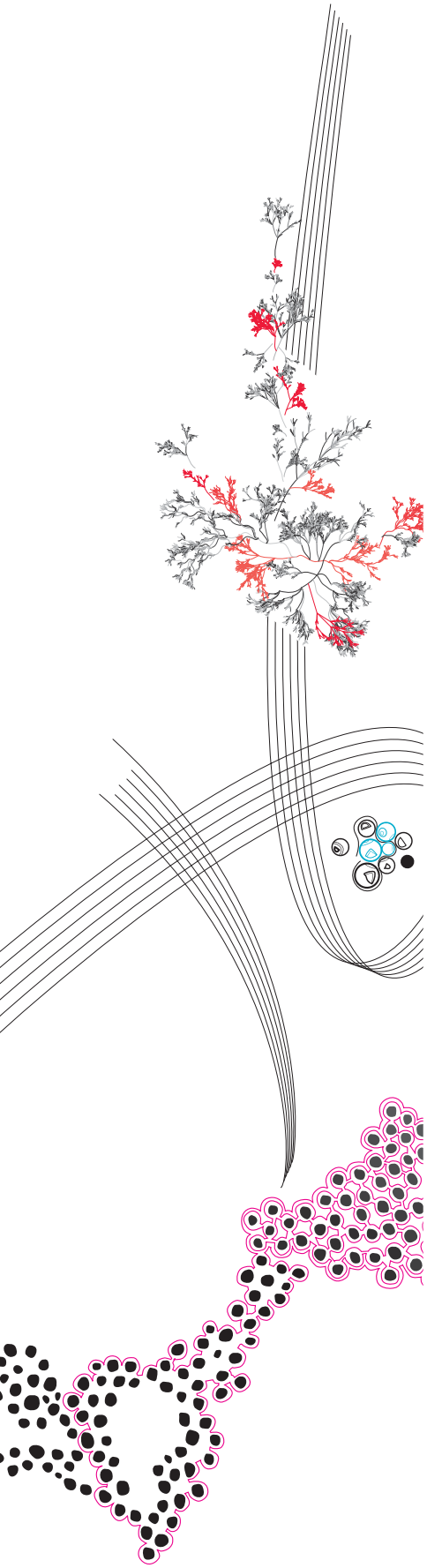
Design Project



Digital Twin-based Control Dashboard for CO_2 Capture Pilot Plant

Rein Fernhout, Eren Kodal, Arnout Luinge,
Yannick Krijnen, Stefan Morriën

Supervisor: Luiz Bonino da Silva Santos

April 18, 2025

Department of Computer Science
Faculty of Electrical Engineering,
Mathematics and Computer Science,
University of Twente

UNIVERSITY OF TWENTE.

Abstract

Climate change is one of the biggest issues that we are facing as a society. Carbon dioxide (CO_2) capture and Storage (CCS) is one such technology that aims to reduce the effects of climate change. Hovyu, the client company, located in the Netherlands aims to create novel solutions in the CCS paradigm. Hovyu has commissioned Coraltech BV, another company in the Netherlands, to build a pilot plant to implement their novel design. Hovyu wants to enhance the efficiency of the pilot CO_2 capture plant being built by Coraltech, through implementing a digital twin dashboard. The development team from University of Twente, has created an implementation of a digital twin dashboard for the client. This dashboard is capable of visualizing the Piping and Instrumentation Diagrams (PID), of the pilot CO_2 capture plant. Sensor data is fetched from the Siemens S7-1200 PLC, visualized for each node in the diagram. Graphs and simulations are made available in an intuitive layout. This report focuses on going in-depth about the design process for the development of the digital twin dashboard for Hovyu. The implemented design will contribute to the CCS paradigm, by incorporating the digital twin which is a novel idea. This implementation will further optimize the CCS technologies. During literature review it was noted that; while much research is available in regards to increasing the efficiency of CO_2 capture plants through changing the chemical solutions used, or changing the design (hardware/process solution); not much has been researched in regards to implementing a digital twin (software solution). The development team and the client believe this implementation will aid the current CCS technologies and help combat climate change and lower CO_2 emissions.

Keywords: Digital Twin, Dashboard Development, Carbon Dioxide Capture and Storage (CCS), Pilot Plant, Programmable Logic Controller (PLC) Integration, Process Simulation, Carbon Emission Reduction, Computer Science

Contents

Chapter 1

Introduction

Every year, numerous governmental and non-governmental organizations convene to discuss strategies for combating global warming. However, the main contributor to global warming, carbon emissions, continue to increase. Carbon dioxide (CO_2) is the most significant greenhouse gas due to its high heat absorption and re-radiation and has several promising countermeasures, one of which is CO_2 capture and storage (CCS) [1], [2].

CCS reduces CO_2 emissions by capturing CO_2 from industrial sources, such as factories and power plants, and storing it underground. The effectiveness of CCS, while also dependent on the type of emission source, is largely determined by the capture technology [2]. One company developing such technology is Hovyu, based in The Netherlands, whose aim is to provide a modular, easily integrated CCS system.

Hovyu has developed a novel process to capture CO_2 based on electrochemical regeneration of solvent[1] (see patent for reference). The process is in its development during this year (2025) a pilot plant will be built for demonstrating the technology.

Currently, most CCS technologies utilize some kind of programmable logic controller (PLC), which are typically limited to physical inputs & outputs hence restricting the ability to monitor the CCS. Due to this limiting nature of the current system, the aim of this design project is to create an application with remote access to the CO_2 capture plant which would allow Hovyu to view real-time data and act on it. Moreover, if the application would not only show the state of the CO_2 capture plant but also simulate it, the capture plant's efficiency could be monitored in advance and deviations from the optimal operating values could be prevented; and in the case of deviations, simulations can be used to return to optimal performance. This concept of simulation, is known as a "Digital Twin".

A digital twin is "a virtual representation of a physical object or process capable of collecting information from the real environment to represent, validate and simulate the physical twin's present and future behavior" [3].

This project explores incorporating the concept of a digital twin to the existing CO_2 capture architecture of Hovyu. This report namely discusses: why incorporating the concept of a digital twin is a novel idea through relevant literature review, elaborates and provides documentation on the implemented Digital Twin dashboard and lastly, discusses potential future improvements.

[1] https://patents.google.com/patent/WO2022240290A1/en

Chapter 2

Literature review

The concept of a digital twin (DT) is an idea that has existed for a very long time; however, its application within the context of CCS remains fairly unexplored. For this reason, this chapter aims to review the existing literature around digital twins, digital twins in industry, digital twins in the context of CCS, and aims at illustrating the the research gap on digital twin usage in CCS technologies. The literature review is short, as it isn't a main part of the design process; as Hovyu has already done their due diligence in the research side. Yet, we believe that a short overview on the reasons why this project is useful through identifying this research gap to be important.

2.1 Digital Twins

The term "digital twin (DT)" was originally coined by Grieves in 2003 in his course at the University of Michigan [4]. At the time, however, it did not gain much traction. This was mostly due to the lack of technological foundation to support the development of a practical application of a DT [5]. The improvement of the technology as a whole between 2003 and 2011 caused a surge in potential capabilities and popularity of DTs. In fact, it was NASA who showed the significant capabilities of DTs in the aerospace industry, a safety-critical field where failures can make the difference between life and death [5]. This potential for DTs to improve reliability and risk mitigation is something that is also sought after in CCS applications; albeit not being a safety-critical technology by definition.

In 2014, the first white paper (an informative report or guide) published by Grieves [4] was published, starting what is referred to as the 'growth stage' [5].

2.2 Digital Twins in Industry

During this growth stage, DTs continued to gain more traction across several disciplines, with goals ranging from synchronizing and optimizing design and production, to enabling real-time monitoring of production processes through a synchronized digital representation of the physical space. It is safe to say, DTs went from simply being a part of only safety-critical industry usage to being part of many industries that wanted to achieve higher efficiency.

