Development of Digital Infrastructure for a Digital Twin

Scientific report to obtain the degree M.Sc. at the Department of Scientific Instrumentation of the Ernst Abbe Hochschule in Jena.

**Supervised by** Prof. Dr. Ronny Gerbach  
Marko Tesanovic, M.Sc.

**Submitted by** Munish Reddy Modugula  
student number 646665

**Date of issue** 01.01.2023

**Submitted** Jena, the 01.09.2023

Statutory Declaration

**Affirmation**

Hereby, I certify that I have written this report independently and have not used any other tools or sources than those indicated. I affirm, to the best of my knowledge and belief, that I have not violated any copyright or other proprietary rights in this thesis. I certify that all ideas, techniques, quotations, or other material have been collected in accordance with standard citation practices. Furthermore, I affirm that the thesis has been neither passed on to third parties nor published. I have complied with all study regulations of the Technical University of Munich. Furthermore, I have followed the guidelines of good scientific practice.

**Notes on the Internet**

Throughout the work, the internet was used for research and review. Many of the keywords, references, and other information given here can be checked on the internet. However, no sources are given, since all statements made in this thesis are completely covered by the cited literature sources.

Place, Date Signature

Acknowledgment

Firstly, I would like to thank my academic supervisor Prof. Dr.-Ing. Ronny Gerbach, who gave me acceptance for my Master Thesis topic to support me as well as to provide me guidance whenever necessary. I am thankful to my academic supervisor who was available whenever needed.

In this document, I would like to thank the Institute of Bioseparation Engineering and especially my supervisor Marko Tesanovic for the inspiration guidance, and support throughout the work on this Thesis.

Furthermore, I would like to thank every member of the STT Team for the support and answering of my questions and to all the students, with whom I had the opportunity to work side by side during the 6 months.

Abstract

This project is about the creation of a digital twin system for a magnetic separation process, merging advanced technologies with industrial processes. The main goal is to create a virtual model that mirrors the real-world process and enhances its efficiency and transparency. The initial phase of the thesis is the development of communication between the field machines and the digital using Python-based API by data collection from PLC and subsequent storage in a PostgreSQL database. The API utilizes the OPC UA communication protocol for seamless interaction between the digital twin and the physical process. This collected data is then processed and organized, facilitating its later analysis. The second phase is containerizing the API using Docker for deployment. A significant challenge arises when deploying on the Raspberry Pi IIoT, because of its limited resources. To address this, a multi-architectural build is adopted, ensuring compatibility with the Pi's architecture. In the next phase docker swarm cluster was created, and Rev pi and the main server were added as nodes to build a connection between API and database. This cluster can be very useful in the future to run multiple applications and to scale the containers both horizontally and vertically. Integrating soft sensors in the digital twin and calculating the MNP concentration of magnetic nanoparticles using the density of the particles. To validate the work and to find the time complexity, multiple latency tests were performed creating different scenarios, and results were discussed. These tests provide very good insights into finding the differences between reading and writing data and show how multiple clients impact the system. Also, the validation of soft sensor tests was performed using the breakthrough curves. A foundational work was done on the development of GUI with Python pyqt5 library for digital transparency.

Table of Content

[Statutory Declaration I](#_Toc157241285)

[Acknowledgment II](#_Toc157241286)

[Abstract III](#_Toc157241287)

[List of figures VI](#_Toc157241288)

[List of Tables VIII](#_Toc157241289)

[Nomenclature Abbreviation IX](#_Toc157241290)

[1 Introduction 1](#_Toc157241291)

[2 Motivation and Objectives 3](#_Toc157241292)

[2.1. Research Trends 4](#_Toc157241293)

[3 Theoretical Background 5](#_Toc157241294)

[3.1. Process Knowledge 5](#_Toc157241295)

[3.2. Process Analytical Technology 8](#_Toc157241296)

[3.2.1. Integration of Soft Sensors in Process Monitoring 9](#_Toc157241297)

[3.3. Industry 4.0 10](#_Toc157241298)

[3.3.1. Cyber–Physical Systems 11](#_Toc157241299)

[3.3.2. Digital Twin 12](#_Toc157241300)

[3.3.3. Automation 13](#_Toc157241301)

[3.4. Communication Protocols 15](#_Toc157241302)

[3.4.1. TCP/IP 15](#_Toc157241303)

[3.4.2. Modbus 15](#_Toc157241304)

[3.4.3. OPC UA 16](#_Toc157241305)

[3.5. Pscopg2 17](#_Toc157241306)

[3.6. PostgreSQL 17](#_Toc157241307)

[3.7. Timescale DB 18](#_Toc157241308)

[3.8. Grafana 18](#_Toc157241309)

[3.9. Virtualisation 19](#_Toc157241310)

[3.9.1. Conventional Virtual Machines (VMs) 19](#_Toc157241311)

[3.9.2. Docker Virtualisation 20](#_Toc157241312)

[4 Materials and Methods 21](#_Toc157241313)

[4.1. RSMS 21](#_Toc157241314)

[4.1.1. Inline Process Analytic 23](#_Toc157241315)

[4.2. Programmable Logic Control 25](#_Toc157241316)

[4.3. Human Machine Interface 25](#_Toc157241317)

[4.4. Raspberry Pi 25](#_Toc157241318)

[4.5. Digital Infrastructure 26](#_Toc157241319)

[4.6. Computer-Setup 28](#_Toc157241320)

[4.7. Cyberspace 29](#_Toc157241321)

[4.7.1. Structure of Cyberspace 29](#_Toc157241322)

[4.7.2. Software Used 30](#_Toc157241323)

[4.8. Methodology for Technology Selection 31](#_Toc157241324)

[4.8.1. Weighting of Digital Twin Data processing specific Criteria 31](#_Toc157241325)

[4.8.2. Radar Chart Visualization 36](#_Toc157241326)

[4.9. Soft Sensor 37](#_Toc157241327)

[4.9.1. Density determination 37](#_Toc157241328)

[4.9.2. MNP Concentration 38](#_Toc157241329)

[5 Concept and Implementation 39](#_Toc157241330)

[5.1. Data Processing using Node-Red 40](#_Toc157241331)

[5.2. Overview of an API 41](#_Toc157241332)

[5.2.1. OPC UA client-server functioning 42](#_Toc157241333)

[5.2.2. Connection to OPC Server 43](#_Toc157241334)

[5.2.3. Node lists 44](#_Toc157241335)

[5.2.4. Connection to PostgreSQL database 45](#_Toc157241336)

[5.2.5. Data retrieval and storage 46](#_Toc157241337)

[5.2.6. Data Processing and Insertion 46](#_Toc157241338)

[6 Implementation of Docker 48](#_Toc157241339)

[6.1. Importance of Docker 48](#_Toc157241340)

[6.2. Creating Docker images 49](#_Toc157241341)

[6.3. Building the Docker Image 50](#_Toc157241342)

[6.4. Docker Volumes 51](#_Toc157241343)

[6.4.1. Docker Bind Volumes 52](#_Toc157241344)

[6.4.2. Docker Volume mounts 52](#_Toc157241345)

[6.5. Creating Docker Network 52](#_Toc157241346)

[6.6. GUI for Digital Transparency 55](#_Toc157241347)

[7 Tests and Results 56](#_Toc157241348)

[7.1. Validation of Soft sensor 56](#_Toc157241349)

[7.2. Latency Test Methodology 58](#_Toc157241350)

[7.3. Comparison of Read and Write Latencies 59](#_Toc157241351)

[7.4. Grafana Monitoring and Database Implications 62](#_Toc157241352)

[8 Conclusion 62](#_Toc157241353)

[9 References 64](#_Toc157241354)

[10 Appendix 66](#_Toc157241355)

[10.1. Final API code 66](#_Toc157241356)

[10.2. Latency test code 76](#_Toc157241357)

[10.3. GUI basic layout 79](#_Toc157241358)

List of figures

[Figure 3.1 Binding of Target Protein [1] 5](#_Toc157865833)

[Figure 3.2 Separation of Magnetic particles [1] 6](#_Toc157865834)

[Figure 3.3 Washing of particles [1] 7](#_Toc157865835)

[Figure 3.4 Elution of target particles [1] 7](#_Toc157865836)

[Figure 3.5 Schematic representation of measurement approaches in process analytics. A distinction is made between on-, in-, at- and off-line (Olivier Henry 2019). 8](#_Toc157865837)

[Figure 3.6 Illustration of a control loop. The sensor data from the physical process goes into the controller as input. The controller processes the data and issues instructions to the actuators of the physical process. 9](#_Toc157865838)

[Figure 3.7. Four Industrial Revolutions (Vaidya, Ambad et al. 2018) 10](#_Toc157865839)

[Figure 3.8 5C Architecture (Lee, Bagheri et al. 2015) 12](#_Toc157865840)

[Figure 3.9. Automation Pyramid (Martinez, Ponce et al. 2021) 14](#_Toc157865841)

[Figure 3.10 Overview of Grafana in the project. Grafana is used for monitoring the processes, sensors and actuators of the plant and it is getting data from the online data table of the database. 18](#_Toc157865842)

[Figure 3.11. Conventional virtualisation 19](#_Toc157865843)

[Figure 3.12 Docker Virtualisation 20](#_Toc157865844)

[Figure 4.1 This picture shows a cross-section view of the RSMS. It consists of a reaction chamber (1), magnet (2), peristaltic pump (3), electric motor (4), valve blocks (5), HMI connected to the controller (6), reset button and emergency button (7) and the in-line analysis island. Figure modified after (Abt, 2020). 22](#_Toc157865845)

[Figure 4.2 P&ID of the RSMS. Adapted from ("Single-Stage Protein Purification," 2022) [1] 23](#_Toc157865846)

[Figure 4.3 Representation of the RSMS in-line analysis panel: a UV sensor (3), a conductivity sensor (4) and a pH sensor (5) are mounted on the mounting plate (1) in a flow-through fitting (6) and a multichannel transmitter (8). The flow is along the direction of the arrow and can be regulated before the sensor by a 2-way ball valve (2) and after the sensor by a needle valve (7) 24](#_Toc157865847)

[Figure 4.4 Existing digital infrastructure after updating with Python API. All the sensors are connected to the McT over HART. The McT as well as the actors are connected to the PLC over PROFIBUS. The PLC connects over a switch to the IIOT Gateway and Server 1 with Ethernet. The IIOT gateway is also connected to Server 1 over Ethernet. 27](#_Toc157865848)

[Figure 4.5 Software running on the servers and IIoT. As rev pi is accessed remotely from server1 python API container was showed in server1 for better understanding. Sensor data was collected from PLC using OPC UA protocol and send this to database using TCP/IP. Matlab models pulls the data for simulation from timescale DB. Grafana is used for monitoring which also collects from Timescale. 28](#_Toc157865849)

[Figure 4.6 Schematic representation of the different functional areas creating cyberspace and their connection. The process control interacts with the physical system and forwards data to the data processing. The data processing writes the data into a database. The data in the database can be accessed by data visualization and the models. The models are connected to the process control. 30](#_Toc157865850)

[Figure 4.7 Weighting of technology selection criteria for a blockchain implementation by a head-to-head comparison. 32](#_Toc157865851)

[Figure 4.8 Radar chart for technology selection of between Node-Red and Python w.r.to Digital Twin 36](#_Toc157865852)

[Figure 5.1 Processing MS, SS, and pre/post process data. First, a trigger block checks if a "Recipe-Reset" signal was sent (1). If that is the case, the data gets processed analogue to the online data until the creation of the input query (2). A separate function block is used for each data type to split up the message containing all the combined data. Additionally, for the process data, it is checked if its pre or post process data (3) as each data set needs to be written in a different DB but are entered at the same point in the HMI. Before the data is written into the database, a filter block is used. This prevents identical data from being written in DB again (4). 41](#_Toc157865853)

[Figure 5.2 Flow chart of the API. The API first initiates the connection with OPC sever and Postgres DB. Once establish it further divided into 2 threads, checks the conditions, and updates the corresponding tables already created in our database. 42](#_Toc157865854)

[Figure 5.3 OPC UA Client-Server set up and functioning in the project. The OPC clients are docker container microservices running in rev pi. And OPC server is activated in PLC. Both are connected to the switch via ethernet to avoid multiple network subnet. 43](#_Toc157865855)

[Figure 5.4 Connection with the OPC server using the URL created by DHCP server. OPC UA inbuilt client method was used to establish connection with server. 44](#_Toc157865856)

[Figure 5.5 Some node ids created in the PLC which are very crucial for collecting sensor data. 45](#_Toc157865857)

[Figure 5.6 TCP/IP connection between API and database to write the data in the Timescale DB 45](#_Toc157865858)

[Figure 5.7 Recipe\_reset button on HMI based on the status of these buttons API collects the data and write on the tables accordingly. 47](#_Toc157865859)

[Figure 6.1 Docker working process. Once after going to application repository where the docker file exists build process can be initiated. Build process uses the Linux kernel and builds the image packaging all the dependencies and pushed into Docker hub registry. Pull process can be done from any machine irrespective of its environment. 48](#_Toc157865860)

[Figure 6.2 Docker overlay network to establish connection between two hosts. 53](#_Toc157865861)

[Figure 6.3 Outline of GUI for digital transparency. 56](#_Toc157865862)

[Figure 7.1 Here are the results of two breakthrough experiments displayed in sequence. A simulation was created for each breakthrough experiment. In experiment one, an inlet concentration of 14.14 g L-1 and a flow rate of 1906 ml min-1 was used. was used. In experiment five, an inlet concentration of 4.92 g L-1 and a flow rate of 1514.2 ml min-1 57](#_Toc157865863)

[Figure 7.2 Here are the results of three breakthrough experiments displayed in sequence. For each breakthrough experiment, a simulation was created, and the concentration of MNP was determined using the soft sensor. Experiment two was performed with an inlet concentration of 12.54 g L-1 and a flow rate of 1372.5 ml min-1, Experiment three used an inlet concentration of 27.4 g L-1 and a flow rate of 1496.4 ml min-1 and experiment four used an inlet concentration of 18.62 g L-1 and a flow rate of 676.2 ml min-1. 58](#_Toc157865864)

[Figure 7.3 Read latencies when Python and Node-Red both are running. 60](#_Toc157865865)

[Figure 7.4 Write latencies when both Python and Node-Red 61](#_Toc157865866)

List of Tables

[Table 3.1 Comparison between Virtual Machines and Docker containers 21](#_Toc157866016)

[Table 4.1 Rev Pi+ specifications 26](#_Toc157866017)

[Table 4.2 Specifications of 2 Servers 29](#_Toc157866018)

[Table 4.3 All the software used in different levels at Cyberspace. 30](#_Toc157866019)

[Table 4.4 Points criterion 32](#_Toc157866020)

[Table 4.5 Scoring logic for the evaluation of criteria I to X 33](#_Toc157866021)

[Table 4.6 Table of weighting the criterion based on our project needs 35](#_Toc157866022)

[Table 6.1 Docker container under custom docker network: Overlay network. 54](#_Toc157866023)

[Table 6.2 Docker Containers under custom Docker network: RSMS network 54](#_Toc157866024)

[Table 7.1 Process information about the breakthrough curves. 57](#_Toc157866025)

[Table 7.2 Read vs Write Latencies 59](#_Toc157866026)

[Table 7.3 Latencies of Python API when tested in 2 different scenarios. 60](#_Toc157866027)

Nomenclature Abbreviation

|  |  |
| --- | --- |
| Abbreviation | Explanation |
| RSMS | Rotary Stator Magnetic Seperator |
| HPLC | High pressure liquid chromatography |
| MNP | Magnetic nanoparticle |
| STT | Bioseparation Engineering Group |
| TUM | Technical University of Munich |
| PAT | Process Analytical Technology |
| DT | Digital Twin |
| HGMS | High Gradient Magnetic Seperation |
| CPS | Cyber Physical System |
| PLC | Programmable Logic Controller |
| API | Application Programming Interface |
| GUI | Graphical User Interface |
| OPC UA | Open Platform Communication Unified Architecture |
| TCP/IP | Transmission Control Protocol/Internet Protocol |
| CAGR | Compound Annual Growth Rate |
| ICS | Industrial Control Systems |
| DCS | Distributed Control Systems |
| SCADA | Supervisory Control And Data Acquisition |
| KPI | Key Performance Index |
| ERP | Enterprise Resource Planning |
| XML | Extensible Markup Language |
| HTTP | HyperText Transfer Protocol |
| SQL | Structured Query Language |
| TIA Portal | Totally Integrated Automation Portal |
| HMI | Human Machine Interface |
| MQTT | Message Queuing Telemetry Transport |
| LIMS | Laboratory Information Management System |
| McT | Memosense Technology |

# Introduction

The purification of proteins from fermentation broths in the bio pharmaceutical and food industries is a crucial process that involves multiple steps, leading to significant losses of valuable components. To address this challenge, the field of bio separation applies engineering and scientific principles to refine biological products on a large scale [2]. This project, aim to optimize the bio separation process by using advanced techniques such as Process Analytical Technology (PAT) and Digital Twin (DT).

One widely used method for protein separation is chromatography, which offers high levels of purity but has limitations in terms of scalability due to multiple batch processing steps. Research initiatives have focused on improving chromatography through approaches such as single-use applications and continuous processes. However, a suitable solution that overcomes these limitations is yet to be found, and downstream processing continues to account for a significant portion of the total production costs of biopharmaceuticals. (Tran, Lacki et al. 2014)

Addressing this challenge, research project explores the potential of High Gradient Magnetic Separation (HGMS) as a novel method for downstream processing. HGMS has the capability to selectively isolate proteins from challenging solutions in a single step, utilizing magnetic fields. This approach offers advantages such as semi-continuous operation, improved time efficiency, and the ability to handle higher volume flows. However, its application in biopharmaceutical processes is relatively new and not yet established in the pharmaceutical industry, primarily due to compliance requirements (Ebeler, Pilgram et al. 2018).

To facilitate the integration of HGMS into the biopharmaceutical industry, our research project leverages state-of-the-art automation and digitalization capabilities within the framework of Industry 4.0. Innovative concepts such as Cyber-Physical Systems (CPS), and Digital Twins have emerged in the ever-evolving industrial technologies. These cutting-edge technologies offer the potential to transform manufacturing processes, providing valuable insights and optimization opportunities through the integration of Machine Learning(Perno, Hvam et al. 2023). The implementation of DTs can significantly impact process efficiency and decision-making within the manufacturing industry.

Furthermore, the adoption of soft sensor technology enhances real-time monitoring and process control, crucial for the advanced bio-separation processes. Soft sensors, using data from robust and precise hardware sensors combined with algorithms, contribute significantly to the quality-by-design approach by providing real-time data that informs process adjustments and ensures consistent product quality.

Traditionally, ensuring product quality in pharmaceutical industries involves post-production quality testing, where batches are tested to meet predefined critical quality specifications. If these specifications are not met, the entire batch will be disposed. However, manufacturers are increasingly adopting a quality-by-design approach, combining a higher level of process understanding with real-time monitoring and control to enhance efficiency, reduce waste, and improve product quality [8]. For the successful implementation of this approach, communication-capable machines and systems are essential, enabling a virtual representation and real-time data exchange. Cyber-Physical Systems serve as the technological basis for IoT projects and enable the integration of physical objects, such as sensors and actuators, with information processing entities, such as PLCs, to control and manage the physical components logically (Olbort, Röhm et al. 2022).

In this project, the focus was mainly on enhancing the Bio Separation process by using the potential of Digital Twin and Industry 4.0 concepts. This work involves the development of a REST API, enabling seamless and sophisticated data collection, processing, and storage. The existing Node-RED system is replaced by this API to reduce latency and improve data transparency. Furthermore, Docker swarm cluster was established to facilitate communication and deployment of the API, allowing efficient data management and processing. Additionally, worked on the foundational aspects of a graphical user interface (GUI) to enhance the transparency of the Digital Twin concept.

Given these challenges, this study seeks to answer the following research questions:

a. How can the integration of IoT and IIoT technologies be optimized to develop a secure and efficient data processing device capable of handling large-scale industrial data in real time?

b. How can a robust and accurate soft sensor can be integrated to measure magnetic nanoparticle concentration in real-time, using mathematical modeling and data fusion methods?

# Motivation and Objectives

The optimization of biotechnology processes poses significant challenges due to their complexity and the limitations in making direct measurements of biochemical events. To address these challenges, the advancement of digitalization and Industry 4.0 offers new methods to improve and accelerate process optimization. By combining mathematical models, digital infrastructure, and physical processes in real-time, opportunities arise to enhance the efficiency and effectiveness of bio separation processes.

According to(Narayanan, Luna et al. 2020), mathematical models can serve as valuable tools for planning processes, improving the understanding of their behaviour, and reducing the number of necessary experiments. This approach enables the identification of non-trivial process conditions and the achievement of optimal process parameters. However, the next step in process optimization lies in integrating digital models with the physical system, leading to the creation of a "digital twin" (DT). As mentioned by (Udugama, Lopez et al. 2021), the DT can dynamically optimize processes during operation, enabling real-time adjustments and predictions.

The main motivation is to establish an intelligent and adaptive digital twin to optimize bio separation processes. The project focuses on developing a robust digital infrastructure for bio separation processes. The important component of the infrastructure is software tool or API which is developed using python. This API establishes the direction connection with the control level unit, which is PLC in this project, which allows real-time data collection using OPC UA protocol which provides bidirectional communication by enhancing client-server mechanism. Once the data is collected it was processed and stored in the database using the TCP/IP. The stored data has been used by the models for simulations to identify optimized parameters.

Docker also plays an important role by allowing microservices concept into our project. Docker was used extensively in packaging the software and deploying it on any environment. Even on resources limited ARM architecture Rev pi thanks to docker multi architectural build feature. Seamless communication with minimal latency is crucial for DT. Docker and microservices again plays a crucial role in achieving this with the help of docker overlay network which allows in building the connection between services running on multiple hosts. To enhance transparency during the simulation process, a foundational GUI layout was developed using the PyQt5 library. This interface displays plots, offering insights into the processes.

