Eskild Godli

Controlling Fronius welding machine with a Computer through Ethernet and DeviceNet.

Master's thesis in Mechanichal Engineering
Supervisor: Lars Tingelstad
June 2023

Eskild Godli

Controlling Fronius welding machine with a Computer through Ethernet and DeviceNet.

Master’s thesis in Mechanichal Engineering
Supervisor: Lars Tingelstad
June 2023

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

Preface

This master thesis represents the final output of my Master's degree in Mechanical Engineering at the Norwegian University of Science and Technology (NTNU) in Trondheim. The thesis has been prepared for the Department of Mechanical and Industrial Engineering, within the specialization of Robotics and Automation.

I would like to extend my thanks to Lars Tinglestad for his supervision of this master thesis. His insights and guidance have significantly contributed to achieving an improved solution for this Master Thesis.

My gratitude also goes towards Lars Bonvik for his contribution as a co-author on the project thesis we wrote last fall and Amund Skavhaug for being the supervisor of that thesis.

Abstract

Programming the robotic welding of complex parts through a robot controller poses notable challenges. Incorporating the robot into ROS2 streamlines the programming process, making the simulation and planning of welding paths more straightforward. This advancement demands not only the control over the robot but also of the welding apparatus. A system to manage the welding apparatuses used for robotic welding at NTNU's laboratory has been developed to cater to this requirement. The approach entails connecting the welding apparatus to a PLC through the DeviceNet protocol, which emulates the role of the robot controller, while a computer controls the PLC via the OPC UA protocol over an Ethernet connection.

Furthermore, a comprehensive, step-by-step guide for configuring and programming the solution has been provided. This guide is intended to aid in deploying the solution across multiple robotic cells within the laboratory. A set of tests has been carried out to assess the functionality of the developed solution. The results of these tests indicate that the solution is capable of effectively controlling various control variables and that this is achieved in a timely manner for the most part.

Sammendrag

Å programmere robotisert sveising av komplekse deler gjennom en robotkontroller byr på betydelige utfordringer. Integreringen av roboten i ROS2 forenkler programmeringsprosessen, noe som gjør simulering og planlegging av sveisebaner mer rett frem. Denne forbedringen krever ikke bare kontroll over roboten, men også over sveiseapparatet. For å imøtekomme dette kravet, er det utviklet et system for å håndtere sveiseapparatene som brukes til robotisert sveising ved NTNUs laboratorium. Tilnærmingen innebærer å koble sveiseapparatet til en PLS via DeviceNet-protokollen, som etterligner rollen til robotkontrolleren, mens PLS-en styres av en datamaskin via OPC UA-protokollen over en Ethernet-forbindelse.

Videre er det gitt en omfattende, trinnvis veiledning for konfigurering og programmering av løsningen. Denne veiledningen er ment å bidra til implementeringen av løsningen i flere robotceller innenfor laboratoriet. For å vurdere funksjonaliteten til den utviklede løsningen, er det utført en serie tester. Resultatene fra disse testene indikerer at løsningen er i stand til å kontrollere forskjellige styringsvariabler effektivt, og dette blir for det meste oppnådd på en tidsriktig måte.

Table of Contents

List of Figures

List of Tables

1 Introduction

1.1 Motivation

The motivation for this master thesis project is to address the challenge of welding complex parts using a robot controller. While programming the robot to perform simple welding tasks is easy, more complex parts require a more advanced solution. This is where ROS2 comes in. The problem is controlling the welding equipment through the robot controller, which is complicated and difficult to do using a Yaskawa robot controller with a PC.

This project will concentrate on the welding apparatus and its communication. It is set up in the robot cells at the Robot Laboratory of the Department of Mechanical and Industrial Engineering at the Norwegian University of Science and Technology (NTNU), where the welding apparatus is connected to a Yaskawa robot controller using the DeviceNet protocol. With this configuration, monitoring and controlling the welding parameters continuously during operation is challenging, especially from a remote location such as a PC or elsewhere.

1.2 Project Description

This project entails establishing a connection between a computer and the Fronius welding machine, which is integrated with the Yaskawa Robot controller. Further, it involves developing a system to control the welding apparatus via the computer. Figure 1 provides a visualization of the connection that requires development, accompanied by an explanatory arrow to the right, outlining potential strategies for addressing this challenge.

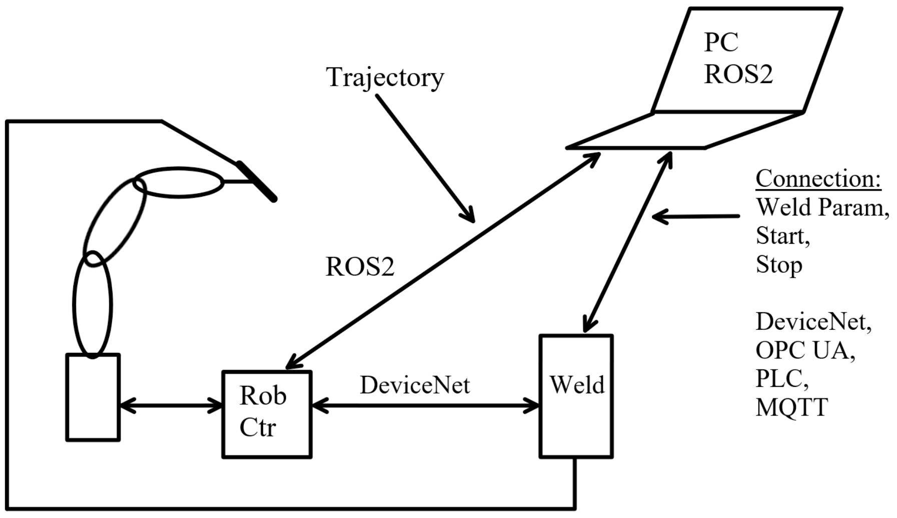


Figure 1: Concept figure for the task to be solved in this project.

Several requirements have been stipulated by the project advisor regarding potential actions and choices to be made. This is due to the fact that this project is developing a solution for the robot cells already established in the laboratory. To preserve the existing functionality, the current equipment cannot be altered, although it can be disconnected, provided it can be easily reconnected.

Given that there are multiple robot cells with the same welding apparatus and robot arm, part of the project also involves creating a guide on how the system is set up and how it can be programmed.

Several types of welding apparatus are housed within the robotic cells. However, the focus of this research has been the Fronius TPS400i. In this master thesis, a specific solution for this system has been developed due to the predominance of this particular model in the robotic cells.

1.3 Previous Work

The author of this master thesis, in their project thesis, has previously worked with Programmable Logic Controllers (PLCs). In collaboration with the co-author of that project thesis, they developed a Raspberry Pi-based PLC for educational use. Through this process and robotics courses undertaken during their studies, the author has acquired knowledge and skills relevant to this project.

1.4 Objective

To achieve a solution for the task outlined in the project description, two main objectives and goals were established:

- Establish communication between the welding apparatus and a computer.
- To control the welding machine via a computer, facilitating actions such as starting, stopping, and modifying the welding parameters in some capacity.

In order to attain the best possible solution to the problem outlined in the project description and to explore other potential opportunities, several secondary objectives were set:

- To do it with an Ethernet connection
- Research other solutions
- Test some other solutions researched
- Develop a step-by-step guide for the implementation of the solution
- Plug-and-play functionality
- Complete control over the welding parameters

1.5 Structure of Thesis

Section 2 introduces the theory and key technologies used in this thesis. It begins with presenting what a Programmable Logic Controller is and continues with how it works.

Section 3 presents the developed solution to the task presented in the Project Description.

Section 4 details a guide on how to set up the developed system.

Section 5 goes through how to program the PLC used for the solution and goes through and describes the developed code for the PLC and the test clients.

Section 6 contains the results of the tests done with the Developed solution.

Section 7 discusses the technology choices, overall results, difficulties, and errors.

Section 8 contains the conclusion and further works of the master thesis.

Appendix A a figure showing a more advanced architecture representation of a PLC from the PLC standard IEC 61131-2.

Appendix B contains the complete I/O mapping for the Fronius welding apparatus found in the Yaskawa robot controller.

Appendix C contains all the PLC code developed for the solution.

Appendix D contains all the important completed Python test clients and the credentials text file used by them.

Appendix E contain a listing of the attached files, including the risk assessment and the author's project thesis, to which the author of this master thesis contributed as a co-author.

This master thesis contains some content that has been reused or based on the project thesis, particularly in the background theory section on PLC. Since most of the project thesis is devoted to PLC, gathering information and describing it has already been done.

2 Background Theory

This chapter provides the foundational theoretical framework required for understanding the subsequent investigations and discussions within this master thesis. The knowledge presented herein serves as a primer to facilitate understanding of the technologies utilized and discussed in the subsequent sections of this Master thesis.

2.1 Programmable Logic Controller (PLC)

"A Programmable Logic Controller (PLC) is a type of device extensively used for different automation applications within industrial processes and manufacturing" [1]. As its name implies, it is a form of controller. This section will give an overview of controllers in general, as well as presenting the function, hardware, and architecture of PLCs.

2.1.1 Control Systems and Controllers

Controllers or control systems may be needed to manage sequences of events, maintain constants, or follow prescribed changes [1]. Automation streamlines tasks previously done manually, reducing costs and increasing safety in the workplace.

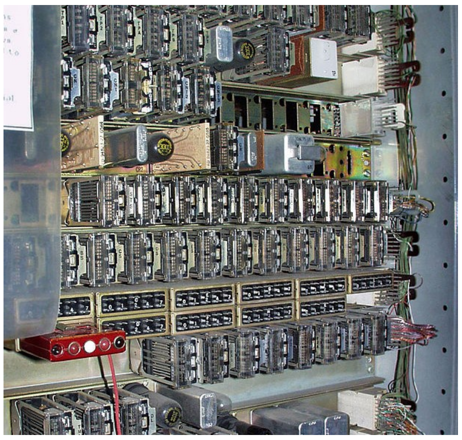


Figure 2: Example of a relay control system in the Number Five Crossbar Switching System, exhibited at the Museum of Communication in Seattle [2].

Relay systems, popular before PLCs, automated tasks requiring precision and tight time constraints. However, they struggle with complex tasks, and modifications are difficult, and they are space-consuming. An example of a relay system is shown in Figure 2 [1], [3], [4].

Microprocessor control systems offer a flexible, cost-effective, and space-efficient alternative. Programmable Logic Controllers (PLCs) standardize microcontroller systems for industrial use with robust programming languages. Ladder Diagram, a PLC language, was designed to make the transition easier for those familiar with relay systems.

Since the first PLC's development in 1969, they have evolved into modular units, allowing I/O expansion and analog I/O usage. PLC-specific programming languages necessitated international standardization, resulting in standards such as IEC 1131 and IEC 61499. Later the IEC 1131 was renamed to IEC 61131 with extensions. The IEC 61131 standard now has ten parts, with the latest released in 2019 [1], [3], [5]–[8].

2.1.2 The Functions of PLCs

A PLC is a complete system with a microprocessor and I/O designed for industrial use, using simple languages for easy programming. Figure 3 illustrates a PLC's function: receiving sensor inputs, interpreting them, and sending output signals to control devices such as motors.

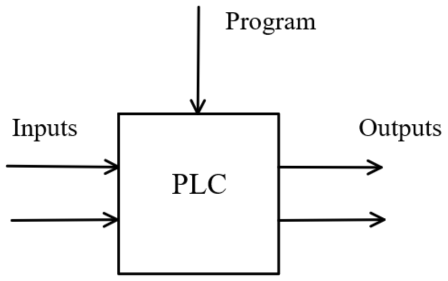


Figure 3: Simple illustration of a PLC

PLCs are designed for industrial tasks such as manufacturing, industrial processes, and automated assembly, where uninterrupted operation is crucial. Their robustness ensures minimal failure, even in harsh environments.

Some applications don't require PLC robustness, e.g., home washing machines, which can use simpler microprocessor controllers. Autopilots for airplanes require more computational power than PLCs offer, making regular computers more suitable. The next subsection will explore PLC hardware and architecture [1], [4], [7].

2.1.3 The Hardware and Architecture of PLCs

A typical PLC's functional parts are shown in Figure 4. They include the processor, program and data memory, communications interface, power supply, and input/output (I/O) interfaces. The I/O interfaces will be further presented.

Figure 5 displays the architecture diagrams from IEC 61131-1 and IEC 61131-2 standards. For in-depth specifications of PLC hardware and architecture functional requirements, refer to Chapter 6 in the IEC 61131-2 standard [1], [3], [4], [7], [9].

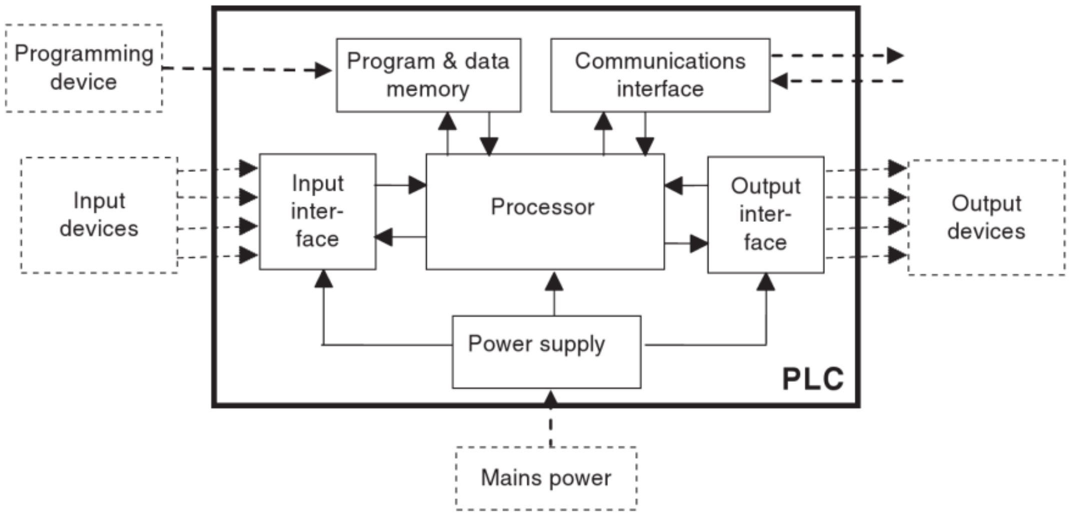
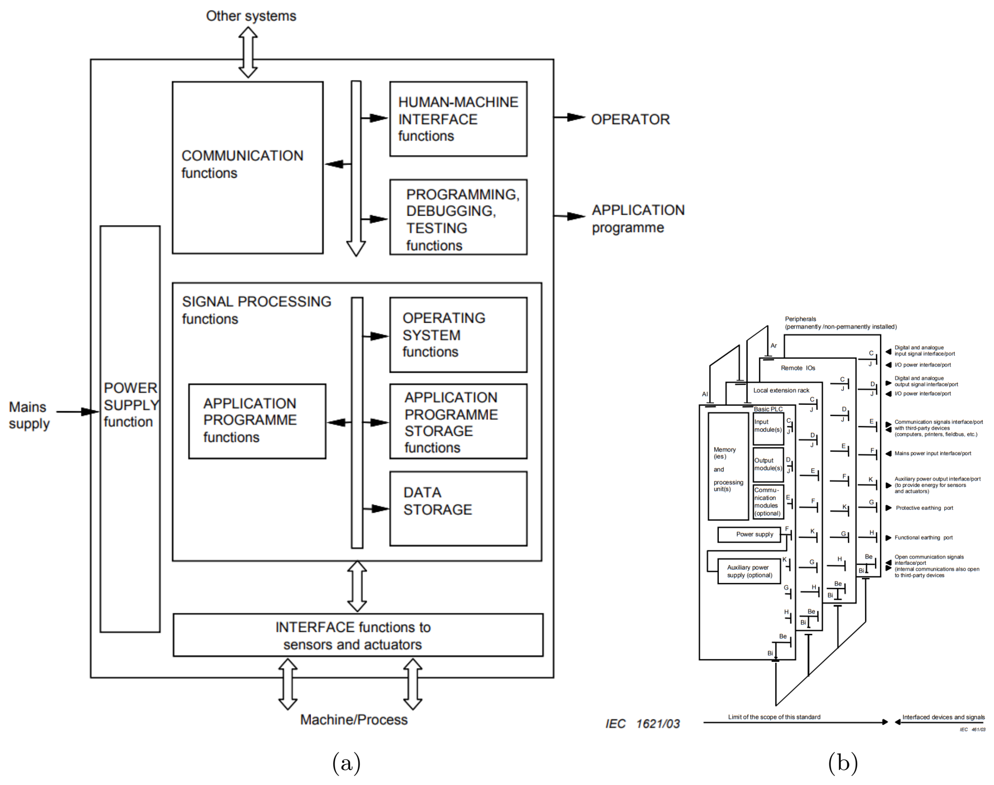


Figure 4: PLC general architecture, [1]



Other systems
COMMUNICATION functions
HUMAN-MACHINE INTERFACE functions
OPERATOR
PROGRAMMING, DEBUGGING, TESTING functions
APPLICATION programme
SIGNAL PROCESSING functions
OPERATING SYSTEM functions
APPLICATION PROGRAMME functions
APPLICATION PROGRAMME STORAGE functions
DATA STORAGE
Mains supply
POWER SUPPLY function
INTERFACE functions to sensors and actuators
Machine/Process
IEC 1621/03
(a)
Peripherals (permanently /non-permanently installed)
Remote IOs
Local extension rack
Basic PLC
Memory (ies) and processing unit(s)
Input module(s)
Output module(s)
Commu-nication modules (optional)
Power supply
Auxiliary power supply (optional)
Limit of the scope of this standard
Interfaced devices and signals
(b)

Figure 5: PLC architecture from IEC 61131: (a) simple version from IEC 61131-1 [7], and (b) extended version from IEC 61131-2 [9]. A larger version can be found in Appendix A.

2.1.4 PLC Inputs and Outputs

The input and output (I/O) interfaces in a PLC serve as the critical communication bridge between the PLC and external devices. These interfaces are responsible for transmitting output signals from the PLC to controlled devices and receiving input signals from control devices. Depending on the specific application, PLCs can have a combination of discrete/digital I/O and analog I/O, each catering to different types of input and output devices.

For instance, discrete inputs can be provided by push buttons or phototransistors, while analog inputs may come from temperature sensors or other variable-signal devices. In terms of outputs, discrete outputs might involve LEDs or single-speed motors, while analog outputs could control variable-speed motors, such as those used in CNC machines.

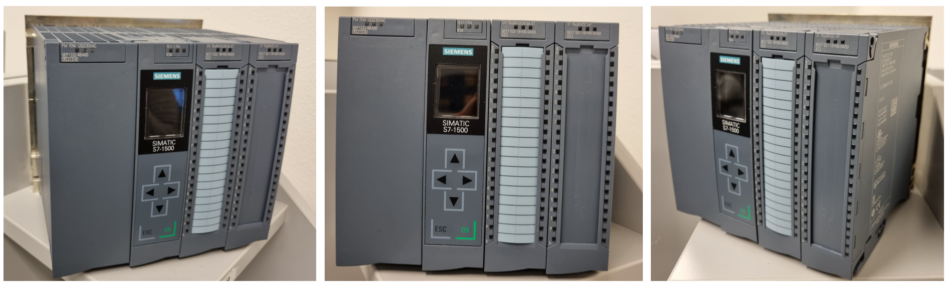
The PLC processor typically operates at low voltages, ranging from 3.3V to 5V, whereas the I/O module commonly uses 24V for discrete signals. Since the processor is not designed to directly handle these higher voltages, optoisolators are utilized for input signals to prevent damage. For output signals, the PLC employs relays, transistors, or triacs to convert the low-voltage signals from the processor to the appropriate output voltage. Relays and transistors function as switches for the correct electrical signal, while triacs are specifically designed for AC current.

To ensure PLCs function reliably and safely, the IEC 61131-2 Chapter 6 standard specifies requirements for I/O interfaces and other functions. For example, a specific output type must not have more than a certain amount of leaking current in its off state. These requirements are crucial for PLCs to provide predictable behaviour, safety, robustness, and longevity in the various environments they are used in [1], [4].

Modern PLCs have evolved to offer greater modularity and flexibility, enabling users to customize their systems by adding modules that support various communication protocols for I/O operations, or more modules with simple I/O. These communication protocols, such as Modbus, ModbusTCP, DeviceNet, and EtherCAT, allow for seamless integration and interoperability between different devices and systems in industrial automation. By utilizing these communication protocols, PLCs can efficiently exchange data with other controllers, sensors, and actuators, thereby enhancing the overall performance and capabilities of the control system. This modular approach also allows for easier system expansion and adaptation, catering to the ever-changing needs of industrial automation processes [4], [10], [11].

2.1.5 PLC Manufacturers and Models

There are numerous PLC manufacturers to choose from when considering a control system. Each manufacturer has its pros and cons and typically requires proprietary environments for coding, uploading code, and facilitating communication between units. However, open standards for communication exist in mixed environments when necessary. Modicon was the first PLC manufacturer, but Siemens is now one of the most popular PLC manufacturers. Beckhoff is another well-known manufacturer, offering a wide range of PLC products and solutions. Figure 6 shows an example of a PLC from Siemens [1], [3], [4].



(a) (b) (c)

Figure 6: Three pictures the author took of a Siemens Simatic S7-1500 PLC.

2.2 ROS2

2.2.1 Introduction to ROS2

ROS2 (Robot Operating System 2) is a middleware platform designed to facilitate the development of robot software. ROS2 is a continuation of the original ROS system, with the goal of improving its performance, scalability, and reliability. It is built on top of a communication layer called Data Distribution Service (DDS) [12], which is a widely used standard for distributed systems. DDS provides a middleware layer for communication between nodes, allowing for efficient and reliable data exchange across a network [13], [14].

2.2.2 Key Features of ROS2

ROS2 introduces several innovative features in comparison to the original ROS system. A primary feature is the support for real-time systems, expanding the range of robotics applications in which ROS2 can be employed. Furthermore, ROS2 presents a more modular architecture, facilitating superior code reuse and providing increased flexibility in the development of intricate robotics systems. This modular approach also offers a broader selection of options for different implementation aspects, such as the middleware, to cater to specific needs [14]–[16].

2.2.3 Standard Packages and Tools in ROS2

ROS2 comes with a multitude of standard packages that offer basic functionality for robotics development, encompassing control, perception, and navigation. These packages can be tailored or expanded to satisfy any application's specific requirements. In addition to these standard packages, ROS2 includes a comprehensive set of tools for debugging, testing, and visualization of robot software [17], [18].

2.2.4 ROS2 Architecture

ROS2 employs a distributed architecture with a layered structure, as illustrated in Figure 7. Within this system, nodes representing specific functionalities communicate via a messaging system. Central to this communication is the Publish-Subscribe (pub-sub) pattern, where nodes can publish to or subscribe to topics, facilitating decoupled interaction between system components. This pub-sub pattern is further exemplified in Figure 8 [14], [15].

These nodes interface with a middleware layer, responsible for data exchange and supporting diverse transport protocols, thus enabling communication over various networks, such as Ethernet or Wi-Fi [14], [15].

2.2.5 Community Support

One of the main benefits of ROS2 is its strong community support, which provides access to a large number of packages, tutorials, and documentation. The community also provides ongoing

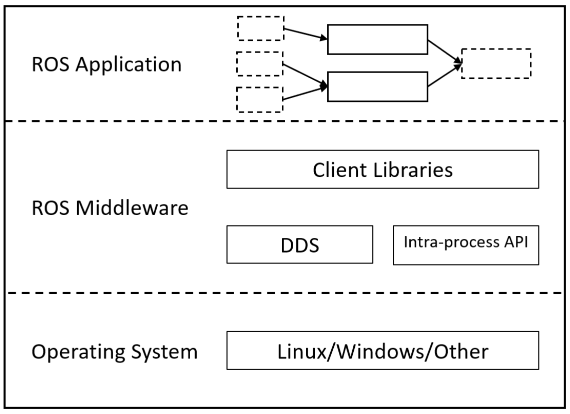


Figure 7: The layered structure of ROS2 based on figure from [15].

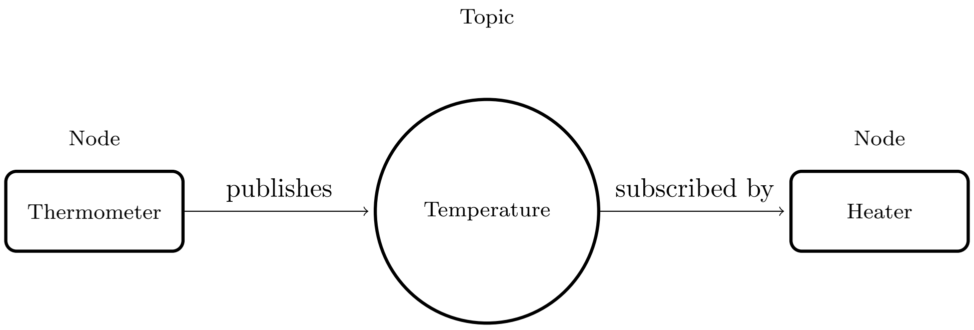


Figure 8: ROS 2 node and topic pattern illustrating a thermometer publishing to a temperature topic, which is subscribed to by a heater. The heater can be designed to activate when the temperature falls below a certain threshold and deactivates when the temperature rises above a specified limit.

development and support for the platform, which ensures that ROS2 remains a relevant and useful tool for robotics development [13].

Overall, ROS2 is a powerful and flexible platform for developing robot software. Its support for real-time systems, modularity, and community support make it a valuable tool for a wide range of robotics applications [14], [15], [19]

2.3 DeviceNet

2.3.1 Introduction and Overview

DeviceNet ODVA

Figure 9: The DeviceNet logo from ODVA's website [20]

DeviceNet is a network protocol designed for industrial automation and control systems [21]. It is based on the Controller Area Network (CAN) protocol and was developed by Allen-Bradley, a Rockwell Automation company[22]. It is now managed by ODVA (Open DeviceNet Vendors Association), an independent organization that has taken over the responsibility of the protocol, and their logo can be seen in Figure 9 [20]. DeviceNet provides a standardized communication interface for industrial devices, such as sensors, actuators, and other automation equipment, to communicate with a PLC or other supervisory devices.

2.3.2 Architecture

DeviceNet employs a server/client architecture, previously often referred to as a master/slave architecture. In this setup, the devices may function as servers, clients, or both. Servers are typically responsible for processing requests and producing responses, while the role of the clients is usually the converse - they generate requests and consume responses. Despite this standard model, the specifications of DeviceNet permit variations[23].

In the traditional model, the roles of the client devices align with the earlier conceptualization of the "master" role, while server devices correspond to the "slave" role. The client device, commonly a Programmable Logic Controller (PLC) or a Personal Computer (PC), oversees the communication and configuration of the network. Conversely, server devices represent the industrial equipment interconnected within the network. Each server device is assigned a unique identifier, utilized for addressing it within the network[20], [21], [23].

2.3.3 Communication and Network Features

DeviceNet supports a variety of communication speeds ranging from 125 kbps to 500 kbps, facilitating fast and reliable data exchange between devices [21]. It also supports multiple communication modes, including peer-to-peer, explicit messaging, and implicit messaging [23]. Peer-to-peer communication allows two devices to communicate directly with each other, while explicit messaging enables a client device to send specific commands to a server device. Implicit messaging permits devices to exchange data without explicit instruction, which proves useful for real-time control applications.

Moreover, DeviceNet accommodates several cable options that can be used either as a trunkline or dropline. However, the end-to-end network length can vary depending on the selected cable type and the data rate used, which can be seen in Table 1 [23]. These options offer flexibility and adaptability, allowing the network to be tailored to specific user requirements and operational conditions.

Table 1: DeviceNet Cable Lengths at Different Data Rates[23]

Cable Type	**125 kbps**	**250 kbps**	**500 kbps**
Thick Round Cable	500 m	250 m	100 m
Thin Round Cable	100 m	100 m	100 m
Flat Cable	420 m	200 m	75 m
Maximum Drop Length	6 m	6 m	6 m
Cumulative Drop Length	156 m	78 m	39 m

2.3.4 Reliability and Fault Tolerance Features

DeviceNet incorporates features to enhance reliable communication and ensure fault tolerance. Notably, it includes error detection mechanisms and provides facilities for resetting a communication relationship if/when errors are encountered [21]. The Controller Area Network (CAN) Data Frame, an integral part of the DeviceNet protocol, is followed by a Cyclic Redundancy Check (CRC) field designed to detect frame errors and is complemented by several frame formatting delimiters. This structure, combined with several types of error detection and fault confinement methods, such as CRC and automatic retries, contributes to the protocol's robustness. These methods are mainly transparent to the network, thereby preventing a faulty node from disrupting the network's operation [23].

Overall, DeviceNet is a reliable and widely used network protocol in industrial automation and control systems. Its features and capabilities make it suitable for a wide range of industrial applications, from simple monitoring and control tasks to complex automation systems [20]–[22], [24].

2.4 Beckhoff Automation and its Technologies

2.4.1 Beckhoff Automation

Beckhoff Automation is a German company that specializes in providing innovative automation solutions for industrial and manufacturing applications. Known for high-performance PC-based control systems, the company serves a wide range of industries, including automotive, food and beverage, pharmaceuticals, and more [25].

Innovative Products: Beckhoff's commitment to developing state-of-the-art automation solutions has resulted in a diverse range of products, from software platforms like TwinCAT to industrial Ethernet protocols like EtherCAT. Their solutions have significantly impacted numerous sectors due to their flexibility, scalability, and performance [26], [27].

Customer Focus: With a strong focus on customer satisfaction, Beckhoff Automation continuously evolves its product offerings to meet industry-specific requirements and changing market demands, reinforcing its reputation as a leading figure in the automation industry [26].

2.4.2 TwinCAT

TwinCAT, an essential product developed by Beckhoff, is a software platform that combines the functionality of a real-time operating system (RTOS) with the capabilities of a PLC [28], [29].

Versatile Platform: TwinCAT provides a wide range of features and tools for developing, configuring, and testing automation solutions. Its modular architecture and support for various programming languages make it adaptable to different types of applications and systems [30].

Integrated Development Environment: The Integrated Development Environment (IDE) of TwinCAT incorporates a visual programming interface, a debugging tool, and a simulation tool. This suite of tools enables developers to thoroughly test their solutions prior to deployment, thereby mitigating potential risks and bolstering the overall reliability of the system. Crucially, the PLC language IDEs utilized conform to the IEC 61131-3 standard, underscoring their robustness and interoperability [28], [30].

Real-Time Control: TwinCAT's support for real-time control enables it to handle time-critical tasks with high precision and accuracy, making it an ideal solution for applications such as motion control, robotics, and process automation [31].

To visually understand the architecture of TwinCAT and its integration of key features, one can refer to Figure 10. This figure provides a representation of the structural framework of TwinCAT.

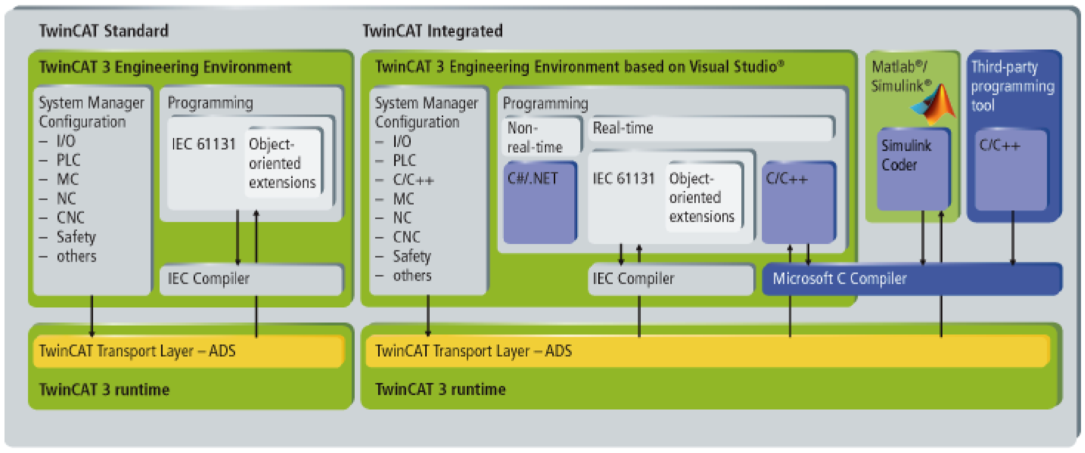

TwinCAT Standard
TwinCAT Integrated
TwinCAT 3 Engineering Environment
System Manager Configuration – I/O – PLC – MC – NC – CNC – Safety – others
Programming
IEC 61131
Object-oriented extensions
IEC Compiler
TwinCAT 3 Engineering Environment based on Visual Studio®
System Manager Configuration – I/O – PLC – C/C++ – MC – NC – CNC – Safety – others
Programming
Non-real-time
Real-time
C#/.NET
IEC 61131
Object-oriented extensions
C/C++
IEC Compiler
Matlab®/ Simulink®
Simulink Coder
Third-party programming tool
C/C++
Microsoft C Compiler
TwinCAT Transport Layer – ADS
TwinCAT 3 runtime
TwinCAT Transport Layer – ADS
TwinCAT 3 runtime

Figure 10: The TwinCAT Architecture/Features from [30]

2.4.3 EtherCAT

EtherCAT, a high-speed industrial Ethernet protocol, was developed by Beckhoff and is now managed by The EtherCAT Technology Group to ensure its openness to all potential users. Known for its exceptional performance, flexibility, and robustness, EtherCAT is designed for real-time communication between devices [32], [33].

Master-Slave Architecture: EtherCAT employs a master-slave architecture, allowing for efficient communication and distributed processing. The master and slave can potentially be implemented using standard network interfaces, which aids in reducing the load on the master device [32]. This design is also responsible for the impressive speed and synchronization accuracy of EtherCAT, making it one of the fastest Industrial Ethernet technologies [34].

Flexible Topologies and Efficient Communication: EtherCAT supports a wide range of network topologies, including line, tree, star, and various combinations thereof [32], [34]. Its unique communication system allows all nodes in an EtherCAT network to read data from and write data to the EtherCAT telegram as it passes by, with only a short constant delay in each slave device, independent of the packet's size. This efficient use of bandwidth optimizes performance and decreases interrupt rates. Thanks to automatic link detection, nodes and network segments can be disconnected and reconnected during operation, providing great flexibility and adaptability to various system configurations [32], [34], [35].

Robust and Simple: EtherCAT simplifies system configuration, diagnostics, and maintenance tasks. Its robustness is evident in its excellent electromagnetic noise immunity and capability to automatically detect and locate disturbances, significantly reducing troubleshooting time [32], [34].

Integrated Safety: EtherCAT incorporates Functional Safety over EtherCAT (FSoE), which meets the requirements for SIL 3 systems [34]. This safety protocol can be used in both centralized and decentralized control systems, providing an integrated approach to safety without compromising performance or cost-efficiency [34], [36].

Affordability: Despite offering the advanced features of Industrial Ethernet, EtherCAT does so at a price point similar to, or even lower than, traditional fieldbus systems. The hardware required for the master device is a standard Ethernet port, and the EtherCAT slave devices, available from various manufacturers, handle all time-critical tasks, further reducing overall system costs [34], [36].