Around 2020, most of the DT applications were related to design, production and prognostics and health management (PHM). However, DT applications were also applied in other areas, such as wind farms for General Electric, and even machine-human interfaces for Siemens [5]. One thing these areas all had in common was significant increases in operation efficiency.

2.3 Digital Twins in the Context of CCS

As opposed to the roughly 50 relevant papers published between 2003 and 2019 on Digital Twin (DT) applications [5], the amount of research done on DT applications in the context of CCS is still relatively small and newly emerging as a whole. Still, DT applications are already utilized throughout the field of CCS to an extent [6]. One similar implementation to our implementation was seen in one of the papers found during the literature review.

A DT application constructed based on the Imperial College CO_2 capture pilot plant bears some resemblance to the scope of this project, but this application primarily focused on the simulation of the pilot plant [7], whereas the aim of this project is not just to simulate, but also to view a real-time digital representation of the physical space.

In this study, several benefits were noted, including but not limited to the ability to conducts tests and simulations without affecting the physical plant and the ability to ensure operational reliability.

Similarly, the research and the purpose of the pilot CO_2 capture plant reflect the industry's efforts to harness the benefits of DTs in the context of CCS.

Chapter 3

Scope

The digital twin dashboard being designed and implemented for this project has a time frame of roughly 8 weeks. Due to this time constraint, the scope of the project is limited by nature. For the scope to be properly conveyed, context of the company Hovyu, and their need for commissioning this project, together with their design requirements are as shown below.

3.1 Context on Company

Hovyu is currently developing several modular CO_2 capture plant technologies. The pilot plant, designed by Hovyu has been commissioned to another company (Coraltech BV) for it to be built. One of these systems is located in Portugal. However, this system utilizes some advanced engineering principles and many different components, making it too complex for an in-depth exploration within the time constraints of this project.

Another CO_2 capture plant is currently being built, and will be located in Eindhoven, The Netherlands. Its logic is relatively simple and consists of far fewer components. Furthermore, Coraltech has provided a PLC in Enschede, similar to the one that will be used for the CO_2 capture plant in Eindhoven. This allows for on-site testing of the application. Thus, the system that will be modeled in this project will be the one that will be used in Eindhoven. The purpose of the design project is to provide a digital twin dashboard for Hovyu, and it has been determined that designing it after a different (less complicated) CO_2 capture plant doesn't diminish the value of the project.

3.2 Stakeholders

The stakeholder list is small in nature. The relevant direct stakeholders are as follows:

- **The client:** Diego Di Domenico Pinto
  Within the report, the client will be referred as the Stakeholder Representative. Diego is the main stakeholder, being the founder of the company, Hovyu. Diego serves as the first point of contact for the project as a whole.
- **Hovyu:** Company working on CO_2 capture technologies, founded in 2018.
- **Coraltech:** Company in Enschede, commissioned by Hovyu to build the CO_2 capture pilot plant itself. Main point of contact is Erik Verbeek, who is first point of contact in regards to getting information about the pilot plant and the PLC interface.

Within the development cycle, close contact has been formed with Diego and Erik.

3.3 Context on Existing Architecture and the Reason Why the Project is Commissioned

Hovyu's pilot plant, which consists of more than 80 sensors; is currently being used to control and monitor the process of the CCS.

As explained by the Hovyu founder (Stakeholder representative and also the client for the project) which we had been in contact with through the design process: "The control strategy is implemented in a Siemens PLC (model S7); which can be modified by using Siemens' own propriety software. A LabVIEW script has also been developed by an engineer from CoralTech, for the purpose of interacting with the PLC."

This current architecture of the process imposes various constraints for Hovyu, which form the basis for commissioning this project.

3.4 Main Constraints

1. Proprietary Software Constraints: The Siemens software currently in use, imposes limitations due to its propriety nature, leading to high costs and reduced flexibility. The stakeholder representative for Hovyu has expressed a preference for a solution that offers more flexibility.

2. LabVIEW Usability Issues: The current script uses LabVIEW, but finding skilled developers is difficult due to its specialized nature and limited talent pool compared to more widely used languages. This creates budget constraints, as LabVIEW developers are more costly. It also creates issues with development pace, where desired features are implemented at a slower pace, due to the limited talent pool. Hence, the product owner wishes to utilize a more wide known, mainstream solution; such as Python.

3. User Interface Limitations: The Siemens interface currently used has been identified as suboptimal. Particularly in terms of user experience. A redesign that is both intuitive and modern would be optimal for the project.

The scope of this project has been identified as, creating a dashboard for the digital twin, and trying to minimize these current constraints. The requirements for the project are discussed in length in the following chapter.

Chapter 4

Requirements Engineering

With the previously stated scope in mind, an initial meeting has been conducted with the client (stakeholder representative from Hovyu) to elicit the requirements for the design of the Digital Twin Dashboard.

For the intended design to be understood by the development team, it is crucial for the elicitation of requirements to be done in detail. Hence through the process, the agile methodology has been utilized.

The development team and the stakeholder representative have done two initial meetings. The former for the requirements to be formed, and the latter for the requirements to be confirmed; to make sure that elicitation of requirements has been done correctly.

The system has the only direct stakeholder which is Hovyu, and they will be the sole user of the system. Hence, the requirements have been formed through the lens of the system; instead of from the lens of potential different users. MoSCoW analysis has been used to give priority classifications for each requirement. Due to both time constraints and also clients preferences on which requirements are more important.

The MoSCoW methodology is giving requirements the following priority levels, from highest to lowest as follows:

- **Must:** Highest priority, needed for the minimum viable product (MVP) to be achieved.
- **Should:** Very relevant to the core functionality, yet not a must level.
- **Could:** Not a core functionality, however it would be nice to have.
- **Won't:** Functionality that has been deemed not relevant, or outside the scope of the current design as a whole.

The requirements have been divided into functional and quality requirements and can be seen in section 4.1 and section 4.2.

4.1 Functional Requirements

Must Have (M)

- **M1** The system must have a graphical user interface (GUI) that visualizes the physical hardware of the system.
- **M2** The system must display the hardware's sensor readings in the GUI.
- **M3** The system must have a mechanism that allows the user to change a sensor's value.
- **M4** The system must have a mechanism that allows the user to change the rate at which the system logs a set of variables.
- **M5** The system must have an emergency button to shut down the system by switching all the equipment off.
- **M6** The system must have a button to switch between the different pages of the PID.

Should Have (S)

- **S1** The system should replace LabVIEW's software fully, operating as a standalone application, deprecating LabView (as the client prefers a language which is more widely used for ease of work in the long term).
- **S2** The system should display real-time data plots & graphs for the system's sensors (temperature, pressure, etc.).
- **S3** The system should have upper and lower bounds / margins of error for sensors to allow for normal fluctuation around the set threshold.
- **S4** The system must visualize text boxes above the sensors in the GUI that define thresholds, minimum and maximum values for proper functioning of the system.

Could Have (C)

- **C1** The system could react to parameters changing by simulating what would happen to the physical hardware (if disconnected).
- **C2** The system could implement a stable condition identifier, achieved when the readings of some sensors remain within a specified threshold.
- **C3** The system could have a function that re-calibrates a sensor based on a minimum and maximum to improve the accuracy of an older sensor.

Won't Have (W)

- **W1** The system won't replace Siemens' proprietary technology to connect to PLC, avoiding the need to make use of costly software licenses.

4.2 Quality Requirements

Quality Requirements (Q)	
Q1	The system must poll sensor data at a minimum frequency of once every 1000 milliseconds to ensure near real-time responsiveness in the GUI.
Q2	The graphical user interface must update sensor readings and visualizations within 1000 milliseconds of receiving new data to ensure smooth user interaction.
Q3	Upon pressing the emergency shutdown button, the system must execute the shutdown procedure within 1 second.
Q4	Logged data must be written to storage within 1 second of collection to avoid data loss during continuous operation.
Q5	The system must be ready for operation (including GUI and sensor polling) within 5 seconds of launch.
Q6	The system must be distributable as a standard file format such as .exe that can run on a target machine without requiring external dependencies such as Python.

