the program more efficient, there were only used two processes instead of four in the previous version of the code. The regulation, find the next place for the robot to move and updating the OPC UA server could all be done within 12ms. However, reading the values from KUKAVARPROXY is too inconsistent and was, therefore, placed in its own process.

A flowchart illustrating how the program worked can be seen in figure 32. The code for the program can be found in appendix B.2.7.

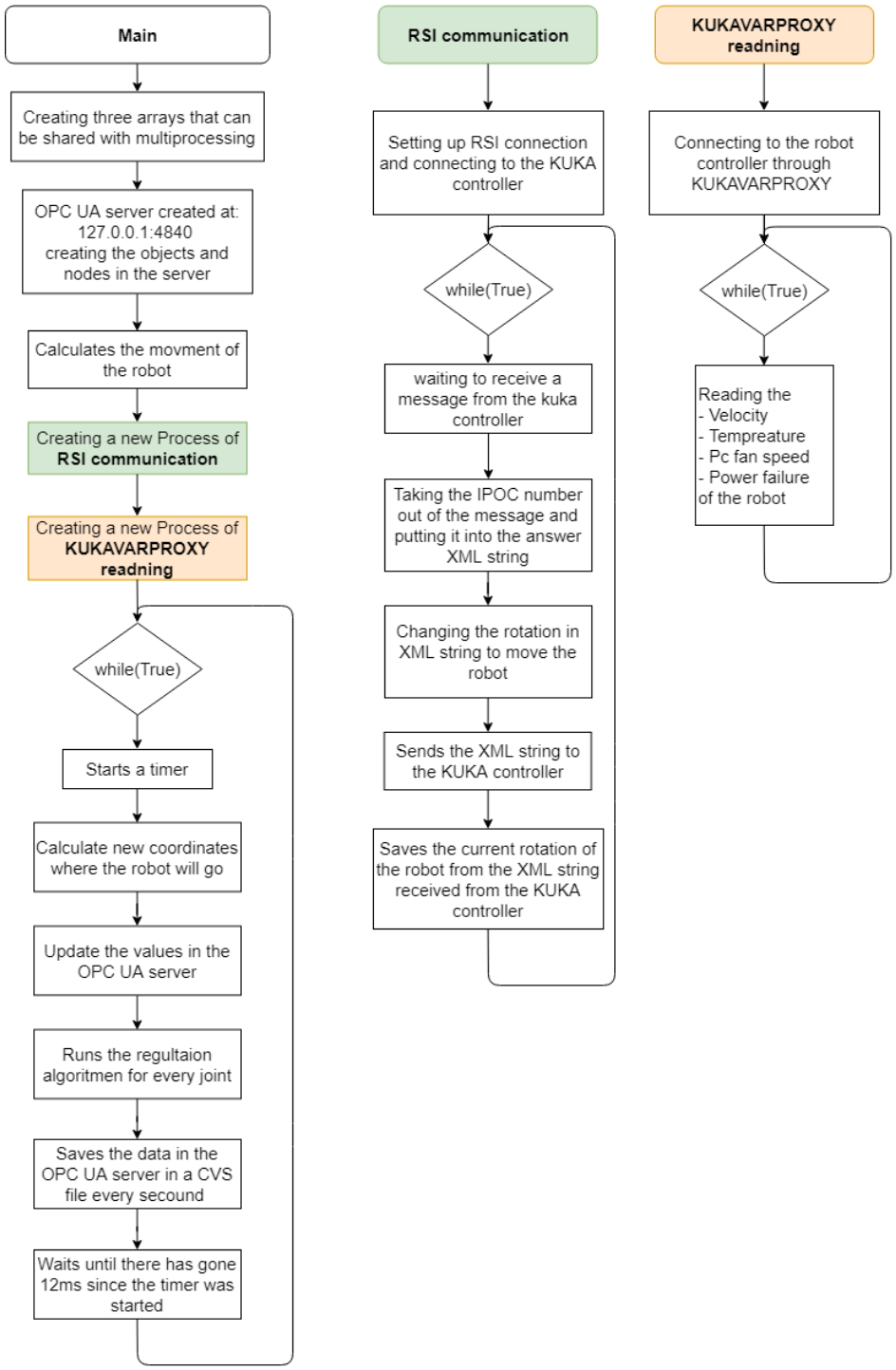


Figure 32: A flow diagram of how the improved version of the code works.

4.6 Improving the OPC UA server

The current OPC UA server doesn't have any security, that means that anyone can connect onto the OPC UA server and monitor and change variables in the server. All that is needed to connect is the URL of the server. As mentioned in chapter 2.3.2, the OPC UA server has integrated encryption mechanisms when sending data, and this chapter will look at what Cryptography is and how it can be implemented to the current OPC UA server.

4.6.1 Cryptography

Cryptography is a method to hide plain text, that uses mathematics to encrypt and decrypt data. It enables people to store and share information across the internet so that it can only be read by the people how the information is for.

There is a plain text that is encrypted to something that is unreadable gibberish so-called ciphertext. The process of transforming the ciphertext to the plain text is called decryption. A illustration of this process can be seen in figure 33 [57].

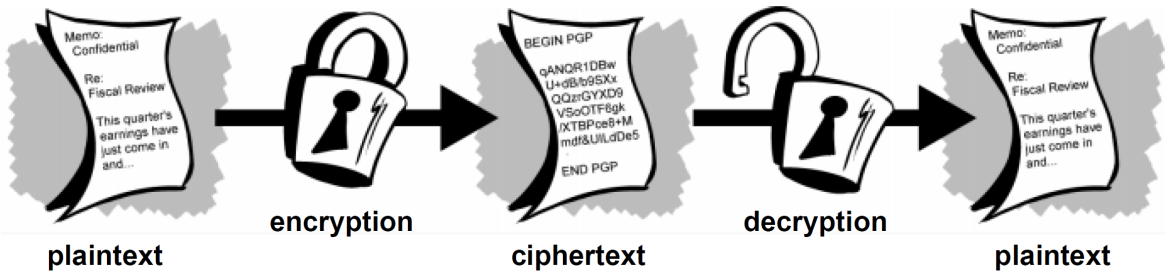


Figure 33: Shows the basic principle on how Cryptography works [57].

Keys Keys are values with a cryptographic algorithm to produce the ciphertext and are just massive numbers. The keys are measured in bits, where a 1024 bit key is enormous, and the bigger the key is, the more secure the ciphertext becomes [57].

Digital signature Digital signatures are used to enable the receiver of information to verify the origin and authenticity of the data. It is also used to verify that the information is intact. By using a public key and digital signature, you'll get authentication and data integrity. It provides the ability to detect if the sender is claiming that he or she did not receive the information [57].

A digital signature works in some ways like a handwritten signature. However, a digital signature is much harder to counterfeit than a handwritten signature. In some cases, a digital signature is used more then encryptions, for example when you're depositing $1000 in your bank account you want to be sure that you were dealing with the bank teller [57].

4.6.2 Security policy in the OPC UA server

Data security is tightly integrated into the architecture of the OPC UA [58].

The Github library used to create the OPC UA server has three security protocols, which are shown under:

- Basic128Rsa15
- Basic256
- Basic256Sha256

For each of the cryptography methods mentioned above, there are to security modes Sign and SignAndEncrypt. In sign, all the messages are signed, but with SignAndEncrypt, the messages are both signed and encrypted [59].

It is recommended that Basic256Sha256 cryptography algorithm is used as the minimum to secure the server [58]. From the OPC foundations website [60], it is noted that Basic128Rsa15 and Basic256 have become obsolete. They are *not considered secure anymore, this Security Policy has been deprecated with the OPC UA Specification Version 1.04* [61] [62].

To add cryptography to the OPC UA server, the encryption methods that are allowed in the server must be specified. Since only one of the methods is considered secure by the OPC Foundation, only Basic256Sha256 will be used. The security policy is set by using the method "set_security_policy" where you specify which methods are allowed to be used. Afterward, you load the certificate and the private key to the server.

The Basic256Sha256 encryption with sign and encrypt was used. This was tested by creating a client in python that connected to the server, and everything worked. The cryptography has been added to the "Planning AGV to the digital twin" program in appendix section B.3.1.

5 Adding an AGV to the digital twin

All the points that were mentioned in the scope in chapter 1.2 were finished before the deadline of the project. It was therefore suggested by my supervisor Wei Deng Solvang to include an Automated/Automatic Guided Vehicle (AGV).