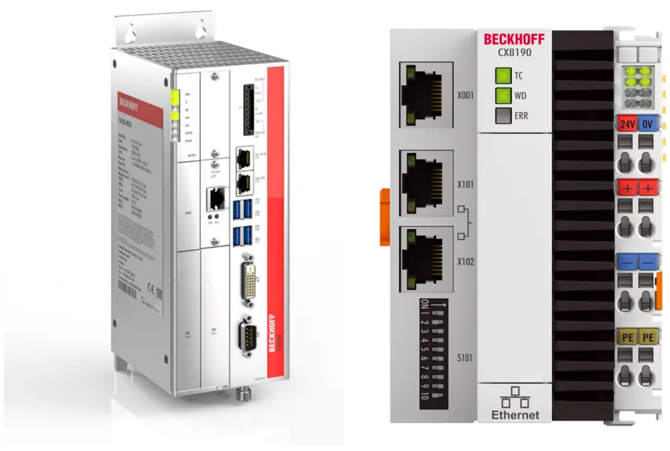
2.4.4 Industrial and Embedded PC PLCs

Beckhoff Automation provides a range of both industrial and embedded PC PLCs, each with its own advantages and intended uses [25].

Industrial PCs: Beckhoff's industrial PCs are high-performance computing solutions designed for robust and powerful control tasks. These systems, while exceptionally capable, require the addition of EtherCAT terminal couplers in order to incorporate IO into the system. This is due to the architecture of the industrial PCs, which are not inherently designed to add terminals directly [25], [37].

Embedded PC PLCs: In contrast to the industrial PCs, Beckhoff's embedded PC PLCs are compact, DIN rail-mountable controllers that can have IO terminals added directly to them. These embedded systems, such as the CX7000 and CX8100 series, provide a compact and cost-effective solution for small to complex automation tasks, with the added convenience of direct terminal integration [25], [38].

Figure 11 displays two examples of Beckhoff's PC PLCs. The C6930 is representative of their industrial PCs, known for robust control tasks. The compact CX8190, from the CX8100 series, epitomizes the company's embedded PCs. Each model highlights the diverse automation solutions offered by Beckhoff.



(a) (b)

Figure 11: Examples of PLCs / PCs from Beckhoff. Figure 11a is the C6930, an Industrial Cabinet PC [39]. Figure 11b shows the CX8190, an Embedded PC PLC from the CX8100 series from [40].

CX7000: The CX7000 is a compact embedded PC that offers PLC and motion control functionalities. In addition to its processing capabilities, it also has built-in digital IO. This makes it a flexible and cost-effective solution for small and medium-sized control tasks, reducing the need for additional components [38], [41].

CX8100: The CX8100 series offers higher performance and expanded interface options while maintaining a compact form factor. It is designed to meet the demanding requirements of complex automation tasks [38], [42].

When IO terminals, specifically the E-series terminals, are added to these embedded systems, the communication between the PLC and the terminal is facilitated over EtherCAT. This offers high-speed, efficient data exchange, contributing to the overall performance and versatility of the system [35].

Overall, Beckhoff Automation is a leading provider of innovative automation solutions for industrial and manufacturing applications. Its advanced products, such as the TwinCAT software platform and EtherCAT protocol, together with the range of compact and powerful embedded PC PLCs like the CX7000 and CX8100 series, contribute significantly to the industry's advancement.

2.5 Introduction to OPC Unified Architecture (OPC UA)

2.5.1 Overview of OPC UA

Figure 12: OPC Foundation logo. [43]

OPC Unified Architecture (OPC UA) is a platform-independent, service-oriented architecture (SOA) developed by the OPC Foundation [44], Figure 12. It is designed to provide seamless communication, integration, and interoperability between industrial devices, systems, and applications. OPC UA offers a standardized and reliable way to exchange data, alarms, events, and historical information in various industrial domains, including manufacturing, energy, transportation, and building automation [44], [45].

2.5.2 Key Features of OPC UA

Platform Independence

OPC UA is noted for its exceptional platform independence, supporting various hardware platforms, operating systems, and programming languages. This includes regular computer hardware (PCs), PLCs, micro-controllers, cloud-based servers, and operating systems from Microsoft Windows to Linux distributions. Its cross-platform communication capability ensures interoperability across an enterprise, encompassing machine-to-machine (m2m) and machine-to-enterprise interactions, enabling a comprehensive automation environment [44].

Information Modeling

OPC UA incorporates a robust framework for information modelling that allows users and companies to create custom data models to represent real-world entities, systems, and processes. This feature enables the integration and semantic understanding of information across different devices, systems, and industries [44], [46].

Scalability

OPC UA's high scalability caters to a wide range of systems, from small-scale setups with limited resources to large-scale ecosystems with thousands of devices. The versatility of the OPC UA specification even allows for its integration down to the level of single field devices, demonstrating its adaptability and comprehensive approach to system integration [45].

Security

OPC UA prioritizes robust security, ensuring data integrity, confidentiality, and authenticity. By employing multiple protocols, session encryption, and user authentication through X509 certificates, it protects communication channels effectively [44]. Also, OPC UA offers built-in services such as Alarms and Conditions, Historizing, Server Aggregation, and Discovery, enhancing its overall functionality. These security measures make OPC UA a highly secure yet firewall-friendly solution for technology implementation [44], [46], [47].

2.5.3 OPC UA Architecture

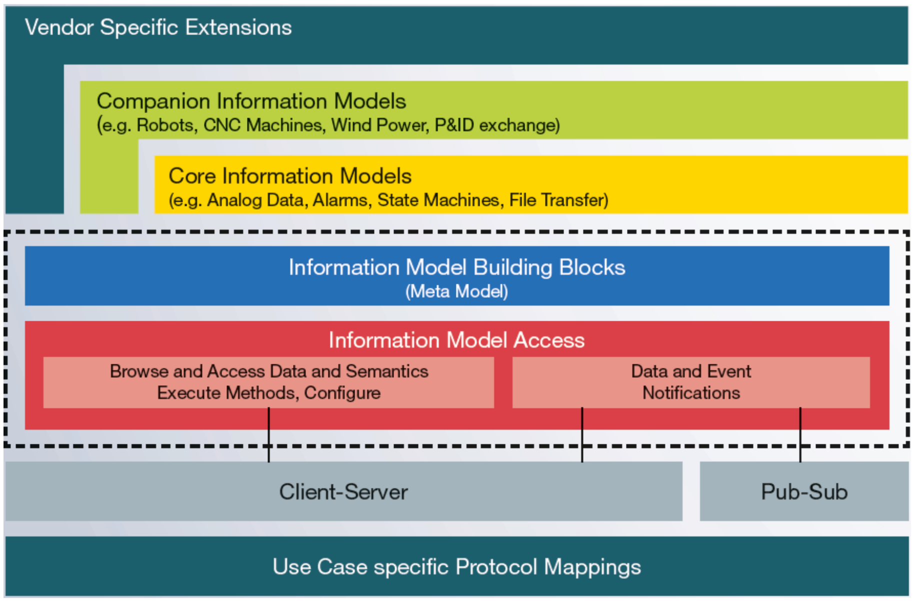

Vendor Specific Extensions
Companion Information Models
(e.g. Robots, CNC Machines, Wind Power, P&ID exchange)
Core Information Models
(e.g. Analog Data, Alarms, State Machines, File Transfer)
Information Model Building Blocks
(Meta Model)
Information Model Access
Browse and Access Data and Semantics
Execute Methods, Configure
Data and Event
Notifications
Client-Server
Pub-Sub
Use Case specific Protocol Mappings

Figure 13: OPC UA architecture from the OPC foundations website [44].

The OPC UA architecture consists of two primary components: OPC UA servers and OPC UA clients. Servers provide access to data, alarms, events, and historical information, while clients consume this information to perform various tasks, such as monitoring, analysis, and control [44], [46]. Figure 13 shows the multilayered architecture of OPC UA that was just presented and the features described in the previous parts.

Address Space

The address space is a fundamental concept in OPC UA, organizing and representing the server's data and metadata as a collection of interconnected nodes. The employed information model defines the nature of these interconnections. These nodes can represent objects, variables, methods, or other types that define the system's structure and behavior [44], [45].

Services

OPC UA servers offer various services that clients can use to interact with the address space. Services from these sets include operations like reading, writing, browsing, method calling, and subscription-based services for managing data, alarms, and events [44], [46].

2.5.4 OPC UA in Industrial Applications

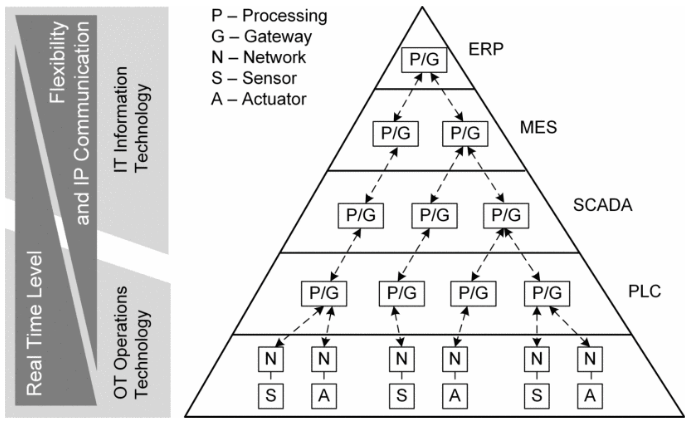


Figure 14: There are numerous variations of the automation pyramid. The following is an example taken from [48].

OPC UA, known for its ubiquity and versatility, has gained immense popularity across industrial applications, notably within the higher layers of the industrial automation pyramid, as depicted in Figure 14. It is particularly becoming prevalent in [47], [49]:

1. Supervisory Control and Data Acquisition (SCADA) systems: By leveraging its real-time capabilities and scalability, OPC UA is becoming an integral component of SCADA systems.
2. Manufacturing Execution Systems (MES): Owing to its platform independence and seamless interoperability, OPC UA enhances data exchange in MES.
3. Industrial Internet of Things (IIoT) applications: OPC UA's robust security features make it an apt choice for IIoT applications.
4. Data Integration in heterogeneous systems: Due to its extensive hardware and operating system support, OPC UA is increasingly used for data integration across diverse systems.

The adoption of OPC UA across these layers demonstrates its vast potential, hinting at its promising role even within the lower tiers of the pyramid [47], [49].

2.5.5 UaExpert - OPC UA Client

UaExpert, developed by Unified Automation, is a comprehensive OPC UA client designed as a general-purpose test client for OPC UA servers [50]. It is a critical tool in assessing various functionalities and performance, including data access, server address space browsing, historical access, alarms and conditions management, and method calls [50].

The graphical user interface (GUI) of UaExpert is both sophisticated and user-friendly, designed to facilitate efficient and effective testing processes. Written in C++, the software exhibits cross-platform compatibility, making it an adaptable choice for diverse systems [50].

Central to UaExpert's design is a core framework that plugins can further enhance. This extensible architecture allows for customized functionality tailored to specific testing requirements, further establishing UaExpert as a versatile tool in OPC UA server development and testing [50].

2.5.6 Free OPC-UA Library

The Free OPC-UA Library is an open-source initiative that provides the necessary resources for developing servers and clients within the OPC UA framework. This project includes libraries for both Python and C++, along with GUI clients and models [51]. The python-opcua library was initially developed for Python clients in pure code form. However, the recommendation for Python versions 3.7 and beyond is to utilize the opcua-asyncio library [52]. This library is an asyncio-based asynchronous OPC UA server and client that builds upon the python-opcua library. One downside is the loss of support for earlier Python versions.

The primary advantage of asynchronous programming is the simplification of code due to less need for locks, along with potential performance enhancements. However, for cases requiring synchronous code, the library provides a synchronous wrapper over the asynchronous API, allowing the opcua-asyncio library to be used instead of python-opcua. Thus, if a recent version of Python is in use, there is no reason to prefer python-opcua over opcua-asyncio [51], [52].

In summary, OPC UA plays a vital role in industry digitization by providing a standardized, secure, and efficient method for data exchange and integration. Its platform independence, powerful information modelling, high scalability, and robust security mechanisms combined with an architecture designed for seamless communication and interoperability make it a go-to solution for various industrial applications. Furthermore, OPC UA's adoption across different layers of the industrial automation pyramid underscores its versatility in addressing complex industrial challenges, solidifying its role as a cornerstone of Industry 4.0.

2.6 MQTT

Figure 15: MQTT logo. [53]

2.6.1 Introduction and History

MQTT (Message Queuing Telemetry Transport) is an efficient and lightweight messaging protocol designed for machine-to-machine (M2M) and Internet of Things (IoT) communication, with the logo shown in Figure 15. It was invented by Dr Andy Stanford-Clark from IBM and Arlen Nipper from Arcom (now Eurotech) in 1999 [54], and has since become one of the most popular protocols for IoT communication [55], [56].

2.6.2 Architecture

MQTT is based on the publish-subscribe pattern, which allows multiple clients to exchange data through a central broker. In MQTT, a client can publish messages to a broker on a specific topic, and any client that has subscribed to that topic will receive the message. This allows for highly efficient and scalable communication between devices, as clients only need to subscribe to the topics they are interested in. A simple example can be seen in Figure 16 [55]–[57].

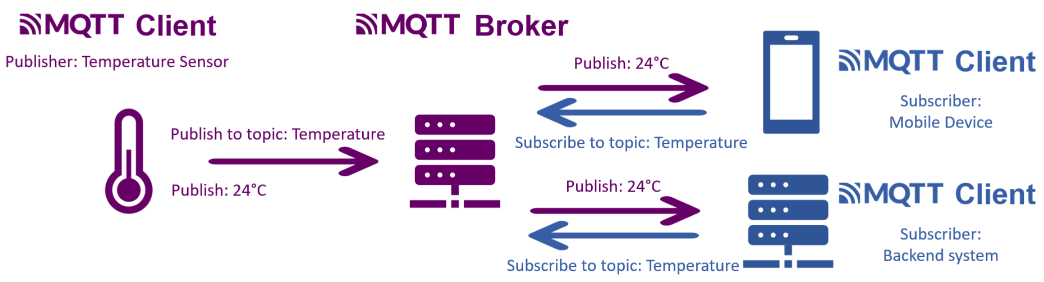

MQTT Client
Publisher: Temperature Sensor
MQTT Broker
Publish to topic: Temperature
Publish: 24°C
Publish: 24°C
Subscribe to topic: Temperature
MQTT Client
Subscriber: Mobile Device
Publish: 24°C
Subscribe to topic: Temperature
MQTT Client
Subscriber: Backend system

Figure 16: An simple example of the MQTT publish-subscribe Architecture, based on the figure from [55].

2.6.3 Key Features and Efficiency

One of the key features of MQTT is its lightweight nature. The protocol is designed to be highly efficient, using minimal bandwidth and requiring low processing power. This makes it ideal for use in low-power and low-bandwidth devices such as sensors and other IoT devices. Mosquitto is a popular open-source broker that implements MQTT and can run on a variety of platforms, including embedded systems and microcontrollers. With its focus on efficiency and scalability, Mosquitto is a great choice for IoT applications that require low latency and high throughput [55], [57], [58].

2.6.4 Quality of Service (QoS) Levels

MQTT also supports Quality of Service (QoS) levels, which define the level of guarantee for message delivery. There are three QoS levels in MQTT [56]:

- **QoS 0**: At most once delivery. Messages are delivered once or not at all, without any acknowledgement from the receiver.

- **QoS 1**: At least once delivery. Messages are guaranteed to be delivered at least once, but may be delivered multiple times.
- **QoS 2**: Exactly once delivery. Messages are guaranteed to be delivered exactly once, ensuring that there are no duplicate messages.

The choice of QoS level depends on the specific use case and the importance of message delivery. For example, a sensor reading may be published with QoS 0, as it can be non-critical to receive every reading. On the other hand, a command to actuate a device may be published with QoS 2, because it can be critical that the command is executed exactly once [55], [56].

Overall, MQTT is a highly efficient and scalable protocol that is well-suited for IoT and M2M communication. Its lightweight nature and support for QoS levels make it a popular choice for a wide range of IoT applications, from home automation to industrial monitoring and control.

2.7 Fronius TPS/i Welding Machine

The Fronius TPS/i is an innovative welding system designed to meet the demands of modern industrial applications. It is a modular, intelligent welding system that offers a high level of flexibility, precision, and efficiency. In addition to its manual welding capabilities, the TPS/i can be integrated with robotic arms, which are often employed in automated welding processes. With the help of a robot interface, the TPS/i welding system can be easily adapted for different types of robotic arms and communication protocols, such as DeviceNet [59], [60].

2.7.1 System Architecture

The Fronius TPS/i welding system consists of several components, including a power source, a wire feeder, a welding torch, a user interface, and an optional robot interface. The modular design enables users to configure the system according to their specific requirements, and easily upgrade or expand the system as needed. The design can be seen in Figure 17. Some of the important components [59], [61], [62]:

Power Source: The power source is the heart of the TPS/i welding system, responsible for generating the required welding current and voltage. It employs advanced inverter technology to ensure a stable and precise welding process and high energy efficiency.

Wire Feeder: The wire feeder supplies the welding wire to the welding torch at a controlled speed, ensuring a consistent and accurate welding process. It is designed to handle various wire diameters and types, offering a high level of versatility.

Welding Torch: The welding torch is the primary interface between the operator and the workpiece. It delivers the welding current and shielding gas to the welding area and supports various welding processes, such as MIG/MAG, TIG, and Stick welding.

User Interface: The user interface provides an intuitive means for the operator to interact with the TPS/i welding system. It offers a touchscreen display, allowing users to easily set up and control the welding process, access system diagnostics, and perform software updates.

Robot Interface: The robot interface, Figure 18, enables seamless integration between the TPS/i

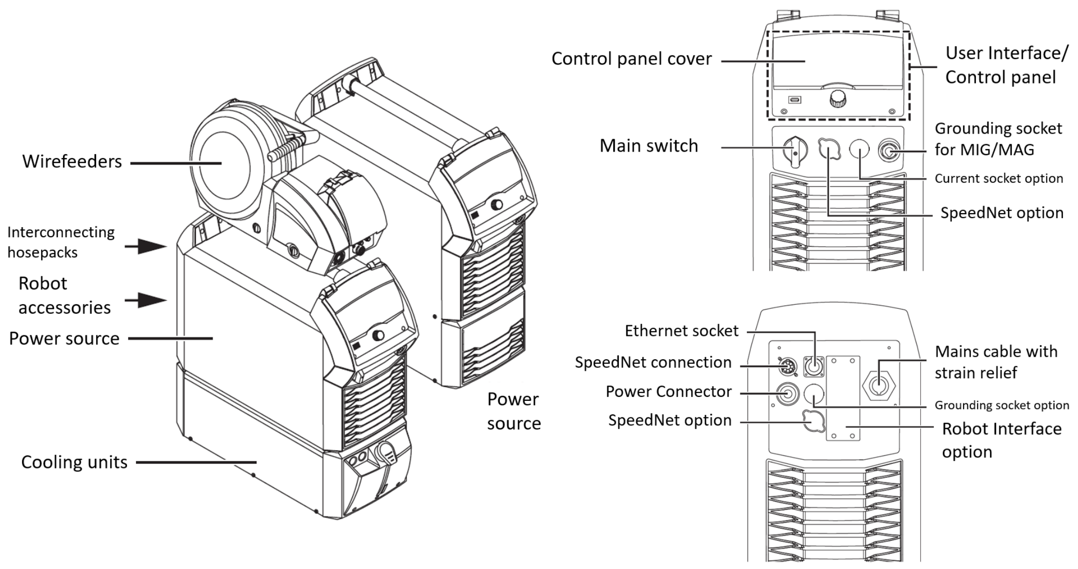


Figure 17: The modular design of Fronius TPS/i welding machine. Based on figures from [62].

welding system and various types of robotic arms. This component allows the welding system to communicate with the robot controller using different communication protocols, such as DeviceNet, facilitating synchronized operation in automated welding processes [60].

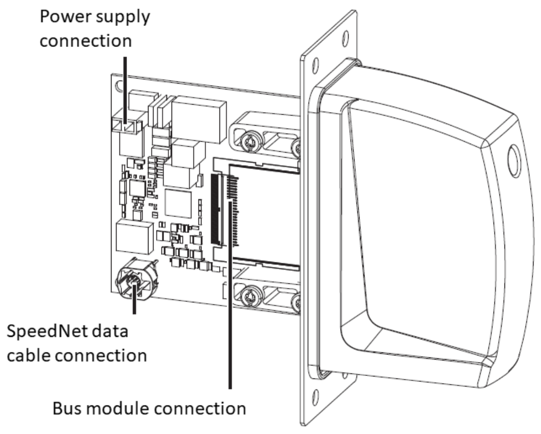


Figure 18: The robot interface. Based on a figure from [60].

2.7.2 Working Principles

The Fronius TPS/i welding machine operates on the principle of electric arc welding, using a consumable electrode (welding wire) to create a weld pool and join the workpieces. The welding process can be categorized into the following steps:

1. The operator initiates the welding process by pulling the trigger on the welding torch, or the robotic arm initiates the process automatically in an automated welding setup.
2. The power source generates the required welding current and voltage, initiating the electric arc between the welding wire and the workpiece.

3. The wire feeder supplies the welding wire to the welding torch at a controlled speed, ensuring a consistent and accurate welding process.
4. The electric arc generates heat, melting the welding wire and the workpieces' edges, creating a weld pool.
5. Shielding gas is supplied to the welding area to protect the molten metal from atmospheric contamination.
6. The weld pool solidifies, forming a strong joint between the workpieces.

2.7.3 Key Features

Intelligent Process Control: The TPS/i welding system incorporates advanced process control algorithms that optimize the welding process in real-time, ensuring high-quality welds and minimizing the need for post-welding rework. It is achieved with features such as LSC (Low Spatter Control), PMC (Pulse Multi Control), and PCS (Pulse Controlled Spray Arc) [59], [63].

Communication and Connectivity: The TPS/i welding system supports various communication and connectivity options, such as Ethernet, USB, and WLAN, enabling remote monitoring, diagnostics, and control. Moreover, the robot interface allows for seamless integration with different types of robotic arms and communication protocols, such as DeviceNet, CANopen, EtherCAT, and Profibus [60], [62], [64].

In summary, the Fronius TPS/i welding machine provides a comprehensive solution for various industrial welding applications due to its flexibility, precision, and efficiency. With its compatibility with robotic arms and support for multiple communication protocols, the TPS/i system is well-equipped to meet the challenges of both manual and automated welding processes, making it a valuable tool in modern manufacturing environments.

2.8 Yaskawa and Robotic Welding

Yaskawa Electric Corporation, established in 1915, is a worldwide leader in the production of industrial robots, motion control systems, and automation solutions [65]. Yaskawa has a significant presence in a variety of industries, including automotive, electronics, and material handling.

Yaskawa Motoman is a subsidiary of Yaskawa established in 1989 in the United States. It provides automation products and solutions across a vast range of robotic applications and industries [65]. One of their key areas of expertise is robotic welding, which has transformed the welding industry by offering heightened precision, productivity, and cost savings [65], [66].

2.8.1 Yaskawa Robotic Welding Solutions

Yaskawa Motoman offers various robotic welding solutions, including arc welding, laser welding, and spot welding [67]. These solutions are designed to address the unique challenges of the welding industry, such as maintaining consistent quality, reducing cost, reducing waste, and ensuring worker safety [66].

Yaskawa's robotic welding solutions integrate innovative technologies and features that augment the welding process, including [68], [69]:

- High-speed, high-accuracy robotic arms, which minimize the necessity for post-weld rework due to consistently high-quality welding.
- State-of-the-art welding power sources from manufacturers like Fronius, which streamlines the welding process for enhanced efficiency and energy conservation.
- User-friendly programming and simulation software, which simplifies the process of designing and optimizing weld programs.
- Comprehensive safety features, which diminish the risk of accidents and injuries during the welding process.

2.9 Yaskawa YRC1000 Robot Controller

The Yaskawa Motoman YRC1000 robot controller is a state-of-the-art control system designed to operate Yaskawa industrial robots, including those used in robotic welding applications. The controller is known for its precise control, advanced safety features, and high-speed performance [70].

2.9.1 High-Speed Processing

The YRC1000 is equipped with a high-speed, multitasking CPU that enables swift and precise robotic control. This contributes to improved cycle times, heightened productivity, and enhanced overall efficiency in the production process. The advanced processing capabilities of the YRC1000, in conjunction with Yaskawa's high-speed servos, allow it to manage complex automation tasks at high speeds and accelerations effortlessly [68], [70].

2.9.2 Safety Features

Safety is one of the most critical concerns when dealing with industrial robotics. To this end, the YRC1000 comes with two primary channels to ensure safe operation: the MSU (Emergency Stop and Safety Signals for safe I/O communication) and the FSU (Functional Safety Board). Devices such as safety scanners or light barriers can also connect directly to the controller. Additionally, the YRC1000 incorporates features like Safe Torque Off (STO), which enhances the safety of both the robot and the operator. These features ensure that the robotic system operates within predefined safety parameters, minimizing the risk of accidents and injuries [68], [70].

2.9.3 Programming

The robot controller can be programmed using a language Yaskawa has developed called INFORM. This can be done using the Standard Teach Pendant, which is a highly functional, menu/command-line-based interface. INFORM is a language that uses instructions paired with additional items, such as tags and numerical data. An example of an INFORM line is "MOVJ VJ=50.00", where "MOVJ" is the instruction, and "VJ=50.00" is the additional item. The instruction determines what operation is to be performed, while the additional item specifies parameters such as speed or time. This structure makes INFORM an intuitive and straightforward programming language for the robot's movements and actions, enabling users to create complex automation tasks with relative ease. The flexibility of the INFORM language ensures that the YRC1000 can be customized to suit a wide range of applications [70], [71].

Motoman robots connected to the YRC1000 controller can also be programmed in other ways, with different methods suiting various types of environments. Moreover, customers can develop their own programs to interface with these systems using Yaskawa's Software Development Kits (SDKs), such as the MotoPlus C/C++ and MotoCom32 SDKs [70].

3 Proposed Solution

The solution to the task presented in the Project Description in Section 1.2 involves utilizing a Programmable Logic Controller (PLC) as a DeviceNet master to control the welding machine via this protocol. Subsequently, this PLC is controlled through the OPC UA protocol, enabling any computer to control it. Ideally, the controlling PC should operate the ROS2 for the robotic arm, facilitating control via ROS2. Figure 19 illustrates how this proposed solution operates.

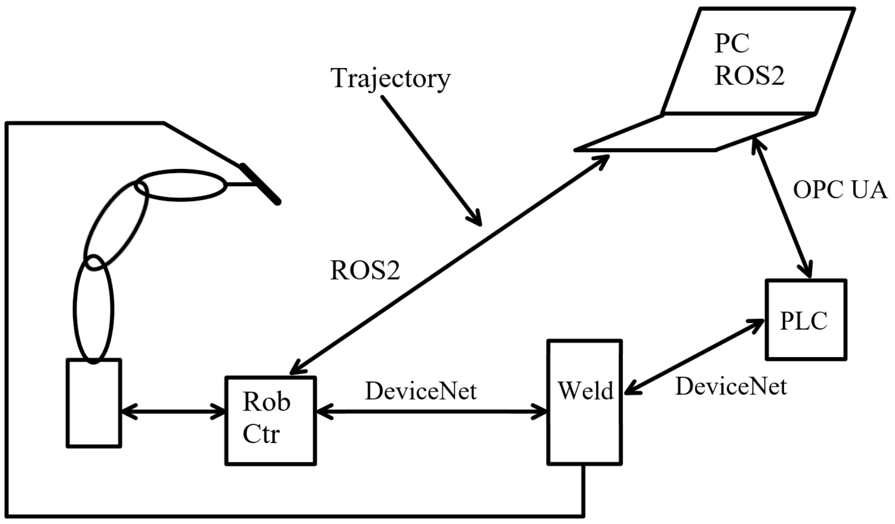


Figure 19: A concept illustration of the devised solution.

The selection was made to implement a PLC from Beckhoff, as equipment from the same manufacturer was already integrated within the pre-existing robot cell. The chosen PLC is a CX8190 Embedded PC, complemented by an EL6752 DeviceNet master terminal. The choice of the CX8190 was guided by its affordability, sufficient computational power, and compatibility with Beckhoff-provided OPC UA software.

The configuration used for testing is illustrated in Figure 20. The control cabinet housing the PLC is evident in the image. The grey Ethernet cable extends from the PLC and connects to the HP network switch. Additionally, the purple DeviceNet cable emerging from the PLC and directed toward the red welding apparatus in the background can be observed. The two computers utilized in the testing process are interfaced with the PLC via the HP network switch.

In Figure 21 and Figure 22, the experimental setup employed to test the functionality of a PC running Linux is portrayed. This test is crucial given that the computer destined to operate ROS2 will be a Linux machine. Figure 21 reveals that the Linux PC and the PC running TwinCAT are connected to the PLC through a small router. This router is non-commercial, rendering it unsuitable for obtaining performance metrics. However, it effectively demonstrates that the communication between the PLC and the Linux PC is viable. As depicted in Figure 22, this router is linked to a PLC which was not in the frame in the preceding figure.

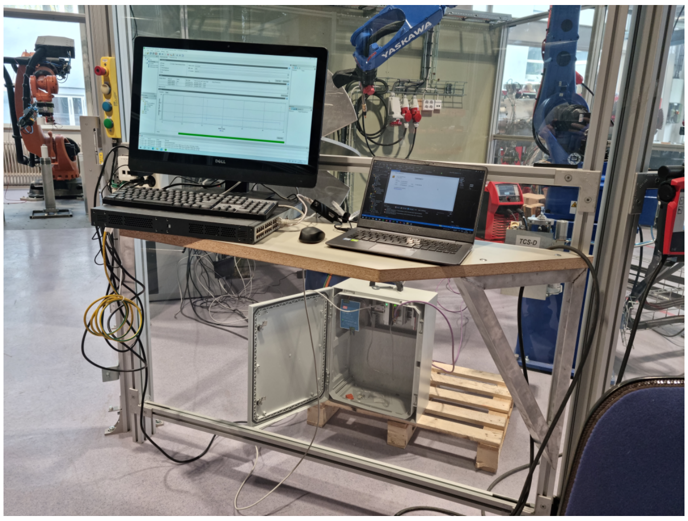


Figure 20: A image depicting the test setup used for the tests done in the results section, Section 6.

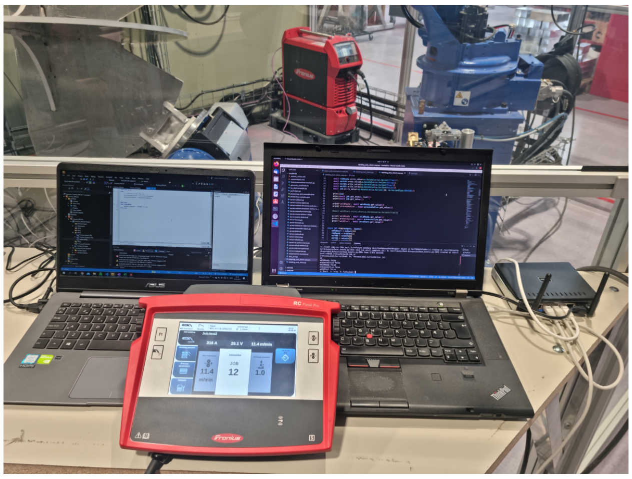


Figure 21: A image depicting the test done with a Linux computer.

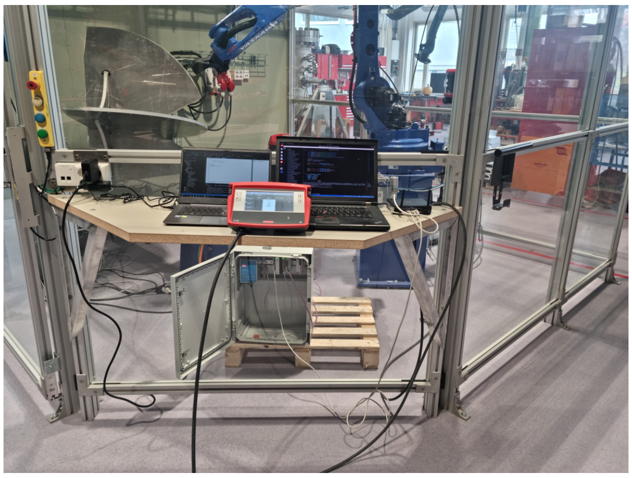


Figure 22: A image depicting the test done with a Linux computer from another angle, showing the PLC in the control cabinet.

4 Setting up the Computer, PLC, and cables

This section will provide a comprehensive walkthrough on how to configure the solution illustrated in Section 3. It aims to clearly outline each step in the process, ensuring a thorough understanding of the setup.

4.1 Setup of the Hardware

4.1.1 Commissioning the PLC

Proper installation of a Programmable Logic Controller (PLC) is crucial for its efficient operation. The suggested practice is to mount the PLC on a D rail inside a control cabinet, as depicted in Figure 23. This specific orientation is strategically advantageous as it allows the PLC's passive cooling mechanism to function effectively. The mechanism operates on the principle of natural convection where warmer air ascends, leaving room for cooler air, thus helping maintain a suitable operational temperature for the PLC.

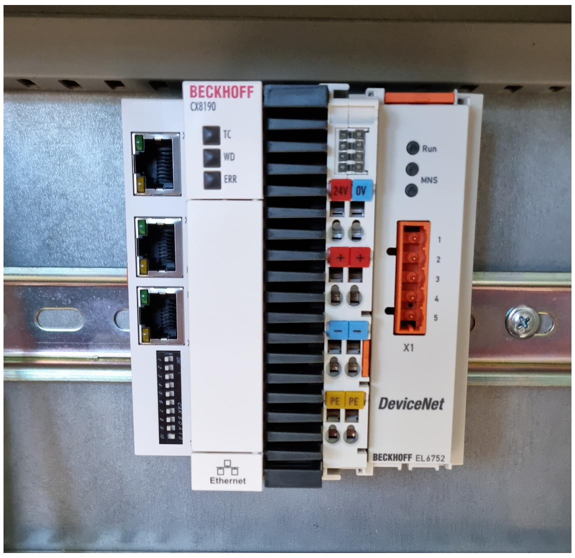
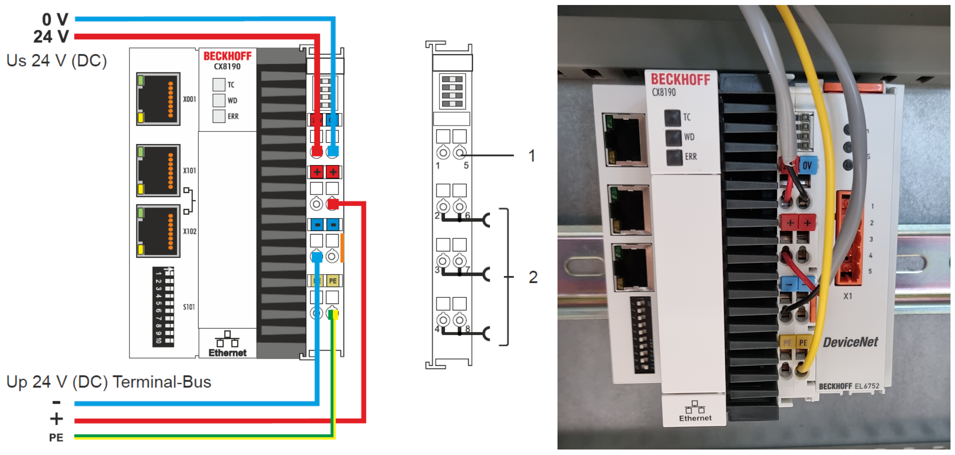


Figure 23: The PLC (CX8190 with EL6752) mounted inside a control cabinet.