## Research Trends

With the value of USD 1597 billion in 2020 the Pharmaceutical industry was one among the least affected during the COVID 19 disruption. A world-wide production value growth rate of 2.8% and profitability of 26% based on the 18 largest global economies. However, 90% of global pharmaceuticals production is concentrated within top 20 countries in 2020 with China and USA in first and second places respectively. Because of high intense competition, the pharma industry is facing the productivity and cost pressures to address the market requirement. Therefore, process development is one of the most important aspects to improve the productivity and efficiency (Chuck-Hernández, Chew et al. 2022) .

The global process analytical technology market is projected to experience a compound annual growth rate (CAGR) of 17.1 % between 2021 and 2030, to reach 13,626.5 USD million by 2030 from an estimated 3,283.8 USD million in 2021 (Stevens, Loudon et al. 2012)[12]. The global digital twin market size was valued at USD 7.48 billion in 2021 and is projected to grow at a CAGR of 39.1 percentage from 2022 to 2030 [13]. The global cyber-physical systems (CPS) market size is predicted to increase from USD 76.98 billion in 2021 to USD 177.57 billion by 2030 at a CAGR of nearly 8.01% during 2022-2030. (Broo, Boman et al. 2021)

# Theoretical Background

The following chapter provides an overview of the theoretical concepts and processes underlying the Rotary Stator Magnetic Seperator (RSMS) for protein purification. The RSMS system employs a series of steps to bind, separate, wash, and elute the target protein from magnetic particles, enabling efficient and high-quality protein purification.

## Process Knowledge

The principle of Magnetic separation process of Andritz GmbH is explained in the following 4 steps.

1. **Binding of Target Proteins**

The first step in the RSMS process is the binding of the target protein to magnetic particles. This step involves the reversible binding of the molecule of interest to the magnetic particles.

It is typically performed in an external steering vessel, allowing for the handling of large batch volumes. In fig. 3.1 binding process is shown and the number of magnetic particles used is adjusted based on the target molecule concentration and the desired yield. Furthermore, comprehensive data on the binding behaviour between the target molecule and the magnetic particles is essential for optimal binding efficiency.

Diagram of a machine with a few spheres

Description automatically generated

Figure 3.1 Binding of Target Protein [1]

1. **Separation of Magnetic particles**

Following the binding step, the magnetic particles need to be separated from the solution.

The separation process in RSMS involves several key steps. Firstly, the magnetic slurry, consisting of the particle-feedstock suspension, is introduced into the process chamber by connecting the stirring vessel. The suspension is then pumped through a magnetic separator while an external magnetic field is applied by an electromagnet. This causes the magnetic particles to be separated from the feedstock by the magnetized separation matrix. The particle-depleted fermentation brew is directed to the waste container for disposal.

Diagram of a magnetic field diagram

Description automatically generated

Figure 3.2 Separation of Magnetic particles [1]

1. **Washing of Particles**

After the separation step, the magnetic particles undergo a washing process to remove any remaining impurities. The washing of particles involves the displacement of the remaining feedstock in the separation chamber with a wash solution. Prior to replacing the wash solution, the magnetic field is applied to recapture the magnetic particles by slowing down the rotor and allowing them to recollect on the separation matrix. It is also recommended to have a bypass flow through the separation chamber to recollect any weakly bound or sedimented magnetic particles.[1]

Diagram of a diagram of a washing process

Description automatically generated

Figure 3.3 Washing of particles [1]

1. **Elution of Target Proteins**

Following the washing steps, the purified target protein is eluted from the magnetic particles.

A diagram of a cycle

Description automatically generatedElution is typically achieved by changing the liquid phase conditions, such as lowering the pH or increasing the salt concentration. The elution fluid is fed into the system while the magnet is turned off, and the magnetic particles are resuspended in the elution fluid for a specified duration. The eluted protein is collected in fractions, and the procedure is repeated multiple times to ensure complete elution. After elution, any strongly bound contaminants or leftover product are removed from the magnetic particles to prepare them for reuse or storage.

Figure 3.4 Elution of target particles [1]

## Process Analytical Technology

Since the RSMS has multiple applications in the pharmaceutical industry, its use is also subject to GMP-related regulatory requirements. PAT, initiated by the US FDA, is instrumental in quality assurance, focusing on the design, analysis, and control of manufacturing processes through the measurement of Critical Process Parameters (CPPs) that influence Critical Quality Attributes (CQAs) (Kim et al. 2021). Monitoring methods are categorized into on-line, in-line, at-line, and off-line approaches, with in-line controls being integral for real-time process information. These controls, directly implemented within the process, reduces errors induced by environmental factors or sampling procedures. However, certain analytical methods, like high-pressure liquid chromatography, needs offline execution due to structural or sensor technology limitations.

A diagram of a process

Description automatically generated

Figure 3.5 Schematic representation of measurement approaches in process analytics. A distinction is made between on-, in-, at- and off-line (Olivier Henry 2019).

Process control based on CPPs often employs a Programmable Logic Controller (PLC) with a CPU that stores and processes algorithms. The PLC interprets data from sensors and relays commands to actuators, vital for maintaining process consistency. This structure ensures direct feedback through on-line and in-line data, a prerequisite for accurate CPP measurement (Hernández Rodríguez and Frahm 2021).

A screenshot of a computer

Description automatically generated

Figure 3.6 Illustration of a control loop. The sensor data from the physical process goes into the controller as input. The controller processes the data and issues instructions to the actuators of the physical process.

Incorporating advanced data handling methods, such as a Python-based API or Node Red facilitates efficient data collection from the PLC, thus enhancing process control. This data, stored in a database, becomes a foundational element for simulation models, further informing and refining control strategies. The feedback from these simulations is transmitted back to the PLC, thereby optimizing actuator responses, and ensuring a streamlined process flow.

### Integration of Soft Sensors in Process Monitoring

Soft sensors are used to combine the online and inline hardware sensors and a software-based calculation algorithm for online monitoring (Kadlec et al. 2009). They are used to determine quantities that cannot be measured directly in-line or on-line and are therefore very important for improved monitoring of critical parameters and process control. The data from the hardware sensors, e.g. for temperature, conductivity, or UV, are sent to software that uses an algorithm to calculate the desired quantity.

Soft sensors can be divided into two types: model-driven and data-driven. Model-driven methods require a deep understanding of the process to develop models that can realistically represent the physical operations in the process. The data-driven soft sensors, on the other hand, are based on historical measurement data, the process does not need to be understood fundamentally. Soft sensors have many applications in the process industry, especially in areas such as the chemical, paper, and steel industries. A practical example of the use of soft sensors can be found in the petrochemical industry. In refineries, for example, soft sensors are used to monitor and control distillation processes. To determine the composition of the product streams, soft sensors can estimate the product composition in real time from easily measurable process parameters such as temperatures, pressures and flow rates using process models. (Kadlec et al. 2009)

Hence, the most important part in the context of work is to collect the real time data from the hardware sensors and store them in the database and software algorithms use the historic data to calculate the parameters which are very difficult to measure. Determination of density which is a crucial parameter in finding out the MNP concentration particularly in ionic solutions using soft sensors are discussed in more detail in coming sections.

## Industry 4.0

The evolution of industrial revolutions has transformed the manufacturing landscape, propelling us into the era of Industry 4.0. Each revolution has brought significant advancements, from the advent of steam power and mechanization in the late 18th century (first industrial revolution) to the discovery of electricity and assembly line mass production in the 19th century (second industrial revolution). The third industrial revolution, which emerged in the 1970s, introduced computer technology and automation, enabling entire production processes to operate without human assistance (Vaidya, Ambad et al. 2018).

A close-up of several labels

Description automatically generated

Figure 3.7. Four Industrial Revolutions (Vaidya, Ambad et al. 2018)

Today, we find ourselves at the fourth industrial revolution, Industry 4.0. This revolution is characterized by the integration of information and communication technologies into industrial practices, fundamentally redefining the manufacturing landscape.

The Industry 4.0 paradigm promotes the connection of physical items such as sensors, devices, and enterprise assets both to each other and to the Internet. Building upon the developments of the previous industrial revolution, Industry 4.0 expands the capabilities of production systems through network connectivity and the concept of the digital twin (Vaidya, Ambad et al. 2018). In Industry 4.0, production systems are interconnected, creating cyber-physical production systems that form the foundation of smart factories. These systems, components, and people communicate seamlessly over a network, facilitating near-autonomous production processes. The digital twin, a virtual representation of physical assets and processes, plays a pivotal role in Industry 4.0 by enabling real-time monitoring, analysis, and optimization of production operations.(Sipsas, Alexopoulos et al. 2016)

### Cyber–Physical Systems

Cyber-physical systems (CPS) are smart systems that integrate computational and physical components, enabling real-time perception and response to the physical world. In the context of Industry 4.0, CPS plays a crucial role in securing industrial systems and manufacturing lines against cyber threats. Reliable communication and robust identity and access management are essential in CPS environments (Lee, Bagheri et al. 2015).

CPS provides the integration of the physical, service, and digital realms, leading to improved information quality for planning, optimization, and operation of manufacturing systems. Decentralization and autonomous behaviour are key characteristics of CPS, allowing for dynamic production processes. CPS evolution relies on the adoption and reconfiguration of product structures, enabling collaborative cyber-physical systems in manufacturing and other domains.(Vaidya, Ambad et al. 2018)

The 5Cs architecture represents five levels in CPS.(Lee, Bagheri et al. 2015)

Connection between the real world and the cyberspace. Conversion of the data to meaningful information. Cyber, by which he means a central information hub that uses analytics to generate additional information. Cognition, which generates knowledge about the monitored system. Configuration, feedback from the cyber to the physical space, supervisory control to make machines self-configure.

A diagram of a pyramid

Description automatically generated

Figure 3.8 5C Architecture (Lee, Bagheri et al. 2015)

Cloud systems play a significant role in facilitating real-time data exchange and intelligent linkage between CPS (Stock and Seliger 2016)[17]. The concept of a "Digital Shadow of Production" represents the virtual representation of physical objects in the information world, enabling real-time manufacturing operations and optimization. Sensor-equipped CPS detect machine failures, enabling automatic fault repair actions and optimizing workstation utilization based on operation cycle time.

### Digital Twin

With the advancement of data acquisition, Information Technology (IT), and networking technologies sophisticated technologies have emerged in Digitalization of manufacturing like Digital Twin. The aim of the digital twin is to develop the intelligent model based on the physical processes that simulate the product behaviour in digital space. The model data is then trained with historical and real-time data of the physical processes to make predictions that can optimise the processes and behaviour of the physical object (Tao, Qi et al. 2019).

According to (Udugama, Lopez et al. 2021) the digital twin must consist of 3 parts.

The physical object, a machine or a plant must be equipped with sensors and actuators. The sensors are needed so that the DT can collect data and the actuators are there so that the DT can influence the process in the plant/machine. A digital infrastructure that connects the cyberspace with the physical object for bidirectional communication. Digital model of the physical object in the cyberspace

As I mentioned above 5C architecture in chapter 3.3.1 the first 3C’s belonged to the CPS and the last 2Cs belonged to the DT. The models and data are assigned to the DT, while the sensors and actuators belong to the CPS (Tao, Qi et al. 2019) According to (Lu, Liu et al. 2020) digital twin is the prerequisite for the development of the CPS as it represents the physical object in digital world.

### Automation

The term automation is a key phenomenon in every industry, aiming to automate repetitive and tiring work to improve system efficiency. Industrial automation involves the automation of industrial processes and machinery using sensors, controllers, and actuators. To comprehend the integration process between hardware and software in industrial automation, the concept of the automation pyramid provides valuable insights. The automation pyramid elucidates the various levels of industrial automation (Martinez, Ponce et al. 2021).

* Level 0, known as the process level, involves the use of sensors and actuators to collect machine data. At this level, parallel wiring is commonly employed, and the field level acts as the interface between the production process and the control level, utilizing field buses and Input/Output (I/O) cards.
* Level 1, referred to as the control level, encompasses the Information Technology (IT) structure responsible for controlling the production process. This level is typically implemented through the utilization of Industrial Control Systems (ICS) like Programmable Logic Controllers (PLC) or Distributed Control Systems (DCS).
* Level 2, known as the process monitoring level, incorporates graphical interfaces, such as SCADA systems. These interfaces serve operational purposes, process monitoring, and recipe management.
* Level 3, termed the operational management level, encompasses systems like Manufacturing Execution Systems (MES), Management Information Systems (MIS), or Laboratory Information Management Systems (LIMS). These systems facilitate detailed production planning, quality management, data collection, material management, and Key Performance Indicator (KPI) documentation.
* Level 4, referred to as the company management level, involves overarching production planning and order management through Enterprise Resource Planning (ERP) systems. This level provides top-level control and visibility, allowing company management to oversee and manage overall production processes.

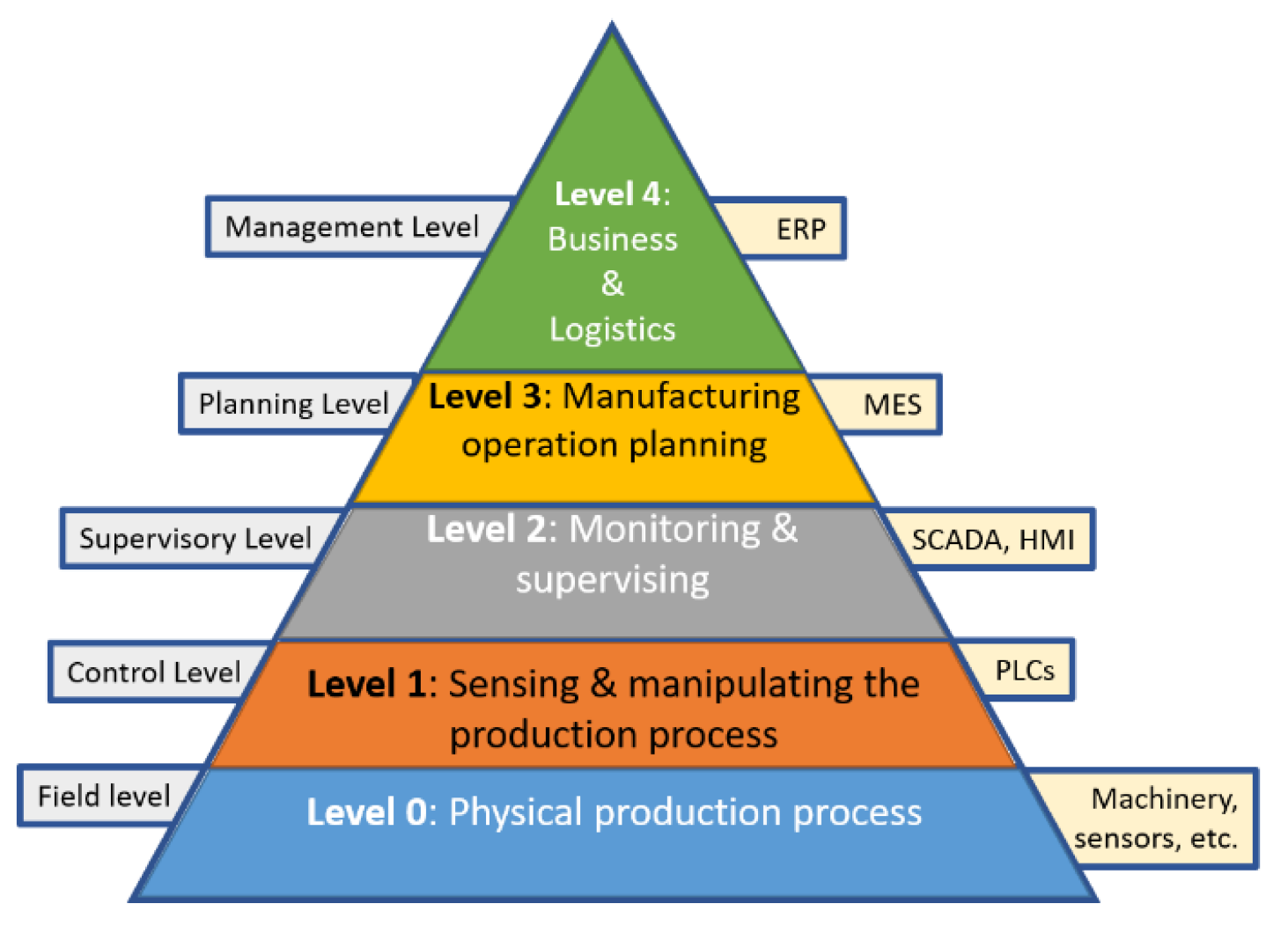


Figure 3.9. Automation Pyramid (Martinez, Ponce et al. 2021)

## Communication Protocols

A digital twin is a complex construct which involves different types of communication protocols based on the situation. As it is relatively new technology there is no blueprint or protocol to follow yet. Building communication between the blocks is one of the complex tasks in DT.

### TCP/IP

TCP/IP is an acronym derived from the two different protocols. The Transport layer, led by TCP, establishes, and maintains connections between devices for data transmission. [32] Data is divided into numbered segments to facilitate reassembly at the receiver's end. Before transmission, the sender and receiver stacks establish a connection and agree on the amount of data to be transmitted. Following this, the receiver sends an acknowledgment, and the data transfer begins. After transmission, the sender awaits an acknowledgment from the receiver, concluding the process. If data packet delivery encounters disruptions, the protocol facilitates retransmission attempts.

The second part is the Internet Protocol (IP), which determines the destinations of data packets. It maintains an overview of interconnected networks and identifies devices within the network using IP addresses and hardware addresses. IP addresses signify the receiver's network, while hardware addresses enable identification within that network. (Lammle and Buhagiar 2020)

### Modbus

Modbus, an enduring and widely utilized communication protocol in the automation industry, is recognized for its speed, reliability, and user-friendliness. Employing a master-slave architecture, the protocol operates through queries sent by the master and responded to by the slave. These queries have a fixed structure comprising four essential elements, consistently arranged. Initially, the device address of the Modbus slave is specified, followed by the function code that determines the data type, registry, and read/write operation. The third element is the actual data being read/written, and the final component is an error check to ensure accuracy (Thomas 2008).

An evolution of the basic protocol is the Modbus TCP/IP, which utilizes the TCP/IP protocol for transmitting queries to a slave server. This enables data reading/writing between masters and slaves over Ethernet or the Internet (Swales 1999)

### OPC UA

OPC Unified Automation was the initiation taken by the OPC foundation to develop the robust communication protocols for data exchange in Industrial Automation context. This standard provides the consistent read-and-write access to the real time data which is very crucial for the projects that involves in creating Digital twin and complex data management systems. OPC UA is designed for wide range of products from intelligent gadgets to complex SCADA systems.   
The core components of the OPC UA are transport mechanism and data modelling. The transport mechanism uses the modified version of TCP protocol which allows the intranet communication at transport layer of an octane. It also supports the XML and HTTP communication methods like GET, WRITE and others ensuring flexibility in various operational environments(Mahnke, Leitner et al. 2009). The second one is data modelling which designs the formulation of data representation within the OPC UA. It establishes the rules for interpretation of information models. It also forms the hierarchy for the data using the address space and base type. The beauty of the OPC UA’s design is its ability to allow us to customise information models to improve the functional structure based on our needs(Roulet-Dubonnet 2018).

UA services: Bridging the server and client components in the architecture are the UA services. The server acts as reservoir and manager of the information models and clients are like the customers of the information models. UA services using the robust transport mechanisms and enables us seamless data exchange. OPC UA base is the combination of transport mechanism, data modelling and UA services together and its modular character enables customisation based on project requirements(Mahnke, Leitner et al. 2009).

## Pscopg2

The psycopg2 library served as a bridge between the API and the PostgreSQL database. PostgreSQL is an advanced open-source database system widely used in various applications. psycopg2 is a Python library specifically designed to facilitate connections and interactions with PostgreSQL databases. It provided a convenient and efficient way to establish a connection with the PostgreSQL database server, execute SQL queries, and handle data insertion, retrieval, and manipulation. With psycopg2, we could seamlessly integrate the API with the PostgreSQL database, enabling efficient storage and retrieval of collected data.

With the capabilities of Python, OPC UA, and psycopg2 libraries, we were able to develop an API that effectively communicated with the PLC, processed data, and seamlessly interact with the PostgreSQL database. These libraries provided the necessary tools and functionality to implement the desired features and meet the requirements of the project.(Fischer, Hirn et al. 2022)

## PostgreSQL

PostgreSQL a renowned open-source relational database system was used as Database in the project. Postgres was widely used and have many advantages based on our DT requirements also. Some of the advantages are it fast and it is easy and straightforward to build the connection with the applications. There are several standard APIs for building connection with the Postgres database. Psycopg2 library was used to establish connection between Postgres and Python in this work. One of main disadvantage was it is not the time series database, it won’t provide timestamp for every entry in the database and that is most important criteria in the DT (Obe and Hsu 2017).

One notable tool is pgAdmin, which offers a user-friendly interface for PostgreSQL, making database interactions much more straightforward. In our project, we consistently used pgAdmin alongside both PostgreSQL and Timescale DB for creating tables and DB servers.

## Timescale DB

To overcome the issue with the timestamps in Postgres, Timescale DB was introduced which is also an open-source and comes with the time-series database. Also, timescale DB is claimed to be significantly faster than PostgreSQL in reading and writing the values (Kammakomati, 2021). We used official docker images of both Postgres and timescale DB in this project to implement microservices approach over monolithic.

## Grafana

For effective real-time visualization of data acquired from databases, we utilized Grafana, a popular open-source platform. This tool had been adopted in a previous project and was chosen again for its ease in connecting with various databases. Grafana not only supports a wide range of database integrations but also simplifies data retrieval using standard SQL queries. As shown in fig.3.10. the design of certain graphs was refined. In the figure below also shown the at which of our digital infrastructure we utilised the Grafana. Key features of Grafana include its open-source nature, user-friendly visualization options, and built-in database connection plugins.

A screenshot of a computer

Description automatically generated

Figure 3.10 Overview of Grafana in the project. Grafana is used for monitoring the processes, sensors and actuators of the plant and it is getting data from the online data table of the database.

## Virtualisation

Running multiple applications simultaneously in the same machine can be challenging as they will be having different external dependencies and runtime requirements. Virtualisation at its base, is the technology that allows the creation of an abstract version of computer resources, providing multiple applications and operating systems to be run on the same physical machine.(Potdar, Narayan et al. 2020) The 2 primary techniques in the virtualisation have become popular for obvious reasons.