4.3 Requirement Iterations

The previously described requirements for the design have been iterated over twice through the requirements elicitation phase and further in early stages of the development, as priority levels have changed due to difficulty of the tasks or due to changes by the client (stakeholder representative).

These iterations are a key part of agile development, and ensure that the final list of requirements as shown above, will lead to the best product to be handed in to the client.

The most noteworthy changes are:

- Replacing the need to utilize Siemens' proprietary technology to connect to the PLC, avoiding the need to make use of costly software licenses. (**W1**)

In the requirements elicitation phase, the client has mentioned that they would like to replace the Siemens' proprietary software, due to budget concerns and others limitations. Due to client needs, this has been initially put as a Could level requirement. However, early on it has been identified there is no simple way of circumventing this issue; hence it has been moved out of the scope of the current design.

- The ability to simulate the CCS' behavior, both while being connected and disconnected to the PLC. (**C1**)

  Completion of the dashboard itself is highly challenging. The client has expressed their preference in a well developed dashboard in contrast to including trivial digital twin simulations while providing a worse quality dashboard.

Chapter 5

Project Timeline and Structure

Before discussing the implementation process of the project, it is important to outline a general plan for the project. Such deadlines for requirements, through an estimated timeline of how long each requirement will take, makes it easier for the progress to be denoted. Hence making sure at the end of the project, the submitted product will be at least a minimum viable product. This goes hand in hand with agile project management. A general risk analysis has also been made, and a plan for testing the product.

5.1 Planning

A gantt chart has been devised as seen in Appendix A. The one in Appendix A is the final version of the gantt chart, where the sections have been denoted with the percentage of completion, after the implementation phase of the project has been over.

This gantt chart has been initially made for the purpose of outlining the timeline of the project. Identifying core features to be implemented. The identified core features have been given time-frames, in regards to how long they would approximately take.

The gantt chart has been a good model in checking progress of the implementation, being able to visualize whether the implementation process is going smoothly, lagging behind the intended time-frame or being completed faster than usual.

In case of lagging behind in core features, such as must level priority requirements, lower priority level requirements have been postponed till the completion of the aforementioned must level requirements. In case of progress faster than foreseen in the planning process, should level and then could level priority requirements have been implemented.

The gantt chart hence has been proven useful during the implementation process.

5.2 Process

Many different tools and methodologies have been used for a smooth design and implementation process.

5.2.1 Methodology

The agile methodology has been chosen for this project. It differs from the classical approach in terms of amount of collaboration. The development team works in close collaboration with the client (stakeholder representative). Weekly meetings have been conducted with the stakeholder representative to get feedback, and reiterate on the priority levels of the requirements or if needed, change requirements altogether. For the purpose of providing a software that is appropriate for the needs of the client (Hovyu).

5.2.2 Tools

Trello has been utilized as a primary tool for management, for the purpose of assigning weekly tasks to each member of the development team. The team has come to the conclusion that Jira was not needed due to the small scale of the project. Instead, a github repository was used with proper usage of branches and merges after extensive testing; to sustain the collaborative nature of the agile approach.

5.2.3 Models

Models closely related to the agile methodology such as Scrum have been used for the weekly meetings with the development team together with the client. Discussing the progress with Scrum leader which changed biweekly, where each member had been the Scrum leader once.

5.3 Risk Analysis

Many potential risks can arise in both design and implementation phases respectively.

For this purpose, it is important to conduct a risk analysis; to make sure risks will arise in the minimum (predicting potential risks that may happen, and mitigating them before they happen). For risks that can't be controlled, it is important to create a general schema on how to minimize their effects on the project.

5.3.1 Risk Analysis during Design Phase

- **Risk: Suboptimal Requirements Elicitation Phase:** If requirements are poorly elicited during the initial phase, misunderstandings about the client's needs can lead to a design that doesn't align with their expectations or the project's objectives. This can result in rework, missed deadlines, or a product that fails to meet client needs.
- **Mitigation:** Requirements Elicitation phase has been done in two separate meetings. The former for the purpose of eliciting the requirements. The latter for the purpose of showing the devised requirements list and double checking if they are appropriate with the client. Close contact with the client is key, also for the later on phases of development.
- **Risk: Improper Planning Phase:** Inadequate planning can lead to unrealistic timelines, due to not understanding how long each requirement might take. For this purpose, the planning stage has been deemed very important.
- **Mitigation:** Early on, before development phase, a general planning overview has been devised, as seen in Appendix A.

5.3.2 Risk Analysis during Implementation Phase

- **Risk: Downtime in development due to PLC connection issues:** PLC connection is a must for the development of the digital twin dashboard. Without one available, development will slow down.
- **Mitigation:** A PLC has been provided by the client, located at the office of Coraltech, located in Enschede. A VPN connection has also been provided to access it remotely.
- **Risk: Scope Creep:** Increasing the scope of the project too much, with way too many requirements will result into not being able to deliver the intended product in time.
- **Mitigation:** With the client, a proper scope has been devised with the 8 weeks development timeline in mind. The scope focused on the dashboard functionality instead of the digital twin simulation related functionality.
- **Risk: Merge Conflicts:** Merge conflicts can arise when multiple development team members work in the same code base.
- **Mitigation:** Good utilization of Git with proper use of branching techniques and timely merging, after properly testing that the functionality to be merged is without issues, is key.

5.4 Plan for Testing

Testing for this design process has been divided into two sections. Namely, client testing for the purpose of usability testing and unit testing.

5.4.1 Usability Testing with Client

Close contact with the client is has been kept throughout the design process. The implemented digital twin dashboard will solely be used by Hovyu, the stakeholder company. Hence the usability of the developed product is solely determined through the appointed stakeholder representative (client for the project).

Weekly meetings have been done with the client, either in person or online. Progress has been explained and a demo of the current iteration has been shown to the client for feedback.

One final usability testing with the client has been planned for the final week before submission of the project and handing in the product to the client. For the purpose of evaluating the product one last time.

5.4.2 Unit Testing

Unit testing is for the purpose of testing "units" of code (separate isolated functions). This ensures that these functionalities work as expected, and then are merged to the master branch. Good coverage of the code base is a must, even with the constraints due to time available in the development phase.

Good unit testing ensures that bugs are caught on early, and do not impact later on stages of development. It improves code quality and facilitates as a safety net during refactoring.

Chapter 6

System Overview

It is important to give an overview of the CCS (CO_2 Capture and Storage) process. This will in turn make the novel process designed by Hovyu easier to understand. This section has been written through the information provided by Hovyu, namely the client representative.

As previously explained, CCS is a technology designed to reduce greenhouse gas emissions (such as CO_2); by first capturing the CO_2 from the industrial processes and then storing it underground to prevent it entering the atmosphere. This process can be outlined in three stages.

- **Capture:** CO_2 is separated from the other gases and then captured.
- **Transportation:** The captured CO_2 is transported to the storage site.
- **Storage:** CO_2 is injected into underground where it gets securely trapped.

The above is a high-level framework of the whole CCS process. The real complexity, and the devised novel solution by Hovyu, lies within the engineering details of the capture stage: Absorption and Regeneration.

- **Absorption:** In this initial step, CO_2 is removed from flue gases or industrial emissions. It involves chemical solvents which bind with CO_2. For an optimized absorption process: Precise control of gas flow rates. and proper selection of solvents is crucial. The precise control of the flow rates is explained further in the Control Narrative section. The solvents chosen by Hovyu falls under confidential details, due to their novel nature; hence will not be discussed.
- **Regeneration:** After the solvent becomes saturated with CO_2, it is treated to release the captured gas. This process, often involving heat, allows the solvent to be restored and reused in the system. The regeneration section of the ZeroEmission-Ultra Stripping (ZEUS) process is a proprietary which is confidential. The neutralization step is a fast process while the electrodialysis is slow process. To couple with this difference, Hovyu has developed a continuous batch process which allows the regeneration section to operate without interruption. A robust monitoring system and control strategy is required to keep the process targets and performance. The specifications of the process cannot be disclosed due to the confidentiality.