Next, we delve into the details of connecting the power supply to the PLC. As depicted in Figure 24a, which is derived from Beckhoff's guide [72], it is crucial to connect +24V and 0V to the ports marked similarly, in order to power the PLC itself.

Moreover, it is essential to connect 24V and 0V to at least one of the two ports labelled with + or - respectively, for providing power to the modules attached to the E-bus or K-bus of the PLC. It is equally important to remember to connect the grounding to one of the two ports designated for that purpose. Figure 24b provides an exemplar illustration of how these connections can be implemented in a real-world setting.



(a) (b)

Figure 24: Sub-figure (a) shows how to connect the power supply to the CX8190 PLC from Beckhoff's guide [72]. Sub-figure (b) showcases a practical implementation.

4.1.2 DeviceNet Cable

Various types of connectors can be utilized for DeviceNet. Both the welding apparatus and the PLC utilize a Straight Female Header Screw-terminal with a 5.08mm Pitch and 5 Poles. These come with the PLC module for DeviceNet, but it does not fit with every type of cable shoe. Also, one of the guide pins in the plastic was placed differently than normal, which can be rectified by removing one of the plastic guide pins from another header.

A 24V power supply is needed on the cable from an external source. This can conveniently be the same source that supplies power to the PLC. It is recommended to use cable shoes, to get a proper and secure connection between the cable and the Female Header type connector. It's important to consider the necessary elements, as one can purchase a pre-made cable. A completed Cable can be seen in Figure 25.

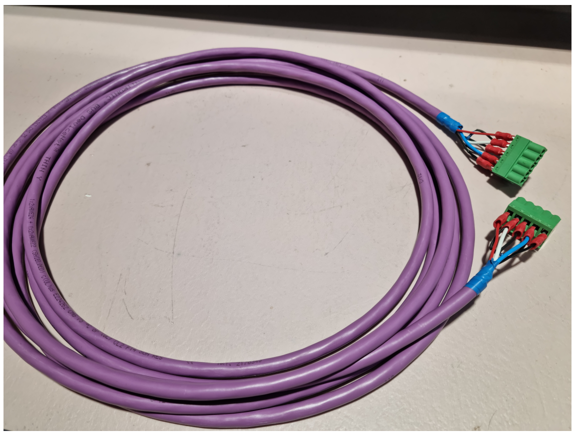
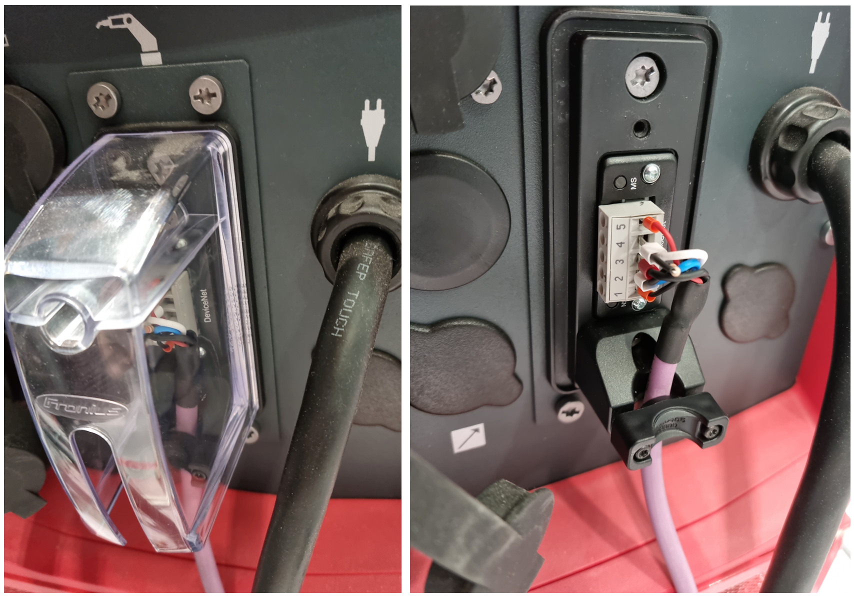


Figure 25: DeviceNet Cable

Connecting to Fronius will require permanent wiring, as shown in the Figure 26. For a permanent connection, it is recommended to use the cable clamp that holds the cable in place and the plastic cover, as demonstrated with the connection with the Yaskawa controller in Figure 26. However, this can be cumbersome for testing purposes where one needs to disconnect and reconnect the original setup when not in operation.



(a) (b)

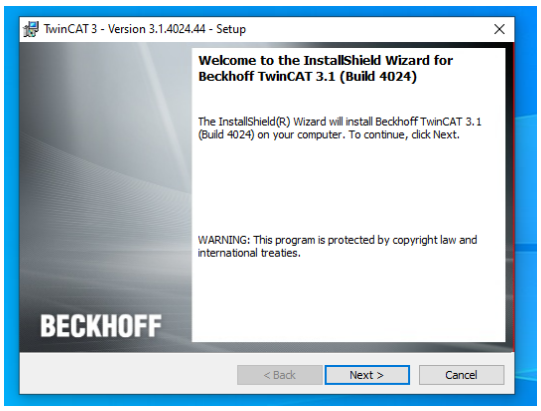
Figure 26: Figure 26a shows the DeviceNet connection to the Fronius welding machine with the protective cover in place. Conversely, Figure 26a shows the connection without the cover, highlighting the cable clamp that ensures that the DeviceNet cable is in place.

4.2 Setup of the Software

In order to facilitate PLC programming and ensure full functionality, various software components need to be installed on both the PC and the PLC.

4.2.1 Installing the Required Software for the PC

The installation of TwinCAT XAE, from [28], is required and necessitates the use of a Beckhoff user account. It is important to note that TwinCAT XAR is included within the TwinCAT XAE package; therefore, separate installation of XAR would result in the uninstallation of XAE. TwinCAT XAE is Visual Studio-based, enabling users who either have or desire to incorporate the XAE application into other Visual Studio versions and utilize them as an alternative. The installation process is relatively straightforward, simply requiring adherence to the steps presented in the installation wizard, which typically involves using the default settings. Figure 27 depicts some of the stages of the installation wizard. If one has other versions of Visual Studio and wishes to integrate TwinCAT XAE into them, it is crucial to ensure this option is selected during the installation process. For local code execution and interim code testing, the utilization of seven-day trial licenses is possible.

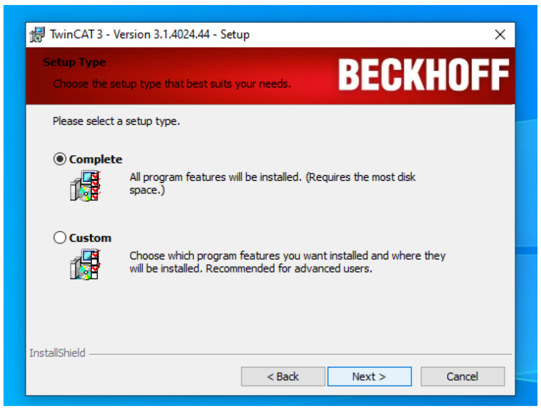

TwinCAT 3 - Version 3.1.4024.44 - Setup

Welcome to the InstallShield Wizard for Beckhoff TwinCAT 3.1 (Build 4024)

The InstallShield(R) Wizard will install Beckhoff TwinCAT 3.1 (Build 4024) on your computer. To continue, click Next.

WARNING: This program is protected by copyright law and international treaties.

(a)


TwinCAT 3 - Version 3.1.4024.44 - Setup

Setup Type

Choose the setup type that best suits your needs.

Please select a setup type.

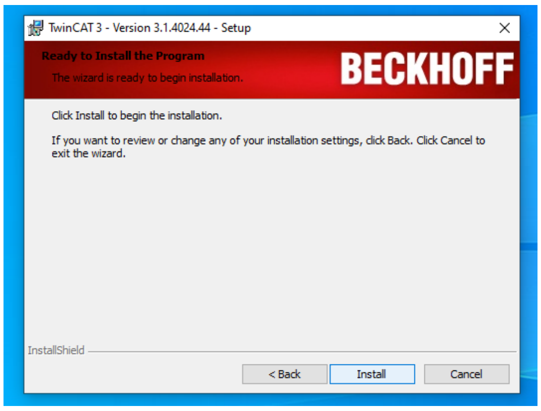
[✓ **Complete**]

All program features will be installed. (Requires the most disk space.)

Custom

Choose which program features you want installed and where they will be installed. Recommended for advanced users.

(b)


TwinCAT 3 - Version 3.1.4024.44 - Setup

Ready to Install the Program

The wizard is ready to begin installation.

Click Install to begin the installation.

If you want to review or change any of your installation settings, click Back. Click Cancel to exit the wizard.

(c)

Figure 27: Depiction of three stages in the installation wizard for Beckhoff's TwinCAT3 XAE.

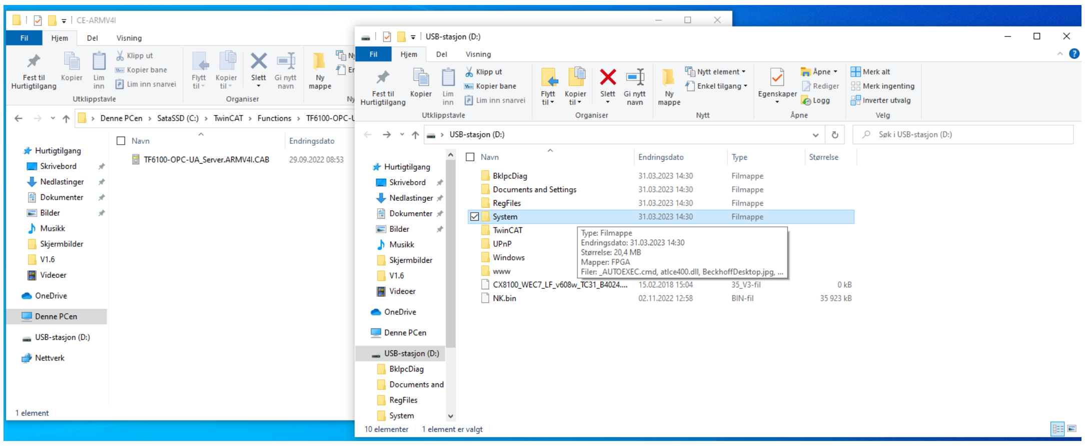
Upon successful installation of TwinCAT XAE, the next step is to install the relevant supplementary modules. In this context, the OPC UA module TF6100 is required and can be obtained from [73]. After the installation of this module, it's necessary to ensure the activation of the corresponding license, which could be either a trial or a full license. Further details regarding this process will be addressed in subsequent sections of this guide.

4.2.2 PLC Software

To install the OPC UA on the PLC, it must first be installed on the PC. Once accomplished, it will reside within the directory where the TwinCAT XAE is installed, specifically under the "Functions" folder. One must then navigate to the OPC UA (TF6100-OPC-UA) folder, wherein various versions of it can be found.

Given that the PLC intended for this installation is a CX8190, an ARM-based PLC with Windows CE, the appropriate folder is "CE-ARMV4I". Within this folder, a CAB file is present, which needs to be copied to the PLC. Multiple methods for this procedure are outlined in [73]. In this case, the chosen method was to directly copy the file to the memory card with the PLC's Operating System (OS).

The location where the file is stored on the PLC is not critical, as long as it is accessible from the PLC's OS. Consequently, the file was copied to the "System" folder, see Figure 28.


CE-ARMV4I

Navn	Endringsdato
TF6100-OPC-UA_Server.ARMV4I.CAB	29.09.2022 08:53

USB-stasjon (D:)

Navn	Endringsdato	Type	Størrelse
BklpcDiag	31.03.2023 14:30	Filmappe	
Documents and Settings	31.03.2023 14:30	Filmappe	
RegFiles	31.03.2023 14:30	Filmappe	
System	31.03.2023 14:30	Filmappe	
TwinCAT			
UPnP			
Windows			
www			
CX8100_WEC7_LF_v608w_TC31_B4024....	15.02.2018 15:04	35_V3-fil	0 kB
NK.bin	02.11.2022 12:58	BIN-fil	35 923 kB

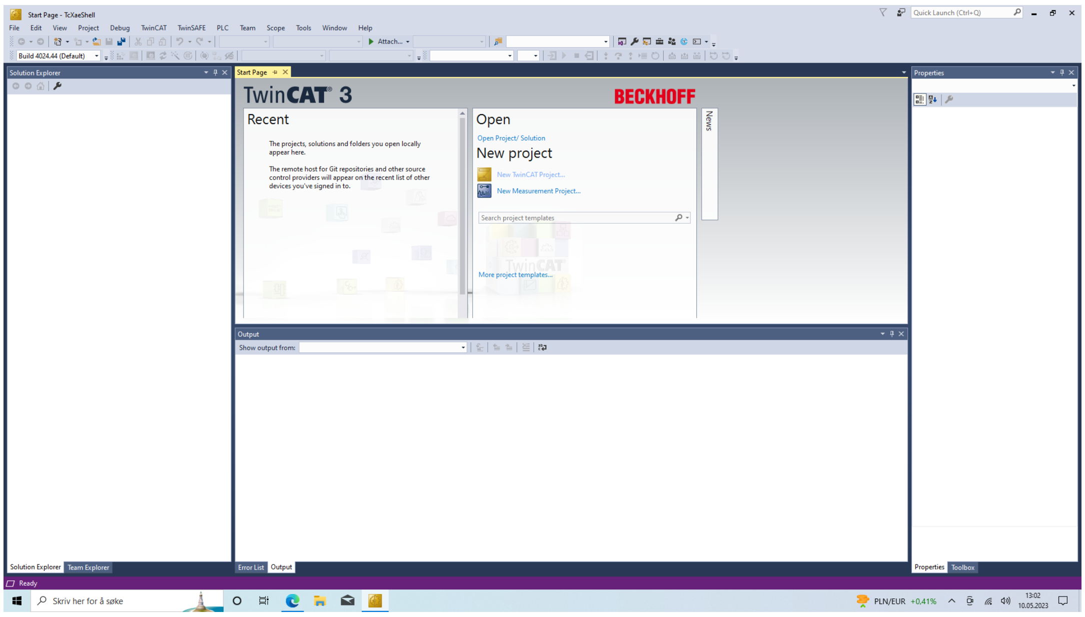
Type: Filmappe
Endringsdato: 31.03.2023 14:30
Størrelse: 20,4 MB
Mapper: FPGA
Filer: _AUTOEXEC.cmd, atlce400.dll, BeckhoffDesktop.jpg, ...

10 elementer 1 element er valgt

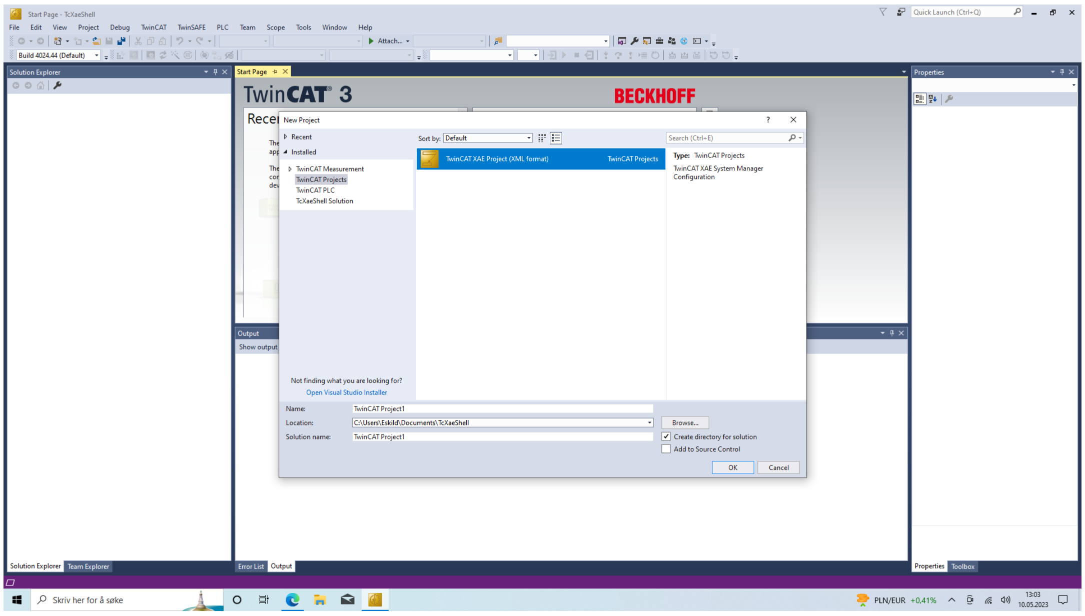
Figure 28: Copying the CAB file to PLC memory card. The computer is in Norwegian, the only relevant difference is that “This PC” is “Denne PCen” in the image.

With the CAB file now on the memory card, the next step involves installing it on the PLC. This process requires remoting into the PLC’s operating system, which can be accomplished using a Remote Desktop Protocol. An appropriate tool for this is “cerhost”, which can be downloaded from Beckhoff via [74]. It is important to ensure that the Remote Desktop feature is activated on the PLC.

Activation can be achieved by opening TwinCAT on the PC when it is connected to the PLC via an Ethernet cable, either directly or through a network switch. Within TwinCAT, create a new project, then select “TwinCAT Projects” and “TwinCAT XAE Project (XML format)”, as shown in Figure 29. Optionally, a name can be assigned to this project.


Start Page - TcXaeShell
File Edit View Project Debug TwinCAT TwinSAFE PLC Team Scope Tools Window Help
Attach...
Solution Explorer
Start Page
TwinCAT 3
BECKHOFF
Recent
The projects, solutions and folders you open locally appear here.
The remote host for Git repositories and other source control providers will appear on the recent list of other devices you've signed in to.
Open
Open Project/ Solution
New project
New TwinCAT Project...
New Measurement Project...
Search project templates
More project templates...
Output
Show output from:
Properties
Ready

(a)



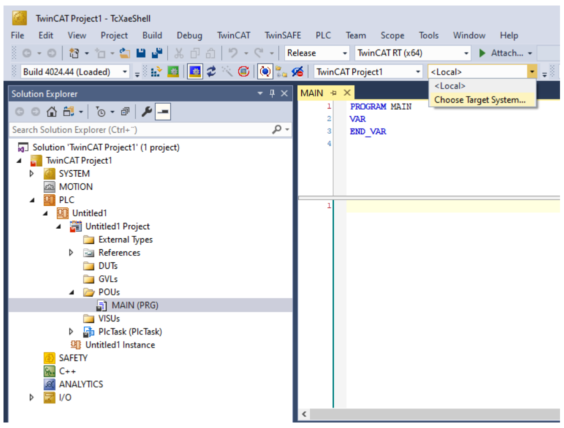
Name: TwinCAT Project1
Location: C:\Users\Eskild\Documents\TcXaeShell
Solution name: TwinCAT Project1
[✓] Create directory for solution
[ ] Add to Source Control

(b)

Figure 29: Creating a new project in TwinCAT

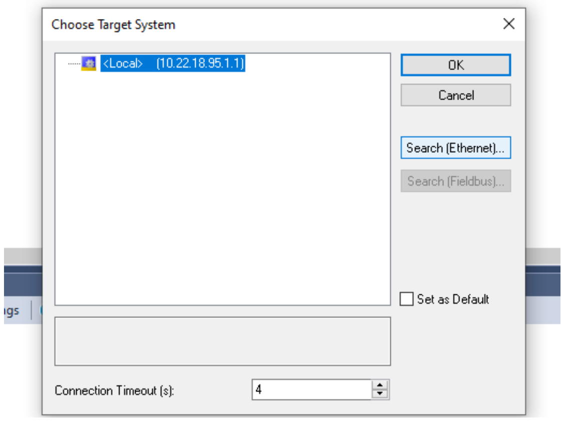
To connect to the PLC, navigate to the dropdown menu under “Help” and “Attach”, where it states “Local” or the name of the PLC you are potentially connected to, as shown in Figure 30a. A window will appear, where you select the “Search (Ethernet)” button, as illustrated in Figure 30b. If anything other than local is present, additional windows will prompt you to switch to local, which must be done.

Upon entering the new window, press the “Broadcast Search” button, followed by the “OK” button in the new window where network cards to be searched are selected. The PLC will then appear in the list, as the one used in this project did see Figure 30c. If multiple options appear, select the correct one and then hit the “Add Route” button.



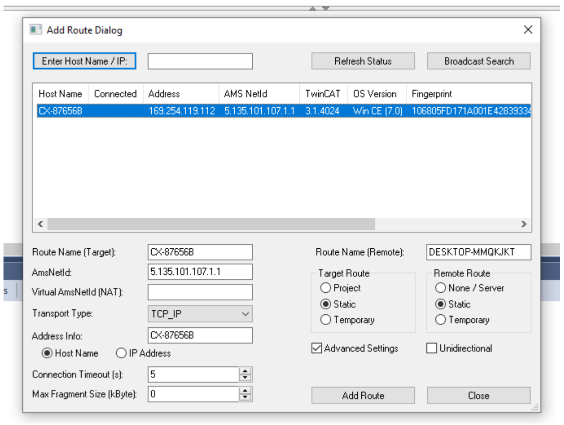
```
PROGRAM MAIN
VAR
END_VAR
```

(a)



Connection Timeout (s): 4

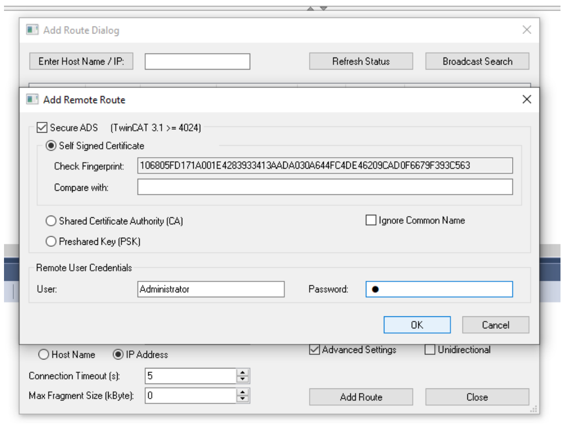
(b)



Host Name	Connected	Address	AMS NetId	TwinCAT	OS Version	Fingerprint
CX-87656B		169.254.119.112	5.135.101.107.1.1	3.1.4024	Win CE (7.0)	106805FD171A001E4283933

Route Name (Target): CX-87656B
AmsNetId: 5.135.101.107.1.1
Virtual AmsNetId (NAT):
Transport Type: TCP_IP
Address Info: CX-87656B
[✓ Host Name] IP Address
Connection Timeout (s): 5
Max Fragment Size (kByte): 0
Route Name (Remote): DESKTOP-MMQKJKT
Target Route: Project [✓ Static] Temporary
Remote Route: None / Server [✓ Static] Temporary
[✓ Advanced Settings] Unidirectional

(c)



[✓ Secure ADS] (TwinCAT 3.1 >= 4024)
[✓ Self Signed Certificate]
Check Fingerprint: 106805FD171A001E4283933413AADA030A644FC4DE46209CAD0F6679F393C563
Compare with:
Shared Certificate Authority (CA)
Ignore Common Name
Preshared Key (PSK)
Remote User Credentials
User: Administrator
Password: ●
Host Name [✓ IP Address]
[✓ Advanced Settings] Unidirectional
Connection Timeout (s): 5
Max Fragment Size (kByte): 0

(d)

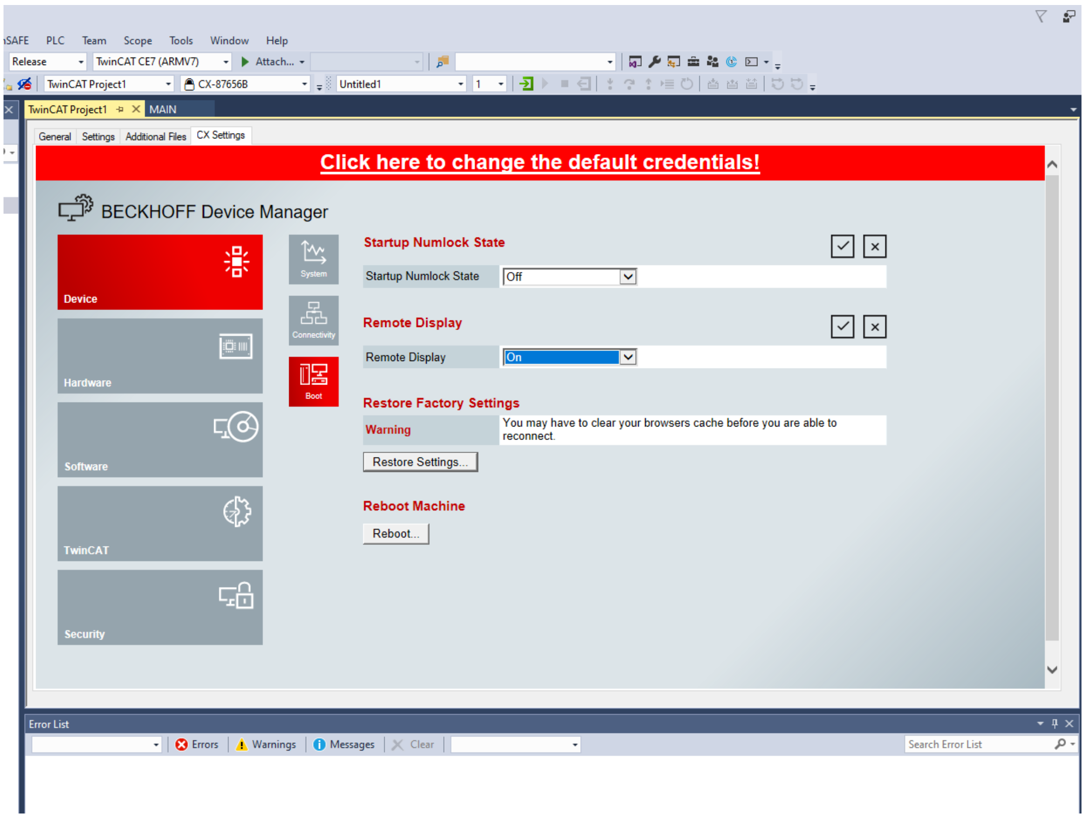
Figure 30: Depictions of the steps to connect TwinCAT XAE to the PLC

Ensure that "Secure ADS" is checked, and enter the username and password, as demonstrated in Figure 30d. The default credentials provided by Beckhoff are user: "Administrator" and password: "1". After pressing "OK" if there is a padlock icon next to the Host Name in the Connected column, the connection has been established. Additionally, note the IP address of the PLC listed in the Address column, as it will be required in subsequent stages. You can then press "Close", followed by "OK" in the next window. A new window will appear, asking whether you would like to switch platforms if the PLC has a different platform than the one previously selected. Click "Yes" to this prompt.

Next, navigate to the "System" section in the Solution Explorer, followed by the "CX-settings". At this stage, a login prompt will appear; ensure that you log in using an administrator account. The default Beckhoff credentials were used during the initial PLC connection in this project. Changing the default credentials within this setting is advisable, which it suggests with a big banner and can be seen in Figure 31.

Proceed to the "Device" section and then to "Boot". Ensure the "Remote Display" setting is switched to "On", and confirm by clicking the check box next to it, as depicted in Figure 31. A window will appear, indicating that the PC/PLC requires a restart. Confirm by clicking "OK". Once the system reboots and is operational again, it is prepared for a remote connection.

To initiate the Remote Desktop, execute the CERHOST.exe file located within the zip file accessible



Click here to change the default credentials!

BECKHOFF Device Manager

Startup Numlock State

Startup Numlock State	Off

Remote Display

Remote Display	On

Restore Factory Settings

Warning	You may have to clear your browsers cache before you are able to reconnect.

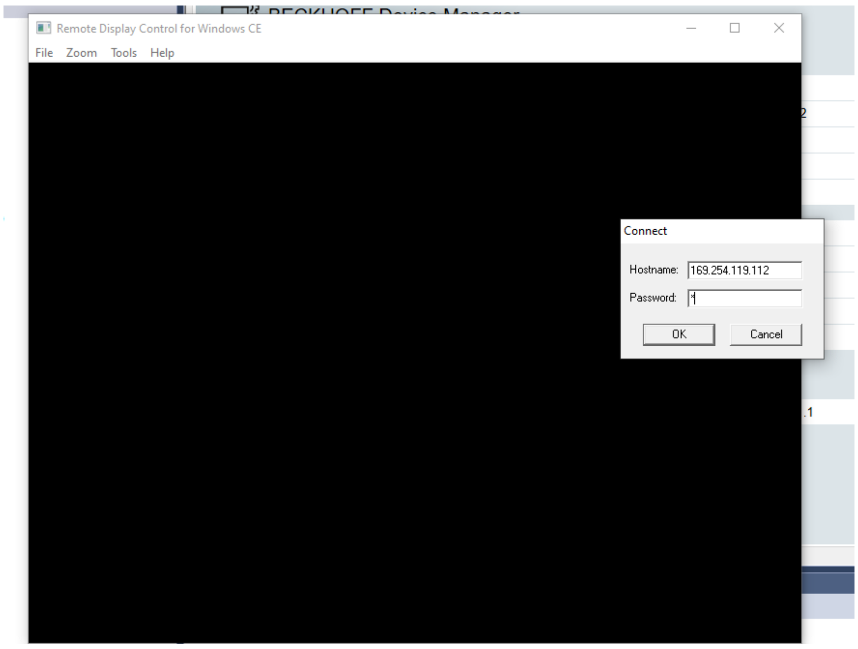
Restore Settings...

Reboot Machine

Reboot...

Figure 31: Activating Remote Display

via [74]. Once the program launches, establish a connection by navigating to “File” and then “Connect”. A window will appear, prompting the input of the Hostname and Password. The password should match the one used for the administrator account. If a connection cannot be established using the Hostname, the IP address may be utilized instead, as shown in Figure 32.



Connect

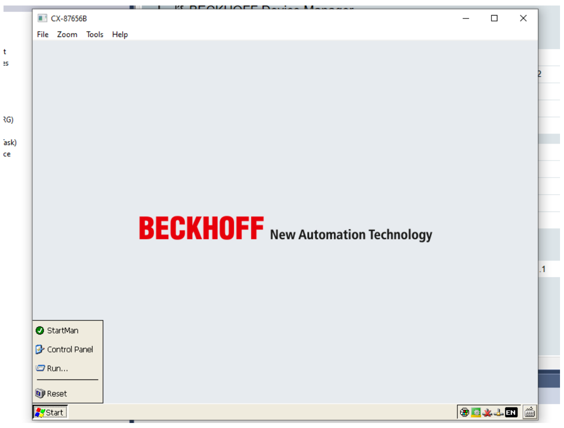
Hostname: 169.254.119.112

Password:

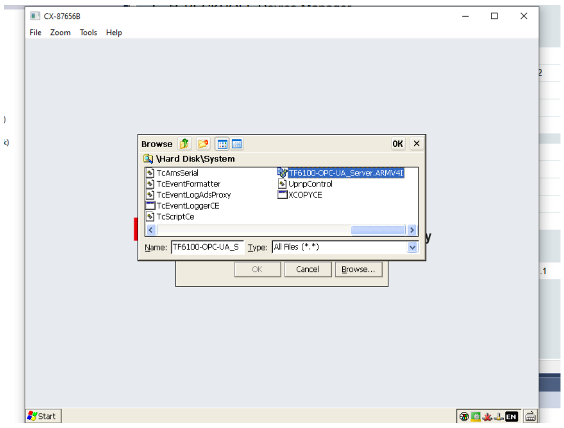
Figure 32: Connecting to the PLC with CERHOST.

Once the connection has been established, access the start menu and select “Run...”, as illustrated in Figure 33a. A new window will appear, in which the “Browse” button should be selected. Another window will then open, prompting the location of the saved CAB file. Navigate to the “Hard Disk” folder, within which the “System” directory in which we saved the file.

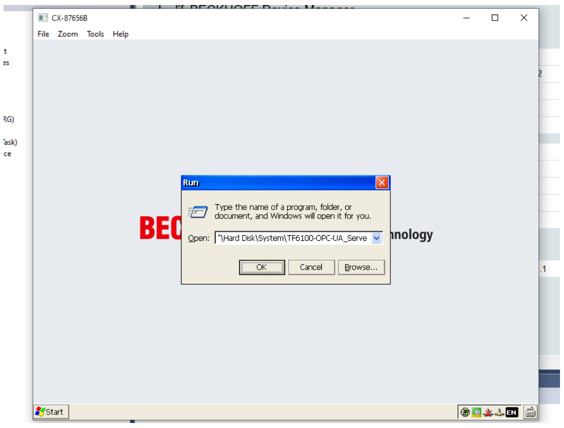
To locate the CAB file, the setting needs to be adjusted from displaying only .exe files to all files. This can be done by selecting “All Files” from the dropdown menu for type. The file can then be found by scrolling to the right, as shown in Figure 33b. Upon confirming the correct file selection, select “OK” to return to the previous window.



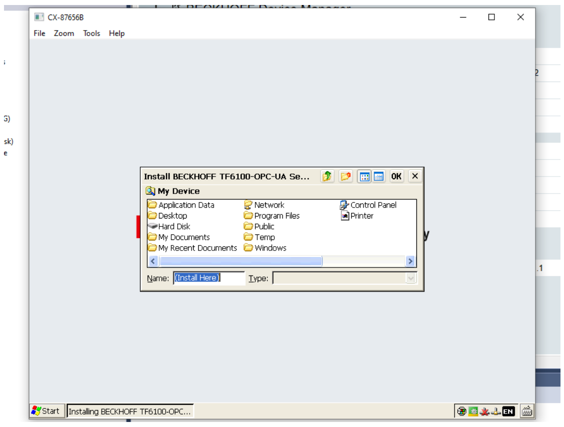
(a)



(b)


CX-87656B
File Zoom Tools Help
Run
Type the name of a program, folder, or document, and Windows will open it for you.
Open: \Hard Disk\System\TF6100-OPC-UA_Serve
OK
Cancel
Browse...
Start

(c)



(d)

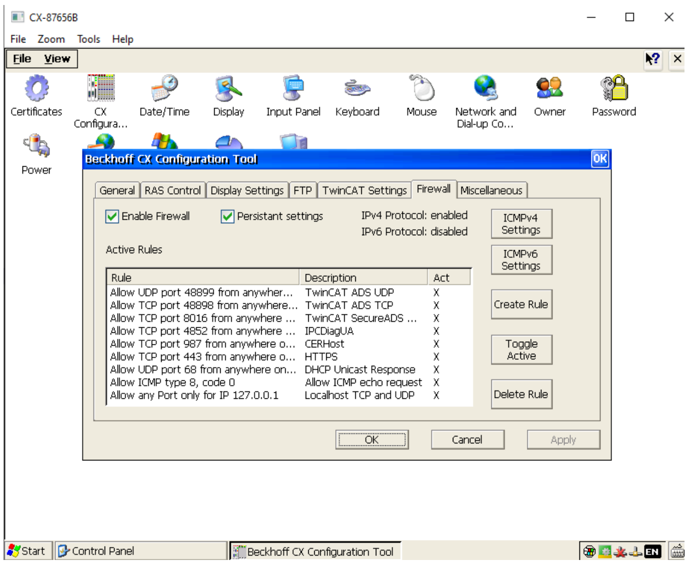
Figure 33: Steps depicting the installation of OPC UA on the PLC.

