NTNU
Norwegian University of
Science and Technology

Industry 4.0 - Digital Twins and OPC UA

Aksel Øvern

Master of Science in Mechanical Engineering

Submission date:	June 2018
Supervisor:	Olav Egeland, MTP
Co-supervisor:	Eirik Njåstad, MTP

Norwegian University of Science and Technology
Department of Mechanical and Industrial Engineering

Preface

This is the concluding Master's thesis of a five-year study programme in Mechanical Engineering at the Department of Mechanical and Industrial Engineering, NTNU. The thesis was written in the spring of 2018, but it is based on work done both in the autumn of 2017 and the spring of 2018. The experimental work was performed in the robotics laboratory of the department at Valgrinda, Trondheim.

With my background from starting my Master's Degree at the Department of Engineering Cybernetics, this project gave me the opportunity to combine my interests for robotics and production technology. The study of the technology used, and the work done in the robotics lab, has been exciting and a really good learning experience.

Trondheim, 11-06-2018

Aksel Øvern

Aksel Øvern

Acknowledgement

I would like to thank my supervisors, Olav Egeland and Eirik Njåstad, for providing me with an interesting project and for guidance and help during the work done. I will especially like to express my gratitude to Eirik, for not only taking time to answer my many questions but also for good discussions, continuously sharing your experience on the robots in the lab and for helping me in my experimental work.

During the work on this project I was allowed to attend ICNSC 2018, the 15th IEEE International Conference on Networking, Sensing and Control in Zhuhai, China. Attending this conference and seeing what work is currently being done within the field of Digital Twins in the industry was a great motivation and provided me with ideas that have been implemented in this project.

I would like to thank Olivier Roulet-Dubonnet at SINTEF Raufoss Manufacturing AS for taking time for a meeting about his work on FreeOpcUa, and Torstein Anderssen Myhre for helping me with a setup in the lab related to his work on KUKA RSI.

Also, I would like to thank the other students at the department working with their Master's thesis in the same facilities as me, for academic cooperation and keeping the morale up.

Last, I would like to thank the department for over the years expanding the robot lab, to facilitate such opportunities for students.

A.Ø.

Summary

This project explores the term Industry 4.0 (I 4.0) and the use of Digital Twins (DTws) as an asset in this modern industrial revolution. A DTw can be described as a digital replica of a physical system including data about this systems interaction with its environment. The goal of this project has been to develop a DTw for a robot cell at MTP Valgrinda, NTNU, and investigate what benefits could be gained from introducing the technology in this system and the domain of automated robotic systems in general.

The DTw was developed using the OPC UA communication architecture. OPC UA is called the pioneer of I 4.0 [1] as it is a communication architecture aiming at the standardization of communication in industry. Three different visualization software solutions were compared. It was concluded that Visual Components 4.0 (VC 4.0) was the strongest candidate for developing a visual representation of the robot cell. Using VC 4.0 and OPC UA a DTw of the robot cell was created.

Most of the work done in this project revolved around creating communication modules able to connect the physical robot cell to the virtual representation in VC 4.0, through the use of OPC UA. The result of this work is a communication library containing the virtual representation of the robot cell and the different communication modules able to give the DTw various functionalities. This library is made open-source and can be found in the digital appendix B.1 and GitHub[1].

The software included in this library enables the DTw to do real-time mirroring of the physical robot's movements, plotting of robotic sensor data, and controlling the robots from the DTw. A graphical user interface was developed to organize the functionalities. Figure 3 illustrates the current communication architecture for the DTw.

The DTw was tested in a fixed case. This was a multi-robot case including mirroring of movements, plotting of sensor data and control from VC 4.0. It was made a video from the case, found in the digital appendix B.2, and published online[2].

It was concluded that OPC UA was a good solution for use in this DTw as it is possible to implement on any platform, and is enabling a more flexible and structured way of communicating than traditional communication software used in client/server based systems. The benefits found with the use of the DTw in this automated robotic system included visibility to operations and a better foundation for statistical analysis to predict future states and for optimizing characteristic parameters associated with the robot cell. Finally, it was concluded that the DTw act as a good foundation for managing a complex system, something that could be beneficial as this specific system is used in training and professional development at the institute.

[1] `https://github.com/akselov/digital-twin-opcua`
[2] `https://youtu.be/xlQhQPmJwlA`

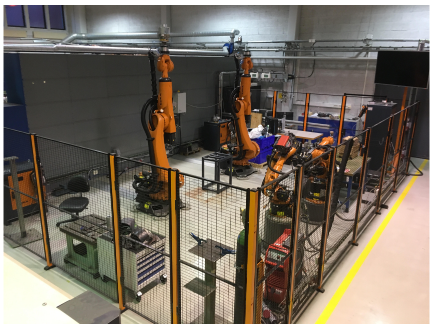


Figure 1: Physical robot cell

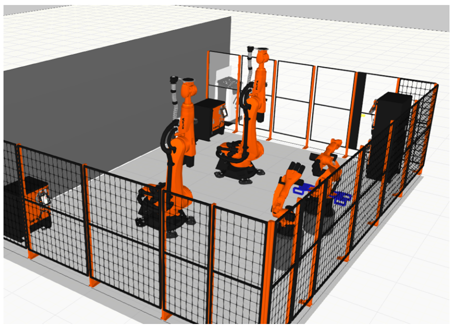


Figure 2: Digital twin

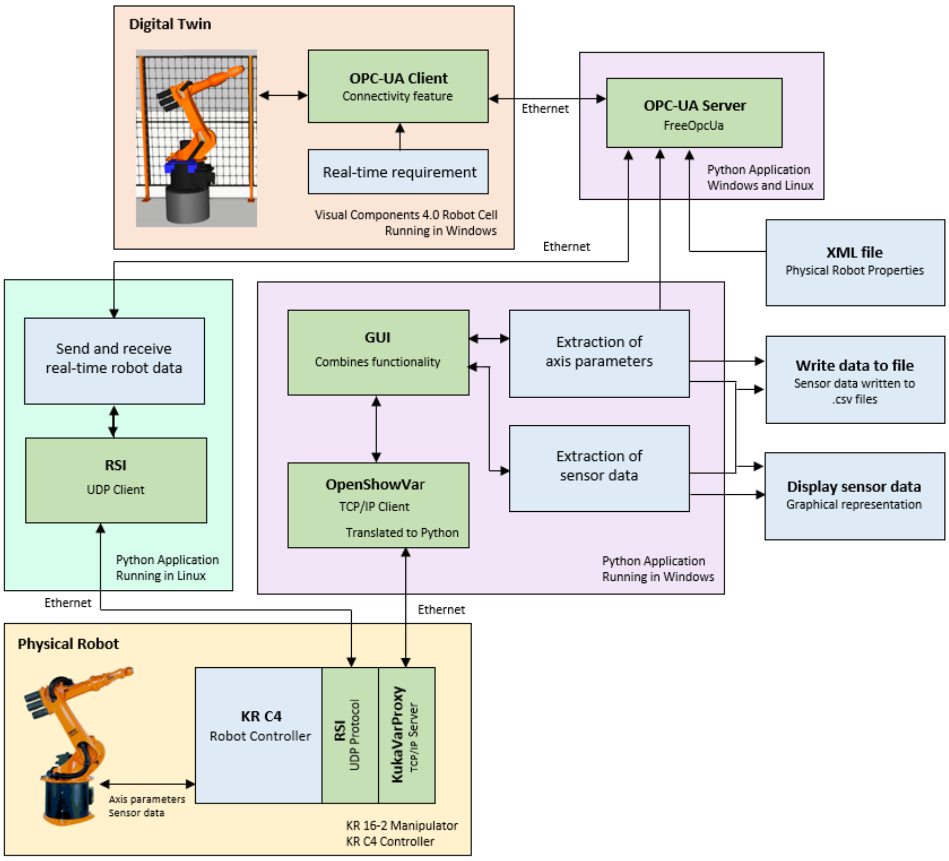


Figure 3: Communication architecture

Sammendrag

Dette prosjektet utforsker begrepet Industri 4.0 (I 4.0) og bruken av Digitale Tvillinger (DTws) som en ressurs i denne moderne industrielle revolusjonen. En DTw kan beskrives som en digital kopi av et fysisk system, inkludert data forbundet til systemet og den interaksjon det har med sine omgivelser. Målet med dette prosjektet har vært å utvikle en DTw for en robotcelle på MTP Valgrinda, NTNU, og utforske hvilke fordeler denne teknologien kan gi for dette systemet og automatiserte robotsystemer generelt.

Det ble utviklet en DTw som tar i bruk OPC UA som kommunikasjonsarkitektur. OPC UA kalles en pionér innenfor I 4.0 da dette er en arkitektur som tar sikte på å standardisere kommunikasjon i industri. Tre forskjellige programvarer for visualisering ble sammenlignet. Det ble konkludert med at Visual Components 4.0 (VC 4.0) var den beste kandidaten for utvikling av en visuell representasjon av dette systemet. Ved bruk av VC 4.0 og OPC UA ble det utviklet en DTw for robotcella.

Majoriteten av arbeidet utført i dette prosjektet dreide seg om utvikling av kommunikasjonsmoduler som kunne koble den fysiske robotcella til den digitale modellen i VC 4.0, ved bruk av OPC UA. Resultatet er et bibliotek som inneholder den virtuelle modellen og kommunikasjonsmodulene som gir dette systemet ulike funksjoner i kontakt med den fysiske robotcella. Dette biblioteket har åpen kildekode som finnes i det digitale vedlegget B.1 og på GitHub[3].

Programvaren i dette biblioteket gjør det mulig for DTw-en å utføre speiling av de fysiske robotbevegelsene i sanntid, plotting av sensordata og ekstern styring av robotene. Et grafisk brukergrensesnitt ble utviklet for å bedre organiseringen av funksjonalitetene. Figur 3 viser nåværende kommunikasjonsarkitektur.

Systemet ble testet i et case. Dette inkluderte speiling av bevegelser i sanntid, plotting av sensordata og styring fra VC 4.0. Det ble laget en video som viser systemet i bruk. Denne finnes i det digitale vedlegget B.2 og er publisert online[4].

Det ble konkludert med at OPC UA var en god løsning for bruk i denne DTw-en da dette er en arkitektur som er mulig å implementere uavhengig av plattform. Dette gir en mer fleksibel og strukturert kommunikasjon sammenlignet med tradisjonelle klient-/serverbaserte systemer. Fordelene som ble trukket frem ved bruk av en DTw i dette automatiserte robotsystemet var en forbedret oversikt over operasjoner, samt et bedre grunnlag for statistisk analyse for prediksjoner av fremtidig tilstand og optimalisering av karakteristiske parametere forbundet til robotcella. Til slutt ble det konkludert med at DTw-en fremstår som et godt grunnlag for administrering av et komplekst system, noe som kan være fordelaktig da denne robotcella er brukt i opplæring og profesjonell utvikling på instituttet.

[3]`https://github.com/akselov/digital-twin-opcua`
[4]`https://youtu.be/xlQhQPmJwlA`

Contents

List of Figures

List of Tables

Terms and Abbreviations

OPC - *Open Platform Communications*, A standard for exchange of data used in the industrial automation space and in other industries [33]. For a more in depth explanation see section 2.3.

XML - *Extensible Markup Language*, A markup language used in computing, that defines a set of rules for encoding documents in a format that is both human-readable and machine-readable [34].

ADI - *Analyzer Device Integration*, OPC UA companion specification for analyzer devices that defines a data model for process and laboratory analyzers [35].

CPS - *Cyber-Physical System*, system which connects real (physical) objects and processes with information processing (virtual) objects and processes via information networks which are open, partially global, and at any time connected [36].

CPPS - *Cyber-Physical Production System*, CPS system used in production, allowing machines and products to exchange information, trigger actions and control other components autonomously [7].

ICT - *Information and Communications Technology*, An extended term for information technology (IT) which stresses the role of unified communications and the integration of telecommunications (telephone lines and wireless signals), computers as well as necessary enterprise software, middleware, storage, and audio-visual systems, which enable users to access, store, transmit, and manipulate information [37].

COM - *Component Object Model,* A platform-independent, distributed, object-oriented system for creating binary software components that can interact [38].

DCOM - *Distributed Component Object Model*, A Microsoft technology for communication between software components on networked computers [39].

OLE - *Object Linking and Embedding*, A technology developed by Microsoft that allows embedding and linking to documents and other objects [40].

IoT - *Internet of Things*, The network of physical devices, vehicles, and other items embedded with electronics, software, sensors, actuators, and network connectivity which enable these objects to collect and exchange data [41].

IoE - *Internet of Everything*, The intelligent connection of people, process, data and things [42].

DTw - *Digital Twin*, Virtual representation of a physical object. For a more in dept explanation see section 2.2.

I 4.0 - *Industry 4.0*, A name for the current trend of automation and data exchange in manufacturing technologies [43].

RAMI 4.0 - *Reference Architectural Model Industry 4.0*, A three-dimensional map showing how to approach the issue of I 4.0 in a structured manner [9].

SOA - *Service-Oriented Architecture*, A style of software design where services are provided to the other components by application components, through a communication protocol over a network [44].

STS - *Socio-Technical system*, A system that considers requirements spanning hardware, software, personal, and community aspects, and applies an understanding of the social structures, roles and rights (the social sciences) to inform the design of systems that involve communities of people and technology [45].

CAD - *Computer-aided design*, The use of computer systems (or workstations) to aid in the creation, modification, analysis, or optimization of a design [46].

VC 4.0 - *Visual Components 4.0*, A commercial 3D simulation software, combines three engineering disciplines in one platform, including, material flow, robotics simulation, and controls validation [47].

.NET - *(pronounced dot net)*, A software framework developed by Microsoft [48].

API - *Application Programming Interface*, Aa set of routines, protocols, and tools for building software applications [49].

UI - *User interface*, Space where interactions between humans and machines occur in the industrial design field of human–computer interaction [50].

SCADA - *Supervisory Control and Data Acquisition*, A control system architecture that uses computers, networked data communications and graphical user interfaces for high-level process supervisory management [51].

KRL - *KUKA Robot Language*, Programming language used in KUKA robots.

RSI - *Robot Sensor Interface*, Software used for configuration of signal processing in a real-time system of KUKA robots [19].

KVP - *KUKAVARPROXY*, A TCP/IP server that runs on a KUKA robot controller, originally created by Massimiliano Fago [52].

cmd - *Command Prompt*, The command-line interpreter on Windows operating systems.

GUI - *Graphical User Interface*, A type of user interface that allows the use of icons or other visual indicators to interact with electronic devices, rather than using only text via the command line [53].

MDA - *Model-Driven Architecture*, An approach to software design, development and implementation providing guidelines for structuring specifications, which are expressed as models [54].

Chapter 1

Introduction

1.1 Purpose and goal of project

The project description that was given by the supervisors was the following:

In order to respond quickly to unexpected events and new demands without extensive system changes, future production systems must be able to work more independently. There is a need for intelligent machines that perform complex tasks without detailed programming and without human interaction. Autonomous systems know their own abilities (which are modeled as "skills") and their state. They are able to choose between a set of possible actions, orchestrating and perform their skills. To succeed, the autonomous systems need to have realistic models of the current state of the production process and the system's own behaviour in interaction with its external environment - usually called a digital twin.

OPC Unified Architecture (OPC UA) is a platform-independent protocol for machine-to-machine communication for industrial automation developed by the OPC Foundation. OPC focuses on accessing large amounts of real-time data at the same time as system performance is affected to a minimum. OPC UA has the potential to become an important foundation in the future industrial environment where machines deliver "production as a service", and all machines and sensors in production are online (Internet of Things).

In this task, implementation of a digital twin will be studied. A solution must be developed that provides seamless communication between robots, PLCs, and other relevant control systems that can be part of an industrial production system, linked to one digital representation of the system. The system must be tested at the institute Robot Laboratory.

a) Describe how a digital twin can be implemented using OPC UA.

b) Examine the advantages and disadvantages of using Visual Components 4.0 or similar simulation software for the digital twin.

c) Use Visual Components 4.0 or similar simulation software to model and simulate the robot cell at the Institute.

d) Examine the advantages and disadvantages of using KUKAVARPROXY and KUKA RSI Ethernet as middleware between the KUKA KR C4 robot controllers and the digital twin.

e) Present a solution for a digital twin of the Institute's Robot Laboratory with use of OPC UA for communication.

f) Try out the system in a fixed case. Evaluate the results.

1.2 Background

"Unlike computers, humans do not process information, at least not in the sense of sequential step-by-step processing that computers do. Instead, humans look at a situation and conceptualize the problem and the context of the problem. Humans take in all the data about the situation there interested in. They then conceptualize the situation, seeing in their mind's eye its various aspects. While they can do this looking at tables of numbers, reports, and other symbolic information, their most powerful and highest bandwidth input device is their visual sight.

What currently happens is that humans take visual information, reduce it to symbols of numbers and letters, and then re-conceptualize it visually. In the process, we lose a great deal of information, and we introduce inefficiencies in time. The capability of the digital twin lets us directly see the situation and eliminate the inefficient and counterproductive mental steps of decreasing the information and translating it from visual information to symbolic information and back to visually conceptual information."
- Dr. Michael Grieves [55]

The concept of digital twins was named one of Gartner's Top 10 Strategic Technology Trends for both 2017 and 2018 [56]. Thomas Kaiser, SAP Senior Vice President of IoT, put it this way: *"Digital twins are becoming a business imperative, covering the entire lifecycle of an asset or process and forming the foundation for connected products and services. Companies that fail to respond will be left behind."* [57]

It is estimated that within three to five years, billions of things will be represented by digital twins [58]. Using physics data on how the components of a thing operate and respond to the environment as well as data provided by sensors in the physical world, a digital twin can be used to analyze and simulate real-world conditions, responds to changes, improve operations and add value.

Figure 4: Chris O'Connor (IBM) at IoT Genius of Things summit 2017 [2]

The term "digital twin" has in the last years gained a lot of attention and hype. It is a technology that is said to have a wide range of benefits in the modern industry. This project aims at analyzing what benefits could be gained from introducing digital twins in the domain of industrial automated robotic systems. The industry has for years used robots to automate single tasks and production lines. How could these processes be improved by the use of digital twins, and is this technology indeed a step in the evolution of industry towards the new industrial revolution, named Industry 4.0.

Multiple papers have been published on how digital twins could benefit production systems. This includes papers such as *About The Importance of Autonomy and Digital Twins for the Future of Manufacturing* by Rosen, von Wichert, Lo and Bettenhausen [59], and *A Review of the Roles of Digital Twin in CPS-based Production Systems* by Negri, Fumagalli and Macchi [60]. However, after doing a literature study on the subject, it was found that little work has been done to make a digital twin for a given system and observe the benefits this provides.

In this project a digital twin of the robot cell at the department of MTP Valgrinda is made, and its performance is studied. Much of the work done is related to the software needed for such a system to provide functionalities that make the system valuable. While the Unified Modelling Language (UML) is established as the major modeling language in software development, there are several standards currently being used in the manufacturing environment [7]. For each of them, the model and the code must be written and maintained separately, which requires high manual effort. OPC UA is called the pioneer of the 4th industrial revolution [1], as it is a communication architecture aiming at the standardization of communication in industry. This project investigates why OPC UA is beneficial for use in modern industrial systems and uses this architecture in the development of a communication system for the digital twin.

1.3 Structure of report

This report is structured as follows:

Chapter 2: Presents theory relevant to the work done in this project. Focuses on general terms and technology that have been used, together with specific theory related to the components that constitute the robot cell at MTP Valgrinda.

Chapter 3: Describes the method used to develop the digital twin and the solutions found along the way. Includes the selection of software used, the development of a virtual representation, and the development of software capable of combining the virtual representation and the physical robot cell. Finally, a description of a fixed case that was performed to test the system in action.

Chapter 4: Presents the result of the work done in this project. Divided into the categories: digital twin in Visual Components 4.0, communication library, the setup used in the fixed case, and a description of the system's functionalities.

Chapter 5: Analysis of the system developed and discussion of software, data acquisition, and communication architecture. Includes a discussion of the uncertainties, safety features, and improvements that should be done to take this digital twin to the next level. Concluded by a discussion of the benefits the digital twin provides, both for the institute and for the evolution towards industry 4.0.

Chapter 6: Conclusion and recommendations for further work. Includes experience from using the system in a fixed case, and a conclusion of the performance of the system and the value it provides for the institute and industry 4.0.

Chapter 2

Theory

This chapter will present theory relevant to the work done in this project. From general terms used, to specific theory related to the components that constitute the robot cell at MTP Valgrinda.

2.1 Industry 4.0

The term Industry 4.0 (I 4.0) became publicly known in 2011, when an initiative named *Industrie 4.0 – an association of representatives from business, politics, and academia* supported the idea as an approach for strengthening the competitiveness of the German manufacturing industry [61]. In later time numerous academic publications, articles, and conferences have tried to approach and describe the topic in a variety of ways. The term is to this date still new to both industry and media, and as a result, a generally accepted understanding of I 4.0 has not been published so far. As a result, discussing the topic on an academic level is difficult, and so is implementing I 4.0 scenarios.

The term I 4.0 is mostly used for describing what is taught to be the next industrial revolution, which is about to take place right now [62][63]. Like illustrated in figure 5 this industrial revolution has been preceded by three other industrial revolutions in modern history. The first industrial revolution was the introduction of mechanical production facilities starting in the second half of the 18th century and being intensified throughout the entire 19th century. From the 1870s on, electrification and the division of labour led to the second industrial revolution. The third industrial revolution, also called *the digital revolution*, set in around the 1970s [61], when advanced electronics and information technology developed further the automation of production processes.

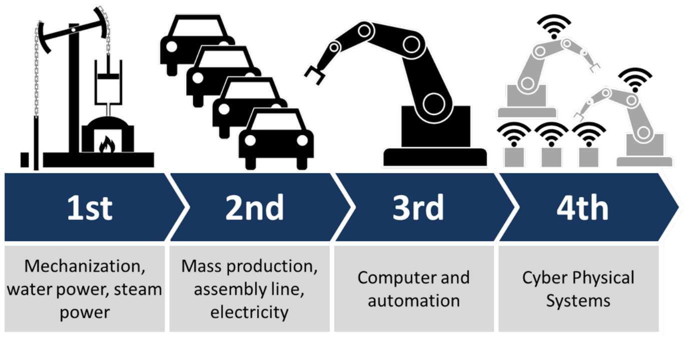

1st
2nd
3rd
4th
Mechanization, water power, steam power
Mass production, assembly line, electricity
Computer and automation
Cyber Physical Systems

Figure 5: Industrial revolutions including apriori prediction [3]

The establishment of I 4.0 is the first time an industrial revolution is predicted apriori, not observed ex-post [64]. This provides various opportunities for companies and research institutes to shape the future actively. The economic impact of this industrial revolution is supposed to be substantial [61], as I 4.0 promises substantially increased operational effectiveness as well as the development of entirely new business models, services, and products.

The idea of I 4.0 is deeply interconnected with the term Internet of Everything (IoE). IoE is also a somewhat diffuse term, but as described by Cisco it has its base in the intelligent connection of people, process, data and things [42]. I 4.0 demands a higher degree of flexibility, adaptability, transparency and general requirements which have to be fulfilled by interconnected components and systems [36].

In the flooding of media articles related to this topic, it is difficult to extract what exactly this new industrial revolution is founded on and how today's industry can advance to this next evolutionary step. Although not including a clear definition on the subject an report by some of the key promoters of the idea of I 4.0, the German *Industrie 4.0 Working Group*, describes the vision and basic technologies the idea of I 4.0 aims at, together with selected scenarios. This gives a better understanding of not only what I 4.0 is, but rather how we can get there. The report was published in 2013 and authored by Prof. Dr. Henning Kagermann (National Academy of Science and Engineering), Prof. Dr. Wolfgang Wahlster (German Research Center for Artificial Intelligence) and Dr. Johannes Helbig (Deutsche Post AG), with the purpose of gaining insight to the future of German manufacturing industry [65]. The Industrie 4.0 Working Group believes that action is needed in the following eight key areas:

1. **Standardization and reference architecture:** Considering I 4.0 will involve networking and integration of several different companies through value networks, this collaborative partnership will only be possible if a single set of common standards is developed. A reference architecture will be needed to provide a technical description of these standards and facilitate their implementation.

2. **Managing complex systems:** Due to products and manufacturing systems becoming more and more complex, appropriate planning and explanatory models should provide a basis for managing this growing complexity. Engineers should, therefore, be equipped with the methods and tools required to develop such models.

3. **A comprehensive broadband infrastructure for industry:** Reliable, comprehensive and high-quality communication networks are a key requirement for I 4.0. Broadband Internet infrastructure, therefore, needs to be expanded on a massive scale.

4. **Safety and security:** Safety and security are critical to the success of smart manufacturing systems. It is important to ensure that production facilities and the products themselves do not pose a danger either to people or to the environment while at the same time, both production facilities and products, in particular, the data and information they contain, need to be protected against misuse and unauthorized access.

5. **Work organization and design:** In smart factories, the role of employees will change significantly. Increasingly real-time oriented control will transform work content, work processes, and the working environment. Emphasis on implementation of an approach to socio-technical systems (STS) to work organizations that will offer workers the opportunity to enjoy greater responsibility and enhance their personal development.

6. **Training and continuing professional development:** I 4.0 will radically transform worker's job and competency profiles. It will, therefore, be necessary to implement appropriate training strategies and to organize work in a way that fosters learning, enabling lifelong learning where digital learning techniques should be investigated.

7. **Regulatory framework:** While the new manufacturing processes and business networks found in I 4.0 will need to comply with the law, existing legislation will also need to be adapted to take account of new innovations. The challenges include the protection of corporate data, liability issues, handling of personal data and trade restrictions.

8. **Resource efficiency:** Manufacturing industry's consumption of large amounts of raw materials and energy poses a number of threats to the environment and security of supply. I 4.0 will deliver gains in resource productivity and efficiency. It will be necessary to calculate the trade-offs between the additional resources that will need to be invested in smart factories and the potential savings generated.

The report emphasize that the journey towards I 4.0 will be an evolutionary process. Current basic technologies and experience will have to be adapted to the specific requirements of manufacturing engineering and innovative solutions for new locations and new markets will have to be explored [65].

2.2 Digital Twins

The idea of a Digital Twin (DTw) is said to originate from NASA's Apollo program, where at least two identical space vehicles were built to allow mirroring the conditions of the space vehicle during a mission [66]. The specific term "Digital Twin" was however first defined by Dr. Michael Grieves at the University of Michigan around 2001-2002 [67][55]. He originally defined the term in the context of Product Lifecycle Management. In his paper, he introduced the concept of "Digital Twin" as a virtual representation of what has been manufactured. He promoted the idea of comparing a DTw to its engineering design to better understand what was produced versus what was designed.

The term DTw is still what can be categorized as a buzzword without any precise definition. However, to conceptualize this technology both industry and academia have tried to define the term in several different ways for years. A publication from Massachusetts Institute of Technology [68] defines a DTw as an integrated model of an as-built product including physics, fatigue, lifecycle, sensor information, performance simulations, that is intended to reflect all manufacturing defects and be continually updated to include wear-and-tear sustained while in use. US Department of Defense (DoD) defines a DTw as an integrated multi-physics, multiscale, probabilistic simulation of an as-built system, enabled by Digital Thread, that uses the best available models, sensor information, and input data to mirror and predict activities/performance over the life of its corresponding physical twin [69]. The more general definition used in media and various industrial publication is the definition of a DTw as a digital replica of physical assets, processes, and systems that can be used for various purposes [70]. One can think of a DTw as a bridge between the physical and the digital world.

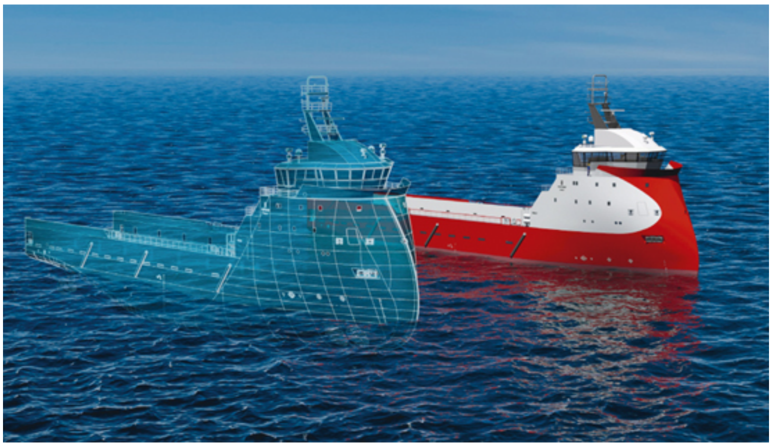


Figure 6: Digital twin for shipbuilding industry [4]

According to the German Association for Information Technology, Telecommunications and New Media (BITKOM), DTws in the manufacturing industry will have a combined economic potential of more than €78 billion by 2025. However, this potential can only be achieved if future systems are not only networked, but are also capable of seeing, understanding, and self-optimizing in order to adapt to future changes [71].

The concept of the DTw is a very powerful one. As this is not a specific technology, it is hard to describe the concrete benefits this concept provides. However, most DTws have characteristics that give some typical benefits [67]:

- **Visibility:** The DTw allows visibility in the operations of the machines as well as in the larger interconnected systems such as a manufacturing plant or an airport.
- **Predictive:** Using various modeling techniques (physics-based and mathematics based), the DTw model can be used to predict the future state of the machines and optimize characteristic parameters for production – from energy consumption to error rates, maintenance and cleaning cycles [71].
- **What if analysis:** Through properly designed interfaces, it is easy to interact with the model and ask the what-if questions to the model to simulate various conditions that are impractical to create in real life.
- **Documentation and communication mechanism to understand and explain behaviour:** A DTw can be used as a communication and documentation mechanism to understand as well as explain the behaviour of an individual machine or a collection of data from several similar systems.
- **Connect disparate systems such as backend business applications:** If designed correctly, the DTw model can be used to connect with the backend business applications to achieve business outcomes in the context of supply chain operations including manufacturing, procurement, warehousing, transportation and logistics, field service, etc.

2.2.1 Digital Twins and Internet of Things

The typical benefits provided by DTws listed in section 2.2 is linked to the idea of I 4.0 and connected systems. Markets dealing with the use of Internet of Things (IoT) have therefore taken much interest in the concept of DTws. In a nutshell, IoT is the concept of connecting any device (so long as it has an on/off switch) to the Internet and to other connected devices [72]. Every device that is activated and registered with an IoT platform has two categories of data [73].

The first category is the metadata that doesn't change very often. It includes details that describe the device such as the serial number, asset identifier, firmware version, make, model and year of manufacturing. The second type of data associated with the device represents the dynamic state. It contains context-specific, real-time data unique to the device. In this dynamic state the system can benefit from the implementation of real-time data connected with a DTw. Often this is accomplished by extensive use of sensor technology. Today almost every IoT platform has implemented some sort of DTw capabilities [67], of course with varying degrees of complexity and apparent differences regarding their maturity and vision. An IoT platform can bring operational and performance data into the DTw and thus provide data on how the product is performing for example in comparison with a theoretical calculated performance. This can be valuable information that closes the loop from the operations group back to the people doing design and R&D.

Without this closed loop, product simulation models used in design lack knowledge of the physical product. With the closed loop, a product simulation model allows analysis of real-time product data and monitoring of product assets to improve product design and operation. As suitable software analyzes data from the closed loop this could potentially prevent downtime, expose new opportunities and in general give a better foundation for taking operational and strategic decisions.

Figure 7: Sensor technology used to assist wind turbine [5]

2.3 OPC

Today the acronym OPC stands for Open Platform Communications [33]. OPC was first defined by a number of players in automation together with Microsoft back in 1995. Over the following ten years, it became the most used versatile way to communicate in the automation layer in all types of industry [6]. Over the years it has evolved to have more extensive functionality and reach.

Originally OPC stood for OLE (Object Linking and Embedding) in Process Control. OLE was a technology developed by Microsoft that allows embedding and linking to documents and other objects. Initially, the OPC standard was restricted to the Windows operating system [33]. The specifications of this version of the OPC is today known as OPC Classic. The OPC Classic specifications are based on Microsoft Windows technology using the COM/DCOM for the exchange of data between software components on networked computers [74].

Like illustrated in figure 8, OPC is client/server based communication which means that it exists one or more servers that wait for several clients to make requests. The server is connected to components for example PLC and Control devices and performs the requested actions. In OPC it is the client that decides when and what data the server will fetch from the underlying systems [6].

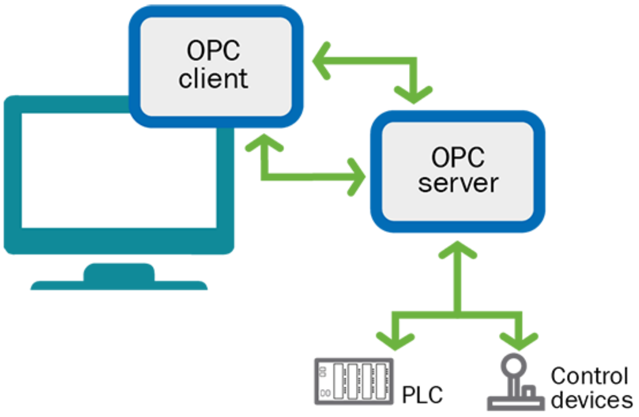


Figure 8: OPC structure [6]

The OPC protocols are completely self-sustained and have nothing in common. In the OPC Classic you have the following protocols [6]:

- **DA** - Data Access: The most basic protocol of the OPC stack that gets data out of the control systems into other systems on the shop floor

- **AE** - Alarm & Events: Subscription based service where the clients gets all the time stamped events that comes in
- **HDA** - Historical Data Access: Contains historical data and supports long record sets of data for one or more data points
- **XML DA** - XML Data Access
- **DX** - Data eXchange

2.4 OPC UA

The OPC Unified Architecture (UA) was released in 2008 as a service-oriented architecture that integrates all the functionality of the individual OPC Classic specifications into one framework that would better meet the emerging needs of industrial automation [75]. The most significant difference between classical OPC and OPC UA is that it doesn't rely on OLE or DCOM technology from Microsoft that makes it possible to implement it on any platform if that being Apple, Linux (JAVA) or Windows [6].

OPC UA is an International Electrotechnical Commission (IEC) standard developed by the global non-profit organization OPC Foundation in collaboration with users, manufacturers and researchers [1].

Characteristics of the OPC UA are [76]:

- Focus on communicating with industrial equipment and systems for *data collection and control*
- Open - Freely available and implementable without restrictions or fees
- Cross-platform - not tied to one operating system or programming language
- Service-oriented architecture (SOA)
- Robust security

OPC UA is enabling technology for flexible, adaptive, and transparent production [36]. Key industries include pharmaceutical, oil and gas, building automation, industrial robotics, security, manufacturing and process control [76]. The variability and flexibility of OPC UA range from simple process data acquisition to complex monitoring, control, and analysis. Additionally, OPC UA is explicitly open for the integration and combination with other standards based on so-called companion specifications. These companion specifications include e.g. general device descriptions (OPC UA for devices) and analyzer device models (ADI) over plant descriptions.

2.4.1 CPS and CPPS

An important difference between OPC Classic and OPC UA is the possibility to use structures or models. This means that the data tags or points can be grouped and be given context which makes governance and maintenance much easier. These models can be identified at runtime which makes it possible for a client to explore possible connections by asking the server [6].

To get an understanding of the importance of this possibility, the term CPS (cyber-physical system) should first be explained. CPS is defined as a system which connects real (physical) objects and processes with information processing (virtual) objects and processes via information networks which are open, partially global, and at any time connected [36]. The introduction of CPS into the manufacturing environment lead to Cyber-Physical Production Systems (CPPS). CPPS work with conventional production and information communications technology, allowing machines and products to exchange information, trigger actions and control other components autonomously [7].

OPC UA is mostly used in higher automation levels for the purpose of monitoring and control, it also increases device connectivity via standard communication in lower automation levels. OPC UA abstracts data from the network technology and software application and offers a generic communication interface. In this way, it represents a technology for a transparent data representation/transmission between heterogeneous system components. Here are some key concepts important to CPPS that OPC UA can help realize [36]:

- **Separation of information model and data base structure:** Hiding hierarchical data base structures behind object-oriented, full-meshed information models in OPC UA to be independent of the fixed structure of the data base.

- **Minimalistic OPC UA client functionality:** Implementing base OPC UA client functionality for applications to separate information complexity from users.

- **Graphical information model definition:** Developing OPC UA servers together with end users (or a system integrator) based on a graphical representation of OPC UA models.

- **Common info view:** Data acquisition by means of OPC UA for example for a management view which is accessible from different sources like web interfaces, and which runs in parallel to the real-time communication.

- **Aggregating OPC UA server functionality:** OPC UA server which integrates and combines different underlying servers and their information.

2.4.2 Information model design and communication

The OPC UA specification consists of 13 parts. The first seven parts are related to the core specifications, e.g. the concept, security model, address space model, services, information model, service mappings, and profiles. The parts eight to thirteen are related to access type specifications like data access, alarms and conditions, programs, historical access, discovery and aggregate [7]. Figure 9 shows the most essential parts of OPC UA necessary for user-specific information models.

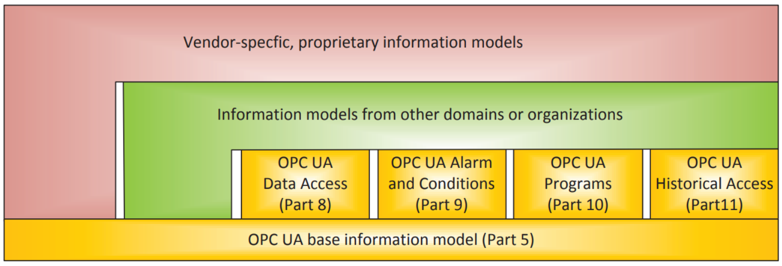

Vendor-specfic, proprietary information models
Information models from other domains or organizations
OPC UA Data Access (Part 8)
OPC UA Alarm and Conditions (Part 9)
OPC UA Programs (Part 10)
OPC UA Historical Access (Part11)
OPC UA base information model (Part 5)

Figure 9: OPC UA information model [7]

The advantage here is that the handling of data structures can be done in a way that data always is grouped and handled as one piece. This is important in many applications where it is essential that the data set is taken at the same time [6].

For securing the flow of information above the standard transport layers, the communication between client and server consist of two layers. One that handles the session and one to establish a secure channel between the client and server. This is illustrated in figure 10.

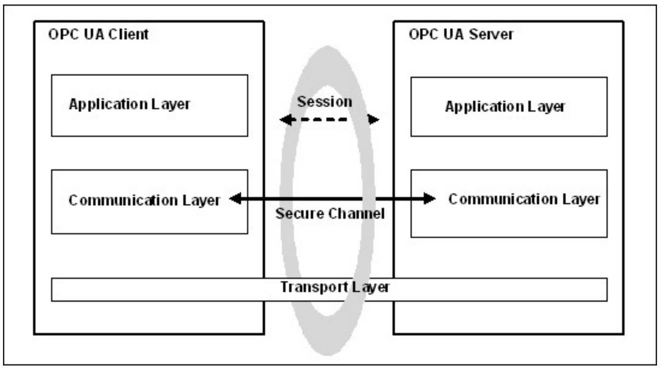


Figure 10: OPC UA communication model [8]

2.5 The role of OPC UA in Industry 4.0

Through digitalization and integrated networking I 4.0 turns simple value chains into fully digitalized value networks and creates new business models. This means IoT services and production objects also communicate with each other autonomously across companies. Several R&D topics arise from the design and implementation of CPPS and the standardization of interfaces for vertical and horizontal data exchange through networks [7].

To create a uniform basis for I 4.0, the German companies ZVEI, VDMA and BITKOM have teamed up to develop RAMI 4.0, the Reference Architecture Model for Industrie 4.0 [77]. According to ZVEI, RAMI 4.0 is the first and most advanced reference architecture model for I 4.0 [78]. The model is a three-dimensional map showing how to approach the issue of I 4.0 in a structured manner that ensures that all participants involved in I 4.0 discussions understand each other [9].