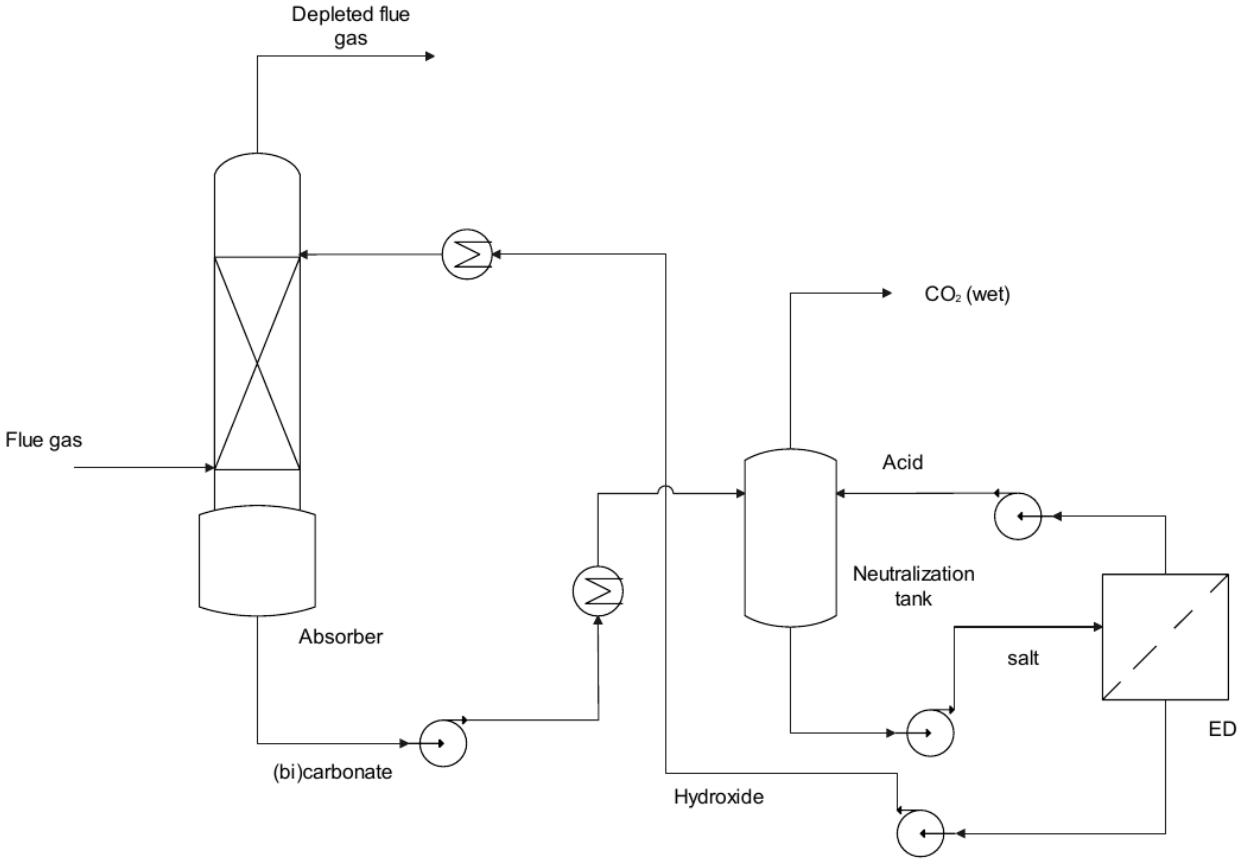


FIGURE 6.1: General (simplified) Process Flow Diagram (PFD) of the plant

Hovyu, in their skid overview has decoupled the regeneration process from the absorption; as their novel idea lies within the regeneration section.

For the purpose of the design, the regeneration process, which is very complicated; has been been fragmented into three separate PIDs (Piping and Instrumentation Diagrams). PIDs serve the purpose outlining the process in detail, in contrast to the general overview-like Process Flow Diagram (PFD).

These three PIDs, represent the novel idea in detail; hence due to its confidential nature, is not part of this report.

For the purpose of the report, a general simplified Process Flow Diagram (PFD) has been provided by Hovyu. PFD is a high level diagram which just outlines the process flow without diving into the design details. This provided PFD also abstains from showing the novel two tank system. (See Figure 6.1)

6.1 Control Narrative

The control narrative of the regeneration process has also been provided, as it is relevant to designing and implementing the digital twin of the system. The control narrative in detail explains how the process functions. This is confidential in nature and will not be explained. In general, the control narrative outlines how the system dynamically operates through a long list of "if else statements", to make sure that the values are within operating range.

6.2 Provided PLC

To have an easier time designing the digital twin dashboard, a Programmable Logic Controller (PLC) has been provided in Coraltech BV located in Enschede. This in turn allowed the development team to be able to test connection to the PLC in person.

A VPN connection was also provided to enable secure remote access to the systems and data necessary for development when working remotely on the software. The provided PLC is similar in nature to the one used in the pilot CCS plant.

The provided PLC can be seen below in Figure 6.2

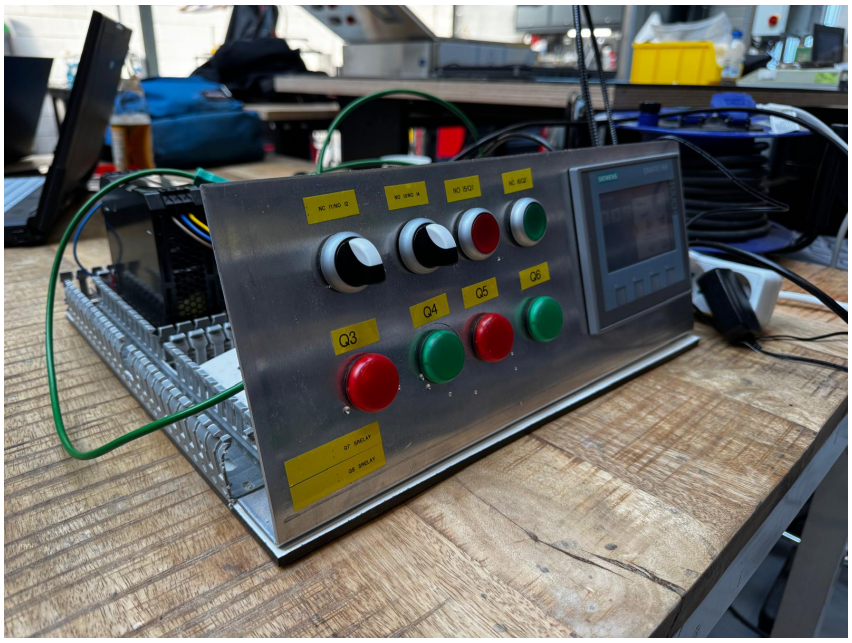


FIGURE 6.2: PLC provided by Coraltech in Enschede

The Siemens PLC is of model S7 1200 with CPU 1214C.

Chapter 7

Implementation of Design

As explained prior, the regeneration PIDs have been decoupled from the absorption process. The digital twin dashboard's main functionality focuses on implementing these given three PIDs as a GUI in the dashboard.

7.1 General System Architecture

The Digital Twin dashboard is a modular and extensible framework designed to simulate, monitor, and interact with physical CO_2 capture pilot plant. It integrates PLC communication, real-time data management, and a PySide6-based GUI dashboard to provide an interactive digital representation of physical operations.

7.1.1 Key Components

- **PLC Communication:** Interfaces with physical sensors and actuators. Uses snap7 library to fetch sensor readings from the PLC. Ensures accurate real time data representation.
- **Data Management:** Centralized data handling through the DataStore module. Supports both real and simulated data sources, for the purpose of a digital twin. Provides an abstraction layer for the front-end to access data.
- **Front-end GUI:** Displays sensor data and system states through an intuitive user interface. Enables user interactions such as monitoring and control, with varied options as settings.

