Satakunnan ammattikorkeakoulu
Satakunta University of Applied Sciences

OLLI UUSIKARTANO

Remote monitoring system for CNC machining equipment

ELECTRICAL AND AUTOMATION ENGINEERING
2025

ABSTRACT

Uusikartano, Olli: Remote monitoring system for CNC machining equipment
Bachelor's thesis
Electrical and Automation Engineering
February 2025
Number of pages: 67

This thesis presents the development of a remote monitoring system for CNC machining equipment at Morehead State University. The system enables real-time data collection, processing, and visualization to improve operational efficiency. Newer Haas CNC machines were monitored using Ethernet/IP, while older machines required programmable relays and sensors connected to a PLC. Data was extracted and processed using Python scripts and transmitted to an OPC UA server, which served as a central hub. The collected data was then displayed in a user interface, providing real-time machine status updates.

The system successfully improved monitoring efficiency, reduced the need for manual supervision, and laid the groundwork for future expansion, such as AI-driven predictive maintenance. This project highlights how Industry 4.0 technologies can optimize manufacturing processes and enhance automation.

Keywords: CNC machines, remote monitoring, OPC UA, Industry 4.0, Python, data acquisition

PREFACE

This thesis is the result of a project undertaken at Morehead State University, where I had the opportunity to work on developing a remote monitoring system for CNC machining equipment. I would like to express my gratitude to Dr. Jorge Ortega Moody, who assigned me the project and provided valuable assistance in resolving technical challenges throughout the development process.

I would also like to extend my thanks to my fellow student, Gunnar Gross, who collaborated with me on the project, contributing to its success. Without their guidance and collaboration, this project would not have been as successful.

CONTENTS

1 INTRODUCTION

Computer numerical control (CNC) machines are vital to modern manufacturing due to their precision and ability to produce repeatable results. They are extensively used across industries by large, small, and medium-sized enterprises. Particularly in small and medium-sized enterprises, monitoring the performance of manufacturing equipment has traditionally been a manual task, requiring operators to be physically present. This approach can lead to inefficiencies, increased operational costs, and result in missed opportunities for optimization. (Doyle & Cosgrove, 2019.)

This project was commissioned by Morehead State University, which sought to implement a monitoring system that integrated technologies such as programmable logic controllers (PLC), Open Platform Communications (OPC), and real-time data visualization. The goal was to create a system that not only provided real-time data from the machines but also presented it in an easy-to-use interface that could be accessed via desktop computers or mobile devices. (Martins et al., 2023.)

Additionally, once the CNC machines are connected to the OPC server, it can facilitate integration with robots and other devices. This will no doubt be useful in future projects at the university.

Recent advancements in artificial intelligence (AI) offer powerful decision support systems (DSS) for optimizing production processes (Martins et al., 2023). By using the gathered data to train AI models, the system could enable predictive and preventative maintenance, thereby reducing downtime caused by machine faults and significantly improving production efficiency. Although the project did not implement AI-based predictive maintenance, it laid a foundation for future work.

This thesis outlines the planning and implementation of the system, from the selection of hardware and software components to the creation of Python scripts for data collection and the development of a user interface using Ignition software. It concludes with an analysis of the results and the potential impact of such systems on operational efficiency.

1.1 Use of AI in this thesis

In this thesis, ChatGPT was used for formatting text, checking grammar, assisting with programming, commenting code and finding sources. Floatz AI was also used to find sources. Some of the text has been rewritten by ChatGPT to make the text clearer while keeping my original meaning.

2 COMMISSIONER OF THE WORK

This thesis was commissioned by Morehead State University (MSU), located in Morehead, Kentucky, USA. Established in 1887, MSU is a public university known for its emphasis on research and community engagement. It serves over 10,000 students annually and offers a wide range of academic programs across disciplines, including engineering, business, the arts, and sciences. MSU takes pride in its commitment to innovation and preparing students for successful careers in an ever-evolving global landscape. (Morehead State University, n.d.)

Within the university's College of Science and Engineering, the Department of Engineering Sciences provides advanced facilities and resources, enabling students and researchers to engage in hands-on learning and impactful projects. MSU’s focus on industrial and technological advancements aligns with the objectives of this thesis project. (Morehead State University, n.d.)

The goal of this project was to meet the university's need to develop a remote monitoring system for CNC machining equipment, improving its educational and research capabilities. By implementing this system, the university aims to integrate more Industry 4.0 technologies into its curriculum, offering students exposure to real-world industrial practices.

3 IOT AND INDUSTRY 4.0

This chapter briefly discusses IoT and Industry 4.0. These technologies are crucial in modern manufacturing and play an important role in the development of the monitoring system described in this thesis.

3.1 IoT

The Internet of Things (IoT) represents a network of connected devices that communicate and share data via the Internet. In industrial environments, IoT enables enhanced automation, efficiency, and decision-making by leveraging sensors, actuators, and communication networks. IoT systems collect, process, and analyze data, providing actionable insights for optimizing operations. They rely on communication technologies such as Wi-Fi, Bluetooth, and 5G networks to facilitate data exchange between devices. (Margaza & Yurchenko, 2024.)

Despite its benefits, the adoption of IoT technologies in industrial settings introduces challenges, particularly in the areas of data security and privacy. IoT devices can be vulnerable to cyber threats, such as unauthorized access by hackers. These risks highlight the need for strong cybersecurity measures and encryption standards to protect sensitive data and ensure the secure operation of IoT-enabled systems. (Margaza & Yurchenko, 2024.)

As IoT enables data exchange across devices, it serves as a foundational pillar of the broader Industry 4.0 revolution, which aims to fully integrate cyber-physical systems into manufacturing operations. This transition to smart factories is the focus of the next section.

3.2 Industry 4.0

Industry 4.0, often referred to as the Fourth Industrial Revolution, marks a transformative change in manufacturing and industrial practices. The foundational technologies of Industry 4.0 include ICT (Information and Communication Technology) and IoT which facilitate the integration of physical and digital systems. These technologies enable real-time data collection. Unlike its predecessors, Industry 4.0 focuses on creating interconnected, intelligent systems that enable seamless communication between devices, systems, and humans. (Ajayi, Bagula, & Maluleke, 2023, pp. 1–2.)

Despite its transformative potential, Industry 4.0 like IoT introduces challenges such as the need for cybersecurity measures to address vulnerabilities in interconnected systems. Data security, privacy concerns, and interoperability between diverse technologies remain critical issues that require comprehensive strategies. (Ajayi, Bagula, & Maluleke, 2023, p. 17.)

4 MACHINE DATA COLLECTION

Machine data collection refers to the process of gathering information from industrial machines, such as CNC equipment. Data collection systems typically use sensors, PLCs, and communication protocols to transmit real-time data to servers or control systems. This data includes various machine parameters, such as temperature, speed, vibration, and production counts.

4.1 PLC and sensors

A programmable logic controller (PLC) is a microcomputer-based device used to control machines and processes through programmable instructions. It can handle logic, sequencing, timing, and arithmetic tasks via digital or analog input/output modules. PLCs are widely used in industries like chemical processing, manufacturing, and material handling. They were introduced in the 1970s to replace hardwired controllers. PLCs offer significant advantages, such as easier programming, reusability, reduced space requirements, greater reliability, and compatibility with computer systems. (Groover, 2015, pp. 257–258.)

4.1.1 Inductive proximity switches

Inductive proximity switches can be used to detect metallic objects. They consist of a coil wound around a ferrous metallic core. When a metallic object is placed near the sensor head there is a change in the inductance of the coil and core. The change in inductance can be used to close or open an electric circuit. The sensing range refers to the maximum distance at which the sensor can detect a metal object approaching it along its axis (Figure 1). The sensing range of these sensors is typically 2mm to 15mm. (Bolton, 2015 p. 30.)

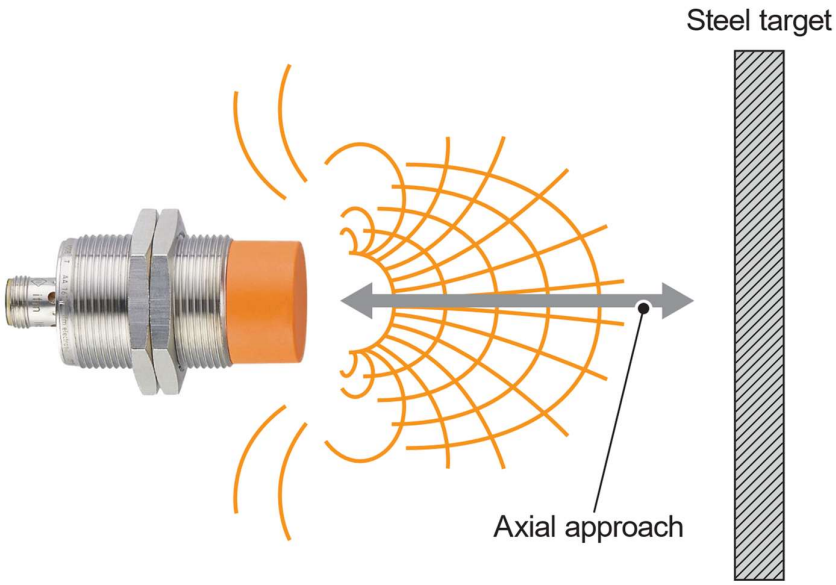

Figure 1. Inductive proximity switch (ifm efector inc, n.d.a).

4.1.2 Capacitive proximity switches

Capacitive proximity switches can be used to detect both non-metallic and metallic objects by the changing capacitance between the sensor head and the object being detected (Figure 2). The closer the object gets to the sensor head the greater the capacitance. These sensors generally have a sensing range between 3mm to 60mm. (Bolton, 2015 pp. 29–30.)

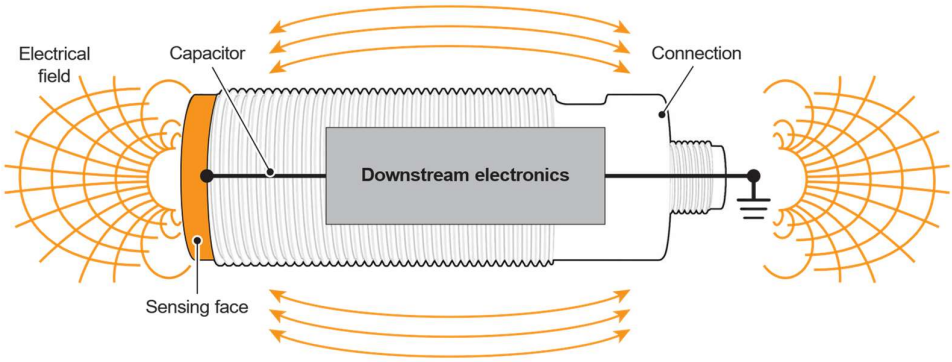

Figure 2. Diagram explaining how a capacitive proximity switch functions. (ifm efector inc, n.d.b).

4.2 Industrial communication networks

Industrial automation has created a need for communication and control across entire plants. This requires interconnecting programmable controllers, computers, robots, and CNC machines. A local area network (LAN) enables communication within a single building or site by linking computers and their peripherals. PLC manufacturers use various network systems tailored to their devices, such as Mitsubishi's MelsecNET, Allen-Bradley's Data Highway Plus, and Siemens' PROFIBUS. (Bolton, 2015 p. 99.)

4.2.1 OPC UA

Unified Architecture (UA) is the next generation of Open Platform Communications (OPC). OPC UA is a secure, open, and reliable mechanism used to transfer information between a server and client. It is a flexible and adaptable mechanism moving data between many kinds of devices and systems from supervisory control and data acquisition (SCADA) systems to individual sensors or CNC machines. (Rinaldi, 2016, p.4.)