Ensure the correct file is selected, then press “OK” as shown in Figure 33c. This action will prompt the opening of a new window similar to the one used for locating the CAB file, but this time for choosing the installation location, as shown in Figure 33d.

However, with this version of TwinCAT, the chosen location is inconsequential, as the installation will automatically occur in the appropriate location, and the CAB file will be removed upon completion. Thus, press the “OK” button in the top right corner.

With OPC UA now installed, it is crucial to open the network port that will be used for this communication. If this step is neglected, the connection might be blocked. Begin by selecting the start menu and choosing “Control Panel”. A full-screen window with multiple icons will appear. Double-click the “CX Configuration” icon, prompting a new window to appear. Select the “Firewall" tab in this window, as demonstrated in Figure 34a. Proceed by clicking the "Create Rule" button.

In the next window, the port will be opened. Check the "Port" option in the "Type" section. Then enter the minimum and maximum ports to be opened. As port 4840 is commonly used, only this port is opened in this project. Enter a suitable description for the rule and click "OK", as shown in Figure 34b. Returning to the previous window, verify that the new rule has been added to the list, and click "OK".



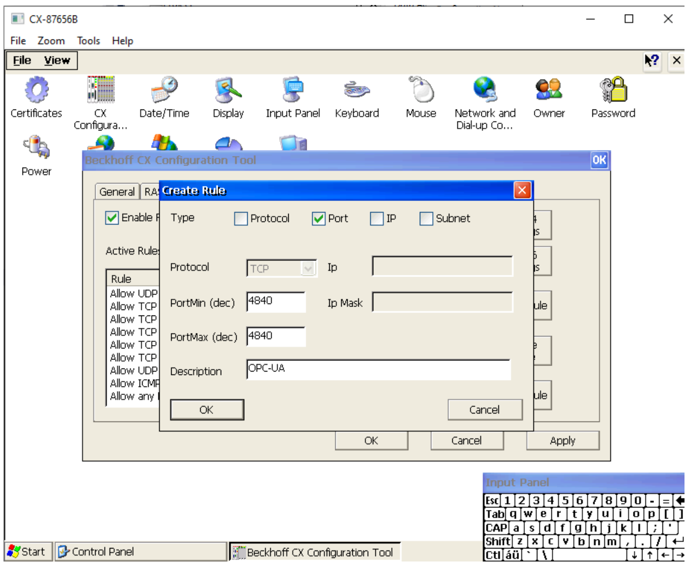
[✓] Enable Firewall [✓] Persistant settings

IPv4 Protocol: enabled
IPv6 Protocol: disabled

Active Rules

Rule	Description	Act
Allow UDP port 48899 from anywher...	TwinCAT ADS UDP	X
Allow TCP port 48898 from anywhere...	TwinCAT ADS TCP	X
Allow TCP port 8016 from anywhere ...	TwinCAT SecureADS ...	X
Allow TCP port 4852 from anywhere ...	IPCDiagUA	X
Allow TCP port 987 from anywhere o...	CERHost	X
Allow TCP port 443 from anywhere o...	HTTPS	X
Allow UDP port 68 from anywhere on...	DHCP Unicast Response	X
Allow ICMP type 8, code 0	Allow ICMP echo request	X
Allow any Port only for IP 127.0.0.1	Localhost TCP and UDP	X

(a)



Type [ ] Protocol [✓ Port] [ ] IP [ ] Subnet

Protocol: TCP
Ip:
PortMin (dec): 4840
Ip Mask:
PortMax (dec): 4840
Description: OPC-UA

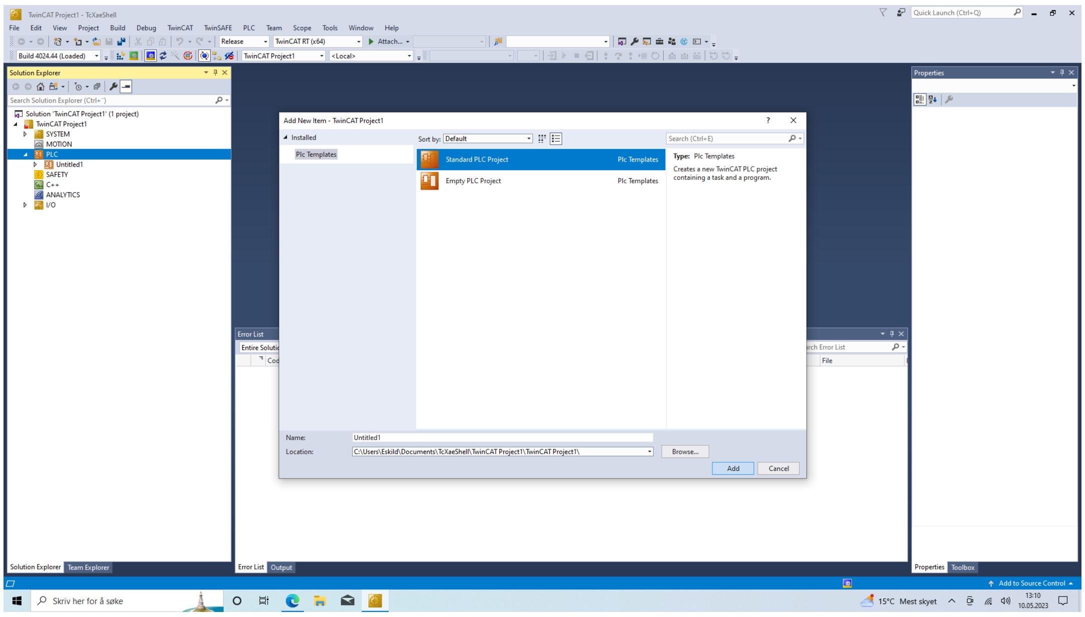
(b)

Figure 34: Steps depicting the opening of the port used for OPC UA.

At this point, the PLC is ready for communication via OPC UA and, thus, is prepared for programming.

5 Programming the PLC

To commence programming the PLC, a new project needs to be created. This can be accomplished in the same way as demonstrated in Section 4.2.2. A PLC project can be initiated by right-clicking the "PLC" option in the Solution Explorer, located on the left side of the TwinCAT window. Next, select "Add new item..." from the appearing menu. A new window will open; here, choose "Standard PLC Project", assign a suitable name, and click the "Add" button, as shown in Figure 35.


Add New Item - TwinCAT Project1

Standard PLC Project — Plc Templates

Empty PLC Project — Plc Templates

Type: Plc Templates

Creates a new TwinCAT PLC project containing a task and a program.

Name: Untitled1

Location: C:\Users\Eskild\Documents\TcXaeShell\TwinCAT Project1\TwinCAT Project1\

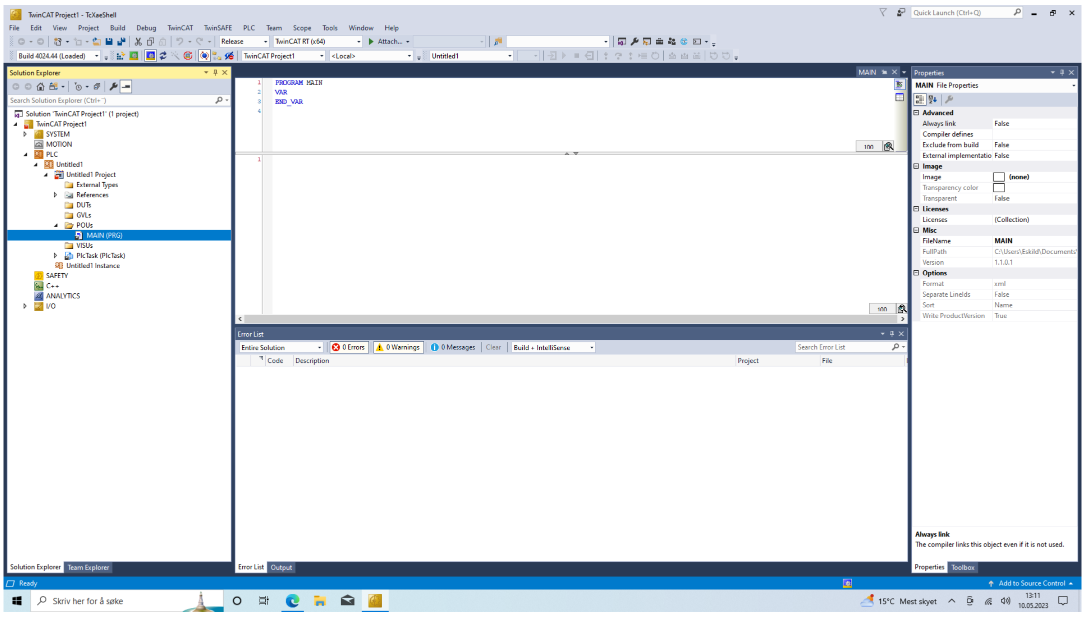
Figure 35: Creating å PLC project.

When programming the PLC, it is essential to toggle from the "Local" system to the PLC. This transition ensures that the programming activities are directly applied to the PLC. Alternatively, one could prototype and execute the project locally when the "Local" system is selected. The procedure for altering the system from "Local" to the PLC is outlined in Section 4.2.2. For those who have previously engaged in this process, the PLC might already be listed in the initial window, simplifying the procedure by allowing direct selection.

Programming the PLC within TwinCAT is performed within the previously established PLC project. Navigating the POUs directory reveals a file named MAIN, which serves as the default location for the program's construction, as depicted in Figure 36. As stipulated in Section 2.4.2, the programming approach conforms to the IEC 61131-3 standard, prescribing the use of variables, data types, functions, function blocks, and so forth in the PLC programming process.

The data types available for utilization within PLC programming are specified in Table 10 of Section 6.4.2.1 in the IEC 61131-3 standard [8]. For instance, the Byte data type is defined as an 8-bit string, whereas a Word constitutes a 16-bit string. Understanding these definitions is paramount, as bits, bytes, and words are used alongside Booleans and arrays of bytes to illustrate the reasoning behind the specific programming and configuration of the PLC.

For a more intricate comprehension of PLC programming, it is recommended to delve into resources such as the IEC 61131-3 standard [8], Beckhoff's guidelines [75], or other relevant literature. These comprehensive resources offer better insight into the intricacies of PLC programming, thereby



```
PROGRAM MAIN
VAR
END_VAR
```

MAIN File Properties

Advanced	
Always link	False
Compiler defines	
Exclude from build	False
External implementatio	False
Image	
Image	(none)
Transparency color	
Transparent	False
Licenses	
Licenses	(Collection)
Misc	
FileName	MAIN
FullPath	C:\Users\Eskild\Documents\
Version	1.1.0.1
Options	
Format	xml
Separate LineIds	False
Sort	Name
Write ProductVersion	True

Always link
The compiler links this object even if it is not used.

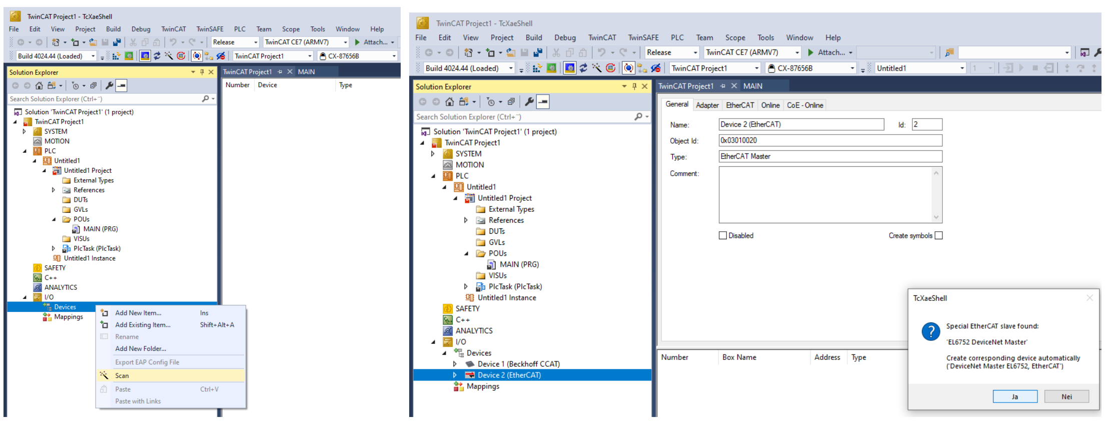
Figure 36: The PLC project structure in TwinCAT3. Depicting where the main program is located.

enabling a broader understanding than what will be elaborated upon in the remainder of this thesis.

5.1 Connecting to the Fronius Welding Machine

To connect to the Fronius Welding Machine, it is necessary to establish a connection to the PLC in TwinCAT, and the PLC's EL6752 module must be connected to the Welding Machine with the DeviceNet cable. Ensure that TwinCAT, and hence the PLC, is in the configuration mode. This can be verified by noting that the cog icon on the blue line at the bottom of TwinCAT XAE is blue and that the blue version, not the green one, is highlighted in the toolbar at the top.

In the Solution Explorer, select the I/O module, and beneath it, there's an option labelled "Devices". Right-click this option and select "Scan", as shown in Figure 37a. A dialogue box will appear - click "OK", then another window will appear. This window displays all the devices the PLC can scan directly. Leave all options checked per the default settings and click "OK". Another dialogue box will appear - click "Yes". A new dialogue box identifying the EL6752 module will appear, as shown in Figure 37b. Click "Yes".



Name: Device 2 (EtherCAT) Id: 2
Object Id: 0x03010020
Type: EtherCAT Master

TcXaeShell

Special EtherCAT slave found:
'EL6752 DeviceNet Master'
Create corresponding device automatically
('DeviceNet Master EL6752, EtherCAT')

Ja Nei

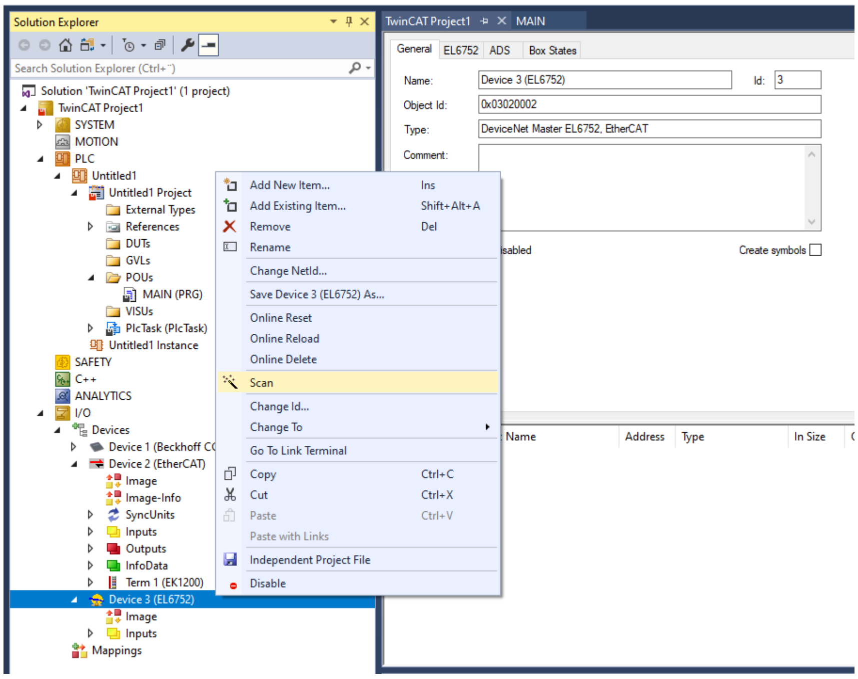
(a) (b)

Figure 37: Scanning for IO in TwinCAT.

Subsequently, a dialogue box will appear where the Baudrate for DeviceNet is selected. Given the setup of the Fronius Welding Machine, choose a 500k Baudrate, and click "OK". In the next window, click "Yes". In the Solution Explorer, one can observe that EL6752 has become a separate device. Thus, the system is ready to scan for DeviceNet I/O.

5.1.1 DeviceNet I/O

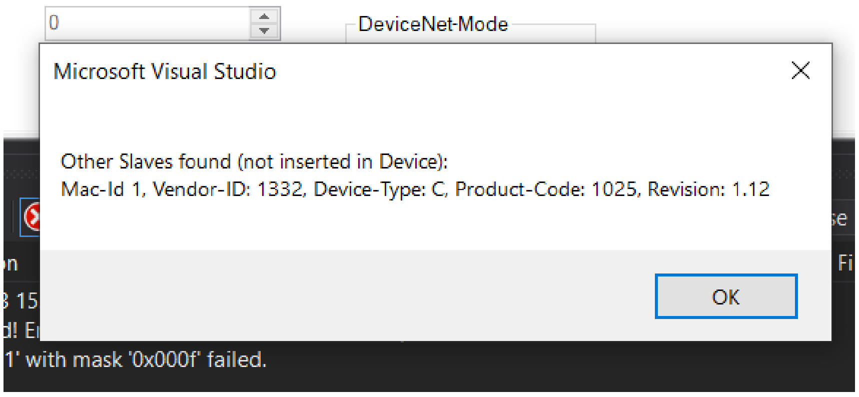
Analogous to the previous procedure where we scanned for devices connected to the PLC, we will now scan for devices connected to the DeviceNet master, EL6752. Under the I/O section in Devices, right-click "Device X (EL6752)", where X refers to the specific device number (for instance, in Figure 38, it is Device 3).



Name: Device 3 (EL6752) Id: 3
Object Id: 0x03020002
Type: DeviceNet Master EL6752, EtherCAT

Figure 38: Scanning for DeviceNet devices.

The system scans all 63 MAC-IDs it can directly communicate with; however, upon completion, it does not find any slaves it can control. Instead, we receive an error message indicating that "Other slaves found (not inserted in Device):" as shown in Figure 39. This is due to the need for an EDS file that dictates how to manage the DeviceNet unit. Furthermore, the error message reveals the MAC-ID of the device, as well as the information contained in the EDS file, such as the Vendor-ID, Device-Type, Product-Code, and revision; these can be viewed in Figure 39. The required file can be obtained from Fronius [64]; however, identifying the correct EDS file is imperative as there are several variations.



Other Slaves found (not inserted in Device):
Mac-Id 1, Vendor-ID: 1332, Device-Type: C, Product-Code: 1025, Revision: 1.12

1' with mask '0x000f' failed.

Figure 39: Other slaves connected message window when searching for DeviceNet slaves.

After downloading the zip file containing the bus systems from Fronius [64], one can navigate to the "DeviceNet - EDS" folder. Here, multiple options are presented. By examining the bus card plugged into the Robot Interface in the Fronius Welding Machine, it is discerned that the correct one is labelled "HMS (Anybus-CC M40)", as depicted in Figure 40. Nevertheless, with numerous choices within this folder, determining the appropriate selection is not immediately evident. One strategy involves using software to open and read the information in the EDS files, thereby identifying the one that corresponds to the details on the bus card and the error message in Figure 40 and Figure 39 respectively.

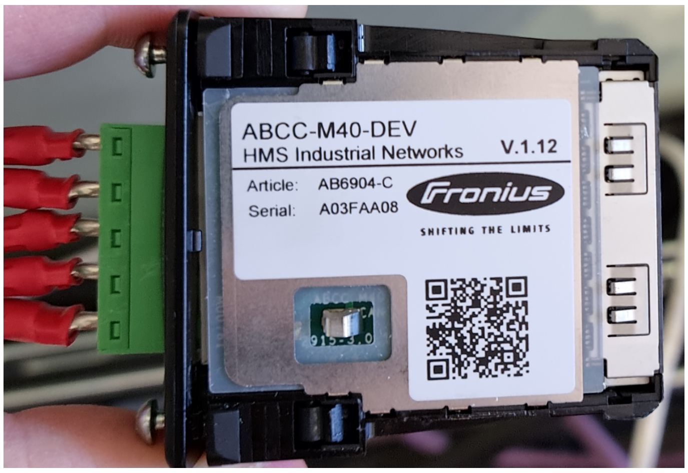
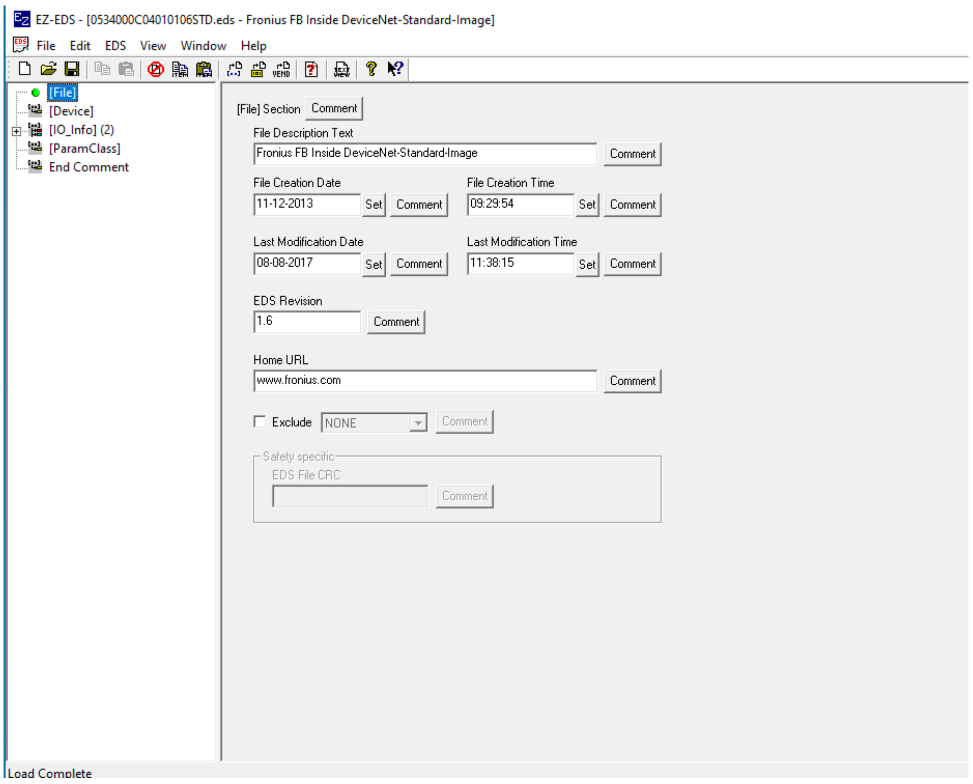


Figure 40: The DeviceNet Bus Card, which is used with the Robot Interface with the Fronius Welding Machine.

One such program, EZ-EDS, developed by the ODVA, can be downloaded from [76]. For the versions utilized in this thesis, the suitable file is located in the "RI FB INSIDE MOD CC-M40 DeviceNet", "EDS", "1. Standard Image", "Previous versions", "V1.6" folder. Refer to Figure 41 where it is demonstrated that this EDS file contains the same information as found in Figure 39 and Figure 40.


EZ-EDS - [0534000C04010106STD.eds - Fronius FB Inside DeviceNet-Standard-Image]

[File] Section

File Description Text: Fronius FB Inside DeviceNet-Standard-Image

File Creation Date: 11-12-2013

File Creation Time: 09:29:54

Last Modification Date: 08-08-2017

Last Modification Time: 11:38:15

EDS Revision: 1.6

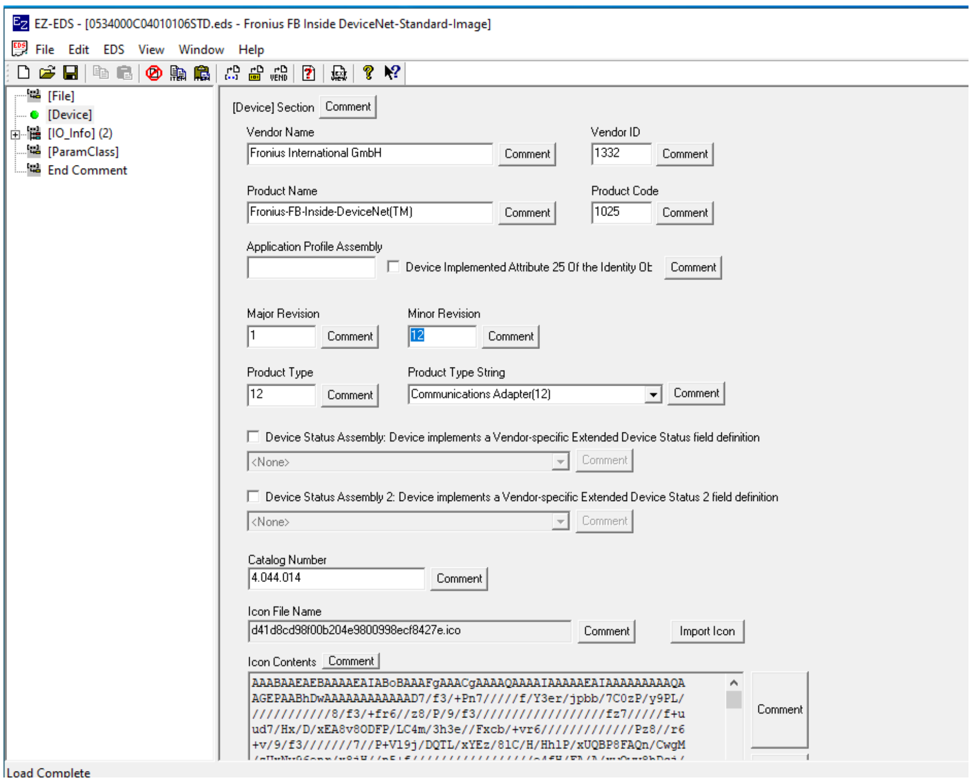
Home URL: www.fronius.com

☐ Exclude: NONE

Safety specific: EDS File CRC

Load Complete

(a)


EZ-EDS - [0534000C04010106STD.eds - Fronius FB Inside DeviceNet-Standard-Image]

[Device] Section

Vendor Name: Fronius International GmbH

Vendor ID: 1332

Product Name: Fronius-FB-Inside-DeviceNet(TM)

Product Code: 1025

Application Profile Assembly:

☐ Device Implemented Attribute 25 Of the Identity Ob

Major Revision: 1

Minor Revision: 12

Product Type: 12

Product Type String: Communications Adapter(12)

☐ Device Status Assembly: Device implements a Vendor-specific Extended Device Status field definition: <None>

☐ Device Status Assembly 2: Device implements a Vendor-specific Extended Device Status 2 field definition: <None>

Catalog Number: 4.044.014

Icon File Name: d41d8cd98f00b204e9800998ecf8427e.ico

Icon Contents:

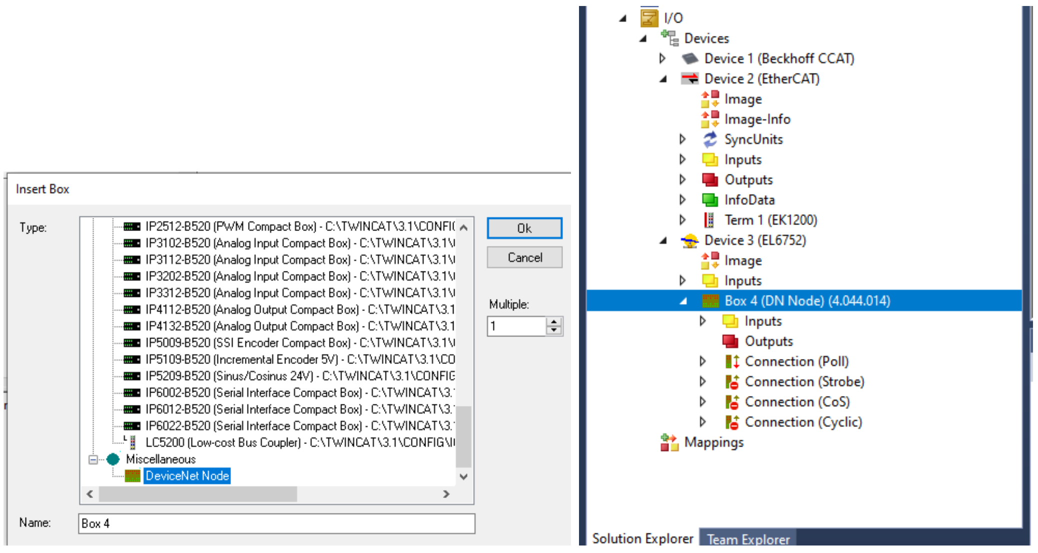
```
AAABAAEAEBAAAAEAIABoBAAAFgAAACgAAAAQAAAAIAAAAAEAIAAAAAAAAAQA
AGEPAABhDwAAAAAAAAAAAAD7/f3/+Pn7/////f/Y3er/jpbb/7C0zP/y9PL/
////////8/f3/+fr6//z8/P/9/f3////////////////////fz7/////f+u
ud7/Hx/D/xEA8v8ODFP/LC4m/3h3e//Fxcb/+vr6////////////Pz8//r6
+v/9/f3///////7//P+Vl9j/DQTL/xYEz/81C/H/HhlP/xUQBP8FAQn/CwgM
```

Load Complete

(b)

Figure 41: The EDS file opened with EZ-EDS, depicts the information in the EDS file, which matches the setup used in this thesis.

To incorporate this EDS file, right-click on "Device X (EL6752)" again, but this time select "Add New Item...". A new window will appear; scroll down to the bottom of the window, and under "Miscellaneous" choose the sole option "DeviceNet Node" and click "OK" as shown in Figure 42a. This action prompts the opening of a Windows file explorer window where one navigates to the correct EDS file, selects it, and click "Open". The file will now be added as a Box under "Device X (EL6752)", as illustrated in Figure 42b.



Insert Box

Type:

- IP2512-B520 (PWM Compact Box) - C:\TWINCAT\3.1\CONFI(
- IP3102-B520 (Analog Input Compact Box) - C:\TWINCAT\3.1\I
- IP3112-B520 (Analog Input Compact Box) - C:\TWINCAT\3.1\I
- IP3202-B520 (Analog Input Compact Box) - C:\TWINCAT\3.1\I
- IP3312-B520 (Analog Input Compact Box) - C:\TWINCAT\3.1\I
- IP4112-B520 (Analog Output Compact Box) - C:\TWINCAT\3.1
- IP4132-B520 (Analog Output Compact Box) - C:\TWINCAT\3.1
- IP5009-B520 (SSI Encoder Compact Box) - C:\TWINCAT\3.1\I
- IP5109-B520 (Incremental Encoder 5V) - C:\TWINCAT\3.1\CO
- IP5209-B520 (Sinus/Cosinus 24V) - C:\TWINCAT\3.1\CONFIG
- IP6002-B520 (Serial Interface Compact Box) - C:\TWINCAT\3.
- IP6012-B520 (Serial Interface Compact Box) - C:\TWINCAT\3.
- IP6022-B520 (Serial Interface Compact Box) - C:\TWINCAT\3.
- LC5200 (Low-cost Bus Coupler) - C:\TWINCAT\3.1\CONFIG\I
- Miscellaneous
  - DeviceNet Node

Multiple: 1

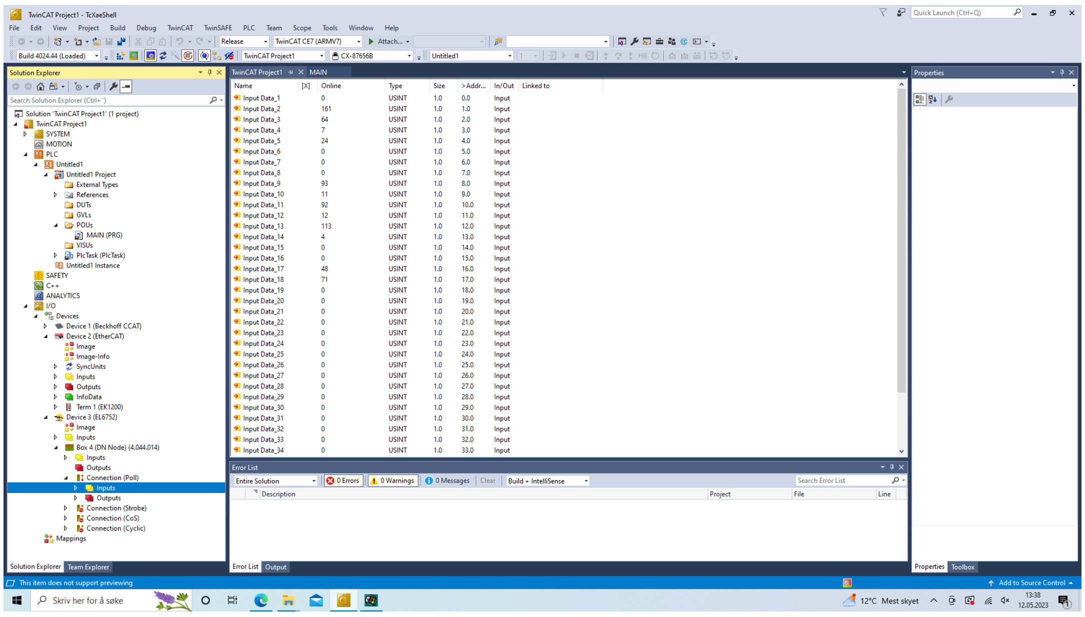
Name: Box 4

- I/O
  - Devices
    - Device 1 (Beckhoff CCAT)
    - Device 2 (EtherCAT)
      - Image
      - Image-Info
      - SyncUnits
      - Inputs
      - Outputs
      - InfoData
      - Term 1 (EK1200)
    - Device 3 (EL6752)
      - Image
      - Inputs
      - Box 4 (DN Node) (4.044.014)
        - Inputs
        - Outputs
        - Connection (Poll)
        - Connection (Strobe)
        - Connection (CoS)
        - Connection (Cyclic)
  - Mappings

(a) (b)

Figure 42: Adding in the DeviceNet node with EDS file as a box under the EL6752 Device.

Scan "Device X (EL6752)" again for an update to correctly establish the connection. If all steps are executed correctly, no error messages will appear. If the welding machine is connected and switched on, navigate to Inputs under "Connection (Poll)" to observe that the variables have updated based on the welding machine's values, as demonstrated in Figure 43.