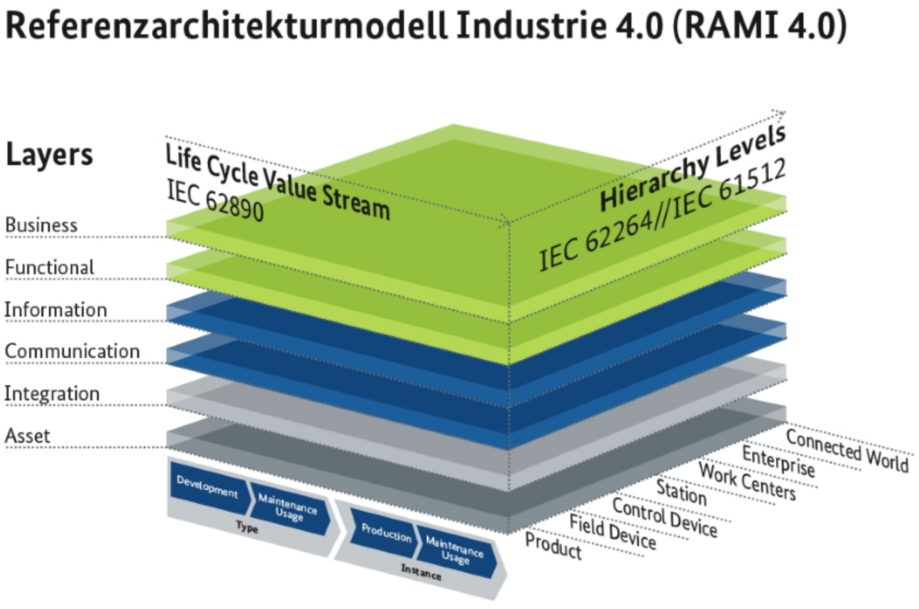


Figure 11: RAMI 4.0 [9]

In addition to describing equipment relevant to I 4.0, RAMI 4.0 describes future production networks consisting of I 4.0 components. These I 4.0-components have Service-Oriented Architecture (SOA) communication capabilities and a virtual representation which adds semantic information to the physical object. Semantic describes the relation between signifiers (e.g. words) and their denotation, what they stand for (the words meaning). This is in contrast with syntax which do not focus on the signifiers meaning [79]. This semantic is necessary because the exchange of information between two or more I 4.0-components requires clear and unambiguous semantics. Here RAMI 4.0 suggest OPC UA because it can be used for both communication as well as to define virtual representations [7]. OPC UA is in fact listed as the one and only recommendation for realizing the communication layer of the model [77].

2.6 Digital twins with OPC UA

From the various definitions of DTws presented in section 2.2 it is imminent that they all have a common denominator: the need for communication. OPC UA has evolved from just transmitting data to providing an encrypted pathway for both data and configuration and continues to evolve with new capability around automatically discovering devices on the network and connecting them without the need for manual configuration [80].

Also, DTws often present the need for model-driven techniques and solutions. These kinds of solutions are already being used to design complex software systems to simplify the process of information model design. The central concept of a model-driven approach is to separate the functional description and the implementation. In this way, it is not necessary to repeat the process of defining an application or the system's functionality and behaviour each time a new communication technology comes along [7]. As listed in section 2.4.1 OPC UA can help the realization of a model-driven structure through separation of the information model and data base structure.

2.6.1 Standardization with OPC UA

The structure of the OPC UA communication architecture developed in this project has been built on the ideas of the article *A systematic approach to OPC UA information model design* by Pauker, Frühwirth, Kittl and Kastner [7]. This article presents a model-driven approach to OPC UA information model design for the virtual representation of a system.

Model-driven techniques, which are already used to design complex software systems tremendously simplify the process of information model design. From the RAMI 4.0 model described in section 2.5 it can be seen that the layers on the vertical axis describe the decomposition of a machine into its properties structured layer by layer. Within the three axes of the RAMI 4.0 critical aspects of a system can be mapped.

The base component of the OPC UA meta model is nodes. Several node classes are defined specializing the base node class (e.g. objects and variables). Each node has a set of attributes depending on the node class. Some attributes are mandatory, and some are optional. For example, each node class requires a "NodeId" uniquely identifying the node, while the ”Description” is optional [81]. Data modeling is a fundamental part of OPC UA. The base principals are as follows [7]:

1. Using object-oriented techniques with type hierarchies and inheritance.
2. Type information is provided and can be accessed the same way as an instance.
3. Full meshed network of nodes allows connecting the information in different ways.
4. Extensibility of the type hierarchies as well as the references types between nodes.
5. No limitations of modeling in order to allow an appropriate information model design.
6. OPC UA information modeling is always done on the server.

In a Model-driven Architecture (MDA) models are the central elements of the software development process. The aim is to derive platform-specific models from platform independent models using a highly automated process. This reduces the effort of software development and the adaptation to new technologies. Any additions and changes to the model will be automatically reflected the next time the code is generated. MDA models can then be used for the production of documentation, acquisition specifications, system specifications, technology artifacts (e.g. source code) and executable systems. MDA defines three levels of abstraction [82]:

1. **The Computer Independent Model (CIM):** Shows what the system will do, but do not respect technology-specific information, to remain independent of the system implementation.
2. **The Platform Independent Model (PIM):** Specifies the system in more detail but still maintains a sufficient degree of generality to allow its mapping to one or more platforms
3. **The Platform Specific Model (PSM):** Combines the specifications defined in the PIM with the details required to stipulate how a system uses a particular type of platform

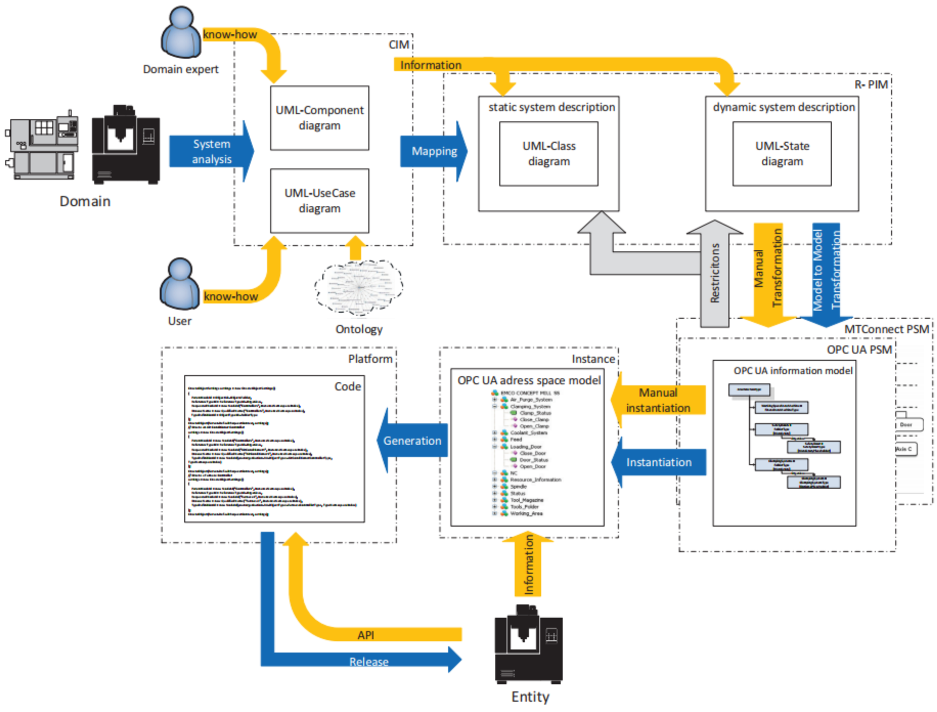


Figure 12: MDA approach to OPC UA information model design [7]

Special tools are often used for OPC UA information model design to generate code. The tools use different versions of XML-schemata. A modeler such as this was used in this project. These tools not only speed up the implementation, but they also increase the software quality by producing well structured and error-free code. As implementation is reduced to modeling and the code is generated automatically, even complex models can be implemented quickly. However, the use of such tools can lead to the developed information models not being compatible with other solutions than the specific system from which the model is created. A solution to this is the MDA approach illustrated in figure 12. The blue arrows show the suggested workflow creating a virtual representation of systems with OPC UA. The yellow arrows represent knowledge and actions done by humans [7].

In the modeler used in this project, and as required for OPC UA information modeling, each node in OPC UA has the mandatory attributes NodeId, NodeClass, BrowseName, and DisplayName. The use of this modeler and the steps that were done for the OPC UA communication in this project to comply with requirements can be found in section 3.5.

2.7 Visualization software

To give the user a visual representation of the DTw a variety of software is available. For a mathematical representation of the physical body of the system, three-dimensional (3D) models can be used. 3D models are widely used anywhere in 3D graphics and Computer-Aided Design (CAD). Here a model can be displayed as a two-dimensional image through a process called 3D rendering used in a computer simulation. Examples of software used for CAD is Simens NX, Solid Works, and AutoCAD.

The development of the DTws connectivity with the physical world is essential. This includes the input of, e.g. sensor data to be interpreted by the software to get the closed loop as discussed in section 2.2.1, and open the possibility for real-time monitoring. Software used for CAD has traditionally not been able to input sensor data to the 3D model and process simulation [83]. However, solutions as Siemens PLM are beginning to integrate this function through an OPC connection [84]. The software solutions considered for use in this project is described in the following.

2.7.1 Visual Components 4.0

Visual Components was founded in 1999 together with electronics automation company JOT Automation. Designed to provide a visualization tool for JOTs marketing, 3DRealize was launched in 2000. The software was made to enable robotics and material flow simulation on the same platform [85]. Visual Components is headquartered in Helsinki, Finland and has subsidiaries in Lake Orion, Michigan, and Munich, Germany, along with a global partner network of partners and resellers [86].

Figure 13: Visual Components [10]

Visual Components 4.0 (VC 4.0) is a commercial 3D discrete event simulation software that enables material flow and robotics simulation. The functionality can be extended with off-line programming capabilities and PLC connectivity [47]. Connectivity add-on includes OPC UA certificate generator [87]. The connectivity feature makes it possible to visualize real-time activity, collect and analyze real-time data, then test and simulate [88]. VC 4.0 also include physics behavior, powered by the NVIDIA PhysX engine, where it is possible to simulate and visualize functionality affected by physical forces, such as collisions, gravity, and material properties [89].

VC 4.0 has a library of pre-defined components. The software includes per July 2017 over 1200 robots from more than 30 of the largest brands in industrial automation [90]. Robotics has built-in features for robot jogging, analyzing reachability and collisions, and defining robot logic and postures with control flow statements. For custom CAD models it is possible to import files directly into the 3D world. Visual Components supports CAD file types from many of the leading CAD vendors, making it compatible with 3D models made by software from Autodesk, Dassault, PTC and Siemens [88].

VC 4.0 is built on the .NET framework with a Python API. This version also redefined and re-organized the core services layer, so it is more accessible and designed to allow for easy customization, from the UI to simulation behaviors [89]. Visual Components is used by manufacturing companies, systems integrators, and machine builders. Several companies are also using the software as a platform for their simulation software. Examples include the KUKASim software developed by KUKA and Octopuz developed by In-House Solutions [91].

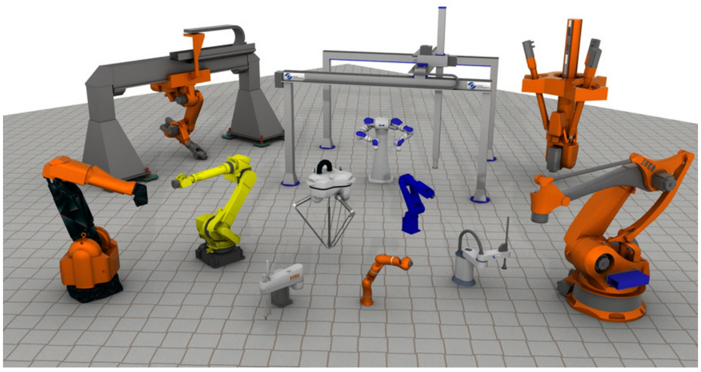


Figure 14: Some of the pre-defined components in VC 4.0 [11]

2.7.2 KUKA.Sim

KUKA.Sim is a program created by KUKA for creating 3D layouts with KUKA robots and is built on the Visual Components platform [92]. Like VC 4.0 the software includes comprehensive functions for designing components intelligently, for example:

KUKA

Figure 15: KUKA [12]

- Integration of I/O signals, sensors such as light barriers or similar physical features
- Conversion of geometries such as grippers, guns, and machine tools into kinematic systems

- I/O mapping for controlling components by way of signals

Like VC 4.0 KUKA.Sim also has an electronic catalogue with grippers, conveyors, safety fences and other components. Also, almost all recognized CAD formats can be imported [12]. The software uses graphical programming in a virtual environment much in the same way as VC 4.0.

KUKA.Sim Pro can be used for offline programming of KUKA robots and enables a real-time connection to the KUKA robot controller. The KUKA.Sim Pro package also includes OPC UA PLC interface for Beckhoff TwinCat, CODESYS or SIEMENS PLCSIM Advanced (TIA Portal) [92].

2.7.2.1 Visual Components 4.0 versus KUKA.Sim

As the KUKA.Sim software is built on Visual Components platform the two has a lot of the same functionality. The main difference is the restriction on the KUKA.Sim library to only contain KUKA products. As mentioned in section 2.7.1 the VC 4.0 library contains 30 of the largest brands.

Since 2010 KUKA has relied on EtherCAT technology as a system bus in all KUKA robot controllers. All internal communication and the communication to the lower-level I/O level take place via standard Ethernet or EtherCAT [93]. When it comes to the connectivity to OPC UA, KUKA on their web-pages advocates the standardization and openness of interfaces to allow all production “things” to communicate with the cloud via the same edge. To this end, KUKA promotes OPC UA as a standardized communication protocol [94]. Performing a comparison of VC 4.0 and KUKA.Sim regarding connectivity with OPC UA is however difficult due to this plug-in being a feature that is still under development.

2.7.3 Siemens SIMATIC WinCC

Alternative visualization software for use in a DTw is Siemens SIMATIC WinCC. This is a Supervisory Control and Data Acquisition (SCADA) and Human-Machine Interface (HMI) system from Siemens, with functions for controlling automated processes. SCADA systems are used to monitor and control physical processes involved in industry and infrastructure on a large scale and over long distances. SIMATIC WinCC can be used in combination with Siemens controllers [95].

With the SCADA system SIMATIC WinCC, Siemens offers scalable process visualization functions for monitoring automated processes whether in a single-user system or a distributed multi-user system with multiple servers. The system offers functionality for all industries and features openness. Siemens promotes this system as scalable, and offer maximum functionality and a user-friendly UI which gives the advantage of absolute openness to both the surrounding environment and production [13].

Figure 16: Conceptual representation of SIMATIC WinCC V7 SCADA [13]

SIMATIC WinCC is scalable for simple and complex tasks. Together with the integrated process database, the software represents information exchange cross-company. Through what Siemens call *Plant Intelligence* it provides transparency in production [96].

2.7.3.1 Visual Components 4.0 versus Siemens SIMATIC WinCC

In short, we define simulation as the production of data and visualization as the representation of data. While 3D visualization technology makes all of the elements look real, simulation technology makes them function as if they were real.

The data from simulation helps us to replicate and reproduce real-world operations of the elements which constitute the DTw. This makes it possible to emulate the different features, properties, and behaviors of a physical object. In the world of industry and production, this generates accurate information about utilization, performance, and overall effect on production over time.

Both VC 4.0 and SIMATIC WinCC possesses the ability to acquire and simulate data. However, it is a difference in the connectivity and how the data is presented. Three fundamental differences is presented in table 1.

Visual Components 4.0	**Siemens SIMATIC WinCC**
- Simulation and visualization in 3D	- Simulation and visualization in 2D GUI
- Support connectivity add-ons like OPC UA	- Not known to support connectivity add-ons like OPC UA
- Library of pre-defined components	- More need for customization of components

Table 1: Software comparison

An overall comparison of the use of VC 4.0 and SIMATIC WinCC shows that larger enterprises use the latter as a complete operation and management software with properties including digital planning and integrated engineering operation [97]. While this in theory also is possible in VC 4.0, this software is mostly used for simulation and visualization of production lines, with a lower threshold for doing changes in a system environment consisting of robots and components used in production.

2.8 Offline robot programming

Offline programming is a robot programming method that does not interfere with production as the program for the robot is written outside the production process on an external PC [98]. This shifts the work of programming the robot from the shop floor to the office. The benefits of this were mentioned in Njåstads thesis *Robotic welding with corrections from 3D-camera* [14]. Here it was also stated that it is beneficial to develop and configure a virtual model of a robot cell, providing the ability to simulate a process while doing offline programming. A DTw could constitute such a virtual model.

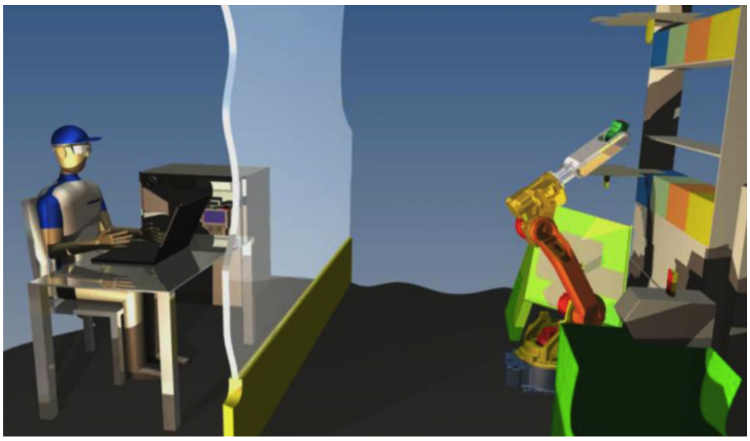


Figure 17: The offline programming concept [14]

Offline programming dramatically simplifies the work related to robot programming. The robot movements can in this environment be tested and simulated before being tried out on the physical robot. Offline programming software can detect Collisions, unwanted paths and cycle times. This can prevent downtime and reduce the cost associated to use of the physical robot system.

2.9 Robot Cell at MTP Valgrinda

The physical system used as the basis for the development of the DTw in this project is a robot cell at MTP Valgrinda consisting of the following:

- 1x Siemens PLC
- 2x KUKA KR 120 R2500 pro, robot manipulator
- 1x KUKA KR 16-2, robot manipulator
- 1x KUKA KR 5 arc, robot manipulator
- 4x KUKA KR C4, robot controller w/smartPAD
- 1x Fronius TransSteel 5000 welding machine

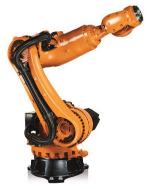


Figure 18: KUKA KR 120 R2500 pro [15]

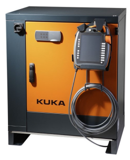


Figure 19: KUKA KR C4 [16]

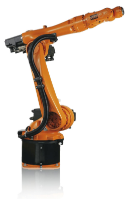


Figure 20: KUKA KR 5 arc [17]

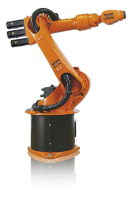


Figure 21: KUKA KR 16-2 [18]

The four KUKA robots are individually connected to their own KUKA KR C4 controller w/smartPAD. The controllers are then connected to a network together with a Siemens PLC. The smartPADs have touch-screen technology used for programming and controlling the robots manually. The Fronius TransSteel 5000 welding machine is connected to the controller of the KUKA KR 5 arc, with the welding torch on the robot manipulator.

The PLC has a variety of safety features to prevent damage to robots and people inside the robot cell. The PLC also has modules for digital and analogue I/O, used for controlling grippers, sensors and other signals and is accompanied by a stationary PC.

2.10 Robot software

The robot manufacturer traditionally makes the software used in KUKA robots such as the ones included in the cell at MTP Valgrinda. After the robots being delivered from the manufacturer, the robot is installed and programmed according to its environment. The programming and integration costs of such work can be more than half the up-front cost of a new robot cell [99]. The extended use for robots in academia and modern industry has created a demand for more accessible programming tools and software. Both software from the manufacturers, third party software from other companies, and open source software have in the later years evolved to satisfy these needs. Two different open source software solutions have been used in this project, KUKAVARPROXY and RSI.

2.10.1 KUKAVARPROXY

KUKAVARPROXY (KVP) is created by the IMTS S.r.L. Company [100], and was first released together with OpenShowVar on Sourceforge. Some sources claim that Massimiliano Fago was the person first to develop KVP and OpenShowVar in C++ [52]. KVP is a server to be run on the KUKA robot controller communicating with the internal system. OpenShowVar is a client that can be run externally to communicate with KVP over a TCP/IP connection. The KVP server supports a maximum of 10 simultaneous connections [26].

KVP runs as a background application on the robot controller and is able to communicate with the VxWorks side of the controller via KUKA CrossCom [101]. KVP has been used successfully in numerous research projects and has also been used in earlier projects in the robot cell at MTP Valgrinda by Bjørlykhaug and Raknes, and co-supervisor for this project, Eirik Njåstad.

KVP is, according to [100], said to be released on GitHub under an open license in the future.

2.10.2 KUKA RSI and kuka-rsi3-communicator

KUKA.RobotSensorInterface, or KUKA RSI, is an add-on technology package for KUKA controllers made by KUKA. This package has the following functions [19]:

1. Configuration of signal processing in the real-time system of the robot controller.
2. Influence on the robot motion or program execution by means of the signal processing.
3. Display of the signals via a monitor.

4. Configuration of real-time data exchange between the robot controller and an external system via Ethernet.
5. Influence on the robot motion or path planning by means of the data exchange via Ethernet.

The robot controller communicates with the sensor system via a field bus. The sensor data and signals are read by the field bus. KUKA RSI accesses the data and signals and processes them. If the data exchange is configured via Ethernet: the robot controller communicates with the external system via a real-time-capable network connection. The exchanged data are transmitted via the Ethernet TCP/IP or UDP/IP protocol as XML strings. Signal processing is established using RSI objects with a functionality and signal inputs and/or outputs as illustrated in figure 22.

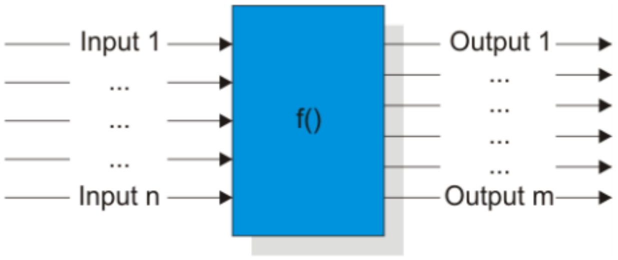


Figure 22: Schematic structure of an RSI object [19]

For message formatting and communication with KUKA RSI, it was used software based on the principles of an open-source KUKA RSI communicator. The software is called *kuka-rsi3-communicator*, is made by C. Eren Sezener, and available on GitHub. This communicator uses the UDP protocol for controlling KUKA robots manipulators in real-time. It communicates with the robot controller in a loop by sending XML strings through UDP. If the controller does not receive any strings, it gives a timeout error. It accepts position update commands through another socket; then sends the update commands to the robot controller in the correct XML format.

2.10.3 TCP versus UDP

Transmission Control Protocol (TCP) provides a reliable, connection-oriented service to the invoking application. User Datagram Protocol (UDP) provides an unreliable, connectionless service to the invoking application [102].

The two protocols are used in different situations depending on the data transmitted. TCP is used by devices that communicate over the Internet and local networks. It provides an ordered and error-checked stream of data packets. UDP is used by applications that require a faster stream of data. This is done by avoiding error-checking as illustrated in figure 23. Both of the protocols are built on top of the IP protocol.

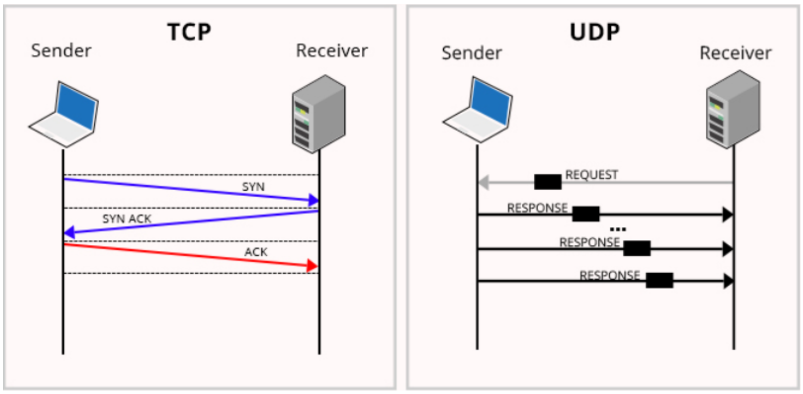


Figure 23: TCP versus UDP communication [20]

2.10.4 OPC Toolbox

OPC Toolbox provides access to live and historical OPC data directly from MATLAB and Simulink [103]. This software was not used in this project but could easily be used for extracting and analyzing data from an OPC UA server connected to a KR C4 robot controller. This toolbox includes read, write, and log OPC data from devices, such as distributed control systems, supervisory control and data acquisition systems, and programmable logic controllers. This simplifies working with data from live servers and data historians that conform to the OPC Data Access (DA) standard, the OPC Historical Data Access (HDA) standard, and the OPC Unified Architecture (UA) standard [103].

Using this toolbox opens up the possibility of doing simultaneous data logging and numerical processing, and simultaneous connections to multiple OPC servers. This makes it easier to access historical data for analysis and statistical processing. With the toolbox, it is possible to browse for available OPC UA servers and connect to an OPC UA server by creating an OPC UA Client object. Also, the toolbox provides functions that enable browsing and search of nodes in the server namespace to determine available data nodes. Interaction with multiple nodes at the same time is possible by creating an OPC UA node array. When reading the current value for a node or node array, the value, timestamp, and an estimate of the quality of the data is received, and it is possible to determine whether the data is a raw or interpolated value [104].

2.11 Robot kinematics

As the robot cell to be modeled contains four industrial robots, it is essential to have some understanding of how the robots operate and how they are represented in robot software. The theory presented on robot kinematics is mainly collected from *Robotics* by Siciliano, Sciavicco, Villani, and Oriolo [22].

A robot, often called a *manipulator*, can be schematically represented from a mechanical viewpoint as a kinematic chain of rigid bodies (*links*) connected by means of revolute or prismatic *joints*. This is illustrated in figure 24. One end of the chain is constrained to a base, while an *end-effector* often is mounted to the end-of-arm.

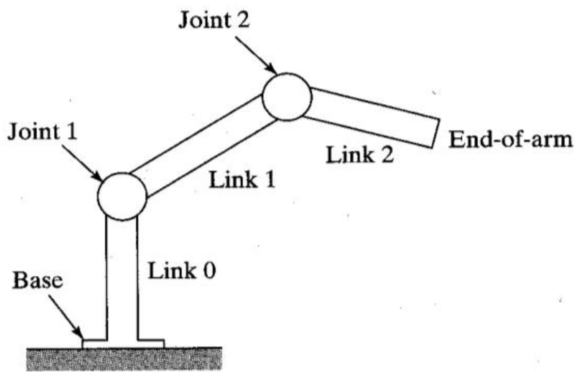


Figure 24: Joints and links of robot [21]

The resulting motion of the structure is obtained by composition of the elementary motions of each link with respect to the previous one. Often it is necessary to describe the position and orientation (*pose)* of the end-effector as a function of the joint variables of the mechanical structure with respect to a reference frame. This is called the *direct kinematics equation.* Determining the joint variables corresponding to a given end-effector pose is called *the inverse kinematics problem.*

A *rigid body* is completely described in space by its *position* and *orientation* with respect to a reference frame. This is illustrated in figure 25. Let $O - xyz$ be the orthonormal reference frame and $\mathbf{x}, \mathbf{y}, \mathbf{z}$ be the unit vectors of the frame axes. The position of a point O' on the rigid body with respect to the coordinate frame $O - xyz$ is expressed by the relation:

$$\mathbf{o}' = o'_x\mathbf{x} + o'_y\mathbf{y} + o'_z\mathbf{z} \tag{2.1}$$

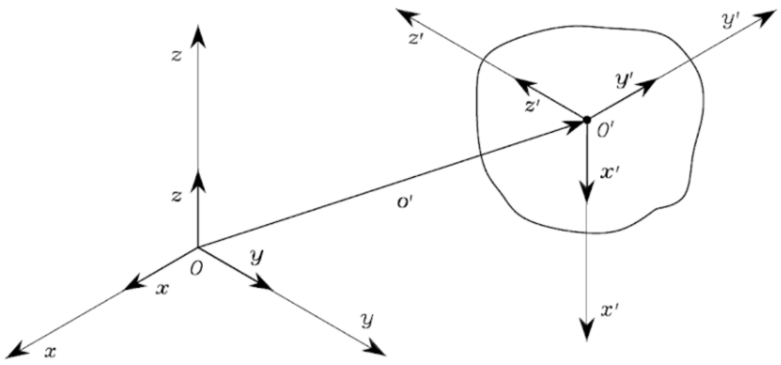


Figure 25: Position and orientation of a rigid body [22]

The position of O' can be compactly written as the bound vector:

$$\mathbf{o}' = \begin{bmatrix} o'_x \\ o'_y \\ o'_z \end{bmatrix} \tag{2.2}$$

To describe the rigid body rotation, we use an orthonormal frame attached to the body and express its unit vectors with respect to the reference frame. Let $O' - x'y'z'$ be such a with O' as the origin and $\mathbf{x}', \mathbf{y}', \mathbf{z}'$ as the unit vectors of the frame axes. The vectors could then be expressed with respect to the reference frame $O - xyz$ by the equations:

$$\begin{aligned} \mathbf{x}' &= x'_x\mathbf{x} + x'_y\mathbf{y} + x'_z\mathbf{z} \\ \mathbf{y}' &= y'_x\mathbf{x} + y'_y\mathbf{y} + y'_z\mathbf{z} \\ \mathbf{z}' &= z'_x\mathbf{x} + z'_y\mathbf{y} + z'_z\mathbf{z} \end{aligned} \tag{2.3}$$

By adopting the compact notation these three unit vectors, describing the body orientation with respect to the reference frame can be combined in a *Rotation matrix*:

$$\mathbf{R} = \begin{bmatrix} \mathbf{x}' & \mathbf{y}' & \mathbf{z}' \end{bmatrix} \tag{2.4}$$

Elementary rotations about the x, y and z coordinate axis are described by the rotation matrices $\mathbf{R}_z(\alpha)$, $\mathbf{R}_y(\beta)$, $\mathbf{R}_x(\gamma)$:

$$\mathbf{R}_z(\alpha) = \begin{bmatrix} \cos\alpha & -\sin\alpha & 0 \\ \sin\alpha & \cos\alpha & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{2.5}$$

$$\mathbf{R}_y(\beta) = \begin{bmatrix} \cos\beta & 0 & \sin\beta \\ 0 & 1 & 0 \\ -\sin\beta & 0 & \cos\beta \end{bmatrix} \tag{2.6}$$

$$\mathbf{R}_x(\gamma) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\gamma & -\sin\gamma \\ 0 & \sin\gamma & \cos\gamma \end{bmatrix} \tag{2.7}$$

Here the unit vectors are:

$$\mathbf{x}' = \begin{bmatrix} \cos\alpha \\ \sin\alpha \\ 0 \end{bmatrix}, \qquad \mathbf{y}' = \begin{bmatrix} -\sin\alpha \\ \cos\alpha \\ 0 \end{bmatrix}, \qquad \mathbf{z}' = \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} \tag{2.8}$$

A rotation about the z-axis with an angle α is illustrated in figure 26.

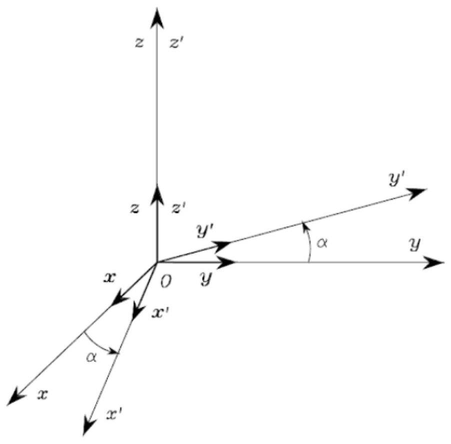


Figure 26: Rotation of frame $O - xyz$ by an angle α about axis z [22]

The coordinate transformation of a vector with the coordinate $\mathbf{v}^A$ in reference frame A to the vector $\mathbf{v}^B$ in reference frame B is then given by:

$$\mathbf{v}^A = \mathbf{R}_B^A \mathbf{v}^B \tag{2.9}$$

With $\mathbf{R}_B^A \in SO(3)$ being the rotation matrix from A to B.

2.11.1 KUKA convention

KUKA robots use A, B, C as Euler angles about the current axis. Euler angles are a representation of orientation in terms of three independent parameters, constituting a *minimal representation*:

$$\phi = [\varphi \quad \vartheta \quad \psi]^T = [\mathbf{A} \quad \mathbf{B} \quad \mathbf{C}]^T \tag{2.10}$$

A = rotation about z-axis (Roll)
B = rotation about y-axis (Pitch)
C = rotation about x-axis (Yaw)

This results in the rotation matrix:

$$\mathbf{R} = \mathbf{R}_z(A)\mathbf{R}_y(B)\mathbf{R}_x(C) \tag{2.11}$$

Roll-Pitch-Yaw is illustrated in figure 27.

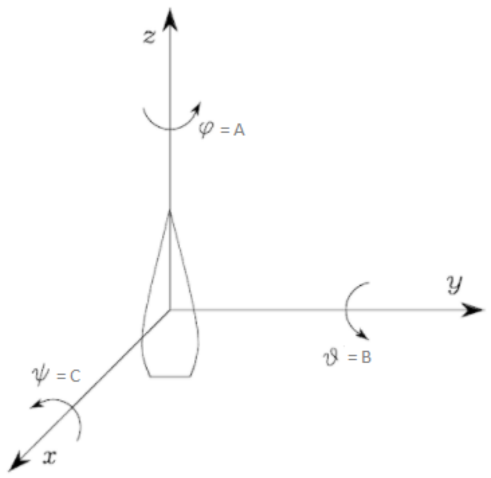


Figure 27: Representation of Roll-Pitch-Yaw angles

Using the specific information on a KUKA robot's links and joints together with Roll-Pitch-Yaw angles, a kinematic structure can be generated. This is illustrated in figure 28 and figure 29.

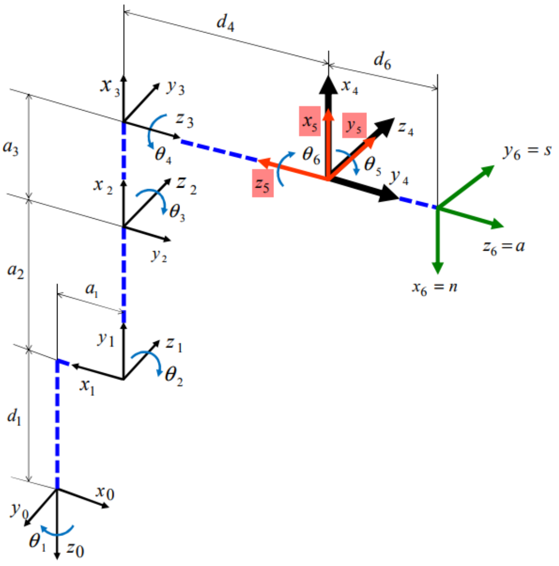


Figure 28: Kinematic chain for KUKA KR family [23]

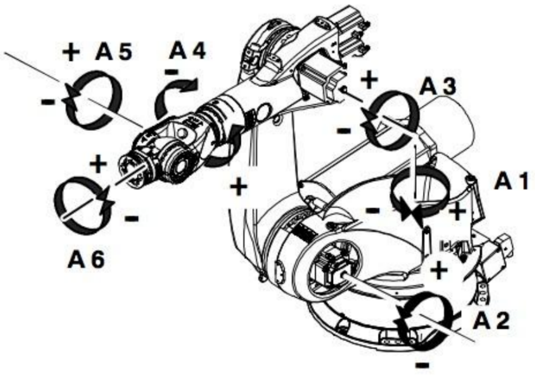


Figure 29: KUKA KR family joints rotation directions [23]

A KUKA robot consists of a series of links connected through joints. Joints can be essentially of two types: *revolute* and *prismatic.* These two principles are illustrated in figure 30. The whole structure forms a *kinematic chain.*

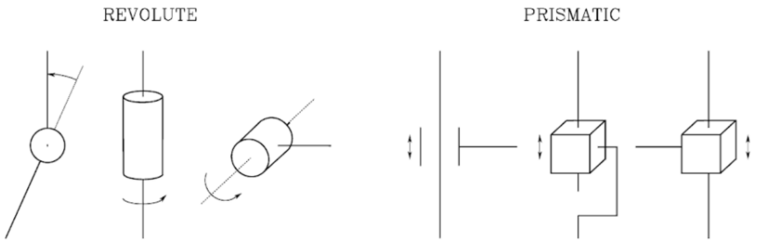


Figure 30: Representation of joints [24]

The mechanical structure of a manipulator is characterized by a number of degrees of freedom (DOFs) which uniquely determine its *posture.* Each joint is associated with a joint articulation and constitutes a *joint variable.* The four KUKA KR robots in the robot cell at MTP Valgrinda all have 6 revolute joints, making them robots with 6DOF.

Calculating the pose of the end-effector as a function of the joint variables is denoted *direct kinematics.* The direct kinematics function is often expressed by the homogeneous transformation matrix:

$$\mathbf{T}_e^b(\mathbf{q}) = \begin{bmatrix} \mathbf{n}_e^b(\mathbf{q}) & \mathbf{s}_e^b(\mathbf{q}) & \mathbf{a}_e^b(\mathbf{q}) & \mathbf{p}_e^b(\mathbf{q}) \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{2.12}$$

Where $\mathbf{q}$ is the $(n \times 1)$ vector of joint variables, $\mathbf{n}_e, \mathbf{s}_e, \mathbf{a}_e$ are the unit vectors of a frame attached to the end-effector, and $\mathbf{p}_e$ is the position vector of the origin of such a frame with respect to the origin to the base frame $O_b - x_b y_b z_b$. This is illustrated in figure 31.