The OPC UA address space is structured around the concept of an object, which serves as a fundamental entity composed of variables and methods. This object-based framework provides a standardized way for servers to transfer information to clients, enabling communication. OPC UA is capable of transmitting data over the internet, enabling connectivity across networks. Additionally, it offers a comprehensive suite of services, including eventing, alarming, reading, writing, discovery, and more. (Rinaldi, 2016, pp.4–5.)

4.2.2 Ethernet TCP/IP

Ethernet is the most widely used method for creating a local area network (LAN) that connects computers and servers. It is the global standard for linking devices using a unified set of electrical and software protocols to enable data exchange. The Institute of Electrical and Electronics Engineers (IEEE) has formalized Ethernet as the international standard 802.3. In industrial settings, Ethernet is adapted into a specialized version called Industrial Ethernet, which extends the 802.3 standard to meet the unique demands of industrial environments. (Rinaldi, 2018, p.17.)

While Ethernet 802.3 defines the electrical and link-layer standards for connecting devices, it is not a complete solution on its own. To enable effective communication over the network, protocols for data transfer are necessary. This need led to the development of TCP/IP, a suite of communication protocols designed in the 1970s by Vint Cerf and Bob Kahn, often referred to as the fathers of the Internet. TCP/IP emerged from a project to create a radio-based

packet network and was built on the idea of open-architecture networking, enabling communication between devices regardless of hardware or software differences. The initial version of TCP, originally called the Transmission Control Program, evolved over time. It was revised and formalized in RFC 675 in December 1974 and later expanded into a suite of protocols that underpin modern networking. (Rinaldi, 2018, pp.17–18.)

Today, TCP/IP includes essential protocols such as the Internet Protocol (IP), the Simple Mail Transfer Protocol (SMTP) used for email and the HyperText Transfer Protocol (HTTP) used for web browsing. This suite enables seamless network connectivity and underpins the functionality of the modern Internet. In both historical and industrial contexts, the combination of Ethernet and TCP/IP has been transformative, with each technology reinforcing the other's success and making possible the interconnected world we rely on today. (Rinaldi, 2018, p.18.)

TCP/IP can be visualized as a series of interconnected pipes facilitating communication between senders and receivers. Just as physical pipes can transport various substances (e.g., water or oil), protocol pipes in networking carry data bits without concern for their content. It is crucial for both the sender and receiver to agree on how to interpret the transmitted data. If a receiver expects water but gets motor oil instead, confusion is inevitable. In networking, senders and receivers establish this understanding through application layer protocols such as EtherNet/IP, PROFINET IO, and Modbus TCP™. TCP/IP serves as the infrastructure, the "pipe," that encapsulates packets from these protocols and ensures delivery from sender to receiver. (Rinaldi, 2018, p.19.)

4.2.3 MTConnect

MTConnect is an open, royalty-free standard designed for the manufacturing industry, enabling communication between manufacturing equipment and software applications through established internet protocols. MTConnect requires

both systems to adhere to the same standard to facilitate communication. (Edstrom, 2014. Section 4 Chapter 23.)

The MTConnect agent acts as a web server that bridges adapters, which connect to machine tools, with applications such as monitoring systems. It plays a vital role in the MTConnect ecosystem by standardizing and delivering machine data in a uniform format to client applications. Applications interact with the agent by accessing data from buffers at the desired rate, ensuring they can retrieve real-time or historical information as needed. This architecture ensures that machine data is always available. (Edstrom, 2014. Section 4 Chapter 23.)

An MTConnect application, often referred to as a client, communicates directly with the MTConnect agent to request and process data. These applications rely on the agent's sequence numbers to continuously retrieve the correct XML data. A typical example of such an application is shop floor monitoring software, which collects real-time data from manufacturing equipment, analyses it, and visualizes the information through charts or graphs to provide insights into machine performance. (Edstrom, 2014. Section 4 Chapter 23.)

4.3 Python

Python is a programming language that is widely appreciated for its simplicity and readability. Its simplicity and ease of use make it a favored option for both novice and seasoned developers. Python supports multiple programming paradigms including, object-oriented, procedural, and functional programming. This enables developers to select the most suitable approach based on their specific requirements. (Python Software Foundation, n.d.)

A major advantage of Python is its comprehensive standard library and the wide range of third-party packages accessible through the Python Package Index (PyPI). This broad collection of libraries allows developers to perform tasks such as web development, data analysis, scientific computing, machine learning, automation, and more, with minimal code. (Analytics Vidhya, n.d.)

Python's flexibility makes it suitable for a wide range of applications, from simple scripts to large-scale web applications (Analytics Vidhya, n.d.). Its ability to integrate with other languages, like Java further improves its functionality. Python is also known for its strong support of community-driven development, with an active user base contributing to the language's growth and development. (Invicktus, 2023.)

5 UTILIZING MACHINE DATA

While collecting machine data is essential, its real value lies in how it is processed and applied to improve manufacturing processes. It can be used in industrial environments to improve efficiency, productivity, and decision-making processes. By collecting and analyzing data from machines such as CNC equipment, organizations can gain valuable insights into operational performance, optimize machine usage, and predict future maintenance needs. The utilization of machine data spans multiple areas, from real-time monitoring to long-term data analysis, helping industries to improve both day-to-day operations and strategic planning.

5.1 Predictive maintenance

By continuously tracking and analyzing data such as machine vibrations, motor temperatures, and usage patterns, predictive maintenance algorithms can detect early signs of wear or malfunction. These algorithms can forecast when a machine is likely to fail or require maintenance, allowing maintenance to be scheduled at the optimal time before a breakdown occurs without interrupting operations unnecessarily. This helps reduce unplanned downtime, extends the life of machinery, and minimizes maintenance costs. Machine data is essential for implementing predictive maintenance strategies. (Dhar, 2021, Chaper 1.)

5.2 Remote troubleshooting

Many companies like MachMotion offer remote troubleshooting for CNC machines (MachMotion, n.d.). In case of an unexpected machine fault the collected data can be sent directly to maintenance support outside the factory. This allows the support staff to access all relevant data. They may be able to diagnose the fault remotely, allowing them to send appropriate spare parts and instructions without visiting the factory.

5.3 Real-time fault diagnosis

Collected data can be used in real-time fault diagnosis (RTFD) systems to identify and isolate faults as they occur, preventing escalation and reducing downtime. By integrating real-time data analytics and predictive models, RTFD systems enhance decision-making and improve the resilience of manufacturing operations. This proactive approach enables higher productivity and cost efficiency. (Yan et al., 2023.)

6 CNC MACHINES

CNC is a technology used in modern manufacturing to automate the control of machine tools. CNC systems rely on computer programming to direct the movements of machines, such as mills and lathes, allowing for precise, repeatable manufacturing processes. The use of CNC technology revolutionized production by improving efficiency, accuracy, and flexibility in producing complex parts for industries like automotive, aerospace, and electronics (FastCut, 2024). Machines equipped with CNC systems can perform operations such as cutting, shaping, and drilling automatically, based on the instructions provided by preprogrammed code. (Miller & Miller, 2004, p. 429.)

Small and medium-sized enterprises (SMEs) use CNC machines to manufacture components for various industries, including custom components for local manufacturers and prototypes for product development. These machines enable SMEs to deliver high-precision results without needing the extensive workforce required for manual machining. Moreover, the flexibility of CNC technology allows SMEs to handle small batch production efficiently. (CNC Machines, 2024.)

Older CNC machines often lack interfaces for real-time data collection, making it challenging or even impossible (Schulte et al., 2023). In contrast, newer CNC machines are often equipped with Ethernet ports and support protocols such as EtherNet/IP, enabling data collection and integration with monitoring systems. This capability can be used to enable improved machine optimization and predictive maintenance, improving overall productivity. (Van Dinter et al., 2022.)

6.1 CNC milling machines

A vertical CNC mill is a machine used to work metal, wood, and other solid materials. It operates by rotating a cutting tool, which removes material from the workpiece. CNC mills are equipped with programmable computer controllers, allowing for operations, like drilling, boring, and milling. The Haas Mini Mill-EDU used at MSU (Figure 3) can also be classified as a machining center due to its automatic tool changer. Mills typically have three or more axes, enabling movement along the X, Y, and Z directions, and often additional rotational axes. (Dixon & Walker, 2019, pp.393–395.)

Figure 3. Example of a CNC milling machine Haas Mini Mill-EDU (Haas Automation, n.d.a).

6.2 CNC Lathes

A CNC lathe is used to perform turning operations where the workpiece is rotated, and a stationary cutting tool shapes it (Figure 4). Lathes are commonly used for cylindrical objects. In a CNC lathe the computer controls allow for precise movement of the cutting tool in relation to the rotating workpiece, leading to high-quality finishes. (Dixon & Walker, 2019, p.395.)

Figure 4. Example of a CNC lathe Haas TL-1 (Haas Automation, n.d.b).

7 DESIGNING THE MONITORING SYSTEM

This chapter explains the planning process for the monitoring system. including researching methods for data collection, transferring data to the OPC server, and creating a user-friendly interface for displaying the information.

7.1 Methods of collecting data from HAAS CNC Machines

During the planning phase, several methods of collecting data from HAAS CNC machines were researched. For the newer HAAS models, one option was to utilize their RJ45 Ethernet IP ports, which allow for direct communication with the machines through standard networking protocols. This would enable

the extraction of key machine data, such as spindle revolutions per minute (RPM), spindle position, and tool status, via Q commands and MT Connect requests. The available documentation from Haas provided instructions on how this could be achieved.

To automate data collection, Python scripts could be developed to send queries to the machines and interpret responses. By using these scripts, data could be extracted from the machines in real-time.

For the older Haas models, which lacked Ethernet capabilities, other options had to be explored. Connecting the machine's programmable relays, which could output status signals, to a PLC was devised as a solution. This would allow the collection of basic machine status information like whether the machine was running or idle, though with fewer data points compared to the newer models. Sensors could also be installed on these machines and connected to the PLC to supplement data collection where necessary.

7.2 Transferring the data for CNC machines to the OPC server

The most straightforward option for newer machines was to transmit the data over a local area network. Connecting the Ethernet ports of the machines to the PC running the KEPServerEX OPC server through an Ethernet switch.

Python scripts were used to facilitate data transfer. Once collected by the Python scripts, the data could be processed and sent to specific nodes on the OPC server by utilizing the Python OPC UA library. Each data point such as spindle RPM, spindle position, and tool status, could be directed to its designated location on the server.

For machines without built-in Ethernet ports, the PLC could act as a bridge between the machine and the OPC server by receiving status signals from the machines and then transmitting this data to the server through Ethernet. This

method, while effective, required additional hardware and configuration compared to the direct Ethernet connection option available with newer machines.

7.3 Planning the user interface

The primary requirement was to create a real-time interface that was easy to use and accessible both on-site and remotely, with compatibility across desktop computers and mobile devices.

One of the options researched was the development of a custom web-based dashboard, which would allow for a tailored interface and custom functionality. However, this approach would require significant development time to ensure the interface was responsive and compatible with different screen sizes, while also providing real-time data from the CNC machines.

An alternative solution was to use existing industrial software platforms designed for data visualization and machine monitoring. Inductive Automation's Ignition was chosen as the software for creating the user interface due to its robust built-in functionality for real-time data updates, cross-device compatibility, and easy integration with OPC servers. While this method might limit customization, it offered a faster and more efficient route to deployment.

7.4 Final system design

The designed system architecture enables data collection, processing, and visualization through an integrated network of hardware and software components (Figure 5). The system retrieves data from both modern and older CNC machines and presents it in real-time on the Ignition UI.