7.1.2 Data Flow

- **Sensors/Actuators in the CO_2 capture pilot plant:** Provides data to the PLC
- **PLC Communication Layer:** Fetches data using snap7 and translates it for the system.
- **DataStore:** Storing and processing sensor data.
- **MockProvider:** Simulates sensor data for development and testing.
- **Dashboard:** Visualizes the digital twin and allows for user interaction.

7.2 Front-end

The front-end is the user-facing layer of the system, providing a graphical interface for monitoring and interacting with the Digital Twin (as seen in Figure 7.1).

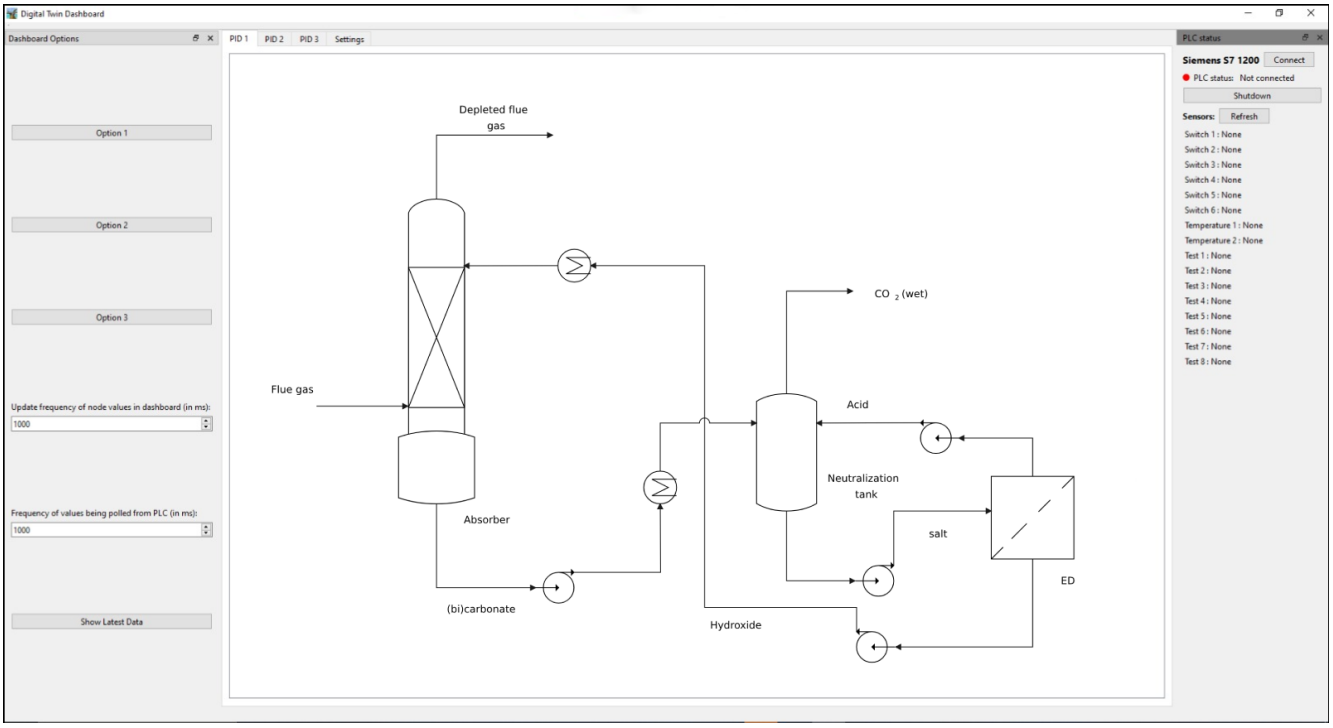


FIGURE 7.1: Digital Twin Dashboard UI, the PIDs have been edited out and swapped with the simplified PFD shown before in Figure 6.1. Due to confidentiality reasons the actual PIDs will not be shown in the report.

7.2.1 Main Dashboard

- **File:** widgets/dashboard.py
- **Class:** DashboardWindow

Serves as the primary container for all GUI components. Initializes the menu bar, toolbar, status bar and central widget. Integrated with DataStore for real-time data visualization. Refer to figure 7.2 for information on the class structure.

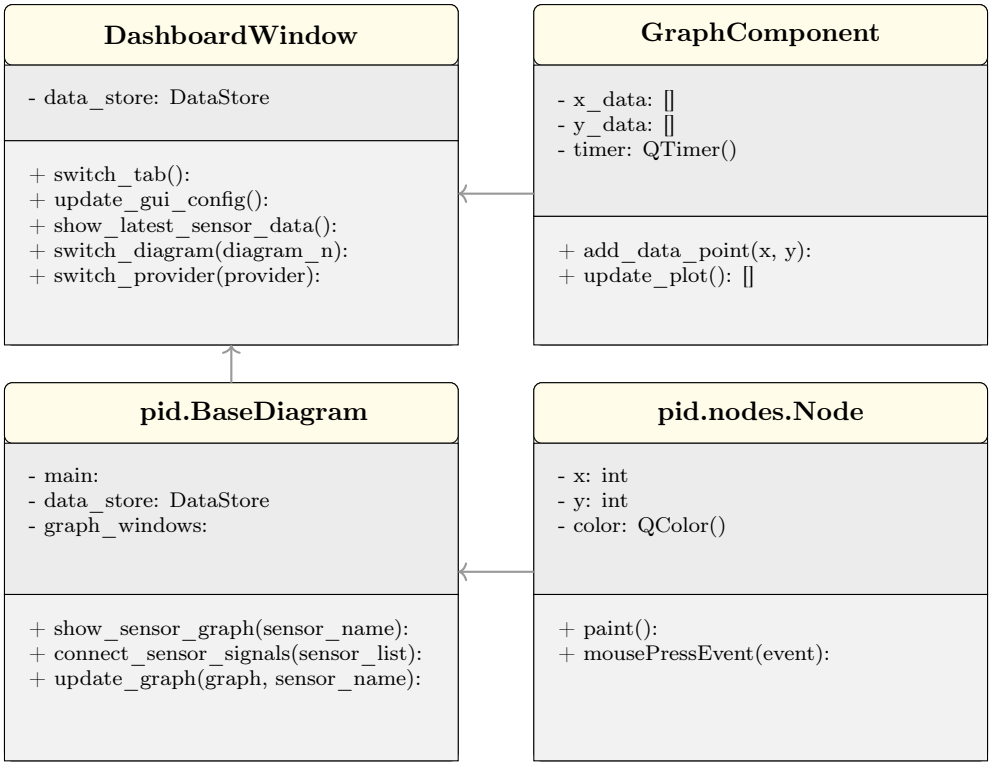


FIGURE 7.2: DashboardWindow, GraphComponent, BaseDiagram and Node Class Diagram

7.2.2 Option & Input Fields

The system provides the user with a column on the left side of the screen containing options and input fields for the PLC. Through this interface, the user can:

- **Change the data polling frequency:**
  The polling frequency determines how often data is retrieved from the PLC, allowing the user to adjust the volume of data.
- **Adjust the data update frequency:**
  The data polling frequency can differ from the update frequency, as the user may not require every minor update in data to be reflected visually in the diagram.
- **Update the calibration values:**
  Some sensors may require calibration values, either because they have degraded physically and are no longer accurate without an offset, or because the sensors require an offset to be human-readable.
- **Force stop the system using an emergency button:**
  This button will attempt to stop any activity in the plant in the case of an emergency.

7.2.3 PLC Connection & Values

The system provides the user with a column on the right side of the screen containing a button to toggle the connection to the PLC. When connected, the raw values retrieved from the PLC are displayed.

In the finalized system, these raw values would typically not be shown in this interface. However, since the actual plant was not yet available during development, testing had to be conducted using a simplified example PLC. This setup did not reflect the full complexity of the real system, so the raw values were displayed to clearly indicate the data's origin and assist with debugging and verification during development.