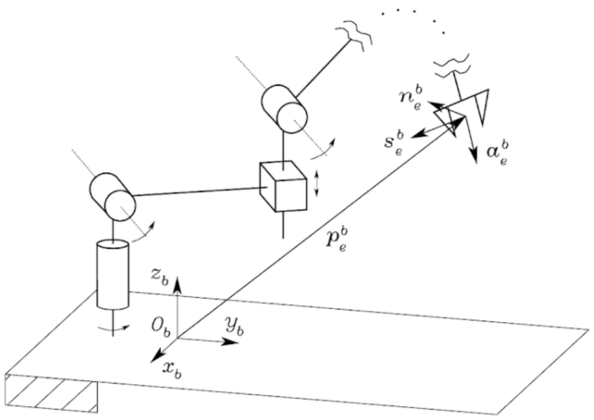


Figure 31: Description of position and orientation of the end-effector frame [22]

When describing a homogeneous transformation, the *Denavit-Hartenberg convention* is often used. The convention is based on describing a transformation as a composite displacement of 4 homogeneous transformation matrices, starting with a rotation θ about the z-axis and then a translation d along the same axis. This is followed by a rotation α about the x-axis and then a translation a along the same axis. This gives a homogeneous transformation matrix that is described in terms of the 4 parameters θ, d, α and a. These parameters are called the Denavit-Hartenberg parameters of the displacement. The result is a displacement given by:

$$\mathbf{T}_i^{i-1} = \mathbf{Rot}_z(\theta_i)\mathbf{Trans}_z(d_i)\mathbf{Rot}_x(\alpha_i)\mathbf{Trans}_x(a_i) \tag{2.13}$$

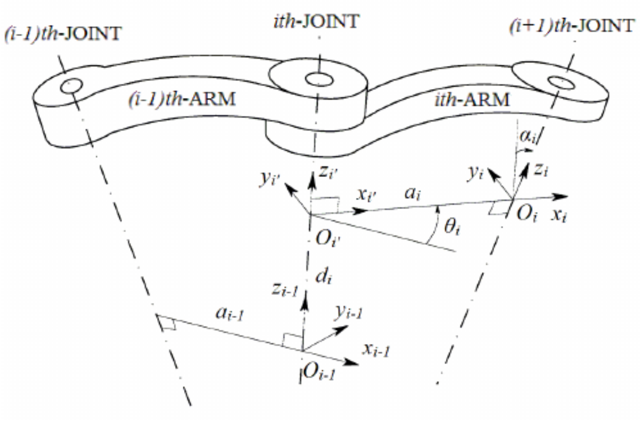


Figure 32: Denavit-Hartenberg parameters [24]

Chapter 3

Method and Solutions

In section 2.1 it was stated that the journey of getting to I 4.0 should start with the following steps:

1. A reference architecture to provide a technical description of standards and facilitate their implementation.

2. Engineers equipped with the methods and tools required to develop appropriate planning and explanatory models.

- *The Industrie 4.0 Working Group, 2013*

This chapter will present the method used to develop a DTw for the robot cell at MTP Valgrinda, and the solutions found along the way. It includes the selection of software to be used, the development of a virtual representation of the robot cell and the development of software capable of combining the virtual representation and the physical robot cell. The final part of this chapter is a description of a fixed case that was performed to test the system in action.

3.1 Selection of simulation software

Goal c) of this project listed in section 1, was to *Use Visual Components 4.0 or similar simulation software to model and simulate the robot cell at the institute.* For this task three simulation and visualization software packages were considered:

1. VC 4.0
2. KUKA.Sim
3. Siemens SIMATIC WinCC

These three are described and compared in section 2.7. As the DTw of the robot cell should be implemented with OPC UA, connectivity to this platform was essential when choosing simulation and visualization software. Using OPC UA for communication is supported by this being the only recommendation for realizing the communication layer of RAMI 4.0 as stated in section 2.5.

Access to pre-defined components was also an essential factor. Both VC 4.0 and KUKA.Sim includes all the KUKA components in the robot cell at MTP Valgrinda. This saves time in the development of the DTw were every component needs to be modeled with associated features.

After comparison of VC 4.0 to Siemens SIMATIC WinCC, summarized in table 1, it was decided that Siemens SIMATIC WinCC would not be suitable for use in this project. As presented in the comparison of VC 4.0 and KUKA.Sim in section 2.7.2.1, these two are very similar when it comes to functionality and use. Both satisfied the conditions that were necessary for use in this project. However, the updated version of KUKA.Sim, KUKA.Sim 4.0, was not available for use during this project.

Solution: Visual Components 4.0 will be used for modeling and simulation

Figure 33: Visual Components [10]

3.2 Making a virtual representation

With the simulation software decided the development platform for the DTw was set. The first step that was done to make the actual DTw was to make a virtual representation of the robot cell that would work as the digital face of the simulations and give a visual representation to the user.

3.2.1 Modelling the robot cell

To make the model as accurate and detailed as possible, it was done several measurements of the physical robot cell. However, since all of this was done by hand and without any form or digital equipment there was a significant uncertainty related to the measurements. With the use of the average value of three measurements, the uncertainty was estimated to be $\pm 1cm$. An elaboration of the various uncertainties can be found in section 5.5.

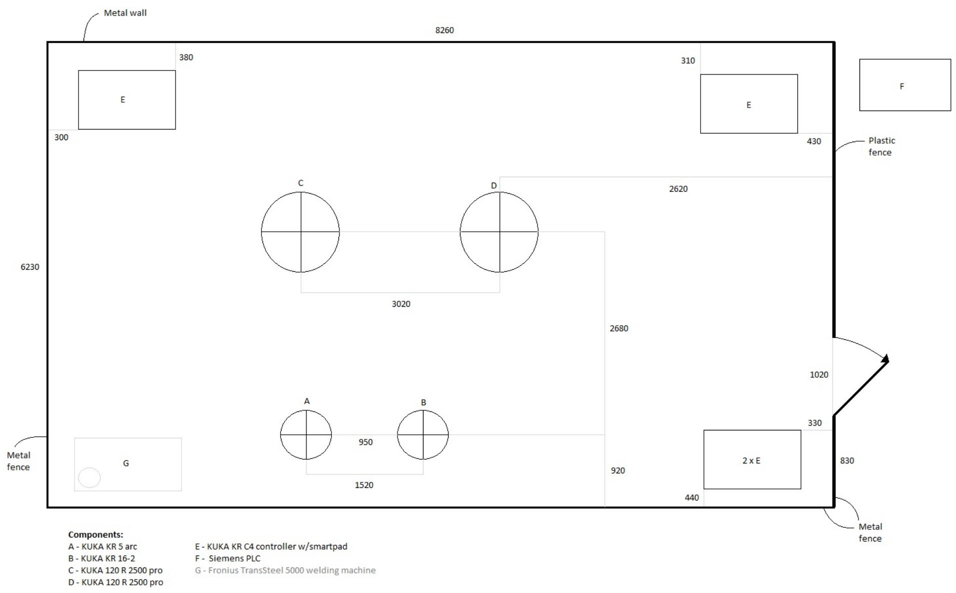


Figure 34: First sketch of robot cell

The significant uncertainty related to the digital model was accepted because the only critical measurements of the system were the measurements of the distance between the KUKA robots and their orientation in the cell. The orientation of the robots would, in any case, be needing a calibration for the robots to be able to work together with sufficient precision. This would come at a later stage as this step just included a first rough model of the system.

After the initial measurements were done, a 3D model based on these measurements was created using VC 4.0. First, a model of the robot cell environment was made as illustrated in figure 35.

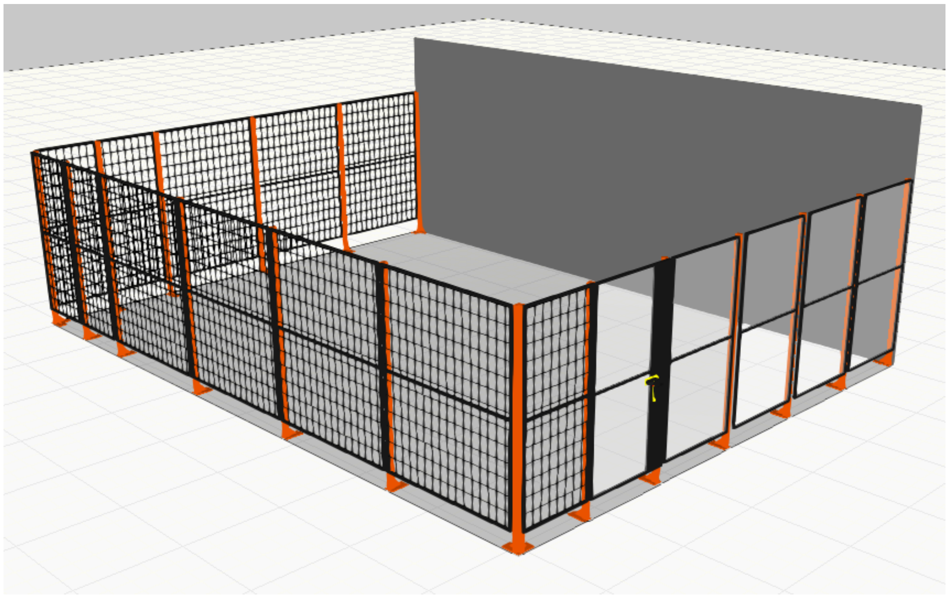
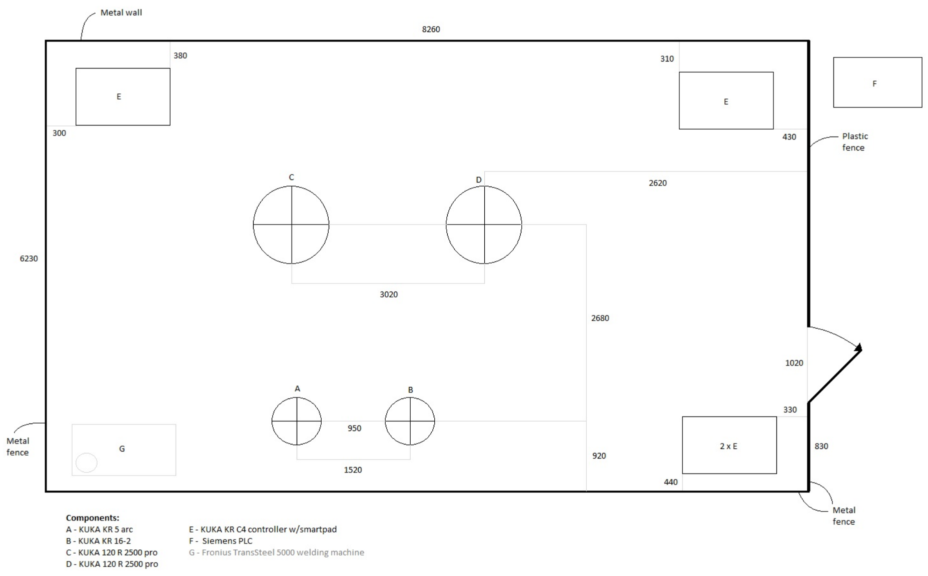
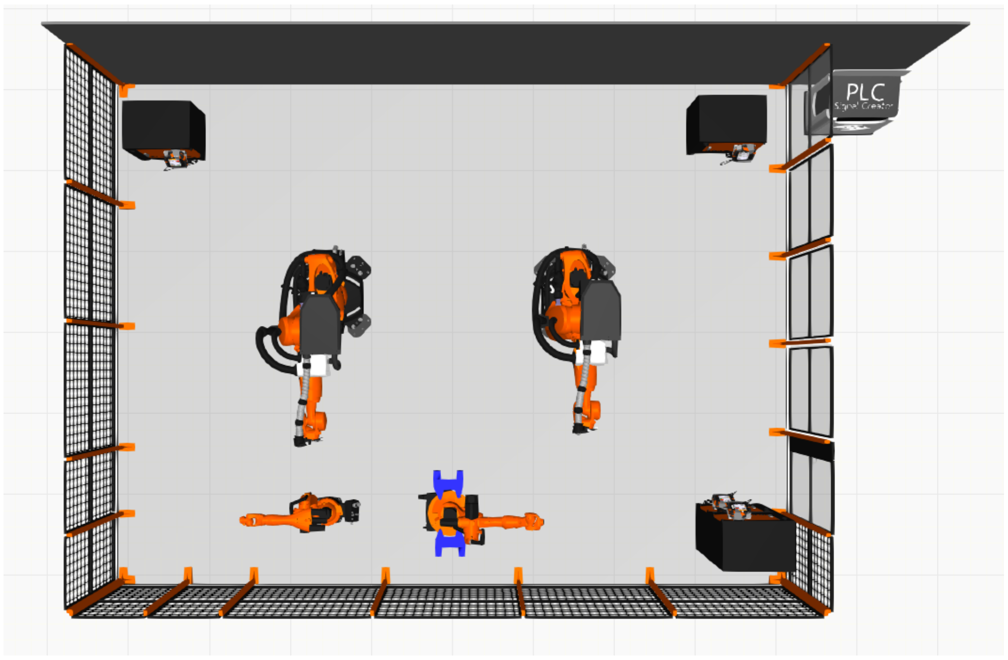


Figure 35: Model of robot cell environment, made in VC 4.0

The second stage was to include the components inside the robot cell. Since this DTw is a first version focused around the KUKA robots and their controllers, the Fronius TransSteel 5000 welding machine was excluded as a part of the digital representation of this robot cell. This could be included in a later version. For insertion of the KUKA robots and their controllers, the advantages with the VC 4.0 software were evident. The pre-defined library included all these components including their associated features. Inserting these components in the environment resulted in the final model shown in comparison with the first sketch in figure 36 and the physical robot cell in figure 37.

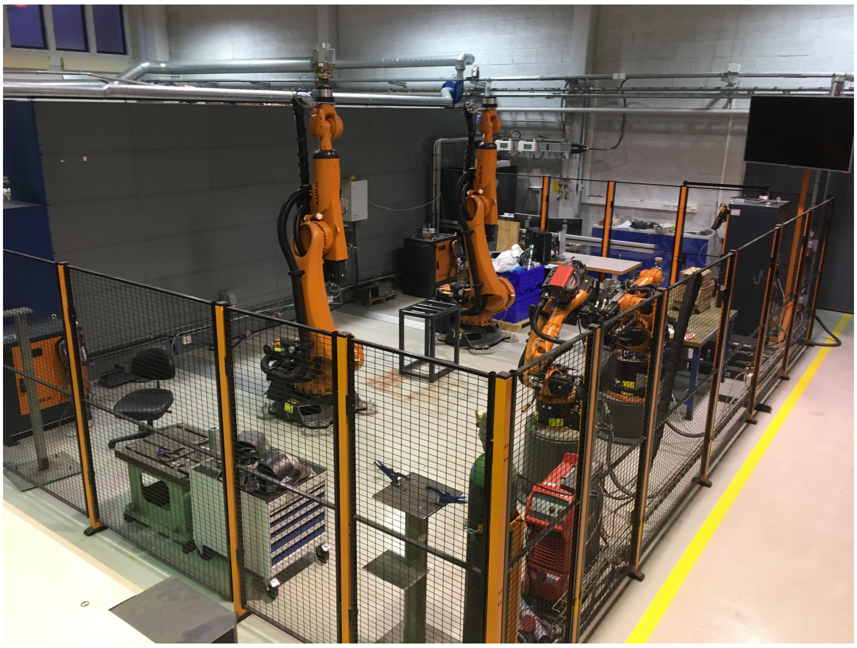


(a) First sketch



(b) Final model from VC 4.0

Figure 36: Comparison of first sketch and final model of the robot cell


(a) Physical

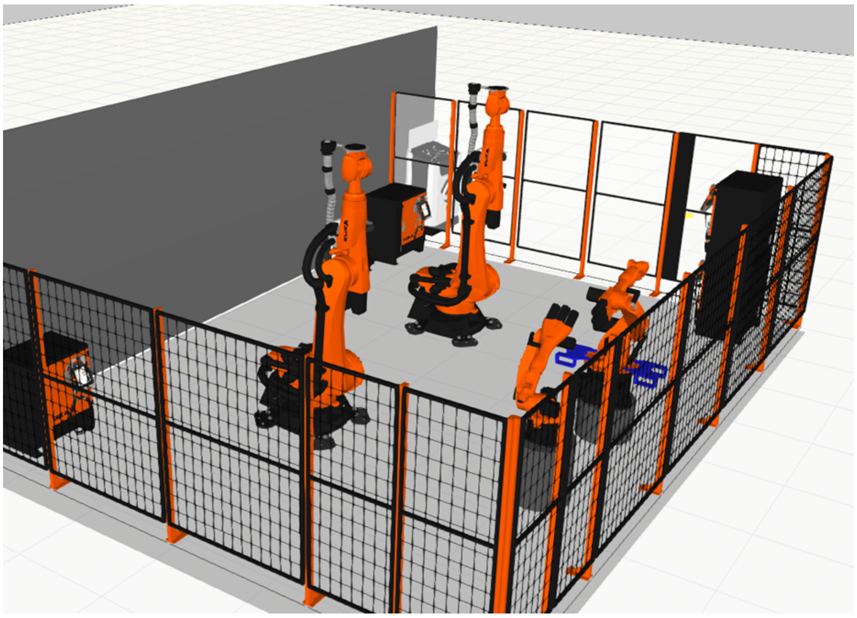

(b) Final model from VC 4.0

Figure 37: Comparison of the physical and final model of the robot cell

3.3 Simulating robot cell

Now the visual representation of the robot cell was made it was time to connect the twins and make them communicate. OPC UA would be used for the communication between the physical robot cell and the visual representation.

3.3.1 Communication

The basis for the communication is built on the principle that the KUKA KR C4 robot controller works as a server and the computer which is running the DTw works as a client. This is in compliance with the OPC UA standard illustrated in figure 8 in section 2.3.

The KUKA KR C4 robot controllers used in the robot cell at MTP Valgrinda did not have OPC UA server functionality implemented as a standard. Thus, this functionality had to be implemented as an add-on. After some research, it turned out KUKA has made an add-on technology package named KUKA.OPC Server 4.1. This server supports OPC Classic which was described in section 2.3. It makes it possible to communicate and exchange data with the robot controller from a local or remote OPC client. This includes features like [25]:

- Reading and writing of system and program variables of the industrial robot
- Reading of robot controller messages
- Reading of keys from the registration database
- Reading of variables from other OPC servers via the proxy interface

Without OPC, two machines or applications must know each other's communications options in order to exchange data. This is illustrated in figure 38.

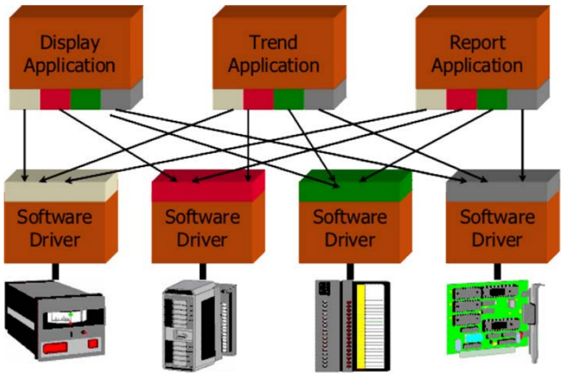


Figure 38: Communication without OPC [25]

With OPC, it suffices to configure an OPC-compliant driver for each machine or application. All programs that can request data as OPC clients work together with the OPC server. This simplifies and standardizes access. This is illustrated in figure 39.

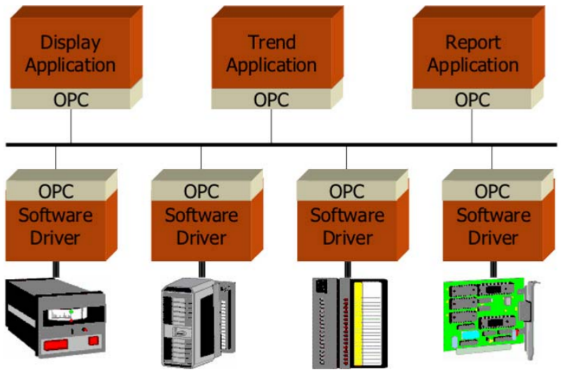


Figure 39: Communication with OPC [25]

However, KUKA.OPC Server 4.1 is not known to support OPC UA connectivity which was one of the requirements in this project. This feature could possibly be implemented in a future version of the software. Also, this add-on technology does not have source code that is available online. As a reference, it was tried to get a hold of this software though contacting KUKA. This is further discussed in section 5.3. KUKA did not provide this software, and a reference for comparing OPC Classic to OPC UA was therefore not available.

In previously completed projects in the robot cell at MTP Valgrinda, an open-source communication interface to KUKA KR C4 controllers has been used named JOpenShowVar and KUKAVARPROXY (KVP). This is mentioned by among others Bjørlykhaug, Raknes on their project on *Robotic welding without fixture* supervised by professor Olav Egeland and Eirik Njåstad, and Bredvold on his master thesis *Robotic welding of Tubes with Correction from 3D Vision and Force Control* also supervised by professor Olav Egeland [105]. Co-supervisor for this master thesis, Eirik Njåstad, also used this interface in his master thesis *Robotic welding with correction from 3D camera* [14].

3.3.2 JOpenShowVar and KUKAVARPROXY

JOpenShowVar is an open-source communication interface for KUKA KR C4 robot controllers developed for cross-platform compatibility. This interface is built on OpenShowVar, introduced together with KVP in section 2.10.1, and further developed by F. Sanfilippo, L.I. Hatledal and H. Zhang in [26] to be used in Java, thus the name JOpenShowVar. KVP is the TCP/IP server that runs on the robot controller, while OpenShowVar/JOpenShowVar is the client which connects to the server. This architecture is illustrated in figure 40. The use of JOpenShowVar gives the possibility of reading and writing global variables on the robot controller from any remote device. This opens up the possibility to develop controller applications on hardware such as computers, smartphones and tablets [105]. This interface, which is compatible with all KUKA robots that use KR C4 and previous versions, runs as a client on a remote computer connected with the controller [26].

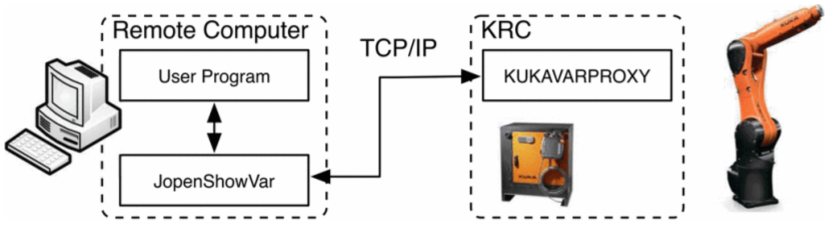


Figure 40: Client-Server architecture [26]

The reason for the use of JOpenShowVar is that the majority of industrial manipulators does not have an open control interface. For KUKA, the standard programming language is the KUKA Robot Language (KRL). This language is text-based and offers the possibility of declaring data types, specifying simple motions, and interacting with tools and sensors via I/O operations. A KRL program can only run on a KUKA Robot Controller, in this robot cell the KR C4, where it is executed according to real-time constraints. The KRL is very limited when it comes to research purposes. This is because the KRL is custom made to the underlying controller and, as a consequence, it only offers a fixed, controller-specific set of instructions. There is no mechanism for including third-party libraries. Due to this design, it is very difficult to extend the KRL with new instructions and functionalities and no external input devices can be directly used [26].

The standard workaround for expanding the robot's capabilities consists of using additional software packages provided by KUKA. Some examples of such packages are the KUKA.RobotSensorInterface (KUKA RSI), which makes it possible to influence the manipulator motion or program execution via sensor data, and KUKA.Ethernet KRL XML, a module that allows the robot controller to be connected with external systems. However, these additional software packages have several drawbacks such as limited I/O, a narrow set of functions and often require major capital investments [26]. None of these were available for use in this project.

JOpenShowVar and KVP work as a middleware between the user program and the KRL and thus opens a backdoor for an alternative implementation of an OPC UA communication on the KR C4 controllers. The proposed communication architecture is illustrated in figure 41.

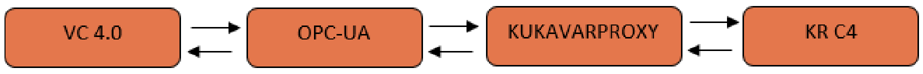


Figure 41: Proposed communication architecture

For this to work, KVP has to run on each robot controller in the cell. This was done in the project by Bjørlykhaug, Raknes by merely transferring the open-source files to the KUKA smartPADs. The program was put in the startup folder in the Windows XP environment, so it will always start and run when resetting the robot controller. The version of KVP used in this project can be found in the forked GitHub repository `https://github.com/akselov/JOpenShowVar` together with a detailed description of how to set up JOpenShowVar and KVP made by the Mechatronics Lab at Ålesund University College [106].

When KVP is running, the Java code needed in order to connect is:

```java
CrossComClient connection = new CrossComClient(ipAddress, port);
```

This piece of code creates a new Thread, meaning the control system will be a multi-threaded system, with a thread for each connection. To read and write to the robot controllers, CrossComClient has two principal methods, readVariable (KRLVariable) and writeVariable (KRLVariable), where a KRLVariable can be any global variable on the robot controller [105].

3.3.3 Setting up JOpenShowVar and KUKAVARPROXY

As mentioned in section 3.3.2 KVP already was running on all of the four robot controllers in the cell before this project. Therefore, getting a connection from the client side with JOpenShowVar was the first step that had to be done for the communication to be established. JOpenShowVar was compiled using Eclipse to a .jar file that could be run from the Windows PC hosting VC 4.0 and containing the DTw.

This connection was successfully established through command prompt (cmd), and variables were successfully read and written to the robot controllers through this connection. Figure 42 show screenshots of initial connection, reading and writing variables from the client side to the KR C4 robot controller controlling the KR 16-2.

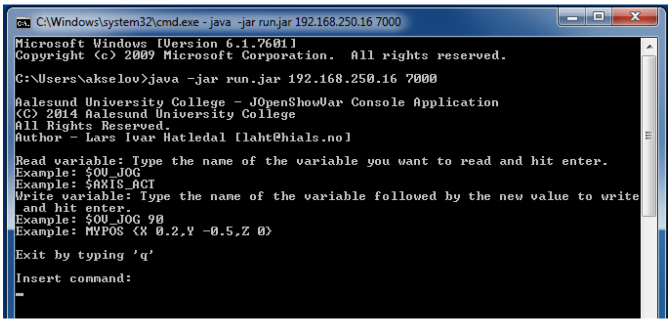
A goal of this project listed in section 1, was to *Present a solution for a digital twin of the Institute's Robot Laboratory with use of OPC UA for communication.* The variables that were needed from the robot controllers were now successfully being communicated from the controller, but this communication did not satisfy the requirements of the OPC UA client running in VC 4.0. To fulfill the proposed communication system illustrated in figure 41 it was needed an open source OPC UA server that could work together with JOpenShowVar extracting the desired robot variables.

3.3.4 Setting up a OPC UA connection

Olivier Roulet-Dubonnet, a Senior Research Scientist at SINTEF Raufoss Manufacturing AS department of Production Technology [107], has made two free open source OPC UA server and client libraries. One in C++ named *FreeOpcUa* and one in Python named *python-opcua*. Both are available at `https://github.com/FreeOpcUa`.

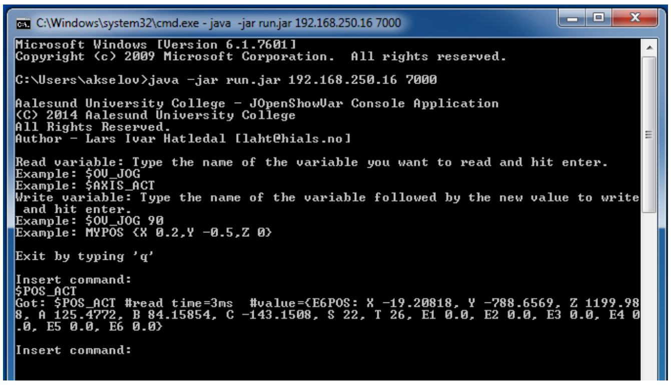
The Python library is written with much code auto-generated from XML specification. The OPC UA server is currently supporting [108]:

- Creating channel and sessions
- Read/set attributes and browse
- Getting nodes by path and nodeIDs
- Auto-generate address space from spec
- Adding nodes to address space
- Data-change events
- Events


C:\Windows\system32\cmd.exe - java -jar run.jar 192.168.250.16 7000

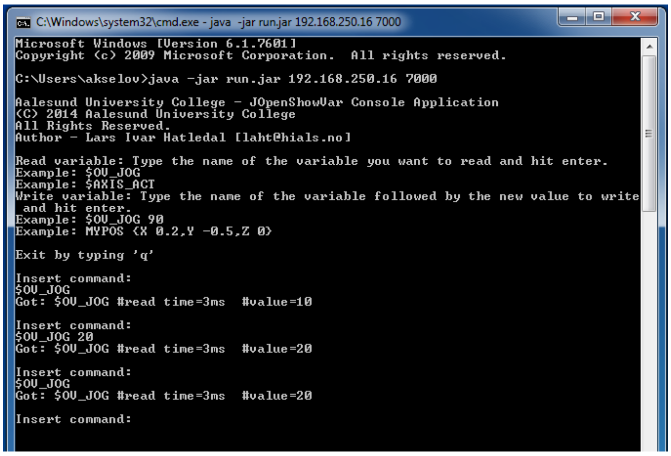
```
Microsoft Windows [Version 6.1.7601]
Copyright (c) 2009 Microsoft Corporation.  All rights reserved.

C:\Users\akselov>java -jar run.jar 192.168.250.16 7000

Aalesund University College - JOpenShowVar Console Application
(C) 2014 Aalesund University College
All Rights Reserved.
Author - Lars Ivar Hatledal [laht@hials.no]

Read variable: Type the name of the variable you want to read and hit enter.
Example: $OV_JOG
Example: $AXIS_ACT
Write variable: Type the name of the variable followed by the new value to write
 and hit enter.
Example: $OV_JOG 90
Example: MYPOS {X 0.2,Y -0.5,Z 0}

Exit by typing 'q'

Insert command:
_
```

(a) Communication between KR C4 and PC established


C:\Windows\system32\cmd.exe - java -jar run.jar 192.168.250.16 7000

```
Microsoft Windows [Version 6.1.7601]
Copyright (c) 2009 Microsoft Corporation.  All rights reserved.

C:\Users\akselov>java -jar run.jar 192.168.250.16 7000

Aalesund University College - JOpenShowVar Console Application
(C) 2014 Aalesund University College
All Rights Reserved.
Author - Lars Ivar Hatledal [laht@hials.no]

Read variable: Type the name of the variable you want to read and hit enter.
Example: $OV_JOG
Example: $AXIS_ACT
Write variable: Type the name of the variable followed by the new value to write
 and hit enter.
Example: $OV_JOG 90
Example: MYPOS {X 0.2,Y -0.5,Z 0}

Exit by typing 'q'

Insert command:
$POS_ACT
Got: $POS_ACT #read time=3ms  #value={E6POS: X -19.20818, Y -788.6569, Z 1199.98
8, A 125.4772, B 84.15854, C -143.1508, S 22, T 26, E1 0.0, E2 0.0, E3 0.0, E4 0
.0, E5 0.0, E6 0.0}

Insert command:
```

(b) Reading the axis variables from robot


C:\Windows\system32\cmd.exe - java -jar run.jar 192.168.250.16 7000

```
Microsoft Windows [Version 6.1.7601]
Copyright (c) 2009 Microsoft Corporation.  All rights reserved.

C:\Users\akselov>java -jar run.jar 192.168.250.16 7000

Aalesund University College - JOpenShowVar Console Application
(C) 2014 Aalesund University College
All Rights Reserved.
Author - Lars Ivar Hatledal [laht@hials.no]

Read variable: Type the name of the variable you want to read and hit enter.
Example: $OV_JOG
Example: $AXIS_ACT
Write variable: Type the name of the variable followed by the new value to write
 and hit enter.
Example: $OV_JOG 90
Example: MYPOS {X 0.2,Y -0.5,Z 0}

Exit by typing 'q'

Insert command:
$OV_JOG
Got: $OV_JOG #read time=3ms  #value=10

Insert command:
$OV_JOG 20
Got: $OV_JOG #read time=3ms  #value=20

Insert command:
$OV_JOG
Got: $OV_JOG #read time=3ms  #value=20

Insert command:
```

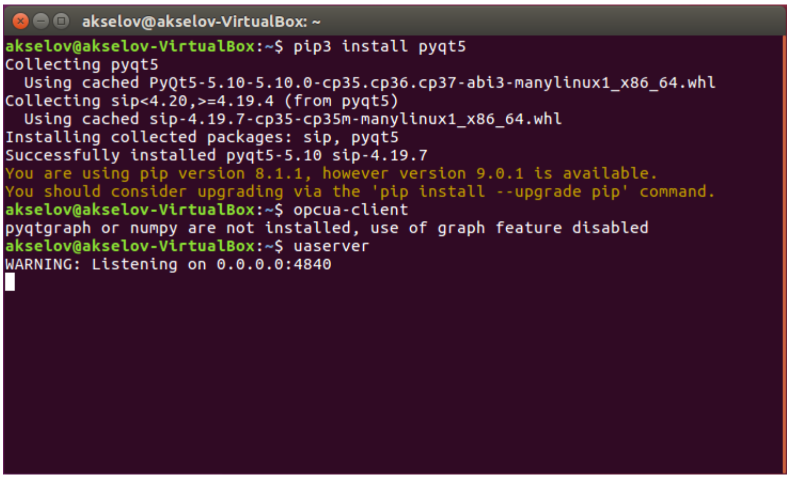
(c) Writing the controllers jog parameter

Figure 42: Screenshot of cmd from PC containing the DTw

- Methods
- Basic user implementation
- Encryption
- Certificate handling
- Removing nodes
- History support for data change and events

OPC UA is said to be free, but doing some research, it was found that it is still difficult to find a sufficient open source library. FreeOpcUa by Roulet-Dubonnet was therefore chosen to be the foundation for the OPC UA communication in this DTw. The version of the *python-opcua* library used in this project can be found in the forked GitHub repository `https://github.com/akselov/python-opcua`.

After downloading this library, an initial client-server communication was set up using Ubuntu Linux. Figure 43 shows a screenshot of the example server in the library, *uaserver*, running and listening to 0.0.0.0:4840 which is all IPv4 addresses on the local machine [109], port 4840.



```
akselov@akselov-VirtualBox:~$ pip3 install pyqt5
Collecting pyqt5
  Using cached PyQt5-5.10-5.10.0-cp35.cp36.cp37-abi3-manylinux1_x86_64.whl
Collecting sip<4.20,>=4.19.4 (from pyqt5)
  Using cached sip-4.19.7-cp35-cp35m-manylinux1_x86_64.whl
Installing collected packages: sip, pyqt5
Successfully installed pyqt5-5.10 sip-4.19.7
You are using pip version 8.1.1, however version 9.0.1 is available.
You should consider upgrading via the 'pip install --upgrade pip' command.
akselov@akselov-VirtualBox:~$ opcua-client
pyqtgraph or numpy are not installed, use of graph feature disabled
akselov@akselov-VirtualBox:~$ uaserver
WARNING: Listening on 0.0.0.0:4840
```

Figure 43: OPC UA server running in Linux

For the client side, it was used a simple GUI client also made by Olivier Roulet-Dubonnet and published in the FreeOpcUa library. This client is named *opcua-client-gui*, and the version used in this project can be found in the forked GitHub repository `https://github.com/akselov/opcua-client-gui`. Figure 44 shows the GUI connected to localhost, port 4840.

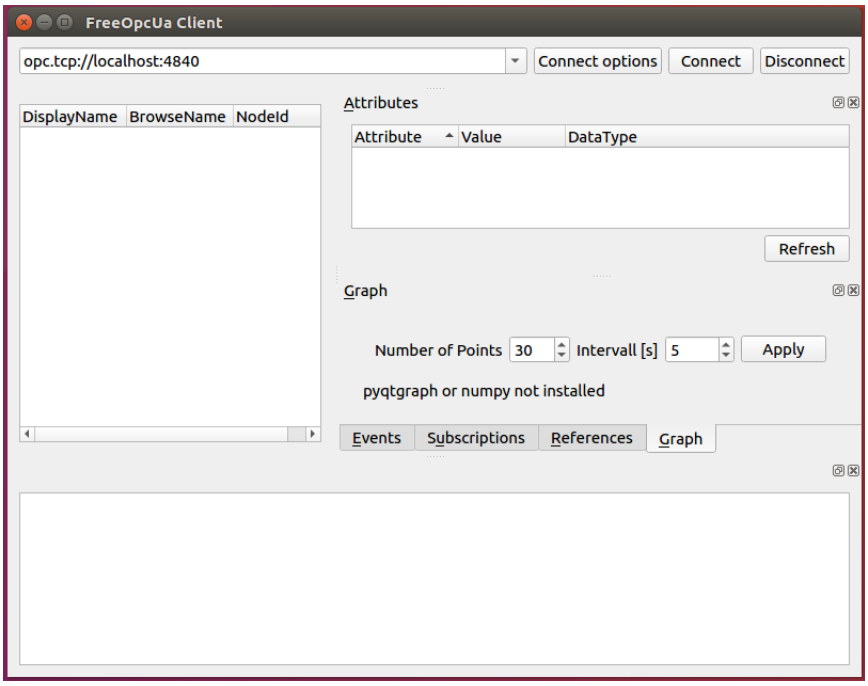
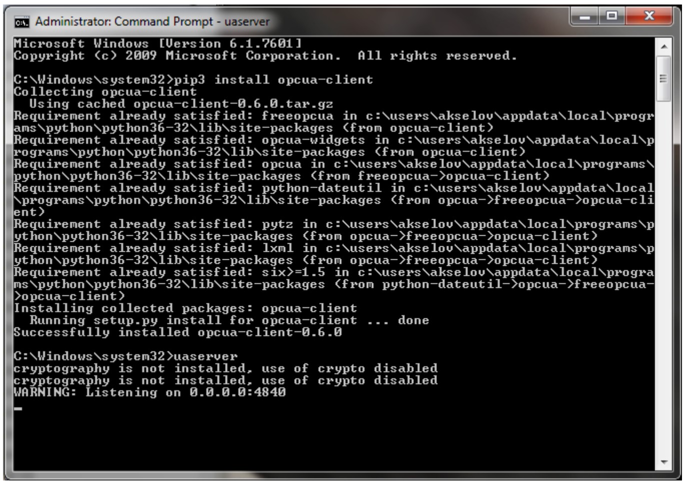


Figure 44: OPC UA client GUI running in Linux

After successfully establishing the client-server connection in Linux, the next step was to implement a similar client-server connection in Windows with VC 4.0 working as the OPC UA client. Using PIP, a package management system used to install and manage software packages written in Python [110], the installation of both the *uaserver* and the *opcua-client-gui* can be done easily through the *pip3 install* command in cmd. This installation, together with a running *uaserver*, is found in the screenshot in figure 45. A local connection to this server through the VC 4.0 OPC UA client was successful.

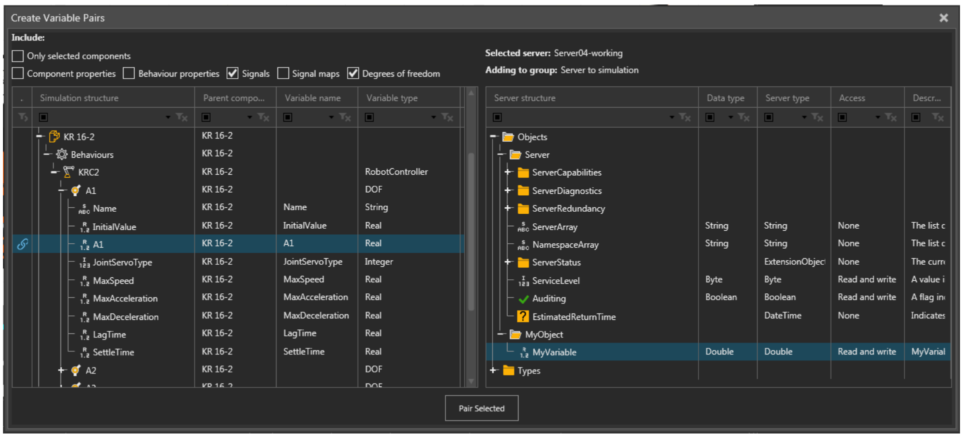
To start the development of a more suitable OPC UA server able to extract data from the KR C4 robot controller a minimal example server was used. This server is also a part of the FreeOpcUa library and can be found in the forked GitHub repository `https://github.com/akselov/python-opcua/blob/master/examples/server-minimal.py`. The server creates an object named "MyObject" and adds one variable to the object named "MyVariable". This variable is set to 0 and incremented with 0.1 for every second the server is running.



```
Microsoft Windows [Version 6.1.7601]
Copyright (c) 2009 Microsoft Corporation.  All rights reserved.

C:\Windows\system32>pip3 install opcua-client
Collecting opcua-client
  Using cached opcua-client-0.6.0.tar.gz
Requirement already satisfied: freeopcua in c:\users\akselov\appdata\local\progr
ams\python\python36-32\lib\site-packages (from opcua-client)
Requirement already satisfied: opcua-widgets in c:\users\akselov\appdata\local\p
rograms\python\python36-32\lib\site-packages (from opcua-client)
Requirement already satisfied: opcua in c:\users\akselov\appdata\local\programs\
python\python36-32\lib\site-packages (from freeopcua->opcua-client)
Requirement already satisfied: python-dateutil in c:\users\akselov\appdata\local
\programs\python\python36-32\lib\site-packages (from opcua->freeopcua->opcua-cli
ent)
Requirement already satisfied: pytz in c:\users\akselov\appdata\local\programs\p
ython\python36-32\lib\site-packages (from opcua->freeopcua->opcua-client)
Requirement already satisfied: lxml in c:\users\akselov\appdata\local\programs\p
ython\python36-32\lib\site-packages (from opcua->freeopcua->opcua-client)
Requirement already satisfied: six>=1.5 in c:\users\akselov\appdata\local\progra
ms\python\python36-32\lib\site-packages (from python-dateutil->opcua->freeopcua-
>opcua-client)
Installing collected packages: opcua-client
  Running setup.py install for opcua-client ... done
Successfully installed opcua-client-0.6.0

C:\Windows\system32>uaserver
cryptography is not installed, use of crypto disabled
cryptography is not installed, use of crypto disabled
WARNING: Listening on 0.0.0.0:4840
_
```

Figure 45: OPC UA server running in Windows

Running this server locally on the Windows machine running VC 4.0 resulted in "MyObject" and "MyVariable" being available in VC 4.0. Figure 46 shows VC 4.0s variable pairing functionality, where "MyVariable" is being paired with the A1 axis parameter of the KR 16-2. This resulted in the KR 16-2 slowly rotating about the A1 axis as the server incremented "MyVariable".