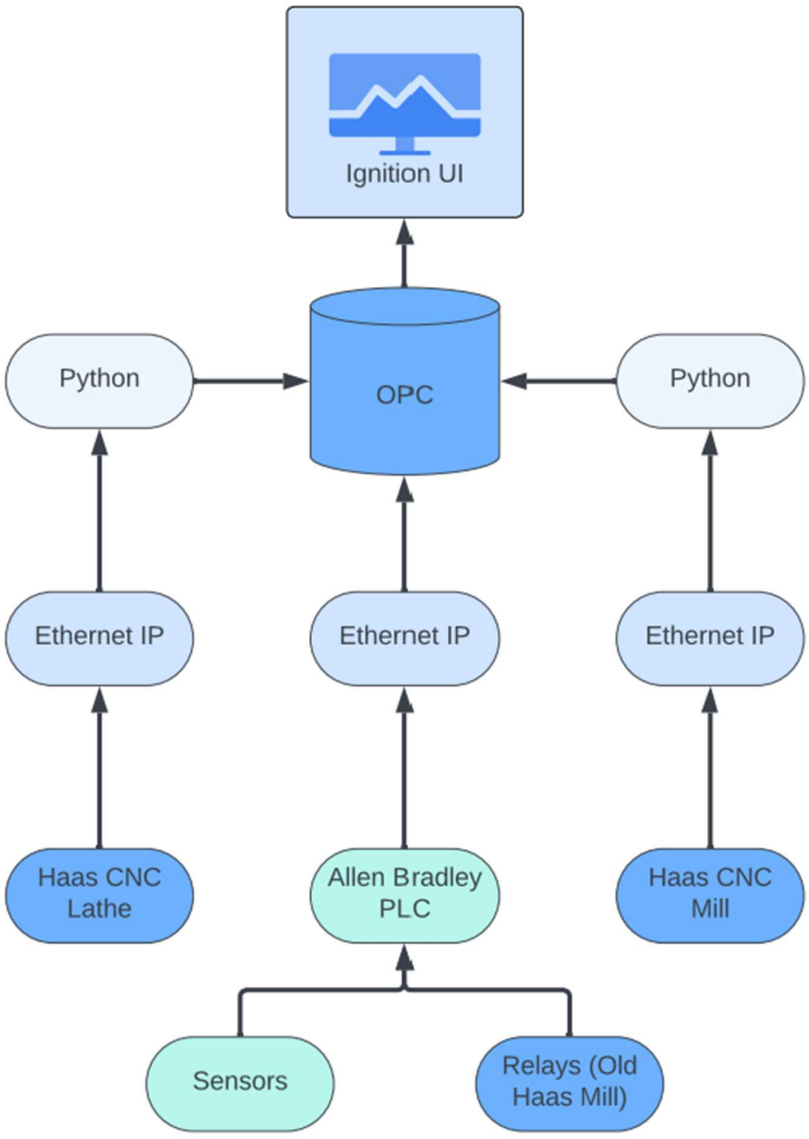

Figure 5. Diagram of the data collection system.

Newer Haas CNC machines communicate via Ethernet IP, with Python scripts handling data extraction, processing, and transmission to the OPC server. Older Haas CNC machines, which lack Ethernet capabilities, rely on their programmable relays and additional sensors connected to an Allen Bradley PLCs input module. The PLC transmits the data directly to the OPC server via Ethernet IP. Finally, the data is read from the OPC server displayed in the Ignition UI.

8 IMPLEMENTATION OF THE MONITORING SYSTEM

This chapter explains the implementation process of the monitoring system. It covers the physical setup, including the installation of sensors and cabling, the configuration of the OPC UA server, the creation of Python scripts for data handling, and finally the development of the user interface.

8.1 Connecting the sensors and relays to the PLC

The first step in building the data collection system was connecting the programmable relays from the HAAS mills without Ethernet ports to the PLC. The programmable relays, with dry contacts, were wired to the PLCs inputs. Additionally, capacitive sensors were installed to monitor the scrap levels in the collection boxes under the machines, as they can detect both metallic and non-metallic objects. This configuration enabled the system to prevent overflows. An inductive sensor was installed on the door of an older Haas mill to monitor its status. An inductive sensor was chosen because the door was primarily made of iron. Both inductive and capacitive sensors were connected to the PLC inputs to enable monitoring (Figure 6).

Figure 6. The PLC program used to connect sensors to PLC tags.

8.2 Building the LAN network

A local area network (LAN) was set up to connect the CNC machines, the PLC, and the server PC (Figure 7). This was achieved by installing an Ethernet switch in an electrical box attached to one of the CNC machines. CAT 6 cables were used to connect the machines and the server PC to the switch. The Ethernet switch was chosen to minimize the number of cables required and simplify the network setup. Since only a relatively small amount of data was being transmitted between the machines and the server, the network's data transmission rate was not considered a major concern. This setup allowed for efficient and reliable device connectivity with minimal cabling.

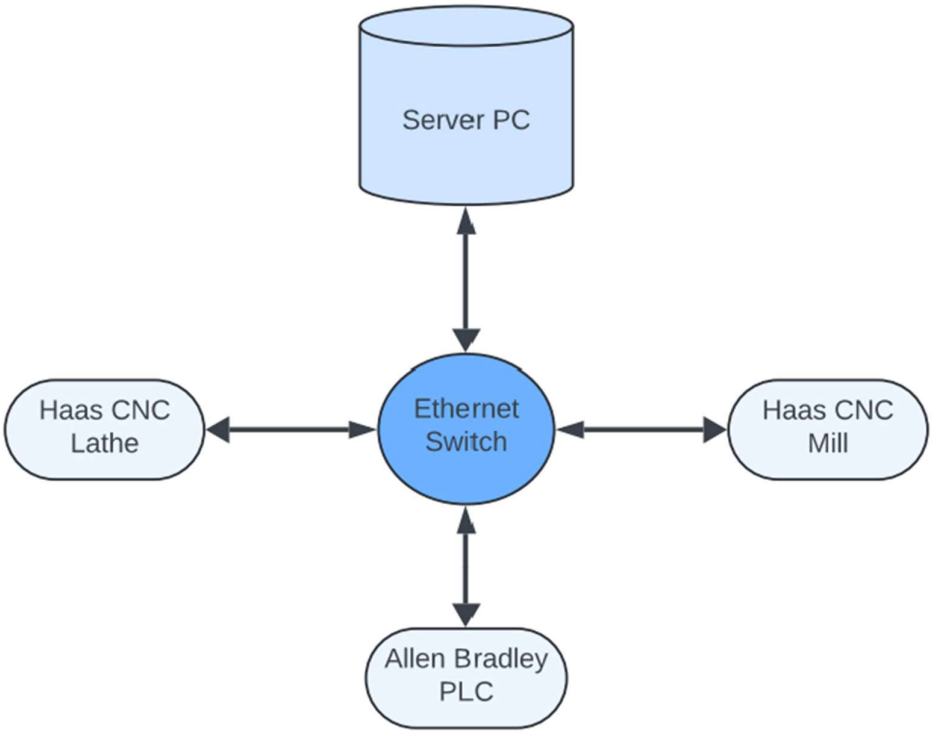

Figure 7. Diagram of the LAN network.

8.3 Setting up the OPC server

The setup of the OPC server began with the installation of the KEPWARE OPC UA server software on the server PC. Once the software had been installed, it was launched, and the configuration process was initiated. Communication channels were created to establish pathways between the server, the CNC machines and PLC. The appropriate device types and settings were selected to ensure compatibility.

After the channels were configured, tags were created within the OPC server. Each tag represented a specific data point, such as spindle RPM or coolant levels, and was assigned a unique name. The tag properties, including the data type and address, were defined.

Access permissions were set for each tag, specifying whether the tag could be read, written to, or both. Additionally, an update rate was chosen to control how frequently the data would be refreshed by the server. The correct data type for each tag was selected so that the incoming data would be interpreted accurately.

Once the tags had been created, they were organized into a hierarchy based on their function and the devices they represented. This organization was intended to simplify future management and ensure that tags could be easily located and updated.

Finally, the OPC server configuration was saved, and the server was set to run continuously so that data collection could occur without interruption, and the system would be ready for integration with other components of the monitoring solution.

8.4 Creating the Python script to collect data from the machines and write to the OPC server

To facilitate communication and data transfer between the CNC machines and the OPC server, Python scripts were developed for the lathe and the newer Haas Mill. The script for the mill can be found in Appendix 1 and the script for the lathe in Appendix 2. These scripts collect data from the CNC machines, process it to ensure accuracy and compatibility, and then transmit the processed information to the OPC server.

8.4.1 Architecture of the Python scripts

A switch statement is a control structure in many programming languages that allows the execution of different code blocks based on the value of an expression or variable. In this program the variable is “lang”.

Within the switch statement, most cases are used for a single machine parameter. For example, case one sends a message to the CNC machine requesting the spindle RPM (Figure 8). After the request is sent, the program waits for a reply from the CNC machine. The received reply is then processed to determine its validity.

```
    case 1:
#Spindle_RPM
        Spindle_RPM = sendMessage("?Q600 3027")
        print("Spindle_RPM",Spindle_RPM)

        # If the "sendMessage" function returns "Invalid"
        # lang is set to the case number so the case is repeated
        if Spindle_RPM == "Invalid" :

            print("Reply Invalid")
            lang = 1
            return lang
        else:
        #If the "sendMessage" function returns valid data
        #The answer is writen to the OPC server
            print("Reply Valid")

            floatMixedNumbers = ProcessReplyToFMN(Spindle_RPM)

            WriteToOPC("ns=2;s=Channel1.Device1.SpindleRPM", floatMixedNumbers)
            lang += 1
            return lang
```

Figure 8. Example of a single case in the switch statement.

If the reply is invalid, the program sets the variable lang to the number of the current case (e.g., lang = 1 for the first case). This causes the same case to repeat during the next execution of the switch statement, ensuring that the task is retried until a valid reply is received (Figure 9).

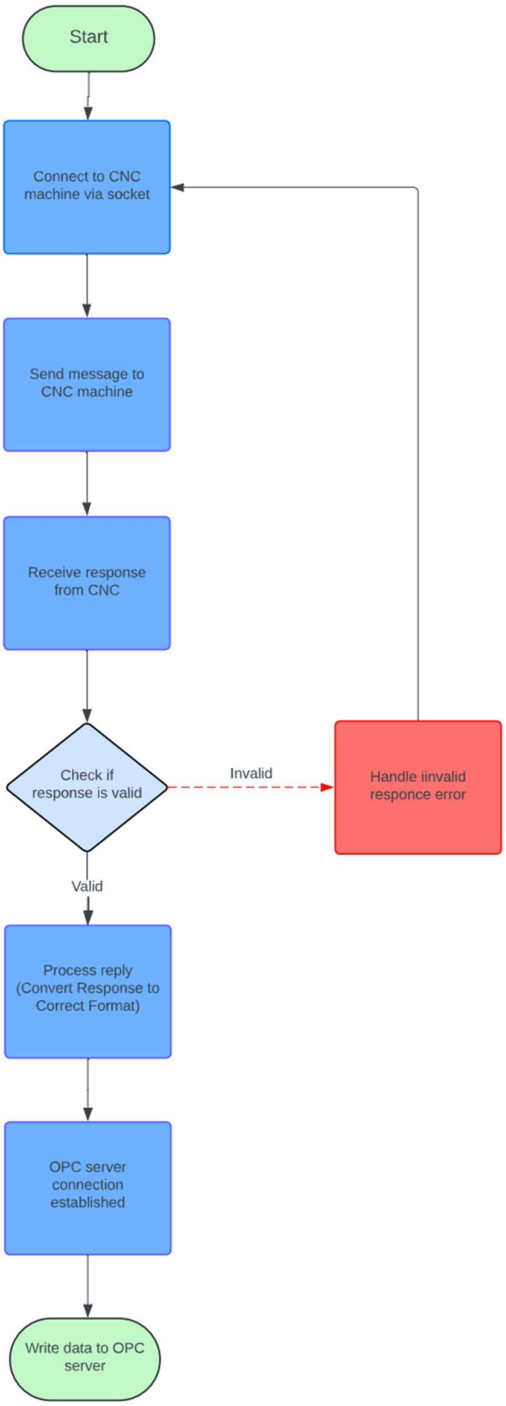

Figure 9. Explanation of a case in the switch structure.