### Conventional Virtual Machines (VMs)

**Architecture**: As shown in fig.3.11. below the VMs run on a hypervisor or a virtual machine monitor. This hypervisor operates directly on the host machine's hardware and oversees the VMs. Each VM contains a full instance of an operating system, a virtual replica of the hardware that the OS needs to run, and the application itself.

**Isolation**: VMs offer robust isolation, as each VM runs in its separate environment. This means the malfunction or cessation of one VM doesn't impact others.

**Overhead and Performance**: Due to the presence of separate OS instances and the associated virtualized hardware, VMs can be resource intensive. This often results in higher overhead, increased boot times, and consumption of more storage. (Morabito, Kjällman et al. 2015)

A screenshot of a computer

Description automatically generated

Figure 3.11. Conventional virtualisation

### Docker Virtualisation

Unlike conventional virtualisation docker virtualisation happens at the application level. The rise of microservices and cloud computing has been the main reason for the evolution of containers in software industry.

**Architecture**: As shown in the fig.3.12 unlike VMs, containers share the host system’s OS kernel, rather than creating their virtualized OS. They encapsulate the application and its dependencies into a "container". This is managed by a container runtime on the host OS.

**Isolation**: Docker containers also provide isolation, but it's lightweight in comparison to VMs. Multiple containers can share the same OS kernel but operate in isolated user spaces.

**Overhead and Performance**: Containers are more lightweight than traditional VMs since they don’t need the extra load of entire virtualized OS and hardware. This results in faster start times and better resource utilization. Additionally, containers are modular, allowing for microservices architectures where each service runs in its separate container. (Morabito, Kjällman et al. 2015)

A screenshot of a computer

Description automatically generated

Figure 3.12 Docker Virtualisation

Table 3.1 Comparison between Virtual Machines and Docker containers

|  |  |  |
| --- | --- | --- |
|  | **Virtual Machines** | **Docker Containers** |
| Isolation Process level | Hardware | Operating System |
| Operating System | Separated | Shared |
| Boot up time | Long | Short |
| Resource Usage | More | Less |
| Pre-built Images | Hard to find and Manage | Already available for home server |
| Customised Pre-Configured Images | Hard to build | Easy to build |
| Size | Bigger because they contain whole OS underneath | Smaller with only docker engines over the host OS |
| Mobility | Easy to move to a new host OS | Destroyed and recreated instead of moving. |
| Creation time | Longer | Within Seconds |

# Materials and Methods

This chapter describes the components of the DT which includes the Control unit, communication protocols and RSMS plant. The digital infrastructure is under the TUM network.

## RSMS

The RSMS device (MES 25/100 RS, Andritz KMPT GmbH, Germany) is a key part of our project. It's where experiments happen, and it helps make the digital twin of our bio-separation process work. RSMS device acts as a bridge between the real-world process of separating biological materials and the computer models we use to simulate and improve those processes. In this section, how the RSMS works and how it connects with the computer systems that are a big part of our project was explained. Also, how information moves from the RSMS device to our digital models, showing how important this device is for linking the real and digital worlds will be discussed.

A diagram of a machine

Description automatically generated

Figure 4.1 This picture shows a cross-section view of the RSMS. It consists of a reaction chamber (1), magnet (2), peristaltic pump (3), electric motor (4), valve blocks (5), HMI connected to the controller (6), reset button and emergency button (7) and the in-line analysis island. Figure modified after (Abt, 2020).

The components of the of RSMS which are shown in the fig.4.1. the briefly explained clearly.

1. Reaction chamber: where the matrix consists of rotor and stator disks are located.
2. Electro-Magnet: AME 55 12 21 So., Steinert GmbH, Germany
3. Peristilatic pump: Verderflex R3S, Verder Germany GmbH & Co. KG, Germany
4. Electric motor
5. Valve Blocks: (Gebr. Müller Apparatebau GmbH & Co. KG, Germany
6. HMI connected to the controller: Siemens HMI TP1200 Comfort, Siemens Inc.
7. Reset button and emergency button.

Following the piping and instrument diagram of RSMS

A diagram of a machine

Description automatically generated

Figure 4.2 P&ID of the RSMS. Adapted from ("Single-Stage Protein Purification," 2022) [1]

As shown in fig 4.2 the deposition occurs in the process chamber (1), where the matrix consisting of rotor and stator disks is located. This chamber is surrounded by the electro-magnet (AME 55 12 21 So., Steinert GmbH, Germany) (2). Following the line that leaves the process chamber upwards, the first thing passed is the in-line analysis unit. The line ends in a valve block (Gebr. Müller Apparatebau GmbH & Co. KG, Germany) (5). The line that leaves the process chamber downwards leads to the flow sensor and then the peristaltic pump (Verderflex R3S, Verder Germany GmbH & Co. KG, Germany) (3) with the associated electric motor (4). The line also ends in a valve block (Gebr. Müller Apparatebau GmbH & Co.KG, Germany) (5). The valve blocks can be used for the supply and discharge of liquids and recirculation. The system can be operated with the aid of the HMI (Siemens HMI TP1200 Comfort, Siemens Inc., Germany) (6). The reset button and the emergency button (7) are located below the HMI [1]. In-line analysis island will be discussed in more detail in 4.1.1. The data collected from these sensors is very crucial in understanding the process as well as calculating parameters like MNP concentrations using soft sensors.

### Inline Process Analytic

During RSMS operation, six parameters can be measured in line: pH, temperature, conductivity, density, UV absorbance, and volume flow. Three sensors are installed on the in-line process analysis island in fig.4.3. The conductivity meter (Memosens CL82D, Endress+Hauser Inc., Switzerland) (4), with a measuring range from 1 µS cm- 1 to 500 mS cm- 1 and an integrated temperature sensor to correct the conductivity if the temperature deviates from 25 °C. A pH sensor (Tophit CPS471D, Endress+Hauser Inc., Switzerland) (5) with a measuring range of 0 to 14 and a Photometer (OUSAF44, Endress+Hauser AG, Switzerland) (3) that can measure between 0 to 3 absorbance units at a wavelength of 280 nm. After each run, the conductivity and pH sensors need to be dismounted. The pH sensor is stored in a 3 mol L - 1 potassium chloride solution, while the conductivity sensor is stored dry with a safety cap. With the two valves 2 and 7, the inline analysis unit can be taken out of operation for maintenance. The data from the photometer, pH- and conductivity sensor is collected, processed, and monitored in the multichannel transmitter (Liquiline CM44P, Endress+Hauser Inc., Switzerland) (8).

A diagram of a machine

Description automatically generated

Figure 4.3 Representation of the RSMS in-line analysis panel: a UV sensor (3), a conductivity sensor (4) and a pH sensor (5) are mounted on the mounting plate (1) in a flow-through fitting (6) and a multichannel transmitter (8). The flow is along the direction of the arrow and can be regulated before the sensor by a 2-way ball valve (2) and after the sensor by a needle valve (7)

The density and the flow velocity are determined with a Coriolis flowmeter (Promass F 100 8F1B08 DN08 3/8', Endress+Hauser AG, Switzerland), which is located between the in-line process analysis island and the upper valve block. It can measure flow rates between 0.1 kg h- 1 and 2000 kg h - 1 and has a measurement deviation for the volume flow of ± 0.1 % and ± 0.5 kg m-3 for the density. As in the conductivity sensor, an additional temperature sensor is built into the Coriolis sensor, since the density is temperature dependent.

## Programmable Logic Control

The Siemens S7-1500 (CPU 1516-3 PN/DP) is used to control the system to automate the tasks using Wincc and TIA portal.

Main features of the S7-1500 PLC

* CPU with 1 MB work memory for program and 5MB for data.
* Profinet and Profibus communication interfaces.
* Built-in OPC UA.

The PLC acts as the brain of our operation, interpreting data from the sensors to make real-time decisions. This intelligent processing enables us to automate complex tasks and the system adapts quickly to any changes in the process conditions. In this thesis, the data is collected from the sensors via PLC using the OPC UA which will be discussed in detail in chapter 5.

## Human Machine Interface

The Siemens TP1200 comfort HMI is installed inside the cabinet for monitoring and performing tasks at the operator control interface.

Features and specs of the HMI:

* Touch screen.
* 12” widescreen TFT display.
* 16 million colors.
* Profinet and Profibus communication interface.
* 12 MB configuration memory.

The HMI is where human operators interact with our system, providing a vital link between the complex processes controlled by the PLC and the users who monitor and manage them. Through the HMI, operators can see process data, adjust process parameters, and receive alerts, making it an essential component for both efficiency and safety.

## Raspberry Pi

Raspberry Pi is the ideal choice for developing IoT applications. With the Raspberry Pi we can install the Raspbian OS which is under the Debian family and at an incredibly low price. It can be accessed remotely via SSH and is very easy to set up. The system is equipped with two Ethernet interfaces, enabling integration into both automation and IT networks simultaneously. This allows machine data to be efficiently transmitted from the shop floor to the cloud or higher-level ERP systems using protocols like MQTT and OPC UA [38]. Additionally, users have the flexibility to program individual applications using various languages such as Node-RED, Python, or C, ensuring versatile functionality and adaptability. In this project, we are using Rev Pi as an IoT device to build communication between the Control Unit (PLC) and the digital infrastructure.

Rev Pi is the perfect fit for our project as it is flexible, economical, and small enough to build inside the cabinet. In this project, Rev Pi is used as a computer and a crucial part of the Python API container is running in Rev Pi which collects the data and writes in our Database which is used for simulations and visualization.

Table . Rev Pi+ specifications

|  |  |
| --- | --- |
| **Item** | **Rev pi connect +** |
| Processor | 1.2 GHz, 4 cores |
| RAM | 1 GB |
| Disk space | 16 GB |
| Operating System | Raspbian GNU/Linux 9 (stretch) |
| IP address | 10.162.80.187 |

## Digital Infrastructure

In this chapter, the key updates made to the digital infrastructure, considering the system's different levels are discussed. A significant update was the integration of Node-Red alongside Python for data acquisition, with each being chosen for distinct roles within our project. This decision was taken by the need to optimize data handling and increase the system's flexibility. Earlier sections have covered the specifications and roles of individual components. Here, we focus on the methods used to link these components, detailing the connections and protocols that join the digital infrastructure together. The process of selecting technologies, including the shift to Node-Red, is further detailed in Section 4.8 through a radar chart, considering the criteria important to our project's goals.

A screenshot of a computer screen

Description automatically generated

Figure 4.4 Existing digital infrastructure after updating with Python API. All the sensors are connected to the McT over HART. The McT as well as the actors are connected to the PLC over PROFIBUS. The PLC connects over a switch to the IIOT Gateway and Server 1 with Ethernet. The IIOT gateway is also connected to Server 1 over Ethernet.

As seen in Figure 4.4 all sensors of the in-line analysis island are connected to the McT via HART and the Coriolis Promass F 100 via Profibus. In the next step, the actuators and the McT are connected to the PLC on the control level via PROFIBUS. From this point on, all other connections on the control, communication, and virtual level are via Ethernet. The PLC is then connected to an IIOT gateway on the communication level. The IIOT gateway and the PLC are directly connected to the virtual level presented by Server 1 over a switch.

The overall digital infrastructure framework is divided into four levels as shown in the fig.4.5.

* Field level with field instruments.
* Control level with PLC and HMI.
* Communication level with Python API docker container running over Raspberry pi.
* Virtual level with Timescale DB, Grafana, and MATLAB container running on one server and another server containing the TIA portal.

Considering the simplicity and availability of various libraries, a decision was made to develop a GUI that shows the plots of Magnetic separation processes before and after simulation generated by MATLAB models. This visualization helps validate the predictions made by models and understand where adjustments might be necessary. To achieve this, the GUI was integrated into the digital infrastructure as shown in fig 4.5.

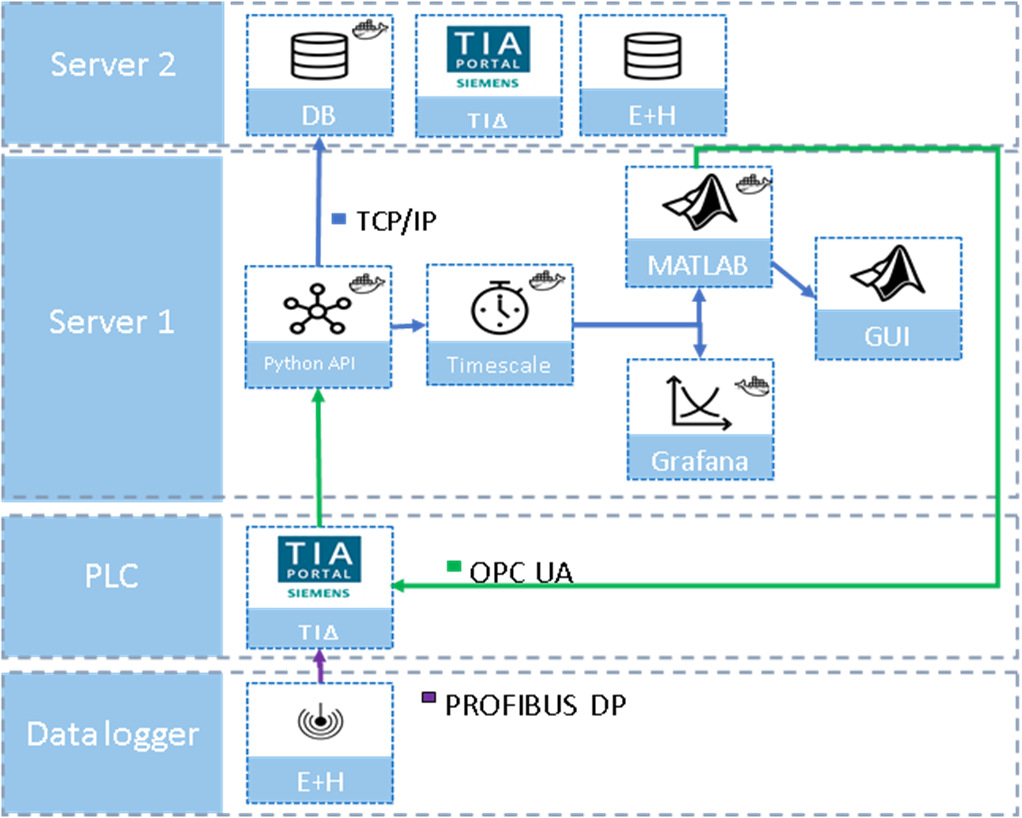


Figure 4.5 Software running on the servers and IIoT. As rev pi is accessed remotely from server1 python API container was showed in server1 for better understanding. Sensor data was collected from PLC using OPC UA protocol and sent this to the database using TCP/IP. Matlab models pull the data for simulation from timescale DB. Grafana is used for monitoring which also collects from Timescale.

## Computer-Setup

Two servers were used for handling modeling, and data storage. The first server uses Windows as an operating system; the second uses Ubuntu. The second server also has considerably better hardware and more computational power. The specifications are listed in Table.

Table . Specifications of 2 Servers

|  |  |  |
| --- | --- | --- |
| Item | Server 1 | Server 2 |
| Processor | Intel Xeon E3-1225 v5, 3.3 GHz | Intel Xeon W-2145 3.7 GHz |
| Cores | 8 | 16 |
| RAM | 32 GB | 64 GB |
| Disk space | 2 TB and 256 SSD | 2 TB HDD and 512 GB HDD |
| Operating System | Microsoft Windows 10 | Ubuntu 20.04.3 LTS |
| IP address | 10.162.80.5 | 10.162.80.250 |

## Cyberspace

In this chapter the different software used in the Cyberspace to create CPS is explained.

### Structure of Cyberspace

In our project's cyber-ecosystem, we've identified five interconnected stages as shown in the fig.4.6 that help structure and define the data's flow and processing. Starting at the ' Physical system’, the entire plant is equipped with sensors and actuators that play a fundamental role in gauging conditions and initiating actions. These devices relay their signals to the 'Control Level', managed by the Programmable Logic Controller (PLC), which controls the processes based on the incoming data. Subsequently, the 'Data Collection & Processing Level' takes over, with our Python API collecting, processing, and channelling this data for storage. This data finds its home at the 'Database Level', serving as the foundation for historical analysis and as the backbone for our advanced modelling at the 'Models & Visualization Level'. Here, simulations run, insights are generated, and real-time visualizations via Grafana offer an idea about the processes. Importantly, the architecture isn't linear our models relay simulation outcomes back to the PLC, creating a feedback loop that optimizes and refines the processes continually. This shown in the fig.4.6 below.

A diagram of a data processing process

Description automatically generated

Figure 4.6 Schematic representation of the different functional areas creating cyberspace and their connection. The process control interacts with the physical system and forwards data to the data processing. The data processing writes the data into a database. The data in the database can be accessed by data visualization and the models. The models are connected to the process control.

### Software Used

To achieve the data processing quickly different software were used at different layers. All the software were containerized and for easy deployment without affecting others performance. Below table lists all the software used and their functionalities.

Table 4.3 All the software used in different levels at Cyberspace.

|  |  |
| --- | --- |
| Name | Functional area |
| TIA Portal V16 | Process Control |
| Node-RED | Data Processing |
| Python | Data Processing |
| PostgreSQL | Database |
| Timescale DB | Database |
| Grafana | Data Visualisation and Monitoring |
| MATLAB, Python | Models |
|  |  |

## Methodology for Technology Selection

In this project, selecting the appropriate technology for data collection and database management is crucial in achieving our objectives. This decision not only influences the efficiency and effectiveness of our data processing but also impacts the scalability and reliability of our system. Given the complexities and specific requirements of our work, particularly in the context of OPC UA client-server model and real-time data processing, we have established a set of criteria to guide our technology selection. These criteria are considered to evaluate and compare the capabilities of Node-Red and Python.

The requirements formulated leads to these 10 criteria:

* Performance Efficiency
* Scalability
* Integration capability
* Development speed
* Real-Time processing
* Historical Data Management
* Community Support and Resources
* Maintenance and Updates
* Language Ecosystem
* Execution Model

### Weighting of Digital Twin Data processing specific Criteria

To weight the criteria, first each criterion is labelled with roman numerals for the sake of simplicity and clarity.

1. Real-Time Processing: Essential for immediate data analysis and feedback.
2. Performance Efficiency: Critical for efficient data handling and processing.
3. Scalability: Vital for managing growing data and client loads.
4. Integration Capability: Key for seamless interaction with existing systems and technologies.
5. Historical Data Management: Important for long-term data analysis and trend identification.
6. Maintenance and Updates: Necessary for ensuring system reliability and relevance.
7. Development Speed: Important but secondary to performance and integration.
8. Language Ecosystem: Relevant for accessing necessary tools and libraries.
9. Community Support and Resources: Useful but less critical than performance-based criteria.
10. Execution Model: Important but less so than the real-time processing and scalability.

The weighting is performed by comparing each criterion pairwise and rated depending on the relative importance. Points are given by following logic.

Table . Points criterion

|  |  |
| --- | --- |
| Criterion: head-to-head rating | Points |
| More important | 2 |
| Equal | 1 |
| Less important | 0 |

Then, the specific points are normed into weights with formula, ranging from one to five, where one indicates the lowest importance and five the highest.

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | I | II | III | IV | V | VI | VII | VIII | IX | X |
| I |  | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 0 |
| II | 1 |  | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 0 |
| III | 1 | 1 |  | 1 | 1 | 0 | 0 | 0 | 0 | 0 |
| IV | 1 | 1 | 1 |  | 1 | 0 | 1 | 1 | 0 | 0 |
| V | 2 | 1 | 1 | 1 |  | 0 | 0 | 0 | 0 | 0 |
| VI | 2 | 1 | 1 | 2 | 2 |  | 1 | 1 | 1 | 0 |
| VII | 2 | 2 | 1 | 1 | 2 | 1 |  | 1 | 0 | 1 |
| VIII | 2 | 2 | 2 | 1 | 2 | 1 | 1 |  | 0 | 0 |
| IX | 2 | 2 | 2 | 2 | 2 | 1 | 1 | 1 |  | 1 |
| X | 2 | 2 | 2 | 2 | 2 | 2 | 1 | 1 | 1 |  |
| Σ | 15 | 13 | 12 | 11 | 14 | 4 | 5 | 4 | 2 | 2 |
| Weight | 5 | 4 | 3 | 3 | 5 | 2 | 2 | 2 | 1 | 1 |

Figure 4.7 Weighting of technology selection criteria for a blockchain implementation by a head-to-head comparison.

# Given scores for each criterion from the pairwise comparison

summed\_scores = {

'Real-Time Processing': 15,

'Performance Efficiency': 13,

'Scalability': 12,

'Integration Capability': 11,

'Historical Data Management': 14,

'Maintenance and Updates': 4,

'Development Speed': 5,

'Language Ecosystem': 4,

'Community Support and Resources': 3,

'Execution Model': 2

}

# Weights calculation parameters

lowest\_weighted\_value = 1

max\_weight = max(summed\_scores.values()) # The highest score from the pairwise comparison

factor = 4

# Calculating the weights using the given formula

weights = {criterion: round(lowest\_weighted\_value + (score / max\_weight) \* factor) for criterion, score in summed\_scores.items()}

This is the script used to calculate the weights of all 10 criteria.

Following the methodology presented in section 4.8, the next step is to evaluate each technology on the level of performance for every single criterion (I – X). The level of performance will be evaluated with following logic shown in Table 4.5: Scoring logic for the evaluation of criteria I to X

Table 4.5 Scoring logic for the evaluation of criteria I to X

|  |  |
| --- | --- |
| Level of performance | Score |
| Complete satisfaction | 5 |
| Almost all functionalities available | 4 |
| Some features missing | 3 |
| Most features missing | 2 |
| No fulfilment | 1 |

Node-RED, an open-source, flow-based programming tool, was developed by IBM's Emerging Technology team and is designed for the seamless integration of Internet of Things (IoT) devices, various APIs, and online services. As a JavaScript tool grounded in the Node.js environment, Node-RED is distinguished by its no-cost, browser-accessible flow editor that showcases nodes with distinct icons for different functions ([K Ferencz](https://scholar.google.de/citations?user=glcyilMAAAAJ&hl=de&oi=sra), [J Domokos](https://scholar.google.de/citations?user=EoCruyIAAAAJ&hl=de&oi=sra) et al 2019).