7.2.4 Graph Component

The graph component was designed with user flexibility in mind. When a sensor is clicked, a separate GUI window opens, which can be moved and resized according to the user's preferences. This also allows the user to open multiple graphs simultaneously and display them side by side for comparison.

The values on the graph are displayed in a sliding window manner, with a default limit of 100

Refer to Figure 7.2 for information on the class structure.

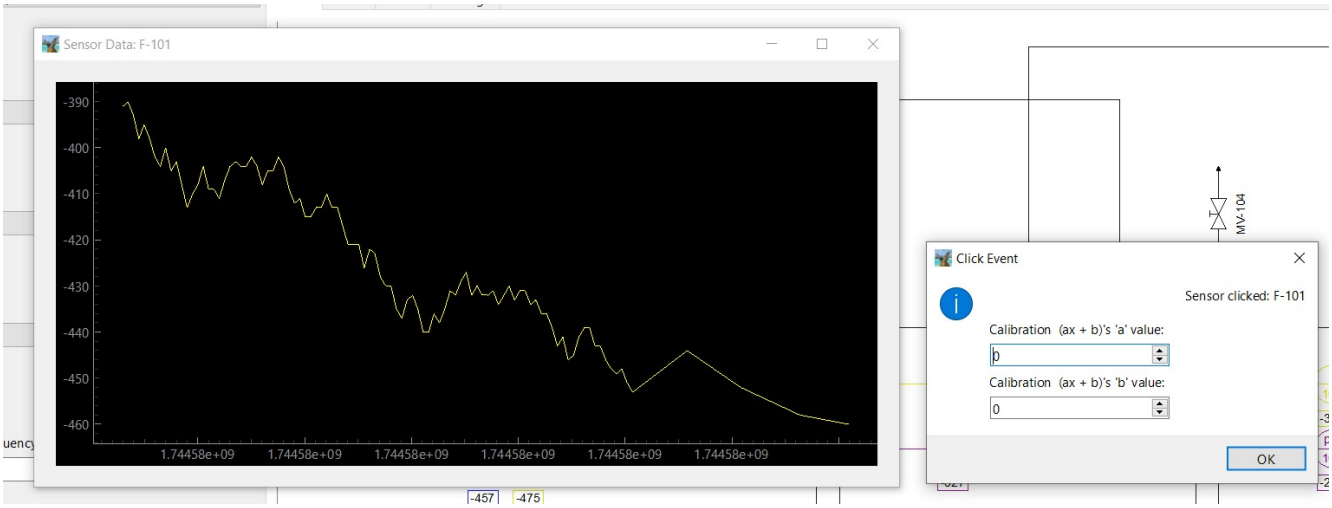


FIGURE 7.3: Graph Component

7.3 Backend

7.3.1 Physically Connecting to the PLC

To make the physical connection the PLC we utilize the Ethernet port on the device. Given that the host PC also comes with networking capabilities this makes for an easy setup.

The bulk of the communication between the PLC and the program is handled by the `Snap7` python library, *"an open-source, 32/64 bit, multi-platform Ethernet communication suite for interfacing natively with Siemens S7 PLCs."* [8]. This library comes with many useful methods, such as reading and writing to DB-addresses and helper functions for converting data to the right formats.

DB-Addresses

In the PLC each sensor and each actuator is configured to be exposed through a DB-Address. This allows its data to be read or be written to. Different behaviors can be configured for each type of sensor/actuator. Each DB-Address corresponds to a single byte, but readings can cover multiple bytes or single bits in a byte.

The simplest ones are binary sensors and actuators. They are stored as single bits, at an offset in a byte. As en example, take a switch at address 34.5, it value can now be found at the 6th bit (start at 0) of the 34th byte. Only full DB's can be read, thus we need to read the whole byte and can then take the 6th bit as a boolean value for the state of that switch.

PLC Config

A configuration file is available to setup the PLC with its sensors and actuators. This yaml file includes configuration for the PLC(s) itself (address, port, name) and for each sensor/actuator the following:

- name: Human readable name
- type: Type of the sensor [switch, temp, hum, etc.]
- address: DB-Address, either byte.bit or just byte
- id: Identifier corresponding to its Id to the PID

This configuration file could in the future be integrated or replaced by the UI.

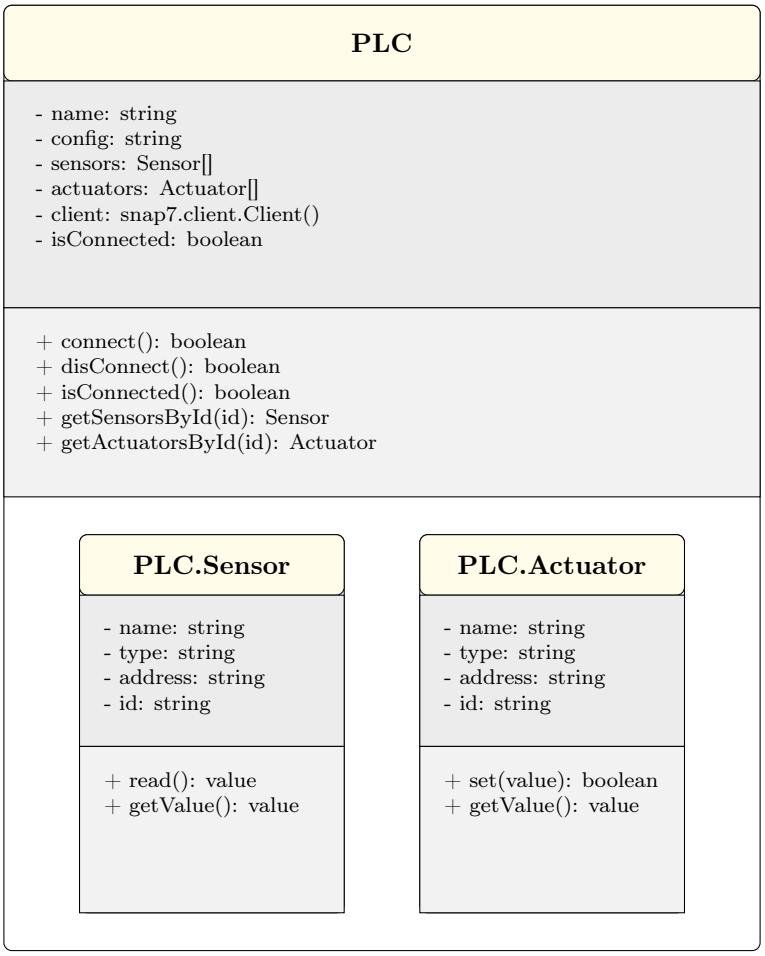


FIGURE 7.4: PLC Class Diagram

The PLC in software

The implementation now is pretty straight forward. The PLC object can be initialized with the settings from the config file. Once the connection is made values can now be read.

The PLC owns an object for each sensor and actuator belonging it. These object have getters and setters, to read from sensors and write to actuators respectively. The sensor's type dictates how the read data should be interpreted, thus also how it should be stored.

A class diagram has also been provided to aid understanding of the class structure, as shown in Figure 7.4

Polling

To periodically read new data from the sensors a poller thread is started. This thread will read at a set interval, and store the values in the sensor objects.