When a valid reply is received, the program processes the data and writes it to the OPC server. Following this, the variable lang is incremented, allowing the switch statement to proceed to the next case during the next execution. This process ensures that the program completes each task sequentially and handles errors by repeating cases as needed.

8.4.2 Communication with CNC Machines

The communication between the CNC machines and the Python script was achieved using TCP/IP sockets (Figure 10) and MT Connect for machines with XML-based data streams. For machines like the mill and lathe, socket-based communication was initiated by defining the machine's IP address and port. For instance, the mill was connected using IP 192.168.1.56 over port 5051, and the lathe with IP 192.168.1.58 on the same port. Queries were sent to the machines using predefined commands, for example: `?Q600 3027` to retrieve the spindle RPM.

```python
def sendMessage(mesg):
    """Function to send a message to the CNC machine and receive a response."""
    try:
        with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as s:  # Create a socket object using TCP/IP
            s.connect((HOST, PORT))  # Connect to the machine at specified IP and port
            client_message = mesg + "\r"  # Append carriage return to message for protocol compliance
            client_message_as_byte_array = client_message.encode('ascii')  # Convert string message to bytes
            s.sendall(client_message_as_byte_array)  # Send the encoded message
            data = s.recv(DataLength)  # Receive a reply from the machine
            data = data.decode('ascii')  # Decode the reply from bytes to ASCII
            if len(data) < 3:  # Check if the received data is sufficient
                data = "Invalid"
            s.close()  # Close the socket connection
            return data
    except:
        s.close()  # Ensure socket is closed in case of error
        print("Error occurred while sending or receiving message")
        data = "Invalid"
        time.sleep(1)
        return data
```

Figure 10. The "`sendMessage`" function used in both Python scripts.

For machines equipped with MTConnect, an Extensible Markup Language (XML) -based communication was utilized. The MTConnect standard allowed for data requests via HTTP, which returned structured XML responses. In

these cases, the script fetched data from the machines using URLs such as `http://192.168.1.56:8082/Current`.

8.4.3 Processing and Formatting the Data

Once the raw data was received from the machines, it needed to be processed and formatted for use by the OPC server. In the case of XML responses from MTConnect-enabled machines, Python's `xml.etree.ElementTree` module was used to parse the incoming data. Elements containing information, such as overrides for spindle speed and feed rates, were extracted from the XML structure and stored in variables for further processing.

For socket-based data, the responses were parsed using regular expressions to extract numeric values. These values were then converted into appropriate data types, such as floats or integers, depending on the information received. For example, data related to spindle coordinates and spindle RPM were extracted from the CNC machine's responses, converted into floats, and prepared for transmission (Figure 11).

```python
def ProcessReplyToFMN(Reply):
    """
    This function processes a string reply from a device to extract and convert
    numeric values found within the string into a single float value.
    """

    try:
        # Use regular expression to find all numbers, capturing both floating-point numbers and integers.
        numbers_float = re.findall(r'\d+\.\d+|\d+', Reply)

        # Convert each extracted string number to a float if it contains a decimal point, otherwise to an integer.
        mixed_numbers = [float(num) if '.' in num else int(num) for num in numbers_float]

        # Convert the list of numbers into a string and remove the square brackets typical of Python list representations.
        strMixedNumbers = str(mixed_numbers)
        strMixedNumbers = strMixedNumbers.strip('[]')

        # Convert the processed string back into a float. This float is the final single numeric value derived from the Reply.
        floatMixedNumbers = float(strMixedNumbers)

        return floatMixedNumbers

    except:
        # If an error occurs during the conversion process, print an error message and pause execution for 1 second.
        print("Error occurred while converting data ")
```

Figure 11. The "`ProcessReplyToFMN`" function used in both Python scripts.

8.4.4 Writing Data to the OPC Server

After processing the data, it was transmitted to the OPC server using Pythons `opcua` library. The Python script established a connection to the OPC UA server, and each data point was written to its corresponding node in the server. The node identifiers were predefined, ensuring that each data value, such as spindle RPM or spindle position, was assigned to the correct node in the OPC hierarchy (Figure 12).

```python
def WriteToOPC(node_id, NewData):
    """
    This function connects to an OPC UA server and writes data to a specified node.
    It handles both float and string data types by selecting the appropriate VariantType.
    """

    try:
        # Create an OPC UA client instance using a globally defined server URL.
        client = Client(server_url)
        # Establish a connection to the OPC UA server.
        client.connect()

        # Retrieve the node object from the OPC server using the provided node ID.
        node = client.get_node(node_id)

        # Check the data type of NewData and create a DataValue instance with the appropriate VariantType.
        # If NewData is a float, use the Float variant type. Otherwise, use the String variant type for other data.
        if isinstance(NewData, float):
            valor = ua.DataValue(ua.Variant(NewData, ua.VariantType.Float))
        else:
            valor = ua.DataValue(ua.Variant(NewData, ua.VariantType.String))

        # Log the data value to be written for debugging purposes.
        print(valor)
        # Set the node's value on the server with the created DataValue instance.
        node.set_value(valor)

        # Disconnect from the OPC UA server once the data is written.
        client.disconnect()
    except:
        # If any error occurs during the connection or data writing process,
        # print an error message and pause the program briefly to handle the error gracefully.
        print("Error occurred while writing to OPC")
        time.sleep(1)
```

Figure 12. The "`WriteToOPC`" function used in both Python scripts.

For instance, the spindle RPM data was written to the node with the identifier `ns=2;s=Channel1.Device1.SpindleRPM`. The script handled both string and numeric data, with appropriate variant types used for each data type.

8.5 Building the Ignition user interface

The Ignition user interface (UI) for the remote monitoring system was structured to provide users with a real-time view of critical data from the CNC machines. The main menu displays all the machines connected to the system and

whether or not the machine is running. These data points were assigned to separate fields, which update in real-time as the machines operate.

Navigation in the UI was structured using tabs and buttons. Users could choose specific machines from the main view (Figure 13).

Figure 13. Main menu from the Ignition UI.

Machine views display information such as spindle coordinates, spindle RPM, coolant levels, door status and power (Figure 14). The UI also displays error messages and alerts on the machine screens. When a machine experiences an issue, an error message will appear, accompanied by an error code and the timestamp of when the error occurred allowing users to respond promptly to machine issues. The interface is optimized for use on both desktop computers and mobile devices.

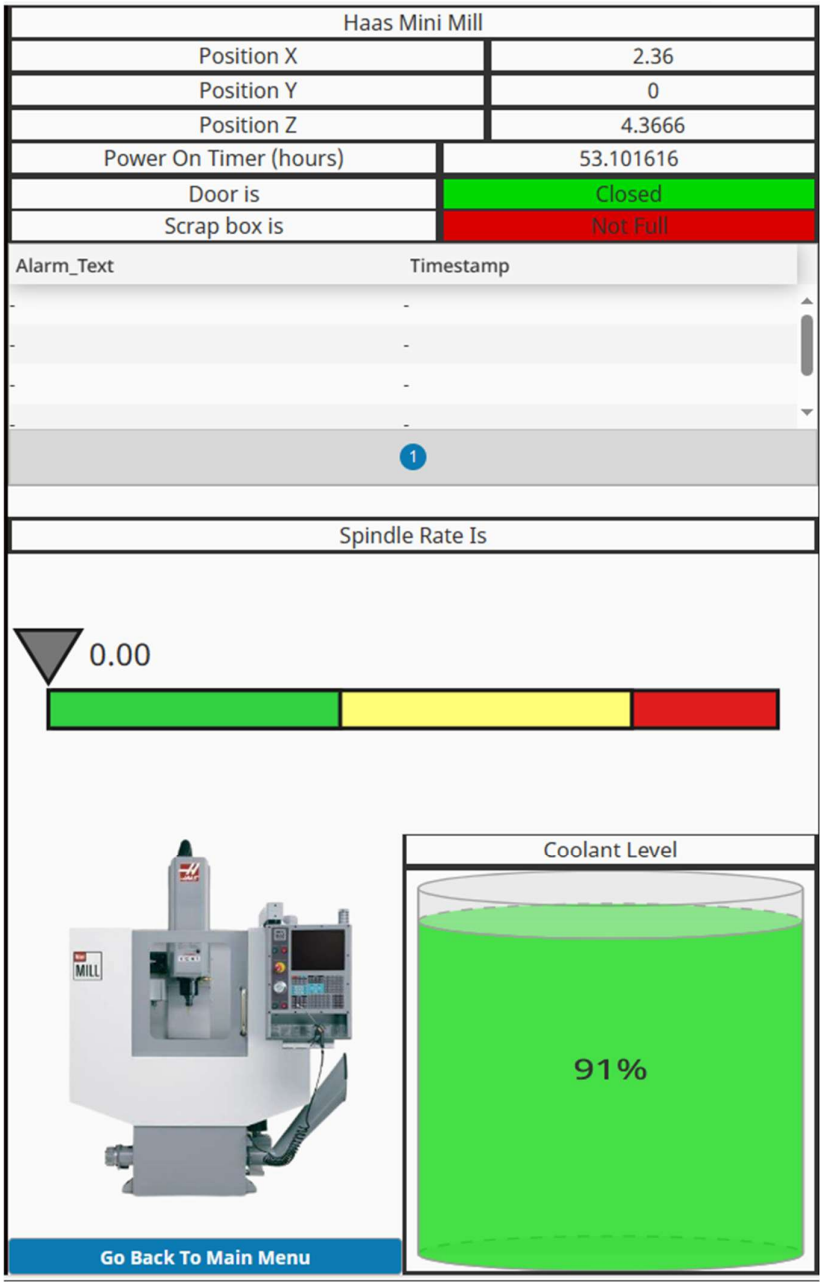

Figure 14. Machine view of the Haas mill from the Ignition UI.

9 SUMMARY AND CONLUSIONS

The thesis addressed the inefficiencies associated with the manual monitoring of CNC machining equipment by developing a remote monitoring system. The system was developed by leveraging technologies, including programmable logic controllers, OPC UA, and Python scripts.

The remote monitoring system for CNC machining equipment was successfully designed and implemented. The system collected data from multiple CNC machines, both newer models with Ethernet connectivity and older models that utilized programmable relays and sensors.

The system facilitates real-time data collection, processing, and visualization. It gathers critical machine performance data, such as spindle RPM, coolant levels and door status, thereby providing actionable insights into the manufacturing processes.

The user interface allowed operators to easily view machine performance data in real time. Each machine had a dedicated section in the user interface. The interface was compatible with both desktop computers and mobile devices, allowing for remote monitoring. Error messages and alerts were prominently displayed within the interface, along with the relevant error codes and timestamps, enabling timely responses to any issues.

The implementation of monitoring systems is a significant step toward the digitization of manufacturing operations. This system successfully addresses the inefficiencies and costs associated with manual monitoring by enabling automatic data collection. By implementing this solution, companies can enhance operational efficiency, minimize downtime, and make informed decisions to optimize production workflows.