The nodes within Node-RED offer a range of functionalities, including flow monitoring and interaction with the GPIO pins on Raspberry Pi hardware. The flows, which are the sequences of nodes that dictate how data is processed or how devices are controlled, are saved in the JSON format, facilitating ease of storage and transfer. Node-RED's primary appeal lies in its ability to connect different web services, custom nodes, and physical devices, thereby enabling tasks like forwarding sensor data via email or integrating with platforms like Twitter. It also allows for straightforward execution of more complex data processing tasks (M. Lekić and G. Gardašević 2018).

There are three principal components in the Node-RED interface:

* Node Panel: A repository of nodes that can be used to construct flows.
* Flow Panel: The workspace where nodes are dragged, dropped, and connected to define the application logic.
* Info and Debug Panel: Sections that provide information about specific nodes and aid in debugging flows by displaying messages and output.

Node-RED's versatility and power lie in its adaptability, enabling rapid application development. This tool is particularly valuable for applications that respond to events, which is a common requirement in IoT applications. Its design philosophy is to empower engineers and technicians to effortlessly devise and configure applications that operate in real-time directly on endpoint devices.

Python is a high-level, interpreted programming language known for its simplicity and readability, which significantly reduces the complexity of writing and maintaining scripts. Python's standard library is vast, offering modules and functions for variable types and operations, system calls, and even Internet protocols. It is a popular language across several domains, from web and software development to scientific and mathematical computing (Roulet-Dubonnet 2018).

Python excels in data analysis, machine learning, and back-end development, thanks to its extensive ecosystem of third-party packages and frameworks. Libraries such as NumPy and pandas for data manipulation, Matplotlib for data visualization, and TensorFlow and scikit-learn for machine learning are integral to Python's popularity in data-centric fields.

Table 4.6 Table of weighting the criterion based on our project needs

|  |  |  |  |
| --- | --- | --- | --- |
| Criterion | Weight | Node Red | Python |
| I | 5 | 5 | 3 |
| II | 4 | 4 | 3 |
| III | 3 | 3 | 5 |
| IV | 3 | 5 | 5 |
| V | 5 | 3 | 5 |
| VI | 2 | 5 | 2 |
| VII | 2 | 4 | 4 |
| IX | 1 | 4 | 5 |
| X | 1 | 4 | 4 |
| Overall performance |  | 110 | 103 |

In the above table we defined the values to the technologies based on the based and the project requirement. After multiplying each score by the respective weight of the criterion, the overall performance scores were calculated, resulting in 110 for Node-RED and 103 for Python.

import matplotlib.pyplot as plt

import numpy as np

# Criteria labels

labels=np.array(["I", "II", "III", "IV", "V", "VI", "VII", "IX", "X"])

# Scores for Node-Red and Python based on the given weights and criteria

scores\_node\_red = np.array([5, 4, 3, 5, 3, 5, 4, 4, 4])

scores\_python = np.array([3, 3, 5, 5, 5, 2, 4, 5, 4])

# Number of variables we're plotting.

num\_vars = len(labels)

angles = np.linspace(0, 2 \* np.pi, num\_vars, endpoint=False).tolist()

# The radar chart expects the first value to be repeated at the end.

scores\_node\_red = np.concatenate((scores\_node\_red,[scores\_node\_red[0]]))

scores\_python = np.concatenate((scores\_python,[scores\_python[0]]))

angles += angles[:1]

# Plot

fig, ax = plt.subplots(figsize=(8, 8), subplot\_kw=dict(polar=True))

# Draw the outline of our data.

ax.fill(angles, scores\_node\_red, color='red', alpha=0.25)

ax.fill(angles, scores\_python, color='blue', alpha=0.25)

# Draw the inside lines

ax.plot(angles, scores\_node\_red, color='red', linewidth=2, label='Node-RED')

ax.plot(angles, scores\_python, color='blue', linewidth=2, label='Python')

# Define the labels for each axis

plt.xticks(angles[:-1], labels)

plt.title('Technology Selection Comparison: Node-RED vs Python')

ax.legend(loc='upper right', bbox\_to\_anchor=(0.1, 0.1))

plt.show()

This is code snippet used to create the radar chart with all these criteria and it is clearly discussed in the next section.

### Radar Chart Visualization

A diagram of a hexagon with red and blue lines

Description automatically generated

Figure 4.8 Radar chart for technology selection of between Node-Red and Python w.r.to Digital Twin

To visualize the comparison, a radar chart was constructed as seen in the fig.4.8 using Matplotlib in Python. The chart plots the weighted scores of each technology against the evaluation criteria on axes starting from the same point. The enclosed areas formed by the lines connecting each point provide a visual representation of the strengths and weaknesses of each technology relative to the project requirements. The chart revealed that Node-RED scored higher in criteria such as 'Real-Time Processing,' 'Performance Efficiency,' and 'Maintenance and Updates,' indicating its suitability for projects where these aspects are crucial. On the other hand, Python exhibited strengths in 'Scalability,' 'Integration Capability,' and 'Historical Data Management,' showcasing its potential in applications demanding these capabilities.

The radar chart thus serves as a decision-making tool, depicting how Node-RED and Python align with the specific technological needs of our digital twin project. It illustrates that while Node-RED may excel in certain areas, Python's strengths in others make it a valuable technology for different aspects of the project. Consequently, a decision was made to utilize both technologies to their respective advantages within our implementation.

## Soft Sensor

As discussed in 3.1.1. a soft sensor calculates a desired quantity based on online measurement data and an algorithm that cannot be measured in or online. The soft sensor developed in this work is designed to calculate the concentration of magnetic particles in the liquid phase based on total density, temperature, and conductivity. The basis for accurately calculated values is the correctness of the values from the sensors on which the soft sensor is based. In preliminary tests, the density and conductivity sensors were therefore analysed for stability and correctness.

### Density determination

For the soft sensor, it is necessary to calculate the ion concentration from the measured conductivity and the density of the ionic solution from the ion concentration. For electrolyte solutions with several ion species, it is complex to calculate the density theoretically as a function of the ion concentrations. A pragmatic approach is to create a calibration line that calculates the density as a function of ion concentration. For this purpose, a MNP synthesis is performed, the magnetic particles are separated with a hand magnet and the supernatant is decanted. A dilution series is prepared from the supernatant and in each case the density is determined via the ratio of mass 𝑚𝑃𝑟𝑜𝑏𝑒 to volume 𝑉𝑃𝑟𝑜𝑏𝑒 according to equation eq.

Based on the weights and stoichiometry of the MNP synthesis, it is assumed that a total of 2.44 mol L - 1 of sodium, chloride and hydroxide ions are present dissolved in the supernatant after synthesis. A defined volume is achieved by transferring the samples into 50 ml volumetric flasks. The weight of each dilution stage is determined with a balance (Entris, Sartorius AG, Germany). The volumetric flasks are measured with and without liquid so that the difference can be calculated.

### MNP Concentration

The MNP concentration is a crucial process parameter and not easy to measure with online physical sensors. In this work, we developed a soft sensor that calculates the MNP concentration based on the density measurement of the available Coriolis flow meter. In the simple case, that we only have MNP in water present following formula can calculate the MNP concentration 𝑐𝑀𝑁𝑃 based on the measured total density 𝜌𝑡𝑜𝑡𝑎𝑙. A prerequisite is, that the density of all single components is known.

If salts are present in the solution, this equation can still be used if the concentration and density are known. If the salt concentration is unknown, it is possible to determine the salt concentration with the help of the conductivity sensor. To determine the MNP concentration online, a short MATLAB script was created.

for i = 1:inf

selectquery\_1 = 'SELECT \* FROM online\_data ORDER BY timestamp DESC LIMIT 1';

database\_1 = select(conn,selectquery\_1);

tempW = table2array(latestData(1, 5)); %[°C]

tempW\_old = table2array(latestData(15, 5)); %[°C]

d\_total = table2array(database\_1(1,8));

%% Warning - Temperature Sensitivity of Coriolis

d\_t = 15\*1.5/60; % [min]

Temperature\_change\_rate = (tempW - tempW\_old)/d\_t;

temp\_warning = 0; % default to no warning

if Temperature\_change\_rate > 0.1

temp\_warning = 1; % set warning flag 34

end

d\_water = ((-4e-5\*(temp\_w)^2)+(0.0013\*temp\_w)+0.9893);

cMNP = (d\_total - d\_water)./ (1-((d\_water)/5.17))\*1000;

c\_mnp = {cMNP};

solution = cell2table(c\_mnp);

sqlwrite(conn,'softsensor',solution);

waitfor(5);

end

Since this soft sensor is to run the entire time during the process, it is based on an infinitely running for-loop that queries a data set of the online data from the database every 5 seconds. The current fluid temperature, old fluid temperature, and the total density are then retrieved from this data set. The temperature change rate is calculated by using the 2 temperatures. If the temperature change rate exceeds a value of 0.1, the variable temp warning is set to 1, which leads to the display of a warning message in Grafana. With the fluid temperature, the density of water can be determined via a created calibration. And following this, the concentration of MNP in the solution. The results must now be formatted and sent back to the database. After that, it waits 5 seconds to the next iteration.

# Concept and Implementation

In this chapter, the methodology used to develop the digital infrastructure or cyber system of the Rotary stator magnetic separator machine's digital twin will be described in detail along with the basic structure of the existing Node Red functionality. The methodology includes the development of the Python API for data collection from the PLC, the utilization of OPC UA client-server protocol, the design and implementation of the database using PostgreSQL, the Containerization on process for isolation and communication, and the creation of the GUI using the PyQt5 library.

## **Data Processing using Node-Red**

The existing architecture of the Node-RED setup involves a complex structure for reading data from the OPC UA server and processing it. The setup includes various blocks such as the 'Read multiple' block for determining data retrieval frequency, the 'Clear array' block for data cleanup, and the 'OPC UA Client' block for establishing a connection with the OPC UA server on the PLC. Additional blocks such as 'OPC UA Node' and 'Inject' are utilized in the data processing flow. Data processing includes tasks such as creating new topics, combining, and assigning topics to messages, modifying data points for database compatibility, and writing the processed data into databases. The system also incorporates checks for specific messages, such as a 'Recipe-Reset' message for process run determination and retrieval of recipe and pre/post process data. However, making changes within this complex Node-RED structure can be challenging compared to the Python API, which simplifies the process by establishing a connection with the OPC server, creating node lists for different tables, periodically reading values, and writing them into the database.

A diagram of a diagram

Description automatically generated

Figure 5.1 Processing MS, SS, and pre/post process data. First, a trigger block checks if a "Recipe-Reset" signal was sent (1). If that is the case, the data gets processed analogue to the online data until the creation of the input query (2). A separate function block is used for each data type to split up the message containing all the combined data. Additionally, for the process data, it is checked if its pre or post process data (3) as each data set needs to be written in a different DB but are entered at the same point in the HMI. Before the data is written into the database, a filter block is used. This prevents identical data from being written in DB again (4).

## Overview of an API

The purpose of the API is to play a crucial role in Digital Twin by connecting to an OPC-UA server, collecting essential data, and storing it in a PostgreSQL database. The project aims to efficiently gather real-time data from various sensors and actuators. By using the capabilities of the OPC-UA protocol, we can establish a reliable connection to the server and retrieve the necessary information. The API acts as a vital component in our data acquisition system, providing the seamless collection and storage of data for further analysis and decision-making processes. In the following sections, the functionalities and implementation details of API, and its significance in overall Digital Twin architecture was explained.

The PLC data is divided into 3 categories, and each must be processed in a different way.

Online data: online data must be retrieved and written to the database every second.

Sub and main steps: This data must be transmitted only once at the beginning of the process.

Pre- and post-process data: This data must be retrieved once at the beginning and once at the end.

The flow chart of the code is shown in fig.5.2.

A diagram of a flowchart

Description automatically generated

Figure 5.2 Flow chart of the API. The API first initiates the connection with OPC sever and Postgres DB. Once establish it further divided into 2 threads, checks the conditions, and updates the corresponding tables already created in our database.

### OPC UA client-server functioning

In industrial automation, the OPC UA protocol is the trusted means of communication between devices and software that uses TCP/IP and transfers data in packets explained clearly in 3.4.1. The core part is the client-server interaction. On the server side, as shown in the figure 5.3. PLC S7-1500 has an integrated OPC server. Once this server is activated, it gets assigned an IP address, often by a DHCP server [37]. However, address assignment is flexible; while the DHCP can auto-assign IP addresses, there's also the option to manually configure them.(Lange, Elliot et al. 2021) The switch was used to connect all the devices in our local network for flexible communication and will create the IPs in the same subnet.

The beauty of OPC UA lies in its simplicity. As it runs on the widespread TCP/IP standard, there’s no need for unique hardware for the connection. S7-1500 models come with an Ethernet port, simplifying the connection process. On the client side, software platforms, be it Node-RED or Python API, using the IP of the OPC-UA server as an endpoint a connection was built. After establishing this connection, the client can then request data or send commands to the PLC, interacting with specific data points or node IDs. Each node ID corresponds to a particular sensor, actuator, or process parameter.

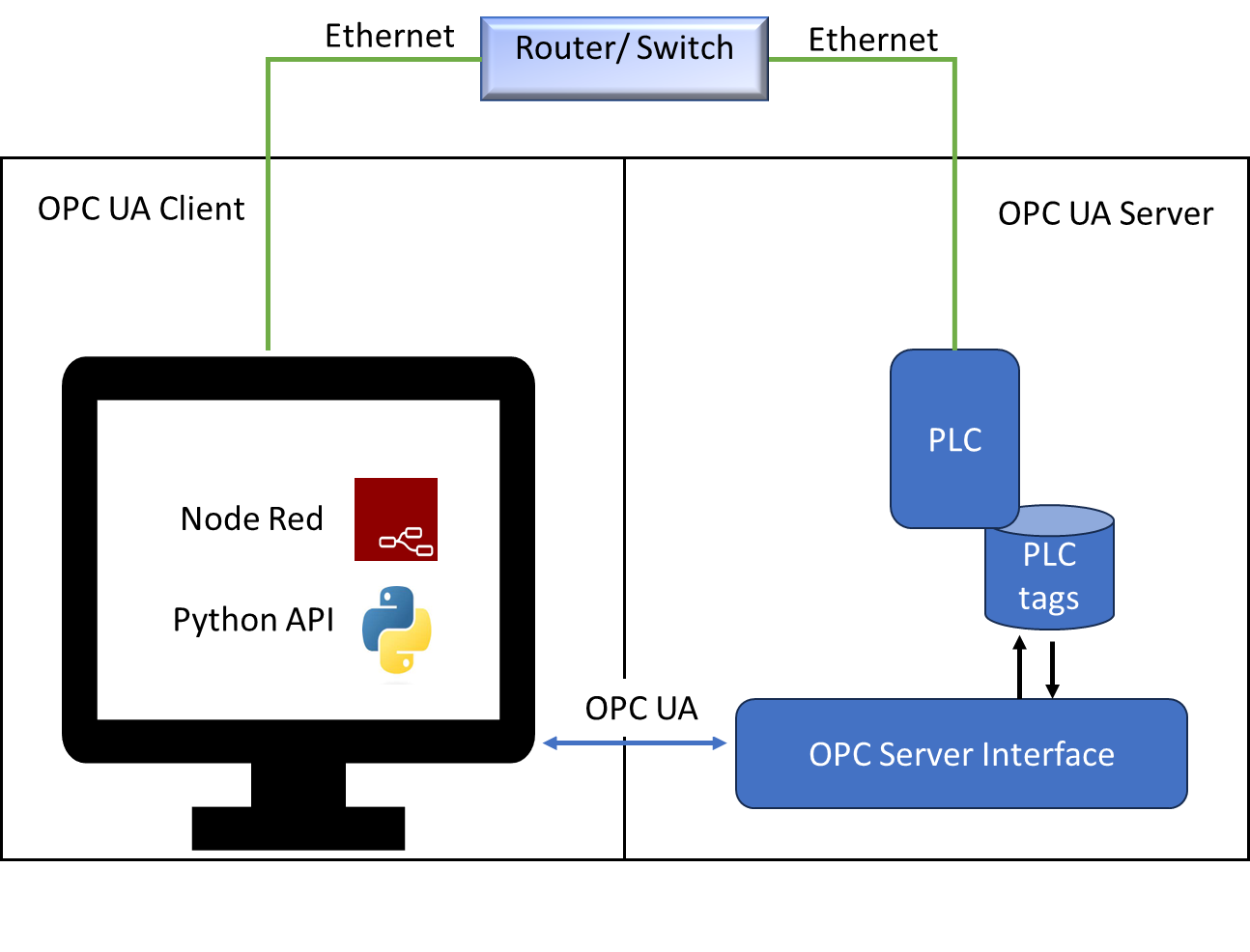


Figure 5.3 OPC UA Client-Server set up and functioning in the project. The OPC clients are docker container microservices running in rev pi. And OPC server is activated in PLC. Both are connected to the switch via ethernet to avoid multiple network subnet.

### Connection to OPC Server

The below-mentioned line establishes a reliable connection with the OPC-UA server utilizing the capabilities of the OPC UA library in Python. It utilizes the provided URL to establish communication and retrieve data from the server. This connection is vital as it forms the foundation for data retrieval and subsequent storage.

A screenshot of a computer

Description automatically generated

Figure 5.4 Connection with the OPC server using the URL created by DHCP server. OPC UA inbuilt client method was used to establish connection with server.

### Node lists

Node lists play a crucial role in the data transfer, as they serve as containers for storing the node IDs/PLC tags defined in the PLC. In our Digital Twin system, each sensor, process parameter, actuator, switch, and button are associated with a specific node ID also known as PLC tags. These node IDs act as bridging units, connecting the hardware components of the system with the software implementation.

To maintain organization and facilitate data collection, the node lists are categorized based on the type of data we intend to collect. As mentioned earlier, we have separate node lists for pre-process, main steps, sub-steps, and post-process data. This categorization ensures efficient data retrieval and processing, allowing us to capture the necessary information from the PLC. These node IDs serve as references, enabling us to read the values from the PLC and collect the real-time data required for our project.

A screenshot of a computer

Description automatically generated

Figure 5.5 Some node ids created in the PLC which are very crucial for collecting sensor data.

### Connection to PostgreSQL database

It is decided to update to timescale DB which is built over a PostgreSQL server and has additional features, particularly for time series data, and provides timestamps for each entry. The below line of code builds the connection with the database using the psycopg2 library in Python.

Considering security concerns, instead of hardcoding the database credentials in the code we decided to create the global environment variables and call those variables which are stored in a system environment which only the limited people have access to.

By setting the corresponding environment variables (DB\_NAME, DB\_PORT, DB\_USER, DB\_PASSWORD, and DB\_HOST), you can build the connection with the database. This approach enhances flexibility and security, as it allows the ability to easily change the connection details without modifying the codebase and ensures that sensitive information such as database credentials is not exposed in the code.

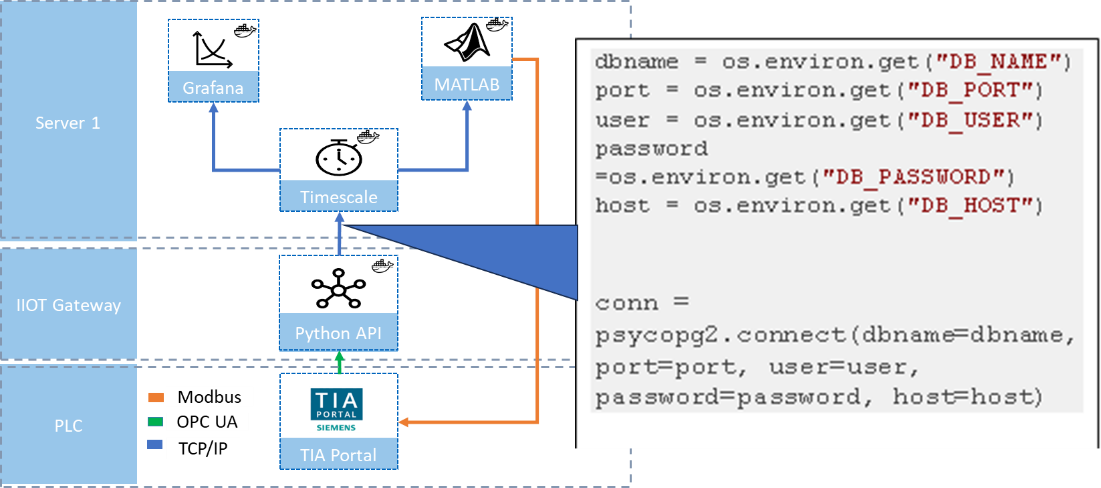


Figure 5.6 TCP/IP connection between API and database to write the data in the Timescale DB

### Data retrieval and storage

To retrieve data, the code utilizes the get\_value() method from the OPC-UA client, which establishes a direct connection with the OPC-UA server using the OPC-UA protocol. This protocol operates on the TCP/IP network stack, providing reliable and ordered data transmission between the client and server (Lange, Elliot et al. 2021). The data retrieval process is carried out separately for each category, including online data, pre-process, main steps, sub-steps, and post-process. This ensures that the code accurately captures the relevant data points specific to each category. A code snippet below creates a new list with values of node ids by using the GET method.

values\_pre\_process = [client.get\_node(node).get\_value() for node in node\_list\_pre\_process]  
values\_main\_steps = [client.get\_node(node).get\_value() for node in node\_list\_main\_steps]  
values\_sub\_steps = [client.get\_node(node).get\_value() for node in node\_list\_sub\_steps]  
values\_post\_process = [client.get\_node(node).get\_value() for node in node\_list\_post\_process]

### Data Processing and Insertion

Once the data is retrieved the data processing is done based on the digital twin and inserted into different tables created in the database. As mentioned in 5.2 the data is divided into three categories. For simplicity and to avoid multiple OPC clients, it was written in two threads one for collecting online data and the other for sub-step main steps, pre, and post-process. With this, both threads can perform simultaneously without affecting others' performance. Collecting online data is straightforward and while a loop is created it will run once after establishing a connection with the database. In thread 2 specific PLC tags such as 'recipe\_transfer\_reset\_DB,' were particularly defined as a button in TIA as shown in fig 5.5 When the PLC logic triggered specific events, such as 'Reset\_pre' or 'Reset\_post', API accessed the corresponding tag values using their unique node IDs. This allowed it to collect real-time data from the PLC and efficiently store it in a database. By utilizing these predefined PLC tags and their associated logic, the data acquisition process enhances the overall efficiency and reliability of the system.