Name	[X]	Online	Type	Size	>Addr...	In/Out	Linked to
Input Data_1		0	USINT	1.0	0.0	Input	
Input Data_2		161	USINT	1.0	1.0	Input	
Input Data_3		64	USINT	1.0	2.0	Input	
Input Data_4		7	USINT	1.0	3.0	Input	
Input Data_5		24	USINT	1.0	4.0	Input	
Input Data_6		0	USINT	1.0	5.0	Input	
Input Data_7		0	USINT	1.0	6.0	Input	
Input Data_8		0	USINT	1.0	7.0	Input	
Input Data_9		93	USINT	1.0	8.0	Input	
Input Data_10		11	USINT	1.0	9.0	Input	
Input Data_11		92	USINT	1.0	10.0	Input	
Input Data_12		12	USINT	1.0	11.0	Input	
Input Data_13		113	USINT	1.0	12.0	Input	
Input Data_14		4	USINT	1.0	13.0	Input	
Input Data_15		0	USINT	1.0	14.0	Input	
Input Data_16		0	USINT	1.0	15.0	Input	
Input Data_17		48	USINT	1.0	16.0	Input	
Input Data_18		71	USINT	1.0	17.0	Input	
Input Data_19		0	USINT	1.0	18.0	Input	
Input Data_20		0	USINT	1.0	19.0	Input	
Input Data_21		0	USINT	1.0	20.0	Input	
Input Data_22		0	USINT	1.0	21.0	Input	
Input Data_23		0	USINT	1.0	22.0	Input	
Input Data_24		0	USINT	1.0	23.0	Input	
Input Data_25		0	USINT	1.0	24.0	Input	
Input Data_26		0	USINT	1.0	25.0	Input	
Input Data_27		0	USINT	1.0	26.0	Input	
Input Data_28		0	USINT	1.0	27.0	Input	
Input Data_29		0	USINT	1.0	28.0	Input	
Input Data_30		0	USINT	1.0	29.0	Input	
Input Data_31		0	USINT	1.0	30.0	Input	
Input Data_32		0	USINT	1.0	31.0	Input	
Input Data_33		0	USINT	1.0	32.0	Input	
Input Data_34		0	USINT	1.0	33.0	Input	

Error List: Entire Solution · 0 Errors · 0 Warnings · 0 Messages · Build + IntelliSense

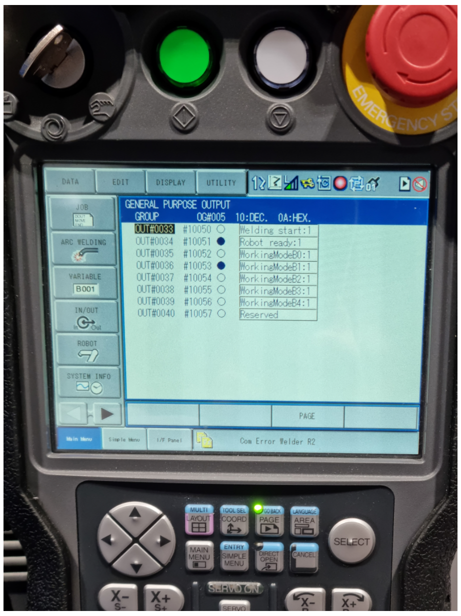
Figure 43: The inputs in TwinCat has been updated based on the Fronius Welding Machine.

Upon examining these inputs and the corresponding outputs, one can identify 40 “Input Data_XX” and “Output Data_XX” elements, where XX ranges from 1 to 40. If one looks at the EDS file under IO_Info, it is evident that 40 Bytes are allocated for both Inputs and Outputs, and they are represented as Unsigned short integers (USINT) in TwinCAT. However, when taken alone, this information may not provide a clear understanding. Therefore, a form of mapping is required to make sense of these numbers.

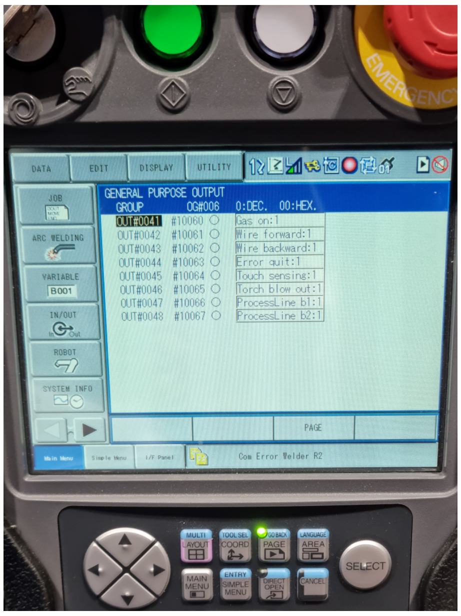
5.1.2 Fronius Welder I/O Mapping

To determine the mapping for these values, referring to Fronius’s DeviceNet documentation might be beneficial. This could lead to the discovery of the DeviceNet Operation Instructions [77], which provides various mappings for DeviceNet and some insights into its functioning. However, it quickly becomes evident that these do not align with our findings, and despite the information being presented in a different format, it is not possible to discern a correlation between the formats. Certain elements correspond, but others are entirely distinct. Although considerable relevant information exists here, the mapping must be found elsewhere.

Consequently, if one accesses the Yaskawa robot controller, to which the Fronius Welding Machine was initially connected, one can discover the mapping of the IO to which it is attached. From here, the IO of the Fronius device can be tracked down; refer to Figure 44 for the first two bytes of output, which are identical to the first two bytes of output in the poll from the DeviceNet box created in TwinCAT.


```
GENERAL PURPOSE OUTPUT
  GROUP        OG#005  10:DEC.  0A:HEX.
  OUT#0033  #10050 ○   Welding start:1
  OUT#0034  #10051 ●   Robot ready:1
  OUT#0035  #10052 ○   WorkingModeB0:1
  OUT#0036  #10053 ●   WorkingModeB1:1
  OUT#0037  #10054 ○   WorkingModeB2:1
  OUT#0038  #10055 ○   WorkingModeB3:1
  OUT#0039  #10056 ○   WorkingModeB4:1
  OUT#0040  #10057 ○   Reserved
```

(a)


```
GENERAL PURPOSE OUTPUT
  GROUP        OG#006  0:DEC.  00:HEX.
  OUT#0041  #10060 ○   Gas on:1
  OUT#0042  #10061 ○   Wire forward:1
  OUT#0043  #10062 ○   Wire backward:1
  OUT#0044  #10063 ○   Error quit:1
  OUT#0045  #10064 ○   Touch sensing:1
  OUT#0046  #10065 ○   Torch blow out:1
  OUT#0047  #10066 ○   ProcessLine b1:1
  OUT#0048  #10067 ○   ProcessLine b2:1
```

(b)

Figure 44: Images of the Yaskawa mapping. It is the two first bytes of the Output mapping

The logic behind locating the mapping here stems from the fact that the Yaskawa robot controller operates the welding apparatus in its default configuration. Hence, it is logical to think that this mapping is accurate, a proposition substantiated through testing and by carefully examining the Operation Instructions for the Bus Card [78]. A notable distinction between the tables found in [78] and those featured in this thesis lies in their indexing scheme. The tables in [78] utilize zero-based indexing, whereas those in this thesis adopt a one-based indexing strategy to align with the TwinCAT environment. The first ten bytes of the output are presented in Table 2.

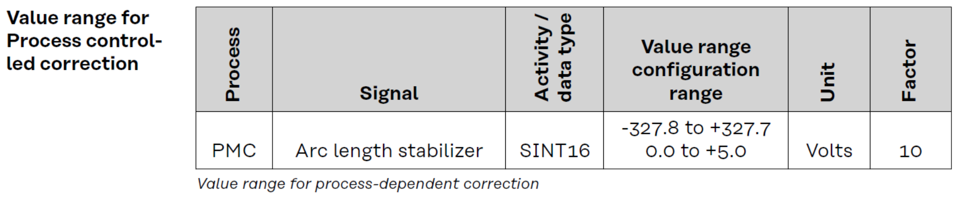
Table 2: The Output mapping of the ten first Bytes for the Fronius Welding Machine, found in the Yaskawa Robot controller.

Odd Bytes			Even Bytes		
Byte	Bit	Description	Byte	Bit	Description
1	1	Welding start	2	1	Gas On
	2	Robot ready		2	Wire forward
	3	WorkingModeB0		3	Wire backward
	4	WorkingModeB1		4	Error quit
	5	WorkingModeB2		5	Touch sensing
	6	WorkingModeB3		6	Torch blow out
	7	WorkingModeB4		7	ProcessLine b1
	8	Reserved		8	ProcessLine b2
3	1	Welding Sim	4	1	Reserved
	2	SynchroPulseOn		2	Teach mode
	3	Reserved		3	Reserved
	4	Reserved		4	ActiveHeatCont
	5	Reserved		5	Reserved
	6	Reserved		6	Reserved
	7	Wire break on		7	Reserved
	8	Torch Xchange		8	Reserved
5	1	Twin Mode b1	6	1	Reserved
	2	Twin Mode b2		2	Reserved
	3	Reserved		3	Reserved
	4	Reserved		4	Reserved
	5	Reserved		5	Reserved
	6	Document Mode		6	Reserved
	7	Reserved		7	Reserved
	8	Reserved		8	DisableProcesCo
7	1	Reserved	8	1	Ext Input 1
	2	Reserved		2	Ext Input 2
	3	Reserved		3	Ext Input 3
	4	Reserved		4	Ext Input 4
	5	Reserved		5	Ext Input 5
	6	Reserved		6	Ext Input 6
	7	Reserved		7	Ext Input 7
	8	Reserved		8	Ext Input 8
9	1	Job nr	10	1	Job nr
	2	Low Byte		2	High Byte
	3			3	
	4			4	
	5			5	
	6			6	
	7			7	
	8			8	

These constitute the outputs from the perspectives of the Robot controller and the PLC; however, from the vantage point of the Fronius welding machine, these will be considered as inputs. Only the first ten are displayed because they are the most relevant, given that they contain bits and encompass the format of the first word. The first word consists of the Low Byte, followed immediately by the High Byte in the list. Refer to Table 3 for a complete table of all words, and Figure 45 for supplementary information. For a detailed understanding of the Working Mode bits in the first byte, as indicated in Table 2, refer to Table 4, and is derived from [78].

Table 3: Low Byte values for the corresponding Output Words. The High Byte for each Word can be calculated by adding one to the Low Byte. The factor indicates the multiplier needed to convert real-world values to the format recognized by the welding machine. For instance, a desired welding speed of 10.0 cm/min must be multiplied by a factor of 10 to achieve the corresponding input value for the welding machine. These values pertain to the output of the PLC.

Low Byte	Word	Description
9	Job Number	0 to 1000, factor 1
11	Wire feed speed	-327.68 to 327.67 [m/min], factor 100
13	Arc Length Correction	-10.0 to 10.0 [steps], factor 10
15	Pulse/dyn Correction	-10.0 to 10.0 [steps] or 0.0 to 10.0, factor 10
17	Wire Retract Correction	0.0 to 10.0 [steps], factor 10
19	Welding Speed	0.0 to 1000.0 [cm/min], factor 10
21	Process Controlled Correction	See Figure 45 from [78] for more info
39	Seam Number	0 to 65,535


Value range for Process control-led correction

Process	Signal	Activity / data type	Value range configuration range	Unit	Factor
PMC	Arc length stabilizer	SINT16	-327.8 to +327.7 0.0 to +5.0	Volts	10

Value range for process-dependent correction

Figure 45: Value range for Process controlled correction from [78].

Table 4: Description of Bit Settings of WorkingMode

Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Description
0	0	0	0	0	Internal welding parameter selection
0	0	0	0	1	Special 2-step mode characteristics
0	0	0	1	0	Job mode
0	1	0	0	0	2-step mode characteristics
0	1	0	0	1	2-Step manual mode
1	0	0	0	1	Stop cooling pump

By investigating the input mapping on the Yaskawa robot controller, we identify a correspondence with the first ten bytes for input, as demonstrated in Table 5. This pattern extends to the words depicted in Table 6. For a comprehensive mapping that includes all bits across the 80 bytes (40 inputs and 40 outputs), Appendix B should be consulted. Detailed elaboration of the mapping can be found in [78].

Now that we have established the mapping of what the IO with Fronius means, we can utilize this information in the program that will be run on the PLC. This process will be outlined in Section 5.3. However, once we have created variables meant to represent the IO in the program, they must be mapped to the IO. This is accomplished by navigating to the "-PLC project name-Instance" within the PLC project in the Solution Explorer. In this program, which we will revisit, an array consisting of 40 bytes for inputs and outputs has been established to reflect the structure of the IO from the DeviceNet master. Navigate to the first byte in the output array as shown in Figure 46. Click on the "Linked to.." button and select the corresponding byte from the IO, as demonstrated in Figure 47, and press ok. Proceed to do this for the remainder of the bytes.

Table 5: The Input mapping of the ten first Bytes for the Fronius Welding Machine, found in the Yaskawa Robot controller.

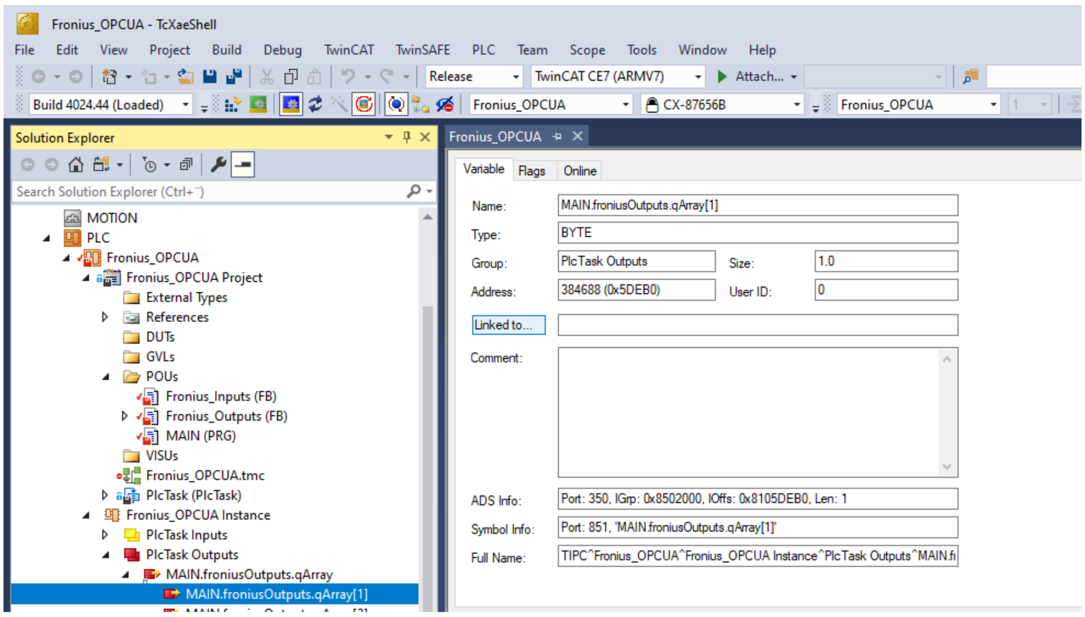
Odd Bytes			Even Bytes		
Byte	Bit	Description	Byte	Bit	Description
1	1	Heartbeat	2	1	Collisionbox
	2	Welder Ready		2	Robot Mot Rel
	3	Reserved		3	Short Circuit T0
	4	Process active		4	Reserved
	5	Current flow		5	Reserved
	6	Arc stable		6	Prm Sel Int
	7	Main current		7	Character num
	8	Touch signal		8	Torch Body Gripp
3	1	CommandValu UoR	4	1	Sensor Status 1
	2	Correction UoR		2	Sensor Status 2
	3	Reserved		3	Sensor Status 3
	4	Limitsignal		4	Reserved
	5	Reserved		5	Reserved
	6	Reserved		6	Reserved
	7	Main Supply Stat		7	Reserved
	8	Reserved		8	Reserved
5	1	Reserved	6	1	Reserved
	2	Reserved		2	Reserved
	3	Reserved		3	Reserved
	4	Reserved		4	Reserved
	5	Reserved		5	Reserved
	6	Reserved		6	Reserved
	7	Reserved		7	Reserved
	8	Reserved		8	Reserved
7	1	Process bit 0	8	1	Ext Output 1
	2	Process bit 1		2	Ext Output 2
	3	Process bit 2		3	Ext Output 3
	4	Process bit 3		4	Ext Output 4
	5	Process bit 4		5	Ext Output 5
	6	Reserved		6	Ext Output 6
	7	Reserved		7	Ext Output 7
	8	Twin Sync Active		8	Ext Output 8
9	1	Welding Voltage	10	1	Welding Voltage
	2	Low Byte		2	High Byte
	3			3	
	4			4	
	5			5	
	6			6	
	7			7	
	8			8	

5.2 Deploying and Running the Code

In order to execute the code, it must initially be built. Navigate to the “Build” option in the top menu and select “Build Solution”. Once the build process is complete, press the “Activate Configuration” button, which is highlighted in yellow in Figure 48. This action prompts a window that displays which project is set to be activated and the system to which it should be linked. There is also an option to select if the system should auto-start when the PLC is turned on; this

Table 6: Low Byte values for the corresponding Input Words. The High Byte for each Word can be determined by adding one to the Low Byte. The values described as shifted to the right are the measurements provided by the welding machine, which need to be adjusted to obtain the actual represented value. For instance, a reported welding voltage of 2950 should be interpreted as 29.5V.

Low Byte	Word	Description
9	Welding Voltage	0.0 to 655.35 [V], shifting the decimal two places to the right.
11	Welding Current	0.0 to 6553.5 [A], shifting the decimal one place to the right.
13	Wire Feed Speed	-327.68 to 327.67 [m/min], shifting the decimal two places to the right.
15	Seam Tracking	Actual real value; 0 to 6.5535, shifting the decimal four places to the right.
17	Errornumber	Gives the error number of the error, 0 is no error
21	Motor Current M1	327.68 to 327.67 [A], shifting the decimal two places to the right.
23	Motor Current M2	327.68 to 327.67 [A], shifting the decimal two places to the right.
25	Motor Current M3	327.68 to 327.67 [A], shifting the decimal two places to the right.

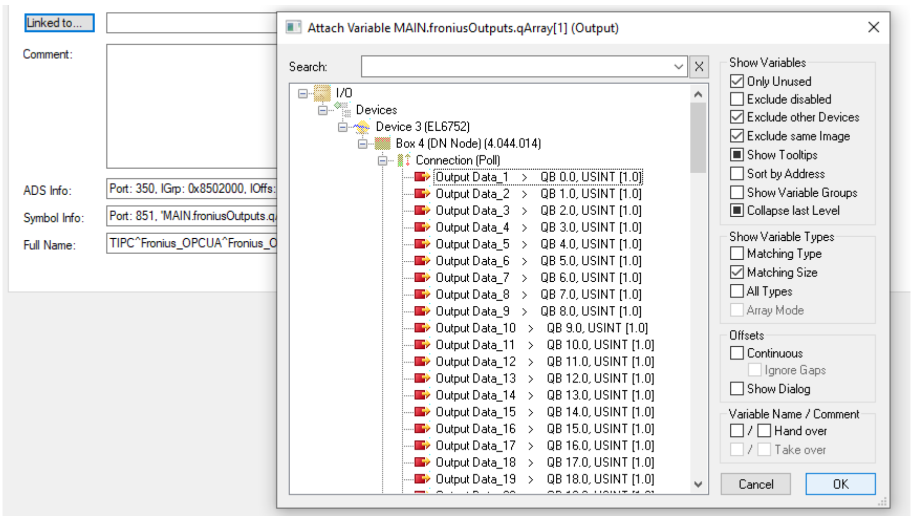


Name: MAIN.froniusOutputs.qArray[1]
Type: BYTE
Group: PlcTask Outputs Size: 1.0
Address: 384688 (0x5DEB0) User ID: 0
Comment:
ADS Info: Port: 350, IGrp: 0x8502000, IOffs: 0x8105DEB0, Len: 1
Symbol Info: Port: 851, 'MAIN.froniusOutputs.qArray[1]'
Full Name: TIPC^Fronius_OPCUA^Fronius_OPCUA Instance^PlcTask Outputs^MAIN.fi

Figure 46: Navigating to the TwinCAT variable to map

should be enabled when the PLC is ready for use. However, it might be advisable to leave this unchecked during the development phase. Press “OK”. If there are no valid licenses, a notification will appear requesting the creation of trial licenses. In such a case, click “Yes”, enter the captcha in the subsequent window, and press “OK”. This can also be done under the “System-License” section; refer to the PLC user manual for detailed instructions on how to proceed and how to add a suitable license.

The system will then prompt if you would like to restart TwinCAT in run mode, click “OK”. Once the system has completed logging in, and potentially started the program, you can set and read the variables in the program via TwinCAT, which will control the welding apparatus. For additional details, refer to the respective manuals.



Attach Variable MAIN.froniusOutputs.qArray[1] (Output)

Comment:

ADS Info: Port: 350, IGrp: 0x8502000, IOffs:
Symbol Info: Port: 851, 'MAIN.froniusOutputs.q
Full Name: TIPC^Fronius_OPCUA^Fronius_O

Search:

```
I/O
  Devices
    Device 3 (EL6752)
      Box 4 (DN Node) (4.044.014)
        Connection (Poll)
          Output Data_1   >  QB 0.0, USINT [1.0]
          Output Data_2   >  QB 1.0, USINT [1.0]
          Output Data_3   >  QB 2.0, USINT [1.0]
          Output Data_4   >  QB 3.0, USINT [1.0]
          Output Data_5   >  QB 4.0, USINT [1.0]
          Output Data_6   >  QB 5.0, USINT [1.0]
          Output Data_7   >  QB 6.0, USINT [1.0]
          Output Data_8   >  QB 7.0, USINT [1.0]
          Output Data_9   >  QB 8.0, USINT [1.0]
          Output Data_10  >  QB 9.0, USINT [1.0]
          Output Data_11  >  QB 10.0, USINT [1.0]
          Output Data_12  >  QB 11.0, USINT [1.0]
          Output Data_13  >  QB 12.0, USINT [1.0]
          Output Data_14  >  QB 13.0, USINT [1.0]
          Output Data_15  >  QB 14.0, USINT [1.0]
          Output Data_16  >  QB 15.0, USINT [1.0]
          Output Data_17  >  QB 16.0, USINT [1.0]
          Output Data_18  >  QB 17.0, USINT [1.0]
          Output Data_19  >  QB 18.0, USINT [1.0]
```

Show Variables
- [x] Only Unused
- [ ] Exclude disabled
- [x] Exclude other Devices
- [x] Exclude same Image
- ■ Show Tooltips
- [ ] Sort by Address
- [ ] Show Variable Groups
- ■ Collapse last Level

Show Variable Types
- [ ] Matching Type
- [x] Matching Size
- [ ] All Types
- [ ] Array Mode

Offsets
- [ ] Continuous
- [ ] Ignore Gaps
- [ ] Show Dialog

Variable Name / Comment
- [ ] / [ ] Hand over
- [ ] / [ ] Take over

Figure 47: Mapping the IO within TwinCAT.

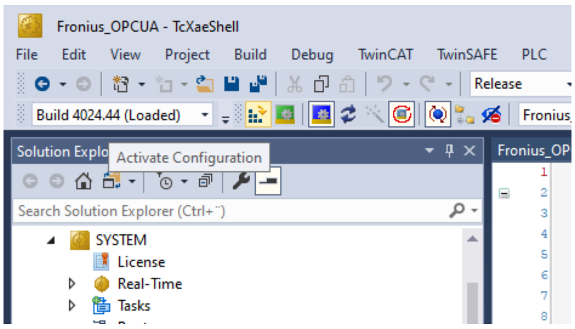


Figure 48: Activating the Configuration button.

5.3 Program Description

5.3.1 PLC Code

The programming of the PLC was carried out using the ST (Structured Text) language, one of the languages defined in the IEC61131-3 standard [8]. As previously mentioned in Section 2.4, the TwinCAT environment adheres to this standard for PLC programming. ST-code closely resembles traditional programming languages, such as C, C++, and Python, which makes it the most intuitive choice for this specific application.

In the Structured Text (ST) environment, the code is divided into two distinct text boxes: one for defining the program and variables, and the other for specifying the code that will be executed in a loop. As depicted in Figure 36, the upper text box is used to define variables, etc., while the lower compartment contains the executable code. Below these is a terminal displaying outputs, errors, and similar notifications.

The program executed in the Main section is relatively straightforward, as all functionality has been delegated to function blocks responsible for handling the inputs and outputs of the Fronius welding apparatus, respectively. Thus, the primary task within Main's definition part involves initializing these function blocks. A constant variable, redundant from an earlier program version, also remains present.

```
PROGRAM MAIN
VAR
  // Initialising the IO
  froniusOutputs : Fronius_Outputs;
  froniusInputs : Fronius_Inputs;


END_VAR
VAR CONSTANT
  ARRAY_LENGTH : UDINT := 40;
END_VAR
```

Within the code loop, the function blocks are simply invoked, ensuring they operate as intended.

```
froniusOutputs();
froniusInputs();
```

Refer to Appendix C for a comprehensive view of the PLC code. In the following sections, the pertinent elements from the code will be extracted and indicate where the excerpt's complete version can be found in the appendix. The entire TwinCAT project incorporates this code and is also included in the digital appendix. Please refer to Appendix E for a description of the included materials.

The function block for the inputs commences similarly to the main program but starts with *FUNCTION_BLOCK* rather than *PROGRAM*. Another distinguishing aspect is the inclusion of variable input and output; being a function block, it can receive variables and return some. While this feature isn't used here, given the overall program design, it is utilized in a portion of the code demonstrated later. This can be observed in the following listing, whereas the complete code can be seen in Section C.2.1.

```
FUNCTION_BLOCK Fronius_Inputs
VAR_INPUT
END_VAR
VAR_OUTPUT
END_VAR
...
```

In the standard variable definition section, *VAR*, all variables are declared according to their respective formats; thus, boolean variables are defined as boolean, and words are defined as such. However, they must be specifically defined to link these variables to the 40 bytes provided by the DeviceNet I/O. This is achieved by appending *AT %I** after the variable's name during its definition, which is applied to an array of bytes encompassing all 40 bytes, even if they may not all be utilized. This entire array is made accessible in OPC UA in case it is needed, but the intended usage involves the other variables set based on their values in the array, as done in the program part of this code.

The line *"{attribute 'OPC.UA.DA' := '1'}"* is used to make a variable available on the OPC UA server hosted by the PLC and is placed over every variable that should be on the server. Conversely, the line *"{attribute 'OPC.UA.DA.Access' := '1'}"* is only used for inputs, as it sets these variables to read-only. The following listing shows that this is implemented for the array, one of the boolean variables, and one of the words.

```
VAR
  {attribute 'OPC.UA.DA' := '1'}
  {attribute 'OPC.UA.DA.Access' := '1'}
  iArray AT %I*: ARRAY[1..ARRAY_LENGTH] OF BYTE;
  Reserved : BOOL;

  // Byte 1
  {attribute 'OPC.UA.DA' := '1'}
  {attribute 'OPC.UA.DA.Access' := '1'}
  Heartbeat : BOOL;        //Bit 1
...
  // All the Words
  {attribute 'OPC.UA.DA' := '1'}
  {attribute 'OPC.UA.DA.Access' := '1'}
  WeldingVoltage  : WORD;
...
END_VAR
...
```

At the conclusion of the input definitions, there is a section where constant variables are established. These are numbers exclusively used in the PLC code, either in the definition or in the loop code section, and hence, are not made available on the OPC UA server, as it is unnecessary. These constant variables include the array length for the previously defined array *iArray* and the low byte indices for all input words. In the following listing, this can be observed for the array and the first word.

```
VAR CONSTANT
  ARRAY_LENGTH : UDINT := 40;
  //All the indexes to the words, they are the index for the Low Byte
  //High Byte index is Low Byte index +1
  WeldingVoltageIndex   : BYTE := 9;
...
END_VAR
```

The loop code initiates by systematically setting all boolean variables controlled by a bit from different bytes. This is accomplished using the *BYTE_TO_BOOL* function. However, to ensure the correct bit within the byte is utilized, bit shifting to the right is performed with the *SHR* function. The bits are also subjected to an “and” operation with 1 to safeguard against any influence from the other bits. The execution of this process for three variables defined by the first three bits of the first byte can be observed in the following listing, while the complete code can be accessed in Section C.2.2.

```
//Bits from Byte 1
Heartbeat     := BYTE_TO_BOOL(SHR(iArray[1], 0) AND 1);
WelderReady   := BYTE_TO_BOOL(SHR(iArray[1], 1) AND 1);
Reserved    := BYTE_TO_BOOL(SHR(iArray[1], 2) AND 1);
...
```

For the word variables, both the low byte and high byte are read and combined to form a word. This is accomplished by multiplying the high byte by 256 (expressed in hexadecimal, this operation is equivalent to multiplying by 100 in decimal) and then adding the low byte. This process can be observed in the following listing for the *WeldingVoltage* variable.

```
// Creating the words by multipling the high byte by 256 and adding the low byte
WeldingVoltage  := (BYTE_TO_WORD(iArray[WeldingVoltageIndex+1]) * 16#0100) +
    BYTE_TO_WORD(iArray[WeldingVoltageIndex]);
...
```

The definition section of the output function block closely resembles the input function block; however, *AT %Q** is added to the array definition since it pertains to output variables instead of input variables. As we intend to write to the outputs, including the additional line for write protection is unnecessary when making the variables accessible for the OPC UA server.

Byte 1 in the outputs is defined with an additional Boolean variable to select whether the variables used by the welding machine should be set individually or if the entire byte should be set through the specific byte in the array. This is simply set to *TRUE*, as it has been chosen that they should be set individually. This functionality has not been extended to the other variables, which must be set individually. The only bytes that can be directly set in the array are those not established as separate variables because they were absent in the mapping.

The end of this section contains a constant variable definition part analogous to that for inputs and hence is omitted below as it follows the same format. In the following listing, one can observe the array definition, this set Boolean, and an example for one of the Boolean variables and words. The entirety of this code segment can be viewed in Section C.3.1.

```
FUNCTION_BLOCK Fronius_Outputs
...
VAR
  {attribute 'OPC.UA.DA' := '1'}
  qArray AT %Q*: ARRAY[1..ARRAY_LENGTH] OF BYTE;

  // Byte 1
  qByte1_Set_Type : BOOL := TRUE; (*True setting with the boolean variables, False
      set in qArray *)
  {attribute 'OPC.UA.DA' := '1'}
  Welding_start : BOOL;      //Bit 1
...
  // The words that can be set
  {attribute 'OPC.UA.DA' := '1'}
  Job_nr : WORD; (*Sets the Job number if in Job mode in range 1-1000. Over 1000 it
       is still 1000 *)
...
```

In the looping code segment for the outputs, the variables are used to set the bytes in the array, contrary to being set from them. To set the bytes that are controlled by the Boolean variables, the Boolean variables are multiplied with their respective bit positions in the byte and subsequently added together. The first byte, which is set in this manner, is contained within an *IF* statement that verifies the set byte. However, as it is always *TRUE*, it is consistently set in the same manner as the other bytes controlled by the Boolean variables. The two bytes set by a single word are established using a method function described subsequent to this segment. This function accepts the word and the low-byte index defined in the constant variables. The following listing illustrates this for the first byte and first word, which is *Job_nr*. The entirety of this code segment can be examined in Section C.3.2.

```
IF qByte1_Set_Type THEN
  qArray[1] := (BOOL_TO_BYTE(Welding_start) * 16#01) +
          (BOOL_TO_BYTE(Robot_ready) * 16#02) +
...
Set_Word_TO_Bytes(setWord := Job_nr, Index := JobIndex);
...
```

The method used to set the bytes from the word is defined similarly to the other parts of the code, but it uses the *METHOD* keyword instead. Notably, it also possesses a return value: a boolean indicating whether the method execution was successful. This method utilizes two inputs: the word that sets the bytes, and the index which determines the appropriate bytes in the array. It requires only these definitions, as methods have access to the variables within the function block they are defined for. Hence, only the variable inputs are defined, as illustrated in the following listing. This represents the entirety of this code segment, although it can also be located in Section C.3.3, where the loop code segment also resides.

```
METHOD Set_Word_TO_Bytes : BOOL
VAR_INPUT
  setWord : WORD;
  Index : BYTE;     // Index for the two Bytes is always High Byte index = Low Byte
      index +1
END_VAR
```

The loop code segment straightforwardly sets the correct byte using the index, where the high byte corresponds to the input index+1, considering that the index pertains to the low byte. For the high byte, the word is divided by 256, thus producing an effect opposite to what was described for the high bytes in the input section; this value is then converted to a byte via the *WORD_TO_BYTE* function. For the low byte, the *AND* operation is employed in conjunction with 255 to ensure that only the low byte portion is converted into the low byte. This process is demonstrated in the following listing.

```
qArray[Index+1] := WORD_TO_BYTE(setWord / 16#0100); // Divide by 256 to get the
    high byte
qArray[Index] := WORD_TO_BYTE(setWord AND 16#00FF); // AND with 255 to get the low
    byte
```

5.3.2 Python Test Client Codes

The Python test clients utilized to assess the functionality of the OPC UA server on the PLC were derived from the open-source project, Free OPC-UA Library, as introduced in Section 2.5.6. Specifically, the opcua-asyncio library was employed for this purpose. The clients developed for functionality and performance testing were based on the examples provided with the opcua-asyncio library [79].

Welding_test_client.py This client was developed to test and provide a proof of concept that it is possible to connect to the server with code-based OPC UA clients rather than solely using pre-built test clients such as UA Expert. Some of the critical components of the code will be highlighted here, but the entire code can be viewed in Section D.1. In the code segments presented below, lines of unimported code have been removed. These are hashed-out lines and print statements used to simplify the terminal output, although they do not contribute to the functionality.

Given that the OPC UA library is based on Asyncio, an async task (loop) is defined for executing the code. In the main definition, this is called in the correct manner, according to Asyncio, with the main initiation occurring at the very bottom. The first action in this task function is to establish the necessary information for connecting to the server, including IP address and credentials. The connection is then made with “async with client:”. All of this can be seen in the listing below.

```
async def task(loop):
    url = "opc.tcp://169.254.119.112:4840/freeopcua/server/"
    client = Client(url=url)
    credentials_file = 'credentialsPLC.txt'
    # Read username and password from the file
    with open(credentials_file, 'r') as f:
        username = f.readline().strip()
        password = f.readline().strip()
    client.set_user(username=username) #########
    client.set_password(password) #########
    await client.set_security(
        SecurityPolicyBasic256Sha256,
        certificate=cert,
        private_key=private_key#,
        #server_certificate="certificate-example.der"
    )
    async with client:
. . .
def main():
    loop = asyncio.get_event_loop()
    loop.set_debug(True)
    loop.run_until_complete(task(loop))
    loop.close()

if __name__ == "__main__":
    main()
```

To access the variables we are interested in, we must browse down through the client's object nodes, as direct addressing was not feasible. An example of the path to one of the variables we wish to inspect is *Robot_ready*, which becomes *Objects* → *PLC1* → *MAIN* → *froniusOutputs* → *Robot_ready*. The code below shows how this browsing is performed from *Objects* to *PLC1*.

```
    async with client:        # This line is the same as above
        objects = client.nodes.objects
        children = await objects.get_children()
        for child in children:
            display_name = await child.read_display_name()
            if 'PLC1' in display_name.Text:
                plc1 = child
            print(display_name)

        print(plc1)
        plc1_children = await plc1.get_children()
```

The same process is carried out all the way down to the lowest part, where all variables in this test client are selected to add the variables from OPC UA to one list for inputs and another list for outputs. In the main section, both *froniusInputs* and *froniusOutputs* are being sought after. The code below shows how the outputs are added to the list, and it is similar for the inputs.

```
        outputs = []
        for child in fOut_children:
            display_name = await child.read_display_name()
            if 'qArray' in display_name.Text:
                pass
            else:
                outputs.append(child)
            dVal = await child.read_data_value()
            var_type = dVal.Value.VariantType
            print('Output Variables: ',display_name.Text, ';    ', var_type)
```

The excerpt below depicts a portion of the start function. It illustrates how to read and write to OPC UA variables by using *await* with *variable.write_value(...)* or *variable.get_value()*. To index the correct variable in the lists, they are printed in the code, though more efficient solutions for storing them for later use can be implemented. Both the start and stop functions must be called with await, and this must be done within the *async with client:* section of the task loop.

```
async def start(outputs, inputs):
    weldStart = outputs[0]
    robReady = outputs[1]
    job = outputs[25]
    weldReady = inputs[1]

    await robReady.write_value(ua.DataValue(ua.Variant(True)))
    await job.write_value(ua.DataValue(ua.Variant(12, ua.VariantType.UInt16)))
    print(await job.get_value())

    await weldStart.write_value(ua.DataValue(ua.Variant(True)))

    print('weldReady', await weldReady.get_value())
    print('processActive', await processActive.get_value())
    print('weldStart', await weldStart.get_value())
```

The stop function bears considerable resemblance but instead turns off the welder by setting *weldStart* to false. Towards the end of the task loop function, within the *async with client:* segment, there is a while loop featuring a match case that calls for start, stop, or ends the program based on basic user input in the terminal.