Key findings from this work include the ease of integrating various technologies like PLCs, OPC UA, and Python scripts to create a cohesive monitoring

system. The design and implementation process highlighted the importance of interoperability and user-centric design in developing systems that are both functional and intuitive. Moreover, the project's success demonstrates the feasibility of applying similar technologies to other industrial contexts for further incorporation of smart manufacturing.

The project shows the potential of leveraging Industry 4.0 technologies in manufacturing to reduce operational costs, improve productivity, and enable predictive maintenance. The outcomes underscore the significance of adopting modern monitoring systems to stay competitive in a rapidly evolving industrial landscape.

While the system meets its intended objectives, future work could focus on expanding its capabilities, by incorporating machine learning algorithms for predictive maintenance. Additionally, extending the system to support diverse CNC machine brands and configurations would further increase its applicability across industries.

Most of my sources were academic papers, journal articles and books, which I consider highly reliable. While some of these sources were older than desired, their content remains relevant and up-to-date.

Additionally, I used several online sources from companies that sell CNC equipment or provide related services, such as FastCut, MachMotion, and Haas Automation. These companies have an incentive to highlight the benefits of CNC equipment and remote monitoring systems, which may introduce a bias in their publications. However, I ensured that the information obtained from these sources was factually accurate by cross-referencing it with academic literature and other independent sources. As a result, I believe the information used in this thesis is both reliable and well-supported.

Before working on this project, my experience with Python, CNC machines, and IoT systems was limited, and I had no prior experience with OPC UA. However, throughout the course of this project, I was able to significantly

expand my knowledge of all these technologies. I gained practical experience in Python scripting, integrating CNC machines into IoT systems using communication protocols, and working with the OPC UA framework, which has significantly improved my technical skills.

REFERENCES

Ajayi, O., Bagula, A., & Maluleke, H. (2022). The Fourth Industrial Revolution: A technological wave of change. In M. Gordan, K. Ghaedi, & V. Saleh (Eds.), Industry 4.0 - Perspectives and Applications (pp. 1–22). IntechOpen. https://doi.org/10.5772/intechopen.106209

Analytics Vidhya (n.d.) Introduction to Python. Retrieved December 30, 2024, from https://courses.analyticsvidhya.com/courses/introduction-to-data-science

Bolton, W. (2015). Programmable logic controllers (6th ed.). Elsevier.

Dhar, A. (2021). Introducing predictive maintenance with SAP predictive asset insights. Rheinwerk Publishing Inc.

Dixon, B., & Walker, J. (2019). CNC machines and their importance in manufacturing (10th ed.). The Goodheart-Willcox Company, Inc.

Doyle, F., & Cosgrove, J. (2019). Steps towards digitization of manufacturing in an SME environment. Procedia Manufacturing, 38, 540–547. https://doi.org/10.1016/j.promfg.2020.01.068

Edstrom, D (2014). MTConnect. Zurawski, R. (Ed.). Industrial communication technology handbook (2nd ed., Section 4, Chapter 24) CRC Press.

FastCut. (2024). Assessing the advantages of CNC machines in modern manufacturing. https://fastcut.co/news/advantages-of-cnc-machine/

Groover, M. (2015). Automation, production systems, and computer-integrated manufacturing (4th ed.). Pearson.

Haas Automation. (n.d.a). Mini Mill-EDU. Retrieved January 15, 2025, from https://www.haascnc.com/machines/vertical-mills/mini-mills/models/minimill-edu.html

Haas Automation. (n.d.b). Toolroom Lathe TL-1. Retrieved January 15, 2025, from https://www.haascnc.com/machines/lathes/toolroom-lathe/models/tl-1.html

ifm efector inc. (n.d.a). Technology overview - inductive sensors. Retrieved January 15, 2025, from https://www.ifm.com/us/en/us/learn-more/inductive/compact-inductive-sensors/technology

ifm efector inc. (n.d.b). Technology overview - capacitive sensors. Retrieved January 15, 2025, from https://www.ifm.com/us/en/us/overview/capacitive/technology-overview/technology-overview

Invicktus (2023). The Power Couple: Using Java and Python to Power Advanced IT Applications. Retrieved January 16, 2025 from

https://www.invicktus.com/blog/the-power-couple-using-java-and-python-to-power-advanced-it-applications/

MachMotion. (n.d.). Service. Retrieved January 15, 2025, from https://machmotion.com/support-main/

Margaza, D., & Yurchenko, Y. (2024). Internet of Things (IoT) technologies: Features, development prospects, and potential threats. Challenges and Issues of Modern Science, 3, 167–172. https://cims.fti.dp.ua/j/article/view/200

Martins, A., Lucas, J., Costelha, H., & Neves, C. (2023). CNC machines integration in smart factories using OPC UA. Journal of Industrial Information Integration, 34. https://doi.org/10.1016/j.jii.2023.100482

Miller, R., & Miller, R. (2004). Audel automated machines and toolmaking (5th ed.). Wiley.

Morehead State University. (n.d.). About Morehead State University. Retrieved January 15, 2025, from https://www.moreheadstate.edu

Python Software Foundation. (n.d.). Python: The programming language. Retrieved January 15, 2025, from https://www.python.org/doc/essays/blurb/

Rinaldi, J. (2016). OPC UA-unified architecture: The Everyman's guide to the most important information technology in industrial automation. CreateSpace Independent Publishing Platform.

Rinaldi, J. (2018). EtherNet/IP: The everyman's guide to the most widely used manufacturing protocol. CreateSpace Independent Publishing Platform.

Schulte, B., Fast, H., Flatt, H., Kleinhans, C., & Schulte, R. (2023, July 18–20). Low-threshold retrofit strategy for CNC machines: A new process data acquisition approach. 2023 IEEE 21st International Conference on Industrial Informatics, Lemgo, Germany. https://doi.org/10.1109/INDIN51400.2023.10218263

van Dinter, R., Tekinerdogan, B., & Catal, C. (2022). Predictive maintenance using digital twins: A systematic literature review. Information and Software Technology, 151. https://doi.org/10.1016/j.infsof.2022.107008

Yan, W., Wang, J., Lu, S., Zhou, M., & Peng, X. (2023). A review of real-time fault diagnosis methods for industrial smart manufacturing. Processes, 11, 369 https://doi.org/10.3390/pr11020369