Include:

☐ Only selected components
☐ Component properties ☐ Behaviour properties ☑ Signals ☐ Signal maps ☑ Degrees of freedom

Selected server: Server04-working
Adding to group: Server to simulation

Simulation structure	Parent compo...	Variable name	Variable type
KR 16-2	KR 16-2		
Behaviours	KR 16-2		
KRC2	KR 16-2		RobotController
A1	KR 16-2		DOF
Name	KR 16-2	Name	String
InitialValue	KR 16-2	InitialValue	Real
A1	KR 16-2	A1	Real
JointServoType	KR 16-2	JointServoType	Integer
MaxSpeed	KR 16-2	MaxSpeed	Real
MaxAcceleration	KR 16-2	MaxAcceleration	Real
MaxDeceleration	KR 16-2	MaxDeceleration	Real
LagTime	KR 16-2	LagTime	Real
SettleTime	KR 16-2	SettleTime	Real
A2	KR 16-2		DOF

Server structure	Data type	Server type	Access	Descr...
Objects				
Server				
ServerCapabilities				
ServerDiagnostics				
ServerRedundancy				
ServerArray	String	String	None	The list c
NamespaceArray	String	String	None	The list c
ServerStatus		ExtensionObject	None	The curr
ServiceLevel	Byte	Byte	Read and write	A value i
Auditing	Boolean	Boolean	Read and write	A flag in
EstimatedReturnTime		DateTime	None	Indicates
MyObject				
MyVariable	Double	Double	Read and write	MyVarial
Types				

Figure 46: Creating variable pairs in VC 4.0

3.3.5 Transforming JOpenShowVar into an OPC UA server

To communicate with the KVP server on the robot controller, the JOpenShowVar client must specify two parameters in the message: the desired type of function and the variable name. As mentioned, the communication protocol relies on the TCP/IP protocol. In particular, on top of the TCP/IP layer, specially formatted text strings are used for message exchanges and KVP actively listens on TCP port 7000.

When reading a specific variable, the type of function must be identified by the character "0". For instance, if the variable to be read is the system variable $OV_PRO, which is used for the speed of the robot, the message that the client has to send to the server will have the format shown in table 2.

Field	**Description**
00	Message ID
09	Length of the next segment
0	Type of desired function
07	Length of the next segment
$OV_PRO	Variable to be read

Table 2: Message format for KVP [26]

In detail, the first two characters of this string specify the message identifier (ID) with an integer number between 00 and 99. The answer from the server will contain the same ID so that it is always possible to associate the corresponding answer to each request even if the feedback from the server is delayed. The two successive reading message characters specify the length of the next segment in hexadecimal units. In this specific case, 09 accounts for one character specifying the function type, two characters indicating the length of the next segment and seven characters for the variable length. The fifth character 0 in the message represents the type of the desired function - reading, in this case. Subsequently, two more characters are interacting the variable length (in hexadecimal units), and finally, the variable itself is contained in the last section of the message [26].

To avoid cross programming language communication between the Java JOpenShowVar and the Python OPC UA server, JOpenShowVar was begun translated into Python. However, after talking to co-supervisor Eirik Njåstad, it turned out that this already had been done by Ahmad Saeed in his GitHub repository `https://github.com/ahmad-saeed/kukavarproxy-msg-format`. The version used in this project can be found in the forked GitHub repository `https://github.com/akselov/kukavarproxy-msg-format`.

Based on the minimal server from the FreeOpcUa library and the translated JOpenShowVar, it was created an OPC UA server able to extract the axis parameters from the KR C4 robot controller and broadcast them to an OPC UA client. This code can be found in the project's GitHub repository `https://github.com/akselov/digital-twin-opcua/blob/master/opcua-server/opcua-server01.py`.

To run this python application, a Windows PC was used as middleware between the KR C4 robot controller and the Windows PC running VC 4.0 and hosting the DTw. Two Ethernet cables were connecting the three hardware units. The information flow of this system is illustrated in figure 47.

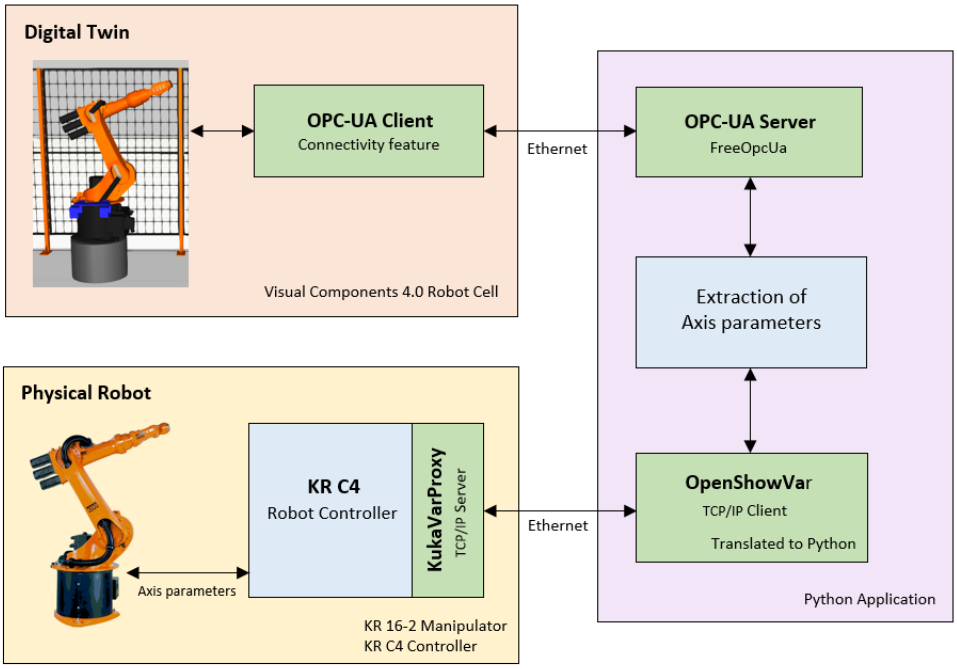
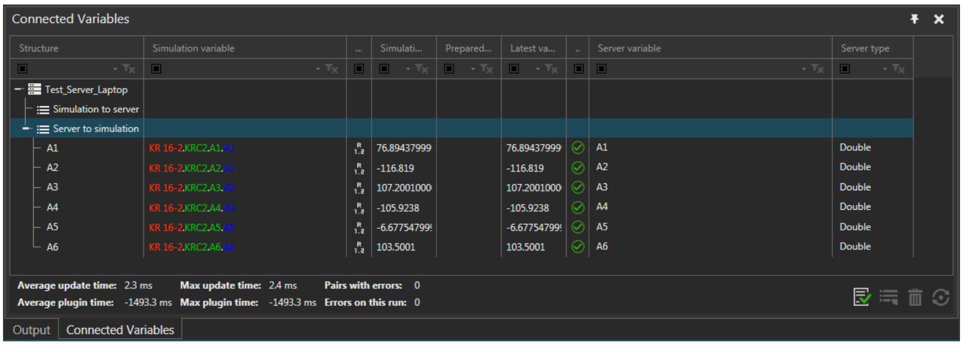


Figure 47: Information flow 1

With KVP running on the KR C4 and the OPC UA server running on the Windows PC, the axis variables of the KR 16-2 would be available as connected variables in VC 4.0. Figure 48 shows a screenshot from VC 4.0 where the axis variables are paired with the axis values of the KR 16-2 in the DTw. This causes the digital KR 16-2 to mirror the movements of the physical KR 16-2 actively.



Connected Variables

Structure	Simulation variable	...	Simulati...	Prepared...	Latest va...	..	Server variable	Server type
Test_Server_Laptop								
Simulation to server								
Server to simulation								
A1	KR 16-2.KRC2.A1.A1		76.89437999		76.89437999	✓	A1	Double
A2	KR 16-2.KRC2.A2.A2		-116.819		-116.819	✓	A2	Double
A3	KR 16-2.KRC2.A3.A3		107.2001000		107.2001000	✓	A3	Double
A4	KR 16-2.KRC2.A4.A4		-105.9238		-105.9238	✓	A4	Double
A5	KR 16-2.KRC2.A5.A5		-6.67754799		-6.67754799	✓	A5	Double
A6	KR 16-2.KRC2.A6.A6		103.5001		103.5001	✓	A6	Double

```
Average update time:  2.3 ms       Max update time:  2.4 ms       Pairs with errors:   0
Average plugin time:  -1493.3 ms   Max plugin time:  -1493.3 ms   Errors on this run:  0
```

Figure 48: List of connected variables from VC 4.0

3.3.6 Read time analysis of the OPC UA server

After setting up the first OPC UA server, *opcua-server01.py*, it was done a read time analysis to investigate the time from sending the axis variables from the KR C4 robot controller to receiving them in VC 4.0. The internal time delay related to KVP extracting the axis variables in the robot controller was not investigated.

A simple Point-To-Point (PTP) code was run on the robot controller of the KR 16-2 robot while recording the read time in the OPC UA server. This test code can be found in listing 3.1.

```
INI

PTP P8 Vel=100 % PDAT8 Tool[1]:ballfresen Base[0]


PTP P12 Vel=100 % PDAT12 Tool[1]:ballfresen Base[0]

PTP P13 Vel=100 % PDAT13 Tool[1]:ballfresen Base[0]


PTP P14 Vel=100 % PDAT14 Tool[1]:ballfresen Base[0]

PTP P9 Vel=100 % PDAT9 Tool[1]:ballfresen Base[0]
```

Listing 3.1: test_run.src

Running this code for 60 seconds resulted in the read times displayed in figure 49.

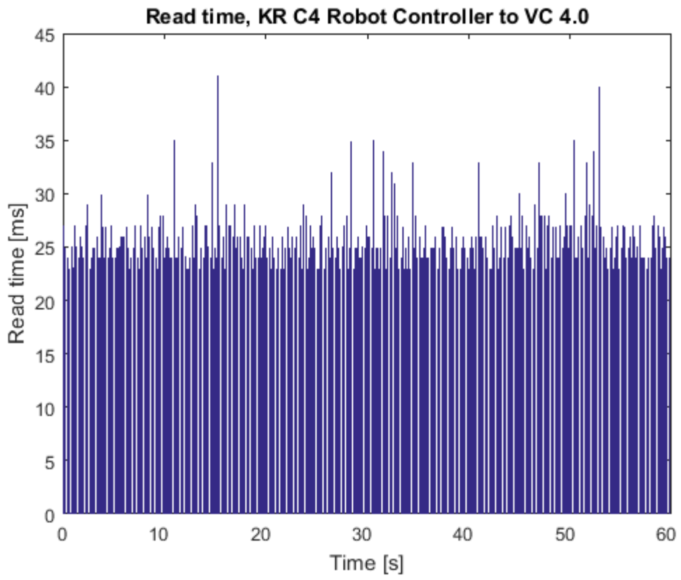


Figure 49: Read time, *opcua-server01.py*

This test resulted in an average read time of the *opcua-server01.py* of 25.69ms. The maximum read time that was obtained in this 60-second interval was 41.13ms.

3.3.7 Reducing read time of the OPC UA server

Tolerable limits to latency for live, real-time processing is a subject of investigation and debate but is estimated to be between 6 and 20 milliseconds [111]. In robot controlling applications, real-time processing often requires adjustments to trajectories based on sensor data. Therefore, real-time processing concerning robot controlling applications often requires even more strict limits to latency. The KUKA Robot Sensor Interface 3.1 (KUKA RSI) operates in two sensor modes. "IPO" which does signal processing at a sensor cycle rate of 12ms, and "IPO_FAST" with a sensor cycle rate of 4ms [112]. It was therefore evident that reducing the read time was an essential factor for this system to be usable in robot control applications.

To reduce the read time of the OPC UA server it was tried a different approach for reading the axis variables from the KR C4 robot controller. In *opcua-server01.py* the axis variables were read separately, resulting in a total of six separate readings from KVP to the OPC UA server. Reading all six axis variables in one single text string using the $AXIS_ACT command and then splitting the string into six separate double values, the number readings required to update the robot position was reduced by a factor of 6.

The source code for this improved OPC UA server, *opcua-server02.py*, can be found in appendix A.1, and in the project's GitHub repository `https://github.com/akselov/digital-twin-opcua/blob/master/opcua-server/opcua-server02.py`.

3.3.8 Read time analysis of the improved OPC UA server

The same PTP code used in the read time analysis for *opcua-server01.py*, found in listing 3.1, was run on the robot controller to record the read time of the *opcua-server02.py*. Running this code for 60 seconds resulted in the read times displayed in figure 50.

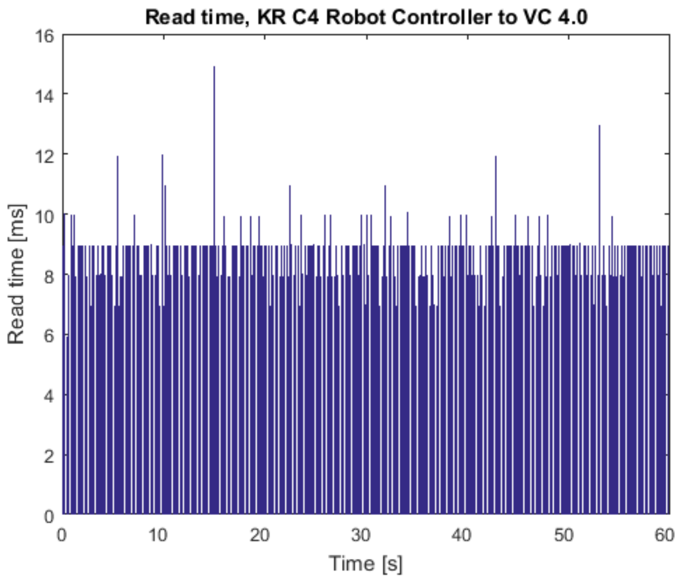


Figure 50: Read time, *opcua-server02.py*

This test resulted in an average read time of the *opcua-server02.py* of 8.75ms. The maximum read time that was obtained in this 60-second interval was 15.04ms.

3.4 Expanding the functionality of the DTw

In section 2.2 it was stated that most DTws have characteristics that give some typical benefits [67]. This included visibility, predictive analysis, what-if analysis, documentation to understand behavior and connection to backend applications. *opcua-server02.py* makes the axis variables of the robots in the physical robot cell available in the DTw. With this information, the DTw gains visibility to the state of the physical robot cell, by that all the robots in the cell are mirroring their physical counterpart. However, the DTw has little information about the state of the robots. With only information about the robot's movement, benefits such as predictive-, what-if analysis, and documentation would be limited.

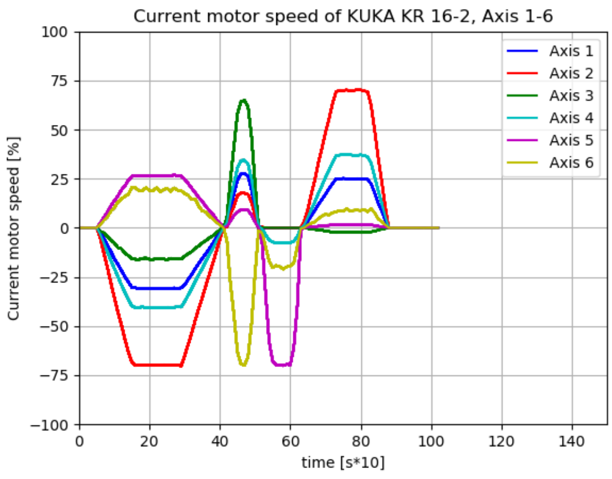
A study of available sensor information on the KR C4 robot controller was done to expand the functionality of the DTw and make it more suitable for predictive-, what-if analysis, and documentation. This study was done with a base in the document [113] describing all KUKA System Variables. It was chosen four variables that would improve the information available on the state of the robots in the robot cell:

1. $VEL_AXIS_ACT[X] - Velocity of motor controlling axis X
2. $TORQUE_AXIS_ACT[X] - Torque on motor controlling axis X
3. $CURR_ACT[X] - Current to motor controlling axis X
4. $MOT_TEMP[X] - Temperature of motor controlling axis X

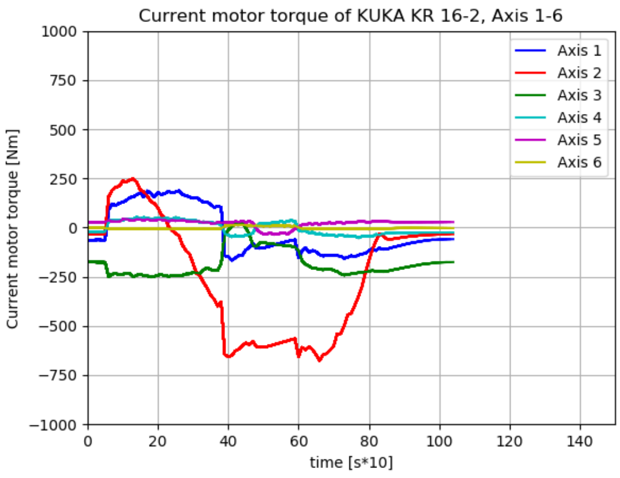
With the *opcua-server02.py* as a basis it was developed four OPC UA servers with the functionality described above. This additional information about the KUKA robot is being plotted by the server in real-time for all six axes simultaneously. The four servers can be found in the project's GitHub repository:

1. **Velocity:** *opcua-server-velocity.py*
   `https://github.com/akselov/digital-twin-opcua/blob/master/opcua-server/opcua-server-velocity.py`
2. **Torque:** *opcua-server-torque.py*
   `https://github.com/akselov/digital-twin-opcua/blob/master/opcua-server/opcua-server-torque.py`
3. **Current:** *opcua-server-current.py*
   `https://github.com/akselov/digital-twin-opcua/blob/master/opcua-server/opcua-server-current.py`
4. **Temperature:** *opcua-server-temperature.py*
   `https://github.com/akselov/digital-twin-opcua/blob/master/opcua-server/opcua-server-temperature.py`

The four servers were run on the Windows PC working as middleware between the KR C4 robot controller and the Windows PC running VC 4.0 and hosting the DTw. To test the servers, they were run while the PTP code listed in listing 3.1 was run on the robot controller of the KR 16-2 robot. Figure 51 and 52 shows screenshots of the real-time plotting done by the servers of the sensor data from the KR C4 robot controller.

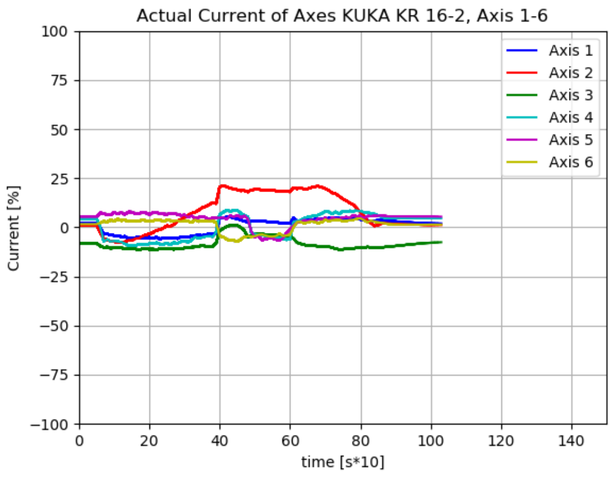


(a) Motor velocity

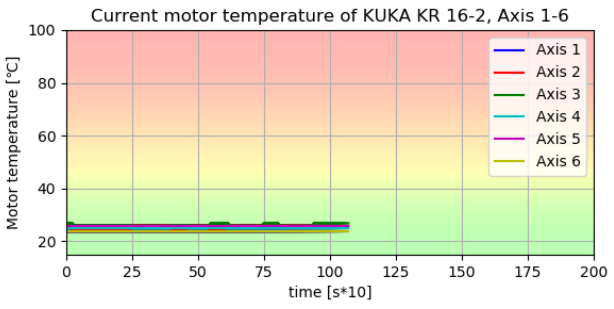


(b) Motor torque

Figure 51: Sensor data from KUKA KR 16-2



(a) Motor current



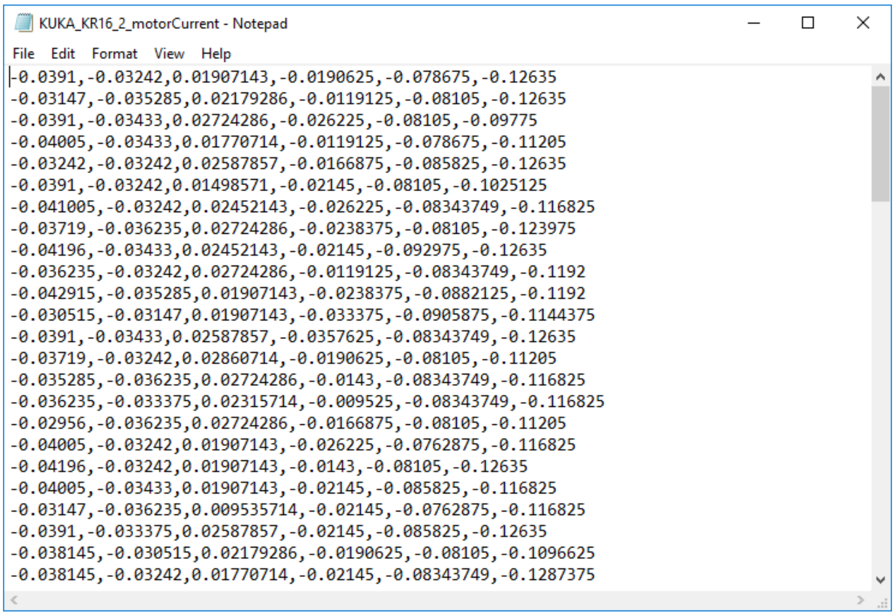
(b) Motor temperature

Figure 52: Sensor data from KUKA KR 16-2

Plots of sensor data can be a good source of information for predictive-, what-if analysis, and documentation. However, the raw data supplying the plots with information is often the element used when doing an actual analysis. In consultation with co-supervisor, Eirik Njåstad, it was decided that it would also be beneficial with the server functionality of writing sensor data to files. The format most often used in such analysis is comma-separated values, .csv, files.

With the *opcua-server02.py* as a basis it was developed four OPC UA servers with the functionality of writing sensor data to .csv files. The four servers can be found in the project's GitHub repository:

1. **Velocity to file:** *opcua-server-velocity-to-file.py*
   `https://github.com/akselov/digital-twin-opcua/blob/master/opcua-server/opcua-server-velocity-to-file.py`
2. **Torque to file:** *opcua-server-torque-to-file.py*
   `https://github.com/akselov/digital-twin-opcua/blob/master/opcua-server/opcua-server-torque-to-file.py`
3. **Current to file:** *opcua-server-current-to-file.py*
   `https://github.com/akselov/digital-twin-opcua/blob/master/opcua-server/opcua-server-current-to-file.py`
4. **Temperature to file:** *opcua-server-temperature-to-file.py*
   `https://github.com/akselov/digital-twin-opcua/blob/master/opcua-server/opcua-server-temperature-to-file.py`


KUKA_KR16_2_motorCurrent - Notepad
File Edit Format View Help

```
-0.0391,-0.03242,0.01907143,-0.0190625,-0.078675,-0.12635
-0.03147,-0.035285,0.02179286,-0.0119125,-0.08105,-0.12635
-0.0391,-0.03433,0.02724286,-0.026225,-0.08105,-0.09775
-0.04005,-0.03433,0.01770714,-0.0119125,-0.078675,-0.11205
-0.03242,-0.03242,0.02587857,-0.0166875,-0.085825,-0.12635
-0.0391,-0.03242,0.01498571,-0.02145,-0.08105,-0.1025125
-0.041005,-0.03242,0.02452143,-0.026225,-0.08343749,-0.116825
-0.03719,-0.036235,0.02724286,-0.0238375,-0.08105,-0.123975
-0.04196,-0.03433,0.02452143,-0.02145,-0.092975,-0.12635
-0.036235,-0.03242,0.02724286,-0.0119125,-0.08343749,-0.1192
-0.042915,-0.035285,0.01907143,-0.0238375,-0.0882125,-0.1192
-0.030515,-0.03147,0.01907143,-0.033375,-0.0905875,-0.1144375
-0.0391,-0.03433,0.02587857,-0.0357625,-0.08343749,-0.12635
-0.03719,-0.03242,0.02860714,-0.0190625,-0.08105,-0.11205
-0.035285,-0.036235,0.02724286,-0.0143,-0.08343749,-0.116825
-0.036235,-0.033375,0.02315714,-0.009525,-0.08343749,-0.116825
-0.02956,-0.036235,0.02724286,-0.0166875,-0.08105,-0.11205
-0.04005,-0.03242,0.01907143,-0.026225,-0.0762875,-0.116825
-0.04196,-0.03242,0.01907143,-0.0143,-0.08105,-0.12635
-0.04005,-0.03433,0.01907143,-0.02145,-0.085825,-0.116825
-0.03147,-0.036235,0.009535714,-0.02145,-0.0762875,-0.116825
-0.0391,-0.033375,0.02587857,-0.02145,-0.085825,-0.12635
-0.038145,-0.030515,0.02179286,-0.0190625,-0.08105,-0.1096625
-0.038145,-0.03242,0.01770714,-0.02145,-0.08343749,-0.1287375
```

Figure 53: Screenshot of .csv file containing data on motor current of KR 16-2

Like shown in figure 53, every line in the .csv file contains data on the current delivered to the six motors controlling the robot's axes. Every value is separated by a comma as required in the .csv format.

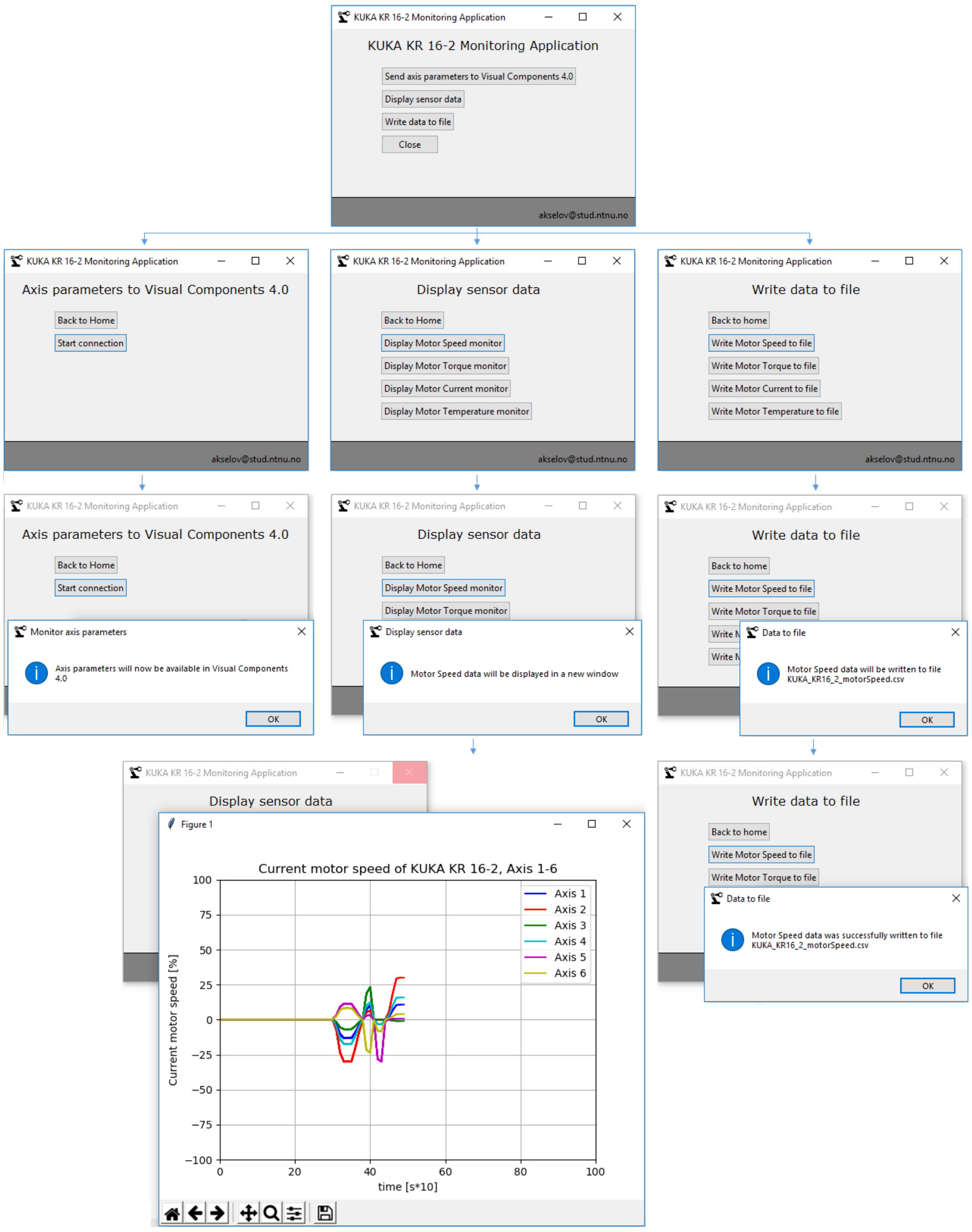
3.4.1 Structuring the expanded functionality in a GUI

With a total of eight new servers with expanded functionality as described in section 3.4, it was needed some structured environment with the capability of using the functionality of the different servers without having to search for the right server for the task. With this in mind, it was developed a Graphical User Interface (GUI) to make the use of the different functionalities more easily available and easier to use.

Figure 54 shows the structure of the GUI that was developed. It has three main categories of functions to be executed:

- Send axis parameters to VC 4.0
- Display sensor data
- Write data to file

Each of these categories is further displayed in the figure with their respective functionalities. The source code for the GUI can be found in appendix A.2 and the project's GitHub repository `https://github.com/akselov/digital-twin-opcua/blob/master/gui/opcua-server-gui.py`.


KUKA KR 16-2 Monitoring Application
KUKA KR 16-2 Monitoring Application
Send axis parameters to Visual Components 4.0
Display sensor data
Write data to file
Close
akselov@stud.ntnu.no
Axis parameters to Visual Components 4.0
Back to Home
Start connection
Display sensor data
Back to Home
Display Motor Speed monitor
Display Motor Torque monitor
Display Motor Current monitor
Display Motor Temperature monitor
Write data to file
Back to home
Write Motor Speed to file
Write Motor Torque to file
Write Motor Current to file
Write Motor Temperature to file
Monitor axis parameters
OK
Data to file
Figure 1
Current motor speed of KUKA KR 16-2, Axis 1-6
Current motor speed [%]
time [s*10]
Axis 1
Axis 2
Axis 3
Axis 4
Axis 5
Axis 6

Axis parameters will now be available in Visual Components 4.0

Motor Speed data will be displayed in a new window

Motor Speed data will be written to file KUKA_KR16_2_motorSpeed.csv

Motor Speed data was successfully written to file KUKA_KR16_2_motorSpeed.csv

Figure 54: GUI structure

3.5 Standardization to comply with OPC UA

The OPC UA server functionality that until now had been developed was based on the specifications of the KUKA KR 16-2 robot. Using these servers for the remaining robots in the cell would not be a problem as these also was of the KUKA brand and used the same KUKA KR C4 robot controllers. However, one of the principles in the OPC UA protocol is that it should be a module based communication system where all specific information about the physical parts is being obtained from XML-files. Extensible Markup Language (XML) is a language used in computing, which defines a set of rules for encoding documents in a format that is both human-readable and machine-readable [34].

As stated in section 2.6, model-driven techniques which are already used to design complex software systems, tremendously simplify the process of information model design. The main concept of a model-driven approach is to separate the functional description and the implementation. Thus, it is not necessary to repeat the process of defining an application or the system's functionality and behavior each time a new communication technology comes along. The model-driven approach behind OPC UA was further discussed in section 2.6.1.

The modularization of the server functionality should also include the KVP communication module. The *opcua-server02.py*, which a lot of the functionality to this point was built on, had the KVP communication module included as a part of the OPC UA server. At a later stage, to satisfy the "IPO_FAST" requirement of a sensor cycle rate of 4ms as discussed in section 3.3.7, the OPC UA server would need a faster communication module, such as RSI. If RSI easily could be included as a communication module in parallel with KVP, without doing significant changes in the OPC UA server, this would be beneficial.

To sum up, the standardization to comply with OPC UA standards boiled down to two steps:

1. An OPC UA server that can obtain specifications about a physical asset from an XML-file and make custom OPC UA objects based on these specifications.
2. Make the OPC UA server ready for more than one communication module. Make KVP a separate module that is only being included if the OPC UA client requests this.

Also, but not as a part of the standardization, it was concluded that it should be done an attempt at including RSI as a separate communication module to satisfy the "IPO_FAST" requirement of a sensor cycle rate of 4ms [112].

3.5.1 Standardization: Step 1

The FreeOpcUa library contains an XML-modeler for construction of XML-files containing information about objects that are included in a communication network. This modeler is much like the UaModeler described by Pauker, Frühwirth, Kittl, and Kastner in [7], that has a structure illustrated in figure 55. These models can be edited, modified, and saved in an XML format. The forked version of the modeler used in this project can be found at `https://github.com/akselov/opcua-modeler`.

With the use of this modeler, it was created an XML-file for the KUKA KR 16-2 robot containing two Boolean variables for KUKAVARPROXY and RSI, six Double variables for each of the robots axis values, and a string containing the IP of the KR C4 robot controller. This file can be found in appendix A.3 and in the project's GitHub repository `https://github.com/akselov/digital-twin-opcua/blob/master/xml/KUKA_KR16_2_object.xml`. Based on this file it was created similar configuration files for the remaining robots in the robot cell. These can be found in the project's GitHub repository at `https://github.com/akselov/digital-twin-opcua/tree/master/xml`.

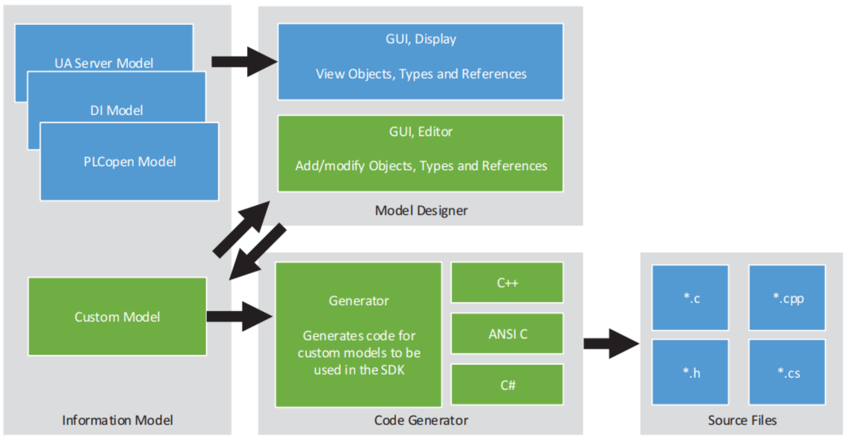
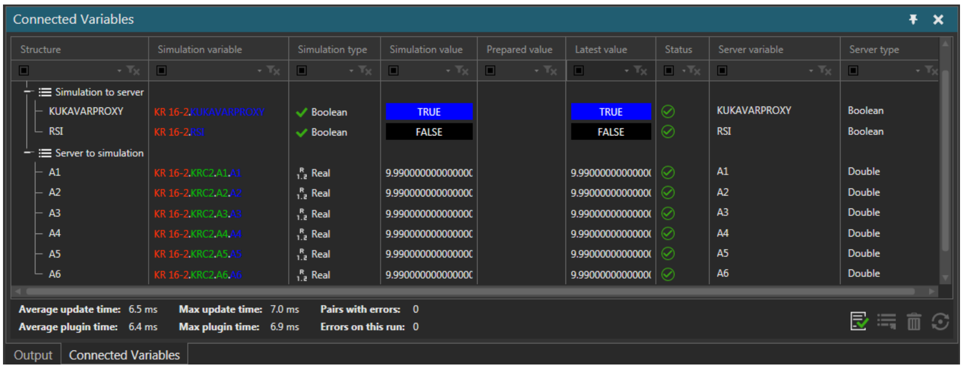


Figure 55: Functions of OPC UA modeling tools [7]

To extract the relevant information from the XML-file a server capable of creating custom OPC UA objects had to be developed. This server was created with a base in a server from the FreeOpcUa library called *server-custom-object.py*. This server can be found in the forked GitHub repository at `https://github.com/akselov/python-opcua/blob/master/examples/server-custom-object.py`.

The server developed sets the two Boolean values, KUKAVARPROXY and RSI, to be writable by OPC UA clients. In this project, it was used VC 4.0 as an OPC UA client, and thus the desired communication has to be set from here. The server also retrieves the IP address of the robot controller from the XML-file and establish a connection to this robot controller through the chosen communication protocol, KVP or RSI. The server sets the robot axis variables as Double 9.99 by default and to be changed by the connection with the KR C4 robot controller. The connection, with KUKAVARPROXY chosen by the client as communication protocol (TRUE) and axis values set to the default value, is seen in the screenshot from VC 4.0 in figure 56.


Connected Variables

Structure	Simulation variable	Simulation type	Simulation value	Prepared value	Latest value	Status	Server variable	Server type
Simulation to server								
KUKAVARPROXY	KR 16-2.KUKAVARPROXY	✓ Boolean	TRUE		TRUE	✓	KUKAVARPROXY	Boolean
RSI	KR 16-2.RSI	✓ Boolean	FALSE		FALSE	✓	RSI	Boolean
Server to simulation								
A1	KR 16-2.KRC2.A1.A1	Real	9.990000000000000		9.9900000000000	✓	A1	Double
A2	KR 16-2.KRC2.A2.A2	Real	9.990000000000000		9.9900000000000	✓	A2	Double
A3	KR 16-2.KRC2.A3.A3	Real	9.990000000000000		9.9900000000000	✓	A3	Double
A4	KR 16-2.KRC2.A4.A4	Real	9.990000000000000		9.9900000000000	✓	A4	Double
A5	KR 16-2.KRC2.A5.A5	Real	9.990000000000000		9.9900000000000	✓	A5	Double
A6	KR 16-2.KRC2.A6.A6	Real	9.990000000000000		9.9900000000000	✓	A6	Double

Average update time: 6.5 ms **Max update time:** 7.0 ms **Pairs with errors:** 0
Average plugin time: 6.4 ms **Max plugin time:** 6.9 ms **Errors on this run:** 0

Figure 56: VC 4.0: Connected variables in OPC UA client

3.5.2 Standardization: Step 2

To make the OPC UA server ready for more than one communication module the KVP configuration and communication had to be modularized and extracted to a separate communication module. This was simply done by extracting the KVP configuration and communication into a separate Python file to be launched from the OPC UA server. The server is set up so that it listens to the OPC UA client to check if communication over KVP or RSI is desired.