Welding_test_client_loop.py

The portions of this code that differ from *Welding_test_client.py* can be viewed in Section D.2. This code was designed to test the continuous operation of the client, enabling observation of the server and PLC's behaviour during use. However, this is not sufficiently reliable for testing time usage; thus, a new variant was developed, as detailed below.

Welding_test_client_counter.py

In this code, the changes involve replacing the match case with a for loop, which only sets the robot-ready signal and reads only the welder-ready signal. The length of time it takes from setting the robot ready until the welder-ready changes are measured. This measurement is taken by recording an end time when it changes and subtracting the start time. To improve accuracy, asyncio sleep is used instead of time sleep, and an additional advantage of this is that it prevents the server connection from being disrupted, which can happen when waiting too long with time sleep. The results are added to a list, which is eventually written to a file. This can be seen below, and all changes to this code from *Welding_test_client.py* can be viewed in Section D.3.

```
        time_elapsed_list = []
        robReady = outputs[1]
        weldReady = inputs[1]
        for i in range(100):
            await robReady.write_value(ua.DataValue(ua.Variant(False)))
            await asyncio.sleep(1)
            #time.sleep(1)

            start_time = time.time()
            await robReady.write_value(ua.DataValue(ua.Variant(True)))

            while not await weldReady.get_value():
                await asyncio.sleep(0.001)  # Check every millisecond

            end_time = time.time()
            time_elapsed = (end_time - start_time) * 1000
            time_elapsed_list.append(time_elapsed)
            print(f"Time elapsed: {time_elapsed} milliseconds")
            await asyncio.sleep(1)

        #print(time_elapsed_list)

        # Write the times to a CSV file
        with open('times.csv', 'w', newline='') as f:
            writer = csv.writer(f)
            writer.writerow(["Elapsed Time (ms)"])
            for t in time_elapsed_list:
                writer.writerow([t])
```

6 Test Results

6.1 Communication Testing Results

Simple communication tests with different OPC UA clients have been conducted successfully. It is feasible to set signals and change the job number. The Python test clients are operational, and the results from the "Welding_test_client_loop.py" script will be presented later in this section under Section 6.2.2. A screenshot of the initial Python test client, presented in Section 5.3.2, can be viewed executed on a Linux PC in Figure 49.

6.2 Metrics

6.2.1 UA Expert Performance Test

With UA Expert, briefly described in Section 2.5.5, one can assess the performance of the OPC UA server. The results of conducting such a test can be viewed in Figure 50. This test was configured for both read and write call testing. In Figure 51, one can see a zoomed-in version of Figure 50b, featuring the first 400 cycles. These tests were conducted with the computer running the UA Expert program directly connected to the PLC.

Figure 50a depicts a graph of the average call duration for all cycles for the different node counts, with points for maximum and minimum call duration for both reading and writing. The maximum points are of particular interest and therefore emphasized. For eight nodes case where it may appear to only account for reading, the corresponding writing point is located underneath because it was so close in value.

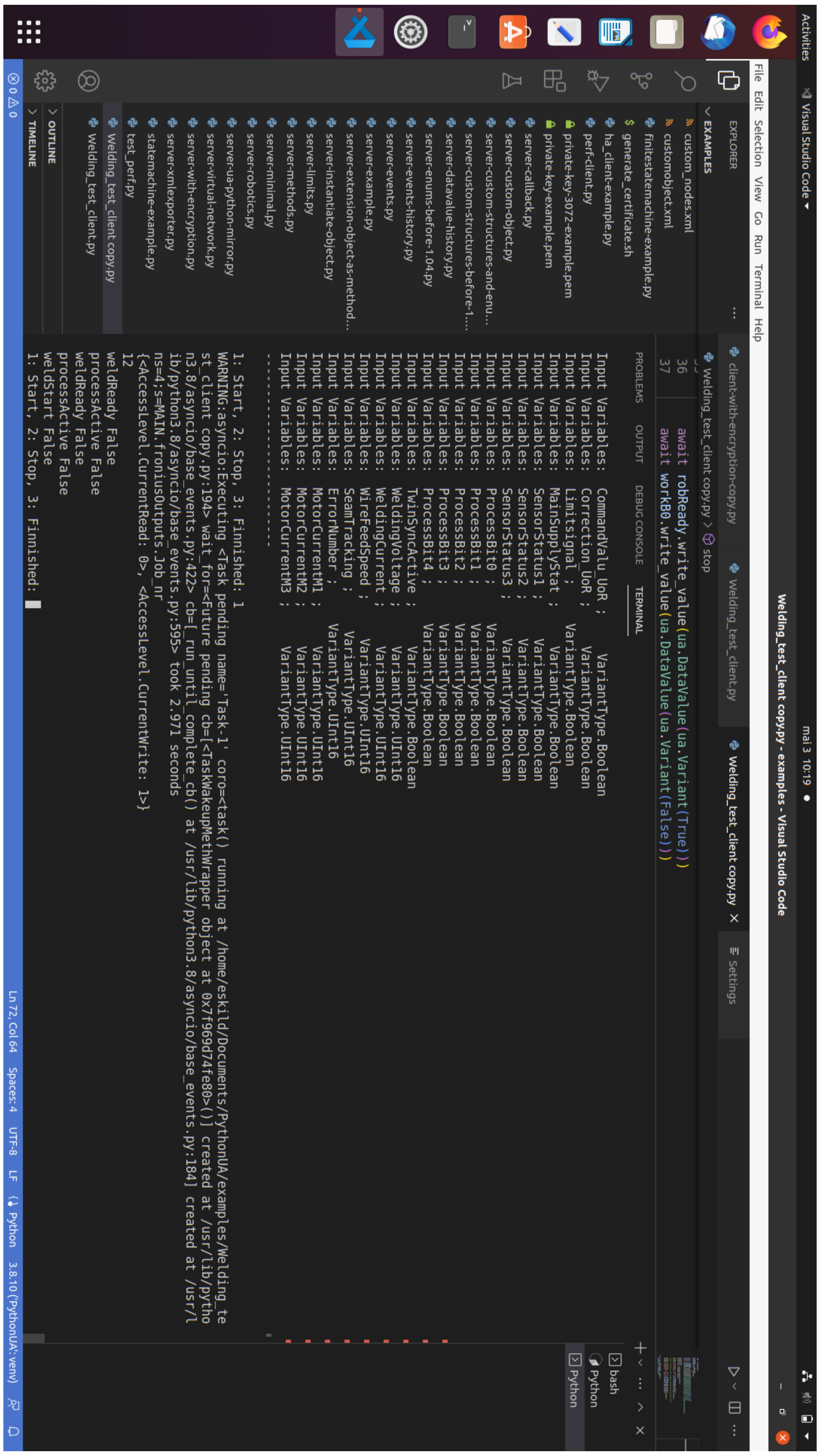
By examining the read graph in Figure 50b and the write graph in Figure 50c, a similar trend is evident, which can be seen more clearly in Figure 51. The results cluster around their respective mean times, as can be seen for reading in Test Run 1 in Table 7, and for the same test run in Table 8. They often peak nearly at 30ms, but occasionally spike up to approximately 100ms. It's important to note that this may not necessarily occur throughout a complete run of 1000 cycles for a given node count. This observation can be drawn from the tables displaying the maximum call time for reading in Table 11 and for writing in Table 12. In the aforementioned tables, we observe that the call duration spikes to approximately 100ms for one to two of the node counts for a single test run.

Table 7: This table provides the average read time during each test run for different node counts, with 1000 cycles each run.

Average Read Times [ms]				
Test Run	8 nodes	16 nodes	34 nodes	69 nodes
1	6.12	6.79	7.73	10.32
2	5.80	6.72	8.27	10.38
3	4.90	6.83	8.35	10.45
4	6.11	6.77	7.75	10.31
5	5.50	6.80	8.30	10.47
6	5.60	6.60	8.23	10.42
7	6.29	6.53	7.37	10.41
8	6.00	6.60	8.32	10.48
9	5.96	6.62	8.22	10.50
10	5.72	7.05	8.16	10.32

Table 8: This table provides the average write time during each test run for different node counts, with 1000 cycles each run.

Average Write Times [ms]			
Test Run	8 nodes	16 nodes	34 nodes
1	6.30	6.45	7.54
2	5.80	6.93	8.25
3	6.03	6.87	8.03
4	6.16	6.39	-
5	5.77	7.02	8.41
6	5.38	6.74	8.20
7	6.29	6.41	7.98
8	5.06	7.00	8.32
9	5.28	6.91	8.15
10	6.24	6.62	7.97



```
await robReady.write_value(ua.DataValue(ua.Variant(True)))
await workB0.write_value(ua.DataValue(ua.Variant(False)))
```

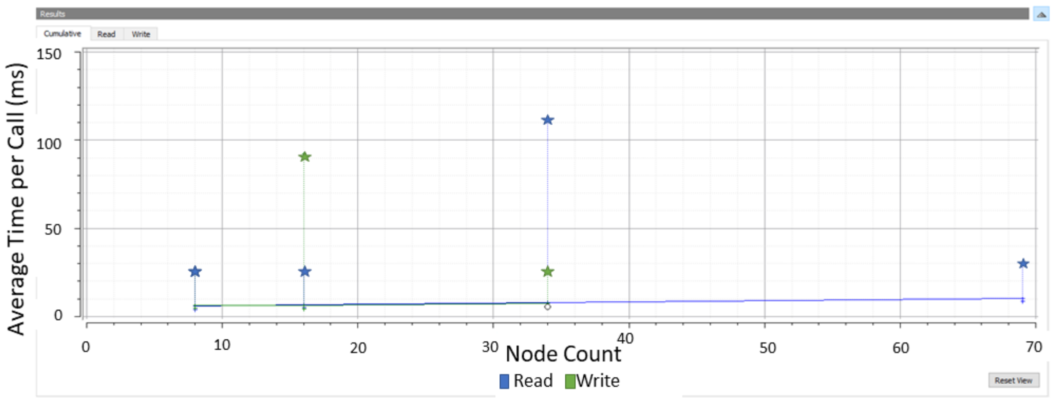
```
Input Variables:  CommandValu_UoR ;     VariantType.Boolean
Input Variables:  Correction_UoR ;     VariantType.Boolean
Input Variables:  Limitsignal ;    VariantType.Boolean
Input Variables:  MainSupplyStat ;     VariantType.Boolean
Input Variables:  SensorStatus1 ;     VariantType.Boolean
Input Variables:  SensorStatus2 ;     VariantType.Boolean
Input Variables:  SensorStatus3 ;     VariantType.Boolean
Input Variables:  ProcessBit0 ;    VariantType.Boolean
Input Variables:  ProcessBit1 ;    VariantType.Boolean
Input Variables:  ProcessBit2 ;    VariantType.Boolean
Input Variables:  ProcessBit3 ;    VariantType.Boolean
Input Variables:  ProcessBit4 ;    VariantType.Boolean
Input Variables:  TwinSyncActive ;     VariantType.Boolean
Input Variables:  WeldingVoltage ;     VariantType.UInt16
Input Variables:  WeldingCurrent ;     VariantType.UInt16
Input Variables:  WireFeedSpeed ;     VariantType.UInt16
Input Variables:  SeamTracking ;     VariantType.UInt16
Input Variables:  ErrorNumber ;    VariantType.UInt16
Input Variables:  MotorCurrentM1 ;     VariantType.UInt16
Input Variables:  MotorCurrentM2 ;     VariantType.UInt16
Input Variables:  MotorCurrentM3 ;     VariantType.UInt16
-------------------------

1: Start, 2: Stop, 3: Finished: 1
WARNING:asyncio:Executing <Task pending name='Task-1' coro=<task() running at /home/eskild/Documents/PythonUA/examples/Welding_te
st_client copy.py:194> wait_for=<Future pending cb=[<TaskWakeupMethWrapper object at 0x7f969d74fe80>()] created at /usr/lib/pytho
n3.8/asyncio/base_events.py:422> cb=[_run_until_complete_cb() at /usr/lib/python3.8/asyncio/base_events.py:184] created at /usr/l
ib/python3.8/asyncio/base_events.py:595> took 2.971 seconds
ns=4;s=MAIN.froniusOutputs.Job_nr
{<AccessLevel.CurrentRead: 0>, <AccessLevel.CurrentWrite: 1>}
12
weldReady False
processActive False
weldReady False
processActive False
weldStart False
1: Start, 2: Stop, 3: Finished:
```

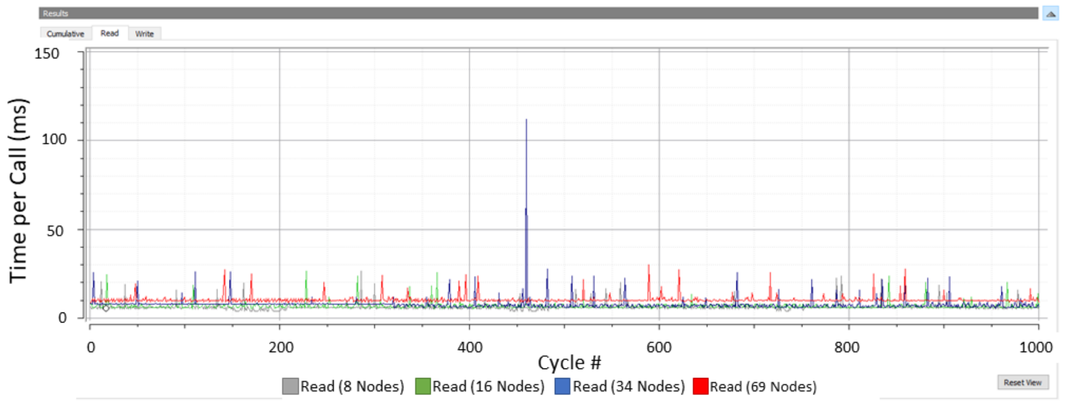
Activities
Visual Studio Code
mai 3 10:19
Welding_test_client copy.py - examples - Visual Studio Code
File Edit Selection View Go Run Terminal Help
EXPLORER
EXAMPLES
custom_nodes.xml
customobject.xml
finitestatemachine-example.py
generate_certificate.sh
ha_client-example.py
perf-client.py
private-key-3072-example.pem
private-key-example.pem
server-callback.py
server-custom-object.py
server-custom-structures-and-enu...
server-custom-structures-before-1....
server-datavalue-history.py
server-enums-before-1.04.py
server-events-history.py
server-events.py
server-example.py
server-extension-object-as-method...
server-instantiate-object.py
server-limits.py
server-methods.py
server-minimal.py
server-robotics.py
server-ua-python-mirror.py
server-virtual-network.py
server-with-encryption.py
server-xmlexporter.py
statemachine-example.py
test_perf.py
Welding_test_client copy.py
Welding_test_client.py
OUTLINE
TIMELINE
client-with-encryption-copy.py
Welding_test_client.py
Welding_test_client copy.py
Settings
Welding_test_client copy.py > stop
PROBLEMS
OUTPUT
DEBUG CONSOLE
TERMINAL
bash
Python
Python
Ln 72, Col 64
Spaces: 4
UTF-8
LF
Python
3.8.10 ('PythonUA': venv)

Figure 49: A screenshot demonstrating the Linux PC in Figure 21 executing the code detailed in Section 5.3.2.

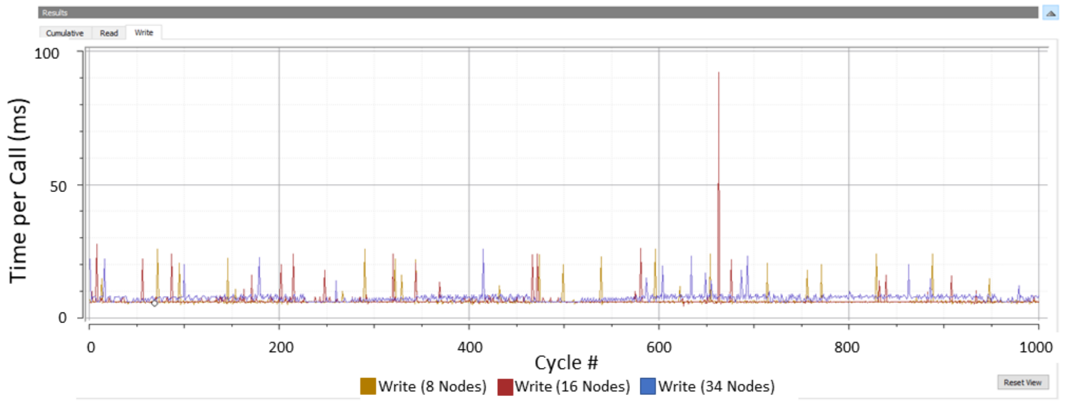
By examining the tables for minimum times Table 9 and Table 10 and comparing them with the tables for average times Table 7 and Table 8, we observe that the minimum times are slightly lower. However, they exhibit a tendency to increase with the number of nodes, similar to the trend seen with the average times.



(a)



(b)



(c)

Figure 50: Figure (a) presents the cumulative graph of UA Expert's Performance View with average, minimum, and maximum indicators for each node count. Figure (b) illustrates a complete read call graph, with each line representing different node counts and the axes showing cycle numbers and milliseconds used, respectively. Figure (c) offers a similar graph for the write call instead.

By further analyzing the tables for maximum times in Table 11 and Table 12, we observe that the times are approximately 26 to 30 ms. These times tend to be around 25-26 ms for lower node counts and closer to 30 ms for higher node counts, excluding the random spikes to about 100 ms. Therefore, except for these significant spikes, the maximum time tables also follow a trend similar to the others, where the time slightly increases with the number of nodes.

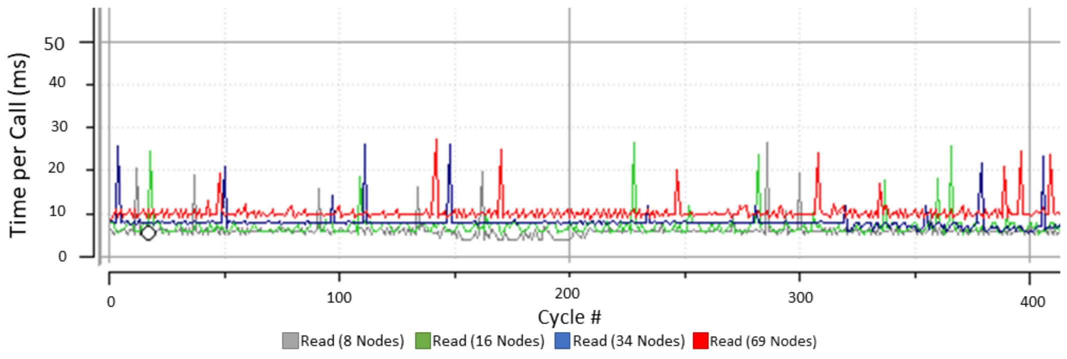


Figure 51: A zoomed in version of Figure 50b.

Table 9: This table provides the minimum read time during each test run for different node counts, with 1000 cycles each run.

Minimum Read Times [ms]				
Test Run	8 nodes	16 nodes	34 nodes	69 nodes
1	3.93	5.09	5.63	8.55
2	3.83	4.59	5.93	8.92
3	3.64	4.85	6.00	8.78
4	3.91	5.16	5.55	8.39
5	3.86	4.76	6.07	8.71
6	3.57	4.83	5.74	8.86
7	3.94	5.14	5.59	8.88
8	3.92	4.97	5.87	8.38
9	3.87	4.91	5.87	8.96
10	3.73	4.92	5.84	8.83

Table 10: This table provides the minimum write time during each test run for different node counts, with 1000 cycles each run.

Minimum Write Times [ms]			
Test Run	8 nodes	16 nodes	34 nodes
1	4.93	4.47	5.72
2	3.58	4.99	5.86
3	3.78	5.05	5.90
4	5.13	5.38	-
5	3.76	5.03	6.01
6	3.55	4.96	6.79
7	4.57	4.89	5.88
8	3.57	5.00	5.87
9	3.57	4.82	6.95
10	3.93	5.10	5.88

Table 11: This table provides the maximum read time during each test run for different node counts, with 1000 cycles each run.

Maximum Read Times [ms]				
Test Run	8 nodes	16 nodes	34 nodes	69 nodes
1	26.44	26.41	112.09	29.99
2	111.95	24.38	28.34	110.03
3	24.64	27.88	28.16	109.45
4	26.82	27.85	105.90	20.00
5	109.16	25.95	28.01	111.64
6	98.82	24.97	28.54	115.14
7	26.01	25.94	113.97	29.99
8	79.98	27.85	27.91	111.00
9	74.85	24.96	29.96	85.97
10	23.69	24.79	110.85	30.02

Table 12: This table provides the maximum write time during each test run for different node counts, with 1000 cycles each run.

Maximum Write Times [ms]			
Test Run	8 nodes	16 nodes	34 nodes
1	25.97	92.15	25.82
2	24.05	23.97	109.88
3	26.05	26.06	91.94
4	26.13	109.34	-
5	23.97	27.93	95.82
6	24.31	28.01	28.17
7	24.72	107.04	26.77
8	24.02	27.92	27.91
9	24.03	28.05	27.59
10	25.92	28.01	89.96

6.2.2 Python Code time test

This testing was performed using the Python test code described in Section 5.3.2. These tests were conducted using the HP network switch shown in Figure 20 from Section 3. The results of the first ten iterations from four different test runs can be viewed in Table 13. The bottom of the table also includes the average of the results. If a specific test run was executed with more than ten iterations, those additional iterations are factored into the calculation of the average. Graphical representations of these results can be seen in Figure 52 and Figure 53.

The first graph, Figure 52, displays the results from the initial test run, which consisted of 10 iterations. As one can also discern from the column for Test Run 1 in Table 13, all results except

Table 13: This table presents the elapsed time (in milliseconds) it takes for a change in the robot's Ready signal to affect a corresponding change in the welder's Ready signal. Each column corresponds to a specific run. Runs 1 and 2 show the times from all their tests, while Runs 3 and 4 were each performed 100 times, with only the first ten results displayed here.

	Elapsed Time (ms)			
	Test Run 1	**Test Run 2**	**Test Run 3**	**Test Run 4**
Cycle 1	12.996	221.941	428.993	14.994
Cycle 2	13.001	16.955	12.993	12.995
Cycle 3	14.006	12.994	11.973	15.991
Cycle 4	16.005	16.949	33.002	13.999
Cycle 5	16.045	14.006	24.992	17.000
Cycle 6	15.997	44.950	14.957	14.954
Cycle 7	15.048	14.964	14.997	16.999
Cycle 8	13.953	16.952	15.055	20.990
Cycle 9	15.012	16.953	14.994	18.008
Cycle 10	27.001	12.954	14.953	16.955
Averages	15.906	38.962	19.272	15.690

the last lie between 13 and 16 ms, while the last one jumps to 27 ms. From the other test runs in Table 13, it can be seen that the time usage usually falls within this range and a bit higher. However, similar to the tests with UA Expert from Section 6.2.1, there are occasional minor spikes and, very rarely, significant spikes. This can be observed in Figure 53, where two of the test runs start with very high spikes and otherwise exhibit a few moderate spikes.

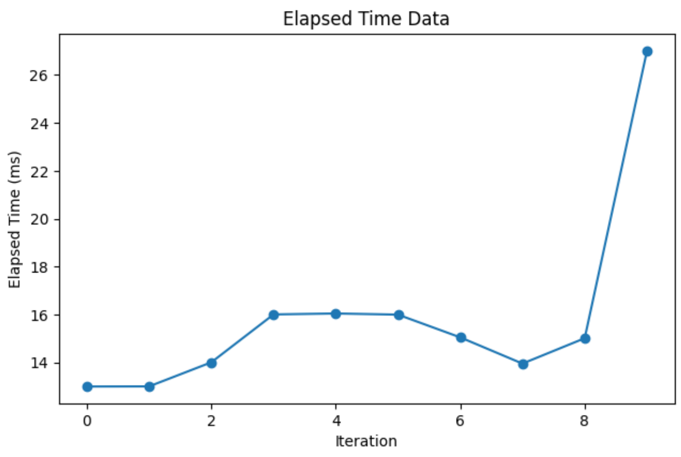


Figure 52: Graph of the first Python test run.

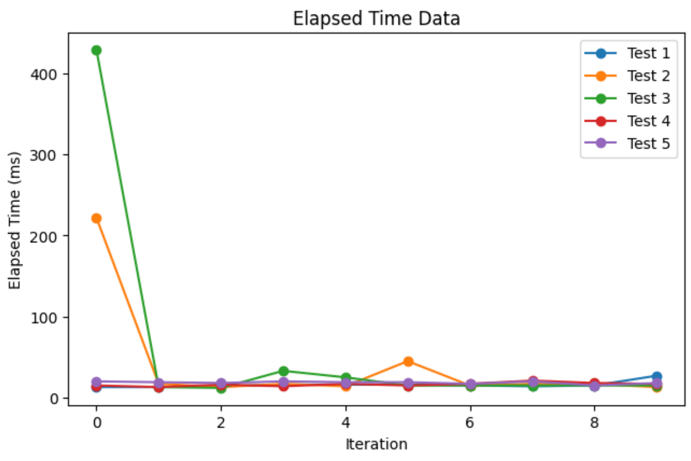


Figure 53: Graph of all python test runs.

7 Discussion

7.1 ROS2

The rationale for the proposed use of ROS2 in controlling the robots designated for welding tasks is detailed in the Motivation section of this thesis, as referenced in Section 1.1. Specifically, ROS2 facilitates the execution of more complex tasks, including the simulation and planning of intricate paths around more challenging shapes for welding purposes. As discussed in Section 2.2, ROS2 provides various advantageous features, albeit with some disadvantages. Some of these aspects are presented below.

Advantages:

- The modular architecture simplifies the development and integration of software components.
- The provision for real-time and safety-critical systems is significant for a multitude of industrial applications.
- ROS2's compatibility with multiple operating systems enhances the ease of deployment and maintenance of industrial systems across diverse hardware platforms.
- As an open-source initiative, it is backed by a large and expanding community of developers. This community support can streamline the process of locating resources and obtaining assistance during application development.

Disadvantages:

- New users unfamiliar with ROS2 may encounter a learning curve, potentially prolonging development and deployment durations.
- Compared to other software platforms specifically designed for industrial applications, ROS2 might lack or have incomplete packages and tools.

- ROS2 may require higher hardware requirements and involve a more intricate installation process than other software platforms, which could augment the cost and complexity of deploying industrial systems.
- Potential compatibility issues might arise with existing software components that were developed for other platforms, thereby requiring further development or integration work to resolve.
- ROS2 might not be the optimal choice for all industrial applications. Accordingly, assessing other software platforms or custom development solutions may be crucial, depending on the specific needs of a given application.

Several of the identified drawbacks of ROS2 are expected to diminish as the platform continues to evolve and garner increased adoption within various industries, leading to more contributions towards its development and refinement.

7.2 OPC-UA vs MQTT

MQTT, as described in Section 2.6, and OPC-UA, as detailed in Section 2.5, represent two widely utilized communication protocols in the realm of Industrial IoT (IIoT). Each protocol carries its unique advantages and disadvantages, and they are typically deployed for distinct purposes. A selection of the most significant advantages and disadvantages of both protocols is presented below.

Advantages of MQTT:

- Lightweight: MQTT is designed to be highly efficient and lightweight, making it ideal for use in low-power and low-bandwidth devices.
- Scalable: MQTT uses a publish-subscribe architecture, which makes it highly scalable and allows for efficient communication between multiple devices.
- Simple: The MQTT protocol is relatively simple and easy to implement, making it accessible to a wide range of developers and devices.
- QoS levels: MQTT supports three levels of Quality of Service (QoS), which provides flexibility in choosing the level of message delivery guarantee.

Disadvantages of MQTT:

- Limited security: MQTT's inherent security mechanisms are restricted, potentially exposing it to cybersecurity threats if not adequately secured.
- Limited metadata support: The support for metadata within MQTT is constrained, potentially complicating the management and understanding of the transmitted data.
- Limited interoperability: MQTT, designed for simplicity and lightweight operation, may not be suitable for complex systems or those necessitating high degrees of interoperability.
- Explicit definition of message structure required: The structure of the message must be defined in more explicit detail when using MQTT.

- Necessity for a broker: The use of MQTT necessitates the presence of a broker to facilitate communication.

Advantages of OPC-UA:

- Enhanced security: OPC-UA is equipped with robust security mechanisms, including encryption and authentication, rendering it a more secure option for IIoT applications.
- Comprehensive metadata: OPC-UA possesses an extensive set of metadata and information modelling capabilities, which can simplify the management and understanding of the transmitted data.
- Interoperability: OPC-UA is engineered to ensure high levels of interoperability, facilitating communication between different devices and systems, irrespective of the manufacturer or platform employed.
- Industry standard: OPC-UA is already widely adopted by a variety of industrial equipment manufacturers, including those producing PLCs.

Disadvantages of OPC-UA:

- Complexity: OPC-UA is a complex protocol with a steep learning curve and higher implementation costs compared to MQTT.
- Resource-intensive: OPC-UA requires more processing power and bandwidth than MQTT, making it less suitable for low-power and low-bandwidth devices.
- Limited scalability: OPC-UA is less scalable than MQTT, as it uses a client-server architecture, which can limit the number of clients that can be connected to a server.

In summary, MQTT represents a lightweight and efficient protocol that is particularly suitable for devices operating with low power and low bandwidth, in more traditional IoT rather than IIoT. On the other hand, OPC-UA is a more intricate and secure protocol designed specifically for complex IIoT applications requiring significant levels of interoperability. The protocol choice largely depends on the application's specific requirements, including available resources, security needs, and data management prerequisites.

For this project, OPC-UA more effectively meets the requirements specified in the Project Description, mainly due to its pre-existing integration into TwinCAT by Beckhoff. However, the same holds true for MQTT. Moreover, given that the solution is not constrained by performance limitations, the increased processing power introduced by OPC-UA poses no issue. Importantly, OPC-UA's architecture aligns more naturally with the ROS2 concepts that the solution is designed to work with, removing the need for an additional intermediary, such as the broker that the MQTT protocol relies upon.

7.3 The Solutions Explored

Multiple methodologies were explored to address the task provided in the Project Description, as referenced in Section 1.2, which inherently served as one of the secondary objectives set. Thereby, three primary approaches were explored in more detail.

- Implementing OPC UA or an analogous protocol directly onto the welding apparatus
- Administering the welding apparatus via the robot controller
- Utilizing the DeviceNet connection more directly rather than through the robot controller

The initial trajectory pursued in depth involved establishing a direct connection to the welding machine using OPC UA. As per the Fronius website [59], this approach initially appeared highly promising. The site indicated full support for OPC UA and MQTT, with the former enabling both the reading and setting of process parameters and the latter used for reading the parameters. This was to be facilitated via their WeldCube solution and the accompanying WeldCube API. However, it was not explicitly communicated that the WeldCube solution relies on a specialized computer sold by Fronius, intended for connection with the welding apparatuses. The requirement of an additional intermediary device rendered the solution less appealing than initially perceived.

The alternative solution investigated more thoroughly involved interfacing with the Fronius welding apparatus via the Yaskawa Robot Controller already connected to it. Utilizing the Yaskawa SDK and programming environment, this might have been a feasible solution. However, the advisor had previously explored these possibilities and recommended pursuing different avenues for solving the given task.

A USB to DeviceNet option was also considered. The advisor acknowledged the potential of employing a device connected via USB, though emphasized that an Ethernet connection would be preferable. An available USB to DeviceNet converter was discovered, but it remains uncertain whether it would meet the project's specific needs. This would require further investigation and perhaps testing. This could facilitate the connection of a PC, Raspberry Pi or similar device to host an OPC-UA server akin to the chosen solution. However, it may not offer the same level of robustness and reliability.

7.3.1 Initial Solution That Commenced Development

The resolution was to employ a PLC - specifically, a CX7000 from Beckhoff - for managing the welding machine through DeviceNet, utilizing the DeviceNet master terminal EL6752. Subsequently, the PLC is administered by a PC via an Ethernet connection. The preference for Beckhoff was primarily dictated by the advisor's recommendation, given the existing utilization of Beckhoff in the robot cells. Other potential alternatives, such as Siemens, are also available.

Beckhoff is significantly more economical than Siemens, and according to several individuals from the institute, Siemens has struggled with maintaining adequate supply. This supply issue was a principal reason for the existence of the project thesis that the author participated in. Conversely, Beckhoff provided a swift response and delivery of their products.

An advantage of this approach was that when the requirement to incorporate OPC UA capabilities arose, even though it necessitated the purchase of a new PLC, it was possible to continue using the same PLC code and other aspects of the system that had been developed. The only required addition was the implementation of the features that enabled OPC UA functionality.

7.3.2 The Final Solution Developed

The solution developed to address the task outlined in the Project Description, and to meet as many of the set objectives as possible, is the one presented in Section 3. This solution is presented extensively throughout the thesis, subsequent to that section.

With the implementation of this solution, the majority of the goals stipulated in the Objectives section were achieved. Two of the secondary objectives pertained to the research and testing of alternative solutions, objectives that inherently could not be fulfilled by a single solution. As illustrated above, several alternatives were indeed examined during this project. However, no additional solutions were tested or implemented.

The connection and communication between a computer and the PLC are facilitated via Ethernet using the OPC UA protocol. This setup allows the control of the welding apparatus by choosing the appropriate job. However, modifying parameters directly poses a more significant challenge. Based on extensive testing of the communication with the welding apparatus, it remains unclear whether the observed limitations are inherent to the Fronius robot interface or are related to unidentified criteria necessary for effective control. Thus, the objective of complete control has not been achieved.

Nonetheless, the existing functionality is plug-and-play. To set it up, one merely needs to connect the PLC to the welding apparatus using the DeviceNet cable, link the PC via Ethernet, and connect the power. At this point, the system is operational. A login is required when connecting to the OPC UA server - a standard feature for such systems. Since the credentials can be stored on the client side, system access can be secured simply by launching the client.

7.4 Time testing with a non-RT programming language and OS

All the tests providing metrics in the results section were performed using Windows PCs. This is not ideal, as they should ideally be conducted on Linux machines that have been patched for real-time operations. Nonetheless, these tests still indicate the time spent on communication and offer worst-case scenario results, as it is possible with Windows PCs.