5.1 AGV

AGV is a programmable mobile vehicle and is used in industrial applications to move materials and in manufacturing facilities. The AGV uses path selection and vision-based modes to become fully automated. An AGV system usually consists of [63]:

- **Vehicle:** Is the central element of an AGV to be able to perform transportation tasks. The AGV must be designed according to the environment it is going to be used in. Amazon uses AGVs to retrieve packages from storage while it can also be used for hotel serving or to improve health care management.
- **Guidance path system:** To be able to move the AGV automatically, it must have pathways. The path is chosen based on the programmed path.
- **Floor control and traffic management:** To operate the AGV efficiently, and increase the productivity of the AGV, the vehicle should be well managed. The time spent on waiting, loading, and unloading should be minimized. It can be done by using algorithms to control the traffic of the AGV.

5.1.1 Application of AGV

AGV can be used in many sectors. It is not limited to manufacturing and storage it can also be used in the service sector. Some examples of where AGVs are used can be seen under [63]:

- **Warehouse:** The AGV is used to transport materials into warehouses.
- **Commercial:** It can be used at airports for baggage transport, supermarkets, malls and as a vacuum cleaner at home.
- **Energy and defense:** The AGV can be used for mine and bomb mapping, where humans can't go. It can also be used for the disposal of nuclear material.
- **Medical service:** Can be used to deliver food, water, and medicine as well as take care of dangerous material.
- **Personal care:** Assist disabled or handicapped people.

5.2 Adding the AGV

The university is planning to buy a new an AGV, has not been chosen or purchased yet. After contacting the lab engineer in Narvik, the university is considering buying an AGV called TIAGo, as shown in figure 34.

Figure 34: The figure shows different configurations of the TIAGo, that the university is think about buying, from [64].

Since the AGV hasn't been bought yet, this chapter will go through how an AGV could be added to the digital twin that has been created.

5.2.1 Including AGV in the OPC UA server

RSI is used to control the robot, and OpenShowVar-KUKAVARPROXY is used to get additional information on the robot. Almost the same program will be used to control the robot as for the improved two-way digital twin.

When it comes to the AGV, a new folder is created inside the OPC UA server to store the position of the AGV, as well as additional data from the sensors in the AGV. It is also beneficial to have data of the speed and acceleration of the AGV. How the communication between the OPC UA server, KUKA robot, and AGV works can be seen in figure 35.

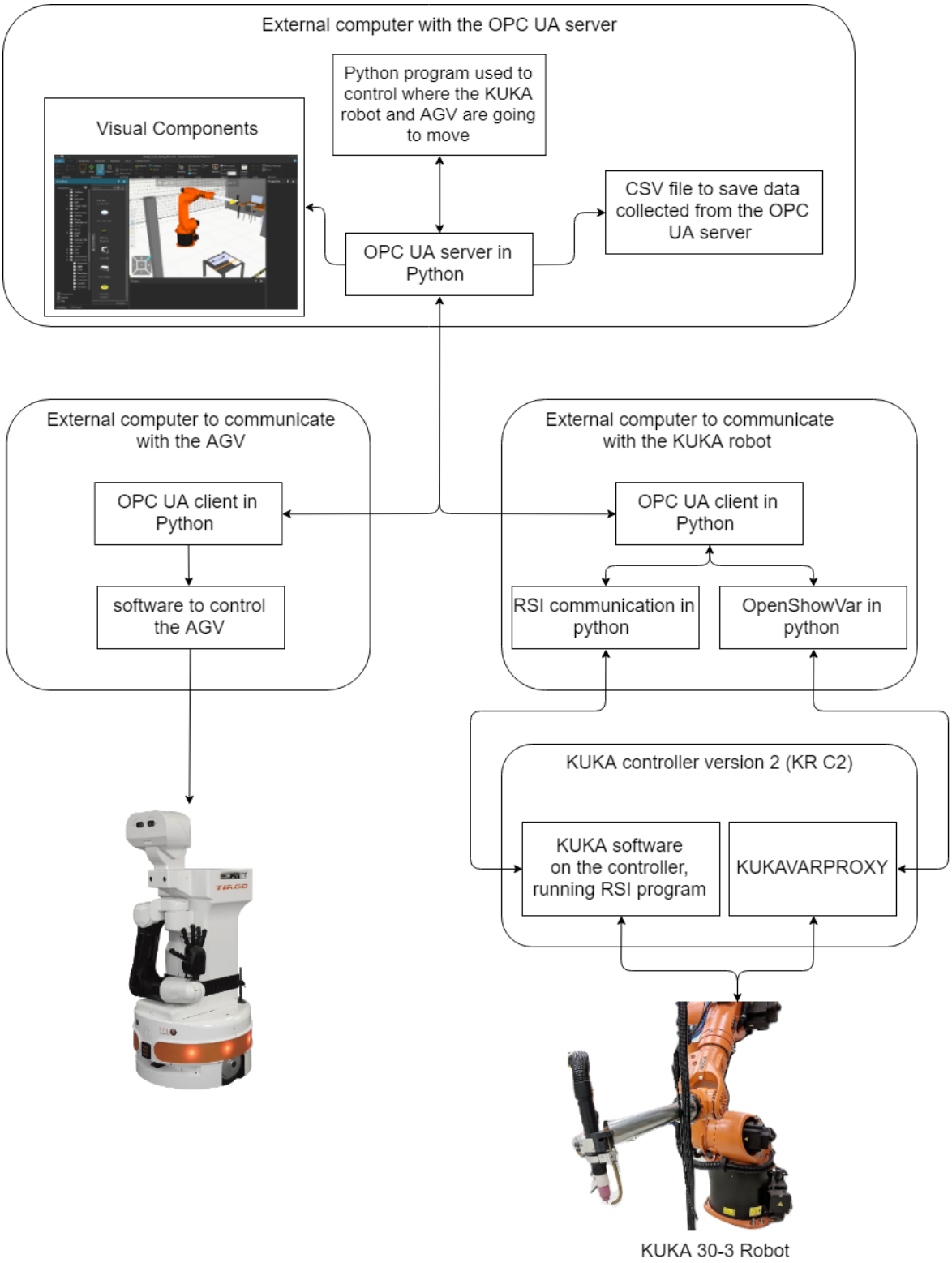


Figure 35: The diagram shows how the AGV can be added to the already created digital twin.

The idea is to have one python program to control both the AGV and the KUKA robot. This program will be placed in the same computer as the OPC UA server is in and will allow for easy collaboration since all the information about them is stored in the server. Visual Components is also placed in the computer with the OPC UA server. It can, be placed in another computer that connects to the OPC UA server as a client. The data collected from the KUKA robot and the AGV is saved in a .CSV file to have historical data, that can be used to optimize and analyze the system.

5.2.2 Adding an AGV to Visual Components

In Visual Components, there are already examples of AGVs, and they were used as a reference on how to add/create an AGV in Visual Components. These AGVs are controlled with vcSimVehicle.

vcSimVehicle In the help section in Visual Components, you can find the Python reference documentation, which contains all the information of the methods and objects that are in Visual Components Python API. One of the methods is vcSimVehicle and is used to move vehicles. To use the method, first, the speed, acceleration, and deceleration have to be defined. Afterward, the position you want to move to is specified in x, y and z coordinate system. The vehicle will then move to the position in a straight line from the starting point. It is also possible to rotate the vehicle in place, detach and attach wagons and lead or follow other vehicles.

For the sake of simplicity and since the university hasn't decided what AGV to buy, a simple blue rectangle was created in Visual Components to reflect an AGV, as seen in figure 36. A CAD model of the AGV can later replace this rectangle that the university will buy.