7.3.2 Data Management

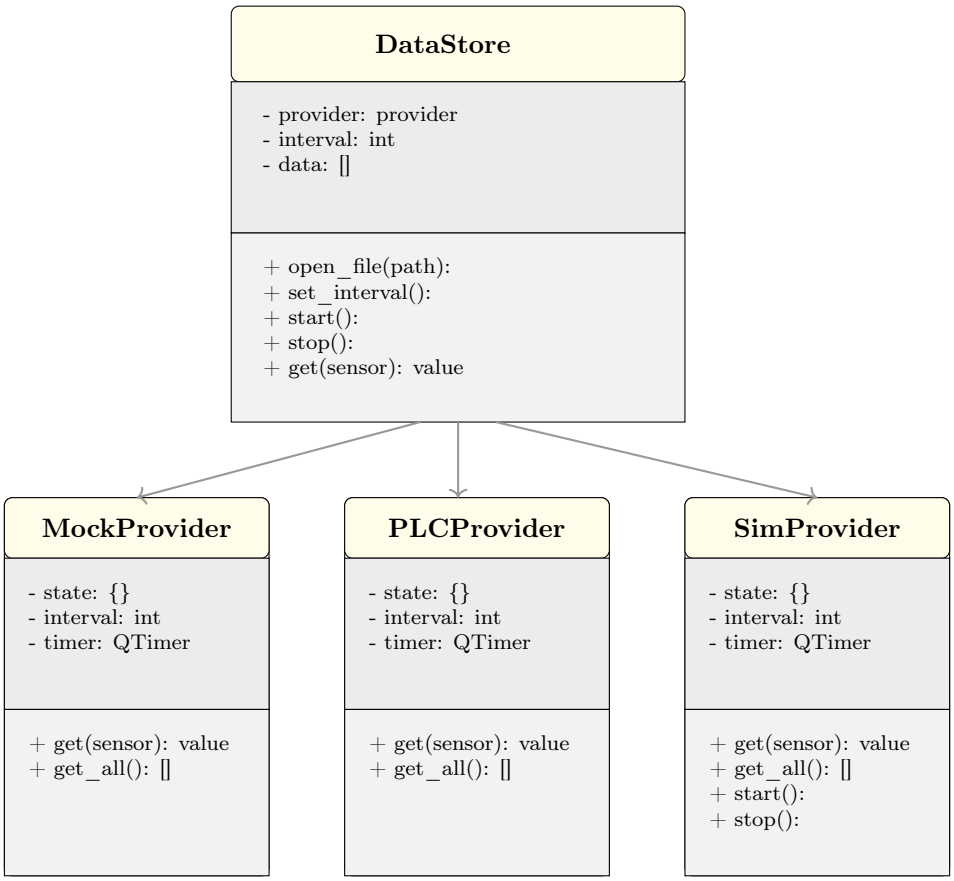


FIGURE 7.5: DataStore and Providers Class Diagram

- **File:** data_store.py
- **Class:** DataStore

Centralizes storage for all sensor data, supports retrieval and processing of sensor data in bulk. Periodically logs data using RepeatTimer to ensure consistency.

The key methods are as follows:

- get(sensor): Fetches data for a specific sensor.
- get_all(): Retrieves the complete dataset.

7.3.3 Data Simulation

- **File:** mock_provider.py
- **Class:** MockProvider

Simulates sensor data in real-time and periodically updates sensor values with random fluctuations. Provides a shutdown mechanism to simulate system downtime.

It in general serves the purpose of acting as a drop-in replacement for live data sources during times when development needs to happen without access to live data from the actual physical PLC.

7.3.4 Diagram implementation

The dashboard displays three diagrams containing schematics of the CCS's three PIDs. To render these schematics, using static images was considered, but this would have decreased the diagram's resolution and required manual setup of interactive regions, such as clickable and hoverable areas, for each sensor.

Instead, the schematic is drawn using `PySide6` components. A generic `Node` class represents these components and can be subclassed to define specific elements such as sensors or valves. This enables the diagram to be rendered programmatically by instantiating generic software models of the hardware, each configured with their respective location, metadata, and possible handlers for events like clicking and hovering.

7.4 Design Choices

- **Modularity** Each system component is implemented as an independent module, ensuring separation of concerns.
- **Extensibility** Supports the addition of new sensors, data providers, and front-end features without significant refactoring.
- **Simulation Support** The MockProvider facilitates development in environments lacking physical hardware. Also at times when you aren't able to connect to the PLC.
- **Scalability** The DataStore and PLC modules are designed to handle a large number of sensors and data streams efficiently.
- **Usability** An early design choice was in regards to implementing the PID's through the painter class, with high visual fidelity and user experience in mind.

Chapter 8

Testing of Product

8.1 Usability Testing with Client

As explained prior, weekly meetings have been conducted with the client. Not much negative feedback has been given to the development team through each meeting. This can be analyzed as a positive takeaway from a well made requirements elicitation phase and understanding the requirements and client needs well.

A final Usability Testing has been conducted on April 15, 2025 with the client. Where the client has been shown the final iteration of the developed product.

The client has mentioned that they are very happy with the implemented product, and that the provided dashboard is a solid foundation (as the scope of the design project doesn't include the whole scope of the actual digital twin dashboard Hovyu wishes to have; due to the 8 week time limit of the project).

We can conclude that the Usability Testing has been passed with the client approving the product and not having any desired specific changes left to be implemented.

8.2 Unit Testing

To test the correctness of the system we implemented a number of unit tests.

Testing is done with a python testing toolset called *pytest*. Some parts of the codebase are inherently hard to test. For instance, because some code may require a real graphical environment to run or a real PLC connection. Or, functions that draw onto the graphical environment and thus have to be tested visually for correctness. However, we have tried to implement as many tests as possible, including tests for nearly all of the data pipelining code.

To monitor the testing progress, we use a tool called *coverage* that monitors which lines of code are being tested. Like mentioned before, some of the code remains untested, but by analyzing the testing coverage we can be aware of the untested portions of the codebase as well as the parts that are already thoroughly tested. This helps us to know what tests to develop next and what code should be modified with extra care.

Using continuous integration tools built into Github, we applied pytest at every commit. This means that commits that fail any tests get marked as such.

The unit tests can be found here in the directory *tests/* within the source code. The tests can be executed using the *pytest* command.

To test the *style* of our code, we use a linting tool called *ruff*. This tool is also run at every commit by the continuous integration. It creates errors when code isn't properly formatted. Because Python

code often leads to properly formatted code by language design, we didn't take the linting errors into too much consideration. But we did occasionally use ruff to automatically fix some errors like removing unused import statements.

Chapter 9

Evaluation

9.1 Conclusion

The goal of this project was to develop a digital twin (a virtual representation of the physical CO_2 capture plant) for CO_2 capture plant being developed by Hovyu together with the commissioned company Coraltech which is building the actual pilot plant. Such a plant plays a crucial role in mitigating climate change, by capturing CO_2 emissions from industrial sources and storing them.

The existing system relied on a LabVIEW script, which was not optimal as its talent pool is limited, meaning development is both time-consuming and costly. To address this, a digital twin was developed using Python, a more widely used language, improving maintainability and reducing both time and budget constraints.

The resulting digital twin features an interactive dashboard featuring the three provided PIDs. The dashboard is drawn using generic software models that represent the physical components of the plant, such as the sensors and the valves. This enables the user to programmatically create diagrams and manage click and hover events directly.

In the digital twin dashboard, the user is able to view all of the sensor data which are rendered next to each sensor node. Clicking on a sensor opens a new windows which includes the graph containing the sensor's historical data, enabling efficient and remote monitoring of the CO_2 capture plant's performance. A more in-depth analysis of the fulfilled requirements is available in the next subsection.

This project demonstrates the feasibility of digital twins in the context of CCS, highlighting the importance of a flexible and maintainable framework. Digital twins improve the plant's operating efficiency and simplify plant monitoring. In the future, the digital twin could include a simulation environment, enabling the user to anticipate errors and identify solutions during runtime. The provided digital twin dashboard will provide a good foundation for Hovyu, for future implementations to be added on top of the current iteration.

With the final usability testing and the feedback from the client, it can be deemed that the project has been concluded with a positive outcome, where the product meets the expectations of the client. We hope that this digital twin dashboard and its future development by Hovyu, will contribute to lessening the effects of climate change and provide a better ecosystem for all of us.

9.1.1 Fulfilled Requirements

The requirements that have been completed, have been added with a check-mark as follows.