The OPC UA server developed after standardization step 1 and 2 can be found in appendix A.4 and in the project's GitHub repository at `https://github.com/akselov/digital-twin-opcua/blob/master/xml/opcua-server-xml.py`.

The extracted KVP communication module can be found in appendix A.5 and in the project's GitHub repository at `https://github.com/akselov/digital-twin-opcua/blob/master/xml/KUKAVARPROXY_communication.py`. With standardization step 1 and 2 completed the system has a more modularized structure. The new information flow in the DTw is illustrated in figure 57.

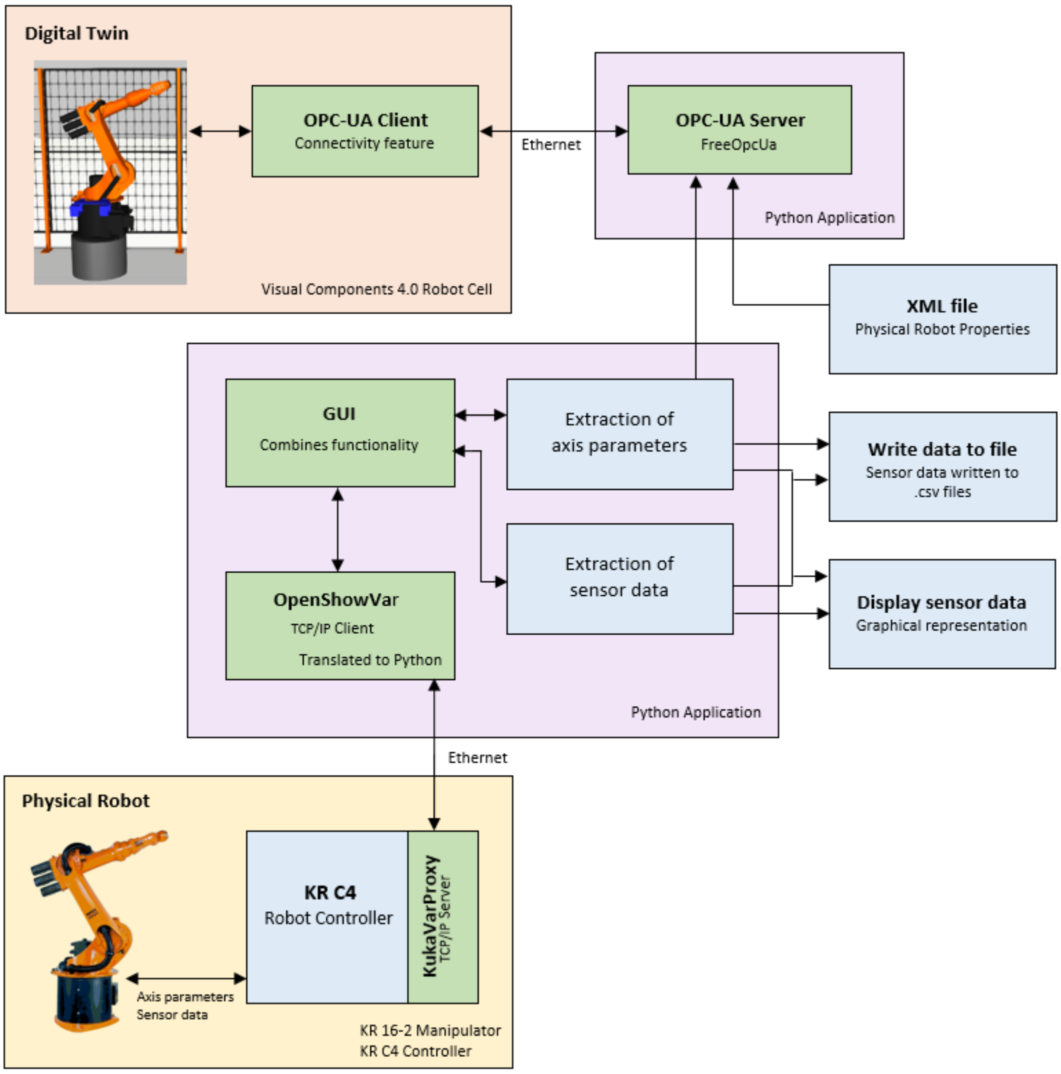


Figure 57: Information flow 2

3.6 Including RSI communication

To satisfy the "IPO_FAST" requirement of a sensor cycle rate of 4ms as discussed in section 3.3.7, it was clear that the OPC UA server in some cases would need a faster communication module, such as RSI. RSI uses UDP for communication, as described in section 2.10.2. The idea was to make a communication architecture similar to the one created around KVP and make this run in parallel with the system created up until now. The basis of the communication architecture developed for KVP was described in figure 41. The proposed communication architecture for the system using RSI is illustrated in figure 58.

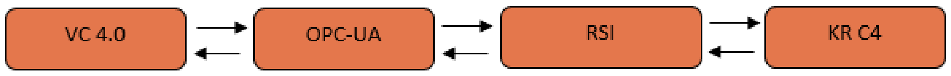


Figure 58: Proposed communication architecture with RSI

As JOpenShowVar and KVP, RSI works as a middleware between the OPC UA server and the KR C4 robot controller and thus opens a backdoor for an alternative implementation of an OPC UA communication. RSI is however not running as a separate server in the background of the robot controller, but instead running as a KRL program that has to be selected and executed on the robot controller for the RSI communication to be established.

After some research and consultation with co-supervisor Eirik Njåstad, it was found that Eren Sezener (`https://github.com/erensezener`) from BCCN - TU Berlin, has developed a KUKA RSI-3 Communicator (KR3C) in Python. KR3C is developed for the Robot Sensor Interface (RSI) 3 add-on. The forked version used in this project can be found at `https://github.com/akselov/kuka-rsi3-communicator`.

Torstein Anderssen Myhre (`https://www.ntnu.edu/employees/torstein.a.myhre`), who did a project with one of the KUKA KR 120 R2500 robots in the robot cell simultaneously as the development of this DTw was being done, has also developed a communication interface for communicating with RSI, which was used in the development of a RSI communication module for this DTw. The RSI files used in his project can be found in the forked GitHub repository at `https://github.com/akselov/examplecode-kukarsi-python`.

3.6.1 Setting up RSI communication

Eren Sezener has a thorough description of how to set up KR3C in the GitHub repository. The configuration of the KR C4 robot controller was followed with two exceptions:

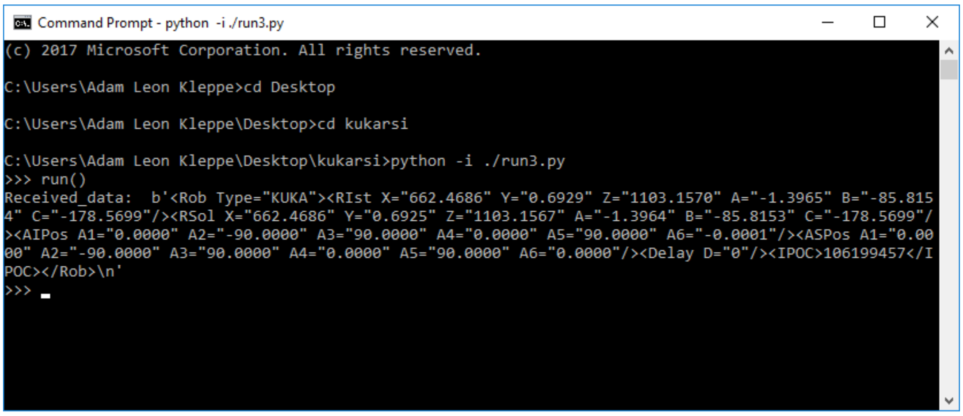
1. Instead of copying the file *RSI_Ethernet.src* to *C:\KRC\ROBOTER\KRC \R1\Program*, the file *RSI_AxisCorr2016v2.src* was copied to the same folder. This file can be found in the project's GitHub repository at:

   `https://github.com/akselov/digital-twin-opcua/blob/master/rsi/RSI_AxisCorr2016v2.src`

2. Instead of copying the file *RSI_EthernetConfig.xml* to *C:\KRC\Roboter \Config\User\Common\SensorInterface*, the file *RSIAxisCorrTestConfig2016v2.xml* was copied to the same folder. This file can be found in the project's GitHub repository at:

   `https://github.com/akselov/digital-twin-opcua/blob/master/rsi/RSIAxisCorrTestConfig2016v2.xml`

The program *RSI_AxisCorr2016v2.src* has to be selected and run from the KR C4 robot controller for the RSI communication to be established. The development of an RSI communication module for the OPC UA server started with a base in the *run.py* file made by Torstein Anderssen Myhre. The file can be found in the forked GitHub repository at `https://github.com/akselov/examplecode-kukarsi-python/blob/master/rsi/run.py`. Running this file, with some minor modifications, on a Windows PC, resulted in receiving the robot position and the axis variables from the KR C4 as shown in the screenshot in figure 59.

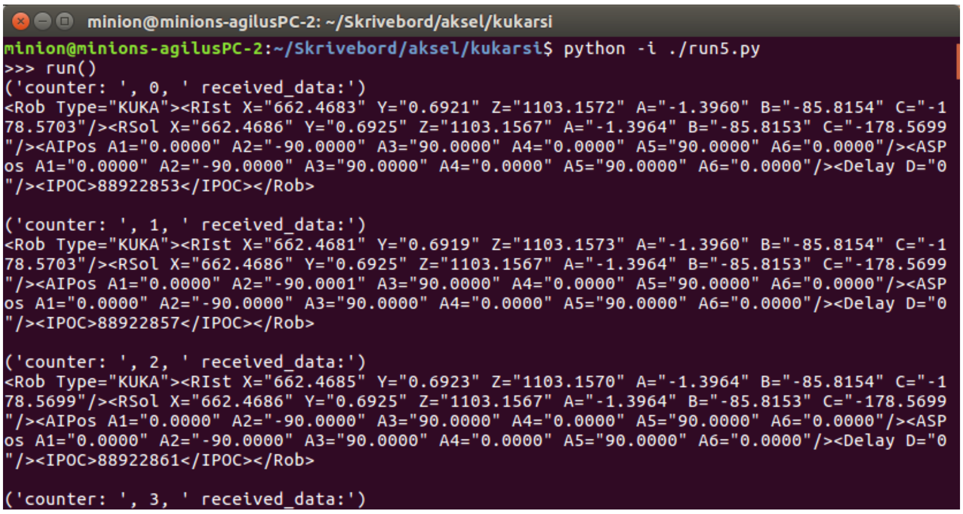

Command Prompt - python -i ./run3.py

```
(c) 2017 Microsoft Corporation. All rights reserved.

C:\Users\Adam Leon Kleppe>cd Desktop

C:\Users\Adam Leon Kleppe\Desktop>cd kukarsi

C:\Users\Adam Leon Kleppe\Desktop\kukarsi>python -i ./run3.py
>>> run()
Received_data:  b'<Rob Type="KUKA"><RIst X="662.4686" Y="0.6929" Z="1103.1570" A="-1.3965" B="-85.815
4" C="-178.5699"/><RSol X="662.4686" Y="0.6925" Z="1103.1567" A="-1.3964" B="-85.8153" C="-178.5699"/
><AIPos A1="0.0000" A2="-90.0000" A3="90.0000" A4="0.0000" A5="90.0000" A6="-0.0001"/><ASPos A1="0.00
00" A2="-90.0000" A3="90.0000" A4="0.0000" A5="90.0000" A6="0.0000"/><Delay D="0"/><IPOC>106199457</I
POC></Rob>\n'
>>> 
```

Figure 59: Screenshot of first RSI communication with Windows PC

While setting up the file to print the received data from the robot controller, it was encountered a problem. The connection broke down every time after precisely 100 iterations. This was most 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.

After consulting with Torstein Anderssen Myhre the same python code was instead run on a Linux PC. Here the data was received successfully after 100 iterations. The reason for this probably lies in the way the two OSs are designed and how they use their cores to run the code. Screenshot of the received data from the robot controller to the Linux PC can be found in figure 60.



```
minion@minions-agilusPC-2: ~/Skrivebord/aksel/kukarsi
minion@minions-agilusPC-2:~/Skrivebord/aksel/kukarsi$ python -i ./run5.py
>>> run()
('counter: ', 0, ' received_data:')
<Rob Type="KUKA"><RIst X="662.4683" Y="0.6921" Z="1103.1572" A="-1.3960" B="-85.8154" C="-1
78.5703"/><RSol X="662.4686" Y="0.6925" Z="1103.1567" A="-1.3964" B="-85.8153" C="-178.5699
"/><AIPos A1="0.0000" A2="-90.0000" A3="90.0000" A4="0.0000" A5="90.0000" A6="0.0000"/><ASP
os A1="0.0000" A2="-90.0000" A3="90.0000" A4="0.0000" A5="90.0000" A6="0.0000"/><Delay D="0
"/><IPOC>88922853</IPOC></Rob>

('counter: ', 1, ' received_data:')
<Rob Type="KUKA"><RIst X="662.4681" Y="0.6919" Z="1103.1573" A="-1.3960" B="-85.8154" C="-1
78.5703"/><RSol X="662.4686" Y="0.6925" Z="1103.1567" A="-1.3964" B="-85.8153" C="-178.5699
"/><AIPos A1="0.0000" A2="-90.0001" A3="90.0000" A4="0.0000" A5="90.0000" A6="0.0000"/><ASP
os A1="0.0000" A2="-90.0000" A3="90.0000" A4="0.0000" A5="90.0000" A6="0.0000"/><Delay D="0
"/><IPOC>88922857</IPOC></Rob>

('counter: ', 2, ' received_data:')
<Rob Type="KUKA"><RIst X="662.4685" Y="0.6923" Z="1103.1570" A="-1.3964" B="-85.8154" C="-1
78.5699"/><RSol X="662.4686" Y="0.6925" Z="1103.1567" A="-1.3964" B="-85.8153" C="-178.5699
"/><AIPos A1="0.0000" A2="-90.0000" A3="90.0000" A4="0.0000" A5="90.0000" A6="0.0000"/><ASP
os A1="0.0000" A2="-90.0000" A3="90.0000" A4="0.0000" A5="90.0000" A6="0.0000"/><Delay D="0
"/><IPOC>88922861</IPOC></Rob>

('counter: ', 3, ' received_data:')
```

Figure 60: Screenshot of first successful RSI communication in Linux

3.6.2 Combine RSI communication with an OPC UA server

As with the development of the OPC UA server based on KVP communication, this OPC UA server was developed with a basis in the *server-minimal.py* from the FreeOpcUa library. This can be found in the forked GitHub repository at `https://github.com/akselov/python-opcua/blob/master/examples/server-minimal.py`. Combining this with the extraction of axis variables through RSI communication resulted in the *rsi-opcua-server.py* that can be found in appendix A.6 and in the project's GitHub repository at `https://github.com/akselov/digital-twin-opcua/blob/master/rsi/rsi-opcua-server.py`.

rsi-opcua-server.py extracts the axis variables from a KUKA KR C4 robot controller through RSI communication and send them to an OPC UA client, here VC 4.0. As with the OPC UA server based on KVP, it was done a read time analysis on the *rsi-opcua-server.py* to investigate the read time from the KR C4 robot controller to VC 4.0. The read time test was done two times. One with the KUKA KR 16-2 standing still, and one with the robot performing a similar motion to the PTP code that was used with the KVP OPC UA server, found in listing 3.1. The results of these tests can be found in figure 61 (no movement), and figure 62 (with movement).

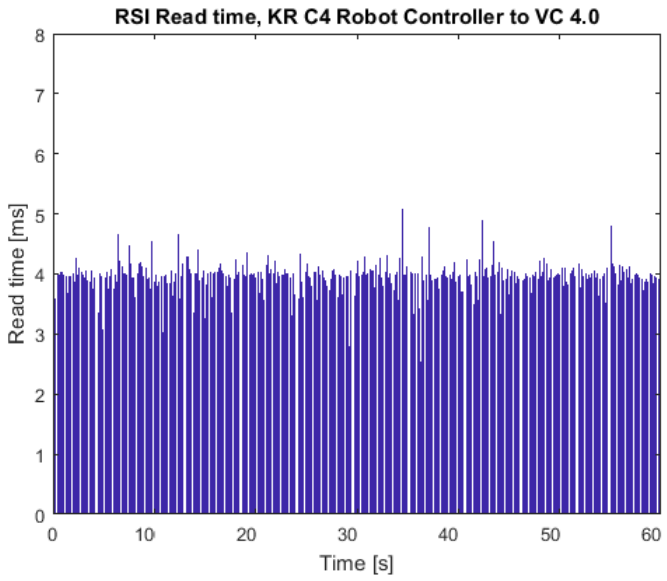


Figure 61: RSI Read time, *rsi-opcua-server.py*, no movement

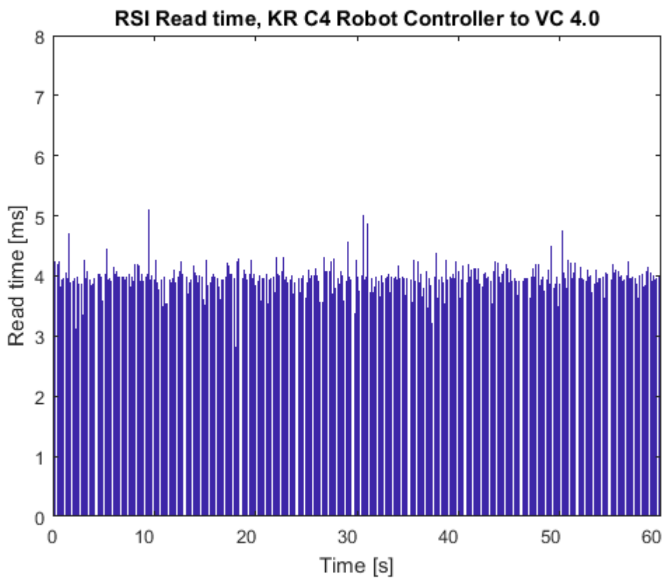


Figure 62: RSI Read time, *rsi-opcua-server.py*, with movement

The tests resulted in an average read time of the *rsi-opcua-server.py* of 4.00ms both with and without robot movement. The maximum read time that was obtained in this 60-second interval was 5.04ms for the test without robot movement and 5.12ms with robot movement.

3.7 Controlling robots from DTw

As a last property of the DTw, it was developed the functionality of controlling the robots in the robot cell from the DTw. This could be used when wanting to run a robot program directly in VC 4.0 and make the physical robots execute the commands simultaneously as the digital robots. This system was developed based on work done by Adam Leon Kleppe (`http://folk.ntnu.no/adamleon/`) on controlling KUKA robots from an external PC using KVP.

For controlling the KUKA robots through KVP, the KR C4 robot controller has to run a program listening to updates on position. With minor changes, this program was made to listen to axis variables. The program consists of two files that have to be copied to *C:\KRC\ROBOTER\KRC\R1\Program* on the KR C4 robot controller:

1. *opcua_ptp_movement.dat*
   `https://github.com/akselov/digital-twin-opcua/blob/master/opcua-control/opcua_ptp_movement.dat`
2. *opcua_ptp_movement.src*
   `https://github.com/akselov/digital-twin-opcua/blob/master/opcua-control/opcua_ptp_movement.src`

Selecting and running the *opcua_ptp_movement.src* results in the robot listening to axis values from a KVP client. This is a similar setup to the *RSI_AxisCorr2016v2.src* used for the RSI communication. Based on the standardized *opcua-server-xml.py* it was created an OPC UA server able to create a custom OPC UA object from specifications in an XML-file, extract axis variables from an OPC UA client, here VC 4.0, and control a KUKA robot by its axis values. This file can be found in appendix A.7 and in the project's GitHub repository `https://github.com/akselov/digital-twin-opcua/blob/master/opcua-control/opcua-control-server.py`.

With this setup, the KR C4 is listening to the axis variables from VC 4.0 and updating the physical robot position to mirror the robot in the DTw. To avoid the robots crashing with other components in the robot cell while being controlled from the DTw, it was tested using working envelopes for limiting the working area of the robots. This was configured from the smart pad the following way:

1. Log on Administrator
2. Navigate to Miscellaneous -> Monitoring working envelope -> Configuration
3. Choose "Axis specific"
4. Set minimum and maximum values for all axis
5. Choose: Mode = "OUTSIDE_STOP"

When this setting is activated, the robot will stop with an error message as soon as it moves outside the set parameters.

3.8 Fixed case

A goal of this project listed in section 1.1, was to *Try out the system in a fixed case. Evaluate the results.* The development of this DTw has resulted in several parallel communication systems, all with different functionalities. The fixed case was tried constructed in such a way that this would test as many of these functionalists as possible. Together with co-supervisor, Eirik Njåstad, a test was created with the goal of studying the functionalities listed in figure 63.

Studied function	**KUKA KR 16-2**	**KUKA KR 120 R2500**
Multi-Robot	✓	✓
Control from DTw		✓
Mirror movement	✓	
Plotting data	✓	✓

Figure 63: Functionalities studied in fixed case

- **Multi-Robot:** Testing if the DTw is able to simultaneously control and mirror movements of more than one single robot.
- **Control from DTw:** Making a PTP program in VC 4.0 and controlling the KR 120 directly from VC 4.0 working as an OPC UA client. The *opcua-control-server.py* in appendix A.7 was used as middleware on a Windows PC.
- **Mirror movement:** Extract axis variables from KR 16-2 and use them to simulate the physical movements of the robot in VC 4.0. The developed *opcua-server-gui.py* in appendix A.2 was used with the functionality "Send axis parameters to VC 4.0".
- **Plotting data:** Plotting the x-, y- and z-values for both the robots using the developed GUI with the functionality "Display sensor data".

3.8.1 Setup

Co-supervisor Eirik Njåstad suggested using a simple camera tracking algorithm based on the Visual Servoing Platform (ViSP). ViSP is a modular C++ library that allows fast development of visual servoing applications [114]. It was used a single camera mounted as an end effector on the KR 16-2 robot. The purpose of the camera was to track the movements of an AprilTag mounted to a plate on the end effector of the KR 120. AprilTag is a visual fiducial system, useful for a wide variety of tasks including augmented reality, robotics, and camera calibration. AprilTags are conceptually similar to QR Codes, in that they are a type of two-dimensional bar code. However, they are designed to encode far smaller data payloads (between 4 and 12 bits), allowing them to be detected more robustly and from longer ranges. They are also designed for high localization accuracy, which makes it possible to compute the precise 3D position of the AprilTag with respect to the camera [27]. Figure 64 shows the AprilTag used in this case.

Figure 64: AprilTag [27]

The aim of the code created in ViSP was to have a vision-based control of the KR 16-2 robot so that the robot would always work towards keeping the AprilTag in the centre of the camera mounted to its end effector. In this way, the distance from the centre is an error that has to be minimized. The error is typically defined by [114]:

$$\mathbf{e}(t) = \mathbf{s}[\mathbf{m}(t), \mathbf{a}] - \mathbf{s}^* \tag{3.1}$$

Traditional image-based control schemes use the image-plane coordinates of a set of points to define the set of visual features $\mathbf{s}$. The image measurements $\mathbf{m}$ are usually the pixel coordinates of the set of image points (but this is not the only possible choice), and the camera intrinsic parameters $\mathbf{a}$ are used to go from image measurements expressed in pixels to the features.

For a 3D point with coordinates $\mathbf{X} = (X, Y, Z)$ in the camera frame, which projects in the image as a 2D point with coordinates $\mathbf{x} = (x, y)$ we have:

$$\begin{cases} x = X/Y = (u - u_0)/p_x \\ y = Y/Z = (v - v_0)/p_y \end{cases} \tag{3.2}$$

Here $\mathbf{m} = (u, v)$ gives the coordinates of the image point expressed in pixel units, and $\mathbf{a} = (u_0, v_0, p_x, p_y)$ is the set of camera intrinsic parameters: u_0 and v_0 are the coordinates of the principal point, while p_x and p_y are the ratio between the focal length and the size of a pixel [114].

If the spatial velocity of the camera be denoted by $\mathbf{v}_c = (v_c, \omega_c)$, v_c the instantaneous linear velocity of the origin of the camera frame and ω_c the instantaneous angular velocity of the camera frame. The relationship between $\dot{\mathbf{x}}$, the time variation of the feature $\mathbf{s} = \mathbf{x}$, and the camera velocity $\mathbf{v}_c$ is given by:

$$\dot{\mathbf{s}} = \mathbf{L_x}\mathbf{v}_c \tag{3.3}$$

Where the interaction matrix $\mathbf{L_x}$ is given by:

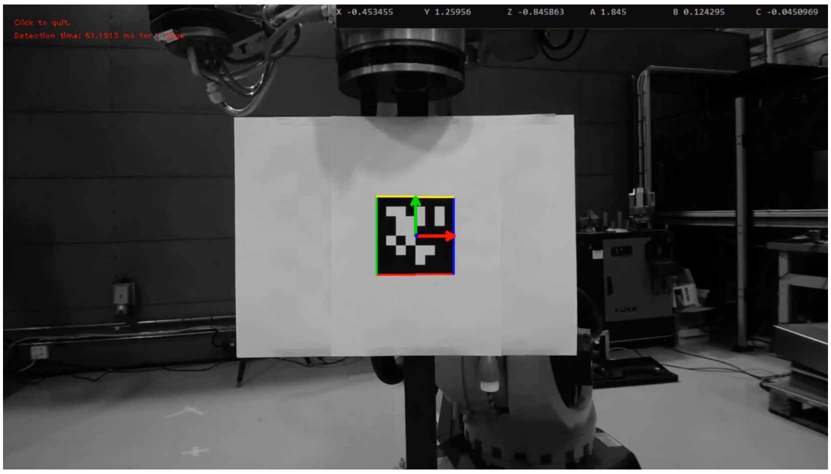
$$\mathbf{L_x} = \begin{bmatrix} -1/Z & 0 & x/Z & xy & -(1+x^2) & y \\ 0 & -1/Z & y/Z & 1+y^2 & -xy & -x \end{bmatrix} \tag{3.4}$$

Considering $\mathbf{v}_c$ as the input to the robot controller, and if we would like for instance to try to ensure an exponential decoupled decrease of the error, we obtain:

$$\mathbf{v}_c = -\lambda \mathbf{L}_\mathbf{x}^{+}\mathbf{e} \tag{3.5}$$

Eirik Njåstad developed a code based on this library and combined it with the same KVP controlling principle used in *opcua-control-server.py*. The code was developed in C++ and can be found in his GitHub repository `https://github.com/eirikn/basicVisualServoing`.

Running this code resulted in the KR 16-2 tracking the AprilTag mounted to the KR 120 and mirrored its movements in real-time. A screenshot of the camera stream from the KR 16-2 is displayed in figure 65.



```
Click to quit.
Detection time: 61.1913 ms for 1 tags
```

```
X -0.453455    Y 1.25956    Z -0.845863    A 1.845    B 0.124295    C -0.0450969
```

Figure 65: Camera stream from KR 16-2

A simple PTP program containing four points was created in the DTw in VC 4.0 moving the KR 120s end effector 500mm along the y-, x- and z-axis. This is conceptually illustrated in figure 66 with the movements represented as the vectors $\mathbf{a_y}$, $\mathbf{a_x}$ and $\mathbf{a_z}$.

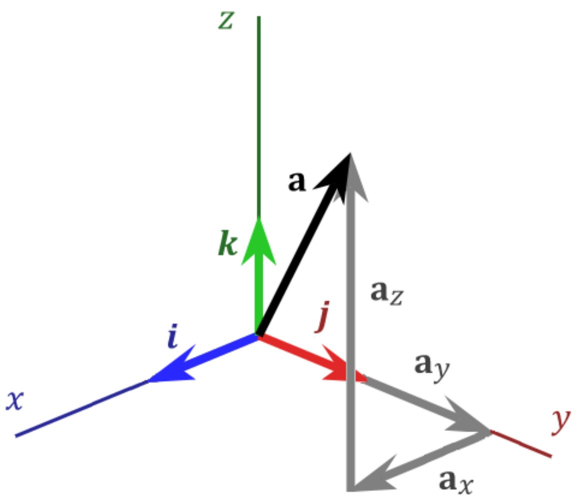


Figure 66: Illustration of PTP path

As the KR 16-2 would mirror the movement of the KR 120 its axis variables would be sent to VC 4.0 so that the DTw could in real-time track its movement. To illustrate the time delay between the KR 120 and the KR 16-2 mirroring its movements, the Windows PC working as middleware between the KR C4 and the PC hosting the DTw would do real-time plotting of the Cartesian x-, y- and z-values from both the KR 120 and Kr 16-2.

The setup of the fixed case is shown in figure 67. The case was filmed using:

1. Overview camera showing the movement of KR 120 and KR 16-2 in robot cell (top left)
2. Camera mounted to KR 16-2 (top middle)
3. Screen capture of PC hosting the DTw in VC 4.0 (bottom left)
4. Screen capture of Windows PC doing the real-time plotting (top and bottom right)

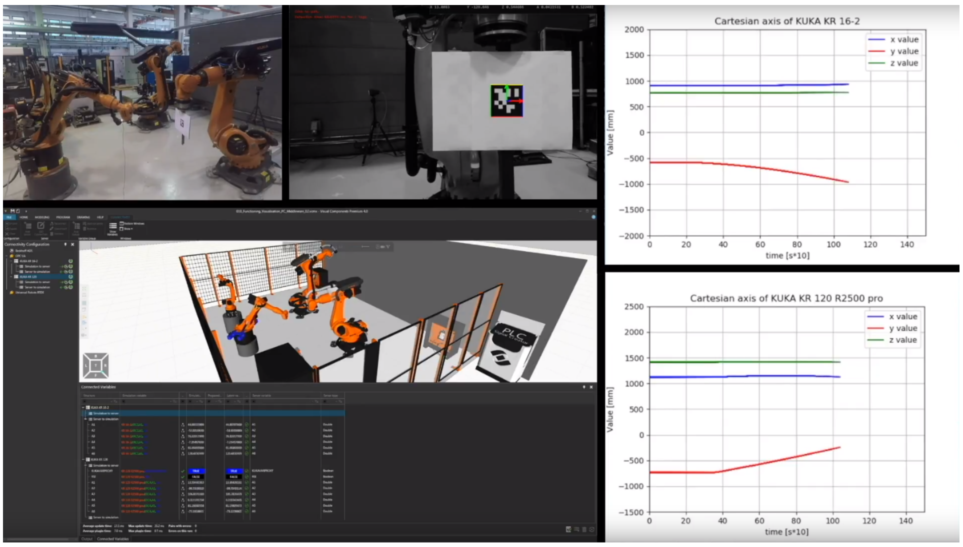


Figure 67: Fixed case setup

The video footage was put into one frame to show how the DTw works with the physical robot cell in real-time. The final video showing the fixed case described can be found in the digital appendix B.2, and is also published online at `https://youtu.be/xlQhQPmJwlA`.

Chapter 4

Result

This project is a study on how a DTw can be implemented using OPC UA. A DTw for the robot cell at MTP Valgrinda was developed. Using this technology for expanding the functionality of the robot cell gave various benefits to the system. The general benefits usually seen when introducing a DTw was presented in section 2.2. It is analyzed the specific benefits this DTw provides for the robot cell in section 5.6, and its value for I 4.0 in section 5.7.

The use of VC 4.0 was measured up against similar simulation software. VC 4.0 was considered the strongest candidate for this project and was used to make the virtual representation of the robot cell. The modeling of the cell was thoroughly described in section 3.2.1. Most of the work done in this project revolved around the simulation and communication system connecting the DTw to the physical robot cell. As a DTw can have multiple different use cases and therefore require a lot of different functionality, it was decided that it would not be sufficient with one single communication system, but rather a library of communication systems suited for different tasks. All developed files, both for VC 4.0 and communication, is described in chapter 3 and can be found in appendix B.1 and the project's GitHub repository

`https://github.com/akselov/digital-twin-opcua`

In all the different communication systems developed it was used a Windows PC as middleware between the KR C4 controllers in robot cell and the Windows PC running VC 4.0 and hosting the DTw. KUKA Norway was contacted to gain access to software that would make the use of middleware unnecessary. This software was not received and tested, but as discussed further in section 5.3, the Institute of MTP is already in contact with KUKA Norway regarding the possibility for updating the KR C4 controllers in the robot cell to KRC ROBOTstar, which has OPC UA built in [115].

The result of the work done in this project is divided into 4 categories:

1. DTw in VC 4.0
2. Communication library
3. The setup
4. Functionalities

In addition, data from the fixed case described in section 3.8 is presented.

4.1 DTw in VC 4.0

Two versions of the final virtual representation of the robot cell is included in the digital appendix and the GitHub repository:

1. **Robot_Cell_MTP.vcmx:** Version compatible with all servers developed, including one example OPC UA client set up for controlling the KUKA KR 16-2 from VC 4.0: `https://github.com/akselov/digital-twin-opcua/blob/master/VC4.0/Robot_Cell_MTP.vcmx`

2. **Robot_Cell_MTP_case.vcmx:** Version used in the fixed case described in section 3.8, including two OPC UA clients set up. One for controlling the KUKA KR 120 R2500 from VC 4.0, and one for extracting the axis variables from the KUKA KR 16-2 and mirror its movements in VC 4.0: `https://github.com/akselov/digital-twin-opcua/blob/master/VC4.0/Robot_Cell_MTP_case.vcmx`

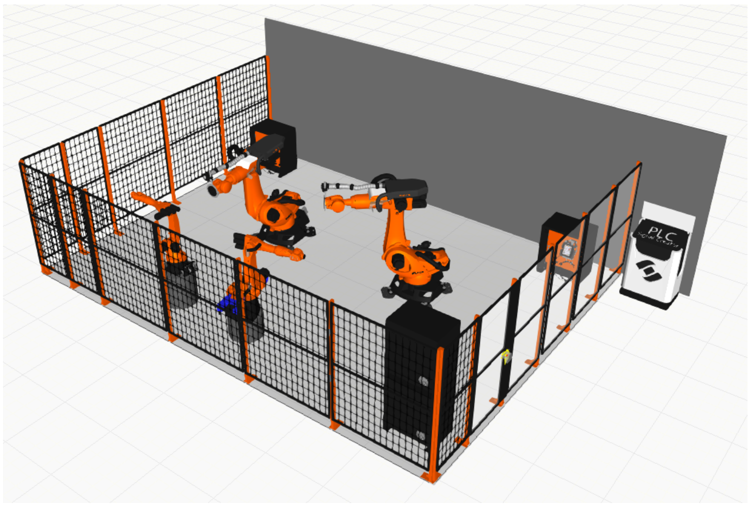


Figure 68: DTw from VC 4.0

4.2 Communication library

It was decided to create a library of various communication systems suitable for different tasks. These tasks could be linked to analysis, monitoring, and control of the system. All of the different functionalities are thoroughly described in section 3.2. Figure 69 shows an overview of the different communication systems available in appendix B.1 and the GitHub repository, and the information flow these communication systems provide between the physical robot cell and the DTw. The OPC UA client outputs data on the real-time requirement of the operation to be executed, and choose whether KVP or RSI should be used for communication.

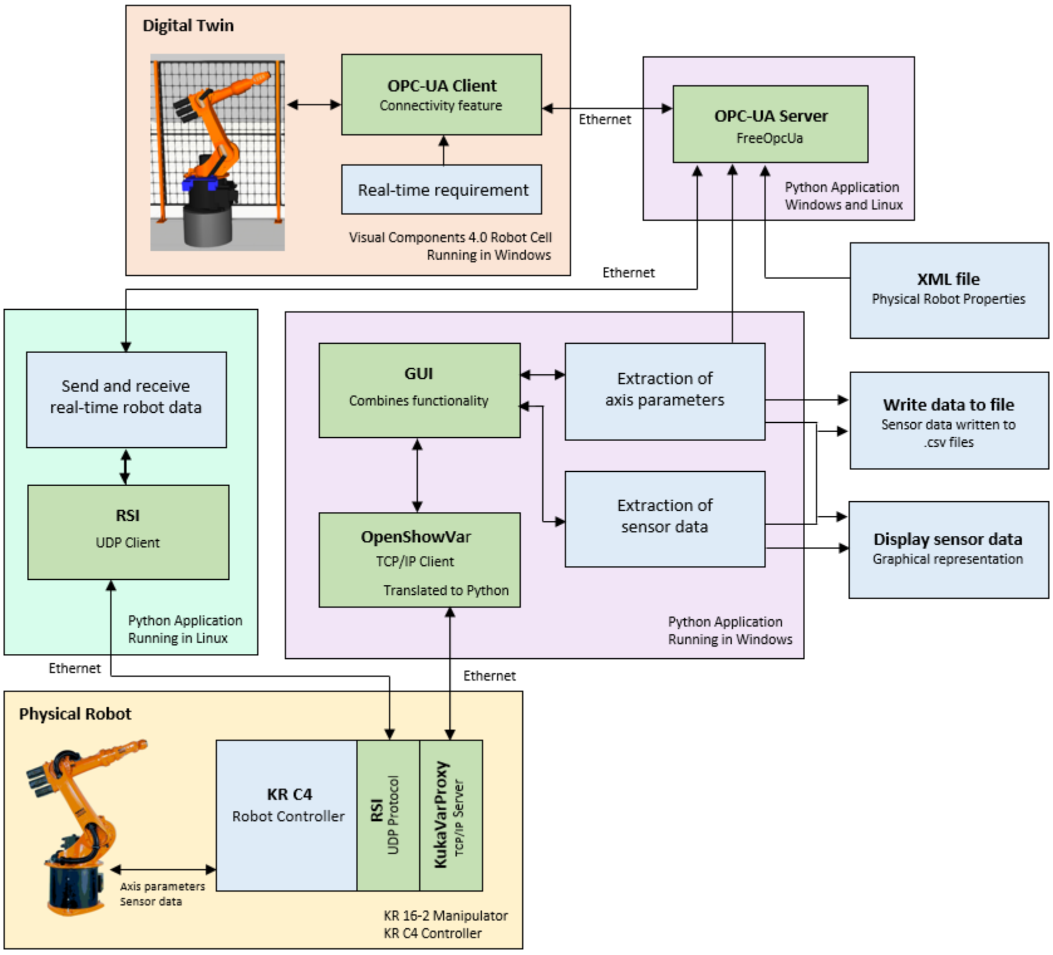


Figure 69: Final information flow

4.3 Setup

The final setup of the DTw used in the fixed case described in section 3.8 is illustrated in figure 70. Here the DTw is running in VC 4.0 and displayed on a Windows PC that is located on the side of the robot cell. The DTw is connected to a Windows laptop working as middleware between the Windows PC hosting the DTw and the KR C4 robot controllers in the robot cell. This laptop is plotting the relevant sensor information in real-time.