APPENDIX 1: PYTHON SCRIPT FOR THE MILL

```
import socket
import sys
import binascii
from opcua import ua
import time
from opcua import Client
import re
import urllib.request
import xml.etree.ElementTree as ET

server_url = "opc.tcp://127.0.0.1:49321"
#client = Client(server_url)
#client.connect()
HOST = "192.168.1.56"
PORT = 5051
target_url = "http://192.168.1.56:8082/Current"
send = False
DataLength = 25000000000
index = 1
floatMixedNumbers = 404
NewData = 404
lang = 0

Coolant_level = "Undefined" #?Q600 13013
Individual_tool_ID = "Undefined" #"?Q600 8550
Year_month_day = "Undefined" #"?Q600 3011
Spindle_RPM = "Undefined" #"?Q600 3027

Machine_positionX = "Undefined" #"?Q600 5021
Machine_positionY = "Undefined" #"?Q600 5022
Machine_positionZ = "Undefined" #"?Q600 5023

Work_positionX = "Undefined" #"?Q600 5041
Work_positionY = "Undefined" #"?Q600 5042
Work_positionZ = "Undefined" #"?Q600 5043
Door = "Undefined"
Power_on_timer= "Undefined"
button= "Undefined"


def sendMessage(mesg):
    """Function to send a message to the CNC machine and
receive a response."""
    try:
        with socket.socket(socket.AF_INET,
socket.SOCK_STREAM) as s:  # Create a socket object using
TCP/IP
```

```
            s.connect((HOST, PORT))  # Connect to the ma-
chine at specified IP and port
            client_message = mesg + "\r"  # Append car-
riage return to message for protocol compliance
            client_message_as_byte_array = client_mes-
sage.encode('ascii')  # Convert string message to bytes
            s.sendall(client_message_as_byte_array)  #
Send the encoded message
            data = s.recv(DataLength)  # Receive a reply
from the machine
            data = data.decode('ascii')  # Decode the re-
ply from bytes to ASCII
            if len(data) < 3:  # Check if the received
data is sufficient
                data = "Invalid"
            s.close()  # Close the socket connection
            return data
    except:
        s.close()  # Ensure socket is closed in case of
error
        print("Error occurred while sending or receiving
message")
        data = "Invalid"
        time.sleep(1)
        return data

def ProcessReplyToFMN(Reply):
    """
    This function processes a string reply from a device
to extract and convert
    numeric values found within the string into a single
float value.
    """

    try:
        # Use regular expression to find all numbers,
capturing both floating-point numbers and integers.
        numbers_float = re.findall(r'\d+\.\d+|\d+', Re-
ply)

        # Convert each extracted string number to a float
if it contains a decimal point, otherwise to an integer.
        mixed_numbers = [float(num) if '.' in num else
int(num) for num in numbers_float]

        # Convert the list of numbers into a string and
remove the square brackets typical of Python list repre-
sentations.
        strMixedNumbers = str(mixed_numbers)
        strMixedNumbers = strMixedNumbers.strip('[]')
```

```
        # Convert the processed string back into a float.
This float is the final single numeric value derived from
the Reply.
        floatMixedNumbers = float(strMixedNumbers)

        return floatMixedNumbers

    except:
        # If an error occurs during the conversion pro-
cess, print an error message and pause execution for 1
second.
        print("Error occurred while converting data ")

def WriteToOPC(node_id, NewData):
    """
    This function connects to an OPC UA server and writes
data to a specified node.
    It handles both float and string data types by se-
lecting the appropriate VariantType.
    """

    try:
        # Create an OPC UA client instance using a glob-
ally defined server URL.
        client = Client(server_url)
        # Establish a connection to the OPC UA server.
        client.connect()

        # Retrieve the node object from the OPC server
using the provided node ID.
        node = client.get_node(node_id)

        # Check the data type of NewData and create a
DataValue instance with the appropriate VariantType.
        # If NewData is a float, use the Float variant
type. Otherwise, use the String variant type for other
data.
        if isinstance(NewData, float):
            valor = ua.DataValue(ua.Variant(NewData,
ua.VariantType.Float))
        else:
            valor = ua.DataValue(ua.Variant(NewData,
ua.VariantType.String))

        # Log the data value to be written for debugging
purposes.
        print(valor)
        # Set the node's value on the server with the
created DataValue instance.
        node.set_value(valor)
```

```
        # Disconnect from the OPC UA server once the data
is written.
        client.disconnect()
    except:
        # If any error occurs during the connection or
data writing process,
        # print an error message and pause the program
briefly to handle the error gracefully.
        print("Error occurred while writing to OPC")
        time.sleep(1)

def switch(lang):

    match lang:

        case 0:
            # Defining arrays for alarm list

            global Alarm_Numbers
            Alarm_Numbers = ["-","-","-","-","-"]
            global Alarm_Timestamps
            Alarm_Timestamps = ["-","-","-","-","-"]
            global Alarm_Texts
            Alarm_Texts = ["-","-","-","-","-"]

            lang = 1
            return lang

        case 1:
    #Spindle_RPM
            Spindle_RPM = sendMessage("?Q600 3027")
            print("Spindle_RPM",Spindle_RPM)

            # If the "sendMessage" function returns "In-
valid"
            # lang is set to the case number so the case
is repeated
            if Spindle_RPM == "Invalid" :

                print("Reply Invalid")
                lang = 1
                return lang
            else:
            #If the "sendMessage" function returns valid
data
            #The answer is writen to the OPC server
                print("Reply Valid")

                floatMixedNumbers = ProcessRe-
plyToFMN(Spindle_RPM)
```

```
                WriteToOPC("ns=2;s=Channel1.Device1.Spin-
dleRPM", floatMixedNumbers)
                lang += 1
                return lang
        case 2:
    #Machine_positionX

            Machine_positionX = sendMessage("?Q600 5021")
            print(Machine_positionX,"Machine_positionX")

            # If the "sendMessage" function returns "In-
valid"
            # lang is set to the case number so the case
is repeated

            if Machine_positionX == "Invalid" :

                print("Reply Invalid")
                lang = 2
                return lang
            else:
            #If the "sendMessage" function returns valid
data
            #The answer is writen to the OPC server

                print("Reply Valid")

                floatMixedNumbers = ProcessReplyToFMN(Ma-
chine_positionX)

                print("Conversion ok",floatMixedNumbers)

                WriteToOPC("ns=2;s=Channel1.Device1.Ma-
chine_positionX", floatMixedNumbers)
                lang += 1
                return lang
        case 3:
     #Machine_positionY

            Machine_positionY = sendMessage("?Q600 5022")
            print("Machine_psitionY",Machine_positionY)

            # If the "sendMessage" function returns "In-
valid"
            # lang is set to the case number so the case
is repeated
            if Machine_positionY == "Invalid" :

                print("Reply Invalid")
                lang = 3
                return lang
```

```
            else:
            #If the "sendMessage" function returns valid
data
            #The answer is writen to the OPC server

                print("Reply Valid")

                floatMixedNumbers = ProcessReplyToFMN(Ma-
chine_positionY)

                print("Conversion ok",floatMixedNumbers)

                WriteToOPC("ns=2;s=Channel1.Device1.Ma-
chine_positionY", floatMixedNumbers)
                lang += 1
                return lang
        case 4:
     #Machine_positionZ

            Machine_positionZ = sendMessage("?Q600 5023")
            print("Machine_positionZ",Machine_positionZ)

            # If the "sendMessage" function returns "In-
valid"
            # lang is set to the case number so the case
is repeated
            if Machine_positionZ == "Invalid" :

                print("Reply Invalid")
                lang = 4
                return lang
            else:
            #If the "sendMessage" function returns valid
data
            #The answer is writen to the OPC server

                print("Reply Valid")

                floatMixedNumbers = ProcessReplyToFMN(Ma-
chine_positionZ)

                print("Conversion ok",floatMixedNumbers)

                WriteToOPC("ns=2;s=Channel1.Device1.Ma-
chine_positionZ", floatMixedNumbers)
                lang += 1
                return lang
        case 5:
    #Work_positionX

            Work_positionX = sendMessage("?Q600 5041")
```

```
            print("Work_positionX",Work_positionX)

            # If the "sendMessage" function returns "In-
valid"
            # lang is set to the case number so the case
is repeated
            if Work_positionX == "Invalid" :

                print("Reply Invalid")
                lang = 5
                return lang
            else:
            #If the "sendMessage" function returns valid
data
            #The answer is writen to the OPC server

                print("Reply Valid")

                floatMixedNumbers = ProcessRe-
plyToFMN(Work_positionX)

                print("Conversion ok",floatMixedNumbers)

                WriteToOPC("ns=2;s=Channel1.De-
vice1.Work_positionX", floatMixedNumbers)
                lang += 1
                return lang
        case 6:
    #Work_positionY

            Work_positionY = sendMessage("?Q600 5042")
            print("Work_positionY",Work_positionY)

            # If the "sendMessage" function returns "In-
valid"
            # lang is set to the case number so the case
is repeated
            if Work_positionY == "Invalid" :

                print("Reply Invalid")
                lang = 6
                return lang
            else:
            #If the "sendMessage" function returns valid
data
            #The answer is writen to the OPC server

                print("Reply Valid")

                floatMixedNumbers = ProcessRe-
plyToFMN(Work_positionY)
```

```
                print("Conversion ok",floatMixedNumbers)

                WriteToOPC("ns=2;s=Channel1.De-
vice1.Work_positionY", floatMixedNumbers)
                lang += 1
                return lang
        case 7:
    #Work_positionZ

            Work_positionZ = sendMessage("?Q600 5043")
            print("Work_positionZ",Work_positionZ)

            # If the "sendMessage" function returns "In-
valid"
            # lang is set to the case number so the case
is repeated
            if Work_positionZ == "Invalid" :

                print("Reply Invalid")
                lang = 7
                return lang
            else:
            #If the "sendMessage" function returns valid
data
            #The answer is writen to the OPC server

                print("Reply Valid")

                floatMixedNumbers = ProcessRe-
plyToFMN(Work_positionZ)

                print("Conversion ok",floatMixedNumbers)

                WriteToOPC("ns=2;s=Channel1.De-
vice1.Work_positionZ", floatMixedNumbers)
                lang += 1
                return lang
        case 8:
    #Coolant_level

            Coolant_level = sendMessage("?Q600 13013")
            print("Coolant_level",Coolant_level)


            # If the "sendMessage" function returns "In-
valid"
            # lang is set to the case number so the case
is repeated
            if Coolant_level == "Invalid" :
```

```
                print("Reply Invalid")
                lang = 8
                return lang
            else:
            #If the "sendMessage" function returns valid
data
            #The answer is writen to the OPC server

                print("Reply Valid")

                floatMixedNumbers = ProcessRe-
plyToFMN(Coolant_level)

                print("Conversion ok",floatMixedNumbers)

                WriteToOPC("ns=2;s=Channel1.Device1.Cool-
ant_level", floatMixedNumbers)
                lang += 1
                return lang

        case 9:
        #Door
             Door = sendMessage("?Q600 11066")
             print("Door",Door)

             # If the "sendMessage" function returns "In-
valid"
             # lang is set to the case number so the case
is repeated
             if Door == "Invalid" :

                print("Reply Invalid")
                lang = 9
                return lang
             else:
            #If the "sendMessage" function returns valid
data
            #The answer is writen to the OPC server

                print("Reply Valid")

                floatMixedNumbers = ProcessRe-
plyToFMN(Door)

                print("Conversion ok",floatMixedNumbers)

                WriteToOPC("ns=2;s=Channel1.De-
vice1.Door", floatMixedNumbers)
                lang += 1
                return lang
        case 10:
```

```
        #Power_on_timer
             Power_on_timer  = sendMessage("?Q600 3002")
             print("Power_on_timer ",Power_on_timer )

             # If the "sendMessage" function returns "In-
valid"
             # lang is set to the case number so the case
is repeated
             if Power_on_timer  == "Invalid" :

                print("Reply Invalid")
                lang = 10
                return lang
             else:
            #If the "sendMessage" function returns valid
data
            #The answer is writen to the OPC server

                print("Reply Valid")

                floatMixedNumbers = ProcessRe-
plyToFMN(Power_on_timer )

                print("Conversion ok",floatMixedNumbers)

                WriteToOPC("ns=2;s=Channel1.De-
vice1.Power_on_timer", floatMixedNumbers)
                lang = 11
                return lang

        case 11:
            response = urllib.request.urlopen(target_url)
            xml_data = response.read()

    # Parse the raw XML data into a structured Ele-
mentTree object.
            root = ET.fromstring(xml_data)

    # Define namespaces used in the XML for more precise
searching.
            namespaces = {
        'm': 'urn:mtconnect.org:MTConnectStreams:1.2',  #
Namespace for MTConnectStreams 1.2
    }
    # Search the XML tree to find all 'DeviceStream' ele-
ments defined under the 'm' namespace.
            device_streams = root.findall('.//m:Devic-
eStream', namespaces)
            print(device_streams)
```

```
    # Iterate over each 'DeviceStream' element to extract
and print its 'name' and 'uuid' attributes.
            for device_stream in device_streams:
                name = device_stream.attrib.get('name')
                uuid = device_stream.attrib.get('uuid')
                print(f"DeviceStream Name: {name}, UUID:
{uuid}")

            rapid_overrides = root.findall('.//m:Ax-
isFeedrate[@name="RapidOverride"]', namespaces)
            feedrate_overrides = root.findall('.//m:Path-
Feedrate[@name="FeedrateOverride"]', namespaces)
            spindle_speed_overrides =
root.findall('.//m:SpindleSpeed[@name="Spin-
dleSpeedOverride"]', namespaces)
            for rapid_override in rapid_overrides:
                print("Rapid Override value:",
rapid_override.text)
                WriteToOPC("ns=2;s=Channel1.De-
vice1.Rapid_Override", rapid_override.text)
            for feedrate_override in feedrate_overrides:
                print("Feedrate Override value:",
feedrate_override.text)
                WriteToOPC("ns=2;s=Channel1.De-
vice1.Feedrate_Override", feedrate_override.text)
            for spindle_speed_override in spin-
dle_speed_overrides:
                print("Spindle Speed Override value:",
spindle_speed_override.text)
                WriteToOPC("ns=2;s=Channel1.Device1.Spin-
dle_Speed_Override", spindle_speed_override.text)
            lang = 12
            return lang
        case 12:
        #Spindle load
             SpindleLoad  = sendMessage("?Q600 ")
             print("------------------Spindle load-------
------------",SpindleLoad )


             # If the "sendMessage" function returns "In-
valid"
             # lang is set to the case number so the case
is repeated
             if SpindleLoad  == "Invalid" :

                print("Reply Invalid")
                lang = 11
                return lang
             else:
```

```
            #If the "sendMessage" function returns valid
data
            #The answer is writen to the OPC server

                print("Reply Valid")

                floatMixedNumbers = ProcessRe-
plyToFMN(SpindleLoad)

                print("Conversion ok",floatMixedNumbers)

                WriteToOPC("ns=2;s=Channel1.Device1.Spin-
dleLoad", floatMixedNumbers)
                lang = 13
                return lang
        case 13:

            try:
                # Attempt to open a URL to read XML data
from a specified endpoint.
                response = urllib.request.urlopen(tar-
get_url)
                xml_data = response.read()
                # Parse the raw XML data into a struc-
tured ElementTree object.
                root = ET.fromstring(xml_data)
                # Define namespaces used in the XML for
more precise searching.
                namespaces = {
                    'm': 'urn:mtconnect.org:MTCon-
nectStreams:1.2',  # Namespace for MTConnectStreams 1.2
                    }
                # Search the XML tree to find all 'Devic-
eStream' elements defined under the 'm' namespace.
                device_streams = root.findall('.//m:De-
viceStream', namespaces)
                print(device_streams)
                # Iterate over each 'DeviceStream' ele-
ment to extract and print its 'name' and 'uuid' attrib-
utes.
                for device_stream in device_streams:
                    name = device_stream.at-
trib.get('name')
                    uuid = device_stream.at-
trib.get('uuid')
                    print(f"DeviceStream Name: {name},
UUID: {uuid}")
                # Search the XML for all 'Alarm' elements
within the specified namespace.
                alarms = root.findall('.//m:Alarm',
namespaces)
```

```
                AlarmCounter = 0
                if not alarms:
                    # Check if the alarm list is empty
and print a message if no alarms were found.
                    print("Alarms array is empty.")
                else:
                    # Process each 'Alarm' element to ex-
tract and print various attributes.
                    for alarm in alarms:
                        alarm_number = alarm.at-
trib.get('alarmNumber')
                        timestamp = alarm.at-
trib.get('timestamp')
                        alarm_text = alarm.text.strip()
# Strip any extra whitespace from the alarm text.

                        # Append alarm details to respec-
tive lists for later use.
                        Alarm_Numbers.append(alarm_num-
ber)
                        Alarm_Timestamps.ap-
pend(timestamp)
                        Alarm_Texts.append(alarm_text)
                        # Increment the alarm counter for
each processed alarm.
                        AlarmCounter += 1
                        print(AlarmCounter)
                print(f"Alarm Number: {Alarm_Numbers}")

                # Write each extracted alarm attribute to
an OPC node, adjusting indices based on AlarmCounter.
                WriteToOPC("ns=2;s=Channel1.De-
vice1.Alarm_Number1", Alarm_Numbers[4 + AlarmCounter])
                WriteToOPC("ns=2;s=Channel1.De-
vice1.Alarm_Number2", Alarm_Numbers[3 + AlarmCounter])
                WriteToOPC("ns=2;s=Channel1.De-
vice1.Alarm_Number3", Alarm_Numbers[2 + AlarmCounter])
                WriteToOPC("ns=2;s=Channel1.De-
vice1.Alarm_Number4", Alarm_Numbers[1 + AlarmCounter])
                WriteToOPC("ns=2;s=Channel1.De-
vice1.Alarm_Number5", Alarm_Numbers[0 + AlarmCounter])

                WriteToOPC("ns=2;s=Channel1.De-
vice1.Alarm_Timestamp1", Alarm_Timestamps[4 + Alarm-
Counter])
                WriteToOPC("ns=2;s=Channel1.De-
vice1.Alarm_Timestamp2", Alarm_Timestamps[3 + Alarm-
Counter])
                WriteToOPC("ns=2;s=Channel1.De-
vice1.Alarm_Timestamp3", Alarm_Timestamps[2 + Alarm-
Counter])
```

```
                WriteToOPC("ns=2;s=Channel1.De-
vice1.Alarm_Timestamp4", Alarm_Timestamps[1 + Alarm-
Counter])
                WriteToOPC("ns=2;s=Channel1.De-
vice1.Alarm_Timestamp5", Alarm_Timestamps[0 + Alarm-
Counter])

                WriteToOPC("ns=2;s=Channel1.De-
vice1.Alarm_Text1", Alarm_Texts[4 + AlarmCounter])
                WriteToOPC("ns=2;s=Channel1.De-
vice1.Alarm_Text2", Alarm_Texts[3 + AlarmCounter])
                WriteToOPC("ns=2;s=Channel1.De-
vice1.Alarm_Text3", Alarm_Texts[2 + AlarmCounter])
                WriteToOPC("ns=2;s=Channel1.De-
vice1.Alarm_Text4", Alarm_Texts[1 + AlarmCounter])
                WriteToOPC("ns=2;s=Channel1.De-
vice1.Alarm_Text5", Alarm_Texts[0 + AlarmCounter])



                time.sleep(1)
                lang = 0
                return lang
            except:
                # Handle any exceptions that occur during
the execution of the try block.
                print("Error occured while reading and
writing alarms")
                time.sleep(1)
                lang = 0
                return lang

while 1==1:

    lang = switch(lang)
```