A screenshot of a computer

Description automatically generated

Figure 5.7 Recipe\_reset button on HMI based on the status of these buttons API collects the data and write on the tables accordingly.

for i in range(len(values\_pre\_process)):  
 if node\_list\_pre\_process[i] == 'ns=3;s="Lab\_Parameter"."Operation"':  
 if values\_pre\_process[i] == 1:  
 values\_pre\_process[i] = 'Washing BIONs'  
 elif values\_pre\_process[i] == 2:  
 values\_pre\_process[i] = 'Protein purification'  
 elif values\_pre\_process[i] == 4:  
 values\_pre\_process[i] = 'Test'  
 elif "AdsorptionModel" in node\_list\_pre\_process[i]:  
 if values\_pre\_process[i] == 1:  
 values\_pre\_process[i] = "Langmuir"  
 elif values\_pre\_process[i] == 2:  
 values\_pre\_process[i] = "Freundlich"  
 elif values\_pre\_process[i] == 4:  
 values\_pre\_process[i] = "test"

Once after processing the data based on the data types. Using the cursor object, an SQL INSERT query is executed to insert the processed data into the appropriate database table. The values are organized within the query to match the respective fields of the table, ensuring accurate data insertion. In the PLC logic, specific processes were assigned in numerical identifiers for efficient data handling. For better understanding and readability in the database, the code translates these numerical values back into their descriptive process names. Now to deploy the API in the Rev Pi with limited resources and ARM architecture which is not compatible with the advanced Python libraries used in API. So docker was used and the implementation methodology is briefly discussed in Chapter 6.

# Implementation of Docker

Docker, a powerful containerization platform, addresses the challenges by providing a standardized and portable approach to packaging and deploying software applications. Docker enables the creation of isolated containers that encapsulate the application and its dependencies, allowing it to run consistently across different environments.

A diagram of a server

Description automatically generated

Figure 6.1 Docker working process. Once after going to application repository where the docker file exists build process can be initiated. Build process uses the Linux kernel and builds the image packaging all the dependencies and pushed into Docker hub registry. Pull process can be done from any machine irrespective of its environment.

## **Importance of Docker**

**Dependency Management**: With traditional software installations, managing dependencies and ensuring compatibility can be complex. Docker simplifies this process by packaging the application and its dependencies into a single container, eliminating conflicts, and ensuring consistent behaviour across different systems.

**Reproducible Environments**: Docker allows to define the exact environment needed for their application to run. By packaging the application, its dependencies, and the operating system into a container, Docker makes sure that the environment is reproducible, irrespective of the host system This reduces the chances of deployment issues caused by differences in system configurations. [34]

**Resource Efficiency**: Docker containers are lightweight and provide efficient resource utilization. Multiple containers can run on a single host, enabling scalability, and optimizing resource allocation. We can also remove some of the dependencies that we need to install some packages once after installing them for example GCC compiler in my docker image. This problem is discussed in 6.2 and how we overcome it by enabling advanced docker features.

**Portability**: Docker containers are portable, meaning they can run on any system that supports Docker, regardless of the underlying operating system or hardware. This flexibility makes it easier to deploy applications across different environments, whether it's a local development machine, a testing server, or a production environment. This is very crucial in our case we don’t have any hardware to access Rev Pi, we just used it remotely with the help of SSH.

**Isolation**: Docker containers provide process isolation, ensuring that applications running within containers are isolated from each other and the host system so that the performance of the application will be improved [34]. In our project, we are running both Node-Red and Python containers in Rev Pi which needs different environment and with the help of containerization we created them inside the containers without affecting the global environment.

## Creating Docker images

After testing API on different platforms Certainly! Here's a sample draft for the "Creating Docker Image" chapter. To package and deploy the application in a portable and consistent manner, a Docker image is created. The Docker image is built using a Docker file, which defines the necessary steps to set up the application's environment and dependencies. The Docker file for this project is as follows.

# Use the official Python image from the Docker Hub  
FROM python:3.8-slim-buster  
# Set the working directory in the docker  
WORKDIR /main  
# Install any dependencies  
RUN apt-get update && \  
 apt-get install -y --no-install-recommends gcc make libxml2-dev   
 pip install --no-cache-dir opcua psycopg2 && \  
 apt-get purge -y --auto-remove gcc make && \  
 apt-get clean && \  
 rm -rf /var/lib/apt/lists/\*  
  
# Copy the content of the local src directory to the working directory  
COPY . .  
  
# Specify the command to run on container start  
ENTRYPOINT ["python3", "merged.py"]  
  
CMD [“. /merged.py"]

The Docker image is based on the official Python image. 3.8 slim buster was used specifically to keep the size of the image low. Based on the packages, several versions including OS like Ubuntu were tried and found this be optimum. It is the slim Python version of the Debian Buster OS. This provides the foundational operating system for running the application. The working directory within the Docker image is set to `/main`, where the application code will be copied. The main.py file, which contains the code for the project, is added to the working directory using the `ADD` command. The two main dependencies of the API are OPCUA and psycopg2 and there are additional system packages like GCC and make which are required to install main packages. As the docker image follows the top-down approach I installed prerequisite system packages first and later I installed main packages as mentioned above post installation, unnecessary system packages (gcc and make) were removed to reduce the image size. Lastly, the cache and temporary files are cleared to further minimize the size of the image. All the files from the host folder must be copied to the main folder which is created as a working directory. This ensures all the files must be available inside the container. When the container starts, it will run the merge\_updated.py script using Python 3. The ENTRYPOINT specifies the base command for the container, and CMD provides default arguments that can be overridden when starting the container.

## Building the Docker Image

1. **Initial Build Challenge**

Initially, the plan was to build the Docker image directly on the Raspberry Pi. However, due to the limited computational resources available on the Rev Pi, building a dependency-intensive Docker image was very difficult. The build process needed more resources and the Rev Pi struggled to manage it crashed after running continuously for the entire day.

1. **Local Machine Build and Docker Hub Push**

For that reason, it is decided to build the docker image on the local machine with more powerful resources. The plan was to build the image on a local machine push it docker hub and then pull that image to the Raspberry Pi.

1. **Architectural issue**

Docker images are typically built for specific system architectures. While most standard general-purpose computers and servers utilize the AMD architecture, devices like the Raspberry Pi run on different architectures, in this case, ARMV7[39]. This difference in system architectures becomes an issue when deploying Docker images. An image built on an AMD architecture system will not function on an ARMV7 system like the Raspberry Pi, and vice versa. The reason is, different architectures have unique instruction sets, and software compiled for one cannot run directly on the other.

1. **Multi-Architecture Docker Build**

Docker Buildx is the solution to this problem. With the buildx we can build the docker images compatible with multiple architectures.[33]

docker buildx create --name mybuilder  
  
docker buildx use mybuilder  
  
docker buildx build –platform linux/armv7 -t dockerhub\_username/image\_name: tag .  
  
docker buildx push dockerhub\_username/image\_name: tag

1. **Deployment on Rev Pi**

The image was pushed to the docker hub and pulled to the rev pi with the following command. Used bind volume mount ensuring a continuation of development of the code. Also used env variables to configure database container.

docker run -e "DB\_HOST=ip" -e docker run -e "DB\_HOST=ip" -e "DB\_PASS=password" -d dockerhub\_username/image\_name

## Docker Volumes

One of the essential features of the docker is its ability to manage volumes, enabling the data to stay persistent even if the container goes down with unexpected issues. Docker volumes serve as a mechanism for decoupling the data lifecycles and container lifecycles [34]. There are 2 types of docker volumes we used both volumes to achieve flexibility as explained below.

### Docker Bind Volumes

These volumes are mappings between the host system's file system and the container's file system. By utilizing bind volumes, one can ensure that the code or data updated on the host is reflected within the container. So, whenever we make any changes in the code, we don’t have to create a new image and run the container from it. This feature is especially beneficial for development environments as it provides a live-sync capability between the container and the host [34].

In this project, bind volumes have been employed for our API or application. This approach guarantees that any future code changes or updates on the host system are instantly reflected within the running container, avoiding the need to rebuild the container every single time.

docker run -v /path/on/host:/path/in/container \ &&  
 -d your\_dockerhub\_username/your\_image\_name: multiarch

### Docker Volume mounts

Unlike bind volumes, these are created and managed by Docker itself. They are stored in a part of the host filesystem that's managed by Docker (/var/lib/docker/volumes/). Using Docker volumes is especially recommended when you want to store data generated by and used by Docker containers [31].

For this project, Docker volumes were particularly useful for our database container. Given databases' dynamic nature, where data constantly gets updated, removed, or added, Docker volumes ensured data persistency. Even if our database container is stopped or deleted, the data remains and can be reattached to a new container.

docker run -d -v /path/file/path:/var/lib/PostgreSQL/data  
 name DB

## Creating Docker Network

"To ensure seamless communication and connectivity between containers, a custom Docker network is created using Docker Swarm. Docker Swarm is a native clustering and orchestration tool for Docker, allowing the creation and management of a swarm of Docker nodes. (Lee 2017)

A screenshot of a computer

Description automatically generated

Figure 6.2 Docker overlay network to establish connection between two hosts.

To create a custom Docker overlay network, the following steps were followed:

Docker Swarm Initialization: The Raspberry Pi, acting as the swarm manager, was initialized using the `**docker swarm init**` command. This command establishes the Raspberry Pi as the swarm manager and generates a join token for other nodes to join the swarm.

Joining the Swarm**:** The other server, acting as a worker node, joined the swarm by using the provided join token. This step enables the server to participate in the Docker swarm and communicate with other nodes in the swarm.(Lee 2017)

Creating the Overlay Network**:** With the swarm set up, an overlay network was created using the `**docker network create**` command. The overlay network allows containers running on different nodes to communicate with each other seamlessly.

Attaching Containers to the Overlay Network: The API container running on the Raspberry Pi was attached to the custom overlay network. This ensures that the API container can communicate with other containers within the network, such as the database container.

Database Container in the Overlay Network: The database container running on the server was also joined to the custom overlay network. This configuration enables the API container to store data in the database, facilitating data persistence and retrieval.

By utilizing a custom Docker overlay network, containers running on different nodes, such as the API container on the rev Pi and the database container on the server, can communicate with each other as if they were on the same network. This ensures efficient data flow and seamless connectivity between the API and the database, enabling the storage and retrieval of data.

Table 6.1 Docker container under custom docker network: Overlay network.

|  |  |  |
| --- | --- | --- |
| Containers | IP address | Port |
| Timescale DB (For API) | 10.0.0.3 | 5432 |
| API | 10.0.0.4 | Not necessary |
| PgAdmin | 172.20.0.34 | 82 |

Table 6.2 Docker Containers under custom Docker network: RSMS network

|  |  |  |  |
| --- | --- | --- | --- |
| Containers | IP address | Docker Port | Host Port |
| Influx DB | 172.20.0.11 | 8086 | 8088 |
| PostgreSQL | 172.20.0.33 | 5432 | 5433 |
| Timescale DB | 172.20.0.35 | 5432 | 6543 |
| PgAdmin | 172.20.0.34 | 80 | 82 |
| MATLAB | 172.20.0.23 | 8888 | 8888 |
| Grafana | 172.20.0.13 | 3000 | 3002 |

Docker networking offers various options for networking containers, including bridge networks, host networks, and overlay networks. Each option has its own use cases and benefits. In this scenario, the overlay network was chosen due to its ability to facilitate communication between containers running on different hosts. The overlay network also provides built-in load balancing and service discovery features, making it suitable for distributed applications and multi-host environments. Additionally, Docker networks offer advanced networking features, such as DNS resolution for service names, network segmentation, and IP address management, further enhancing the flexibility and scalability of the application architecture.[34]

RSMS bridge network also exists in which all the containers running in the same host are attached to create a static Ips to avoid configuration mismatch if any container dies with custom network, we can assign Ips to the containers so that we can have track of existing Ips when assigning IP to ne container. By utilizing the power of Docker networking and Docker Swarm, the API container on the Raspberry Pi seamlessly communicates with the database container on the server, enabling efficient data storage and retrieval, even across different nodes within the Docker Swarm."

## GUI for Digital Transparency

In our digital twin project, MATLAB simulation models create plots based on given parameters and recipe data. To make these findings clear and useful, we developed a GUI using PyQt5 library in Python. It shows two plot widgets: one for simulation results before optimization and the other for results after optimization. Accept and Reject buttons help users decide on optimized parameters. A dropdown button lets users select scenarios for visualization. The GUI works with the simulation system. When optimized parameters are accepted, it connects to the OPC UA server, finds the right node, and sends the parameters for the next simulation. We also made the GUI show external plots from MATLAB. These plots are stored in a specific path, and the GUI fetches and displays them. By updating MATLAB to replace old plots with new ones, users can compare simulation and MATLAB-generated plots for better decision-making.

A screenshot of a computer

Description automatically generated

Figure 6.3 Outline of GUI for digital transparency.

# Tests and Results

This chapter presents the work and results of the latency tests conducted, which aimed in evaluating the efficiency and responsiveness of the data retrieval from PLC OPC server and data storage in the timescale database. The latency tests play a crucial role in assessing the system ability to handle real-time data and provide valuable insights in API overall performance, reliability, and stability in DT.

## Validation of Soft sensor

A first validation of the models can be performed with the help of the breakthrough curves generated at the RSMS. To generate the curves, the RSMS was loaded with MNP. The loading was stopped when the concentration at the inlet was equal to the concentration at the outlet. During the experiments, samples were taken to perform dry mass (DM) measurements to determine the concentration of MNP. Before the experiments, the size distribution of the MNP was determined with the help of DLS. These curves were generated as part of Chen's work (Chen, 2022). In Table 7.1, important process information about the breakthrough experiments is displayed.

Table 7.1 Process information about the breakthrough curves.

|  |  |  |  |
| --- | --- | --- | --- |
| Experiment Nr. | MNP Concentration  [] | Flow rate  [] | SigmaMax  [g] |
| 1 | 14.14 | 1906 | 36.44 |
| 2 | 12.54 | 1372.5 | 50.36 |
| 3 | 27.4 | 1496.4 | 47.93 |
| 4 | 18.62 | 676.2 | 41.3 |
| 5 | 4.92 | 1514.2 | 43.56 |

As can be seen in Table the different breakthrough curves were performed with different flow speeds and inlet MNP concentrations. The results of the simulation and experimental data are shown in the fig below.

A graph of a number of different types of data

Description automatically generated with medium confidence

Figure 7.1 Here are the results of two breakthrough experiments displayed in sequence. A simulation was created for each breakthrough experiment. In experiment one, an inlet concentration of 14.14 g L-1 and a flow rate of 1906 ml min-1 was used. was used. In experiment five, an inlet concentration of 4.92 g L-1 and a flow rate of 1514.2 ml min-1

From these breakthrough curves and the simulated graphs created, the improved models can reproduce the experimentally determined results using the introduced size distribution, the optimized volume, and the max binding capacity of the matrix and the magnetization. For some of the performed experiments, the density data was also available to validate the designed soft sensor equation offline. In the fifth experiment, the size distribution shows that the inlet concentration also contains tiny particles. In Figure 7.2, the dry mass determination, the simulation from the model, and the concentration of MNP determined with the help of the soft sensor are shown.

A graph of a number of data

Description automatically generated with medium confidence

Figure 7.2 Here are the results of three breakthrough experiments displayed in sequence. For each breakthrough experiment, a simulation was created, and the concentration of MNP was determined using the soft sensor. Experiment two was performed with an inlet concentration of 12.54 g L-1 and a flow rate of 1372.5 ml min-1, Experiment three used an inlet concentration of 27.4 g L-1 and a flow rate of 1496.4 ml min-1 and experiment four used an inlet concentration of 18.62 g L-1 and a flow rate of 676.2 ml min-1.

Within these graphs, it can again be seen that the concentration curves determined with the help of dry mass can be mapped with the help of the models. In addition, the concentration change during magnetic separation can also be mapped using the equation developed for the soft sensor.

## Latency Test Methodology

For these tests, Python was chosen. It will be easy as the entire system is also in Python and has multiple libraries that will facilitate real-time data collection and analysis. In the existing Node-Red we are achieving a latency of around 1200 milliseconds. We implemented Python to see if we can improve data transfer latencies.

* 1. **Monotonic Clock library**

To accurately measure the time intervals, especially in the context of latency testing, it's crucial to use a clock that isn't affected by system clock updates. This is where the monotonic clock comes into play. The time.monotonic() function in Python returns the value of a monotonic clock, which can't go backward. This ensures that our latency measurements remain accurate and consistent even if the system clock is adjusted.

* 1. **Read vs. Write Latency Measurements**

Latency measurements were broadly divided into two categories.

Read Latency: The time taken to request and receive a response.

Write Latency: The time taken to store the data once it's read.

To measure these latencies, time stamps were taken right before sending a request and immediately after receiving a response, using the time.monotonic() function. The difference between these two timestamps gives the latency in terms of fractional seconds.

* 1. **Testing Environment and Variables**

Multiple tests were conducted under varying conditions, such as running Python alone, running Node-Red alone, and running both simultaneously. This was done to study the difference between different systems and their impact on latencies.

## Comparison of Read and Write Latencies

Table 7.2 Read vs Write Latencies

|  |  |  |
| --- | --- | --- |
| Metric | Average Latency (ms) | Standard Deviation (ms) |
| Read latency | 2543.4 | 35.4 |
| Write Latency | 36.8 | 5.3 |

One of the primary observations from our latency test results was the noticeable difference between the read and write latencies when using the Python implementation. So, the latency is in getting data from the server and the database part doesn’t have an issue. An essential factor believed to be influencing this disparity is the number of OPC UA clients accessing the server at the same time.

Table 7.3 Latencies of Python API when tested in 2 different scenarios.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Scenario | Avg. Read Latency | Std. Deviation (Read) | Avg. Write Latency (ms) | Std. Deviation (Write) |
| Python only | 1419.6 | 29.45 | 17.67 | 2.46 |
| Python and Node-Red | 2453.4 | 68.4 | 18.37 | 5.3 |

A graph of a line with blue and orange lines

Description automatically generated

Figure 7.3 Read latencies when Python and Node-Red both are running.

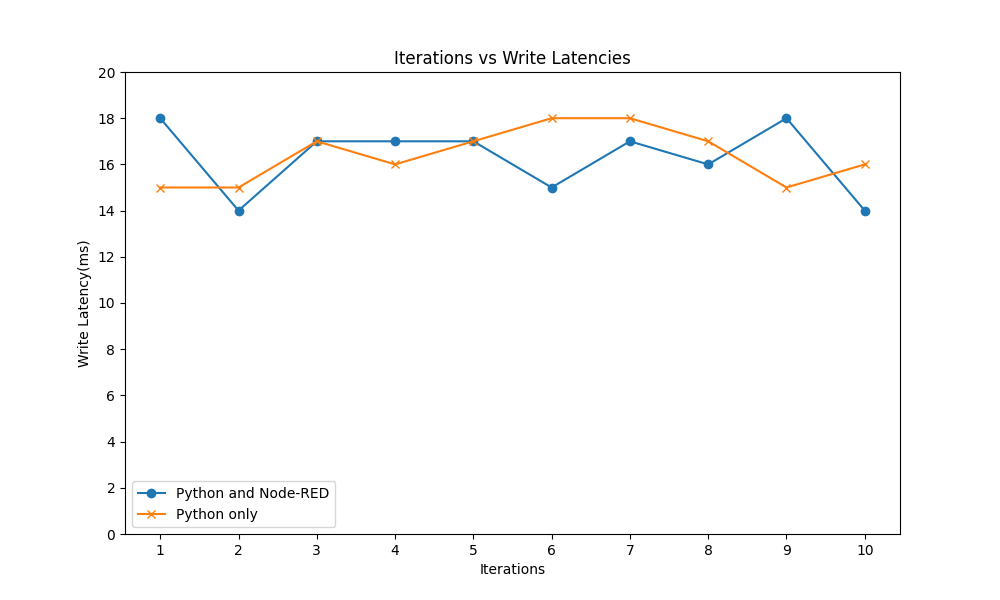


Figure 7.4 Write latencies when both Python and Node-Red

The generated plots illustrate the read and write latency outcomes from two distinct scenarios. It became evident that the write latency of the Python API remains unaffected by Node-RED. But a difference of one second was observed in the read latency results.

**Results of the various test cases:**

Several tests were done with different scenarios and did 10 iterations to find the average and standard deviation of read and write latencies when only python client is working and when both the Node-Red clients and python clients are working.

Read\_Latencies\_Python\_nodered: [2443.81, 2409.62, 2498.09, 2450.73, 2413.42, 2406.20, 2490.23, 2456.67, 2454.46, 2446.37]

Average read latency when both python and node red are running over 10 iterations: 2543.4 ms.

Standard deviation: 68.4ms

Read\_latencies\_Python: [1415.94, 1378.36, 1407.0183894954616, 1405.35, 1416.21, 1444.79, 1417.09, 1415.03, 1407.28, 1391.26]

Average read latency when only python is running over 10 iterations: 1419.6 ms.

Standard deviation of read latencies: 29.45ms

Write\_Latencies\_Python\_nodered: [18.1860, 18.8631, 17.1251, 17.1431, 18.5723, 18.1538, 18.8220, 18.8683, 16.6983, 17.9851]

Average write latency over 10 iterations: 18.37 ms.

Standard deviation of write latencies: 5.3 ms.

Write\_Latencies\_Python\_nodered: [17.5975, 16.9602, 17.1728, 16.9020, 17.5396, 17.3742, 17.1635, 17.6168, 17.7264, 18.4740]

Average write latency over 10 iterations: 17.67 ms.

Standard deviation of write latencies: 2.46 ms.