The test involving measuring the milliseconds from when a command is sent via OPC UA until changes can be read. In this instance, a potential source of error is that the tests should ideally be written in a real-time capable language such as C++ instead of Python. However, due to time constraints, the tests were performed in Python as the test client had already been developed in that language.

7.5 Test results

From the results in Section 6.2, it can be seen that the communication between the PC and PLC, as per the UA Expert results in Section 6.2.1, and the entire round trip to the welding apparatus and back, as demonstrated in the Python timing test in Section 6.2.2, occurs in the order of 10 ms, though it can occasionally spike higher.

The results from UA Expert reveal that the majority of the time, the latency remains around the averages reported in the respective results section. However, frequent spikes to approximately 30 ms and occasionally to about 100 ms are observed. The smaller spikes may originate from the general communication with the PLC, while the larger spikes may be due to the PC being busy. As mentioned earlier, these tests should ideally be conducted with an operating system optimized for real-time operations. Without such testing, it is impossible to definitively determine whether this is the cause of the larger spikes in latency.

The results shown in Section 6.2.2 exhibit similar tendencies to those seen from the UA Expert results. Often, the response times hover around the average when numerous measurements are taken. However, they occasionally spike to 2-3 times the average and, in rare instances, even more. This behaviour can be seen quite distinctly in the first result of tests 2 and 3 in Table 13 and very clearly in Figure 53. These particularly large spikes in time consumption might be due to the PC being occupied with other tasks. Using a PC optimized for real-time operations might help alleviate this, but testing would be necessary to confirm this hypothesis.

7.6 Errors and troubleshooting

Progress was sometimes stalled during the project due to minor errors or important aspects being overlooked when setting up the various components. Below are some of the most significant errors and the troubleshooting steps taken to resolve them:

7.6.1 No external power on the DeviceNet cable

The DeviceNet cable requires power from an external source. Without it, the terminal still appears in TwinCAT, and things function partially. However, numerous error messages about ADS and a WCstate set to 1 occur, indicating a fault. Searching for this problem online yields no solution, merely indicating there is an issue and the communication isn't functioning as it should, which isn't very helpful. However, this issue arises because the communication requires 24V power from an external source. Once connected, the communication functions properly, and the WCstate becomes zero. This error transpired despite the known fact that DeviceNet uses a twisted pair for 24 V power. An incorrect assumption was made that the DeviceNet terminal supplied this power, given the requirement for a separate power connection on the PLC for the IO and IO terminals.

7.6.2 Opening the port on the PLC

During the installation of OPC-UA, issues were encountered when trying to connect to the server. Considerable time was spent identifying the cause of these issues. In the documentation from Beckhoff, which can be found here [73], various aspects regarding the port are mentioned. It is pointed out that a router must be able to forward the port, and that 4840 is a default port for many OPC-UA implementations and thus might already be in use, but there is little or no mention that the port must be opened on the PLC itself. This is the reason why it is explicitly included in the guide that the port must be opened on the PLC, and how to accomplish this with Windows CE on the PLC, as this was the operating system of the PLC used in this master thesis.

7.6.3 Variable trouble and Mapping

As mentioned in Section 5.1.2, the DeviceNet documentation from Fronius is not accurate compared to what is present on the welding apparatus when it is connected to the PLC. Some parts are similar, but it's difficult to determine their exact placement by just referencing this documentation. Much trial and error was required to identify some of these elements, and others are entirely different. Therefore, finding a mapping inside the Yaskawa robot controller was beneficial. It makes sense that this mapping is correct, as it was the default controller for the welding apparatus before the new control solution was developed in this master thesis. The problem with this mapping lies in the lack of detail regarding variables in the 16-bit word format, which encompasses variables controlling aspects like the job number, welding current, etc., each composed of two distinct bytes. As these are indeed formed from two bytes, they were presumed to represent 16-bit words directly.

However, the mapping from Fronius, located in the operation instructions for the bus card and described the format of the 16-bit words, was unfortunately discovered too late to be implemented into the program. With the EDS file, we only input the "USINT" (unsigned integers) data type with the 80 bytes used by the PLC. Relying solely on the mapping from Yaskawa, we overlooked that several of the set variables are "SINT16" (signed integers) and others are "UINT16" (unsigned integers), where the latter of which corresponds one-to-one with the "WORD" data type. These data types can both be represented by a generic 16-bit Word, as they are merely sequences of 16 bits. Generic Words can be interpreted in various ways depending on the context. For instance, "UINT16" can only represent positive numbers, while "SINT16" can represent both positive and negative numbers. In a different context, such as text encoding, a format like UTF-16 uses a 16-bit Word to represent a single character.

It's possible to incorporate and accommodate this factor in the client-side code that interacts with ROS2 for complete control over the entire system, including the robot arm and welding apparatus, for the variables that require SINT16. However, the optimal solution would be to modify the PLC program so that it is not necessary for this client to consider this, simplifying the development of future solutions.

8 Conclusion and Further works

8.1 Conclusion

The solution developed to address the task outlined in the project description in Section 1.2, has effectively achieved most of the objectives set in the objectives portion of the introduction at Section 1.4. A detailed discussion of the solution presented in Section 3 is provided at Section 7.3.2.

The majority of the primary and secondary objectives stipulated were indeed achieved. The main achievements include:

- Establishing communication between the welding apparatus and a computer.
- Enabling control over the Fronius welding apparatus by setting Boolean variables and selecting the predefined job for the welding task.

Supplementarily, the accomplishment of several secondary objectives is noted:

- Establishment of the connection over Ethernet.
- Investigation into alternative solutions.
- Provision of a step-by-step guide for implementation, as presented in Section 4 and Section 5.
- Achievement of plug-and-play functionality.

Nevertheless, a couple of the objectives were not completely fulfilled:

- The alternative solutions that were researched were not empirically tested.
- Comprehensive control over all the welding parameters was not achieved directly.

In conclusion, a satisfactory solution has been developed for controlling the welding apparatus. This solution is prepared for integration with a system using ROS2 to control a welding robot.

8.2 Further work and possible expansions

8.2.1 Updating the PLC code

The PLC code should be updated to accommodate variables that should be in the SINT16 format, as opposed to their current state as unformatted 16-bit words. This adjustment was the topic of discussion in Section 7.6.3.

8.2.2 Implementing it into ROS2

A logical progression from the solution developed in this master thesis would be creating the OPC UA client for controlling the welding apparatus. This client should be constructed to interact effectively with ROS2, enhancing the control operations. This is the core purpose of the developed solution, to ensure efficient integration with ROS2 via the OPC UA client.

Bibliography

[1] W. Bolton, *Programmable logic controllers.* Newnes, 2015.

[2] 'File:originating register, number five crossbar switching system (museum of communications, seattle).jpg', Wikipedia. (2007), [Online]. Available: https://en.wikipedia.org/wiki/File:Originating_Register,_Number_Five_Crossbar_Switching_System_(Museum_of_Communications,_Seattle).jpg.

[3] M. G. Hudedmani, R. Umayal, S. K. Kabberalli and R. Hittalamani, 'Programmable logic controller (plc) in automation', *Advanced Journal of Graduate Research*, vol. 2, no. 1, pp. 37–45, 2017.

[4] E. R. Alphonsus and M. O. Abdullah, 'A review on the applications of programmable logic controllers (plcs)', *Renewable and Sustainable Energy Reviews*, vol. 60, pp. 1185–1205, 2016.

[5] G. Frey and L. Litz, 'Formal methods in plc programming', in *Smc 2000 conference proceedings. 2000 ieee international conference on systems, man and cybernetics. 'cybernetics evolving to systems, humans, organizations, and their complex interactions' (cat. no.0*, vol. 4, 2000, 2431–2436 vol.4. DOI: 10.1109/ICSMC.2000.884356.

[6] 'Nek iec 61131-3:2013', Standard-Norge. (2013), [Online]. Available: https://www.standard.no/no/Nettbutikk/produktkatalogen/Produktpresentasjon/?ProductID=627454.

[7] IEC, 'Programmable controllers – part 1: General information', en, International Electrotechnical Commission, Geneva, CH, Standard IEC 61131-1:2003, 2003. [Online]. Available: https://webstore.iec.ch/publication/4550.

[8] IEC, 'Programmable controllers – part 3: Programming languages', en, International Electrotechnical Commission, Geneva, CH, Standard IEC 61131-3:2013, 2013. [Online]. Available: https://webstore.iec.ch/publication/31007.

[9] IEC, 'Industrial-process measurement and control – programmable controllers – part 2: Equipment requirements and tests', en, International Electrotechnical Commission, Geneva, CH, Standard IEC 61131-2:2017, 2017. [Online]. Available: https://webstore.iec.ch/publication/31007.

[10] W. Bolton, *Programmable logic controllers.* Newnes, 2015, pp. 12–16.

[11] R. J. M. Rao. 'Difference between compact plc and modular plc', Instrumentation Tools. (), [Online]. Available: https://instrumentationtools.com/difference-between-compact-plc-and-modular-plc/ (visited on 19/05/2023).

[12] Object Management Group (OMG), 'Data distribution service', en, Object Management Group (OMG), Specifications DDS v1.4, 2015. [Online]. Available: https://www.omg.org/spec/DDS/.

[13] 'Why ros?' (2023), [Online]. Available: https://www.ros.org/blog/why-ros/ (visited on 21/02/2023).

[14] Y. Maruyama, S. Kato and T. Azumi, 'Exploring the performance of ros2', in *Proceedings of the 13th International Conference on Embedded Software*, 2016, pp. 1–10.

[15] D. Casini, T. Blaß, I. Lütkebohle and B. Brandenburg, 'Response-time analysis of ros 2 processing chains under reservation-based scheduling', in *31st Euromicro Conference on Real-Time Systems*, Schloss Dagstuhl, 2019, pp. 1–23.

[16] R. Documentation. 'About different middleware vendors'. (2023), [Online]. Available: http://docs.ros.org/en/humble/Concepts/About-Different-Middleware-Vendors.html (visited on 06/06/2023).

[17] R. Documentation. 'Getting started'. (2023), [Online]. Available: https://www.ros.org/blog/getting-started/# (visited on 06/06/2023).

[18] R. Documentation. 'Ros 2 documentation - iron'. (2023), [Online]. Available: https://docs.ros.org/en/iron/ (visited on 06/06/2023).

[19] 'Ros - robot operating system', ROS.org. (2023), [Online]. Available: https://www.ros.org/ (visited on 21/02/2023).

[20] 'Devicenet odva', ODVA. (2023), [Online]. Available: https://www.odva.org/technology-standards/key-technologies/devicenet/ (visited on 19/05/2023).

[21] S. Biegacki and D. VanGompel, 'The application of devicenet in process control', *ISA transactions*, vol. 35, no. 2, pp. 169–176, 1996.

[22] M. Imoto, 'Global standardization activities of devicenet', in *Proceedings of the 41st SICE Annual Conference. SICE 2002.*, Institute of Electrical and Electronics Engineers (IEEE), vol. 2, 2002, pp. 913–916.

[23] 'Devicenet cip on can technology', ODVA. (2021), [Online]. Available: https://www.odva.org/wp-content/uploads/2021/05/PUB00026R5_Tech-Series-DeviceNet.pdf (visited on 19/05/2023).

[24] 'Devicenet – designed for factory automation', CAN in Automation (CiA). (2023), [Online]. Available: https://www.can-cia.org/can-knowledge/hlp/devicenet/ (visited on 24/02/2023).

[25] B. Automation. 'Beckhoff automation'. (2023), [Online]. Available: https://www.beckhoff.com/en-en/company/ (visited on 15/05/2023).

[26] B. Automation. 'Scalable industrial pc solutions'. (2023), [Online]. Available: https://www.beckhoff.com/en-en/products/ipc/#tab_1 (visited on 15/05/2023).

[27] B. Automation. 'Scalable industrial pc solutions'. (2023), [Online]. Available: https://www.beckhoff.com/en-en/products/i-o/ (visited on 15/05/2023).

[28] 'Te1000 — twincat 3 engineering'. (), [Online]. Available: https://www.beckhoff.com/en-en/products/automation/twincat/texxxx-twincat-3-engineering/te1000.html (visited on 10/05/2023).

[29] B. Automation. 'Beckhoff information system - twincat 3'. (2023), [Online]. Available: https://infosys.beckhoff.com/english.php?content=../content/1033/tc3_automationinterface/242682763.html&id= (visited on 15/05/2023).

[30] B. Automation. 'Beckhoff information system - twincat 3 philosophy'. (2023), [Online]. Available: https://infosys.beckhoff.com/english.php?content=../content/1033/tc3_automationinterface/242682763.html&id= (visited on 15/05/2023).

[31] B. Automation. 'Beckhoff information system - twincat 3 basics real-time'. (2023), [Online]. Available: https://infosys.beckhoff.com/english.php?content=../content/1033/tc3_automationinterface/242682763.html&id= (visited on 15/05/2023).

[32] G. Prytz, 'A performance analysis of ethercat and profinet irt', in *2008 IEEE International Conference on Emerging Technologies and Factory Automation*, 2008, pp. 408–415. DOI: 10.1109/ETFA.2008.4638425.

[33] E. T. Group. ‘Ethercat technology group’. (2023), [Online]. Available: https://www.ethercat.org/en/tech_group.html (visited on 16/05/2023).

[34] E. T. Group. ‘Why use ethercat?’ (2023), [Online]. Available: https://www.ethercat.org/en/why_use_ethercat.htm (visited on 16/05/2023).

[35] B. Automation. ‘Beckhoff information system - ethercat general’. (2023), [Online]. Available: https://infosys.beckhoff.com/english.php?content=../content/1033/tf6100_tc3_opcua/1466709259.html&id= (visited on 16/05/2023).

[36] M. Rostan, J. E. Stubbs and D. Dzilno, ‘Ethercat enabled advanced control architecture’, in *2010 IEEE/SEMI Advanced Semiconductor Manufacturing Conference (ASMC)*, 2010, pp. 39–44. DOI: 10.1109/ASMC.2010.5551414.

[37] B. Automation. ‘Industrial pcs for all control requirements’. (2023), [Online]. Available: https://www.beckhoff.com/en-en/products/ipc/pcs/ (visited on 16/05/2023).

[38] B. Automation. ‘Embedded pcs – din rail-mountable industrial pcs’. (2023), [Online]. Available: https://www.beckhoff.com/en-en/products/ipc/embedded-pcs/ (visited on 16/05/2023).

[39] B. Automation. ‘C6930 — control cabinet industrial pc’. (2023), [Online]. Available: https://www.beckhoff.com/en-en/products/ipc/pcs/c69xx-compact-industrial-pcs/c6930.html (visited on 15/05/2023).

[40] B. Automation. ‘Cx8190 — embedded pc with different ethernet protocols’. (2023), [Online]. Available: https://www.beckhoff.com/en-en/products/ipc/embedded-pcs/cx8100-arm-cortex-a9/cx8190.html (visited on 16/05/2023).

[41] B. Automation. ‘Cx7000 — embedded pc series’. (2023), [Online]. Available: https://www.beckhoff.com/en-en/products/ipc/embedded-pcs/cx7000-arm-cortex-m7/ (visited on 16/05/2023).

[42] B. Automation. ‘Cx8100 — embedded pc series’. (2023), [Online]. Available: https://www.beckhoff.com/en-en/products/ipc/embedded-pcs/cx8100-arm-cortex-a9/ (visited on 16/05/2023).

[43] OPC-Foundation, *Welcome to the world of opc*, 2023. [Online]. Available: https://opcfoundation.org.

[44] OPC-Foundation, *Unified architecture*, 2023. [Online]. Available: https://opcfoundation.org/about/opc-technologies/opc-ua/.

[45] T. Hannelius, M. Salmenpera and S. Kuikka, ‘Roadmap to adopting opc ua’, in *2008 6th IEEE International Conference on Industrial Informatics*, 2008, pp. 756–761. DOI: 10.1109/INDIN.2008.4618203.

[46] S.-H. Leitner and W. Mahnke, ‘Opc ua–service-oriented architecture for industrial applications’, *ABB Corporate Research Center*, vol. 48, no. 61-66, p. 22, 2006.

[47] D. Bruckner, M.-P. Stănică, R. Blair *et al.*, ‘An introduction to opc ua tsn for industrial communication systems’, *Proceedings of the IEEE*, vol. 107, no. 6, pp. 1121–1131, 2019. DOI: 10.1109/JPROC.2018.2888703.

[48] S. Schriegel, T. Kobzan and J. Jasperneite, ‘Investigation on a distributed sdn control plane architecture for heterogeneous time sensitive networks’, in *2018 14th IEEE International Workshop on Factory Communication Systems (WFCS)*, 2018, pp. 1–10. DOI: 10.1109/WFCS.2018.8402356.

[49] OPC-Foundation. 'What is opc?' (2023), [Online]. Available: https://opcfoundation.org/about/what-is-opc/.

[50] 'Uaexpert—a full-featured opc ua client', Unified Automation. (), [Online]. Available: https://www.unified-automation.com/products/development-tools/uaexpert.html (visited on 19/05/2023).

[51] 'Free opc-ua library', GitHub. (), [Online]. Available: https://github.com/FreeOpcUa (visited on 19/05/2023).

[52] 'Opcua-asyncio', GitHub. (), [Online]. Available: https://github.com/FreeOpcUa/opcua-asyncio (visited on 19/05/2023).

[53] MQTT.org, *Mqtt specifications*, 2023. [Online]. Available: https://mqtt.org/mqtt-specification/.

[54] MQTT.org, *Mqtt faq*, 2023. [Online]. Available: https://mqtt.org/faq/.

[55] MQTT.org, *Mqtt: The standard for iot messaging*, 2023. [Online]. Available: https://mqtt.org/.

[56] A. Banks, E. Briggs, K. Borgendale and R. Gupta, Eds. 'Mqtt version 5.0'. Latest version: https://docs.oasis-open.org/mqtt/mqtt/v5.0/mqtt-v5.0.html, OASIS Standard. (Mar. 2019), [Online]. Available: https://docs.oasis-open.org/mqtt/mqtt/v5.0/os/mqtt-v5.0-os.html.

[57] D. Thangavel, X. Ma, A. Valera, H.-X. Tan and C. K.-Y. Tan, 'Performance evaluation of mqtt and coap via a common middleware', in *2014 IEEE Ninth International Conference on Intelligent Sensors, Sensor Networks and Information Processing (ISSNIP)*, 2014, pp. 1–6. DOI: 10.1109/ISSNIP.2014.6827678.

[58] R. A. Light, 'Mosquitto: Server and client implementation of the mqtt protocol', *Journal of Open Source Software*, vol. 2, no. 13, p. 265, 2017.

[59] 'Fronius international gmbh: Welding technology'. (), [Online]. Available: https://www.fronius.com/en/welding-technology (visited on 03/05/2023).

[60] F. I. GmbH, *Operating instructions ri fb inside/i*, 42,0410,1912, 2021.

[61] 'Fronius international gmbh: Welding with the tps/i. the intelligent mig/mag welding system'. (), [Online]. Available: https://www.fronius.com/en/welding-technology/product-information/mig-mag-welding-system-tpsi (visited on 04/05/2023).

[62] F. I. GmbH, *Operating instructions tps320i/400i/500i/600i tps 400i lsc adv*, 42,0426,0114,EN, 2022.

[63] 'Fronius international gmbh: Intelligent welding process management as a competitive advantage'. (), [Online]. Available: https://www.fronius.com/en/welding-technology/info-centre/press/intelligent-welding-process-management-170521 (visited on 04/05/2023).

[64] 'Fronius international gmbh: Search and find, info bussystem cd'. (), [Online]. Available: www.fronius.com/Tools-BusSystems (visited on 04/05/2023).

[65] 'About yaskawa motoman robotics', Yaskawa Motoman. (2023), [Online]. Available: https://www.motoman.com/en-us/about/company (visited on 09/06/2023).

[66] 'Robotics', Yaskawa Motoman. (2023), [Online]. Available: https://www.motoman.com/en-us/about/company/robotics (visited on 09/06/2023).

[67] 'Welding and cutting robots', Yaskawa. (2023), [Online]. Available: https://www.yaskawa.eu.com/products/robots/welding-cutting (visited on 09/06/2023).

[68] Motoman. 'Controller functions', Yaskawa. (2023), [Online]. Available: https://www.motoman.com/en-us/products/programming/controllers/controller-functions (visited on 09/06/2023).

[69] Motoman. 'Arc welding', Yaskawa. (2023), [Online]. Available: https://www.motoman.com/en-us/applications/arc-welding (visited on 09/06/2023).

[70] 'Yrc1000', Yaskawa. (2023), [Online]. Available: https://www.yaskawa.eu.com/products/robots/controller/productdetail/product/yrc1000_583 (visited on 09/06/2023).

[71] *Yrc1000 options instructions for inform language*, Manual No. RE-CKI-A466 3, Yaskawa, 2023.

[72] Beckhoff, *Connecting the power supply*, 2023. [Online]. Available: https://infosys.beckhoff.com/english.php?content=../content/1033/ipcinfosys/index.html&id=444084617613099609.

[73] 'Tf6100 — twincat 3 opc ua'. (), [Online]. Available: https://www.beckhoff.com/en-en/products/automation/twincat/tfxxxx-twincat-3-functions/tf6xxx-tc3-connectivity/tf6100.html (visited on 10/05/2023).

[74] 'Starting a remote connection', Beckhoff Automation. (), [Online]. Available: https://infosys.beckhoff.com/english.php?content=../content/1033/cx51x0_hw/5046749835.html&id= (visited on 19/05/2023).

[75] 'Beckhoff information system', Beckhoff Automation. (), [Online]. Available: https://infosys.beckhoff.com/index_en.htm (visited on 28/05/2023).

[76] 'Ez-eds download', ODVA. (), [Online]. Available: https://www.odva.org/subscriptions-services/additional-tools/ez-eds-download/ (visited on 19/05/2023).

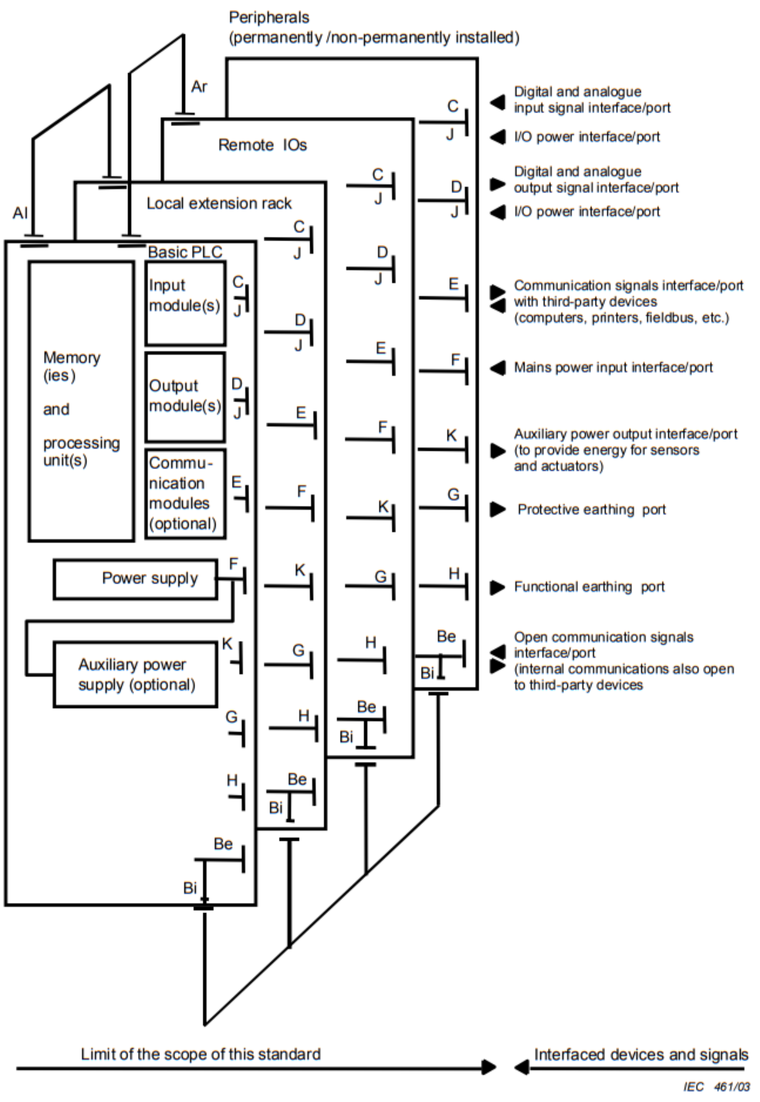
[77] F. I. GmbH, *Operating instructions devicenet*, 42,0410,0635, 2022.

[78] F. I. GmbH, *Operating instructions ri fb inside/i; ri mod/i cc-m40 devicenet*, 42,0410,1914, 2022.

[79] 'Opcua-asyncio examples', GitHub. (), [Online]. Available: https://github.com/FreeOpcUa/opcua-asyncio/tree/master/examples (visited on 19/05/2023).

Appendix

A PLC architecture from IEC 61131-2


Peripherals
(permanently /non-permanently installed)
Ar
Al
Remote IOs
Local extension rack
Basic PLC
Memory (ies) and processing unit(s)
Input module(s)
Output module(s)
Commu-nication modules (optional)
Power supply
Auxiliary power supply (optional)
C
D
E
F
G
H
J
K
Be
Bi
Digital and analogue input signal interface/port
I/O power interface/port
Digital and analogue output signal interface/port
I/O power interface/port
Communication signals interface/port with third-party devices (computers, printers, fieldbus, etc.)
Mains power input interface/port
Auxiliary power output interface/port (to provide energy for sensors and actuators)
Protective earthing port
Functional earthing port
Open communication signals interface/port (internal communications also open to third-party devices
Limit of the scope of this standard
Interfaced devices and signals
IEC 461/03

Key

Al	Communication interface/port for local I/O
Ar	Communication interface/port for remote I/O station
Be	Open-communication interface/port also open to third-party devices (for example, personal computer used for programming instead of a PADT)
Bi	Internal communication interface/port for peripherals
C	Interface/port for digital and analogue input signals
D	Interface/port for digital and analogue output signals
E	Serial or parallel communication interfaces/ports for data communication with third-party devices
F	Mains power interface/port. Devices with F ports have requirements on keeping downstream devices intelligent during power-up, power-down and power interruptions.
G	Port for protective earthing
H	Port for functional earthing
J	I/O power interface/port used to power sensors and actuators
K	Auxiliary power output interface/port

Figure 54: Typical interface/port diagram of a PLC-system (from IEC 61131-2)[9]

B The Complete I/O Tables

The complete mapping from the Yaskawa robot controller of the DeviceNet I/O of the Fronius welding machine TPS400/i. It has the two tables Table 2 and Table 5 from Section 5.1.2, to have a complete mapping in one place.

Table 14: The Output mapping of the ten first Bytes for the Fronius Welding Machine, found in the Yaskawa Robot controller.

Output					
Odd Bytes			Even Bytes		
Byte	Bit	Description	Byte	Bit	Description
1	1	Welding start	2	1	Gas On
	2	Robot ready		2	Wire forward
	3	WorkingModeB0		3	Wire backward
	4	WorkingModeB1		4	Error quit
	5	WorkingModeB2		5	Touch sensing
	6	WorkingModeB3		6	Torch blow out
	7	WorkingModeB4		7	ProcessLine b1
	8	Reserved		8	ProcessLine b2
3	1	Welding Sim	4	1	Reserved
	2	SynchroPulseOn		2	Teach mode
	3	Reserved		3	Reserved
	4	Reserved		4	ActiveHeatCont
	5	Reserved		5	Reserved
	6	Reserved		6	Reserved
	7	Wire break on		7	Reserved
	8	Torch Xchange		8	Reserved
5	1	Twin Mode b1	6	1	Reserved
	2	Twin Mode b2		2	Reserved
	3	Reserved		3	Reserved
	4	Reserved		4	Reserved
	5	Reserved		5	Reserved
	6	Document Mode		6	Reserved
	7	Reserved		7	Reserved
	8	Reserved		8	DisableProcesCo
7	1	Reserved	8	1	Ext Input 1
	2	Reserved		2	Ext Input 2
	3	Reserved		3	Ext Input 3
	4	Reserved		4	Ext Input 4
	5	Reserved		5	Ext Input 5
	6	Reserved		6	Ext Input 6
	7	Reserved		7	Ext Input 7
	8	Reserved		8	Ext Input 8
9	1	Job nr	10	1	Job nr
	2	Low Byte		2	High Byte
	3	M183		3	
	4			4	
	5			5	
	6			6	
	7			7	
	8			8	

Table 15: The Output mapping of Byte 11-20 for the Fronius Welding Machine, found in the Yaskawa Robot controller.

Output					
Odd Bytes			Even Bytes		
Byte	Bit	Description	Byte	Bit	Description
11	1	Wire Feed Speed	12	1	Wire Feed Speed
	2	Low Byte		2	High Byte
	3	M560		3	
	4			4	
	5			5	
	6			6	
	7			7	
	8			8	
13	1	Arc length Cor	14	1	Arc length Cor
	2	Low Byte		2	High Byte
	3	M561		3	
	4			4	
	5			5	
	6			6	
	7			7	
	8			8	
15	1	Pulse/dyn cor	16	1	Pulse/dyn cor
	2	Low Byte		2	High Byte
	3	M562		3	
	4			4	
	5			5	
	6			6	
	7			7	
	8			8	
17	1	Wire Retract Cor	18	1	Wire Retract Cor
	2	Low Byte		2	High Byte
	3			3	
	4			4	
	5			5	
	6			6	
	7			7	
	8			8	
19	1	Welding Speed	20	1	Welding Speed
	2	Low Byte		2	High Byte
	3			3	
	4			4	
	5			5	
	6			6	
	7			7	
	8			8	

Table 16: The Output mapping of Byte 21-30 for the Fronius Welding Machine, found in the Yaskawa Robot controller.

Output					
Odd Bytes			Even Bytes		
Byte	Bit	Description	Byte	Bit	Description
21	1	Process Cont Cor	22	1	Process Cont Cor
	2	Low Byte		2	High Byte
	3			3	
	4			4	
	5			5	
	6			6	
	7			7	
	8			8	
23	1	Reserved	24	1	Reserved
	2	Reserved		2	Reserved
	3	Reserved		3	Reserved
	4	Reserved		4	Reserved
	5	Reserved		5	Reserved
	6	Reserved		6	Reserved
	7	Reserved		7	Reserved
	8	Reserved		8	Reserved
25	1	Reserved	26	1	Reserved
	2	Reserved		2	Reserved
	3	Reserved		3	Reserved
	4	Reserved		4	Reserved
	5	Reserved		5	Reserved
	6	Reserved		6	Reserved
	7	Reserved		7	Reserved
	8	Reserved		8	Reserved
27	1	Reserved	28	1	Reserved
	2	Reserved		2	Reserved
	3	Reserved		3	Reserved
	4	Reserved		4	Reserved
	5	Reserved		5	Reserved
	6	Reserved		6	Reserved
	7	Reserved		7	Reserved
	8	Reserved		8	Reserved
29	1	Reserved	30	1	Reserved
	2	Reserved		2	Reserved
	3	Reserved		3	Reserved
	4	Reserved		4	Reserved
	5	Reserved		5	Reserved
	6	Reserved		6	Reserved
	7	Reserved		7	Reserved
	8	Reserved		8	Reserved

Table 17: The Output mapping of Byte 31-40 for the Fronius Welding Machine, found in the Yaskawa Robot controller.

Output					
Odd Bytes			Even Bytes		
Byte	Bit	Description	Byte	Bit	Description
31	1	Reserved	32	1	Reserved
	2	Reserved		2	Reserved
	3	Reserved		3	Reserved
	4	Reserved		4	Reserved
	5	Reserved		5	Reserved
	6	Reserved		6	Reserved
	7	Reserved		7	Reserved
	8	Reserved		8	Reserved
33	1	Reserved	34	1	Reserved
	2	Reserved		2	Reserved
	3	Reserved		3	Reserved
	4	Reserved		4	Reserved
	5	Reserved		5	Reserved
	6	Reserved		6	Reserved
	7	Reserved		7	Reserved
	8	Reserved		8	Reserved
35	1	Reserved	36	1	Reserved
	2	Reserved		2	Reserved
	3	Reserved		3	Reserved
	4	Reserved		4	Reserved
	5	Reserved		5	Reserved
	6	Reserved		6	Reserved
	7	Reserved		7	Reserved
	8	Reserved		8	Reserved
37	1	Reserved	38	1	Reserved
	2	Reserved		2	Reserved
	3	Reserved		3	Reserved
	4	Reserved		4	Reserved
	5	Reserved		5	Reserved
	6	Reserved		6	Reserved
	7	Reserved		7	Reserved
	8	Reserved		8	Reserved
39	1	Seam Number	40	1	Seam Number
	2	Low Byte		2	High Byte
	3			3	
	4			4	
	5			5	
	6			6	
	7			7	
	8			8	

Table 18: The Input mapping of the ten first Bytes for the Fronius Welding Machine, found in the Yaskawa Robot controller.

Input					
Odd Bytes			Even Bytes		
Byte	Bit	Description	Byte	Bit	Description
1	1	Heartbeat	2	1	Collisionbox
	2	Welder Ready		2	Robot Mot Rel
	3	Reserved		3	Short Circuit T0
	4	Process active		4	Reserved
	5	Current flow		5	Reserved
	6	Arc stable		6	Prm Sel Int
	7	Main current		7	Character num
	8	Touch signal		8	Torch Body Gripp
3	1	CommandValu UoR	4	1	Sensor Status 1
	2	Correction UoR		2	Sensor Status 2
	3	Reserved		3	Sensor Status 3
	4	Limitsignal		4	Reserved
	5	Reserved		5	Reserved
	6	Reserved		6	Reserved
	7	Main Supply Stat		7	Reserved
	8	Reserved		8	Reserved
5	1	Reserved	6	1	Reserved
	2	Reserved		2	Reserved
	3	Reserved		3	Reserved
	4	Reserved		4	Reserved
	5	Reserved		5	Reserved
	6	Reserved		6	Reserved
	7	Reserved		7	Reserved
	8	Reserved		8	Reserved
7	1	Process bit 0	8	1	Ext Output 1
	2	Process bit 1		2	Ext Output 2
	3	Process bit 2		3	Ext Output 3
	4	Process bit 3		4	Ext Output 4
	5	Process bit 4		5	Ext Output 5
	6	Reserved		6	Ext Output 6
	7	Reserved		7	Ext Output 7
	8	Twin Sync Active		8	Ext Output 8
9	1	Welding Voltage	10	1	Welding Voltage
	2	Low Byte		2	High Byte
	3	M180		3	
	4			4	
	5			5	
	6			6	
	7			7	
	8			8	

Table 19: The Input mapping of Byte 11-20 for the Fronius Welding Machine, found in the Yaskawa Robot controller.