APPENDIX 2: PYTHON SCRIPT FOR THE LATHE

```
import socket
import sys
import binascii
from opcua import ua
import time
from opcua import Client
import re
import urllib.request
import xml.etree.ElementTree as ET

server_url = "opc.tcp://127.0.0.1:49321"
#client = Client(server_url)
#client.connect()
HOST = "192.168.1.58"
PORT = 5051
target_url = "http://192.168.1.58:8082/Current"
send = False
DataLength = 25000000000
index = 1
floatMixedNumbers = 404
NewData = 404
lang = 0

Coolant_level = "Undefined" #?Q600 13013
Individual_tool_ID = "Undefined" #"?Q600 8550
Year_month_day = "Undefined" #"?Q600 3011
Spindle_RPM = "Undefined" #"?Q600 3027

Machine_positionX = "Undefined" #"?Q600 5021
Machine_positionY = "Undefined" #"?Q600 5022
Machine_positionZ = "Undefined" #"?Q600 5023

Work_positionX = "Undefined" #"?Q600 5041
Work_positionY = "Undefined" #"?Q600 5042
Work_positionZ = "Undefined" #"?Q600 5043
Rigth_Door_Open = "Undefined"
Left_Door_Open = "Undefined"
Power_on_timer= "Undefined"
button= "Undefined"
#Programmable_stop_with_message= "Undefined" #?Q600 3006

def sendMessage(mesg):
    """Function to send a message to the CNC machine and
receive a response."""
    try:
```

```
        with socket.socket(socket.AF_INET,
socket.SOCK_STREAM) as s:  # Create a socket object using
TCP/IP
            s.connect((HOST, PORT))  # Connect to the ma-
chine at specified IP and port
            client_message = mesg + "\r"  # Append car-
riage return to message for protocol compliance
            client_message_as_byte_array = client_mes-
sage.encode('ascii')  # Convert string message to bytes
            s.sendall(client_message_as_byte_array)  #
Send the encoded message
            data = s.recv(DataLength)  # Receive a reply
from the machine
            data = data.decode('ascii')  # Decode the re-
ply from bytes to ASCII
            if len(data) < 3:  # Check if the received
data is sufficient
                data = "Invalid"
            s.close()  # Close the socket connection
            return data
    except:
        s.close()  # Ensure socket is closed in case of
error
        print("Error occurred while sending or receiving
message")
        data = "Invalid"
        time.sleep(1)
        return data

def ProcessReplyToFMN(Reply):
    """
    This function processes a string reply from a device
to extract and convert
    numeric values found within the string into a single
float value.
    """

    try:
        # Use regular expression to find all numbers,
capturing both floating-point numbers and integers.
        numbers_float = re.findall(r'\d+\.\d+|\d+', Re-
ply)

        # Convert each extracted string number to a float
if it contains a decimal point, otherwise to an integer.
        mixed_numbers = [float(num) if '.' in num else
int(num) for num in numbers_float]

        # Convert the list of numbers into a string and
remove the square brackets typical of Python list repre-
sentations.
```

```
        strMixedNumbers = str(mixed_numbers)
        strMixedNumbers = strMixedNumbers.strip('[]')

        # Convert the processed string back into a float.
This float is the final single numeric value derived from
the Reply.
        floatMixedNumbers = float(strMixedNumbers)

        return floatMixedNumbers

    except:
        # If an error occurs during the conversion pro-
cess, print an error message and pause execution for 1
second.
        print("Error occurred while converting data ")

def WriteToOPC(node_id, NewData):
    """
    This function connects to an OPC UA server and writes
data to a specified node.
    It handles both float and string data types by se-
lecting the appropriate VariantType.
    """

    try:
        # Create an OPC UA client instance using a glob-
ally defined server URL.
        client = Client(server_url)
        # Establish a connection to the OPC UA server.
        client.connect()

        # Retrieve the node object from the OPC server
using the provided node ID.
        node = client.get_node(node_id)

        # Check the data type of NewData and create a
DataValue instance with the appropriate VariantType.
        # If NewData is a float, use the Float variant
type. Otherwise, use the String variant type for other
data.
        if isinstance(NewData, float):
            valor = ua.DataValue(ua.Variant(NewData,
ua.VariantType.Float))
        else:
            valor = ua.DataValue(ua.Variant(NewData,
ua.VariantType.String))

        # Log the data value to be written for debugging
purposes.
        print(valor)
```

```
        # Set the node's value on the server with the
created DataValue instance.
        node.set_value(valor)

        # Disconnect from the OPC UA server once the data
is written.
        client.disconnect()
    except:
        # If any error occurs during the connection or
data writing process,
        # print an error message and pause the program
briefly to handle the error gracefully.
        print("Error occurred while writing to OPC")
        time.sleep(1)

def switch(lang):

    match lang:

        case 0:

            global Alarm_Numbers
            Alarm_Numbers = ["-","-","-","-","-"]
            global Alarm_Timestamps
            Alarm_Timestamps = ["-","-","-","-","-"]
            global Alarm_Texts
            Alarm_Texts = ["-","-","-","-","-"]

            lang = 1
            return lang

        case 1:
    #Spindle_RPM
            Spindle_RPM = sendMessage("?Q600 3027")
            print("Spindle_RPM",Spindle_RPM)

            if Spindle_RPM == "Invalid" :

                print("Reply Invalid")
                lang = 1
                return lang
            else:

                print("Reply Valid")

                floatMixedNumbers = ProcessRe-
plyToFMN(Spindle_RPM)

                WriteToOPC("ns=2;s=Channel1.Lathe.Spin-
dleRPM_HaasLathe", floatMixedNumbers)
                lang += 1
```

```
                return lang
        case 2:
    #Machine_positionX

            Machine_positionX = sendMessage("?Q600 5021")
            print(Machine_positionX,"Machine_positionX")

            if Machine_positionX == "Invalid" :

                print("Reply Invalid")
                lang = 2
                return lang
            else:

                print("Reply Valid")

                floatMixedNumbers = ProcessReplyToFMN(Ma-
chine_positionX)

                print("Conversion ok",floatMixedNumbers)

                WriteToOPC("ns=2;s=Channel1.Lathe.Ma-
chine_positionX_HaasLathe", floatMixedNumbers)
                lang += 1
                return lang
        case 3:
     #Machine_positionY

            Machine_positionY = sendMessage("?Q600 5022")
            print("Machine_psitionY",Machine_positionY)

            if Machine_positionY == "Invalid" :

                print("Reply Invalid")
                lang = 3
                return lang
            else:

                print("Reply Valid")

                floatMixedNumbers = ProcessReplyToFMN(Ma-
chine_positionY)

                print("Conversion ok",floatMixedNumbers)

                WriteToOPC("ns=2;s=Channel1.Lathe.Ma-
chine_positionY_HaasLathe", floatMixedNumbers)
                lang += 1
                return lang
        case 4:
     #Machine_positionZ
```

```
            Machine_positionZ = sendMessage("?Q600 5023")
            print("Machine_positionZ",Machine_positionZ)

            if Machine_positionZ == "Invalid" :

                print("Reply Invalid")
                lang = 4
                return lang
            else:

                print("Reply Valid")

                floatMixedNumbers = ProcessReplyToFMN(Ma-
chine_positionZ)

                print("Conversion ok",floatMixedNumbers)

                WriteToOPC("ns=2;s=Channel1.Lathe.Ma-
chine_positionZ_HaasLathe", floatMixedNumbers)
                lang += 1
                return lang
        case 5:
    #Work_positionX

            Work_positionX = sendMessage("?Q600 5041")
            print("Work_positionX",Work_positionX)


            if Work_positionX == "Invalid" :

                print("Reply Invalid")
                lang = 5
                return lang
            else:

                print("Reply Valid")

                floatMixedNumbers = ProcessRe-
plyToFMN(Work_positionX)

                print("Conversion ok",floatMixedNumbers)

                WriteToOPC("ns=2;s=Chan-
nel1.Lathe.Work_positionX_HaasLathe", floatMixedNumbers)
                lang += 1
                return lang
        case 6:
    #Work_positionY

            Work_positionY = sendMessage("?Q600 5042")
```

```
            print("Work_positionY",Work_positionY)

            if Work_positionY == "Invalid" :

                print("Reply Invalid")
                lang = 6
                return lang
            else:

                print("Reply Valid")

                floatMixedNumbers = ProcessRe-
plyToFMN(Work_positionY)

                print("Conversion ok",floatMixedNumbers)

                WriteToOPC("ns=2;s=Chan-
nel1.Lathe.Work_positionY_HaasLathe", floatMixedNumbers)
                lang += 1
                return lang
        case 7:
    #Work_positionZ

            Work_positionZ = sendMessage("?Q600 5043")
            print("Work_positionZ",Work_positionZ)


            if Work_positionZ == "Invalid" :

                print("Reply Invalid")
                lang = 7
                return lang
            else:

                print("Reply Valid")

                floatMixedNumbers = ProcessRe-
plyToFMN(Work_positionZ)

                print("Conversion ok",floatMixedNumbers)

                WriteToOPC("ns=2;s=Chan-
nel1.Lathe.Work_positionZ_HaasLathe", floatMixedNumbers)
                lang += 1
                return lang
        case 8:
    #Coolant_level

            Coolant_level = sendMessage("?Q600 13013")
            print("Coolant_level",Coolant_level)
```

```
            if Coolant_level == "Invalid" :

                print("Reply Invalid")
                lang = 8
                return lang
            else:

                print("Reply Valid")

                floatMixedNumbers = ProcessRe-
plyToFMN(Coolant_level)

                print("Conversion ok",floatMixedNumbers)

                WriteToOPC("ns=2;s=Channel1.Lathe.Cool-
ant_level_HaasLathe", floatMixedNumbers)
                lang += 1
                return lang

        case 9:
        #Rigth_Door_Open
             Rigth_Door_Open = sendMessage("?Q600 11066")