## Grafana Monitoring and Database Implications

Node-Red is performance is still better concerning latency as the latency of Node-Red is around 1200 ms in contrast with Python. However, the Grafana performance when using data from the table with Python data is better. The presence of a large amount of historical data in the Node-RED table, can have an impact on reading times and visualization software responsiveness. As the Python table created is relatively new, we decided to go with both tables. Node-Red table we decided to use for simulations which are time-dependent processes. And a Python table for monitoring with a retention time of one month.

## Discussion and Observations

As OPC UA works on a client-server model. In a typical scenario, a server hosts data and provides services, while the client requests the data and gets the response from the server. With the increase in the number of clients, the server will be loaded with multiple requests, and it is affecting the performance of the server which results in high latencies. Also, OPC-UA uses TCP/IP protocol which will make sure all the data packets are transferred with multiple checks which will always be relatively slower that UDP protocol. The UDP protocol is faster than TCP/IP but there will data discrepancies. In our project we have multiple OPC UA clients sending requests simultaneously. Node-Red is having 4 clients sending requests simultaneously and Python API uses 1 client. Several latency tests were performed creating different scenarios and the results are presented. Also the softsensors were successfully installed and the results are discussed clearly in the results section using the break through curves.

# Conclusion

This research aimed to measure and understand the latencies associated with an OPC UA client-server model using Python and compare it with a Node-Red-based system. The outcomes were revealing insights that could guide similar implementations in the future. One of the main findings from the tests was the considerable difference between read and write latencies in the Python-based system. Specifically, while the write latency remained low and consistent irrespective of the number of clients sending requests to the server, whereas the read latencies was notably higher. The data shows that the presence of multiple OPC UA clients could contribute significantly to increased latencies. A clear correlation was evident between the number of active clients and the extent of latency, with the combined usage of Python and Node-Red resulting in the highest observed read latencies.

**Difference between Node-Red and Python:** Although Node-Red showed lower latencies than the Python system, there was an issue in terms of Grafana performance due to the vast historical data in the Node-Red table. The choice between these two systems requires a careful evaluation based on the specific needs of the project, whether it is real-time monitoring or time-dependent simulations. One of the major reasons for node-red performance is it is built over java which is compiler based whereas python is interpreted language which is always bit slower compared to java but easy in implementation.

**Database Decisions for Optimal Performance:** Given the challenges observed with Grafana's performance when interfacing with the Node-Red table, a strategic approach was adopted. By using both the Node-Red table for simulations and a Python table (with a retention time of one month) for real-time monitoring, it's possible to achieve a balance between historical data access and real-time monitoring efficiency.

Limitations in this study mainly pertain to the specifics of the current system setup and may not generalize to all possible configurations of the OPC UA client-server model. However, these findings provide valuable insights for future implications in employing the OPC UA protocol, particularly when considering the implications of client count and database management on system latency.

**Future Work:** Building on this research, future investigations might explore optimizing server configurations, diving deeper into the intricacies of the OPC UA protocol with multiple clients, and possibly evaluating the performance across various other programming languages and platforms. Also, GUI integration with the digital twin which is initiated will be very great asset for transparency and validation of the predictions made by the models.

# References

1. Andritz. Rotor stator high gradient magnetic separator MES-RS. <https://www.andritz.com/resource/blob/269472/db33e03298223693667155ba878bbd1d/rotor-stator-high-gradient-magnetic-separator-mes-rs-data.pdf>. Accessed: 2023-05-06.
2. Chromatographytoday. Bio Analytical an introduction to bio separations. (2015). Retrieved from https://www.chromatographytoday.com/news/bioanalytical/40/breakingnews/an-introduction-to-bioseparations/34425. Published: 2015-04-30 , Accessed: 2023-07-05.
3. Matteo Perno, L. H. (2023). A machine learning digital twin approach for critical process parameter prediction in a catalyst manufacturing line. doi:https://doi.org/10.1016/j.compind.2023.103987
4. Broo, D. G., et al. (2021). "Cyber-physical systems research and education in 2030: Scenarios and strategies." DOI: <https://doi.org/10.1016/j.jii.2020.100192>
5. Chuck-Hernández, C., et al. (2022). Separation Processes in Pharmaceutical Manufacturing, Frontiers Media SA. 4: 865635.DOI:  <https://doi.org/10.3389/fceng.2022.865635>
6. Ebeler, M., et al. (2018). "Magnetic separation on a new level: characterization and performance prediction of a cGMP compliant “rotor‐stator” high‐gradient magnetic separator." DOI: https://doi.org/10.1002/biot.201700448
7. Fischer, T., et al. (2022). Snakes on a plan: Compiling python functions into plain SQL queries. Proceedings of the 2022 International Conference on Management of Data. DOI: [https://doi.org/10.1145/3514221.3520175](https://doi.org/10.1016/j.promfg.2018.02.034)
8. Jonathan Lustri. PAT how to implement process analytical technology in pharmaceutical manufacturing. <https://blog.isa.org/how-to-implement-process-analytical-technology-in-pharmaceutical-manufacturing>. Accessed 2023.08.05
9. Hernández Rodríguez and Frahm (2021) "Digital Seed Train Twins and Statistical Methods." https://link.springer.com/chapter/10.1007/10\_2020\_137
10. Lammle, T. and J. Buhagiar (2020). CompTIA Network+ Study Guide with Online Labs. pp. 1-38
11. Lange, R., et al. (2021). Integrating OPC UA Devices in EPICS. 18th International Conference on Accelerator and Large Experimental Physics Control Systems. doi:[10.18429/JACoW-ICALEPCS2021-MOPV026](https://doi.org/10.1016/j.promfg.2018.02.034)
12. Lee, H.-C. (2017). "DCCN docker swarm cluster documentation."
13. Lee, J., et al. (2015). "A cyber-physical systems architecture for industry 4.0-based manufacturing systems." DOI: https://doi.org/10.1016/j.mfglet.2014.12.001
14. Lu, Y., et al. (2020). "Digital Twin-driven smart manufacturing: Connotation, reference model, applications and research issues." DOI: <https://doi.org/10.1016/j.rcim.2019.101837>
15. Kim et al. (2021). "Process Analytical Technology Tools for Monitoring Pharmaceutical Unit Operations: A Control Strategy for Continuous Process Verification." DOI: <https://doi.org/10.3390/pharmaceutics13060919>
16. Mahnke, W., et al. (2009). OPC unified architecture, Springer Science & Business Media. DOI: [10.1007/978-3-540-68899-0](https://doi.org/10.1016/j.promfg.2018.02.034)
17. Martinez, E. M., et al. (2021). "Automation pyramid as constructor for a complete digital twin, case study: A didactic manufacturing system." DOI:  <https://doi.org/10.3390/s21144656>
18. Morabito, R., et al. (2015). Hypervisors vs. lightweight virtualization: a performance comparison. 2015 IEEE International Conference on cloud engineering, IEEE. doi: [10.1109/IC2E.2015.74](https://doi.org/10.1016/j.promfg.2018.02.034)
19. Narayanan, H., et al. (2020). "Bioprocessing in the digital age: the role of process models."DOI: <https://doi.org/10.1002/biot.201900172>
20. Obe, R. O. and L. S. Hsu (2017). PostgreSQL: up and running: a practical guide to the advanced open source database.
21. Olbort, J., et al. (2022). "Integration of Communication using OPC UA in MBSE for the Development of Cyber-Physical Systems." DOI: <https://doi.org/10.1016/j.procir.2022.05.241>
22. Kadlec, Petr; Gabrys, Bogdan; Strandt, Sibylle (2009): Data-driven Soft Sensors in the process industry. In: Computers & Chemical Engineering. DOI: <https://doi.org/10.1016/j.compchemeng.2008.12.012>
23. Potdar, A. M., et al. (2020). "Performance evaluation of docker container and virtual machine."DOI: <https://doi.org/10.1016/j.procs.2020.04.152>
24. Roulet-Dubonnet, O. (2018). Python OPC-UA Documentation, Dec. pp. 1-9
25. Sipsas, K., et al. (2016). "Collaborative maintenance in flow-line manufacturing environments: An Industry 4.0 approach." DOI: <https://doi.org/10.1016/j.procir.2016.09.013>
26. Stevens, R. E., et al. (2012). The marketing research guide, Routledge. Accessed 2023.05.30
27. Stock, T. and G. Seliger (2016). "Opportunities of sustainable manufacturing in industry 4.0."DOI: <https://doi.org/10.1016/j.procir.2016.01.129>
28. Swales, A. (1999). "Open modbus/tcp specification." Schneider Electric 29(3): 19.
29. Tao, F., et al. (2019). "Digital twins and cyber–physical systems toward smart manufacturing and industry 4.0: Correlation and comparison." DOI: <https://doi.org/10.1016/j.eng.2019.01.014>
30. Thomas, G. (2008). "Introduction to the modbus protocol." The Extension 9(4): 1-4.
31. Tran, R., et al. (2014). "Changing manufacturing paradigms in downstream processing and the role of alternative bioseparation technologies." DOI: [https://doi.org/10.1002/jctb.4234](https://doi.org/10.1016/j.promfg.2018.02.034)
32. Udugama, I. A., et al. (2021). "Digital Twin in biomanufacturing: Challenges and opportunities towards its implementation." DOI: <https://doi.org/10.1007/s43393-021-00024-0>
33. Vaidya, S., et al. (2018). "Industry 4.0–a glimpse." Procedia manufacturing DOI: <https://doi.org/10.1016/j.promfg.2018.02.034>
34. Docker. Docker Guide - Get started. <https://docs.docker.com/get-started/>. Accessed: 2023.05.30
35. OPC UA foundation. <https://opcfoundation.org/>
36. M. Lekić and G. Gardašević, "IoT sensor integration to Node-RED platform," DOI: 10.1109/INFOTEH.2018.8345544
37. Mohamed Tabaa, Brahim Chouri, Safa Saadaoui, Karim Alami. "Industrial Communication based on Modbus and Node-RED" DOI: <https://doi.org/10.1016/j.procs.2018.04.107>
38. Digital Guide IONOS TCP/IP. <https://www.ionos.de/digitalguide/server/knowhow/tcpip-vorgestellt/>.
39. [Tyler Charboneau](https://www.docker.com/author/tyler-charboneaudocker-com/). How to Rapidly Build Multi-Architecture Images with Buildx. <https://www.docker.com/blog/how-to-rapidly-build-multi-architecture-images-with-buildx/>. Accessed: 2023.04.12
40. Olivier Henry (2019): How can chips enable realtime inline monitoring of bioprocesses. Online verfügbar unter <https://www.imec-int.com/en/imec-magazine/imec-magazine-september2019/how-can-chips-enable-realtime-inline-monitoring-of-bioprocesses.>
41. [K Ferencz](https://scholar.google.de/citations?user=glcyilMAAAAJ&hl=de&oi=sra), [J Domokos](https://scholar.google.de/citations?user=EoCruyIAAAAJ&hl=de&oi=sra) (2019). Using Node-Red platform in an Industrial environment.

# Appendix

## Final API code

from logging import exception  
import threading  
import opcua  
import psycopg2  
import time  
import os  
  