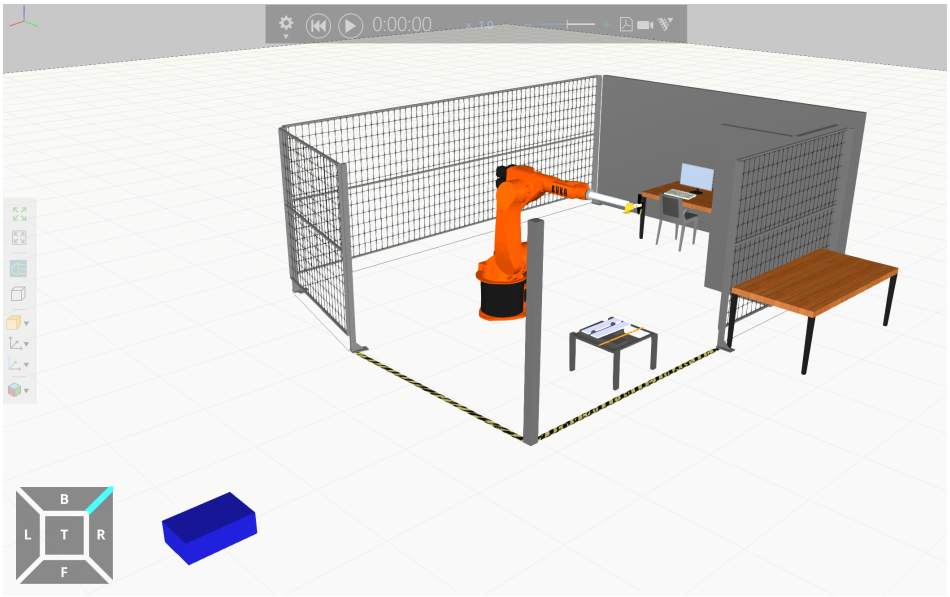


Figure 36: A screen shot of Visual Components, where the blue rectangle is used to represent a AGV.

5.2.3 Python code to control the AGV and KUKA robot

As mentioned in chapter 5.2.1, python code is used to control the AGV and the KUKA robot. The python program creates an OPC UA server and then waits until Visual Components connects to the server.

A point to point movement system is created to move both the AGV and KUKA robot. First, the points where the robot will move, and the speed is defined, as seen in figure 37. The RSI, which is used to control the KUKA robot, uses a 12ms update cycle and therefore the next movement of the robot will be calculated. The same 12ms cycle is used to control the AGV for simplicity sake and can be changed later. The steps of the robot and AGV is calculated and saved in an array. At last, if statements are used to control when the robot and AGV move and makes it possible for the AGV and KUKA robot to move simultaneously.

```
71      #-------------------------------------------------------------------------------
72      #Calculating the movment of the robot
73      point_start = [-90, -90, 90, 0, 0, 0]
74      point1 = [-74.03281, -87.10163, 135.8689, -1.373797, -49.31733, -8.351775, 600]
75      point2 = [-60.03281, -87.10163, 135.8689, -1.373797, -49.31733, -8.351775, 500]
76      point3 = [-80.03281, -87.10163, 135.8689, -1.373797, -49.31733, -8.351775, 250]
77      point4 = [-60.03281, -87.10163, 135.8689, -1.373797, -49.31733, -8.351775, 250]
78      # Can add more points where the KUKA robot will move here ...
79
80      all_points = [point_start, point1, point2, point3, point4]
81      calc_move = point_to_point(all_points, 6)
82      #-------------------------------------------------------------------------------
83      #-------------------------------------------------------------------------------
84      #Calculating the movment of the AGV
85      AVG_point_start = [5000, 2000, 0]
86      AVG_point1 = [2700, 2000, 0, 1000]
87      AVG_point2 = [2700, 3000, 0, 500]
88      AVG_point3 = [0, 3000, 0, 1000]
89      AVG_point4 = [300, 100, 0, 600]
90      # Can add more points where the AVG will move here ...
91
92      AVG_all_points = [AVG_point_start, AVG_point1, AVG_point2, AVG_point3]
93      AVG_calc_move = point_to_point(AVG_all_points, 3)
94      #-------------------------------------------------------------------------------
```

Figure 37: A screen shot of the python code used to control the AGV and KUKA robot. It shows where the points of the AGV and KUKA robot are defined.

To test the control program, Visual Components was used to see if the KUKA robot and AGV moved as planned. The KUKA and AGV start in their home position, as shown in figure 36. Then the AGV moves closer to the fence where the KUKA robot is while the KUKA robot moves away from its home position. When the AGV is right outside the fence, it waits until the KUKA robot has stopped moving and then drives under the KUKA robot and stops. The KUKA robot moves closer to the AGV, moves across the AGV, and then starts to move away from the AGV. In the end, the AGV starts driving back while the KUKA robot moves to its home position.

Two video recording of the simulation where made, one that uses the recording tool inside Visual Components and the other video records the whole screen of the computer. The code can be found in appendix B.3.1.

- **Visual Components recording**
  `https://www.youtube.com/watch?v=MPSw5PO2B5I`
- **Screen recording**
  `https://www.youtube.com/watch?v=GiW_2C9F4Hs&t=2s`

5.2.4 The next steps

The next steps for adding the AGV to the digital twin is to find out which AGV the university is going to buy. Since the plan is to control the AGV through the OPC UA server, it would be beneficial that the university buys an AGV that can either be controlled by python or another object-oriented programming language. This makes it possible to create a client that can connect to the OPC UA server. It would also be beneficial if the AGV could directly connect to the OPC UA server since OPC UA is a new standard it can be hard to find.

With this digital twin, it is easy to add other equipment to the digital twin and expand it. The use of OPC UA creates a central place to store and share information throughout the system.

6 Results

The main objective of the project was to create a digital twin of the KUKA KR 30-3 robot. The plan was to use Visual Components to get a visual representation of the KUKA robot. To enable communication between the robot and Visual Components, an OPC UA server was used.

The one-way digital twin can mirror the KUKA robot in Visual Components when it runs on controlled using the KR C2 controller. The OpenShowVar-KUKAVARPROXY that runs on the controller runs in the background and therefore doesn't interfere with the robot controller, and the controller can be used as normal.

The two-way digital twin is the opposite of the one-way digital twin, where the robot is controlled by the external computer connected to the robot controller. Controlling the robot with an external computer in a stable and controlled way proved to be more challenging than planned.

Nevertheless, a stable and constants two-way digital twin was created. The program was used RSI communication to control the robot, and OpenShowVar-KUKAVARPROXY is used to collect data to collect additional data about the robot. Visual Components is used to get a visual representation. However, only used to imitate the KUKA robot in real time and not to control the robot. The OPC UA server was used to communicate between the python program and Visual Components. Other computers that are on the same network can also connect to the OPC UA server to get information about the KUKA robot.

6.1 Adding an AGV to the digital twin

Planning on how a potential AGV could be added to the digital twin was done. The architecture of a digital twin with the KUKA robot and AGV was designed. A block was added to the visual model in Visual Components, which represents the AGV because the University hasn't decided on what AGV to buy. The block can be controlled by three variables (x, y and z position) in the OPC UA server. A better python program was also created to manage the KUKA robot and AGV, where you could define to which point they would move to and in what sequence they would move in.

6.2 Programs created

During the masters project, mainly three programs were created: one-way and two-way digital twin and adding an AGV to the digital twin. However, a lot of other programs were created in the process of enabling good control of the KUKA robot. All the programs created during the masters are listed below:

- **One way digital twin**
  - It uses OpenShowVar-KUKAVARPROXY to get joint values of the robot and an OPC UA server to enable communication between Visual Components and the external computer. Can be found in appendix B.1.1

- **Two way digital twin**
  - Two-way digital twin that uses OpenShowVar-KUKAVARPROXY to control the robot. Visual Components was used to control the robot with the OPC UA server to enable communication. Can be found in appendix B.2.1
  - A program that moves the robot from a starting point to an endpoint using OpenShowVar-KUKAVARPROXY. Can be found in appendix B.2.2
  - A program that uses the arrow keys to control the robot using OpenShowVar-KUKAVARPROXY. The A1 joint of the robot was controlled with the left and right arrow keys and the A2 joint with the up and down arrow keys. Can be found in appendix B.2.3
  - A C sharp code that can control the robot trough RSI communication. It can control the robot with either x, y and z coordinates or the by changing the rotation of the joints. Can be found in appendix B.2.4.
  - Controlling the robot with RSI communication, but with a python code instead of the C sharp code. It uses a Visual Components to tell the robot where to go and the OPC UA server to enable communication. Can be found in appendix B.2.5.
  - Program that uses a python code that moves the robot from a starting point to an endpoint using RSI communication. Visual Components is only used to visualize the robot and not to control it, and the task of the OPC UA server was to enable communication between Visual Components and the python code. Can be found in appendix B.2.6.
  - The last two-way digital twin improves in the previous version by using OpenShowVar-KUKAVARPROXY to collect more data on the robot and adding it to the OPC UA server. The last version also improves on the previous code and make it less demanding for the computer. Can be found in appendix B.2.7.
- **Planning and finding a good way to add a AGV to the digital twin**
  - Digital twin that contains the KUKA robot and an AGV. The AGV has been added to Visual Component and can be controlled through the same python script that controls the KUKA robot. This version allowed the user to choose several points for the AGV and KUKA robot to move to and in what sequence they will move towards the points. Can be found in appendix B.3.1.