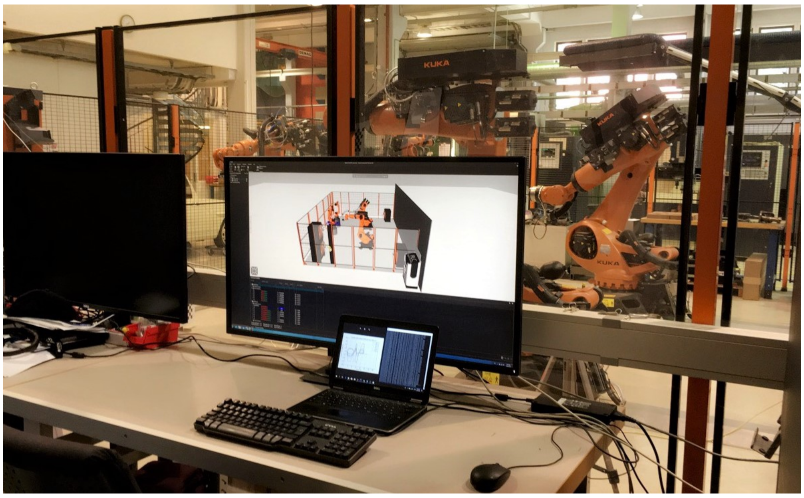
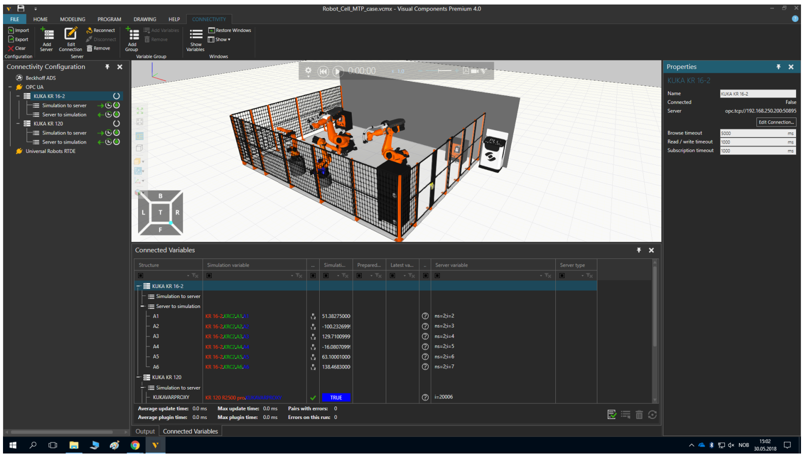


Figure 70: DTw setup in lab



Properties

KUKA KR 16-2

Name	KUKA KR 16-2
Connected	False
Server	opc.tcp://192.168.250.200:50895
Browse timeout	5000 ms
Read / write timeout	1000 ms
Subscription timeout	1000 ms

Connected Variables

Structure	Simulation variable	...	Simulati...	Prepared...	Latest va...	..	Server variable	Server type
KUKA KR 16-2								
Simulation to server								
Server to simulation								
A1	KR 16-2.KRC2.A1.A1		51.38275000				ns=2;i=2	
A2	KR 16-2.KRC2.A2.A2		-100.232699				ns=2;i=3	
A3	KR 16-2.KRC2.A3.A3		129.7100999				ns=2;i=4	
A4	KR 16-2.KRC2.A4.A4		-16.0807099				ns=2;i=5	
A5	KR 16-2.KRC2.A5.A5		63.10001000				ns=2;i=6	
A6	KR 16-2.KRC2.A6.A6		138.4683000				ns=2;i=7	
KUKA KR 120								
Simulation to server								
KUKAVARPROXY	KR 120 R2500 pro.KUKAVARPROXY	✓	TRUE				i=20006	

Average update time: 0.0 ms Max update time: 0.0 ms Pairs with errors: 0
Average plugin time: 0.0 ms Max plugin time: 0.0 ms Errors on this run: 0

Figure 71: Screenshot from VC 4.0

4.4 Functionalities

The GitHub library contains OPC UA servers giving the DTw the following functionalities:

1. Programming, planning, and control in VC 4.0
2. Mirror movements of physical robot cell in real-time using KVP, average read time 8.75ms
3. Mirror movements of physical robot cell in real-time using RSI, average read time 4.00ms
4. Plot sensor data for every KUKA robot:
    - Velocity of the motors controlling all axes
    - Torque on the motors controlling all axes
    - Current to the motors controlling all axes
    - Temperature of the motors controlling all axes
5. Write sensor data for every KUKA robot to .csv files:
    - Velocity to file
    - Torque to file
    - Current to file
    - Temperature to file
6. GUI for access to functionalities
7. Extraction of physical robot properties through XML
8. Controlling physical robots from VC 4.0 using KVP

How to set up the DTw for the different functionalities is described in chapter 3. To illustrate some of the functionalities the DTw was tested in a fixed case described in section 3.8. The result of this case was a video showing how the DTw works with the physical robot cell in real-time. This video can be found in the digital appendix B.2 and is also published online at

`https://youtu.be/xlQhQPmJwlA`

4.5 Data from fixed case

In the fixed case the Cartesian axes of the KR 16-2 and the KR 120 were plotted in real-time as illustrated in the video. These plots are found in figure 72 and 73. As the two OPC UA servers plotting these values were not started at the same point in time, the timestamps of the data from KR 16-2 and the KR 120 is not coherent. Comparing the timestamps between the data collected shows that the timestamp for the KR 120 is lagging 2.6 seconds behind the KR 16-2.

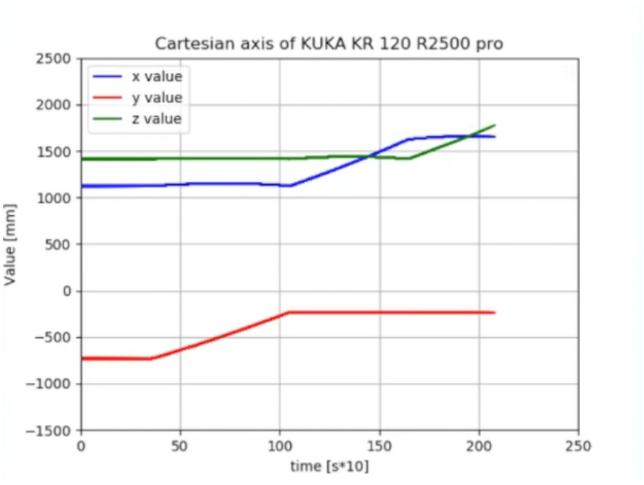


Figure 72: Cartesian axis of KR 120 R2500 pro

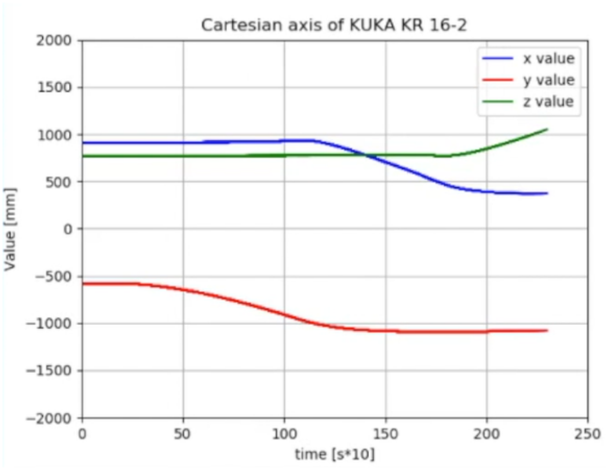


Figure 73: Cartesian axis of KR 16-2

It can be seen, from comparing the two plots, that the positive Cartesian movement of the KR 120 along the robot's y-axis results in the KR 16-2 responding with a negative Cartesian movement along its respective y-axis. This is a result of the robots facing each other in the robot cell and mirroring each other's movements. Likewise, the positive Cartesian movement of the KR 120 along the robot's x-axis results in the KR 16-2 responding with a negative Cartesian movement along its x-axis. For the final positive Cartesian movement along the z-axis, the KR 16-2 responds with a similar positive movement along its z-axis.

From the two plots, it can also be seen that while the KR 120 performs linear movement between the points, the KR 16-2 tries to follow this movement, but the visual control is not fast enough for it to completely mirror the linear motion. This results in the movement of the KR 16-2 being more smoothed out. It can also be noticed that the correction of the KR 16-2 seems to be faster along the robot's x-axis than along the y-axis.

Chapter 5

Analysis and discussion

This chapter contains an analysis of the system developed and discussion of software, data acquisition, and communication architecture. This includes a discussion of the uncertainties, safety features, and improvements that should be done to take this DTw to the next level. The chapter is concluded in a final discussion of the benefits the DTw provides, both for the institute and for the evolution towards I 4.0.

5.1 The project

The work that has been done in this project started in August 2017 and finished in June 2018. The work done in the autumn of 2017 was, for the most part, a study in the technologies that could be used to develop a DTw and what had earlier been done within this field. The result was an analysis of the use of a DTw for the robot cell at MTP Valgrinda and a digital representation of the robot cell made in VC 4.0. The work done in the spring of 2018 has for the most part been concerned with software development and the communication between the physical and the digital model. All along it has been an iterative process with a lot of tries and errors.

By far the most challenging part of this project revolved around the development of a robust communication system, complying with OPC UA standards and including the functionalities that could be useful in a DTw. In the early spring of 2018, it was decided, in consultation with supervisor Olav Egeland, that the primary focus at this stage in the project should be to get a minimal communication up and running as soon as possible. This was established a couple of months into the project and was later the cornerstone for all the following functionalities developed.

During the work with this project, it was realized how much work went into searching for documentation on implementations of software and the software itself. People had done much of this work while being in the same position as me, also in the same laboratory working on the same robot cell. Therefore, together with supervisor Olav Egeland, it was decided that a lot of effort should be put into gathering relevant documentation and libraries that would in the future be useful when working with the robot cell. This is the reason why the GitHub repository was created.

This project is unique in that way that the product is not fixed, but rather a DTw with a communication library where the user hopefully has the tools needed to set up a customized DTw satisfying the requirements for the task at hand. As a DTw can be used for a wide range of different applications, this was seen as the best solution for future use.

5.1.1 Fixed case

The goal of the fixed case was to illustrate a scenario where a DTw would be useful. It was not focused on doing the tracking algorithm based on the ViSP library as efficient as possible. The algorithm used was based on position-based control which could, with not too much effort, be improved to include a velocity based correction system.

During the case, it was discovered that using the *opcua-control-server.py* for controlling the KR 120 from VC 4.0 had some drawbacks that need improvement. Even though a program with linear PTP movements along the robots axis was created in VC 4.0, the physical robot did not keep a constant speed between the set points. This was a visual observation. To establish that the KR 120 robot was undoubtedly not moving with constant speed along the pre-planned paths the speed of the motors controlling the axes of the robot was plotted. This was done using GUI -> Display sensor data -> Display Motor Speed monitor. The result of this test can be found in figure 74. It was clear from this plot that the velocity of the motors was not constant. The same test was done with the KR 16-2 that was not controlled from VC 4.0. The plot of this test is displayed in figure 75. Comparing the two plots, it is clear that the velocity of the motors controlling the KR 16-2s axes is also not constant, but is varying much less than the motors of the KR 120, resulting in the KR 16-2 running more smoothly.

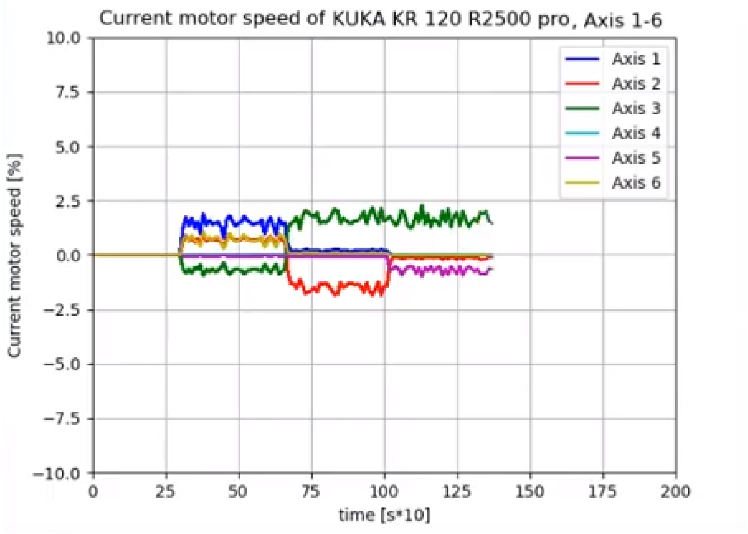


Figure 74: Motor speed of KR 120 from fixed case

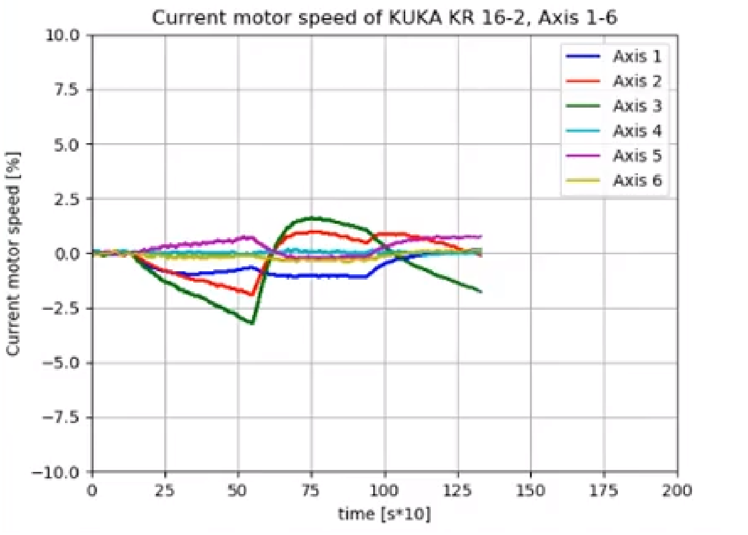


Figure 75: Motor speed of KR 16-2 from fixed case

An analysis was done to find out why the control from VC 4.0 resulted in the KR 120 robot not moving with constant velocity. The *opcua-control-server.py* was investigated to break down the steps from creating axis values in VC 4.0 from the PTP program to executing the axis values on the robot controller. The process was divided into the four steps displayed in figure 76.

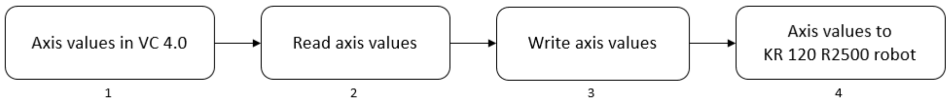


Figure 76: Break down of the control of KR 120 R2500 from VC 4.0

As step 2, 3 and 4 were executed in a similar way for controlling the KR 16-2, step 1 had to be the root of this problem. After consulting with co-supervisor Eirik Njåstad, it was made a hypothesis that this was a result of VC 4.0 not being able to produce a sufficiently constant stream of points when doing a simulation. The server developed uses the points created by the simulation in real-time. Even though the values were read from VC 4.0 fast enough the simulation would not be able to deliver a sufficient amount of points per time interval.

This hypothesis was tested using a simple program only controlling axis 1 of the KR 120 and moving it with constant speed while the stream of points from VC 4.0 was recorded and plotted. This plot can be found in figure 77.

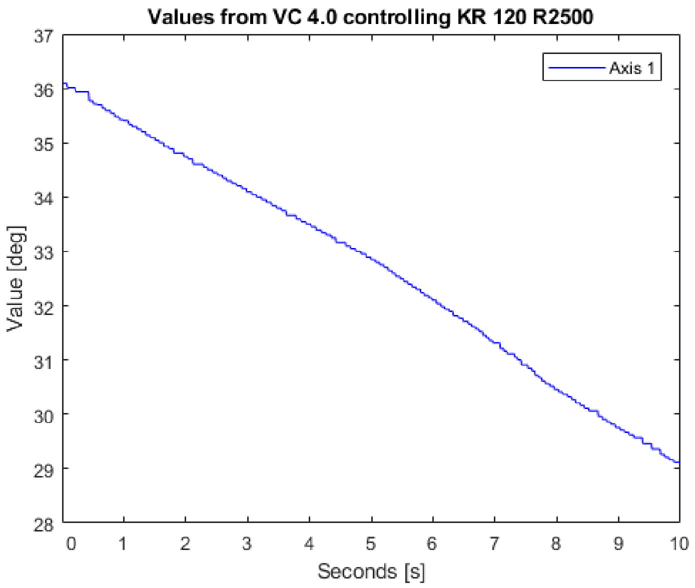


Figure 77: Axis values from VC 4.0

Analyzing this plot confirmed the hypothesis. It can be seen that in a 10-second interval the points that make up the linear movement of axis 1 is for more prolonged periods constant and do not follow the planned linear movement. The most extended interval with a constant axis value delivered form VC 4.0 was between 0.23s and 0.44s, amounting to a total of 210ms.

The effect this had on the physical KR 120 robot was investigated by plotting the axis value of axis 1 when running the same program controlling the robot from VC 4.0. The plot of this recording can be found in figure 78.

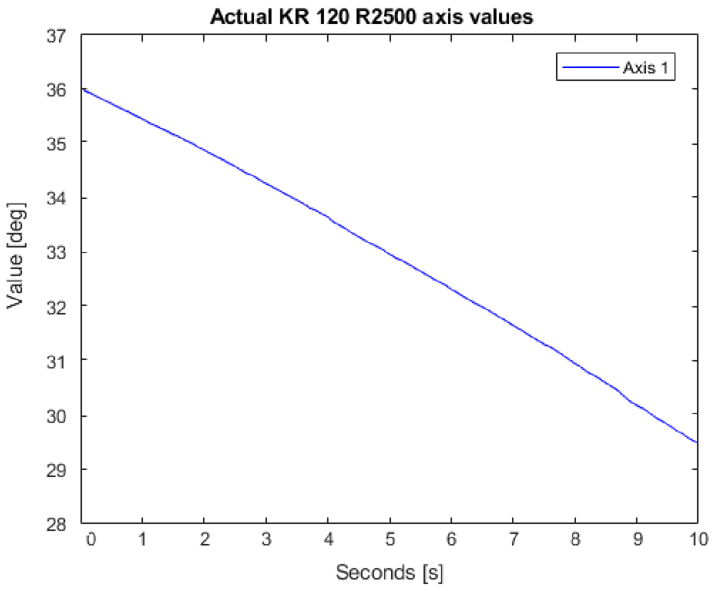


Figure 78: Axis values from physical KR 120 R2500

Comparing the two plots 77 and 78 it can be seen that the motor which is controlling axis 1 works as a low-pass filter for the stream of points. This results in the movement being smoothed out. Still, the non-linear movement could be seen by observing the robot when performing the test.

It is worth mentioning that the stream of points from VC 4.0 also does deviate from a line representing the ideal linear movement as illustrated in figure 79 and 80.

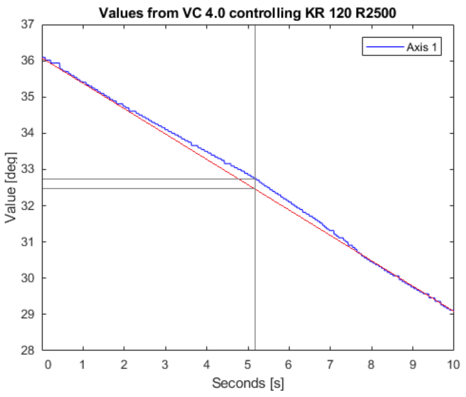


Figure 79: Values from VC 4.0

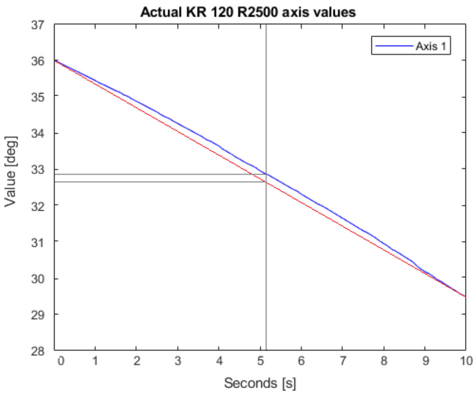


Figure 80: Values from KR 120 R2500

The maximum deviation from ideal behavior is 0.20 degrees in the values from VC 4.0 and 0.18 degrees in the values from KR 120 R2500. After consulting with co-supervisor, Eirik Njåstad, it was concluded that this probably was a result of the robot performing a PTP movement. This will result in the robot performing a close-to-linear movement of a trajectory with variable velocity profile through the movement.

5.1.2 Improvements

With a big GitHub library and a numerous communication systems developed, it is a lot that can be done to improve the functionality of the system as a whole. The VC 4.0 model of the robot cell, *Robot_Cell_MTP.vcmx*, was developed as an alpha version with an uncertainty estimated to be $\pm 1cm$. As discussed briefly in section 3.2.1 this uncertainty was accepted because the critical distances in the cell, the orientation of the robots would, in any case, need calibration for the robots to be able to work together with sufficient precision for fine-tuned tasks. A potential improvement could be to use more advanced measurements to pinpoint the exact locations of the objects in the robot cell and improve the VC 4.0 model.

Regarding the software developed this would need gradually to be improved step by step. GitHub is a platform where software development is made easy by the idea of branching and merging of code. This idea is built on the principle of a master branch with forked development branches in parallel. This is illustrated in figure 81. Developing software in this way allows it always to be a functioning master branch that can be used when the system needs to be up and running. It is encouraged for people interested in the work done in this project to fork code that has been developed and customize and improve for specific needs.

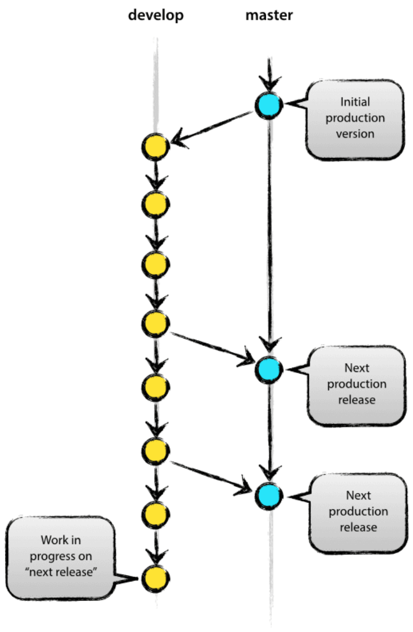

develop
master
Initial production version
Next production release
Next production release
Work in progress on "next release"

Figure 81: Git branching model [28]

For future software development, the improvements seen as most prominent is divided into four categories:

1. **Gather all functionalities in one server.** The GUI developed in section 3.4.1 was an attempt at this, but has a long way to go for it to be complete. It has to be taken into account that the server should be able to be run on both Windows and Linux considering the *rsi-opcua-server.py* has to be run from Linux for it to work. This could again be solved by the RSI communication being improved so it could run in Windows.

2. **MDA approach to OPC UA information model design.** The standardization process of this system included modularization extracting information about the system in XML-files as described in section 3.5. As described in section 2.6.1, this method can lead to the developed information models not being compatible with other solutions than the specific system from which the model is created. An improvement would be to do the information model design in line with the MDA approach suggested by [7].

3. **Avoid middleware between PC running VC 4.0 and KR C4 robot controllers.** As OPC UA is not implemented as a standard in the robot controllers in the cell used in this project, it was used middleware to open a backdoor for implementing OPC UA servers extracting and delivering information. This could be solved by the servers being run locally on the robot

controllers in the cell. As the MTP faculty is in the process of modernizing the robot cell at Valgrinda this could potentially be fixed by switching to the new generation of KUKA robot controllers, KUKA KRC ROBOTstar, which have OPC UA built in as add-on technology [115]. This is further discussed in section 5.3.

4. **Remove lag in control of robots through VC 4.0 using KVP.** This phenomenon was discussed in section 5.1.1. The root of the problem was found to be the stream of axis variables from the simulation in VC 4.0. A similar control server should be developed that uses more robust variables from VC 4.0 as output for controlling the robots.

5. **Development of an OPC UA server controlling robots through VC 4.0 using RSI.** This work was begun in the last stage of this project but was not finished. The work was based on an RSI communication architecture made by Torstein Anderssen Myhre that can be found in his GitHub repository: `https://github.com/torstem/examplecode-kukarsi-python`.

5.2 Simulation software

In section 3.1 VC 4.0 was selected to be used as simulation software for this project. KUKA.Sim was considered as an alternative, but due to this software not being available for use in this project it was never actually tested as an alternative to VC 4.0. As KUKA.Sim is developed by KUKA; it was thought this could be better suited for connections to KUKA.OPC Server 4.1 as discussed in section 3.3.1, and future servers from KUKA supporting OPC UA. However, as of December 2017, KUKA acquired Visual Components [29]. This supports VC 4.0 as a solution for the future of interconnections with KUKA systems.

„3D simulation is an important element in the design of the factory of the future. Visual Components offers innovative solutions in this field.“

Dr. Till Reuter, CEO of KUKA AG

Figure 82: KUKA acquires Visual Components [29]

The significant advantage of VC 4.0 is that it opens up the possibility for the integration of robots from other manufacturers than KUKA into the existing DTw. This includes major brands like ABB and Kawasaki. This is important when it comes to scaling. If the faculty of MTP should decide to include components of other brands than KUKA, these could also be imported as objects in the DTw.

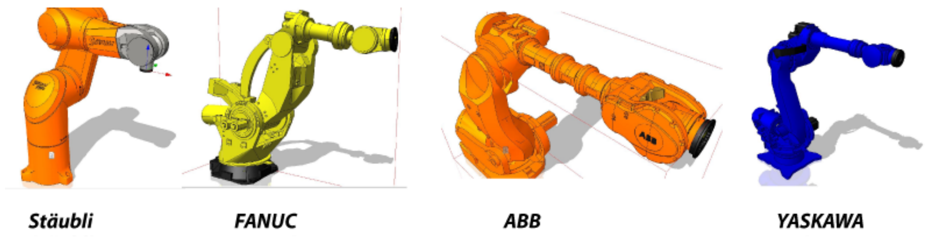


Figure 83: Some of the robot brands available in VC 4.0 [30]

VC 4.0's connectivity to OPC UA is a relatively new feature for the software. Before this being available, the connectivity configuration only included Universal Robots RTDE. This version also provides an interface for OPC that allows users to perform PLC validation by connecting a simulation to external controllers [116]. Integration of a PLC in a robot cell is done when requiring high reliability, ease of programming and sophisticated fault diagnosis. The robot cell at MTP Valgrinda includes a Siemens PLC that was included in the visual representation of the system in section 3.2.1 and could be used to validate the components in the cell. Including this also in the communication architecture is seen as beneficial for the overall structure of the communication in the robot cell.

As of April 2018, Visual Components launched VC 4.1. This version was not tested in this project. VC 4.1 has among other things a wizard to simplify the workflow for creating charts, graphs, and dashboards [117]. This improved functionality could make it easier to include plotting of sensor data from the robots in the cell directly in VC 4.1. The servers developed in this project relies on the middleware to do the plotting. As the middleware is removed in the future, as suggested as an improvement in section 5.1.2, this would be a useful feature.

5.3 Data acquisition and communication

In section 2.2.1 it was stated that every device that is activated and registered with an IoT platform has two categories of data. The first being the metadata that includes details to describe the device. In this project's DTw, much of the metadata is included in the VC 4.0 library. This could again be accompanied by specific data on every component, such as the serial number, an asset identifier, firmware version, make, model, and year of manufacturing. Data such as this should be included in XML-files for easier configuration. An implementation of OPC UA objects being configured through the use of XML-files was done in section 3.5. This complies with the standardization required in the OPC UA architecture described in section 2.6.1. However, the XML files created for the components in this robot cell only contains basic information about the robot's IP address and the joint variables. This could be extended to include more specific data.

The second type of data associated with the device represents the dynamic state. It contains context-specific, real-time data unique to the device. In this dynamic state, the system can benefit from the implementation of real-time data connected with a DTw. Through the communication library developed for information exchange between the physical robot and digital model, real-time data should be more easily available. In this project it was mainly focused on five sources of data:

1. **Axis values** and thereby the position of the robots end-effectors
2. **Velocity** of motors controlling all robots axes
3. **Torque** of motors controlling all robots axes
4. **Current** of motors controlling all robots axes
5. **Temperature** of motors controlling all robots axes

This data can provide information about how the different components are performing and interacting in the physical world, for example in comparison with a theoretical calculated performance or historical data gathered from the component. Analyzing the behavior and comparing this to the behavior of similar components could also be possible through uploading sensor data to commonly used databases. The base of analysis such as Big Data analytics is good and available sources of data. Such analysis could potentially prevent downtime, expose new opportunities and in general give a better foundation for taking operational and strategic decisions.

When deciding the communication architecture that should be chosen for use in this DTw, it was first considered using an already developed OPC server made by KUKA, named KUKA.OPC Server 4.1. This server supports OPC Classic as described in section 3.3.1. KUKA.OPC Server 4.1 was not suited for this project as it was not supporting OPC UA, but it was seen as attractive for testing as a reference to the OPC UA servers that had to be developed. As this software was not available online, KUKA Norway was contacted about the possibility of gaining access. After talking to co-supervisor Eirik Njåstad it turned out that the Institute of MTP is already in contact with KUKA Norway regarding a potential update of the KR C4 controllers in the robot cell to KRC ROBOTstar, which has OPC UA built in and the possibility for all components to be connected to the robot controller for data exchange [115]. However, this upgrade is not fully verified. The latest update on the situation, as of June 2018, is that the institute probably has to replace the whole robot park in order to get this new robot controller. This backs up the conclusion that it was still applicable to implement a solution for communication with OPC UA on the KR C4 controllers currently in the robot cell, and for all other KUKA controllers not currently supporting OPC UA connectivity.

KVP was the alternative first used to establish a communication between the physical robots and the digital model. The reasoning behind the use of this structure was done in section 3.3.2. KVP was also used as a basis for most of the OPC UA servers developed in section 3.3. It should be emphasized that the use of KVP for an alternative implementation of OPC UA is not an ideal solution. Even though the JOpenShowVar/KVP structure is tested and applied by various projects, there is no guarantee for this software to work as described. This software is open-source and has not officially been verified by any robotics manufacturer. Using KVP as a middleware can compromise the safety of the system and opens the possibility for malfunctions not experienced in a server purely run as an OPC UA server made by authorized software developers.

Use of the JOpenShowVar/KVP client/server architecture also includes a time-delay described by Sanfilippo, Hatledal and Zang in their publication *JOpenShowVar: An open-source cross-platform communication interface to KUKA robots* [26]. This analysis is carried out for Cartesian movement of an end-effector and is displayed in figure 84. In a 20-second interval, an average access time of 4.27 ms was obtained. A similar time-delay analysis was done in section 3.3.8 for the *opcua-server02.py*. With simple Cartesian movement, an average read time of 8.75ms in a 60-second interval was obtained. In section 3.6.2 it was done the same time-delay analysis for the *rsi-opcua-server.py* with an average time-delay of 4.0ms in a 60-second interval.

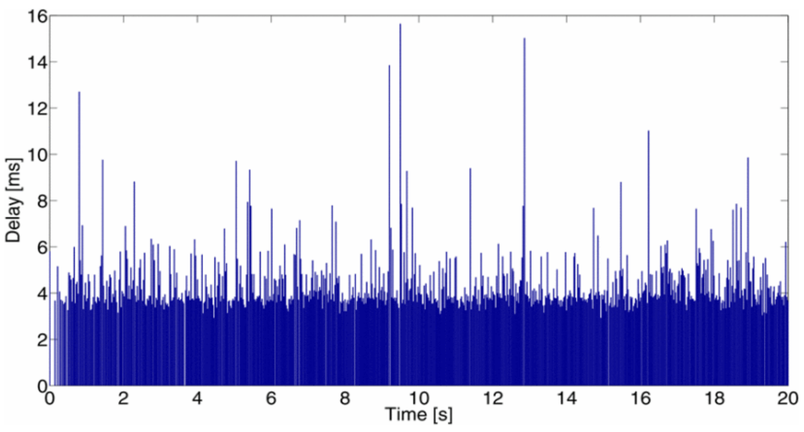


Figure 84: Time-delay analysis for use of JOpenShowVar and KVP [26]

Comparing the time-delay obtained by Sanfilippo, Hatledal and Zang with the time delay obtained in the analysis of the *opcua-server02.py*, we find a 119% increase. The reason for this is that while the JOpenShowVar/KVP client/server architecture only performs reading and writing of variables from or to the robot controller, the *opcua-server02.py* also does reading and writing of variables from or to an OPC UA client. In this case VC 4.0. This amounts to more than a doubling of the time-delay. Replacing the middleware and the use of KVP with a local OPC UA server, as discussed as an improvement in section 5.1.2, would presumably reduce the read-time and thereby lower the time-delay.

5.3.1 KVP versus RSI

As the four KUKA KR C4 robot controllers in the robot cell did not include OPC UA as an add-on technology, two different types of middleware were tested. A variety of OPC UA servers based on KVP and one server based on the RSI communication architecture had been developed. Comparing the two types of middleware it was found that RSI would have a time-delay to VC 4.0 that was in average about 46% (RSI: 4.0ms, KVP: 8.75ms) of the KVP OPC UA server with the same functionality.

The reason for this can be found in the way the RSI communication is built, as described in section 2.10.2. KVP uses TCP and RSI UDP. The difference is also due to RSI including harder real-time requirements than KVP. Because of these hard real-time requirements, the RSI server had to be run on a Linux machine as described in section 3.6.1. After using the two systems in parallel, it was seen that the RSI server had a higher possibility of breaking down while running due to these real-time requirements. As a result, KVP could be seen as a solution more reliable, but with more time-delay.

The advantages of using KVP in comparison with RSI as middleware included that the KVP server on the KR C4 robot controllers was always running in the background. This resulted in the possibility to run other tasks and programs on the smartPAD without this server being in the way. RSI had to be run as a specific program on the robot controllers losing the possibility of running other tasks and programs. However, this difference was not the case with the function of controlling the robots in the cell from VC 4.0. Here both KVP and RSI would need a specific program to be run on the KR C4 controllers for the robot to listen to updates on position.

5.3.2 OPC Toolbox

OPC Toolbox was described in section 2.10.4. Co-supervisor Eirik Njåstad discovered this software in the later stages of the project. It was therefore not tested as an alternative to VC 4.0. Through the use of this software, a simple implementation of an OPC UA client could have been developed. As this toolbox opens up the possibility of doing simultaneous data logging and numerical processing, and simultaneous connections to multiple OPC servers, this is an easy way to get DTw functionalities directly into MATLAB.

Using this software could simplify data acquisition and plotting of sensor data. MATLAB has numerous functions for processing data and performing a statistical analysis which could streamline the handling of data from the KR C4 robot controllers.

5.4 Safety and security

Safety and cyber security are becoming more and more critical in the interconnected systems used in modern manufacturing. This was mentioned in section 2.1 as one of the key areas where the German *Industrie 4.0 Working Group* believes that action is needed. Cyber security, computer security or IT security is the protection of computer systems from the theft and damage to their hardware, software or information, as well as from disruption or misdirection of the services they provide [118]. This is especially the case for systems that are directly interconnected with a physical system such as a DTw. A breach of the security related to a DTw could potentially lead to unwanted personnel gaining access to data from the physical system, and even worse, loss of control of the physical components. This could lead to systems out of control, and dangerous situations could occur.

You can only achieve a truly safe system if all existing risks are recognized in advance and, where necessary, reduced as quickly as possible. A DTw used in the manufacturing industry will also face the challenges of meeting current standards and legislation, such as IEC 62061, ISO 13849-1, ISO 14121 [119]. With regards to safety, OPC-UA is already an IEC standard (IEC 62541), and tools and test laboratories for testing and certifying conformity are available. Additionally, various validations regarding data security and functional safety have been performed by external test and certification bodies [1].

As mentioned in the previous section, use of custom-made OPC UA servers based on KVP poses a reliability issue when it comes to safety. Again it should be emphasized that the use of this system together with OPC UA can be rewarding for academical research and gaining a deeper understanding of the OPC UA connection, but should not without further research be implemented as a permanent solution in a DTw used for professional purposes. Appropriately implemented, tested, and verified OPC UA servers, such as an improved version of the KUKA.OPC Server 4.1 supporting OPC UA, would to a greater extent ensure the safety of this system. Removing middleware as proposed as an improvement in section 5.1.2 would also strengthen the safety and security of the system as it would include less hardware and interconnections prone to error or security breaches.

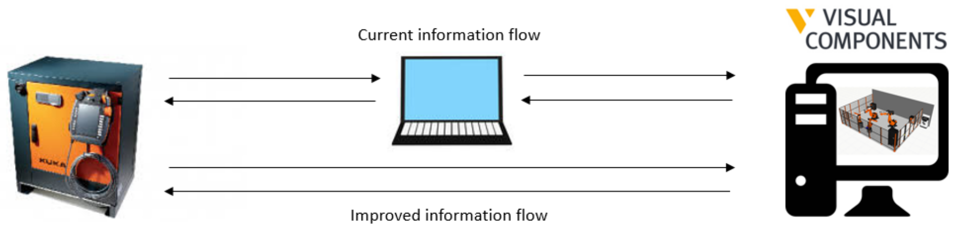


Figure 85: Visualization of improved information flow

Using the open-source FreeOpcUa created by Olivier Roulet-Dubonnet does also constitute a safety threat since it exists just a small amount of documentation on this project, and it still being work in progress. The version used in this project does also not support security such as authentication and certificates handling [120]. This software is not thoroughly tested and should like the KVP system not be implemented as a permanent solution in a DTw used for professional purposes, without further research.

Using the DTw for running real-time applications also constitute a general safety threat as the robot cell at MTP Valgrinda contains four large robots with the capability of imposing potentially hazardous situations. The main hazards identified in this robot cell is:

1. **Collision:** With itself, other components in the robot cell, the environment surrounding the cell or people working in the cell.
2. **Electrocution:** Mainly from the current supplying the robots and the KR C4 robot controllers. Especially when performing maintenance or doing change in the hardware connections to the KR C4s. Opening the KR C4 cabinet to access Ethernet ports was done several times during the work with this project.
3. **Unwanted behaviour:** Such as ejection of a tool during operation or programmed algorithms choosing unwanted paths when performing PTP movement. This was especially evident in the last part of the project when trying to develop a control server using RSI.

To minimize the risk of robot-human collisions the robot cell is fenced in and has a sensor connected to the door of the cell activating an emergency stop signal to all four KR C4 robot controllers when opened. Still, people could close the door from inside the cell, and the robots could keep running in AUT mode. This should be avoided. To avoid robots colliding with themselves, robot-robot collision and robot-environment collision, it was set up a working envelope when using the KR 16-2 robot. This limits the movement of the robot to a fixed space. This configuration was explained in detail in section 3.7 and could also be activated for the remaining robots in the cell.

5.5 Uncertainties

Uncertainty is a central concept quantifying the dispersion one may reasonably attribute to a result. Uncertainties could also be the source of error. The creation of a digital representation of a physical system includes many uncertainties because it is impossible to replicate something physical in a digital environment fully. The DTw is merely an imitation of the actual system, with as similar characteristics as possible.

The main contributors to uncertainties identified are:

1. Uncertainties related to the individual components in the physical robot cell
2. Measurements of the physical robot cell
3. Data transferring and communication
4. Use of KVP and RSI module
5. Use of FreeOpcUa

The uncertainties related to the individual components in the physical robot cell can be broken down into several sources. One source is the deviation due to changes in temperature. The KUKA robots in the cell has an allowed ambient temperature range of approximately 10 °C to 50 °C [121] [122]. Using the robots outside this range of temperature could cause their operation not being as in their documentation. As the digital representations of the components in the VC 4.0 library are built on the documentation of the robots, this could cause an error in the digital representation. Changes in temperature between the components ambient temperature is also a source of error. Extraction of information from temperature sensors connected to the motors controlling the axes of the robots was done in this project. These measurements could be used as information to the digital twin about the operating state of the robots. However, it is still difficult to calculate how the temperature differences in the robot affect the expansion and contraction of the materials which constitutes the robots for this to be included in the a more precise represented DTw.

Position repeatability of the individual components can also be characterized as a source of error. In the KUKA robots in the robot cell this repeatability lies between $\pm 0,04mm$ [121] and $\pm 0,06mm$ [122]. This is a deviation that is difficult for the digital representation to take into account as VC 4.0 does path planning assuming ideal behaviour. This problem is however evident regardless of the robot being controlled from VC 4.0 or programs run on directly on the KR C4 smart pads. Using RSI for controlling the robots, real-time adjustments and path-corrections would be possible. This is one of the main reasons why RSI also should be developed as a parallel communication to KVP as mentioned in section 5.1.2.