Input					
Odd Bytes			Even Bytes		
Byte	Bit	Description	Byte	Bit	Description
11	1	Welding Current	12	1	Welding Current
	2	Low Byte		2	High Byte
	3	M181		3	
	4			4	
	5			5	
	6			6	
	7			7	
	8			8	
13	1	Wire Feed Speed	14	1	Wire Feed Speed
	2	Low Byte		2	High Byte
	3	M182		3	
	4			4	
	5			5	
	6			6	
	7			7	
	8			8	
15	1	Seam Tracking	16	1	Seam Tracking
	2	Low Byte		2	High Byte
	3	M185		3	
	4			4	
	5			5	
	6			6	
	7			7	
	8			8	
17	1	Errornumber	18	1	Errornumber
	2	Low Byte		2	High Byte
	3	M184		3	
	4			4	
	5			5	
	6			6	
	7			7	
	8			8	
19	1	Reserved	20	1	Reserved
	2	Reserved		2	Reserved
	3	Reserved		3	Reserved
	4	Reserved		4	Reserved
	5	Reserved		5	Reserved
	6	Reserved		6	Reserved
	7	Reserved		7	Reserved
	8	Reserved		8	Reserved

Table 20: The Input mapping of Byte 21-30 for the Fronius Welding Machine, found in the Yaskawa Robot controller.

Input					
Odd Bytes			Even Bytes		
Byte	Bit	Description	Byte	Bit	Description
21	1	Motor Current M1	22	1	Motor Current M1
	2	Low Byte		2	High Byte
	3	M186		3	
	4			4	
	5			5	
	6			6	
	7			7	
	8			8	
23	1	Motor Current M2	24	1	Motor Current M2
	2	Low Byte		2	High Byte
	3	M187		3	
	4			4	
	5			5	
	6			6	
	7			7	
	8			8	
25	1	Motor Current M3	26	1	Motor Current M3
	2	Low Byte		2	High Byte
	3	M188		3	
	4			4	
	5			5	
	6			6	
	7			7	
	8			8	
27	1	Reserved	28	1	Reserved
	2	Reserved		2	Reserved
	3	Reserved		3	Reserved
	4	Reserved		4	Reserved
	5	Reserved		5	Reserved
	6	Reserved		6	Reserved
	7	Reserved		7	Reserved
	8	Reserved		8	Reserved
29	1	Reserved	30	1	Reserved
	2	Reserved		2	Reserved
	3	Reserved		3	Reserved
	4	Reserved		4	Reserved
	5	Reserved		5	Reserved
	6	Reserved		6	Reserved
	7	Reserved		7	Reserved
	8	Reserved		8	Reserved

Table 21: The Input mapping of Byte 31-40 for the Fronius Welding Machine, found in the Yaskawa Robot controller.

Input					
Odd Bytes			Even Bytes		
Byte	Bit	Description	Byte	Bit	Description
31	1	Reserved	32	1	Reserved
	2	Reserved		2	Reserved
	3	Reserved		3	Reserved
	4	Reserved		4	Reserved
	5	Reserved		5	Reserved
	6	Reserved		6	Reserved
	7	Reserved		7	Reserved
	8	Reserved		8	Reserved
33	1	Reserved	34	1	Reserved
	2	Reserved		2	Reserved
	3	Reserved		3	Reserved
	4	Reserved		4	Reserved
	5	Reserved		5	Reserved
	6	Reserved		6	Reserved
	7	Reserved		7	Reserved
	8	Reserved		8	Reserved
35	1	Reserved	36	1	Reserved
	2	Reserved		2	Reserved
	3	Reserved		3	Reserved
	4	Reserved		4	Reserved
	5	Reserved		5	Reserved
	6	Reserved		6	Reserved
	7	Reserved		7	Reserved
	8	Reserved		8	Reserved
37	1	Reserved	38	1	Reserved
	2	Reserved		2	Reserved
	3	Reserved		3	Reserved
	4	Reserved		4	Reserved
	5	Reserved		5	Reserved
	6	Reserved		6	Reserved
	7	Reserved		7	Reserved
	8	Reserved		8	Reserved
39	1	Reserved	40	1	Reserved
	2	Reserved		2	Reserved
	3	Reserved		3	Reserved
	4	Reserved		4	Reserved
	5	Reserved		5	Reserved
	6	Reserved		6	Reserved
	7	Reserved		7	Reserved
	8	Reserved		8	Reserved

C PLC Code

C.1 Main

Main definitions:

```
PROGRAM MAIN
VAR
  // Initialising the IO
  froniusOutputs : Fronius_Outputs;
  froniusInputs : Fronius_Inputs;


END_VAR
VAR CONSTANT
  ARRAY_LENGTH : UDINT := 40;
END_VAR
```

Main code:

```
froniusOutputs();
froniusInputs();
```

C.2 Fronius_Inputs Function Block

C.2.1 The definitions for Input Function Block:

```
FUNCTION_BLOCK Fronius_Inputs
VAR_INPUT
END_VAR
VAR_OUTPUT
END_VAR
VAR
  {attribute 'OPC.UA.DA' := '1'}
  {attribute 'OPC.UA.DA.Access' := '1'}
  iArray AT %I*: ARRAY[1..ARRAY_LENGTH] OF BYTE;
  Reserved : BOOL;

  // Byte 1
  {attribute 'OPC.UA.DA' := '1'}
  {attribute 'OPC.UA.DA.Access' := '1'}
  Heartbeat : BOOL;        //Bit 1
  {attribute 'OPC.UA.DA' := '1'}
  {attribute 'OPC.UA.DA.Access' := '1'}
  WelderReady : BOOL;        //Bit 2
  {attribute 'OPC.UA.DA' := '1'}
  {attribute 'OPC.UA.DA.Access' := '1'}
  ProcessActive : BOOL;      //Bit 4
  {attribute 'OPC.UA.DA' := '1'}
  {attribute 'OPC.UA.DA.Access' := '1'}
  CurrentFlow : BOOL;        //Bit 5
  {attribute 'OPC.UA.DA' := '1'}
  {attribute 'OPC.UA.DA.Access' := '1'}
  ArcStable : BOOL;        //Bit 6
  {attribute 'OPC.UA.DA' := '1'}
```

```
{attribute 'OPC.UA.DA.Access' := '1'}
MainCurrent : BOOL;       //Bit 7
{attribute 'OPC.UA.DA' := '1'}
{attribute 'OPC.UA.DA.Access' := '1'}
TouchSignal : BOOL;       //Bit 8

// Byte 2
{attribute 'OPC.UA.DA' := '1'}
{attribute 'OPC.UA.DA.Access' := '1'}
Collisionbox : BOOL;      //Bit 1
{attribute 'OPC.UA.DA' := '1'}
{attribute 'OPC.UA.DA.Access' := '1'}
RobotMotRel : BOOL;       //Bit 2
{attribute 'OPC.UA.DA' := '1'}
{attribute 'OPC.UA.DA.Access' := '1'}
ShortCircuitTO : BOOL;      //Bit 3
{attribute 'OPC.UA.DA' := '1'}
{attribute 'OPC.UA.DA.Access' := '1'}
PrmSelInt : BOOL;       //Bit 6
{attribute 'OPC.UA.DA' := '1'}
{attribute 'OPC.UA.DA.Access' := '1'}
CharacterNum : BOOL;      //Bit 7
{attribute 'OPC.UA.DA' := '1'}
{attribute 'OPC.UA.DA.Access' := '1'}
TorchBodyGripp : BOOL;      //Bit 8

// Byte 3
{attribute 'OPC.UA.DA' := '1'}
{attribute 'OPC.UA.DA.Access' := '1'}
CommandValu_UoR : BOOL;     //Bit 1
{attribute 'OPC.UA.DA' := '1'}
{attribute 'OPC.UA.DA.Access' := '1'}
Correction_UoR : BOOL;      //Bit 2
{attribute 'OPC.UA.DA' := '1'}
{attribute 'OPC.UA.DA.Access' := '1'}
Limitsignal : BOOL;       //Bit 4
{attribute 'OPC.UA.DA' := '1'}
{attribute 'OPC.UA.DA.Access' := '1'}
MainSupplyStat : BOOL;      //Bit 7

// Byte 4
{attribute 'OPC.UA.DA' := '1'}
{attribute 'OPC.UA.DA.Access' := '1'}
SensorStatus1 : BOOL;     //Bit 1
{attribute 'OPC.UA.DA' := '1'}
{attribute 'OPC.UA.DA.Access' := '1'}
SensorStatus2 : BOOL;     //Bit 2
{attribute 'OPC.UA.DA' := '1'}
{attribute 'OPC.UA.DA.Access' := '1'}
SensorStatus3 : BOOL;     //Bit 3


// Byte 7
{attribute 'OPC.UA.DA' := '1'}
{attribute 'OPC.UA.DA.Access' := '1'}
ProcessBit0 : BOOL;       //Bit 1
{attribute 'OPC.UA.DA' := '1'}
{attribute 'OPC.UA.DA.Access' := '1'}
```

```
ProcessBit1 : BOOL;        //Bit 2
{attribute 'OPC.UA.DA' := '1'}
{attribute 'OPC.UA.DA.Access' := '1'}
ProcessBit2 : BOOL;        //Bit 3
{attribute 'OPC.UA.DA' := '1'}
{attribute 'OPC.UA.DA.Access' := '1'}
ProcessBit3 : BOOL;        //Bit 4
{attribute 'OPC.UA.DA' := '1'}
{attribute 'OPC.UA.DA.Access' := '1'}
ProcessBit4 : BOOL;        //Bit 5
{attribute 'OPC.UA.DA' := '1'}
{attribute 'OPC.UA.DA.Access' := '1'}
TwinSyncActive : BOOL;       //Bit 8

(*
//Byte 8
{attribute 'OPC.UA.DA' := '1'}
ExtOutput1 : BOOL;         //Bit 1
{attribute 'OPC.UA.DA' := '1'}
ExtOutput2 : BOOL;         //Bit 2
{attribute 'OPC.UA.DA' := '1'}
ExtOutput3 : BOOL;         //Bit 3
{attribute 'OPC.UA.DA' := '1'}
ExtOutput4 : BOOL;         //Bit 4
{attribute 'OPC.UA.DA' := '1'}
ExtOutput5 : BOOL;         //Bit 5
{attribute 'OPC.UA.DA' := '1'}
ExtOutput6 : BOOL;         //Bit 6
{attribute 'OPC.UA.DA' := '1'}
ExtOutput7 : BOOL;         //Bit 7
{attribute 'OPC.UA.DA' := '1'}
ExtOutput8 : BOOL;         //Bit 8
*)

// All the Words
{attribute 'OPC.UA.DA' := '1'}
{attribute 'OPC.UA.DA.Access' := '1'}
WeldingVoltage  : WORD;
{attribute 'OPC.UA.DA' := '1'}
{attribute 'OPC.UA.DA.Access' := '1'}
WeldingCurrent  : WORD;
{attribute 'OPC.UA.DA' := '1'}
{attribute 'OPC.UA.DA.Access' := '1'}
WireFeedSpeed   : WORD;
{attribute 'OPC.UA.DA' := '1'}
{attribute 'OPC.UA.DA.Access' := '1'}
SeamTracking  : WORD;
{attribute 'OPC.UA.DA' := '1'}
{attribute 'OPC.UA.DA.Access' := '1'}
ErrorNumber   : WORD;
{attribute 'OPC.UA.DA' := '1'}
{attribute 'OPC.UA.DA.Access' := '1'}
MotorCurrentM1  : WORD;
{attribute 'OPC.UA.DA' := '1'}
{attribute 'OPC.UA.DA.Access' := '1'}
MotorCurrentM2  : WORD;
{attribute 'OPC.UA.DA' := '1'}
{attribute 'OPC.UA.DA.Access' := '1'}
```

```
  MotorCurrentM3   : WORD;


END_VAR
VAR CONSTANT
  ARRAY_LENGTH : UDINT := 40;
  //All the indexes to the words, they are the index for the Low Byte
  //High Byte index is Low Byte index +1
  WeldingVoltageIndex   : BYTE := 9;
  WeldingCurrentIndex   : BYTE := 11;
  WireFeedSpeedIndex    : BYTE := 13;
  SeamTrackingIndex     : BYTE := 15;
  ErrorNumberIndex    : BYTE := 17;
  MotorCurrentM1Index   : BYTE := 21;
  MotorCurrentM2Index   : BYTE := 23;
  MotorCurrentM3Index   : BYTE := 25;
END_VAR
```

C.2.2 The executing Code for Inputs Function Block:

```
//Bits from Byte 1
Heartbeat      := BYTE_TO_BOOL(SHR(iArray[1], 0) AND 1);
WelderReady    := BYTE_TO_BOOL(SHR(iArray[1], 1) AND 1);
Reserved     := BYTE_TO_BOOL(SHR(iArray[1], 2) AND 1);
ProcessActive    := BYTE_TO_BOOL(SHR(iArray[1], 3) AND 1);
CurrentFlow    := BYTE_TO_BOOL(SHR(iArray[1], 4) AND 1);
ArcStable      := BYTE_TO_BOOL(SHR(iArray[1], 5) AND 1);
MainCurrent    := BYTE_TO_BOOL(SHR(iArray[1], 6) AND 1);
TouchSignal    := BYTE_TO_BOOL(SHR(iArray[1], 7) AND 1);

//Bits from Byte 2
Collisionbox   := BYTE_TO_BOOL(SHR(iArray[2], 0) AND 1);
RobotMotRel    := BYTE_TO_BOOL(SHR(iArray[2], 1) AND 1);
ShortCircuitTO   := BYTE_TO_BOOL(SHR(iArray[2], 2) AND 1);
PrmSelInt    := BYTE_TO_BOOL(SHR(iArray[2], 5) AND 1);
CharacterNum   := BYTE_TO_BOOL(SHR(iArray[2], 6) AND 1);
TorchBodyGripp   := BYTE_TO_BOOL(SHR(iArray[2], 7) AND 1);

//Bits from Byte 3
CommandValu_UoR  := BYTE_TO_BOOL(SHR(iArray[3], 0) AND 1);
Correction_UoR   := BYTE_TO_BOOL(SHR(iArray[3], 1) AND 1);
Limitsignal    := BYTE_TO_BOOL(SHR(iArray[3], 3) AND 1);
MainSupplyStat   := BYTE_TO_BOOL(SHR(iArray[3], 6) AND 1);

//Bits from Byte 4
SensorStatus1    := BYTE_TO_BOOL(SHR(iArray[4], 0) AND 1);
SensorStatus2  := BYTE_TO_BOOL(SHR(iArray[4], 1) AND 1);
SensorStatus3  := BYTE_TO_BOOL(SHR(iArray[4], 2) AND 1);

//Bits from Byte 7
ProcessBit0    := BYTE_TO_BOOL(SHR(iArray[7], 0) AND 1);
ProcessBit1    := BYTE_TO_BOOL(SHR(iArray[7], 1) AND 1);
ProcessBit2    := BYTE_TO_BOOL(SHR(iArray[7], 2) AND 1);
ProcessBit3    := BYTE_TO_BOOL(SHR(iArray[7], 3) AND 1);
ProcessBit4    := BYTE_TO_BOOL(SHR(iArray[7], 4) AND 1);
TwinSyncActive   := BYTE_TO_BOOL(SHR(iArray[7], 7) AND 1);

```

```
// Creating the words by multipling the high byte by 256 and adding the low byte
WeldingVoltage  := (BYTE_TO_WORD(iArray[WeldingVoltageIndex+1]) * 16#0100) +
    BYTE_TO_WORD(iArray[WeldingVoltageIndex]);
WeldingCurrent  := (BYTE_TO_WORD(iArray[WeldingCurrentIndex+1]) * 16#0100) +
    BYTE_TO_WORD(iArray[WeldingCurrentIndex]);
WireFeedSpeed   := (BYTE_TO_WORD(iArray[WireFeedSpeedIndex+1]) * 16#0100) +
    BYTE_TO_WORD(iArray[WireFeedSpeedIndex]);
SeamTracking  := (BYTE_TO_WORD(iArray[SeamTrackingIndex+1]) * 16#0100) +
    BYTE_TO_WORD(iArray[SeamTrackingIndex]);
ErrorNumber   := (BYTE_TO_WORD(iArray[ErrorNumberIndex+1]) * 16#0100) +
    BYTE_TO_WORD(iArray[ErrorNumberIndex]);
MotorCurrentM1  := (BYTE_TO_WORD(iArray[MotorCurrentM1Index+1]) * 16#0100) +
    BYTE_TO_WORD(iArray[MotorCurrentM1Index]);
MotorCurrentM2  := (BYTE_TO_WORD(iArray[MotorCurrentM2Index+1]) * 16#0100) +
    BYTE_TO_WORD(iArray[MotorCurrentM2Index]);
MotorCurrentM3  := (BYTE_TO_WORD(iArray[MotorCurrentM3Index+1]) * 16#0100) +
    BYTE_TO_WORD(iArray[MotorCurrentM3Index]);
```

C.3 Fronius_Outputs Function Block

C.3.1 The definitions for Output Function Block:

```
FUNCTION_BLOCK Fronius_Outputs
VAR_INPUT
END_VAR
VAR_OUTPUT
END_VAR
VAR
  {attribute 'OPC.UA.DA' := '1'}
  qArray AT %Q*: ARRAY[1..ARRAY_LENGTH] OF BYTE;

  // Byte 1
  qByte1_Set_Type : BOOL := TRUE; (*True setting with the boolean variables, False
      set in qArray *)
  {attribute 'OPC.UA.DA' := '1'}
  Welding_start : BOOL;     //Bit 1
  {attribute 'OPC.UA.DA' := '1'}
  Robot_ready : BOOL;       //Bit 2
  {attribute 'OPC.UA.DA' := '1'}
  WorkingModeB0 : BOOL;     //Bit 3
  {attribute 'OPC.UA.DA' := '1'}
  WorkingModeB1 : BOOL;     //Bit 4
  {attribute 'OPC.UA.DA' := '1'}
  WorkingModeB2 : BOOL;     //Bit 5
  {attribute 'OPC.UA.DA' := '1'}
  WorkingModeB3 : BOOL;     //Bit 6
  {attribute 'OPC.UA.DA' := '1'}
  WorkingModeB4 : BOOL;     //Bit 7

  //Byte 2
  {attribute 'OPC.UA.DA' := '1'}
  GasOn : BOOL;         //Bit 1
  {attribute 'OPC.UA.DA' := '1'}
  WireForward : BOOL;       //Bit 2
  {attribute 'OPC.UA.DA' := '1'}
  WireBackward : BOOL;      //Bit 3
```

```
{attribute 'OPC.UA.DA' := '1'}
ErrorQuit : BOOL;        //Bit 4
{attribute 'OPC.UA.DA' := '1'}
TouchSensing : BOOL;       //Bit 5
{attribute 'OPC.UA.DA' := '1'}
TorchBlowOut : BOOL;       //Bit 6
{attribute 'OPC.UA.DA' := '1'}
ProcessLineB1 : BOOL;      //Bit 7
{attribute 'OPC.UA.DA' := '1'}
ProcessLineB2 : BOOL;      //Bit 8

// Byte 3
{attribute 'OPC.UA.DA' := '1'}
WeldingSim : BOOL;         //Bit 1
{attribute 'OPC.UA.DA' := '1'}
SynchroPulseOn : BOOL;       //Bit 2
{attribute 'OPC.UA.DA' := '1'}
WireBreakOn : BOOL;        //Bit 7
{attribute 'OPC.UA.DA' := '1'}
TorchXchange : BOOL;       //Bit 8

//Byte 4
{attribute 'OPC.UA.DA' := '1'}
TeachMode : BOOL;        //Bit 2
{attribute 'OPC.UA.DA' := '1'}
ActiveHeatCont : BOOL;       //Bit 4

// Byte 5
{attribute 'OPC.UA.DA' := '1'}
TwinModeB1 : BOOL;         //Bit 1
{attribute 'OPC.UA.DA' := '1'}
TwinModeB2 : BOOL;         //Bit 2
{attribute 'OPC.UA.DA' := '1'}
DocumentMode : BOOL;       //Bit 6

// Byte 6
{attribute 'OPC.UA.DA' := '1'}
DisableProcesCo : BOOL;      //Bit 8

(*
//Byte 8
{attribute 'OPC.UA.DA' := '1'}
ExtInput1 : BOOL;        //Bit 1
{attribute 'OPC.UA.DA' := '1'}
ExtInput2 : BOOL;        //Bit 2
{attribute 'OPC.UA.DA' := '1'}
ExtInput3 : BOOL;        //Bit 3
{attribute 'OPC.UA.DA' := '1'}
ExtInput4 : BOOL;        //Bit 4
{attribute 'OPC.UA.DA' := '1'}
ExtInput5 : BOOL;        //Bit 5
{attribute 'OPC.UA.DA' := '1'}
ExtInput6 : BOOL;        //Bit 6
{attribute 'OPC.UA.DA' := '1'}
ExtInput7 : BOOL;        //Bit 7
{attribute 'OPC.UA.DA' := '1'}
ExtInput8 : BOOL;        //Bit 8
*)
```

```

  // The words that can be set
  {attribute 'OPC.UA.DA' := '1'}
  Job_nr : WORD; (*Sets the Job number if in Job mode in range 1-1000. Over 1000 it
       is still 1000 *)
  {attribute 'OPC.UA.DA' := '1'}
  WireFeedSpeed : WORD;
  {attribute 'OPC.UA.DA' := '1'}
  ArcLengthCor : WORD;
  {attribute 'OPC.UA.DA' := '1'}
  PulseDynCor : WORD;
  {attribute 'OPC.UA.DA' := '1'}
  WireRetractCor : WORD;
  {attribute 'OPC.UA.DA' := '1'}
  WeldingSpeed : WORD;
  {attribute 'OPC.UA.DA' := '1'}
  ProcessContCor : WORD;
  {attribute 'OPC.UA.DA' := '1'}
  SeamNumber : WORD;

END_VAR
VAR CONSTANT
  ARRAY_LENGTH : UDINT := 40;
  JobIndex       : BYTE := 9;
  WireFeedSpeedIndex  : BYTE := 11;
  ArcLengthCorIndex   : BYTE := 13;
  PulseDynCorIndex  : BYTE := 15;
  WireRetractCorIndex : BYTE := 17;
  WeldingSpeedIndex   : BYTE := 19;
  ProcessContCorIndex : BYTE := 21;
  SeamNumberIndex   : BYTE := 39;
END_VAR
```

C.3.2 The executing Code for Outputs Function Block:

```
IF qByte1_Set_Type THEN
  qArray[1] := (BOOL_TO_BYTE(Welding_start) * 16#01) +
          (BOOL_TO_BYTE(Robot_ready) * 16#02) +
          (BOOL_TO_BYTE(WorkingModeB0) * 16#04) +
          (BOOL_TO_BYTE(WorkingModeB1) * 16#08) +
          (BOOL_TO_BYTE(WorkingModeB2) * 16#10) +
          (BOOL_TO_BYTE(WorkingModeB3) * 16#20) +
          (BOOL_TO_BYTE(WorkingModeB4) * 16#40) +
          (BOOL_TO_BYTE(0) * 16#80);
END_IF

Set_Word_TO_Bytes(setWord := Job_nr, Index := JobIndex);
Set_Word_TO_Bytes(setWord := WireFeedSpeed, Index := WireFeedSpeedIndex);
Set_Word_TO_Bytes(setWord := ArcLengthCor, Index := ArcLengthCorIndex);
Set_Word_TO_Bytes(setWord := PulseDynCor, Index := PulseDynCorIndex);
Set_Word_TO_Bytes(setWord := WireRetractCor, Index := WireRetractCorIndex);
Set_Word_TO_Bytes(setWord := WeldingSpeed, Index := WeldingSpeedIndex);
Set_Word_TO_Bytes(setWord := ProcessContCor, Index := ProcessContCorIndex);
Set_Word_TO_Bytes(setWord := SeamNumber, Index := SeamNumberIndex);

qArray[2] := (BOOL_TO_BYTE(GasOn) * 16#01) +
          (BOOL_TO_BYTE(WireForward) * 16#02) +
          (BOOL_TO_BYTE(WireBackward) * 16#04) +
          (BOOL_TO_BYTE(ErrorQuit) * 16#08) +
          (BOOL_TO_BYTE(TouchSensing) * 16#10) +
          (BOOL_TO_BYTE(TorchBlowOut) * 16#20) +
          (BOOL_TO_BYTE(ProcessLineB1) * 16#40) +
          (BOOL_TO_BYTE(ProcessLineB2) * 16#80);

qArray[3] := (BOOL_TO_BYTE(WeldingSim) * 16#01) +
          (BOOL_TO_BYTE(SynchroPulseOn) * 16#02) +
          (BOOL_TO_BYTE(WireBreakOn) * 16#40) +
          (BOOL_TO_BYTE(TorchXchange) * 16#80);

qArray[4] := (BOOL_TO_BYTE(TeachMode) * 16#02) +
          (BOOL_TO_BYTE(ActiveHeatCont) * 16#08);

qArray[5] := (BOOL_TO_BYTE(TwinModeB1) * 16#01) +
          (BOOL_TO_BYTE(TwinModeB2) * 16#02) +
          (BOOL_TO_BYTE(WireBackward) * 16#04) +
          (BOOL_TO_BYTE(DocumentMode) * 16#20);

qArray[6] := (BOOL_TO_BYTE(DisableProcesCo) * 16#80);
```

C.3.3 Method for Output Function Block

The variables part:

```
METHOD Set_Word_TO_Bytes : BOOL
VAR_INPUT
  setWord : WORD;
  Index : BYTE;    // Index for the two Bytes is always High Byte index = Low Byte
      index +1
END_VAR
```

The code part:

```
qArray[Index+1] := WORD_TO_BYTE(setWord / 16#0100); // Divide by 256 to get the
    high byte
qArray[Index] := WORD_TO_BYTE(setWord AND 16#00FF); // AND with 255 to get the low
    byte
```

D Python Code

D.1 “Welding_test_client.py”

The Python test client developed, based on the examples from the opcua-asyncio library:

```python
import asyncio
import logging
import sys
import time
sys.path.insert(0, "..")
from datetime import datetime
from asyncua import Client, Node, ua
from asyncua.crypto.security_policies import SecurityPolicyBasic256Sha256

logging.basicConfig(level=logging.INFO)
_logger = logging.getLogger("asyncua")

cert_idx = 1
cert = f"certificates/peer-certificate-example-{cert_idx}.der"
private_key = f"certificates/peer-private-key-example-{cert_idx}.pem"

async def write_variable_fronius(node, value):
    if type(value) == bool:
        await node.write_attribute(ua.AttributeIds.Value, ua.DataValue(ua.Variant(value)))
    elif isinstance(value, (int) ):
        await node.write_value(ua.DataValue(ua.Variant(value, ua.VariantType.UInt16)))
    else:
        print('Unsupported type for fronius')

async def start(outputs, inputs):
    weldStart = outputs[0]
    robReady = outputs[1]
    workB0 = outputs[2]
    workB1 = outputs[3]
    workB2 = outputs[4]
    job = outputs[25]

    weldReady = inputs[1]
    processActive = inputs[3]

    await robReady.write_value(ua.DataValue(ua.Variant(True)))
    await workB0.write_value(ua.DataValue(ua.Variant(False)))
    await workB1.write_value(ua.DataValue(ua.Variant(True)))
    await workB2.write_value(ua.DataValue(ua.Variant(False)))
    await job.write_value(ua.DataValue(ua.Variant(12, ua.VariantType.UInt16)))

    print(job)
    print(await job.get_access_level())
    print(await job.get_value())

    print('weldReady', await weldReady.get_value())
    print('processActive', await processActive.get_value())

    await weldStart.write_value(ua.DataValue(ua.Variant(True)))

    print('weldReady', await weldReady.get_value())
    print('processActive', await processActive.get_value())
    print('weldStart', await weldStart.get_value())


async def stop(outputs, inputs):
    weldStart = outputs[0]
    robReady = outputs[1]

    weldReady = inputs[1]
    processActive = inputs[3]

    print('weldReady', await weldReady.get_value())
    print('processActive', await processActive.get_value())

    await weldStart.write_value(ua.DataValue(ua.Variant(False)))
    await robReady.write_value(ua.DataValue(ua.Variant(False)))

    print('weldReady', await weldReady.get_value())
    print('processActive', await processActive.get_value())
    print('weldStart', await weldStart.get_value())





async def task(loop):
    #url = "opc.tcp://127.0.0.1:4840/freeopcua/server/"
    url = "opc.tcp://169.254.119.112:4840/freeopcua/server/"
    #url = "opc.tcp://localhost:4840/freeopcua/server/"
    client = Client(url=url)

    credentials_file = 'credentialsPLC.txt'
    #credentials_file = 'credentials.txt'
    # Read username and password from the file
    with open(credentials_file, 'r') as f:
        username = f.readline().strip()
        password = f.readline().strip()
    client.set_user(username=username) #########
    client.set_password(password) #########
    #client.s
    await client.set_security(
        SecurityPolicyBasic256Sha256,
        certificate=cert,
        private_key=private_key#,
        #server_certificate="certificate-example.der"
    )
    async with client:
        objects = client.nodes.objects
        ##############################################################
        #await browse_children_recursively(objects)
        print('\n-------\n',objects,'\n--------------\n')

        children = await objects.get_children()
        print('\n--------------------------')
        for child in children:
            #print(await child.get_path())
            display_name = await child.read_display_name()
            if 'PLC1' in display_name.Text:
                plc1 = child
            print(display_name)
        print('--------------------------\n')

        print(plc1)
        plc1_children = await plc1.get_children()
        #print(plc1_children)

        for child in plc1_children:
            display_name = await child.read_display_name()
            if 'MAIN' in display_name.Text:
                main = child
            print(display_name)
        print('--------------------------\n')

        print(main)

        main_children = await main.get_children()
        #print(main_children)

        for child in main_children:
            display_name = await child.read_display_name()
            if 'froniusOutputs' in display_name.Text:
                fOut = child
            elif 'froniusInputs' in display_name.Text:
                fIn = child
            print(display_name)
        print('--------------------------\n')

        print(fOut)

        fOut_children = await fOut.get_children()
        fIn_children = await fIn.get_children()

        #print(fOut_children)
        outputs = []
        inputs  = []

        for child in fOut_children:
            display_name = await child.read_display_name()
            if 'qArray' in display_name.Text:
                pass
            else:
                outputs.append(child)

            dVal = await child.read_data_value()
            var_type = dVal.Value.VariantType
            print('Output Variables: ',display_name.Text, ';    ', var_type)
        print('--------------------------\n')

        #print(outputs)

        #print(fIn_children)

        for child in fIn_children:
            inputs.append(child)
            display_name = await child.read_display_name()
            if 'iArray' in display_name.Text:
                pass
            else:
                inputs.append(child)

            dVal = await child.read_data_value()
```

```python
173             var_type = dVal.Value.VariantType
174             print('Input Variables: ',display_name.Text, ';    ', var_type)
175         print('--------------------------\n')
176 
177         my_var = 0
178         while True:
179             my_var = int(input('1: Start, 2: Stop, 3: Finnished: '))
180             match my_var:
181                 case 0:
182                     pass
183                 case 1:
184                     await start(outputs, inputs)
185                     my_var = 0
186                 case 2:
187                     await stop(outputs, inputs)
188                     my_var = 0
189                 case 3:
190                     break
191                 case _:
192                     await stop(outputs, inputs)
193                     print('error')
194                     break
195 
196         # await start(outputs, inputs)
197 
198         # time.sleep(5)
199 
200         # await stop(outputs, inputs)
201 
202 
203 
204 
205 def main():
206     loop = asyncio.get_event_loop()
207     loop.set_debug(True)
208     loop.run_until_complete(task(loop))
209     loop.close()
210 
211 
212 if __name__ == "__main__":
213     main()
```

The credentials file that is read in lines 83-90:

```
Administrator
1
```

The two other Python clients created for testing are modified versions of the client shown above. Therefore, only the altered parts will be detailed below, while all complete files are attached as described in Appendix E.

D.2 "Welding_test_client_loop.py":

```python
...
async def start(outputs, inputs):
    weldStart = outputs[0]
    robReady = outputs[1]
    workB0 = outputs[2]
    workB1 = outputs[3]
    workB2 = outputs[4]
    job = outputs[25]

    weldReady = inputs[1]
    processActive = inputs[3]

    await robReady.write_value(ua.DataValue(ua.Variant(True)))
    await workB0.write_value(ua.DataValue(ua.Variant(False)))
    await workB1.write_value(ua.DataValue(ua.Variant(True)))
    await workB2.write_value(ua.DataValue(ua.Variant(False)))
    await job.write_value(ua.DataValue(ua.Variant(12, ua.VariantType.UInt16)))

    print(job)
    print(await job.get_access_level())
    print(await job.get_value())

    print('weldReady', await weldReady.get_value())
    print('processActive', await processActive.get_value())

    #await weldStart.write_value(ua.DataValue(ua.Variant(True)))

    print('weldReady', await weldReady.get_value())
    print('processActive', await processActive.get_value())
    print('weldStart', await weldStart.get_value())
...
        my_var = 0
        counter = 0
        while True:
            counter +=1
            if counter % 2 == 0:
                my_var = 2
            else:
                my_var = 1
            time.sleep(1000/1000) # waiting 500 [ms]
            match my_var:
                case 0:
                    pass
                case 1:
                    await start(outputs, inputs)
                    my_var = 0
                case 2:
                    await stop(outputs, inputs)
                    my_var = 0
                case 3:
                    break
                case _:
                    await stop(outputs, inputs)
                    print('error')
                    break
```

The unique aspects of this loop code reside in the start function, where it only sets the robot's ready signal, not the weld start signal. Additionally, towards the end of the match case, rather than awaiting user input, it alternates between start and stop every second.

D.3 "Welding_test_client_counter.py":

```python
import asyncio
import logging
import sys
import time
import csv
sys.path.insert(0, "..")
from datetime import datetime
from asyncua import Client, Node, ua
from asyncua.crypto.security_policies import SecurityPolicyBasic256Sha256

...

        time_elapsed_list = []
        robReady = outputs[1]
        weldReady = inputs[1]
        for i in range(100):
            await robReady.write_value(ua.DataValue(ua.Variant(False)))
            await asyncio.sleep(1)
            #time.sleep(1)

            start_time = time.time()
            await robReady.write_value(ua.DataValue(ua.Variant(True, ua.VariantType.Boolean)))

            while not await weldReady.get_value():
                await asyncio.sleep(0.001)  # Check every millisecond

            end_time = time.time()
            time_elapsed = (end_time - start_time) * 1000
            time_elapsed_list.append(time_elapsed)
            print(f"Time elapsed: {time_elapsed} milliseconds")
            await asyncio.sleep(1)

        #print(time_elapsed_list)

        # Write the times to a CSV file
        with open('times.csv', 'w', newline='') as f:
            writer = csv.writer(f)
            writer.writerow(["Elapsed Time (ms)"])
            for t in time_elapsed_list:
                writer.writerow([t])

...
```

The variations in the counter code primarily lie in the import of the CSV library. Although the start function from the loop code is used, it remains inactive, as the match case component has been replaced with a for loop. Additionally, a section for writing to a file has been incorporated.

E Attachments

E.1 Attached files:

- The TwinCAT project with the PLC code
- The Python test and testing clients
- The author's Project Thesis
- The risk assessment done with a fellow student
- A video of the Linux test

E.2 Hardware:

The control cabinet with the Beckhoff CX8190 Embedded PC that was used as a PLC with the EL6752 DeviceNet master terminal and the DeviceNet cable that was created.