All the programs that are listed above can be found in the appendix in section B.

7 Discussion

This chapter will discuss and analyse the results. It will look at OpenShowVar-KUKAVARPROXY, RSI communication, OPC UA, Visual Components, and what look at the benefits of the digital twin.

7.1 Controlling the KUKA robot

The purpose of the thesis was to see if it is possible to create a digital twin of with an older KUKA controller then has been done at NTNU [16]. The KR C2 controller used to control the KUKA robot is relatively old, which could make it hard to create a digital twin of the robot. Many manufacturers are not willing to publish internal details of the system architecture due to the competitive robot market [40]. This makes it hard to get full control over the robot and the system.

7.1.1 OpenShowVar and KUKAVARPROXY

During this master thesis, using OpenShowVar-KUKAVARPROXY to control the robot was very time consuming. The OpenShowVar-KUKAVARPROXY communication is for the most part based on a master thesis from NTNU [16].

A test was done of the read and write time of variables using OpenShowVar and KUKAVARPROXY. The variable AXIS_ACT which contains the current rotation of the joint of the robot was read 2000 times. The test was done three times. A variable was written 2000 times, and this was also done three times. The average time of these three tests is shown in table 3. The raw data can be found in appendix 4.4.5.

	Max	**Min**	**Average**
Read time	22.58ms	1.93ms	2.5ms
Write time	48.55ms	1.94ms	2.5ms

Table 3: The read and write time of OpenShowVar and KUKAVARPROXY.

From the tests, the average read and write time was approximately the same. Average time of 2.5ms is tolerant enough to create a good one-way digital twin. One of the problems when using it to control the KUKA robot was that the maximum write time is 48.55ms. The problem was that when you are controlling a robot, you want a constant and stable flow of data.
If a PID regulator is added to regulator the speed of the robot you will have to have a sample time as is mentioned in chapter 4.4.5.

Submit interpreter To be able to update the variable without the robot reaching the position a submit interpreter was created. The submit interpreter runs separately from the main KUKA program and might be able to update the position to the robot.

It was added but did not seem to affect the program. However the time I had to work on the submit interpreter was limited. The focus was, therefore, shifted over to RSI communication.

7.1.2 RSI communication

RSI communication compared to OpenShowVar-KUKAVARPROXY is much more constant, since it uses a 12ms cycle to receive and send messages. However, RSI communication is more unstable. In default, if you don't answer ten messages, the RSI communication will stop. This can be increased, but to make sure you have a stable and good code, you should have the breakpoint on the communication at 10 when the system is developed.

There is however a fast mode for the RSI communication which can enable a data send and receive time of 2 ms. The fast mode was not tested in this project because of limited time with the robot. It can enable a more responsive digital twin as well as better simulation of the robot.

As mentioned before, when RSI communication is set up, there is a default limit on how much the robot can move. It is important not to increase this limit too much when testing new code. In the user documentation of the RSI communication [52], it states that "*If used incorrectly, KUKA.RobotSensorInterface can cause personal injury and material damage*", and "*Unexpected movements may cause serious injuries and substantial material damage*". Therefore it is vital to be careful when using RSI communication.

Regulating the robot From the RSI HTML guide, it is possible to create an RSI object with P, PD, I, D or a PID regulator. There is also the possibility to add filters and other add-ons. Because of time limitation with the KUKA robot, these add-ons were not tested, but maybe the PID regulator in the RSI object could replace the PID regulator in Python.

The PID regulator in Python used to regulate the speed of the robot, has not been tuned because of time constraints. When the testing was done only a P regulator was used. The I and D part of the regulator should also be added and regulated to give a more smooth movement and stopping of the robot.

All the data of the joint values of the robot was saved in the OPC UA server and the values of the P, I and D parts can be added to the OPC UA server. Then a python program can be created that connects to the OPC UA server as a client and plots the values of P, I and D parts. This will allow for live regulation of the robot and make it easier to tune the values after the requirements of the system.

7.2 OPC UA server

The OPC UA server has, for the most part, been used to connect the python program that communicates with the robot and Visual Components. Using OPC UA to communicate with Visual Components creates a rather simple solution.

When the AGV was added to the digital twin, the real benefit of using the OPC UA server comes to light. By having all the data of the manufacturing equipment in one server, makes

it easy to add robots, AGVs and other machines to the digital twin. Other computers can connect to the server and monitor the data in the server.

The python library from Github used to create the OPC UA server does not include the user and password security model [47]. It is, however, possible to use cryptography, which can provide a suitable and secure method to secure the data in the server. There are three methods of cryptography in the python library. Nevertheless, two of the method's are considered obsolete and not secure by the OPC foundation[61].

7.2.1 Visual Components as a Visual tool

Visual Components is useful for offline application, and not so stable when it comes to real time. The problem is that there is to much delay and lag in the program, as mentioned in chapter 4.4.4. Another problem is that when a while loop in Python is used, there has to be a delay in the loop. If not the Visual Components program crashes and you loos all the work that hasn't been saved.

This digital twin was made with Visual Components 4.1. However future versions of the program might fix these issues. There are good and easy ways to control a robot or AGV in Visual Components and using Visual Components to control the robot and AGV can be beneficial. The program has to be updated before this happens.

7.3 Benefits from a digital twin

There were a couple of benefits discovered through the project, which are listed below:

- **Offline testings**
  One of the biggest advantages of having the digital copy of a physical system is to have the ability to test the programs offline. Trough this project when a new program was created, it was always tested offline using Visual Components to see what happens when the program runs. It provides an extra layer of safety and if there are any movements the will collide with something in the laboratory they are easy to spot.

  One example of using the digital twin to test offline is shown in chapter 7.4.

- **Monitoring of the equipment/system**
  Using the OPC UA server, computers can connect onto the server and monitor the system. This again makes it easier to monitor the system from a distance.

- **Historical data**
  Data collected from the system can be stored and analyzed to optimize the system. It can also be used to find errors in a system. When you have a recording and the data about the robot, it can be easier to find the fault.

- **What if analysis**
  One of the best benefits of having a digital twin is to have the ability to test scenarios offline while the robot is not connected.

- **Autonomous system**
  The digital twin created is a simple autonomous system. More complex code can be developed for controlling the robot and AGV, and a completely autonomous system can be created.

- **Human robot collaboration**
  If more sensors in the laboratory were added, the digital twin could replace the current fence, which is being used and can allow for a more open system.

7.4 Example of offline testing of the digital twin

One example of using the digital twin for offline testing is from a fellow student who how was writing his master's thesis at the same time and used the KUKA robot for robot welding. As can be seen in the top of figure 38, there is a welding table in front of the robot. Usually, the table is in front of the robot. The problem is that the robot can't weld in all directions with the current placement of the table.