# Define OPC-UA connection parameters  
url = "opc.tcp://10.162.80.8:4840"  
  
# Connection to OPC-UA server  
client = opcua.Client(url)  
client.connect()  
  
# List of nodes we want to read  
node\_list\_pre\_process = ['ns=3;s="Lab\_Parameter"."TestName"', 'ns=3;s="Lab\_Parameter"."Operation"', 'ns=3;s="Lab\_Parameter"."pre\_ParticleSizes"',  
 'ns=3;s="Lab\_Parameter"."pre\_ParticleSizeDistribution"', 'ns=3;s="Lab\_Parameter"."AdsorptionModel"',  
 'ns=3;s="Lab\_Parameter"."AdsorptionModelParameters"', 'ns=3;s="Lab\_Parameter"."Initial\_BIONS\_concentration"',  
 'ns=3;s="Lab\_Parameter"."InitialVolume"', 'ns=3;s="Lab\_Parameter"."Param1"', 'ns=3;s="Lab\_Parameter"."Param2"',  
 'ns=3;s="Lab\_Parameter"."Param3"', 'ns=3;s="Lab\_Parameter"."Param4"', 'ns=3;s="Lab\_Parameter"."Param5"']  
node\_list\_main\_steps = ['ns=3;s="Datatostring"."MainStep\_Concat"[0]', 'ns=3;s="Datatostring"."MainStep\_Concat"[1]', 'ns=3;s="Datatostring"."MainStep\_Concat"[2]',  
 'ns=3;s="Datatostring"."MainStep\_Concat"[3]', 'ns=3;s="Datatostring"."MainStep\_Concat"[4]', 'ns=3;s="Datatostring"."MainStep\_Concat"[5]',  
 'ns=3;s="Datatostring"."MainStep\_Concat"[6]', 'ns=3;s="Datatostring"."MainStep\_Concat"[7]', 'ns=3;s="Datatostring"."MainStep\_Concat"[8]',  
 'ns=3;s="Datatostring"."MainStep\_Concat"[9]', 'ns=3;s="Datatostring"."MainStep\_Concat"[10]', 'ns=3;s="Datatostring"."MainStep\_Concat"[11]',  
 'ns=3;s="Datatostring"."MainStep\_Concat"[12]', 'ns=3;s="Datatostring"."MainStep\_Concat"[13]', 'ns=3;s="Datatostring"."MainStep\_Concat"[14]',  
 'ns=3;s="Datatostring"."MainStep\_Concat"[15]', 'ns=3;s="Datatostring"."MainStep\_Concat"[16]', 'ns=3;s="Datatostring"."MainStep\_Concat"[17]',  
 'ns=3;s="Datatostring"."MainStep\_Concat"[18]', 'ns=3;s="Datatostring"."MainStep\_Concat"[19]', 'ns=3;s="Datatostring"."MainStep\_Concat"[20]',  
 'ns=3;s="Datatostring"."MainStep\_Concat"[21]', 'ns=3;s="Datatostring"."MainStep\_Concat"[22]', 'ns=3;s="Datatostring"."MainStep\_Concat"[23]',  
 'ns=3;s="Datatostring"."MainStep\_Concat"[24]', 'ns=3;s="Datatostring"."MainStep\_Concat"[25]', 'ns=3;s="Datatostring"."MainStep\_Concat"[26]',  
 'ns=3;s="Datatostring"."MainStep\_Concat"[27]', 'ns=3;s="Datatostring"."MainStep\_Concat"[28]', 'ns=3;s="Datatostring"."MainStep\_Concat"[29]',  
 'ns=3;s="Datatostring"."MainStep\_Concat"[30]', 'ns=3;s="Datatostring"."MainStep\_Concat"[31]', 'ns=3;s="Lab\_Parameter"."TestName"']  
node\_list\_sub\_steps = ['ns=3;s="Lab\_Parameter"."TestName"', 'ns=3;s="SS\_Datatostring"."SS\_String\_Concat"[0,0]', 'ns=3;s="SS\_Datatostring"."SS\_String\_Concat"[0,1]',  
 'ns=3;s="SS\_Datatostring"."SS\_String\_Concat"[0,2]', 'ns=3;s="SS\_Datatostring"."SS\_String\_Concat"[0,3]', 'ns=3;s="SS\_Datatostring"."SS\_String\_Concat"[0,4]',  
 'ns=3;s="SS\_Datatostring"."SS\_String\_Concat"[0,5]', 'ns=3;s="SS\_Datatostring"."SS\_String\_Concat"[0,6]', 'ns=3;s="SS\_Datatostring"."SS\_String\_Concat"[0,7]',  
 'ns=3;s="SS\_Datatostring"."SS\_String\_Concat"[0,8]', 'ns=3;s="SS\_Datatostring"."SS\_String\_Concat"[0,9]', 'ns=3;s="SS\_Datatostring"."SS\_String\_Concat"[0,10]',  
 'ns=3;s="SS\_Datatostring"."SS\_String\_Concat"[0,11]', 'ns=3;s="SS\_Datatostring"."SS\_String\_Concat"[0,12]', 'ns=3;s="SS\_Datatostring"."SS\_String\_Concat"[0,13]',  
 'ns=3;s="SS\_Datatostring"."SS\_String\_Concat"[0,14]', 'ns=3;s="SS\_Datatostring"."SS\_String\_Concat"[0,15]', 'ns=3;s="SS\_Datatostring"."SS\_String\_Concat"[1,0]',  
 'ns=3;s="SS\_Datatostring"."SS\_String\_Concat"[1,1]', 'ns=3;s="SS\_Datatostring"."SS\_String\_Concat"[1,2]', 'ns=3;s="SS\_Datatostring"."SS\_String\_Concat"[1,3]',  
 'ns=3;s="SS\_Datatostring"."SS\_String\_Concat"[1,4]', 'ns=3;s="SS\_Datatostring"."SS\_String\_Concat"[1,5]', 'ns=3;s="SS\_Datatostring"."SS\_String\_Concat"[1,6]',  
 'ns=3;s="SS\_Datatostring"."SS\_String\_Concat"[1,7]', 'ns=3;s="SS\_Datatostring"."SS\_String\_Concat"[1,8]', 'ns=3;s="SS\_Datatostring"."SS\_String\_Concat"[1,9]',  
 'ns=3;s="SS\_Datatostring"."SS\_String\_Concat"[1,10]', 'ns=3;s="SS\_Datatostring"."SS\_String\_Concat"[1,1]', 'ns=3;s="SS\_Datatostring"."SS\_String\_Concat"[1,12]',  
 'ns=3;s="SS\_Datatostring"."SS\_String\_Concat"[1,13]', 'ns=3;s="SS\_Datatostring"."SS\_String\_Concat"[1,14]', 'ns=3;s="SS\_Datatostring"."SS\_String\_Concat"[1,15]',  
 'ns=3;s="SS\_Datatostring"."SS\_String\_Concat"[2,0]', 'ns=3;s="SS\_Datatostring"."SS\_String\_Concat"[2,1]', 'ns=3;s="SS\_Datatostring"."SS\_String\_Concat"[2,2]',  
 'ns=3;s="SS\_Datatostring"."SS\_String\_Concat"[2,3]', 'ns=3;s="SS\_Datatostring"."SS\_String\_Concat"[2,4]', 'ns=3;s="SS\_Datatostring"."SS\_String\_Concat"[2,5]',  
 'ns=3;s="SS\_Datatostring"."SS\_String\_Concat"[2,6]', 'ns=3;s="SS\_Datatostring"."SS\_String\_Concat"[2,7]', 'ns=3;s="SS\_Datatostring"."SS\_String\_Concat"[2,8]',  
 'ns=3;s="SS\_Datatostring"."SS\_String\_Concat"[2,9]', 'ns=3;s="SS\_Datatostring"."SS\_String\_Concat"[2,10]', 'ns=3;s="SS\_Datatostring"."SS\_String\_Concat"[2,11]',  
 'ns=3;s="SS\_Datatostring"."SS\_String\_Concat"[2,12]', 'ns=3;s="SS\_Datatostring"."SS\_String\_Concat"[2,13]', 'ns=3;s="SS\_Datatostring"."SS\_String\_Concat"[2,14]',  
 'ns=3;s="SS\_Datatostring"."SS\_String\_Concat"[2,15]', 'ns=3;s="SS\_Datatostring"."SS\_String\_Concat"[3,0]', 'ns=3;s="SS\_Datatostring"."SS\_String\_Concat"[3,1]',  
 'ns=3;s="SS\_Datatostring"."SS\_String\_Concat"[3,2]', 'ns=3;s="SS\_Datatostring"."SS\_String\_Concat"[3,3]', 'ns=3;s="SS\_Datatostring"."SS\_String\_Concat"[3,4]',  
 'ns=3;s="SS\_Datatostring"."SS\_String\_Concat"[3,5]', 'ns=3;s="SS\_Datatostring"."SS\_String\_Concat"[3,6]', 'ns=3;s="SS\_Datatostring"."SS\_String\_Concat"[3,7]',  
 'ns=3;s="SS\_Datatostring"."SS\_String\_Concat"[3,8]', 'ns=3;s="SS\_Datatostring"."SS\_String\_Concat"[3,9]', 'ns=3;s="SS\_Datatostring"."SS\_String\_Concat"[3,10]',  
 'ns=3;s="SS\_Datatostring"."SS\_String\_Concat"[3,11]', 'ns=3;s="SS\_Datatostring"."SS\_String\_Concat"[3,12]', 'ns=3;s="SS\_Datatostring"."SS\_String\_Concat"[3,13]',  
 'ns=3;s="SS\_Datatostring"."SS\_String\_Concat"[3,14]', 'ns=3;s="SS\_Datatostring"."SS\_String\_Concat"[3,15]', 'ns=3;s="SS\_Datatostring"."SS\_String\_Concat"[4,0]',  
 'ns=3;s="SS\_Datatostring"."SS\_String\_Concat"[4,1]', 'ns=3;s="SS\_Datatostring"."SS\_String\_Concat"[4,2]', 'ns=3;s="SS\_Datatostring"."SS\_String\_Concat"[4,3]',  
 'ns=3;s="SS\_Datatostring"."SS\_String\_Concat"[4,4]', 'ns=3;s="SS\_Datatostring"."SS\_String\_Concat"[4,5]', 'ns=3;s="SS\_Datatostring"."SS\_String\_Concat"[4,6]',  
 'ns=3;s="SS\_Datatostring"."SS\_String\_Concat"[4,7]', 'ns=3;s="SS\_Datatostring"."SS\_String\_Concat"[4,8]', 'ns=3;s="SS\_Datatostring"."SS\_String\_Concat"[4,9]',  
 'ns=3;s="SS\_Datatostring"."SS\_String\_Concat"[4,10]', 'ns=3;s="SS\_Datatostring"."SS\_String\_Concat"[4,11]', 'ns=3;s="SS\_Datatostring"."SS\_String\_Concat"[4,12]',  
 'ns=3;s="SS\_Datatostring"."SS\_String\_Concat"[4,13]', 'ns=3;s="SS\_Datatostring"."SS\_String\_Concat"[4,14]', 'ns=3;s="SS\_Datatostring"."SS\_String\_Concat"[4,15]',  
 'ns=3;s="SS\_Datatostring"."SS\_String\_Concat"[5,0]', 'ns=3;s="SS\_Datatostring"."SS\_String\_Concat"[5,1]', 'ns=3;s="SS\_Datatostring"."SS\_String\_Concat"[5,2]',  
 'ns=3;s="SS\_Datatostring"."SS\_String\_Concat"[5,3]', 'ns=3;s="SS\_Datatostring"."SS\_String\_Concat"[5,4]', 'ns=3;s="SS\_Datatostring"."SS\_String\_Concat"[5,5]',  
 'ns=3;s="SS\_Datatostring"."SS\_String\_Concat"[5,6]', 'ns=3;s="SS\_Datatostring"."SS\_String\_Concat"[5,7]', 'ns=3;s="SS\_Datatostring"."SS\_String\_Concat"[5,8]',  
 'ns=3;s="SS\_Datatostring"."SS\_String\_Concat"[5,9]', 'ns=3;s="SS\_Datatostring"."SS\_String\_Concat"[5,10]', 'ns=3;s="SS\_Datatostring"."SS\_String\_Concat"[5,11]',  
 'ns=3;s="SS\_Datatostring"."SS\_String\_Concat"[5,12]', 'ns=3;s="SS\_Datatostring"."SS\_String\_Concat"[5,13]', 'ns=3;s="SS\_Datatostring"."SS\_String\_Concat"[5,14]',  
 'ns=3;s="SS\_Datatostring"."SS\_String\_Concat"[5,15]', 'ns=3;s="SS\_Datatostring"."SS\_String\_Concat"[6,0]', 'ns=3;s="SS\_Datatostring"."SS\_String\_Concat"[6,1]',  
 'ns=3;s="SS\_Datatostring"."SS\_String\_Concat"[6,2]', 'ns=3;s="SS\_Datatostring"."SS\_String\_Concat"[6,3]', 'ns=3;s="SS\_Datatostring"."SS\_String\_Concat"[6,4]',  
 'ns=3;s="SS\_Datatostring"."SS\_String\_Concat"[6,5]', 'ns=3;s="SS\_Datatostring"."SS\_String\_Concat"[6,6]', 'ns=3;s="SS\_Datatostring"."SS\_String\_Concat"[6,7]',  
 'ns=3;s="SS\_Datatostring"."SS\_String\_Concat"[6,8]', 'ns=3;s="SS\_Datatostring"."SS\_String\_Concat"[6,9]', 'ns=3;s="SS\_Datatostring"."SS\_String\_Concat"[6,10]',  
 'ns=3;s="SS\_Datatostring"."SS\_String\_Concat"[6,11]', 'ns=3;s="SS\_Datatostring"."SS\_String\_Concat"[6,12]', 'ns=3;s="SS\_Datatostring"."SS\_String\_Concat"[6,13]',  
 'ns=3;s="SS\_Datatostring"."SS\_String\_Concat"[6,14]', 'ns=3;s="SS\_Datatostring"."SS\_String\_Concat"[6,15]', 'ns=3;s="SS\_Datatostring"."SS\_String\_Concat"[7,0]',  
 'ns=3;s="SS\_Datatostring"."SS\_String\_Concat"[7,1]', 'ns=3;s="SS\_Datatostring"."SS\_String\_Concat"[7,2]', 'ns=3;s="SS\_Datatostring"."SS\_String\_Concat"[7,3]',  
 'ns=3;s="SS\_Datatostring"."SS\_String\_Concat"[7,4]', 'ns=3;s="SS\_Datatostring"."SS\_String\_Concat"[7,5]', 'ns=3;s="SS\_Datatostring"."SS\_String\_Concat"[7,6]',  
 'ns=3;s="SS\_Datatostring"."SS\_String\_Concat"[7,7]', 'ns=3;s="SS\_Datatostring"."SS\_String\_Concat"[7,8]', 'ns=3;s="SS\_Datatostring"."SS\_String\_Concat"[7,9]',  
 'ns=3;s="SS\_Datatostring"."SS\_String\_Concat"[7,10]', 'ns=3;s="SS\_Datatostring"."SS\_String\_Concat"[7,11]', 'ns=3;s="SS\_Datatostring"."SS\_String\_Concat"[7,12]',  
 'ns=3;s="SS\_Datatostring"."SS\_String\_Concat"[7,13]', 'ns=3;s="SS\_Datatostring"."SS\_String\_Concat"[7,14]', 'ns=3;s="SS\_Datatostring"."SS\_String\_Concat"[7,15]',  
 'ns=3;s="SS\_Datatostring"."SS\_String\_Concat"[8,0]', 'ns=3;s="SS\_Datatostring"."SS\_String\_Concat"[8,1]', 'ns=3;s="SS\_Datatostring"."SS\_String\_Concat"[8,2]',  
 'ns=3;s="SS\_Datatostring"."SS\_String\_Concat"[8,3]', 'ns=3;s="SS\_Datatostring"."SS\_String\_Concat"[8,4]', 'ns=3;s="SS\_Datatostring"."SS\_String\_Concat"[8,5]',  
 'ns=3;s="SS\_Datatostring"."SS\_String\_Concat"[8,6]', 'ns=3;s="SS\_Datatostring"."SS\_String\_Concat"[8,7]', 'ns=3;s="SS\_Datatostring"."SS\_String\_Concat"[8,8]',  
 'ns=3;s="SS\_Datatostring"."SS\_String\_Concat"[8,9]', 'ns=3;s="SS\_Datatostring"."SS\_String\_Concat"[8,10]', 'ns=3;s="SS\_Datatostring"."SS\_String\_Concat"[8,11]',  
 'ns=3;s="SS\_Datatostring"."SS\_String\_Concat"[8,12]', 'ns=3;s="SS\_Datatostring"."SS\_String\_Concat"[8,13]', 'ns=3;s="SS\_Datatostring"."SS\_String\_Concat"[8,14]',  
 'ns=3;s="SS\_Datatostring"."SS\_String\_Concat"[8,15]', 'ns=3;s="SS\_Datatostring"."SS\_String\_Concat"[9,0]', 'ns=3;s="SS\_Datatostring"."SS\_String\_Concat"[9,1]',  
 'ns=3;s="SS\_Datatostring"."SS\_String\_Concat"[9,2]', 'ns=3;s="SS\_Datatostring"."SS\_String\_Concat"[9,3]', 'ns=3;s="SS\_Datatostring"."SS\_String\_Concat"[9,4]',  
 'ns=3;s="SS\_Datatostring"."SS\_String\_Concat"[9,5]', 'ns=3;s="SS\_Datatostring"."SS\_String\_Concat"[9,6]', 'ns=3;s="SS\_Datatostring"."SS\_String\_Concat"[9,7]',  
 'ns=3;s="SS\_Datatostring"."SS\_String\_Concat"[9,8]', 'ns=3;s="SS\_Datatostring"."SS\_String\_Concat"[9,9]', 'ns=3;s="SS\_Datatostring"."SS\_String\_Concat"[9,10]',  
 'ns=3;s="SS\_Datatostring"."SS\_String\_Concat"[9,11]', 'ns=3;s="SS\_Datatostring"."SS\_String\_Concat"[9,12]', 'ns=3;s="SS\_Datatostring"."SS\_String\_Concat"[9,13]',  
 'ns=3;s="SS\_Datatostring"."SS\_String\_Concat"[9,14]', 'ns=3;s="SS\_Datatostring"."SS\_String\_Concat"[9,15]', 'ns=3;s="SS\_Datatostring"."SS\_String\_Concat"[10,0]',  
 'ns=3;s="SS\_Datatostring"."SS\_String\_Concat"[10,1]', 'ns=3;s="SS\_Datatostring"."SS\_String\_Concat"[10,2]', 'ns=3;s="SS\_Datatostring"."SS\_String\_Concat"[10,3]',  
 'ns=3;s="SS\_Datatostring"."SS\_String\_Concat"[10,4]', 'ns=3;s="SS\_Datatostring"."SS\_String\_Concat"[10,5]', 'ns=3;s="SS\_Datatostring"."SS\_String\_Concat"[10,6]',  
 'ns=3;s="SS\_Datatostring"."SS\_String\_Concat"[10,7]', 'ns=3;s="SS\_Datatostring"."SS\_String\_Concat"[10,8]', 'ns=3;s="SS\_Datatostring"."SS\_String\_Concat"[10,9]',  
 'ns=3;s="SS\_Datatostring"."SS\_String\_Concat"[10,10]', 'ns=3;s="SS\_Datatostring"."SS\_String\_Concat"[10,11]', 'ns=3;s="SS\_Datatostring"."SS\_String\_Concat"[10,12]',  
 'ns=3;s="SS\_Datatostring"."SS\_String\_Concat"[10,13]', 'ns=3;s="SS\_Datatostring"."SS\_String\_Concat"[10,14]', 'ns=3;s="SS\_Datatostring"."SS\_String\_Concat"[10,15]',  
 'ns=3;s="SS\_Datatostring"."SS\_String\_Concat"[11,0]', 'ns=3;s="SS\_Datatostring"."SS\_String\_Concat"[11,1]', 'ns=3;s="SS\_Datatostring"."SS\_String\_Concat"[11,2]',  
 'ns=3;s="SS\_Datatostring"."SS\_String\_Concat"[11,3]', 'ns=3;s="SS\_Datatostring"."SS\_String\_Concat"[11,4]', 'ns=3;s="SS\_Datatostring"."SS\_String\_Concat"[11,5]',  
 'ns=3;s="SS\_Datatostring"."SS\_String\_Concat"[11,6]', 'ns=3;s="SS\_Datatostring"."SS\_String\_Concat"[11,7]', 'ns=3;s="SS\_Datatostring"."SS\_String\_Concat"[11,8]',  
 'ns=3;s="SS\_Datatostring"."SS\_String\_Concat"[11,9]', 'ns=3;s="SS\_Datatostring"."SS\_String\_Concat"[11,10]', 'ns=3;s="SS\_Datatostring"."SS\_String\_Concat"[11,11]',  
 'ns=3;s="SS\_Datatostring"."SS\_String\_Concat"[11,12]', 'ns=3;s="SS\_Datatostring"."SS\_String\_Concat"[11,13]', 'ns=3;s="SS\_Datatostring"."SS\_String\_Concat"[11,14]',  
 'ns=3;s="SS\_Datatostring"."SS\_String\_Concat"[11,15]']  
node\_list\_post\_process = ['ns=3;s="Lab\_Parameter"."TestName2"', 'ns=3;s="Lab\_Parameter"."post\_ParticleSizes"',  
 'ns=3;s="Lab\_Parameter"."post\_ParticleSizeDistribution"', 'ns=3;s="Lab\_Parameter"."Yield"',  
 'ns=3;s="Lab\_Parameter"."Recovery"', 'ns=3;s="Lab\_Parameter"."PurificationFactor"', 'ns=3;s="Lab\_Parameter"."Param6"',  
 'ns=3;s="Lab\_Parameter"."Param7"', 'ns=3;s="Lab\_Parameter"."Param8"', 'ns=3;s="Lab\_Parameter"."Param9"',  
 'ns=3;s="Lab\_Parameter"."Param10"', 'ns=3;s="Lab\_Parameter"."Param11"', 'ns=3;s="Lab\_Parameter"."Param12"']  
node\_list\_online\_data = ['ns=3;s="Lab\_Parameter"."TestName3"', 'ns=3;s="RSG45\_DATA"."AO\_MagnetTemperature"."rAnalogSignalValue"',  
 'ns=3;s="RSG45\_DATA"."AO\_MixerIstFreq"."rAnalogSignalValue"', 'ns=3;s="RSG45\_DATA".AO\_PumpIstFre.rAnalogSignalValue',  
 'ns=3;s="FluidTemperatur"', 'ns=3;s="Conductivity"', 'ns=3;s="PH"', 'ns=3;s="Density"', 'ns=3;s="VolumeFlow"',  
 'ns=3;s="VolumeFlowAnalog"', 'ns=3;s="TempCoriolis"', 'ns=3;s="MassFlow"', 'ns=3;s="Photometer"', 'ns=3;s="PID\_Setpoint"',  
 'ns=3;s="ValveCtrl".aValves[0].bMsgIsOpen', 'ns=3;s="ValveCtrl".aValves[1].bMsgIsOpen', 'ns=3;s="ValveCtrl".aValves[2].bMsgIsOpen',  
 'ns=3;s="ValveCtrl".aValves[3].bMsgIsOpen', 'ns=3;s="ValveCtrl".aValves[4].bMsgIsOpen', 'ns=3;s="ValveCtrl".aValves[5].bMsgIsOpen',  
 'ns=3;s="ValveCtrl".aValves[7].bMsgIsOpen', 'ns=3;s="ValveCtrl".aValves[8].bMsgIsOpen', 'ns=3;s="ValveCtrl".aValves[9].bMsgIsOpen',  
 'ns=3;s="ValveCtrl".aValves[10].bMsgIsOpen', 'ns=3;s="ValveCtrl".aValves[11].bMsgIsOpen', 'ns=3;s="ValveCtrl".aValves[12].bMsgIsOpen',  
 'ns=3;s="DIMagnetIsOn"', 'ns=3;s="LP\_Data".VISU.StepNames.sMainStep.sValue', 'ns=3;s="LP\_Data".VISU.StepNames.sSubStep.sValue',  
 'ns=3;s="LP\_Data".VISU.StepTime.iActStepTime', 'ns=3;s="LP\_Data".VISU.StepTime.iSetSteptime', 'ns=3;s="Counter"."Output Counter"',  
 'ns=3;s="Lab\_Parameter"."AutoTransferOnOff"', 'ns=3;s="Lab\_Parameter"."ContTransferOnOff"', 'ns=3;s="ParameterProcess"."uWorkingSet"."uSubStepParams"[12]."Params"[4]."iPumpSpeed"',  
 'ns=3;s="VolumeFlowNew"', 'ns=3;s="DensityNew"', 'ns=3;s="MassFlowNew"', 'ns=3;s="TemperatureNew"']  
  
# Connection to PostgreSQL database  
DB\_NAME = os.environ.get("DB\_NAME", "postgres") # Default to "postgres" if not provided  
DB\_PORT = int(os.environ.get("DB\_PORT", 5432)) # Default to 5432 if not provided  
DB\_USER = os.environ.get("DB\_USER", "postgres") # Default to "postgres" if not provided  
DB\_PASSWORD = os.environ.get("DB\_PASSWORD", "inseldip2023") # Default password if not provided  
DB\_HOST = os.environ.get("DB\_HOST", "172.20.0.35") # Default host if not provided  
  
conn = psycopg2.connect(dbname=DB\_NAME, port=DB\_PORT, user=DB\_USER, password=DB\_PASSWORD, host=DB\_HOST)  
  
def collect\_data():  
 while True:  
 values\_online\_data = [client.get\_node(  
 node).get\_value() for node in node\_list\_online\_data]  
 cur = conn.cursor()  
 cur.execute('''INSERT INTO online\_data1 (testid3, magnettemp, rotorfreq, pumpfreq, fluidtemp, conductivity, ph, density,   
 voumeflow, volumeflowanalog, tempCoriolis, massflow, absorbance, volumeflowsetpoint, valve1, valve2,   
 valve3, valve4, valve5, valve6, valve8, valve9, valve10, valve11, valve12, valve13, magnetstatus, mainstepname,   
 substepname, actsubsteptime, setsubsteptime, totalprocesstime, manualtransferonoff, conttransferonoff, recipe\_vf,   
 volumeFlowNew, densityNew, massFlowNew, temperatureNew) VALUES   
 (%s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s,   
 %s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s)''',  
 (values\_online\_data[0], values\_online\_data[1], values\_online\_data[2], values\_online\_data[3], values\_online\_data[4], values\_online\_data[5],  
 values\_online\_data[6], values\_online\_data[7], values\_online\_data[  
 8], values\_online\_data[9], values\_online\_data[10], values\_online\_data[11],  
 values\_online\_data[12], values\_online\_data[13], values\_online\_data[  
 14], values\_online\_data[15], values\_online\_data[16], values\_online\_data[17],  
 values\_online\_data[18], values\_online\_data[19], values\_online\_data[  
 20], values\_online\_data[21], values\_online\_data[22], values\_online\_data[23],  
 values\_online\_data[24], values\_online\_data[25], values\_online\_data[  
 26], values\_online\_data[27], values\_online\_data[28], values\_online\_data[29],  
 values\_online\_data[30], values\_online\_data[31], values\_online\_data[  
 32], values\_online\_data[33], values\_online\_data[34], values\_online\_data[35],  
 values\_online\_data[36], values\_online\_data[37], values\_online\_data[38]))  
 conn.commit()  
 cur.close()  
 time.sleep(10)  
# except exception as e:  
 # print(f"error occured: ")  
  
  
data\_collection\_thread = threading.Thread(target=collect\_data)  
data\_collection\_thread.daemon = True  
data\_collection\_thread.start()  
  
# while True:  
# Read node values\_sub\_steps  
values\_pre\_process = [client.get\_node(node).get\_value() for node in node\_list\_pre\_process]  
values\_main\_steps = [client.get\_node(node).get\_value() for node in node\_list\_main\_steps]  
values\_sub\_steps = [client.get\_node(node).get\_value() for node in node\_list\_sub\_steps]  
values\_post\_process = [client.get\_node(node).get\_value() for node in node\_list\_post\_process]  
  
if ('"Lab\_Parameter"."Recipe\_Reset"' != 'Reset\_post') and ('"Lab\_Parameter"."Recipe\_Reset"' != 'Reset\_steps') and ('"Lab\_Parameter"."Recipe\_Reset"' != 'Reset'):  
 # Change values if necessary  
 for i in range(len(values\_pre\_process)):  
 if node\_list\_pre\_process[i] == 'ns=3;s="Lab\_Parameter"."Operation"':  
 if values\_pre\_process[i] == 1:  
 values\_pre\_process[i] = 'Washing BIONs'  
 elif values\_pre\_process[i] == 2:  
 values\_pre\_process[i] = 'Protein purification'  
 elif values\_pre\_process[i] == 4:  
 values\_pre\_process[i] = 'Test'  
 elif "AdsorptionModel" in node\_list\_pre\_process[i]:  
 if values\_pre\_process[i] == 1:  
 values\_pre\_process[i] = "Langmuir"  
 elif values\_pre\_process[i] == 2:  
 values\_pre\_process[i] = "Freundlich"  
 elif values\_pre\_process[i] == 4:  
 values\_pre\_process[i] = "test"  
# try:  
 cur = conn.cursor()  
 cur.execute('''INSERT INTO pre\_process1(testid, operation, particle\_sizes, particle\_size\_distribution, absorption\_model,  
 absorption\_model\_params, "initial\_BIONs\_concentration", initial\_volume, param1, param2, param3, param4, param5)  
 VALUES (%s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s)''',  
 (values\_pre\_process[0], values\_pre\_process[1], values\_pre\_process[2], values\_pre\_process[3],  
 values\_pre\_process[4], values\_pre\_process[5], values\_pre\_process[6], values\_pre\_process[7], values\_pre\_process[8],  
 values\_pre\_process[9], values\_pre\_process[10], values\_pre\_process[11], values\_pre\_process[12]))  
 conn.commit()  
 cur.close()  
  
# Writing values to main\_steps table  
 cur = conn.cursor()  
 cur.execute('''INSERT INTO ms (testid, ms0, ms1, ms2, ms3, ms4, ms5, ms6, ms7, ms8, ms9, ms10, ms11, ms12, ms13, ms14, ms15, ms16,   
 ms17, ms18, ms19, ms20, ms21, ms22, ms23, ms24, ms25, ms26, ms27, ms28, ms29, ms30, ms31)  
 VALUES (%s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s,   
 %s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s)''',  
 (values\_main\_steps[0], values\_main\_steps[1], values\_main\_steps[2], values\_main\_steps[3], values\_main\_steps[4], values\_main\_steps[5], values\_main\_steps[6], values\_main\_steps[7], values\_main\_steps[8], values\_main\_steps[9],  
 values\_main\_steps[10], values\_main\_steps[11], values\_main\_steps[12], values\_main\_steps[13], values\_main\_steps[  
 14], values\_main\_steps[15], values\_main\_steps[16], values\_main\_steps[17], values\_main\_steps[18],  
 values\_main\_steps[19], values\_main\_steps[20], values\_main\_steps[21], values\_main\_steps[22], values\_main\_steps[  
 23], values\_main\_steps[24], values\_main\_steps[25], values\_main\_steps[26], values\_main\_steps[27],  
 values\_main\_steps[28], values\_main\_steps[29], values\_main\_steps[30], values\_main\_steps[31], values\_main\_steps[32]))  
 conn.commit()  
 cur.close()  
  