The measurements of the physical robot cell, done in section 3.2.1 is also a source of error. This measurement was done by hand using a tape measure. With the use of the average value of three measurements, the uncertainty was estimated to be $\pm 10mm$. These measurements give uncertainty about the orientation of the components in the robot cell. To reduce this error, digital equipment for measurement could be used. As argued in section 3.2.1 the measurements done was accepted because the only critical measurements of the system were the measurements of the distance between the KUKA robots and their orientation in the cell. The orientation of the robots would, in any case, be needing a calibration for the robots to be able to work together in tasks that demand a high level of precision. Co-supervisor Eirik Njåstad tested using camera measurements creating a point cloud of the robot cell. This could potentially improve the accuracy of the orientation of the components in the cell. The point cloud is illustrated in figure 86.

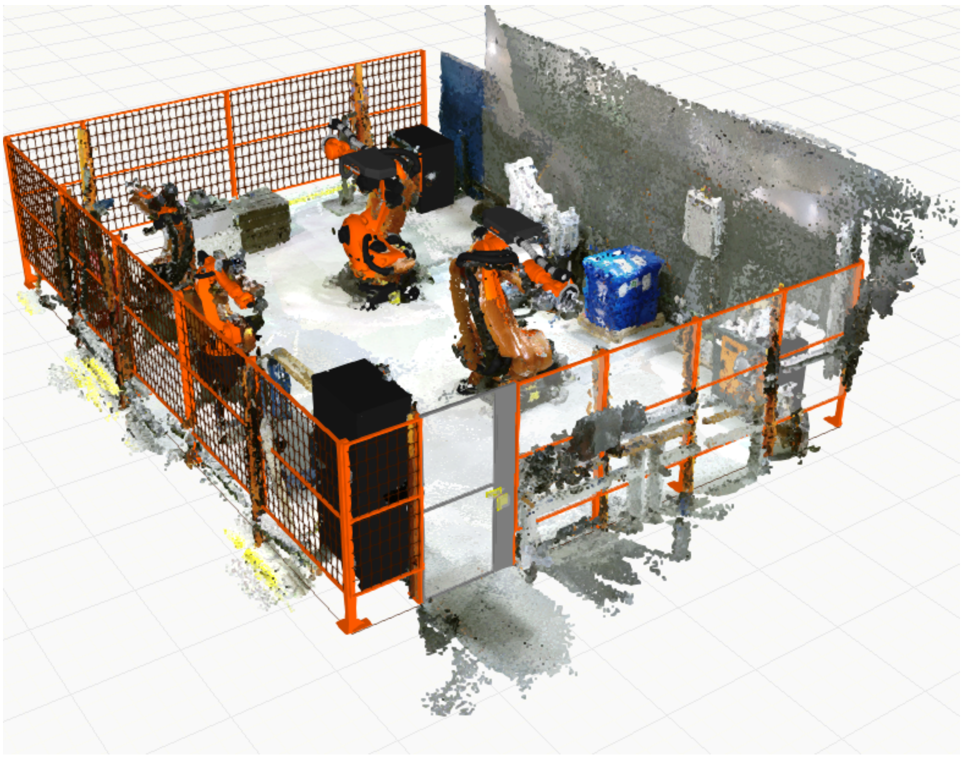


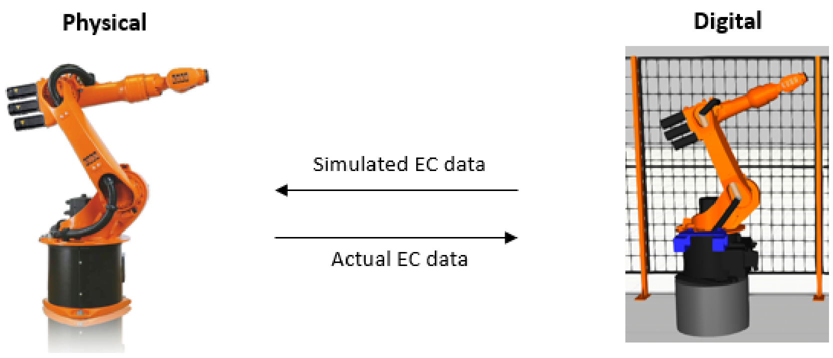
Figure 86: Point cloud combined with VC 4.0 model, made by Eirik Njåstad

The uncertainty included in the use of KVP and FreeOpcUa was mentioned in section 5.4. This uncertainty is related to the data transferring and the communication between the physical and the digital twin. As none of these has been verified to be completely bug-free, uncertainty about the data transferring is therefore also a contributing factor in this system.

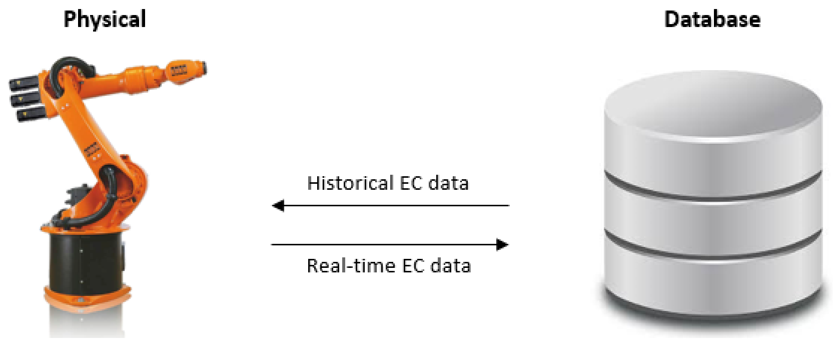
5.6 Robot cell as DTw

In section 2.2 it was presented certain characteristics related to DTws, which gives some common benefits. These characteristics will be used as the basis for the discussion on the benefits of the DTw for the robot cell at MTP Valgrinda.

1. **Visibility:** The DTw gives improved visibility and overview of the robot cell. This would especially be beneficial when performing multi-robot planning and control.
2. **Statistical and predictive analysis:** Using the programming and simulation functionality in VC 4.0, predictions on scenarios and the state of the components in the cell could be done. Sensor data from the physical components could be used for more detailed operational analysis and be processed to optimize characteristic parameters in a robot program. This data could also be used to detect an error and predict a need for maintenance.
3. **Energy Consumption (EC):** Running simulations in the digital domain requires less energy than running the actual robot movements. In the manual for the KUKA KR C4 controller, it is stated that the KUKA Power Pack, which drive power supply and generates rectified intermediate circuit voltage from an AC power supply for one KUKA KR C4, delivers power output of 14 kW with 400V supply voltage [123]. As this robot cell includes four of these controllers the power supply needed for full operation is high. As the development of robot programs often require an iterative process of testing the code on the robots, energy could be saved by this testing being transferred to the digital model. Optimizing performance on the basis of sensor data could also potentially save energy when the robot cell is used. EC of the physical system could be compared to simulated EC data to monitor the abrupt EC changes caused by sudden disturbances and compared to historical data to monitor the gradual EC changes due to component degradation. This is illustrated in figure 87.
4. **What if analysis:** With a DTw that is able to imitate the physical robot cell closely, it is possible to run simulations that would not be able in reality. Such simulations would give insight to the limitations of the system and how the interconnection between the components in the cell would react to scenarios that could not be performed in the real world.
5. **Documentation and communication mechanism to understand and explain behaviour:** With the use of OPC UA in the communication layer that interconnects the components of the robot cell, the DTw will present a better foundation for organizing, documenting and explaining behaviour and data transfer in the cell. Comparing this data to historical data obtained from similar components, as discussed in section 5.3, could also be beneficial for optimizing the use of the robots and the OPC UA structure.



(a) Physical to virtual monitoring



(b) Self-monitoring

Figure 87: EC monitoring

The fact that it was used OPC UA as a standard for communication would also open up the possibility for connecting this DTw to other similar systems running on OPC UA. In this way, this robot cell would be more equipped for future changes and in line with the vision for I 4.0.

5.7 Value for I 4.0

As discussed in section 2.1 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. However, the work done in this project can be seen as in line with the recommendations from the German *Industrie 4.0 Working Group* presented in section 2.1, which includes the vision and basic technologies promoted by the idea of I 4.0. The project delivers value to the following key areas presented by the group:

1. **Standardization and reference architecture**

   As mentioned in section 2.5, OPC UA is listed as the one and only recommendation for realizing the communication layer of RAMI 4.0. Using this standard make it possible for this DTw to connect to other systems and expand its functionality as more of the industrial communication is done with OPC UA. I 4.0 will involve networking and integration of systems across different industries. For this DTw, this networking could include a common platform for communication with other academical institutions or companies which has implemented OPC UA in their systems.

2. **Managing complex systems**

   The increasing complexity of systems used in industry presents a need for management that is different from the systems traditionally used. In these systems robots no longer just do repetitive work, but work autonomously with the help of artificial intelligence and the use of sensor technology such as computer vision. The DTw can facilitate and organize a complex system that is continuously changing. Through the processing of data from the physical robot cell, a DTw could work as a control centre for the user and make handling the system easier.

6. **Training and continuing professional development**

   As the robot cell at MTP Valgrinda is part of an educational system, it will quickly change users and undergo personnel with varying degree of relevant competence. An operating DTw with a library developed for future use could help users of the robot cell in their respective work. Also, a robot cell that is easier to use through control and analysis in VC could facilitate users in a way that is beneficial for the educational program at the faculty. This could potentially lead to more students at NTNU gaining insights to the use of this robot sell and its use for I 4.0.

8. **Resource efficiency**

   As discussed in section 5.6, the robot cell could potentially save energy by using the DTw to simulate scenarios and testing code. EC could also be decreased by comparing the physical data to the digital and historical data.

   Testing in a digital environment before testing in the physical robot cell could also reduce wear on the components in the cell and thereby extend the lifetime of the components while at the same time reduce the risk associated with trying out new software on the system.

THE THREE I'S OF INDUSTRY 4.0 INNOVATION
INCREMENTAL
Renovation, not reinvention
ITERATIVE
Experiment, fail fast, respond and adapt
INTEGRATED
Cross-functional collaboration

Figure 88: The three i's of I 4.0 innovation [31]

As mentioned in section 2.1 the journey towards I 4.0 will be an evolutionary process. This system is a way of slowly starting to integrate the functionality of modern industry, and lay the groundwork for better communication between the components that make up the robot cell at MTP Valgrinda.

Chapter 6

Conclusion

In this thesis, implementation of a DTw was studied. This study was done with a focus on how a DTw can be implemented using OPC UA, according to goal a) section 1.1. Using a DTw as a virtual representation for the robot cell at MTP Valgrinda described in section 2.9 was considered to have various benefits. The use of Visual Components 4.0 was measured up against similar simulation software according to goal b) section 1.1. VC 4.0 was seen as the strongest candidate for use in this project, and the software was used to make a virtual representation of the robot cell, according to goal c) section 1.1.

Most of the work done in this project revolved around creating communication modules able to connect the physical robot cell with the virtual representation in VC 4.0, through the use of OPC UA. The result of this work was a communication library that has been made open-source on GitHub. The library contains the virtual representation of the robot cell as well as the different communication modules able to give the DTw various functionalities. This system is the first version of a DTw for the robot cell at MTP Valgrinda, according to goal e) section 1.1. However, it is emphasized that this is an alpha version open for future development, and not a finished product.

As the four KUKA KR C4 robot controllers in the robot cell do not include OPC UA as an add-on technology, two different types of middleware were tested, according to goal d) section 1.1. A variety of OPC UA servers based on KVP and one server based on the RSI communication architecture were developed. When comparing the two types of middleware, it was found that RSI would have a time-delay to VC 4.0 that was about 46% of the KVP OPC UA server with the same functionality. It was concluded that KVP could be seen as a solution more reliable, but with more time-delay, than RSI.

The system was tried out in a fixed case, according to goal f) section 1.1. This was a multi-robot case including two of the four robots in the cell. The KUKA KR 120 R2500 was controlled from VC 4.0 while the KR 16-2 was controlled externally and mirrored in VC 4.0. From working with this case, it was concluded that the DTw gave improved visibility to the operation and made it easier to analyze the different sensor data from the robots. From this data, it was found that the control from VC 4.0 poses the problem of the simulation not being able to deliver a sufficient stream of iterative points, something that made the robot lag slightly when moving. This should be analyzed more closely and improved in further work with the DTw.

An analysis of the safety and security of the robot cell was done. It was concluded that the system potentially could impose serious harm to both humans and environment. As some of the functionalities in the DTw directly control physical assets with allot of power, the DTw should be used with caution. It is also emphasized that as the system is open-source and not approved or tested by any authorized personnel, a permanent solution for professional purposes should be thoroughly tested.

Before developing the DTw, it was stated that the general benefits for such systems include visibility to operations of machines and systems, modeling techniques used to predict future states, and a foundation for optimizing characteristic parameters and improve the use of the physical asset that is modeled. After developing the DTw, it was concluded that these benefits would also be valid for the robot cell at MTP Valgrinda, as well as the DTw being a good foundation for statistical and predictive analysis with Energy Consumption (EC) as one of the main areas of interest. Analysis of EC data could both be done by comparing actual EC data with simulated EC data, and by comparing real-time EC data with historical EC data.

Concerning I 4.0, it was concluded that the DTw would represent one step in an evolutionary process and give benefits such as a new foundation for managing a complex system that can be used in training and professional development at the institute. This would be beneficial for NTNU as OPC UA is listed as the one and only recommendation for realizing the communication layer of the Reference Architecture Model for Industrie 4.0 (RAMI 4.0) created by the German companies ZVEI, VDMA and BITKOM, and will most likely be an essential part of the industry of the future.

6.1 Further work

As mentioned in section 5.3 the Institute of MTP is already in contact with KUKA Norway regarding the possibility for updating the KR C4 controllers in the robot cell to KRC ROBOTstar, which has OPC UA built in [115]. This could improve the communication flow of the system by removing middleware as illustrated in figure 85. With a base in the hardware currently in the robot cell, further essential work revolves around software development and improving the communication modules that have been developed in this project. The first steps could be the following:

1. Develop RSI server controlling robots from VC 4.0
2. Include the RSI modules in GUI
3. Move plotting of data from middleware to VC 4.0
4. Include camera stream in VC 4.0

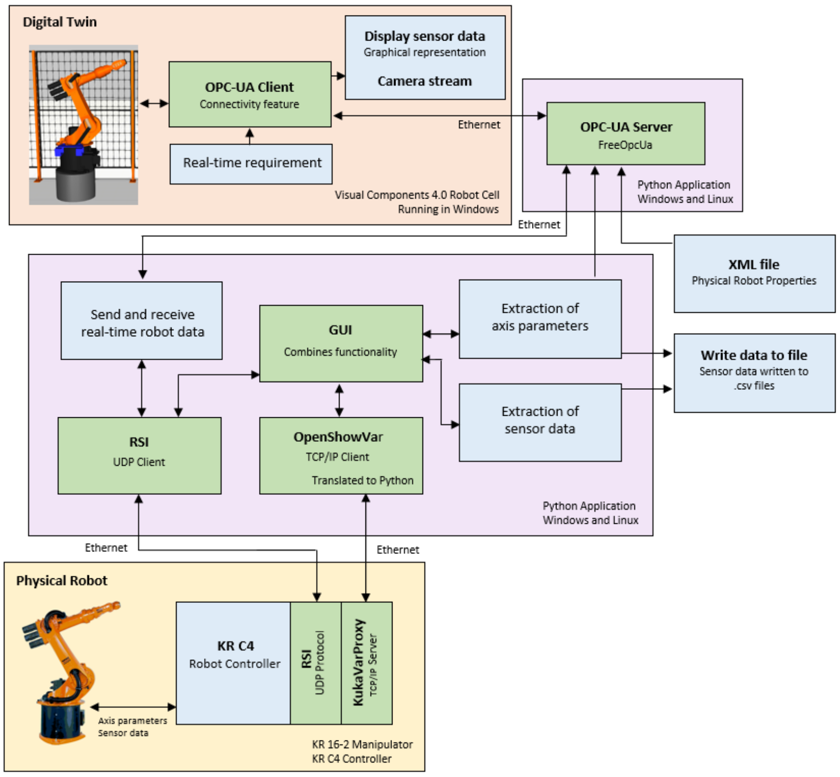


Figure 89: Improved information flow

As described in section 5.5 the visual representation of the robot cell in VC 4.0 include uncertainty about the orientation of the components, estimated to be $\pm 10mm$. Further work on this DTw should include a more accurate calibrated model of the robot cell. This could be done for example by creating a detailed point cloud of the cell with an accuracy below $10mm$, and base the visual representation in VC 4.0 on measurements from the point cloud.

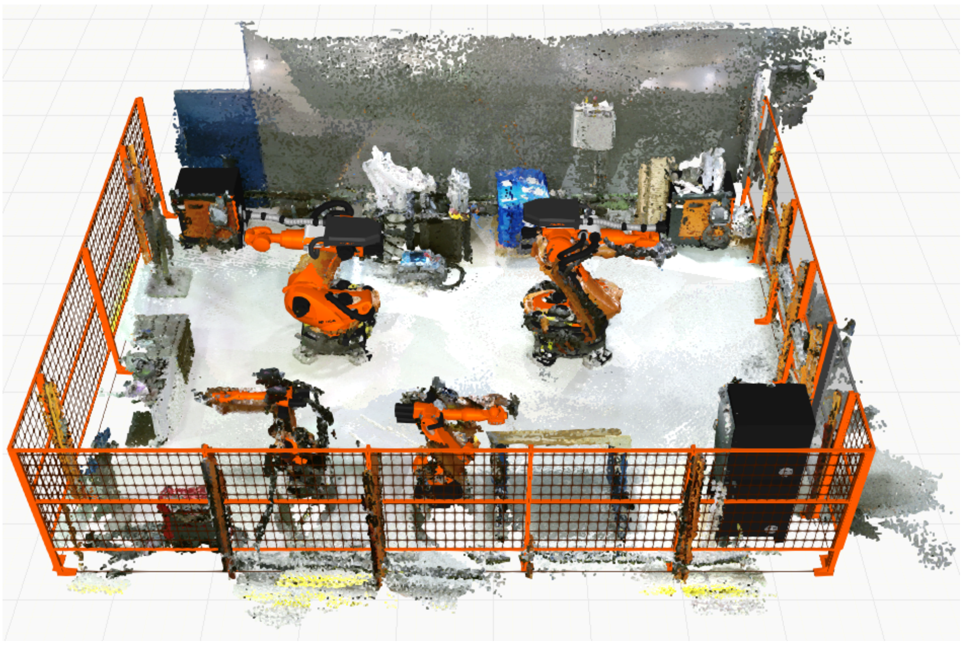


Figure 90: Point cloud combined with VC 4.0 model, made by Eirik Njåstad

As OPC UA have characteristics such as platform independence, scalability, high availability and internet capability [124], further work should also include the development of software architecture making it possible for the data from the robot cell at MTP Valgrinda to be shared to online databases as described in section 5.3. Databases like this, could, with the use of cloud technology, open up the possibility of making this robot cell a part of a bigger picture. In this way, insights could be gained from historical data being spread across both academia and industry.

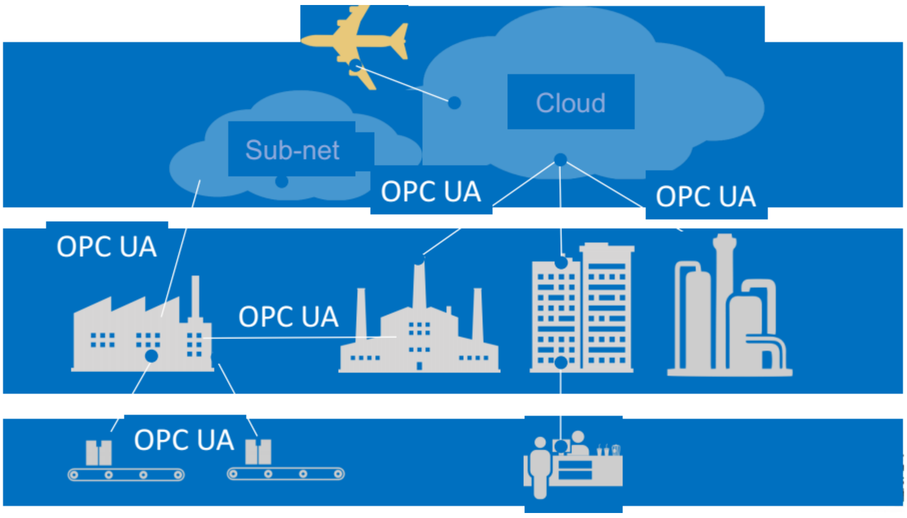


Figure 91: Future vision for OPC UA [32]

Bibliography

[1] OPC Foundation. Opc unified architecture. https://opcfoundation.org/wp-content/uploads/2016/05/OPC-UA-Interoperability-For-Industrie4-and-IoT-EN-v5.pdf, 2016. Accessed: 29-09-2017.

[2] IBM. What is digital twin? https://www.ibm.com/blogs/internet-of-things/digital-twin/. Accessed: 08-12-2017.

[3] Christoph Roser. How industry 4.0 came into existence. http://www.allaboutlean.com/industry-4-0/, 2015. Accessed: 06-11-2017.

[4] DNV GL. Digital twin. http://community.aras.com/en/exploring-digital-twin/, 2017. Accessed: 18-10-2017.

[5] Denis Sproten. Wind turbine. https://www.linkedin.com/pulse/industry-4zero-iot-digital-manufacturing-denis-sproten/, 2017. Accessed: 05-11-2017.

[6] Novotek Corp. Opc and opc ua explained. https://www.novotek.com/en/solutions/kepware-communication-platform/opc-and-opc-ua-explained, 2017. Accessed: 06-10-2017.

[7] Florian Pauker, Thomas Frühwirth, Burkhard Kittl, and Wolfgang Kastner. A systematic approach to opc ua information model design. *Procedia CIRP*, 57:321–326, 2016.

[8] OPC Foundation. Security layers. http://wiki.opcfoundation.org/index.php/File:SecurityLayers.jpg, 2017. Accessed: 18-10-2017.

[9] Platform Industrie 4.0. Reference architectural model industrie 4.0 (rami 4.0). https://www.plattform-i40.de/I40/Redaktion/EN/Downloads/Publikation/rami40-an-introduction.pdf?__blob=publicationFile&v=4, 2017. Accessed: 05-11-2017.

[10] Tina Petzold. New member – visual components oy. https://www.automationml.org/o.red.c/news-179.html, 2016. Accessed: 06-11-2017.

[11] Hannovermesse. 3d simulation with visual components. http://www.hannovermesse.de/product/3d-simulation-with-visual-components/2238209/C171973. Accessed: 06-11-2017.

[12] KUKA. Technology kuka.sim. https://www.kuka.com/-/media/kuka-downloads/imported/d5d0533e621841d296c8cc5ef289258d/kuka_sim_en.p. Accessed: 12-11-2017.

[13] Siemens. Scada system simatic wincc v7. http://w3.siemens.com/mcms/human-machine-interface/en/visualization-software/scada/Pages/Default.aspx. Accessed: 06-11-2017.

[14] Eirik Bjørndal Njåstad. Robotic welding with corrections from 3d-camera. https://brage.bibsys.no/xmlui/handle/11250/2351215. Accessed: 06-06-2018.

[15] Direct Industry. Articulated robot / 6-axis / surface treatment / for assembly. http://www.directindustry.com/prod/kuka-roboter-gmbh/product-17587-1439831.html. Accessed: 08-11-2017.

[16] RobotWorx. Kuka kr c4 controller. https://www.robots.com/kuka/controller/kr-c4. Accessed: 08-11-2017.

[17] Direct Industry. Articulated robot / 6-axis / surface treatment / for assembly. http://www.directindustry.com/prod/kuka-roboter-gmbh/product-17587-417160.html. Accessed: 08-11-2017.

[18] Direct Industry. Articulated robot / 6-axis / surface treatment / for assembly. http://www.directindustry.com/prod/kuka-roboter-gmbh/product-17587-417172.html. Accessed: 08-11-2017.

[19] KUKA. Kuka.robotsensorinterface 2.3. http://vip.gatech.edu/wiki/images/3/3c/KUKARobotSensorInterface.pdf. Accessed: 05-12-2017.

[20] Oodles. Tcp vs udp. https://www.oodlestechnologies.com/blogs/Why-UDP-is-preferred-for-Live-Streaming. Accessed: 06-06-2018.

[21] ResearchGate. Joints and links of robot. https://www.researchgate.net/figure/Joints-and-Links-of-Robot_fig4_301895257. Accessed: 07-06-2018.

[22] Villani Siciliano, Sciavicco and Oriolo. *Robotics.* Springer-Verlag London Limited, London, England, 2010.

[23] Kevac Djuric, Filipovic. Graphical representation of the significant 6r kuka robots spaces. https://pdfs.semanticscholar.org/a012/0bf2649ac0b9db1080988ed8819e4c01a6a9.pdf. Accessed: 07-06-2018.

[24] ResearchGate. Denavit-hartenberg kinematic parameters. https://www.researchgate.net/figure/Denavit-Hartenberg-kinematic-parameters-4_fig1_257067449. Accessed: 07-06-2018.

[25] KUKA. Kuka.opc server 4.1. http://supportwop.com/IntegrationRobot/content/6-Syst%C3%A8mes_int%C3%A9grations/OPC-Server/KST_OPC_Server_41_en.pdf. Accessed: 20-11-2017.

[26] F. Sanfilippo, L. I. Hatledal, H. Zhang, M. Fago, and K. Y. Pettersen. Jopenshowvar: An open-source cross-platform communication interface to kuka robots. In *2014 IEEE International Conference on Information and Automation (ICIA)*, pages 1154–1159, July 2014.

[27] April robotics laboratory. Apriltag. https://april.eecs.umich.edu/software/apriltag.html. Accessed: 31-05-2018.

[28] Vincent Driessen. A successful git branching model. https://nvie.com/posts/a-successful-git-branching-model/. Accessed: 01-06-2018.

[29] KUKA. Kuka invests in the factory of the future. https://www.kuka.com/en-de/press/news/2017/12/kuka-akquiriert-visual-components. Accessed: 31-05-2018.

[30] SoliCAD. Visual components - robotická simulace. http://solicad.com/c/visual-components-robotizace. Accessed: 02-12-2017.

[31] BDO. The middle market manufacturer's roadmap to industry 4.0. https://www.bdo.com/insights/industries/manufacturing-distribution/the-middle-market-manufacturer-s-roadmap-to-in-(1)/the-middle-market-manufacturer-s-roadmap-to-indust. Accessed: 02-06-2018.

[32] Kudzai Manditereza. How opc ua works and its role in industrial iot. https://www.linkedin.com/pulse/how-opc-ua-works-its-role-industrial-iot-kudzai-manditereza/. Accessed: 02-12-2017.

[33] OPC Foundation. What is opc? https://opcfoundation.org/about/what-is-opc/, 2017. Accessed: 05-10-2017.

[34] IGI Global. What is xml. https://www.igi-global.com/dictionary/application-semantic-web-technology-business/32919, 2017. Accessed: 07-10-2017.

[35] OPC Foundation. Analyzer device integration (adi). http://wiki.opcfoundation.org/index.php/Analyzer_Device_Integration_(ADI), 2017. Accessed: 06-10-2017.

[36] Miriam Schleipen, Syed-Shiraz Gilani, Tino Bischoff, and Julius Pfrommer. Opc ua & industrie 4.0 - enabling technology with high diversity and variability. *Procedia CIRP*, 57:315–320, 2016.

[37] IGI Global. What is ict. https://www.igi-global.com/dictionary/enhancing-autonomy-persons-intellectual-impairments/13620, 2017. Accessed: 07-06-2018.

[38] Microsoft. The component object model. https://msdn.microsoft.com/en-us/library/windows/desktop/ms694363(v=vs.85).aspx, 2018. Accessed: 07-06-2018.

[39] Microsoft. Distributed component object model. https://msdn.microsoft.com/en-us/library/6zzy7zky.aspx, 2017. Accessed: 01-06-2018.

[40] Microsoft. Object linking and embedding. https://msdn.microsoft.com/en-us/library/19z074ky.aspx, 2018. Accessed: 03-05-2018.

[41] ITU. Internet of things global standards initiative. http://www.itu.int/en/ITU-T/gsi/iot/Pages/default.aspx, 2016. Accessed: 19-10-2017.

[42] Cisco. What is the internet of everything? http://ioeassessment.cisco.com/learn/ioe-faq, 2017. Accessed: 06-11-2017.

[43] S. F. Su, I. J. Rudas, J. M. Zurada, M. J. Er, J. H. Chou, and D. Kwon. Industry 4.0: A special section in ieee access. *IEEE Access*, 5:12257–12261, 2017.

[44] Microsoft. Understanding service-oriented architecture. https://msdn.microsoft.com/en-us/library/aa480021.aspx, 2017. Accessed: 03-06-2018.

[45] Interaction Design Foundation. Socio-technical systems. https://www.interaction-design.org/literature/topics/socio-technical-systems, 2017. Accessed: 06-06-2018.

[46] M.M.M. Sarcar K. Lalit Narayan, K. Mallikarjuna Rao. *Computer Aided Design in Manufacturing*. Prentice-Hall of India Private Limited, New Delhi, India, 2008.

[47] ABCO Automation. 3d simulation software: Visual components 4.0. https://goabco.com/3d-simulation-software/, 2017. Accessed: 06-11-2017.

[48] Microsoft. .net. https://www.microsoft.com/net/. Accessed: 06-11-2017.

[49] Vangie Beal. Api - application program interface. https://www.webopedia.com/TERM/A/API.html. Accessed: 06-06-2018.

[50] Arun Singh. The active impact of human computer interaction (hci) on economic, cultural and social life. *The IIOAB Journal*, 8:141–146, 09 2017.

[51] Dzineia. Scada systems. http://www.dzineia.com/scada-systems/. Accessed: 06-06-2018.

[52] Massimiliano Fago. openshowvar. https://github.com/cyberpro4/openshowvar/blob/master/src/openshowvardock.cpp. Accessed: 25-11-2017.

[53] Computer Hope. Graphical user interface. https://www.computerhope.com/jargon/g/gui.htm. Accessed: 10-05-2018.

[54] OMG. Mda - the architecture of choice for a changing world. https://www.omg.org/mda/, 2018. Accessed: 06-06-2018.

[55] Dr. Michael Grieves. Digital twin: Manufacturing excellence through virtual factory replication. http://innovate.fit.edu/plm/documents/doc_mgr/912/1411.0_Digital_Twin_White_Paper_Dr_Grieves.pdf, 2014. Accessed: 19-10-2017.

[56] Gartner. Gartner top 10 strategic technology trends for 2018. https://www.gartner.com/smarterwithgartner/gartner-top-10-strategic-technology-trends-for-2018/. Accessed: 03-06-2018.

[57] Forbes. What is digital twin technology. https://www.forbes.com/sites/bernardmarr/2017/03/06/what-is-digital-twin-technology-and-why-is-it-so-important/#40fb27fa2e2a, 2017. Accessed: 18-10-2017.

[58] Kasey Panetta. Gartner's top 10 strategic technology trends for 2017. https://www.gartner.com/smarterwithgartner/gartners-top-10-technology-trends-2017/, 2016. Accessed: 05-11-2017.

[59] Roland Rosen, Georg von Wichert, George Lo, and Kurt D. Bettenhausen. About the importance of autonomy and digital twins for the future of manufacturing. *IFAC-PapersOnLine*, 48(3):567 – 572, 2015. 15th IFAC Symposium onInformation Control Problems inManufacturing.

[60] Elisa Negri, Luca Fumagalli, and Marco Macchi. A review of the roles of digital twin in cps-based production systems. *Procedia Manufacturing*, 11:939 – 948, 2017. 27th International Conference on Flexible Automation and Intelligent Manufacturing, FAIM2017, 27-30 June 2017, Modena, Italy.

[61] Mario Hermann, Tobias Pentek, and Boris Otto. Design principles for industrie 4.0 scenarios. In *System Sciences (HICSS), 2016 49th Hawaii International Conference*, pages 3928–3937. IEEE, 2016.

[62] Department of Industry Innovation Australian Government and Science. Industry 4.0. https://industry.gov.au/industry/Industry-4-0/Pages/default.aspx, 2018. Accessed: 11-06-2018.

[63] Forbes. What everyone must know about industry 4.0. https://www.forbes.com/sites/bernardmarr/2016/06/20/what-everyone-must-know-about-industry-4-0/#37a8375c795f, 2016. Accessed: 11-06-2018.

[64] Rainer Drath and Alexander Horch. Industrie 4.0: Hit or hype? [industry forum]. *Industrial Electronics Magazine, IEEE*, 8(2):56–58, June 2014.

[65] Dr. Johannes Helbig Prof. Dr. Henning Kagermann, Prof. Dr. Wolfgang Wahlster. Recommendations for implementing the strategic initiative

industrie 4.0. In *Securing the future of German manufacturing industry.* acatech – National Academy of Science and Engineering, 2013. Accessed: 06-11-2017.

[66] Roland Rosen, Georg Von Wichert, George Lo, and Kurt D. Bettenhausen. About the importance of autonomy and digital twins for the future of manufacturing. *IFAC PapersOnLine*, 48(3):567–572, 2015.

[67] Oracle. Digital twins for iot applications. `http://www.oracle.com/us/solutions/internetofthings/digital-twins-for-iot-apps-wp-3491953.pdf`, 2017. Accessed: 19-10-2017.

[68] Donna H. Rhodes Jack B. Reid. Digital system models: An investigation of the non-technical challenges and research needs. *Massachusetts Insttute of Technology*, 2016.

[69] US Department of Defense (DoD) w/Gary Hagan. Glossary: Defense acquisition acronyms and terms. `http://seari.mit.edu/documents/preprints/REID_CSER16.pdf`, 2015. Accessed: 05-11-2017.

[70] GE Digital. Minds + machines: Meet a digital twin. `https://www.youtube.com/watch?v=2dCz3oL2rTw`, 2016. Accessed: 19-10-2017.

[71] Siemens. Twins with potential. `https://www.siemens.com/customer-magazine/en/home/industry/digitalization-in-machine-building/the-digital-twin.html`, 2017. Accessed: 18-10-2017.

[72] IBM. What is the internet of things? `https://www.ibm.com/blogs/internet-of-things/what-is-the-iot/`, 2016. Accessed: 11-06-2018.

[73] Janakiram MSV. The role of device twins in industrial iot solutions. `https://www.forbes.com/sites/janakirammsv/2017/10/30/the-role-of-device-twins-in-designing-industrial-iot-solutions/#3b47a86820ac`, 2017. Accessed: 05-11-2017.

[74] OPC Foundation. Classic. `https://opcfoundation.org/about/opc-technologies/opc-classic/`, 2017. Accessed: 05-10-2017.

[75] OPC Foundation. Unified architecture. `https://opcfoundation.org/about/opc-technologies/opc-ua/`, 2017. Accessed: 05-10-2017.

[76] Sven Goldstein OPC Foundation. Opc ua information model for advanced manufacturing (industry 4.0). `https://www.hiindustryexpo.com/activities/digitalisation-conference`, 2017. Accessed: 05-05-2018.

[77] OPC Connect. Opc ua in the reference architecture model rami 4.0. `http://opcconnect.opcfoundation.org/2015/06/opc-ua-in-the-reference-architecture-model-rami-4-0/`, 2017. Accessed: 05-11-2017.

[78] ZVEI Die Elektroindustrie. Zvei explains rami 4.0. `https://www.youtube.com/watch?v=NO4qWRVvMiU`.

[79] D. A. Cruse. *Meaning in Language: An Introduction to Semantics and Pragmatics*. Oxford University Press, 2004.

[80] GE w/Matt Wells. The industrial internet and interoperability. https://www.ge.com/digital/blog/opc-standard-offers-new-horizons, 2017. Accessed: 05-11-2017.

[81] Wolfgang Mahnke Stefan-Helmut Leitner. Opc ua – service-oriented architecture for industrial applications. http://pi.informatik.uni-siegen.de/stt/26_4/01_Fachgruppenberichte/ORA2006/07_leitner-final.pdf, 2006. Accessed: 06-06-2018.

[82] OMG. Object management group model driven architecture (mda), mda guide rev. 2.0. http://www.smallake.kr/wp-content/uploads/2016/04/ormsc-14-06-01-2.pdf, 2014. Accessed: 06-06-2018.

[83] Visual Components. A highly developed ready-made solution. http://www.visualcomponents.com/insights/case-studies/kuka/, 2014. Accessed: 06-11-2017.

[84] Siemens. Process simulate virtual commissioning. https://www.pmcorp.com/Portals/5/_Downloads/b07_Process%20Simulate%20Virtual%20Commissioning.pdf, 2017. Accessed: 11-06-2018.

[85] Jot Automation Group. Annual report. http://web.lib.hse.fi/FI/yrityspalvelin/pdf/1999/Eelektrobit1999.pdf, 1999. Accessed: 06-11-2017.

[86] United States MI. Visual components company profile. https://www.robotics.org/company-profile-detail.cfm/Supplier/Visual-Components/company/833, 2017. Accessed: 06-11-2017.

[87] Visual Components. Introducing visual components 4.0.4. http://www.visualcomponents.com/insights/blog/introducing-visual-components-4-0-4/. Accessed: 04-12-2017.

[88] Visual Components. Essentials. http://www.visualcomponents.com/products/visual-components-4-0/essentials/. Accessed: 06-11-2017.

[89] Visual Components. Visual components 4.0. http://www.visualcomponents.com/products/visual-components-4-0/, 2017. Accessed: 06-11-2017.

[90] Visual Components. Robots ecatalog. http://www.visualcomponents.com/insights/blog/new-robots-ecatalog/, 2017. Accessed: 06-11-2017.

[91] Tyler Yanta. In-house solutions introduces octopuz robotic and simulation software. http://www.inhousesolutions.com/2014/01/house-solutions-introduces-octopuz-robotic-simulation-software/, 2014. Accessed: 06-11-2017.

[92] KUKA. Kuka.sim software. https://www.kuka.com/en-de/products/robot-systems/software/planning-project-engineering-service-safety/kuka_sim. Accessed: 12-11-2017.

[93] KUKA. Kuka kr c4 robot controller uses ethercat. https://www.pc-control.net/pdf/012014/solutions/pcc_0114_kuka_e.pdf. Accessed: 20-11-2017.

[94] KUKA. Kuka digital domains. https://www.kuka.com/en-us/technologies/industrie-4-0/industry-4-0-digital-domains. Accessed: 20-11-2017.

[95] ICS-CERT. Siemens simatic hmi devices. https://ics-cert.us-cert.gov/advisories/ICSA-15-099-01E. Accessed: 06-11-2017.

[96] Siemens. Scada system simatic wincc - system overview. http://w3.siemens.com/mcms/human-machine-interface/en/visualization-software/scada/Pages/system-overview.aspx. Accessed: 06-11-2017.

[97] Siemens. Your gateway to automation in the digital enterprise. https://c4b.gss.siemens.com/resources/images/articles/dffa-b10161-00-7600.pdf. Accessed: 08-11-2017.

[98] Delfoi. Offline programming. https://www.delfoi.com/web/solutions/robotiikka/en_GB/offline/, 2018. Accessed: 06-06-2018.

[99] Manufacturing Institue. The new hire: How a new generation of robots is transforming manufacturing. https://www.nist.gov/sites/default/files/documents/mep/data/TheNewHire.pdf. Accessed: 06-06-2018.

[100] IMTS. Kukavarproxy. https://www.imts.eu/open-sourcing-kukavarproxy/. Accessed: 06-06-2018.

[101] Ivar Eriksen. Setup and interfacing of a kuka robotics lab. https://github.com/aauc-mechlab/JOpenShowVar. Accessed: 09-05-2018.

[102] Ross Kurosem. *Computer Networking, A Top-Down Approach.* Pearson Education Limited, Harlow, England, 2017.

[103] MathWorks. Opc toolbox. https://se.mathworks.com/products/opc.html, 2018. Accessed: 11-06-2018.