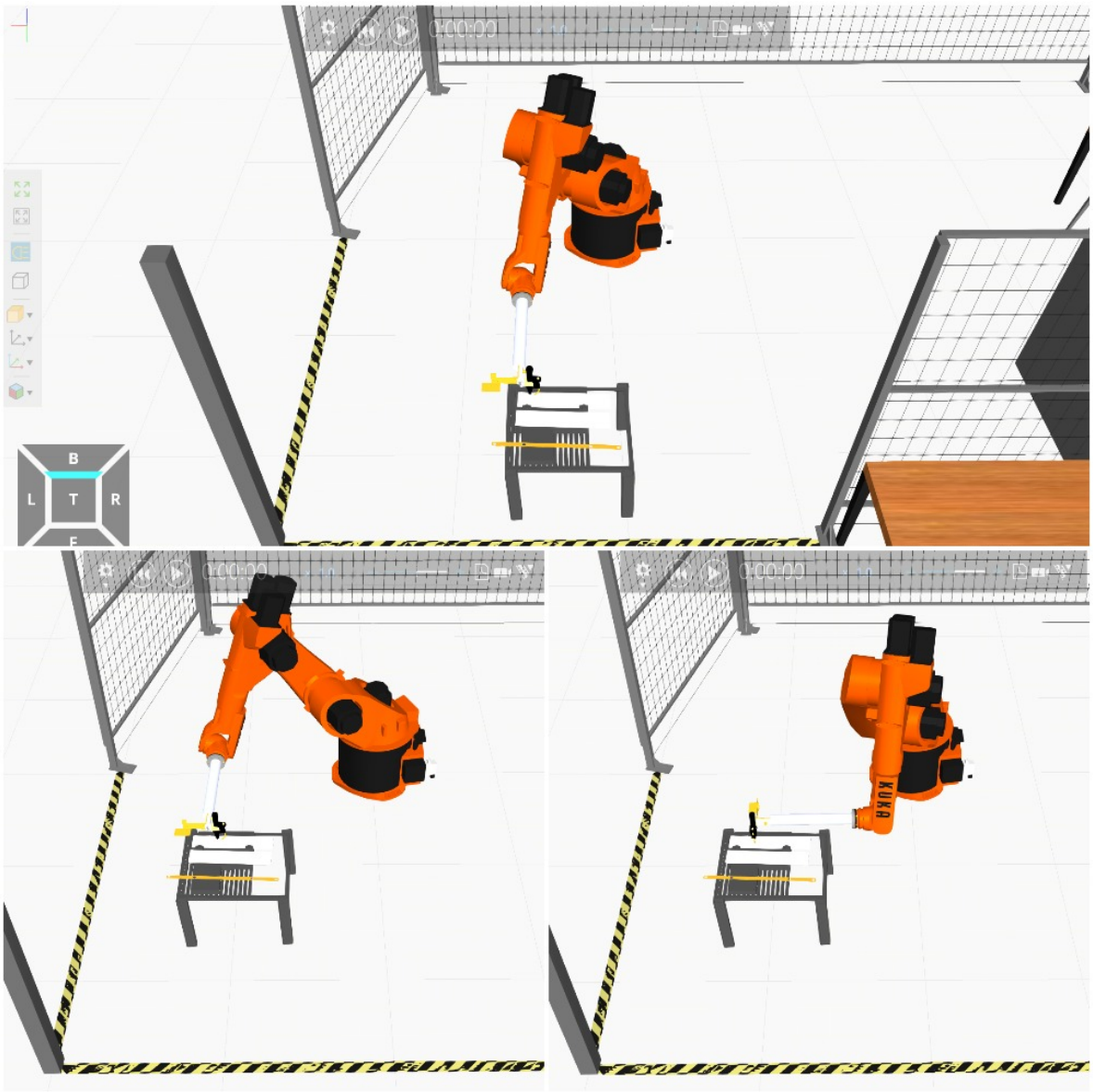


Figure 38: Screen shot of the digital twin model in Visual Components. The top picture is where the welding table was originally placed, while the two pictures on bottom show the two different welding position when a plus is welded.

To be able to weld in a cross and a plus, as is shown in figure 39, the table had to be moved. To find out at where to place the table so that the robot could weld in the two directions shown in figure 38, the digital twin was used to find the placement of the welding table and if the robot could reach the required position.

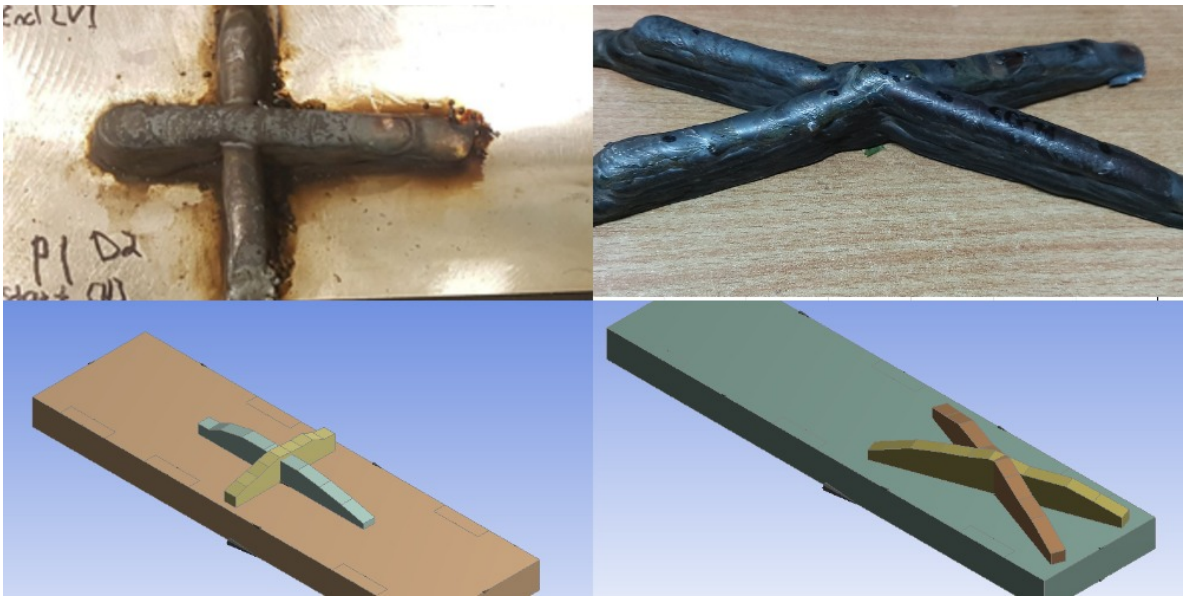


Figure 39: Welding design from the master thesis ”Thin Wall Structure by Welding” by Hans Ivar Arumairasa.

8 Conclusion

Most of the work done throughout this thesis revolves around creating a stable and proper method of controlling the KUKA robot. The conclusion is that using OpenShowVar and KUKAVARPROXY to control the robot was not the right solution, because the robot movement is unsmooth, and there is a long delay. However, the KUKA controller has the RSI add-on, which allows for real-time control of the robot. The RSI proved to be an excellent solution for controlling the robot.

One of the scopes for the master thesis was to find ways to control the robot in Visual Components. It was found that Visual Components sends data unevenly, and there is to much lag. Another problem is that Visual Components sends the joint coordinates to the KUKA robot, which means that the KUKA robot will always be behind the Visual Components program. The conclusion is that using Visual Components to control the robot is not a good solution. Therefore, Visual Components is only used to imitate the movement of the robot in real time.

The OPC UA server is an excellent solution for connecting the simulation model. It makes it easy to expand the digital, and it was easy to add the AGV to the digital twin that had already been created.

The main objective of the thesis is to see if it is possible to create a digital twin of the KUKA KR 30-3 robot with the KR C2 controller. It was found that creating a digital twin of the KUKA robot with the KR C2 controller using Visual Components for simulation and the OPC UA server for communication is possible. The digital twin that was created can simulate the robot in real-time and collect data about the robot in the OPC UA server.

8.1 Further work

As mentioned in chapter 7.1.2, the RSI regulation should be tested and compared to the current regulation algorithm I python currently used. To improved the robot movement with RSI communication, the I and D part of the regulator should be included and turned to improve the robot movement. Also tuning of the PID regulator has to be done for a more better movement of the robot.

The fast mode for RSI communication should be used. This would give a quicker and more responsive control of the robot. However, it can be hard to develop because the external computer has to be able to answer the message within 2ms instead of 12ms. It can create a more unstable code.

A more sophisticated program to control the KUKA robot and AGV can be developed. The program could use a GUI where you could write in the coordinates to where you want the robot or AGV to move and at what speed.
The plan and design of architecture on how an AGV could be added has been developed. A block has been created in Visual Components to represent the AGV and is able to move it through the OPC UA server. For further work a CAD model of the AGV that the university buys, to replace the block. The parts that are left can be seen in figure 40.