  
# Writing values into sub\_steps table  
 cur = conn.cursor()  
 cur.execute('''INSERT INTO ss (testid,ss0,ss1,ss2,ss3,ss4,ss5,ss6,ss7,ss8,ss9,ss10,ss11,ss12,ss13,ss14,ss15,ss16,ss17,ss18,ss19,ss20,  
 ss21,ss22,ss23,ss24,ss25,ss26,ss27,ss28,ss29,ss30,ss31,ss32,ss33,ss34,ss35,ss36,ss37,ss38,ss39,ss40,ss41,ss42,ss43,ss44,ss45,ss46,ss47,ss48,  
 ss49,ss50,ss51,ss52,ss53,ss54,ss55,ss56,ss57,ss58,ss59,ss60,ss61,ss62,ss63,ss64,ss65,ss66,ss67,ss68,ss69,ss70,ss71,ss72,ss73,ss74,ss75,ss76,  
 ss77,ss78,ss79,ss80,ss81,ss82,ss83,ss84,ss85,ss86,ss87,ss88,ss89,ss90,ss91,ss92,ss93,ss94,ss95,ss96,ss97,ss98,ss99,ss100,ss101,ss102,ss103,  
 ss104,ss105,ss106,ss107,ss108,ss109,ss110,ss111,ss112,ss113,ss114,ss115,ss116,ss117,ss118,ss119,ss120,ss121,ss122,ss123,ss124,ss125,ss126,  
 ss127,ss128,ss129,ss130,ss131,ss132,ss133,ss134,ss135,ss136,ss137,ss138,ss139,ss140,ss141,ss142,ss143,ss144,ss145,ss146,ss147,ss148,ss149,  
 ss150,ss151,ss152,ss153,ss154,ss155,ss156,ss157,ss158,ss159,ss160,ss161,ss162,ss163,ss164,ss165,ss166,ss167,ss168,ss169,ss170,ss171,ss172,  
 ss173,ss174,ss175,ss176,ss177,ss178,ss179,ss180,ss181,ss182,ss183,ss184,ss185,ss186,ss187,ss188,ss189,ss190,ss191)  
 VALUES (%s,%s,%s,%s,%s,%s,%s,%s,%s,%s,%s,%s,%s,%s,%s,%s,%s,%s,%s,%s,%s,%s,%s,%s,%s,%s,%s,%s,%s,%s,%s,%s,  
 %s,%s,%s,%s,%s,%s,%s,%s,%s,%s,%s,%s,%s,%s,%s,%s,%s,%s,%s,%s,%s,%s,%s,%s,%s,%s,%s,%s,%s,%s,%s,%s,   
 %s,%s,%s,%s,%s,%s,%s,%s,%s,%s,%s,%s,%s,%s,%s,%s,%s,%s,%s,%s,%s,%s,%s,%s,%s,%s,%s,%s,%s,%s,%s,%s,  
 %s,%s,%s,%s,%s,%s,%s,%s,%s,%s,%s,%s,%s,%s,%s,%s,%s,%s,%s,%s,%s,%s,%s,%s,%s,%s,%s,%s,%s,%s,%s,%s,  
 %s,%s,%s,%s,%s,%s,%s,%s,%s,%s,%s,%s,%s,%s,%s,%s,%s,%s,%s,%s,%s,%s,%s,%s,%s,%s,%s,%s,%s,%s,%s,%s,  
 %s,%s,%s,%s,%s,%s,%s,%s,%s,%s,%s,%s,%s,%s,%s,%s,%s,%s,%s,%s,%s,%s,%s,%s,%s,%s,%s,%s,%s,%s,%s,%s,%s  
 )''',  
 (values\_sub\_steps[0], values\_sub\_steps[1], values\_sub\_steps[2], values\_sub\_steps[3], values\_sub\_steps[4], values\_sub\_steps[5], values\_sub\_steps[6], values\_sub\_steps[7], values\_sub\_steps[8], values\_sub\_steps[9],  
 values\_sub\_steps[10], values\_sub\_steps[11], values\_sub\_steps[12], values\_sub\_steps[13], values\_sub\_steps[  
 14], values\_sub\_steps[15], values\_sub\_steps[16], values\_sub\_steps[17], values\_sub\_steps[18], values\_sub\_steps[19],  
 values\_sub\_steps[20], values\_sub\_steps[21], values\_sub\_steps[22], values\_sub\_steps[23], values\_sub\_steps[  
 24], values\_sub\_steps[25], values\_sub\_steps[26], values\_sub\_steps[27], values\_sub\_steps[28], values\_sub\_steps[29],  
 values\_sub\_steps[30], values\_sub\_steps[31], values\_sub\_steps[32], values\_sub\_steps[33], values\_sub\_steps[  
 34], values\_sub\_steps[35], values\_sub\_steps[36], values\_sub\_steps[37], values\_sub\_steps[38], values\_sub\_steps[39],  
 values\_sub\_steps[40], values\_sub\_steps[41], values\_sub\_steps[42], values\_sub\_steps[43], values\_sub\_steps[  
 44], values\_sub\_steps[45], values\_sub\_steps[46], values\_sub\_steps[47], values\_sub\_steps[48], values\_sub\_steps[49],  
 values\_sub\_steps[50], values\_sub\_steps[51], values\_sub\_steps[52], values\_sub\_steps[53], values\_sub\_steps[  
 54], values\_sub\_steps[55], values\_sub\_steps[56], values\_sub\_steps[57], values\_sub\_steps[58], values\_sub\_steps[59],  
 values\_sub\_steps[60], values\_sub\_steps[61], values\_sub\_steps[62], values\_sub\_steps[63], values\_sub\_steps[  
 64], values\_sub\_steps[65], values\_sub\_steps[66], values\_sub\_steps[67], values\_sub\_steps[68], values\_sub\_steps[69],  
 values\_sub\_steps[70], values\_sub\_steps[71], values\_sub\_steps[72], values\_sub\_steps[73], values\_sub\_steps[  
 74], values\_sub\_steps[75], values\_sub\_steps[76], values\_sub\_steps[77], values\_sub\_steps[78], values\_sub\_steps[79],  
 values\_sub\_steps[80], values\_sub\_steps[81], values\_sub\_steps[82], values\_sub\_steps[83], values\_sub\_steps[  
 84], values\_sub\_steps[85], values\_sub\_steps[86], values\_sub\_steps[87], values\_sub\_steps[88], values\_sub\_steps[89],  
 values\_sub\_steps[90], values\_sub\_steps[91], values\_sub\_steps[92], values\_sub\_steps[93], values\_sub\_steps[  
 94], values\_sub\_steps[95], values\_sub\_steps[96], values\_sub\_steps[97], values\_sub\_steps[98], values\_sub\_steps[99],  
 values\_sub\_steps[100], values\_sub\_steps[101], values\_sub\_steps[102], values\_sub\_steps[103], values\_sub\_steps[  
 104], values\_sub\_steps[105], values\_sub\_steps[106], values\_sub\_steps[107], values\_sub\_steps[108], values\_sub\_steps[109],  
 values\_sub\_steps[110], values\_sub\_steps[111], values\_sub\_steps[112], values\_sub\_steps[113], values\_sub\_steps[  
 114], values\_sub\_steps[115], values\_sub\_steps[116], values\_sub\_steps[117], values\_sub\_steps[118], values\_sub\_steps[119],  
 values\_sub\_steps[120], values\_sub\_steps[121], values\_sub\_steps[122], values\_sub\_steps[123], values\_sub\_steps[  
 124], values\_sub\_steps[125], values\_sub\_steps[126], values\_sub\_steps[127], values\_sub\_steps[128], values\_sub\_steps[129],  
 values\_sub\_steps[130], values\_sub\_steps[131], values\_sub\_steps[132], values\_sub\_steps[133], values\_sub\_steps[  
 134], values\_sub\_steps[135], values\_sub\_steps[136], values\_sub\_steps[137], values\_sub\_steps[138], values\_sub\_steps[139],  
 values\_sub\_steps[140], values\_sub\_steps[141], values\_sub\_steps[142], values\_sub\_steps[143], values\_sub\_steps[  
 144], values\_sub\_steps[145], values\_sub\_steps[146], values\_sub\_steps[147], values\_sub\_steps[148], values\_sub\_steps[149],  
 values\_sub\_steps[150], values\_sub\_steps[151], values\_sub\_steps[152], values\_sub\_steps[153], values\_sub\_steps[  
 154], values\_sub\_steps[155], values\_sub\_steps[156], values\_sub\_steps[157], values\_sub\_steps[158], values\_sub\_steps[159],  
 values\_sub\_steps[160], values\_sub\_steps[161], values\_sub\_steps[162], values\_sub\_steps[163], values\_sub\_steps[  
 164], values\_sub\_steps[165], values\_sub\_steps[166], values\_sub\_steps[167], values\_sub\_steps[168], values\_sub\_steps[169],  
 values\_sub\_steps[170], values\_sub\_steps[171], values\_sub\_steps[172], values\_sub\_steps[173], values\_sub\_steps[  
 174], values\_sub\_steps[175], values\_sub\_steps[176], values\_sub\_steps[177], values\_sub\_steps[178], values\_sub\_steps[179],  
 values\_sub\_steps[180], values\_sub\_steps[181], values\_sub\_steps[182], values\_sub\_steps[183], values\_sub\_steps[  
 184], values\_sub\_steps[185], values\_sub\_steps[186], values\_sub\_steps[187], values\_sub\_steps[188], values\_sub\_steps[189],  
 values\_sub\_steps[190], values\_sub\_steps[191], values\_sub\_steps[192]))  
 conn.commit()  
 cur.close()  
  
  
elif ('"Lab\_Parameter"."Recipe\_Reset"' == 'Reset\_post'):  
 cur = conn.cursor()  
 cur.execute('''INSERT INTO post\_process(testid, particle\_sizes, particle\_size\_distribution, yield,recovery,  
 purification\_factor, param6, param7, param8, param9, param10, param11, param12)  
 VALUES (%s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s)''',  
 (values\_post\_process[0], values\_post\_process[1], values\_post\_process[2], values\_post\_process[3],  
 values\_post\_process[4], values\_post\_process[5], values\_post\_process[6], values\_post\_process[7], values\_post\_process[8],  
 values\_post\_process[9], values\_post\_process[10], values\_post\_process[11], values\_post\_process[12]))  
 conn.commit()  
 cur.close()

## Latency test code

import opcua  
import psycopg2  
import time  
import numpy as np  
import matplotlib.pyplot as plt  
# Define OPC-UA connection parameters  
url = "opc.tcp://10.162.80.8:4840"  
  
# Connection to OPC-UA server  
client = opcua.Client(url)  
client.connect()  
  
  
# List of nodes we want to read  
node\_list\_online\_data = ['ns=3;s="Lab\_Parameter"."TestName3"', 'ns=3;s="RSG45\_DATA"."AO\_MagnetTemperature"."rAnalogSignalValue"',   
 'ns=3;s="RSG45\_DATA"."AO\_MixerIstFreq"."rAnalogSignalValue"', 'ns=3;s="RSG45\_DATA".AO\_PumpIstFre.rAnalogSignalValue',   
 'ns=3;s="FluidTemperatur"', 'ns=3;s="Conductivity"', 'ns=3;s="PH"', 'ns=3;s="Density"', 'ns=3;s="VolumeFlow"',   
 'ns=3;s="VolumeFlowAnalog"', 'ns=3;s="TempCoriolis"', 'ns=3;s="MassFlow"', 'ns=3;s="Photometer"', 'ns=3;s="PID\_Setpoint"',   
 'ns=3;s="ValveCtrl".aValves[0].bMsgIsOpen', 'ns=3;s="ValveCtrl".aValves[1].bMsgIsOpen', 'ns=3;s="ValveCtrl".aValves[2].bMsgIsOpen',   
 'ns=3;s="ValveCtrl".aValves[3].bMsgIsOpen', 'ns=3;s="ValveCtrl".aValves[4].bMsgIsOpen', 'ns=3;s="ValveCtrl".aValves[5].bMsgIsOpen',   
 'ns=3;s="ValveCtrl".aValves[7].bMsgIsOpen', 'ns=3;s="ValveCtrl".aValves[8].bMsgIsOpen', 'ns=3;s="ValveCtrl".aValves[9].bMsgIsOpen',   
 'ns=3;s="ValveCtrl".aValves[10].bMsgIsOpen', 'ns=3;s="ValveCtrl".aValves[11].bMsgIsOpen', 'ns=3;s="ValveCtrl".aValves[12].bMsgIsOpen',   
 'ns=3;s="DIMagnetIsOn"', 'ns=3;s="LP\_Data".VISU.StepNames.sMainStep.sValue', 'ns=3;s="LP\_Data".VISU.StepNames.sSubStep.sValue',   
 'ns=3;s="LP\_Data".VISU.StepTime.iActStepTime', 'ns=3;s="LP\_Data".VISU.StepTime.iSetSteptime', 'ns=3;s="Counter"."Output Counter"',   
 'ns=3;s="Lab\_Parameter"."AutoTransferOnOff"', 'ns=3;s="Lab\_Parameter"."ContTransferOnOff"', 'ns=3;s="ParameterProcess"."uWorkingSet"."uSubStepParams"[12]."Params"[4]."iPumpSpeed"',   
 'ns=3;s="VolumeFlowNew"', 'ns=3;s="DensityNew"', 'ns=3;s="MassFlowNew"', 'ns=3;s="TemperatureNew"']  
  
  
# Connection to PostgreSQL database  
conn = psycopg2.connect(dbname="postgres", port= 5432, user="postgres", password="inseldip2023", host="172.20.0.35")  
  
read\_latencies = []  
write\_latencies = []  
# Set the number of iterations  
for \_ in range(10): # Repeat the process 10 times  
 try:  
 # Measure the time it takes to read from OPC-UA server  
 start\_time = time.monotonic()  
 values\_online\_data = [client.get\_node(node).get\_value() for node in node\_list\_online\_data]  
 end\_time = time.monotonic()  
 read\_latency = (end\_time - start\_time) \* 1000  
 read\_latencies.append(read\_latency)  
 #print(f"Read latency: {read\_latency:.2f} ms")  
  
# Measure the time it takes to write to the PostgreSQL database  
 start\_time = time.monotonic()  
 cur = conn.cursor()  
 cur.execute('''INSERT INTO online\_data1 (testid3, magnettemp, rotorfreq, pumpfreq, fluidtemp, conductivity, ph, density,   
 voumeflow, volumeflowanalog, tempCoriolis, massflow, absorbance, volumeflowsetpoint, valve1, valve2,   
 valve3, valve4, valve5, valve6, valve8, valve9, valve10, valve11, valve12, valve13, magnetstatus, mainstepname,   
 substepname, actsubsteptime, setsubsteptime, totalprocesstime, manualtransferonoff, conttransferonoff, recipe\_vf,   
 volumeFlowNew, densityNew, massFlowNew, temperatureNew) VALUES   
 (%s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s,   
 %s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s)''',   
 (values\_online\_data[0], values\_online\_data[1], values\_online\_data[2], values\_online\_data[3], values\_online\_data[4], values\_online\_data[5],   
 values\_online\_data[6], values\_online\_data[7], values\_online\_data[8], values\_online\_data[9], values\_online\_data[10], values\_online\_data[11],   
 values\_online\_data[12], values\_online\_data[13], values\_online\_data[14], values\_online\_data[15], values\_online\_data[16], values\_online\_data[17],   
 values\_online\_data[18], values\_online\_data[19], values\_online\_data[20], values\_online\_data[21], values\_online\_data[22], values\_online\_data[23],   
 values\_online\_data[24], values\_online\_data[25], values\_online\_data[26], values\_online\_data[27], values\_online\_data[28], values\_online\_data[29],   
 values\_online\_data[30], values\_online\_data[31], values\_online\_data[32], values\_online\_data[33], values\_online\_data[34], values\_online\_data[35],   
 values\_online\_data[36], values\_online\_data[37], values\_online\_data[38]))  
 conn.commit()  
 cur.close()  
 end\_time = time.monotonic()  
 write\_latency = (end\_time - start\_time) \* 1000  
 write\_latencies.append(write\_latency)  
  
 #print(f"Write latency: {write\_latency:.2f} ms")  
   
 except Exception as e:  
 print(f"Error during iteration: ")  
  
# Printing the arrays and average latencies:  
print(f"\nRead Latencies Array: {read\_latencies}")  
if len(read\_latencies) > 0:  
 avg\_read\_latency = np.mean(read\_latencies)  
 std\_read\_latency = np.std(read\_latencies)  
 print(f"\nAverage read latency over {len(read\_latencies)} iterations: {avg\_read\_latency:.2f} ms")  
 print(f"Standard deviation of read latencies: {std\_read\_latency:.2f} ms")  
else:  
 print("\nNo successful read operations to calculate average latency.")  
  
print(f"\nWrite Latencies Array: {write\_latencies}")  
if len(write\_latencies) > 0:  
 avg\_write\_latency = np.mean(write\_latencies)  
 std\_write\_latency = np.std(write\_latencies)  
 print(f"Average write latency over {len(write\_latencies)} iterations: {avg\_write\_latency:.2f} ms")  
 print(f"Standard deviation of write latencies: {std\_write\_latency:.2f} ms")  
else:  
 print("No successful write operations to calculate average latency.")  
# Plotting latencies  
iterations = list(range(1, len(read\_latencies) + 1))  
  
plt.figure(figsize=(10, 6))  
plt.plot(iterations, read\_latencies, marker='o', label="Read Latencies")  
#plt.plot(iterations, write\_latencies, marker='x', label="Write Latencies")  
  
plt.title("Read\_Latencies over iterations")  
plt.xlabel("Iterations")  
plt.ylabel("Read\_Latency (ms)")  
plt.legend()  
plt.grid(True)  
plt.show()  
  
#Plotting write latencies  
iterations = list(range(1, len(read\_latencies) + 1))  
  
plt.figure(figsize=(10, 6))  
#plt.plot(iterations, read\_latencies, marker='o', label="Read Latencies")  
plt.plot(iterations, write\_latencies, marker='x', label="Write Latencies")  
  
plt.title("Write\_Latencies over iterations")  
plt.xlabel("Iterations")  
plt.ylabel("Write\_Latency (ms)")  
plt.legend()  
plt.grid(True)  
plt.show()

## GUI basic layout

import sys  
import numpy as np  
import matplotlib.pyplot as plt  
from PyQt5.QtWidgets import QMainWindow, QApplication, QVBoxLayout, QHBoxLayout, QPushButton, QWidget, QMessageBox, QComboBox, QSpacerItem, QSizePolicy  
from PyQt5.QtGui import QPixmap, QPalette, QBrush  
from matplotlib.backends.backend\_qt5agg import FigureCanvasQTAgg as FigureCanvas  
from matplotlib.figure import Figure  
from opcua import Client  
from PyQt5.QtCore import Qt  
  
  
class MainWindow(QMainWindow):  
 def \_\_init\_\_(self):  
 super(MainWindow, self).\_\_init\_\_()  
  
 # Set window title  
 self.setWindowTitle("GUI Transparency")  
 self.setStyleSheet("background-color: lightblue;")  
  
 # Create a QComboBox for the dropdown button  
 dropdown\_button = QComboBox()  
 dropdown\_button.addItem("Option 1")  
 dropdown\_button.addItem("Option 2")  
 dropdown\_button.addItem("Option 3")  
 dropdown\_button.addItem("Option 4")  
  
 # Create main layout  
 layout = QVBoxLayout()  
  
 # Create horizontal layout for dropdown and spacer  
 top\_layout = QHBoxLayout()  
  
 # Add horizontal spacer item to push the dropdown button to the right  
 spacer\_item = QSpacerItem(40, 20, QSizePolicy.Expanding, QSizePolicy.Minimum)  
 top\_layout.addItem(spacer\_item)  
  
 top\_layout.addWidget(dropdown\_button)  
  
 # Add the top layout to the main layout  
 layout.addLayout(top\_layout)  
  
 # Create initial plot widget  
 initial\_plot = PlotWidget(self, title="Initial Plot")  
 layout.addWidget(initial\_plot)  
  
 # Create optimized plot widget  
 optimized\_plot = PlotWidget(self, title="Optimized Plot")  
 layout.addWidget(optimized\_plot)  
  
 # Create accept and reject buttons  
 button\_layout = QHBoxLayout()  
 accept\_button = QPushButton("Accept")  
 accept\_button.setStyleSheet("background-color: green; color: white;")  
 accept\_button.clicked.connect(self.on\_accept)  
 accept\_button.setFixedSize(80, 30)  
  
 # Create reject button  
 reject\_button = QPushButton("Reject")  
 reject\_button.setStyleSheet("background-color: red; color: white;")  
 reject\_button.clicked.connect(self.on\_reject)  
 reject\_button.setFixedSize(80, 30)  
  
 button\_layout.addWidget(accept\_button)  
 button\_layout.addWidget(reject\_button)  
 layout.addLayout(button\_layout)  
  
 # Create central widget and set the layout  
 central\_widget = QWidget()  
 central\_widget.setLayout(layout)  
 self.setCentralWidget(central\_widget)  
  
 # File paths of the generated plots  
 initial\_plot\_path = "path/to/initial\_plot.png"  
 optimized\_plot\_path = "path/to/optimized\_plot.png"  
  
 # Load and display the plot images in the plot widgets  
 initial\_plot.load\_plot\_image(initial\_plot\_path)  
 optimized\_plot.load\_plot\_image(optimized\_plot\_path)  
  
 def on\_accept(self):  
 result = QMessageBox.question(  
 self, "Accept Parameters",  
 "Do you want to accept the optimized parameters?",  
 QMessageBox.Yes | QMessageBox.No  
 )  
 if result == QMessageBox.Yes:  
 print("Accepted")  
 # Perform further actions for accepting parameters  
  
 # Connect to OPC UA server  
 client = Client("opc.tcp://plc\_address:port")  
 client.connect()  
  
 # Browse OPC UA address space and find the target node to write parameters  
 # target\_node = "modbus\_buffer"."iSpeedandTime"...  
 #"DigitalTwin\_To\_Recipe"."SS1\_accept"  
  
 # Write the accepted parameters to the target node  
 # client.write\_value(target\_node, accepted\_parameters)  
  
 # Disconnect from OPC UA server  
 client.disconnect()  
 else:  
 print("Not Accepted")  
  
 def on\_reject(self):  
 result = QMessageBox.question(  
 self, "Reject Parameters",  
 "Do you want to reject the optimized parameters?",  
 QMessageBox.Yes | QMessageBox.No  
 )  
 if result == QMessageBox.Yes:  
 print("Rejected")  
 # Perform actions for rejecting parameters  
 else:  
 print("Not Rejected")  
  
  
class PlotWidget(QWidget):  
 def \_\_init\_\_(self, parent=None, title=""):  
 super(PlotWidget, self).\_\_init\_\_(parent)  
 self.figure = Figure()  
 self.canvas = FigureCanvas(self.figure)  
 self.ax = self.figure.add\_subplot(111)  
 self.ax.set\_title(title)  
 self.layout = QVBoxLayout()  
 self.layout.addWidget(self.canvas)  
 self.setLayout(self.layout)  
  
 def load\_plot\_image(self, file\_path):  
 # Load the plot image from the specified file path  
 pixmap = QPixmap(file\_path)  
  
 # Scale the image to fit the widget size  
 pixmap = pixmap.scaled(self.size(), aspectRatioMode=Qt.KeepAspectRatio)  
  
 # Set the image as the background of the widget  
 self.setAutoFillBackground(True)  
 palette = self.palette()  
 palette.setBrush(QPalette.Window, QBrush(pixmap))  
 self.setPalette(palette)  
  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 app = QApplication(sys.argv)  
 window = MainWindow()  
 window.show()  
 sys.exit(app.exec())