             print("Rigth_Door_Open",Rigth_Door_Open)

             if Rigth_Door_Open == "Invalid" :

                print("Reply Invalid")
                lang = 9
                return lang
             else:

                print("Reply Valid")

                floatMixedNumbers = ProcessRe-
plyToFMN(Rigth_Door_Open)

                print("Conversion ok",floatMixedNumbers)

                WriteToOPC("ns=2;s=Chan-
nel1.Lathe.Rigth_Door_Open_HaasLathe", floatMixedNumbers)
                lang += 1
                return lang
        case 10:
        #Left_Door_Open
             Left_Door_Open = sendMessage("?Q600 11067")
             print("Left_Door",Left_Door_Open)

             if Left_Door_Open == "Invalid" :

                print("Reply Invalid")
                lang = 10
```

```
                return lang
            else:

                print("Reply Valid")

                floatMixedNumbers = ProcessRe-
plyToFMN(Left_Door_Open)

                print("Conversion ok",floatMixedNumbers)

                WriteToOPC("ns=2;s=Chan-
nel1.Lathe.Left_Door_Open_HaasLathe", floatMixedNumbers)
                lang += 1
                return lang
        case 11:
        #Power_on_timer
            Power_on_timer  = sendMessage("?Q600 3002")
            print("Power_on_timer ",Power_on_timer )

            if Power_on_timer  == "Invalid" :

                print("Reply Invalid")
                lang = 11
                return lang
            else:

                print("Reply Valid")

                floatMixedNumbers = ProcessRe-
plyToFMN(Power_on_timer )

                print("Conversion ok",floatMixedNumbers)

                WriteToOPC("ns=2;s=Chan-
nel1.Lathe.Power_on_timer_HaasLathe", floatMixedNumbers)
                lang = 12
                return lang
        case 12:
            response = urllib.request.urlopen(target_url)
            xml_data = response.read()

    # Parse the raw XML data into a structured Ele-
mentTree object.
            root = ET.fromstring(xml_data)

    # Define namespaces used in the XML for more precise
searching.
            namespaces = {
        'm': 'urn:mtconnect.org:MTConnectStreams:1.2',  #
Namespace for MTConnectStreams 1.2
    }
```

```
    # Search the XML tree to find all 'DeviceStream' ele-
ments defined under the 'm' namespace.
            device_streams = root.findall('.//m:Devic-
eStream', namespaces)
            print(device_streams)

    # Iterate over each 'DeviceStream' element to extract
and print its 'name' and 'uuid' attributes.
            for device_stream in device_streams:
                name = device_stream.attrib.get('name')
                uuid = device_stream.attrib.get('uuid')
                print(f"DeviceStream Name: {name}, UUID:
{uuid}")
        # Search the XML for all 'Overrides' elements
within the specified namespace.
            rapid_overrides = root.findall('.//m:Ax-
isFeedrate[@name="RapidOverride"]', namespaces)
            feedrate_overrides = root.findall('.//m:Path-
Feedrate[@name="FeedrateOverride"]', namespaces)
            spindle_speed_overrides =
root.findall('.//m:SpindleSpeed[@name="Spin-
dleSpeedOverride"]', namespaces)
            for rapid_override in rapid_overrides:
                print("Rapid Override value:",
rapid_override.text)
                WriteToOPC("ns=2;s=Chan-
nel1.Lathe.Rapid_Override_Lathe", rapid_override.text)
            for feedrate_override in feedrate_overrides:
                print("Feedrate Override value:",
feedrate_override.text)
                WriteToOPC("ns=2;s=Chan-
nel1.Lathe.Feedrate_Override_Lathe", feedrate_over-
ride.text)
            for spindle_speed_override in spin-
dle_speed_overrides:
                print("Spindle Speed Override value:",
spindle_speed_override.text)
                WriteToOPC("ns=2;s=Channel1.Lathe.Spin-
dle_Speed_Override_Lathe", spindle_speed_override.text)
            lang = 13
            return lang
        case 13:
        #Spindle load
             Cycle  = sendMessage("?Q600 3023")
             print("Cycle", Cycle)


             # If the "sendMessage" function returns "In-
valid"
             # lang is set to the case number so the case
is repeated
```

```
             if Cycle  == "Invalid" :

                print("Reply Invalid")
                lang = 13
                return lang
             else:
            #If the "sendMessage" function returns valid
data
            #The answer is writen to the OPC server

                print("Reply Valid")

                floatMixedNumbers = ProcessReplyToFMN(Cy-
cle)

                print("Conversion ok",floatMixedNumbers)

                WriteToOPC("ns=2;s=Channel1.Lathe.Cy-
cle_Lathe", floatMixedNumbers)
                lang = 14
                return lang
        case 14:
        #Spindle load
             SpindleLoad  = sendMessage("?Q600 1098")
             print("Spindle Load",SpindleLoad )


             # If the "sendMessage" function returns "In-
valid"
             # lang is set to the case number so the case
is repeated
             if SpindleLoad  == "Invalid" :

                print("Reply Invalid")
                lang = 14
                return lang
             else:
            #If the "sendMessage" function returns valid
data
            #The answer is writen to the OPC server

                print("Reply Valid")

                floatMixedNumbers = ProcessRe-
plyToFMN(SpindleLoad)

                print("Conversion ok",floatMixedNumbers)

                WriteToOPC("ns=2;s=Channel1.Lathe.Spin-
dleLoad_Lathe", floatMixedNumbers)
                lang = 15
```

```
                return lang
        case 15 :
            try:
                # Attempt to open a URL to read XML data
from a specified endpoint.
                response = urllib.request.urlopen(tar-
get_url)
                xml_data = response.read()

                # Parse the raw XML data into a struc-
tured ElementTree object.
                root = ET.fromstring(xml_data)

                # Define namespaces used in the XML for
more precise searching.
                namespaces = {
                    'm': 'urn:mtconnect.org:MTCon-
nectStreams:1.2',  # Namespace for MTConnectStreams 1.2
                }

                # Search the XML tree to find all 'Devic-
eStream' elements defined under the 'm' namespace.
                device_streams = root.findall('.//m:De-
viceStream', namespaces)
                print(device_streams)
                # Iterate over each 'DeviceStream' ele-
ment to extract and print its 'name' and 'uuid' attrib-
utes.
                for device_stream in device_streams:
                    name = device_stream.at-
trib.get('name')
                    uuid = device_stream.at-
trib.get('uuid')
                    print(f"DeviceStream Name: {name},
UUID: {uuid}")

                # Search the XML for all 'Alarm' elements
within the specified namespace.
                alarms = root.findall('.//m:Alarm',
namespaces)
                AlarmCounter = 0

                if not alarms:
                    # Check if the alarm list is empty
and print a message if no alarms were found.
                    print("Alarms array is empty.")
                else:
                    # Process each 'Alarm' element to ex-
tract and print various attributes.
                    for alarm in alarms:
```

```
                        alarm_number = alarm.at-
trib.get('alarmNumber')
                        timestamp = alarm.at-
trib.get('timestamp')
                        alarm_text = alarm.text.strip()
# Strip any extra whitespace from the alarm text.

                        # Append alarm details to respec-
tive lists for later use.
                        Alarm_Numbers.append(alarm_num-
ber)
                        Alarm_Timestamps.ap-
pend(timestamp)
                        Alarm_Texts.append(alarm_text)

                        # Increment the alarm counter for
each processed alarm.
                        AlarmCounter += 1
                        print(AlarmCounter)

                print(f"Alarm Number: {Alarm_Numbers}")

                # Write each extracted alarm attribute to
an OPC node, adjusting indices based on AlarmCounter.
                WriteToOPC("ns=2;s=Chan-
nel1.Lathe.LatheAlarm_Number1", Alarm_Numbers[4 + Alarm-
Counter])
                WriteToOPC("ns=2;s=Chan-
nel1.Lathe.LatheAlarm_Number2", Alarm_Numbers[3 + Alarm-
Counter])
                WriteToOPC("ns=2;s=Chan-
nel1.Lathe.LatheAlarm_Number3", Alarm_Numbers[2 + Alarm-
Counter])
                WriteToOPC("ns=2;s=Chan-
nel1.Lathe.LatheAlarm_Number4", Alarm_Numbers[1 + Alarm-
Counter])
                WriteToOPC("ns=2;s=Chan-
nel1.Lathe.LatheAlarm_Number5", Alarm_Numbers[0 + Alarm-
Counter])

                WriteToOPC("ns=2;s=Chan-
nel1.Lathe.LatheAlarm_Timestamp1", Alarm_Timestamps[4 +
AlarmCounter])
                WriteToOPC("ns=2;s=Chan-
nel1.Lathe.LatheAlarm_Timestamp2", Alarm_Timestamps[3 +
AlarmCounter])
                WriteToOPC("ns=2;s=Chan-
nel1.Lathe.LatheAlarm_Timestamp3", Alarm_Timestamps[2 +
AlarmCounter])
```

```
                WriteToOPC("ns=2;s=Chan-
nel1.Lathe.LatheAlarm_Timestamp4", Alarm_Timestamps[1 +
AlarmCounter])
                WriteToOPC("ns=2;s=Chan-
nel1.Lathe.LatheAlarm_Timestamp5", Alarm_Timestamps[0 +
AlarmCounter])

                WriteToOPC("ns=2;s=Chan-
nel1.Lathe.LatheAlarm_Text7", Alarm_Texts[4 + Alarm-
Counter])
                WriteToOPC("ns=2;s=Chan-
nel1.Lathe.LatheAlarm_Text2", Alarm_Texts[3 + Alarm-
Counter])
                WriteToOPC("ns=2;s=Chan-
nel1.Lathe.LatheAlarm_Text3", Alarm_Texts[2 + Alarm-
Counter])
                WriteToOPC("ns=2;s=Chan-
nel1.Lathe.LatheAlarm_Text4", Alarm_Texts[1 + Alarm-
Counter])
                WriteToOPC("ns=2;s=Chan-
nel1.Lathe.LatheAlarm_Text5", Alarm_Texts[0 + Alarm-
Counter])

                # Pause for a second before resetting the
language switch to 0.
                time.sleep(1)
                lang = 0
                return lang
            except:
                # Handle any exceptions that occur during
the execution of the try block.
                print("Error occurred while reading and
writing alarms")
                time.sleep(1)
                lang = 0
                return lang

while index<100000000:
    lang = switch(lang)
    if lang == 0:

        index = 0

    else:
        index += 1
```