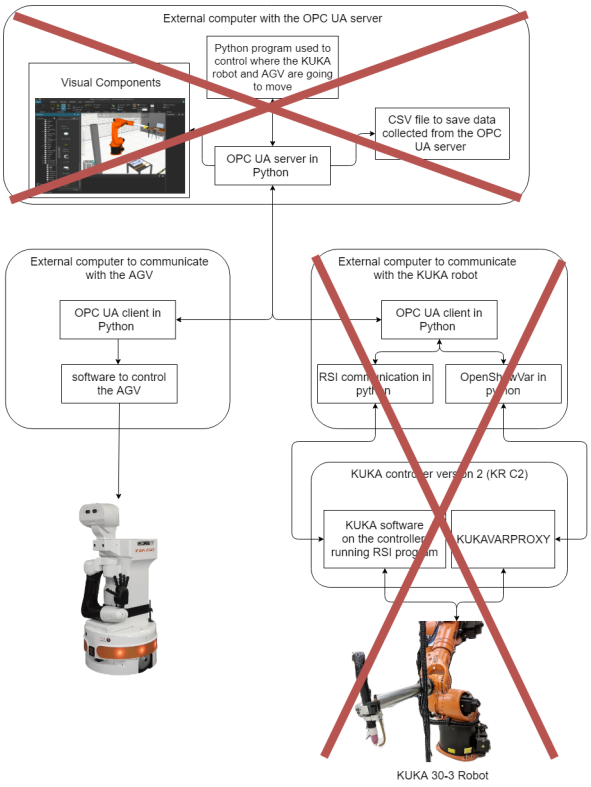


Figure 40: The figure illustrates what part is left of the AGV add on. The regions with a red x, is finished.

References

[1] J. Guo et al. "Modular based flexible digital twin for factory design". In: *Journal of Ambient Intelligence and Humanized Computing* (2018). ISSN: 18685137.

[2] Torbjørn F. Folgerø. *Establishing a Digital Centre of Excellence*. URL: https://www.equinor.com/en/magazine/statoil-2030---putting-on-digital-bionic-boots.html. (accessed: 10.12.2018).

[3] Christy Pettey. *Prepare for the Impact of Digital Twins*. URL: https://www.gartner.com/smarterwithgartner/prepare-for-the-impact-of-digital-twins/. (accessed: 05.27.2019).

[4] Kagermann Henning. "Recommendations for implementing the strategic initiative INDUSTRIE 4.0". In: (Apr. 2013). URL: http://thuvienso.dastic.vn:801/dspace/handle/TTKHCNDaNang_123456789/357.

[5] Mohd Aiman Kamarul Bahrin et al. "Industry 4.0: A review on industrial automation and robotic". In: *Jurnal Teknologi* 78 (June 2016). DOI: 10.11113/jt.v78.9285.

[6] Christoph Roser. *How Industry 4.0 Came into Existence*. URL: https://www.allaboutlean.com/industry-4-0/. (accessed: 31.10.2018).

[7] S. Boschert and R. Rosen. "Digital twin-the simulation aspect". In: *Mechatronic Futures: Challenges and Solutions for Mechatronic Systems and Their Designers*. Springer International Publishing, 2016, pp. 59–74. ISBN: 9783319321561.

[8] Roland Rosen et al. "About The Importance of Autonomy and Digital Twins for the Future of Manufacturing". In: *IFAC-PapersOnLine* 48.3 (2015). 15th IFAC Symposium onInformation Control Problems inManufacturing, pp. 567–572. ISSN: 2405-8963. DOI: https://doi.org/10.1016/j.ifacol.2015.06.141. URL: http://www.sciencedirect.com/science/article/pii/S2405896315003808.

[9] Michael Grieves. "Digital Twin: Manufacturing Excellence through Virtual Factory Replication". In: (Mar. 2015).

[10] D. Preuveneers, W. Joosen, and E. Ilie-Zudor. "Robust Digital Twin Compositions for Industry 4.0 Smart Manufacturing Systems". In: *2018 IEEE 22nd International Enterprise Distributed Object Computing Workshop (EDOCW)*. Oct. 2018, pp. 69–78. DOI: 10.1109/EDOCW.2018.00021.

[11] GE Digital. *What is a digital twin?* URL: https://www.ge.com/digital/applications/digital-twin. (accessed: 04.01.2019).

[12] SIEMENS. *Digital Twin*. URL: https://www.plm.automation.siemens.com/global/en/our-story/glossary/digital-twin/24465. (accessed: 04.01.2019).

[13] ABB. *Digital twin applications*. URL: https://new.abb.com/control-systems/features/digital-twin-applications. (accessed: 04.01.2019).

[14] IBM. *Digital twin: Helping machines tell their story*. URL: https://www.ibm.com/internet-of-things/trending/digital-twin. (accessed: 04.01.2019).

[15] Alasdair Gilchrist. *Industry 4.0: The Industrial Internet of Things*. eng. Berkeley, CA: Apress, 2016. ISBN: 9781484220467.

[16] Aksel Øvern. "Industry 4.0 - Digital Twins and OPC UA". 2018. URL: http://hdl.handle.net/11250/2561319.

[17] Fei Tao et al. "Digital twin-driven product design, manufacturing and service with big data". eng. In: *The International Journal of Advanced Manufacturing Technology* 94.9 (2018), pp. 3563–3576. ISSN: 0268-3768.

[18] Brian Scassellati and Bradley Hayes. "Human-robot collaboration". In: *AI Matters* 1 (Dec. 2014), pp. 22–23. DOI: 10.1145/2685328.2685335.

[19] Klaus Dröder et al. "A Machine Learning-Enhanced Digital Twin Approach for Human-Robot-Collaboration". In: *Procedia CIRP* 76 (2018). 7th CIRP Conference on Assembly Technologies and Systems (CATS 2018), pp. 187–192. ISSN: 2212-8271. DOI: https://doi.org/10.1016/j.procir.2018.02.010. URL: http://www.sciencedirect.com/science/article/pii/S2212827118300295.

[20] Ali Ahmad Malik and Arne Bilberg. "Digital twins of human robot collaboration in a production setting". In: *Procedia Manufacturing* 17 (2018). 28th International Conference on Flexible Automation and Intelligent Manufacturing (FAIM2018), June 11-14, 2018, Columbus, OH, USAGlobal Integration of Intelligent Manufacturing and Smart Industry for Good of Humanity, pp. 278–285. ISSN: 2351-9789. DOI: https://doi.org/10.1016/j.promfg.2018.10.047. URL: http://www.sciencedirect.com/science/article/pii/S2351978918311636.

[21] Vladimir Kuts et al. "Exploiting Factory Telemetry to Support Virtual Reality Simulation in Robotics Cell". In: *Augmented Reality, Virtual Reality, and Computer Graphics*. Ed. by Lucio Tommaso De Paolis, Patrick Bourdot, and Antonio Mongelli. Cham: Springer International Publishing, 2017, pp. 212–221. ISBN: 978-3-319-60922-5.

[22] F. Tao et al. "Digital Twin in Industry: State-of-the-Art". In: *IEEE Transactions on Industrial Informatics* (2018), pp. 1–1. ISSN: 1551-3203. DOI: 10.1109/TII.2018.2873186.

[23] gartner. *Gartner equips executives across the enterprise to make the right decisions and stay ahead of change*. URL: https://www.gartner.com/en/about. (accessed: 10.01.2019).

[24] Kasey Panetta. *5 Trends Emerge in the Gartner Hype Cycle for Emerging Technologies, 2018*. URL: https://www.gartner.com/smarterwithgartner/5-trends-emerge-in-gartner-hype-cycle-for-emerging-technologies-2018/. (accessed: 08.01.2019).

[25] Hermann Kopetz. "Internet of Things". In: *Real-Time Systems: Design Principles for Distributed Embedded Applications*. Boston, MA: Springer US, 2011, pp. 307–323. ISBN: 978-1-4419-8237-7. DOI: 10.1007/978-1-4419-8237-7_13. URL: https://doi.org/10.1007/978-1-4419-8237-7_13.

[26] J. Imtiaz and J. Jasperneite. "Scalability of OPC-UA down to the chip level enables "Internet of Things"". In: *2013 11th IEEE International Conference on Industrial Informatics (INDIN)*. July 2013, pp. 500–505. DOI: 10.1109/INDIN.2013.6622935.

[27] Sebastian Lehnhoff et al. “OPC Unified Architecture: A Service-oriented Architecture for Smart Grids”. In: (June 2012). DOI: 10.1109/SE4SG.2012.6225723.

[28] Stefan Hoppe. *Classic*. URL: https://opcfoundation.org/about/opc-technologies/opc-classic/. (accessed: 22.11.2018).

[29] www.opcfoundation.org. *OPC Unified Architecture Interoperability for Industrie 4.0 and the Internet of Things*. URL: https://opcfoundation.org/wp-content/uploads/2017/11/OPC-UA-Interoperability-For-Industrie4-and-IoT-EN.pdf. (accessed: 22.11.2018).