[104] MathWorks. Opc toolbox fetures. https://se.mathworks.com/products/opc/features.html, 2018. Accessed: 11-06-2018.

[105] Simen Hagen Bredvold. Robotic welding of tubes with correction from 3d vision and force control. https://brage.bibsys.no/xmlui/bitstream/handle/11250/2404424/15620_FULLTEXT.pdf?sequence=1. Accessed: 20-11-2017.

[106] Mechatronics Lab at Ålesund University College. Jopenshowvar. https://brage.bibsys.no/xmlui/bitstream/handle/11250/2490915/16482_FULLTEXT.pdf?sequence=1&isAllowed=y. Accessed: 06-06-2018.

[107] Sintef. Sintef employees. https://www.sintef.no/en/all-employees/employee/?empId=4104. Accessed: 22-11-2017.

[108] Olivier Roulet-Dubonnet. python-opcua. https://github.com/FreeOpcUa/python-opcua. Accessed: 10-05-2018.

[109] Bradley Mitchell. 0.0.0.0. https://www.lifewire.com/four-zero-ip-address-818384. Accessed: 10-05-2018.

[110] Python Software Foundation. Installing packages. https://packaging.python.org/tutorials/installing-packages/. Accessed: 10-05-2018.

[111] S. Kudrle, M. Proulx, P. Carrières, and M. Lopez. Fingerprinting for solving a/v synchronization issues within broadcast environments. *SMPTE Motion Imaging Journal*, 120(5):36–46, July 2011.

[112] KUKA. Kuka.robotsensorinterface 3.1. http://supportwop.com/IntegrationRobot/content/6-Syst%C3%A8mes_int%C3%A9grations/RobotSensorInterface/KST_RSI_31_en.pdf. Accessed: 15-05-2018.

[113] KUKA. Kuka system variables. http://www.wtech.com.tw/public/download/manual/kuka/krc4/KUKA%20System%20Variables%208.1%208.2%208.3.pdf. Accessed: 13-05-2018.

[114] ViSP. Visual servoing platform. http://visp-doc.inria.fr/doxygen/visp-daily/index.html. Accessed: 31-05-2018.

[115] KUKA. Krc robotstar: a star among controllers. https://www.kuka.com/en-de/products/robot-systems/robot-controllers/krc-robotstar. Accessed: 05-12-2017.

[116] Visual Components. The importance of plc validation in manufacturing. http://www.visualcomponents.com/insights/articles/plc-validation-manufacturing/. Accessed: 04-12-2017.

[117] KUKA. Introducing visual components 4.1. https://www.visualcomponents.com/insights/blog/introducing-visual-components-4-1/. Accessed: 31-05-2018.

[118] Nikola Zlatanov. Computer security and mobile security challenges. https://www.researchgate.net/publication/298807979_Computer_Security_and_Mobile_Security_Challenges, 12 2015. Accessed: 05-06-2018.

[119] Siemens. Safety for the manufacturing industry – functional safety services. https://www.industry.siemens.com/datapool/industry/

industrysolutions/services/en/Safety-manufacturing-industry-en.pdf. Accessed: 29-11-2017.

[120] Olivier Roulet-Dubonnet. Freeopcua. https://github.com/FreeOpcUa/freeopcua. Accessed: 22-11-2017.

[121] KUKA. Kr cybertech. https://www.kuka.com/en-in/products/robotics-systems/industrial-robots/kr-cybertech. Accessed: 19-11-2017.

[122] KUKA. Kr quantec pro. https://www.kuka.com/en-de/products/robot-systems/industrial-robots/kr-quantec-pro. Accessed: 19-11-2017.

[123] KUKA. Kr c4. http://www.wtech.com.tw/public/download/manual/kuka/krc4/KUKA%20KR%20C4%20Operating%20Instructions.pdf. Accessed: 19-11-2017.

[124] OPC Foundation. Opc ua. https://opcfoundation.org/wp-content/uploads/2014/05/OPC-UA_Overview_EN.pdf. Accessed: 03-12-2017.

Appendix A

Source Code

A.1 OPC UA Server, *opcua-server02.py*

```
'''
Source code for an OPC UA server able to extract axis variables
from a KUKA KR C4 robot controller and send them to an
OPC UA client.

Based on work by:
- Massimiliano Fago (https://sourceforge.net/projects/openshowvar)
- Mechatronics Lab at AAlesund University College
  (https://github.com/aauc-mechlab/JOpenShowVar)
- Ahmad Saeed (https://github.com/akselov/kukavarproxy-msg-format)
- Olivier Roulet-Dubonnet (https://github.com/FreeOpcUa)

Author: Aksel Oevern
NTNU 2018
'''

import sys
import socket
sys.path.insert(0, "..")
import time
from opcua import ua, Server

class KUKA(object):
    # Open socket
    # KUKAVARPROXY actively listens on TCP port 7000
    def __init__(self, TCP_IP):
        try:
            client.connect((TCP_IP, 7000))
        except:
            self.error_list(1)

    # Sending messages on the KUKAVARPROXY message format:
    # Msg ID in HEX                       2 bytes
    # Msg length in HEX                   2 bytes
    # Read (0) or write (1)               1 byte
    # Variable name length in HEX         2 bytes
    # Variable name in ASCII              # bytes
    # Variable value length in HEX        2 bytes
    # Variable value in ASCII             # bytes
    def send (self, var, val, msgID):
        try:
            msg = bytearray()
            temp = bytearray()
            if val != "":
                val = str(val)
                msg.append((len(val) & 0xff00) >> 8)
                msg.append((len(val) & 0x00ff))
                msg.extend(map(ord, val))
            temp.append(bool(val))
            temp.append(((len(var)) & 0xff00) >> 8)
            temp.append((len(var)) & 0x00ff)
            temp.extend(map(ord, var))
            msg = temp + msg
            del temp[:]
            temp.append((msgID & 0xff00) >> 8)
            temp.append(msgID & 0x00ff)
            temp.append((len(msg) & 0xff00) >> 8)
            temp.append((len(msg) & 0x00ff))
            msg = temp + msg
        except :
            self.error_list(2)
        try:
            client.send(msg)
            return  client.recv(1024)
        except :
            self.error_list(1)

    # Get variables from KUKAVARPROXY response format
    def __get_var(self, msg):
        try:
            lsb = int( msg[5])
            msb = int( msg[6])
            lenValue = (lsb <<8 | msb)
            return str(msg [7: 7+lenValue],'utf-8')
        except:
            self.error_list(2)

    # Read variables from KUKAVARPROXY
    def read (self, var, msgID=0):
        try:
            return self.__get_var(self.send(var,"",msgID))
        except :
            self.error_list(2)

    # Write variables to KUKAVARPROXY
    def write (self, var, val, msgID=0):
        try:
            if val != (""):
                return self.__get_var(self.send(var,val,msgID))
            else:
                raise self.error_list(3)
        except :
            self.error_list(2)

    # Close socket
    def disconnect (self):
            client.close()

    # In case of error
    def error_list (self, ID):
        if ID == 1:
            print ("Network Error (tcp_error)")
            print ("Check your KRC's IP address on the network,
                    and make sure kukavarproxy is running.")
            self.disconnect()
            raise SystemExit
        elif ID == 2:
            print ("Python Error.")
            print ("Check the code and uncomment the lines related to
                    your python version.")
            self.disconnect()
            raise SystemExit
        elif ID == 3:
            print ("Error in write() statement.")
            print ("Variable value is not defined.")

if __name__ == "__main__":
    # Initialize OPC UA Client connection
    client = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

    # Setup robot object
    robot = KUKA('192.168.250.16')

    # Setup OPC UA Server
    server = Server()
    server.set_endpoint("opc.tcp://192.168.250.200:50895")

    # Setup namespace
    uri = "http://examples.freeopcua.github.io"
    idx = server.register_namespace(uri)

    # Get objects node
    objects = server.get_objects_node()

    # Populating address space
    myObject = objects.add_object(idx, "KUKA_KR16_2")

    # Adding desired variables
    A1 = myObject.add_variable(idx, "A1", 6.7)
    A2 = myObject.add_variable(idx, "A2", 6.7)
    A3 = myObject.add_variable(idx, "A3", 6.7)
    A4 = myObject.add_variable(idx, "A4", 6.7)
    A5 = myObject.add_variable(idx, "A5", 6.7)
    A6 = myObject.add_variable(idx, "A6", 6.7)

    A1.set_writable()
    A2.set_writable()
    A3.set_writable()
    A4.set_writable()
    A5.set_writable()
    A6.set_writable()

    # Starting OPC UA Server
    server.start()

    try:
        while(True):
            time.sleep(0.1)

            output = robot.read("$AXIS_ACT")
            output = output.replace(",", "")
            output_elements = output.split(" ")

            A1.set_value(output_elements[2])
            A2.set_value(output_elements[4])
            A3.set_value(output_elements[6])
            A4.set_value(output_elements[8])
            A5.set_value(output_elements[10])
            A6.set_value(output_elements[12])

    finally:
        # Close OPC UA client connection, remove subcsriptions, etc
        server.stop()
```

Listing A.1: opcua-server02.py

A.2 OPC UA Server GUI, *opcua-server-gui.py*

```python
'''
Source code for a GUI able to execute different OPC UA servers
with a variety of functionalities for a KUKA robot:
    - Send axis parameters to Visual Components 4.0
    - Display sensor data of motors controlling each axis
      (real-time plotting):
        # Motor velocity
        # Motor torque
        # Motor current
        # Motor temperature
    - Write sensor data to .csv files

Author: Aksel Oevern
NTNU 2018
'''

import tkinter as tk
from tkinter import ttk
from tkinter import messagebox
from tkinter import *
LARGE_FONT= ("Verdana", 12)
import sys
import subprocess

# Setup KUKA Monitiong Application GUI
class KUKA_APP(tk.Tk):

    def __init__(self, *args, **kwargs):
        tk.Tk.__init__(self, *args, **kwargs)
        tk.Tk.iconbitmap(self, "robot-16.ico")
        tk.Tk.wm_title(self, "KUKA KR 16-2 Monitoring Application")
        tk.Tk.geometry(self,'400x260')

        container = tk.Frame(self)
        container.pack(side="top", fill="both", expand = True)
        container.grid_rowconfigure(0, weight=1)
        container.grid_columnconfigure(0, weight=1)

        canvas = Canvas(self, width=2000, height=2000)
        canvas.place(x = -100, y = 220)
        blackLine = canvas.create_line(0,2,2000,2)
        canvas.configure(background='gray')

        canvas_id = canvas.create_text(432,25)
        canvas.itemconfig(canvas_id, text = "akselov@stud.ntnu.no")
        canvas.insert(canvas_id, 12, "")

        self.frames = {}

        for F in (StartPage, PageOne, PageTwo, PageThree):
            frame = F(container, self)
            self.frames[F] = frame
            frame.grid(row=0, column=0, sticky="nsew")

        self.show_frame(StartPage)

    def show_frame(self, cont):
        frame = self.frames[cont]
        frame.tkraise()

class StartPage(tk.Frame):
    def __init__(self, parent, controller):
        tk.Frame.__init__(self,parent)
        label = tk.Label(self, text="KUKA KR 16-2 Monitoring Application",
     font=LARGE_FONT)
        label.place(x=44,y=10)

        button1 = ttk.Button(self, text="Send axis parameters to Visual
    Components 4.0",
                            command=lambda: controller.show_frame(PageOne)
    )
        button1.place(x=65,y=50)

        button2 = ttk.Button(self, text="Display sensor data",
                            command=lambda: controller.show_frame(
    PageThree))
        button2.place(x=65,y=80)

        button3 = ttk.Button(self, text="Write data to file",
                            command=lambda: controller.show_frame(PageTwo)
    )
        button3.place(x=65,y=110)

        close_button = ttk.Button(self, text="Close",command=self.quit)

        close_button.place(x=65,y=140)

class PageOne(tk.Frame):
    def __init__(self, parent, controller):
        tk.Frame.__init__(self, parent)
        label = tk.Label(self, text="Axis parameters to Visual Components
    4.0", font=LARGE_FONT)
```

```
        label.place(x=20,y=10)

        button1 = ttk.Button(self, text="Back to Home",
                            command=lambda: controller.show_frame(
    StartPage))
        button1.place(x=65,y=50)

        button2 = ttk.Button(self, text="Start connection",
                            command= send_axis_parameters)
        button2.place(x=65,y=80)

class PageTwo(tk.Frame):
    def __init__(self, parent, controller):
        tk.Frame.__init__(self, parent)
        label = tk.Label(self, text="Write data to file", font=LARGE_FONT)
        label.place(x=120,y=10)

        button1 = ttk.Button(self, text="Back to home",
                            command=lambda: controller.show_frame(
    StartPage))
        button1.place(x=65,y=50)

        button2 = ttk.Button(self, text="Write Motor Speed to file",
                            command = write_motorSpeed_to_file)
        button2.place(x=65,y=80)

        button3 = ttk.Button(self, text="Write Motor Torque to file",
                            command = write_motorTorque_to_file)
        button3.place(x=65,y=110)

        button4 = ttk.Button(self, text="Write Motor Current to file",
                            command = write_motorCurrent_to_file)
        button4.place(x=65,y=140)

        button5 = ttk.Button(self, text="Write Motor Temperature to file",
                            command = write_motorTemp_to_file)
        button5.place(x=65,y=170)

class PageThree(tk.Frame):
    def __init__(self, parent, controller):
        tk.Frame.__init__(self, parent)
        label = tk.Label(self, text="Display sensor data", font=LARGE_FONT
    )
        label.place(x=110,y=10)

        button0 = ttk.Button(self, text="Back to Home",
                            command=lambda: controller.show_frame(
    StartPage))
        button0.place(x=65,y=50)

        button1 = ttk.Button(self, text="Display Motor Speed monitor",
                             command= motor_speed_monitor)
        button1.place(x=65,y=80)

        button2 = ttk.Button(self, text="Display Motor Torque monitor",
                             command= motor_torque_monitor)
        button2.place(x=65,y=110)

        button3 = ttk.Button(self, text="Display Motor Current monitor",
                             command= motor_current_monitor)
        button3.place(x=65,y=140)

        button4 = ttk.Button(self, text="Display Motor Temperature monitor
    ",
                             command= motor_temp_monitor)
        button4.place(x=65,y=170)

def send_axis_parameters():
    messagebox.showinfo("Monitor axis parameters", "Axis parameters will
    now be available in Visual Components 4.0")
    subprocess.call([sys.executable, 'server_gui_VC4_0.py', 'argument1', '
    argument2'])

def motor_speed_monitor():
    messagebox.showinfo("Display sensor data", "Motor Speed data will be
    displayed in a new window")
    subprocess.call([sys.executable, 'server_gui_motorSpeed.py', '
    argument1', 'argument2'])

def motor_torque_monitor():
    messagebox.showinfo("Display sensor data", "Motor Torque data will be
    displayed in a new window")
    subprocess.call([sys.executable, 'server_gui_motorTorque.py', '
    argument1', 'argument2'])

def motor_current_monitor():
    messagebox.showinfo("Display sensor data", "Motor Current data will be
     displayed in a new window")
    subprocess.call([sys.executable, 'server_gui_motorCurrent.py', '
    argument1', 'argument2'])

def motor_temp_monitor():
    messagebox.showinfo("Display sensor data", "Motor Temperature data
    will be displayed in a new window")
```

```python
    subprocess.call([sys.executable, 'server_gui_motorTemperature.py', 'argument1', 'argument2'])

def write_motorSpeed_to_file():
    messagebox.showinfo("Data to file", "Motor Speed data will be written to file KUKA_KR16_2_motorSpeed.csv")
    subprocess.call([sys.executable, 'server_gui_write_motorSpeed.py', 'argument1', 'argument2'])
    messagebox.showinfo("Data to file", "Motor Speed data was successfully written to file KUKA_KR16_2_motorSpeed.csv")

def write_motorTorque_to_file():
    messagebox.showinfo("Data to file", "Motor Torque data will be written to file KUKA_KR16_2_motorTorque.csv")
    subprocess.call([sys.executable, 'server_gui_write_motorTorque.py', 'argument1', 'argument2'])
    messagebox.showinfo("Data to file", "Motor Torque data was successfully written to file KUKA_KR16_2_motorTorque.csv")

def write_motorCurrent_to_file():
    messagebox.showinfo("Data to file", "Motor Current data will be written to file KUKA_KR16_2_motorCurrent.csv")
    subprocess.call([sys.executable, 'server_gui_write_motorCurrent.py', 'argument1', 'argument2'])
    messagebox.showinfo("Data to file", "Motor Current data was successfully written to file KUKA_KR16_2_motorCurrent.csv")

def write_motorTemp_to_file():
    messagebox.showinfo("Data to file", "Motor Temperature data will be written to file KUKA_KR16_2_motorTemp.csv")
    subprocess.call([sys.executable, 'server_gui_write_motorTemperature.py', 'argument1', 'argument2'])
    messagebox.showinfo("Data to file", "Motor Temperature data was successfully written to file KUKA_KR16_2_motorTemp.csv")

app = KUKA_APP()
app.mainloop()
```

Listing A.2: opcua-server-gui.py

A.3 XML-file, *KUKA_KR16_2_object.xml*

```
<!--
Source code for a XML file including information about
variables relevant for a KUKA KR 16-2 robot.

The file is made with use of the FreeOpcUa modeler made by
Olivier Roulet-Dubonnet, found at:
https://github.com/FreeOpcUa/opcua-modeler

Author: Aksel Oevern
NTNU 2018
-->

<?xml version='1.0' encoding='utf-8'?>
<UANodeSet xmlns="http://opcfoundation.org/UA/2011/03/UANodeSet.xsd"
    xmlns:uax="http://opcfoundation.org/UA/2008/02/Types.xsd" xmlns:xsd="
    http://www.w3.org/2001/XMLSchema" xmlns:xsi="http://www.w3.org/2001/
    XMLSchema-instance">
  <NamespaceUris />
  <Aliases>
    <Alias Alias="Boolean">i=1</Alias>
    <Alias Alias="Double">i=11</Alias>
    <Alias Alias="String">i=12</Alias>
    <Alias Alias="Organizes">i=35</Alias>
    <Alias Alias="HasTypeDefinition">i=40</Alias>
    <Alias Alias="HasProperty">i=46</Alias>
    <Alias Alias="HasComponent">i=47</Alias>
  </Aliases>
  <UAObject BrowseName="0:KUKA_KR16_2" NodeId="i=20001" ParentNodeId="i=85
    ">
    <DisplayName>KUKA_KR16_2</DisplayName>
    <Description>The base type for all object nodes.</Description>
    <References>
      <Reference IsForward="false" ReferenceType="Organizes">i=85</
    Reference>
      <Reference ReferenceType="HasTypeDefinition">i=58</Reference>
      <Reference ReferenceType="HasProperty">i=20006</Reference>
      <Reference ReferenceType="HasProperty">i=20007</Reference>
      <Reference ReferenceType="HasComponent">i=20008</Reference>
      <Reference ReferenceType="HasComponent">i=20009</Reference>
      <Reference ReferenceType="HasComponent">i=20010</Reference>
      <Reference ReferenceType="HasComponent">i=20011</Reference>
      <Reference ReferenceType="HasComponent">i=20012</Reference>
      <Reference ReferenceType="HasComponent">i=20013</Reference>
      <Reference ReferenceType="HasProperty">i=20016</Reference>
    </References>
```

```xml
</UAObject>
<UAVariable BrowseName="0:KUKAVARPROXY" DataType="Boolean" NodeId="i=20006" ParentNodeId="i=20001">
  <DisplayName>KUKAVARPROXY</DisplayName>
  <Description>KUKAVARPROXY</Description>
  <References>
    <Reference IsForward="false" ReferenceType="HasProperty">i=20001</Reference>
    <Reference ReferenceType="HasTypeDefinition">i=68</Reference>
  </References>
  <Value>
    <uax:Boolean>false</uax:Boolean>
  </Value>
</UAVariable>
<UAVariable BrowseName="0:RSI" DataType="Boolean" NodeId="i=20007" ParentNodeId="i=20001">
  <DisplayName>RSI</DisplayName>
  <Description>RSI</Description>
  <References>
    <Reference IsForward="false" ReferenceType="HasProperty">i=20001</Reference>
    <Reference ReferenceType="HasTypeDefinition">i=68</Reference>
  </References>
  <Value>
    <uax:Boolean>false</uax:Boolean>
  </Value>
</UAVariable>
<UAVariable BrowseName="0:A1" DataType="Double" NodeId="i=20008" ParentNodeId="i=20001">
  <DisplayName>A1</DisplayName>
  <Description>A1</Description>
  <References>
    <Reference IsForward="false" ReferenceType="HasComponent">i=20001</Reference>
    <Reference ReferenceType="HasTypeDefinition">i=63</Reference>
  </References>
  <Value>
    <uax:Double>9.99</uax:Double>
  </Value>
</UAVariable>
<UAVariable BrowseName="0:A2" DataType="Double" NodeId="i=20009" ParentNodeId="i=20001">
  <DisplayName>A2</DisplayName>
  <Description>A2</Description>
  <References>
    <Reference IsForward="false" ReferenceType="HasComponent">i=20001</Reference>
```

```
      <Reference ReferenceType="HasTypeDefinition">i=63</Reference>
    </References>
    <Value>
      <uax:Double>9.99</uax:Double>
    </Value>
  </UAVariable>
  <UAVariable BrowseName="0:A3" DataType="Double" NodeId="i=20010" ParentNodeId="i=20001">
    <DisplayName>A3</DisplayName>
    <Description>A3</Description>
    <References>
      <Reference IsForward="false" ReferenceType="HasComponent">i=20001</Reference>
      <Reference ReferenceType="HasTypeDefinition">i=63</Reference>
    </References>
    <Value>
      <uax:Double>9.99</uax:Double>
    </Value>
  </UAVariable>
  <UAVariable BrowseName="0:A4" DataType="Double" NodeId="i=20011" ParentNodeId="i=20001">
    <DisplayName>A4</DisplayName>
    <Description>A4</Description>
    <References>
      <Reference IsForward="false" ReferenceType="HasComponent">i=20001</Reference>
      <Reference ReferenceType="HasTypeDefinition">i=63</Reference>
    </References>
    <Value>
      <uax:Double>9.99</uax:Double>
    </Value>
  </UAVariable>
  <UAVariable BrowseName="0:A5" DataType="Double" NodeId="i=20012" ParentNodeId="i=20001">
    <DisplayName>A5</DisplayName>
    <Description>A5</Description>
    <References>
      <Reference IsForward="false" ReferenceType="HasComponent">i=20001</Reference>
      <Reference ReferenceType="HasTypeDefinition">i=63</Reference>
    </References>
    <Value>
      <uax:Double>9.99</uax:Double>
    </Value>
  </UAVariable>
  <UAVariable BrowseName="0:A6" DataType="Double" NodeId="i=20013" ParentNodeId="i=20001">
```

```xml
    <DisplayName>A6</DisplayName>
    <Description>A6</Description>
    <References>
      <Reference IsForward="false" ReferenceType="HasComponent">i=20001</
    Reference>
      <Reference ReferenceType="HasTypeDefinition">i=63</Reference>
    </References>
    <Value>
      <uax:Double>9.99</uax:Double>
    </Value>
  </UAVariable>
  <UAVariable BrowseName="0:IP" DataType="String" NodeId="i=20016"
    ParentNodeId="i=20001">
    <DisplayName>IP</DisplayName>
    <Description>IP</Description>
    <References>
      <Reference IsForward="false" ReferenceType="HasProperty">i=20001</
    Reference>
      <Reference ReferenceType="HasTypeDefinition">i=68</Reference>
    </References>
    <Value>
      <uax:String>192.168.250.16</uax:String>
    </Value>
  </UAVariable>
</UANodeSet>
```

Listing A.3: KUKA_KR16_2_object.xml

A.4 OPC UA Server XML, *opcua-server-xml.py*

```
'''
Source code for an OPC UA server able to create a custom OPC UA
object from specifications in a XML-file, estract axis variables
from a KUKA KR C4 robot controller and send them to an OPC UA client.

Based on work by:
- Massimiliano Fago (https://sourceforge.net/projects/openshowvar)
- Mechatronics Lab at AAlesund University College
  (https://github.com/aauc-mechlab/JOpenShowVar)
- Ahmad Saeed (https://github.com/akselov/kukavarproxy-msg-format)
- Olivier Roulet-Dubonnet (https://github.com/FreeOpcUa)

Author: Aksel Oevern
NTNU 2018
'''

import sys
sys.path.insert(0, "..")
import time
import KUKAVARPROXY_communication
from opcua import ua, Server

if __name__ == "__main__":
    # Setup OPC-UA Server
    server = Server()
    server.set_endpoint("opc.tcp://192.168.250.200:50895")

    # Setup namespace
    uri = "http://examples.freeopcua.github.io"
    idx = server.register_namespace(uri)

    # Get objects node
    objects = server.get_objects_node()

    # Import customobject type
    server.import_xml('KUKA_KR16_2_object.xml')

    # Check if client wants communication over KUKAVARPROXY or RSI
    kuka_object = objects.get_child("0:KUKA_KR16_2")
    ip_adress = kuka_object.get_child("0:IP")
    variable_kukavarproxy = kuka_object.get_child("0:KUKAVARPROXY")
    variable_rsi = kuka_object.get_child("0:RSI")

    # Extracting axis variables and removing KUKAVARPROXU/RSI
    configuration variables
```

```python
    variables_all = kuka_object.get_children()

    for variable in variables_all:
        variable.set_writable()

    # Removing configuration properties
    variables_all.remove(ip_adress)
    variables_all.remove(variable_kukavarproxy)
    variables_all.remove(variable_rsi)
    variables_axis = variables_all

    # Starting OPC—UA Server
    server.start()

    try:
        print("Select KUKAVARPROXY or RSI communication configuration in
OPC UA Client")
        while True:
            time.sleep(0.1)

            # Check for client preference KUKAVARPROXY/RSI communication
            if(variable_kukavarproxy.get_value() and variable_rsi.
get_value()):
                raise ValueError("KUKAVARPROXY and RSI can not both be
selected")
                SystemExit

            if(variable_kukavarproxy.get_value()):
                print("Launching KUKAVARPROXY communication")
                KUKAVARPROXY_communication.run(variables_axis, ip_adress)
                break

            if(variable_rsi.get_value()):
                print("Launching RSI communication")
                #Run RSI communication module

    finally:
        # Close OPC UA client connection, remove subcsriptions, etc
        server.stop()
```
Listing A.4: opcua-server-xml.py

A.5 Module, *KUKAVARPROXY_communication.py*

```python
'''
Source code for a KUKAVARPROXY communication module being used
by the opcua-server-xml.py for communication with a KUKA KR C4
robot controller. The module extracts axis variables from the
robot controller.

- Massimiliano Fago (https://sourceforge.net/projects/openshowvar)
- Mechatronics Lab at AAlesund University College
  (https://github.com/aauc-mechlab/JOpenShowVar)
- Ahmad Saeed (https://github.com/akselov/kukavarproxy-msg-format)

Author: Aksel Oevern
NTNU 2018
'''

import socket
import time

# Initialize OPC-UA Client connection
client = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

class KUKA(object):

    # Open socket
    # KukaVarProxy actively listens on TCP port 7000
    def __init__(self, TCP_IP):
        try:
            client.connect((TCP_IP, 7000))
        except:
            self.error_list(1)

    # Sending messages on the KukaVarProxy message format:
    # Msg ID in HEX                       2 bytes
    # Msg length in HEX                   2 bytes
    # Read (0) or write (1)               1 byte
    # Variable name length in HEX         2 bytes
    # Variable name in ASCII              # bytes
    # Variable value length in HEX        2 bytes
    # Variable value in ASCII             # bytes
    def send (self, var, val, msgID):
        try:
            msg = bytearray()
            temp = bytearray()
            if val != "":
                val = str(val)
                msg.append((len(val) & 0xff00) >> 8)
                msg.append((len(val) & 0x00ff))
                msg.extend(map(ord, val))
            temp.append(bool(val))
            temp.append(((len(var)) & 0xff00) >> 8)
            temp.append((len(var)) & 0x00ff)
            temp.extend(map(ord, var))
            msg = temp + msg
            del temp[:]
            temp.append((msgID & 0xff00) >> 8)
            temp.append(msgID & 0x00ff)
            temp.append((len(msg) & 0xff00) >> 8)
            temp.append((len(msg) & 0x00ff))
            msg = temp + msg
        except :
            self.error_list(2)
        try:
            client.send(msg)
            return  client.recv(1024)
        except :
            self.error_list(1)

    # Get variables from KukaVarProxy response format
    def __get_var(self, msg):
        try:
            lsb = int( msg[5])
            msb = int( msg[6])
            lenValue = (lsb <<8 | msb)
            return str(msg [7: 7+lenValue],'utf-8')
        except:
            self.error_list(2)

    # Read variables from KukaVarProxy
    def read (self, var, msgID=0):
        try:
            return self.__get_var(self.send(var,"",msgID))
        except :
            self.error_list(2)

    # Write variables to KukaVarProxy
    def write (self, var, val, msgID=0):
        try:
            if val != (""): return self.__get_var(self.send(var,val,msgID))
            else: raise self.error_list(3)
        except :
            self.error_list(2)

    # Close socket
    def disconnect (self):
            client.close()

    # In case of error
    def error_list (self, ID):
        if ID == 1:
            print ("Network Error (tcp_error)")
            print ("Check your KRC's IP address on the network, and make sure kukaproxyvar is running.")
            self.disconnect()
            raise SystemExit
        elif ID == 2:
            print ("Python Error.")
            print ("Check the code and uncomment the lines related to your python version.")
            self.disconnect()
            raise SystemExit
        elif ID == 3:
            print ("Error in write() statement.")
            print ("Variable value is not defined.")

def run(variables_axis, ip_adress):
    # Setup robot object
    robot = KUKA(ip_adress.get_value())
    print("KUKAVARPROXY communication running successfully \nObject's IP adress:", ip_adress.get_value())

    while True:
      time.sleep(0.01)

      output = robot.read("$AXIS_ACT")
      output = output.replace(",", "")
      output_elements = output.split(" ")

      i = 0
      for axis in variables_axis:
        axis.set_value(output_elements[2+i])
        i+=2
```

Listing A.5: KUKAVARPROXY_communication.py

A.6 RSI OPC UA Server, *rsi-opcua-server.py*

```python
'''
Source code for an OPC UA server able to extract axis variables
from a KUKA KR C4 robot controller through RSI communication
and send them to an OPC UA client.

Based on work by:
- Olivier Roulet-Dubonnet (https://github.com/FreeOpcUa)
- Eren Sezener (https://github.com/akselov/kuka-rsi3-communicator)
- Torstein Anderssen Myhre (https://github.com/torstem/examplecode-kukarsi
    -python)

Author: Aksel Oevern
NTNU 2018
'''

import socket
import numpy
import command
import time
import time_keeper
from opcua import ua, Server

tiden = time_keeper.time_keeper()

def simple_joint_correction_command(from_kuka):
    #Correction example (not used)
    A1 = 10.0 * numpy.sin(tiden.get() / 10.0)
    joint_desired = [0.0, 0.0, 0.0, 0.0, 0.0, 0.0]
    return command.joint_correction_command(from_kuka, joint_desired)

if __name__ == "__main__":
    # Initialize OPC-UA Client connection
    client = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

    # Setup OPC-UA Server
    server = Server()
    server.set_endpoint("opc.tcp://192.168.250.119:50895")

    # Setup namespace
    uri = "http://examples.freeopcua.github.io"
    idx = server.register_namespace(uri)

    # Get objects node
    objects = server.get_objects_node()

```

```python
# Populating address space
myObject = objects.add_object(idx, "KUKA_KR16_2")

# Adding desired variables
A1 = myObject.add_variable(idx, "A1", 6.7)
A2 = myObject.add_variable(idx, "A2", 6.7)
A3 = myObject.add_variable(idx, "A3", 6.7)
A4 = myObject.add_variable(idx, "A4", 6.7)
A5 = myObject.add_variable(idx, "A5", 6.7)
A6 = myObject.add_variable(idx, "A6", 6.7)

A1.set_writable()
A2.set_writable()
A3.set_writable()
A4.set_writable()
A5.set_writable()
A6.set_writable()

# Starting OPC-UA Server
server.start()

# Setup RSI connection
BUFFER_SIZE = 1024
sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
sock.bind(("192.168.1.198", 49001))
counter = 0

try:
    while True:
        # Buffer size is 1024 bytes

        #time.sleep(0.1)
        received_data, socket_of_krc = sock.recvfrom(BUFFER_SIZE)
        received_data = received_data.decode("utf-8")
        output_elements = received_data.split(' ')
        raw_axis_variables = output_elements[14:20]
        axis_variables = []

        for raw_axis in raw_axis_variables:
            axis_variables.append(raw_axis.split('"')[1])

        print(axis_variables)

        A1.set_value(axis_variables[0])
        A2.set_value(axis_variables[1])
        A3.set_value(axis_variables[2])
        A4.set_value(axis_variables[3])
```

```
            A5.set_value(axis_variables[4])
            A6.set_value(axis_variables[5])

            kuka_command = simple_joint_correction_command(received_data)
            reply = bytes(kuka_command, 'utf-8')
            sock.sendto(reply, socket_of_krc)

    finally:
        # Close OPC UA client connection, remove subcsriptions, etc
        server.stop()
```
Listing A.6: rsi-opcua-server.py

A.7 KVP Control Server, *opcua-control-server.py*

```
'''
Source code for an OPC UA server able to create a custom OPC UA
object from specifications in a XML-file, extract axis variables
from an OPC UA client and control a KUKA robot by its axis values.
Based on work by:

- Massimiliano Fago (https://sourceforge.net/projects/openshowvar)
- Mechatronics Lab at AAlesund University College
  (https://github.com/aauc-mechlab/JOpenShowVar)
- Ahmad Saeed (https://github.com/akselov/kukavarproxy-msg-format)
- Olivier Roulet-Dubonnet (https://github.com/FreeOpcUa)

To be done:
- Custom setup of axis variables from OPC UA object
- Include RSI control module

Author: Aksel Oevern
NTNU 2018
'''

import sys
import socket
sys.path.insert(0, "..")
import time
from xml.dom import minidom
from opcua import ua, Server

class KUKA(object):
    # Open socket
    # KukaVarProxy actively listens on TCP port 7000
    def __init__(self, TCP_IP):
        try:
            client.connect((TCP_IP, 7000))
        except:
            self.error_list(1)

    # Sending messages on the KukaVarProxy message format:
    # Msg ID in HEX                       2 bytes
    # Msg length in HEX                   2 bytes
    # Read (0) or write (1)               1 byte
    # Variable name length in HEX         2 bytes
    # Variable name in ASCII              # bytes
    # Variable value length in HEX        2 bytes
    # Variable value in ASCII             # bytes
    def send (self, var, val, msgID):
        try:
            msg = bytearray()
            temp = bytearray()
            if val != "":
                val = str(val)
                msg.append((len(val) & 0xff00) >> 8)
                msg.append((len(val) & 0x00ff))
                msg.extend(map(ord, val))
            temp.append(bool(val))
            temp.append(((len(var)) & 0xff00) >> 8)
            temp.append((len(var)) & 0x00ff)
            temp.extend(map(ord, var))
            msg = temp + msg
            del temp[:]
            temp.append((msgID & 0xff00) >> 8)
            temp.append(msgID & 0x00ff)
            temp.append((len(msg) & 0xff00) >> 8)
            temp.append((len(msg) & 0x00ff))
            msg = temp + msg
        except :
            self.error_list(2)
        try:
            client.send(msg)
            return  client.recv(1024)
        except :
            self.error_list(1)

    # Get variables from KukaVarProxy response format
    def __get_var(self, msg):
        try:
            lsb = int( msg[5])
            msb = int( msg[6])
            lenValue = (lsb <<8 | msb)
            return str(msg [7: 7+lenValue],'utf-8')
        except:
            self.error_list(2)

    # Read variables from KukaVarProxy
    def read (self, var, msgID=0):
        try:
            return self.__get_var(self.send(var,"",msgID))
        except :
            self.error_list(2)

    # Write variables to KukaVarProxy
    def write (self, var, val, msgID=0):
        try:
            if val != (""):
                return self.__get_var(self.send(var,val,msgID))
            else:
                raise self.error_list(3)
        except :
            self.error_list(2)

    # Close socket
    def disconnect (self):
            client.close()

    # In case of error
    def error_list (self, ID):
        if ID == 1:
            print ("Network Error (tcp_error)")
            print ("Check your KRC's IP address on the network, and make sure kukaproxyvar is running.")
            self.disconnect()
            raise SystemExit
        elif ID == 2:
            print ("Python Error.")
            print ("Check the code and uncomment the lines related to your python version.")
            self.disconnect()
            raise SystemExit
        elif ID == 3:
            print ("Error in write() statement.")
            print ("Variable value is not defined.")


if __name__ == "__main__":
    # Initialize OPC-UA Client connection
    client = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

    # Setup robot object
    robot = KUKA('192.168.250.16')

    # Setup OPC-UA Server
    server = Server()
    server.set_endpoint("opc.tcp://192.168.250.200:50895")

    # Setup namespace
    uri = "http://examples.freeopcua.github.io"
    idx = server.register_namespace(uri)

    # Get objects node
    objects = server.get_objects_node()
```

```python

    # Import customobject type
    server.import_xml('KUKA_KR16_2_object.xml')

    # Check if client wants communication over KUKAVARPROXY or RSI
    kuka_object = objects.get_child("0:KUKA_KR16_2")
    variable_kukavarproxy = kuka_object.get_child("0:KUKAVARPROXY")
    variable_rsi = kuka_object.get_child("0:RSI")

    # Manual setup of axis variables for OPC UA object
    A1_var = kuka_object.get_child("0:A1")
    A2_var = kuka_object.get_child("0:A2")
    A3_var = kuka_object.get_child("0:A3")
    A4_var = kuka_object.get_child("0:A4")
    A5_var = kuka_object.get_child("0:A5")
    A6_var = kuka_object.get_child("0:A6")

    # Make KVP and RSI configuration writable by OPC UA client
    variable_kukavarproxy.set_writable()
    variable_rsi.set_writable()

    # Extracting axis variables and removing KUKAVARPROXU/RSI configuration variables
    variables_all = kuka_object.get_children()
    variables_all.remove(variable_kukavarproxy)
    variables_all.remove(variable_rsi)
    variables_axis = variables_all

    for variable_axis in variables_axis:
        variable_axis.set_writable()

    # Starting OPC—UA Server
    server.start()

    try:
        # Give user information about communication that is being used
        print("Kuka: ", variable_kukavarproxy.get_value())
        print("RSI: ", variable_rsi.get_value())
        print("Select KUKAVARPROXY or RSI control configuration in OPC UA Client")

        while True:
            time.sleep(0.1)

            # Check for client preference KUKAVARPROXY/RSI control
            if(variable_kukavarproxy.get_value() and variable_rsi.get_value()):
```

```
            raise ValueError("KUKAVARPROXY and RSI can not both be selected")
            SystemExit

        if(variable_kukavarproxy.get_value()):
            print("Launching KUKAVARPROXY control")
            robot.write("TARGET_AXIS","{E6AXIS: A1 " + str(A1_var.get_value()) + ", A2 " + str(A2_var.get_value()) + ", A3 " + str(A3_var.get_value()) + ", A4 " + str(A4_var.get_value()) + ", A5 " + str(A5_var.get_value()) + ", A6 " + str(A6_var.get_value()) + "}")

        if(variable_rsi.get_value()):
            print("Launching RSI control")
           #Run RSI control module

finally:
    # Close OPC UA client connection, remove subcsriptions, etc
    server.stop()
```

Listing A.7: opcua-control-server.py

Appendix B

Digital Appendix

B.1 GitHub Repository

Can be found in the attached Zip-folder, and online at
`https://github.com/akselov/digital-twin-opcua`

B.2 Video of Fixed Case

Can be found in the attached Zip-folder, and online at
`https://youtu.be/xlQhQPmJwlA`