[30] Stefan Hoppe. *There is no Industrie 4.0 without OPC UA*. URL: https://opcconnect.opcfoundation.org/2017/06/there-is-no-industrie-4-0-without-opc-ua/. (accessed: 22.11.2018).

[31] *Advances in Practical Applications of Cyber-Physical Multi-Agent Systems: The PAAMS Collection : 15th International Conference, PAAMS 2017, Porto, Portugal, June 21-23, 2017, Proceedings*. eng. Cham, 2017.

[32] Stefan-Helmut Leitner and Wolfgang Mahnke. “OPC UA - Service-oriented Architecture for Industrial Applications”. In: *Softwaretechnik-Trends* 26 (2006).

[33] Dare Obasanjo. *Understanding XML*. URL: https://msdn.microsoft.com/en-us/library/aa468558.aspx. (accessed: 10.12.2018).

[34] A. Braune, S. Hennig, and S. Hegler. “Evaluation of OPC UA secure communication in web browser applications”. In: *2008 6th IEEE International Conference on Industrial Informatics*. July 2008, pp. 1660–1665. DOI: 10.1109/INDIN.2008.4618370.

[35] tutorialspoint. *SOAP - Message Structure*. URL: https://www.tutorialspoint.com/soap/soap_message_structure.htm. (accessed: 10.12.2018).

[36] Visual Components. *OUR STORY*. URL: https://www.visualcomponents.com/about-us/. (accessed: 07.11.2018).

[37] Visual Components. *ESSENTIALS*. URL: https://www.visualcomponents.com/products/visual-components-4-0/essentials/. (accessed: 07.11.2018).

[38] Visual Components. *VISUAL COMPONENTS 4.1*. URL: https://www.visualcomponents.com/products/visual-components-4-0/. (accessed: 07.11.2018).

[39] KUKA. *KR 30*. URL: https://www.kuka.com/en-de/products/robot-systems/industrial-robots/kr-30. (accessed: 04.12.2018).

[40] F. Sanfilippo et al. “Controlling Kuka Industrial Robots: Flexible Communication Interface JOpenShowVar”. In: *IEEE Robotics Automation Magazine* 22.4 (Dec. 2015), pp. 96–109. ISSN: 1070-9932. DOI: 10.1109/MRA.2015.2482839.

[41] F. Sanfilippo et al. “JOpenShowVar: An open-source cross-platform communication interface to Kuka robots”. In: *2014 IEEE International Conference on Information and Automation (ICIA)*. July 2014, pp. 1154–1159. DOI: 10.1109/ICInfA.2014.6932823.

[42] Margaret Rouse. *TCP/IP (Transmission Control Protocol/Internet Protocol)*. URL: https://searchnetworking.techtarget.com/definition/TCP-IP. (accessed: 05.12.2018).

[43] Filippo Sanfilippo. "Alternative and Flexible Control Approaches for Robotic Manipulators: on the Challenge of Developing a Flexible Control Architecture that Allows for Controlling Different Manipulators". PhD thesis. June 2015. DOI: 10.13140/RG.2.1.2775.1762. URL: http://hdl.handle.net/11250/2360280.

[44] Katrin Stuber-Koeppe. *KUKA invests in the factory of the future*. URL: https://www.kuka.com/en-de/press/news/2017/12/kuka-akquiriert-visual-components. (accessed: 12.11.2018).

[45] Ahmad Saeed. *A class that can send and receive messages from KUKAVARPROXY throught Python or CPP*. URL: https://github:com/ahmad-saeed/kukavarproxy-msg-format. (accessed: 16.12.2018).

[46] Google LLC. *Measure - Quick Everyday Measurements*. URL: https://play.google.com/store/apps/details?id=com.google.tango.measure&hl=en. (accessed: 08.02.2019).

[47] Thilo Cestonaro. *python-opcua*. URL: https://github.com/FreeOpcUa/python-opcua. (accessed: 24.05.2019).

[48] Visual Components. *Connect a Remote OPC UA Server*. URL: http://academy.visualcomponents.com/lessons/connect-a-remote-opc-ua-server/. (accessed: 21.11.2018).

[49] aauc-mechlab. *JOpenShowVar*. URL: https://github.com/aauc-mechlab/JOpenShowVar. (accessed: 10.01.2019).

[50] KUKA Roboter GmbH. *SOFTWARE KR C2 / KR C3 Expert Programming KUKA System Software (KSS) Release 5.2*. Aug. 2013.

[51] KUKA Roboter GmbH. *KR C Submit interpreter Operation and Programming Release 1*. May 2005.

[52] KUKA Roboter GmbH. *KUKA.RobotSensorInterface 2.3*. May 2009.

[53] Michael M. W. de Silva. *Discrete PI and PID Controller Design and Analysis for Digital Implementation*. URL: https://www.scribd.com/doc/19070283/Discrete-PI-and-PID-Controller-Design-and-Analysis-for-Digital-Implementation. (accessed: 17.02.2019).

[54] www.mathwords.com. *Trapezoid Rule*. URL: https://www.mathwords.com/t/trapezoid_rule.htm. (accessed: 29.05.2019).

[55] KUKA Roboter GmbH. *System Variables For KUKA System Software 8.1, 8.2 and 8.3*. Aug. 2012.

[56] Unified Automation GmbH. *OPC UA Clients – Downloads*. URL: https://www.unified-automation.com/downloads/opc-ua-clients.html. (accessed: 20.03.2019).

[57] Inc. Network Associates. *An Introduction to Cryptography*. URL: https://courses.cs.vt.edu/cs5204/fall09-kafura/Papers/Security/IntroToCryptography.pdf. (accessed: 09.05.2019).

[58] www.opcfoundation.org. *Practical Security Recommendations for building OPC UA Applications*. URL: https://opcfoundation.org/wp-content/uploads/2017/11/OPC-UA-Security-Advise-EN.pdf. (accessed: 08.05.2019).

[59] Unified Automation GmbH. *Discovery and Security Configuration*. URL: https://documentation.unified-automation.com/uasdkhp/1.0.0/html/_l2_ua_discovery_connect.html. (accessed: 09.05.2019).

[60] www.opcfoundation.org. *This webpage provides links that will redirect to the correct profile in the UA Profile Reporting Tool, for any profiles whose URI's may not be directly addressed due to limits in the web browser.* URL: https://opcfoundation.org/UA/SecurityPolicy/#Basic128Rsa15. (accessed: 09.05.2019).

[61] www.opcfoundation.org. *SecurityPolicy – Basic128Rsa15 Profile*. URL: https://apps.opcfoundation.org/ProfileReporting/index.htm?ModifyProfile.aspx?ProfileID=a84d5b70-47b2-45ca-a0cc-e98fe8528f3d. (accessed: 09.05.2019).

[62] www.opcfoundation.org. *SecurityPolicy – Basic256 Profile*. URL: https://apps.opcfoundation.org/ProfileReporting/index.htm?ModifyProfile.aspx?ProfileID=45f01bb8-4a15-44f0-94ac-ff28f15869a5. (accessed: 09.05.2019).

[63] Suman Das. "Design and Methodology of Automated Guided Vehicle". In: *IOSR journal of mechanical and civil engineering* (Apr. 2016).

[64] PAL Robotics. *TIAgo TECHNICAL SPECIFICATIONS*. URL: http://tiago.pal-robotics.com/wp-content/uploads/2018/03/Datasheet_TIAGo-Hardware-Software.pdf. (accessed: 09.05.2019).

Appendices

A Videos from tests

Description	Number	URL
RSI communication with python control	1	https://www.youtube.com/watch?v=2fQ7irRJt2g
	2	https://www.youtube.com/watch?v=yvbaAGOpaHc
RSI communication with Visual Components	3	https://www.youtube.com/watch?v=P7q6WmCTeL8
	4	https://www.youtube.com/watch?v=LuittMhbgwc&t=1s
Movment KUKAVARPROXY	5	https://www.youtube.com/watch?v=I_3kEDNeW5s
		https://www.youtube.com/watch?v=K-Yqe1s_gfk
Keyboard program KUKAVARPROXY	7	https://www.youtube.com/watch?v=Oyrc2VMLHGw
One way digital twin KUKAVARPROXY	8	https://www.youtube.com/watch?v=EWYvDx7DCG8
	9	https://www.youtube.com/watch?v=wdgzXG8xIz4
Two way digital twin KUKAVARPROXY, with 10% and 100% speed	10	https://www.youtube.com/watch?v=vQxy7p4qPIU
Screen recording of the AGV program	11	https://www.youtube.com/watch?v=GiW_2C9F4Hs
Visual Components recording of the AGV program	12	https://www.youtube.com/watch?v=MPSw5PO2B5I
Playlist of all the videos on Youtube		https://www.youtube.com/playlist?list=PLgmnyD3tnVWsQjkMbDW_uZFgVja6l60oS

Table 4: The videos taken form the diffrent programs made through this project.

B Code developed under this project

The code used in this project can be found in the attachment, with the same name as the chapter number as shown under.

B.1 One way digital twin

B.1.1 One way digital twin KUKAVARPROXY

B.2 Two way digital twin

B.2.1 Two way digital twin KUKAVARPROXY

B.2.2 Movment KUKAVARPROXY

B.2.3 Keyboard program KUKAVARPROXY

B.2.4 RSI communication in C sharp

B.2.5 RSI communication with Visual Components control

B.2.6 RSI communication with python control

B.2.7 Improved version of RSI communication with python control

B.3 AGV add-on

B.3.1 Planning AGV to the digital twin

B.4 KUKA code

B.4.1 Control OpenShowVar-KUKAVARPROXY

B.4.2 RSI_XYZ

B.4.3 RSI XML

B.4.4 RSI_A1_A6

C Read and write test of OpenShowVar-KUKAVARPROXY

D Setting up RSI joint control

This is a manual that goes through how to set up the KRL code to be able to control the robot with RSI by sending correction values of the joint axis.

The RSI KUKA.RobotSensorInterface 2.3 manual [52] and the HTML help guide called *"rsiCommands"* is used to create RSI communication with the external computer. *"rsiCommands"* includes all the different RSI commands, parameter, and how to create the objects.

D.1 defining the variables in the .dat file

When a program is created in the KUKA controller, you have a .dat file where you define the variables that are used. There is also a .src file that contains the program or the instructions of what will happen.

The variable hEthernet, err, and hmapare used to create RSI communication. While A_lim_low and A_lim_high are used to set how many degrees the robot is allowed to move. In this case, the robot is limited to 100 degrees in each direction in every joint.

```
DECL RSIERR err
INT hEthernet,hmap

INT hDin,hDout1,hDout2,hDout3,hAxis,hPath,hMapDiO

INT A_lim_low=-100
INT A_lim_high=100
```

D.2 Creating the objects in .src file

ST_ETHERNET is used to create an RSI object that allows for communicate over Ethernet. The object uses the XML file "RSIethernet.xml" which contains all the parameters required to develop communication between the server program and the robot controller [52].

```
; Create RSI Object ST_Ethernet, read object configuration .../INIT/ERXConfig.xml
err = ST_ETHERNET(hEthernet,0,"RSIethernet.xml")
IF (err <> #RSIOK) THEN
  HALT
ENDIF
```

When the robot is moving along the programmed path and KUKA.RobotSensorInterface calculates a path correction or axis angle correction, the robot corrects its path or axis angles. The path or axis angle correction can be absolute in relation to the programmed path, or relative to the correction of the previous interpolation cycle.

```
err = ST_AXISCORR(hAxis,0)
IF (err <> #RSIOK) THEN
  HALT
```

```
ENDIF
```

As mentioned in chapter 4.4.2, the default movement limit is ±5 degrees. Therefore the command ST_SETPARAM is used to expand the movement of the robot with the variable A_lim_low and A_lim_high. In the *"rsiCommands"* as shown in figure 41, you can specify how many degrees each joint can move and that the default value is ±5 degrees.

Object Parameters:

1 INTEG (INT): Integration mode
-> 0: Absolut
-> 1: Relativ (Default)
2 (REAL): Lower bound of correction A1, Default: -5
3 (REAL): Lower bound of correction A2, Default: -5
4 (REAL): Lower bound of correction A3, Default: -5
5 (REAL): Lower bound of correction A4, Default: -5
6 (REAL): Lower bound of correction A5, Default: -5
7 (REAL): Lower bound of correction A6, Default: -5
8 (REAL): Lower bound of correction E1, Default: -5
9 (REAL): Lower bound of correction E2, Default: -5
10 (REAL): Lower bound of correction E3, Default: -5
11 (REAL): Lower bound of correction E4, Default: -5
12 (REAL): Lower bound of correction E5, Default: -5
13 (REAL): Lower bound of correction E6, Default: -5
14 (REAL): Upper bound of correction A1, Default: 5
15 (REAL): Upper bound of correction A2, Default: 5
16 (REAL): Upper bound of correction A3, Default: 5
17 (REAL): Upper bound of correction A4, Default: 5
18 (REAL): Upper bound of correction A5, Default: 5
19 (REAL): Upper bound of correction A6, Default: 5
20 (REAL): Upper bound of correction E1, Default: 5
21 (REAL): Upper bound of correction E2, Default: 5
22 (REAL): Upper bound of correction E3, Default: 5
23 (REAL): Upper bound of correction E4, Default: 5
24 (REAL): Upper bound of correction E5, Default: 5
25 (REAL): Upper bound of correction E6, Default: 5

Figure 41: A screen shot from the HTML help guide called *"rsiCommands"*. It shows details about the command ST_AXISCORR.

```
err=ST_SETPARAM(hAxis,2,A_lim_low)
IF (err <> #RSIOK) THEN
  HALT
ENDIF
err=ST_SETPARAM(hAxis,3,A_lim_low)
IF (err <> #RSIOK) THEN
  HALT
ENDIF
err=ST_SETPARAM(hAxis,4,A_lim_low)
IF (err <> #RSIOK) THEN
  HALT
ENDIF
err=ST_SETPARAM(hAxis,5,A_lim_low)
IF (err <> #RSIOK) THEN
  HALT
ENDIF
err=ST_SETPARAM(hAxis,6,A_lim_low)
IF (err <> #RSIOK) THEN
```

```
  HALT
ENDIF
err=ST_SETPARAM(hAxis,7,A_lim_low)
IF (err <> #RSIOK) THEN
  HALT
ENDIF

err=ST_SETPARAM(hAxis,14,A_lim_high)
IF (err <> #RSIOK) THEN
  HALT
ENDIF
err=ST_SETPARAM(hAxis,15,A_lim_high)
IF (err <> #RSIOK) THEN
  HALT
ENDIF
err=ST_SETPARAM(hAxis,16,A_lim_high)
IF (err <> #RSIOK) THEN
  HALT
ENDIF
err=ST_SETPARAM(hAxis,17,A_lim_high)
IF (err <> #RSIOK) THEN
  HALT
ENDIF
err=ST_SETPARAM(hAxis,18,A_lim_high)
IF (err <> #RSIOK) THEN
  HALT
ENDIF
err=ST_SETPARAM(hAxis,19,A_lim_high)
IF (err <> #RSIOK) THEN
  HALT
ENDIF
```

ST_NEWLINK is used to link *"the signal output of an RSI object to an optional signal input of a different RSI object"* [52].

```
err = ST_NEWLINK(hEthernet,7,hAxis,1)
IF (err <> #RSIOK) THEN
  HALT
ENDIF
err = ST_NEWLINK(hEthernet,8,hAxis,2)
IF (err <> #RSIOK) THEN
  HALT
ENDIF
err = ST_NEWLINK(hEthernet,9,hAxis,3)
IF (err <> #RSIOK) THEN
  HALT
ENDIF
err = ST_NEWLINK(hEthernet,10,hAxis,4)
IF (err <> #RSIOK) THEN
  HALT
ENDIF
err = ST_NEWLINK(hEthernet,11,hAxis,5)
```

```
IF (err <> #RSIOK) THEN
  HALT
ENDIF
err = ST_NEWLINK(hEthernet,12,hAxis,6)
IF (err <> #RSIOK) THEN
  HALT
ENDIF
```

At last the RSI communication is turned on by the command ST_ON1. The ST_SKIPSENS() is then used to hold on and wait so that the code doesn't finish and stop.

```
err = ST_ON1(#BASE,1)
IF (err <> #RSIOK) THEN
  HALT
ENDIF
; *=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*
ST_SKIPSENS() ;Hold on - until RSI-Break reason occur
; *=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*
```