Functional Requirements

Must Have (M)

M1 The system must have a graphical user interface (GUI) that visualizes the physical hardware of the system. ✓
M2 The system must display the hardware's sensor readings in the GUI. ✓
M3 The system must have a mechanism that allows the user to change a sensor's value.
M4 The system must have a mechanism that allows the user to change the rate at which the system logs a set of variables. ✓
M5 The system must have an emergency button to shut down the system by switching all the equipment off. ✓
M6 The system must have a button to switch between the different pages of the PID. ✓

Should Have (S)

S1 The system should replace LabView's software fully, operating as a standalone application, deprecating LabView (as the client prefers a language which is more widely used for ease of work in the long term). ✓
S2 The system should display real-time data plots & graphs for the system's sensors (temperature, pressure, etc.). ✓
S3 The system should have upper and lower bounds / margins of error for sensors to allow for normal fluctuation around the set threshold.
S4 The system must visualize text boxes above the sensors in the GUI that define thresholds, minimum and maximum values for proper functioning of the system.

Could Have (C)

C1 The system could react to parameters changing by simulating what would happen to the physical hardware (if disconnected).
C2 The system could implement a stable condition identifier, achieved when the readings of some sensors remain within a specified threshold.
C3 The system could have a function that re-calibrates a sensor based on a minimum and maximum to improve the accuracy of an older sensor.

Won't Have (W)

W1 The system won't replace Siemens' proprietary technology to connect to PLC, avoiding the need to make use of costly software licenses.

Quality Requirements

Quality Requirements (Q)	
Q1	The system must poll sensor data at a minimum frequency of once every 1000 milliseconds to ensure near real-time responsiveness in the GUI. ✓
Q2	The graphical user interface must update sensor readings and visualizations within 1000 milliseconds of receiving new data to ensure smooth user interaction. ✓
Q3	Upon pressing the emergency shutdown button, the system must execute the shutdown procedure within 1 second.
Q4	Logged data must be written to storage within 1 second of collection to avoid data loss during continuous operation. ✓
Q5	The system must be ready for operation (including GUI and sensor polling) within 5 seconds of launch. ✓
Q6	The system must be distributable as a standard file format such as .exe that can run on a target machine without requiring external dependencies such as Python. ✓

As seen above, all of the "Must Level" requirements have been fulfilled with the exception of requirement **M4**. Perhaps this would have been better to reiterate early-on that this wouldn't be a Must Level priority, as the client later on deemed it wasn't necessary. All other Must level requirements have been completed, which shows the project is a success, together with the positive feedback about the usability of the product by the client.

Two of the "Should Level" priority requirements, namely **S1** and **S2**, have been fulfilled. This shows the development team has identified the amount of requirements well early on, which led to fulfilling more niche requirements.

None of the "Could Level" priority requirements have been fulfilled due to time constraints.

It can be evaluated that the development team has done a good job of creating the digital twin dashboard up to the standards of the client, with the requirements that were necessary being met.

An in-depth step by step user manual has been created for the client and the reader of the report. While the design itself has been made intuitive, the development team thought it would be a good addition. It can be viewed in Appendix B.

9.2 Discussion

9.2.1 Collaboration within the Team

Workload has been divided between the team members as follows.

- **Eren Kodal:** Significant contribution by implementing UI features and integrating PID1, PID2, and PID3 into the dashboard. Took the lead in writing the majority of the report, creating the poster, and contributing to the final presentation. Also completed the reflection assignment. Their contributions were acknowledged with a +1 point on the project grade.
- **Yannick Krijnen:** Significant contribution by implementing graph functionality and enhancing the maintainability of the codebase. Assisted in writing the report and actively contributed to the final presentation.
- **Rein Fernhout:** Significant contribution by designing and developing the aggregation and pipelining of sensor readings, developing the sensor pop-up window and multiple refactors of the codebase.
- **Arnout Luinge:** Significant contribution by leading efforts in testing and implementing PLC communication, including configuring, reading, writing, and polling PLC-connected sensors and actuators. Developed the PLC widget and established the PLC-UI connection. Contributed to writing the report.

- **Stefan Morriën:** Significant contribution by working on implementing general UI features and functionality, such as displaying node values, setting update frequencies, and calibration. Also developed unit tests and contributed to writing the report.

We believe everyone in the team has worked efficiently and this in return resulted into the product which has met the expectations of the client.

9.2.2 Recommendations

Potential Use of SysML

The implementation of the diagram using generic software models of the hardware (see Section 7.3.4) enabled a scalable and interactive representation of the PID controller schematic. However, this implementation still requires tedious and error prone manual labor when defining the component layout and interactivity.

To improve this, Systems Modeling Language (SysML) could be used to model the structure and behavior of the hardware. SysML is an extension of the Unified Modeling Language (UML), a widely recognized modeling standard. While UML gained popularity with software engineering, it did not yet meet the requirements of system engineers, who missed key aspects in the language such as the ability to model both hardware and software and their behavior together [9]. SysML addresses this gap by expanding the UML framework to accommodate for the needs of system engineers.

Hovyu already utilizes models for their CCS, but these models are primarily intended for human interpretation, where the viewer might resolve inconsistencies in the way the hardware is modeled visually [10]. By adapting these models to be machine-recognizable, the upfront effort of creating the models increases, but the same models can then be utilized to automatically generate code and transfer the behavior of the hardware into a software framework [10].

This means changes in the design are able to be reflected immediately in the software framework without manual work, and the behavior does not have to be redefined in the code.

Bibliography

[1] *Climate Change: Atmospheric Carbon Dioxide | NOAA Climate.gov*, en, Apr. 2024. [Online]. Available: https://www.climate.gov/news-features/understanding-climate/climate-change-atmospheric-carbon-dioxide (visited on 03/19/2025).

[2] K. Goto, K. Yogo, and T. Higashii, "A review of efficiency penalty in a coal-fired power plant with post-combustion CO2 capture," *Applied Energy*, vol. 111, pp. 710–720, Nov. 2013, ISSN: 0306-2619. DOI: 10.1016/j.apenergy.2013.05.020. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0306261913004212 (visited on 03/19/2025).

[3] D. M. Botín-Sanabria, A.-S. Mihaita, R. E. Peimbert-García, M. A. Ramírez-Moreno, R. A. Ramírez-Mendoza, and J. d. J. Lozoya-Santos, "Digital Twin Technology Challenges and Applications: A Comprehensive Review," en, *Remote Sensing*, vol. 14, no. 6, p. 1335, Jan. 2022, Number: 6 Publisher: Multidisciplinary Digital Publishing Institute, ISSN: 2072-4292. DOI: 10.3390/rs14061335. [Online]. Available: https://www.mdpi.com/2072-4292/14/6/1335 (visited on 03/19/2025).

[4] M. Grieves, "Digital twin: Manufacturing excellence through virtual factory replication," 2014. [Online]. Available: https://orkg.org/resource/R154580.

[5] F. Tao, H. Zhang, A. Liu, and A. Y. C. Nee, "Digital Twin in Industry: State-of-the-Art," *IEEE Transactions on Industrial Informatics*, vol. 15, no. 4, pp. 2405–2415, Apr. 2019, ISSN: 1941-0050. DOI: 10.1109/TII.2018.2873186. [Online]. Available: https://ieeexplore.ieee.org/document/8477101 (visited on 04/15/2025).

[6] P. Bielka, "The Future of Carbon Capture and Storage Technology: Innovative Approach with Digital Twin," en, *Journal of Geotechnology and Energy*, vol. 40, no. 2, pp. 5–10, Jun. 2023, Number: 2, ISSN: 2720-3581. DOI: 10.7494/jge.2023.40.2.5413. [Online]. Available: https://journals.agh.edu.pl/jge/article/view/5413 (visited on 04/14/2025).

[7] S. Guo, W. Le, M. Mehmet, H. Jiang, and C. Hale, *Development and Validation of a Digital Twin for the Abb-Imperial College London Carbon Capture Pilot Plant*, en, SSRN Scholarly Paper, Rochester, NY, Jan. 2024. DOI: 10.2139/ssrn.4701352. [Online]. Available: https://papers.ssrn.com/abstract=4701352 (visited on 04/14/2025).

[8] G. Molenaar, *Python wrapper for the snap7 library*, https://pypi.org/project/python-snap7/, [Accessed 12-04-2025].

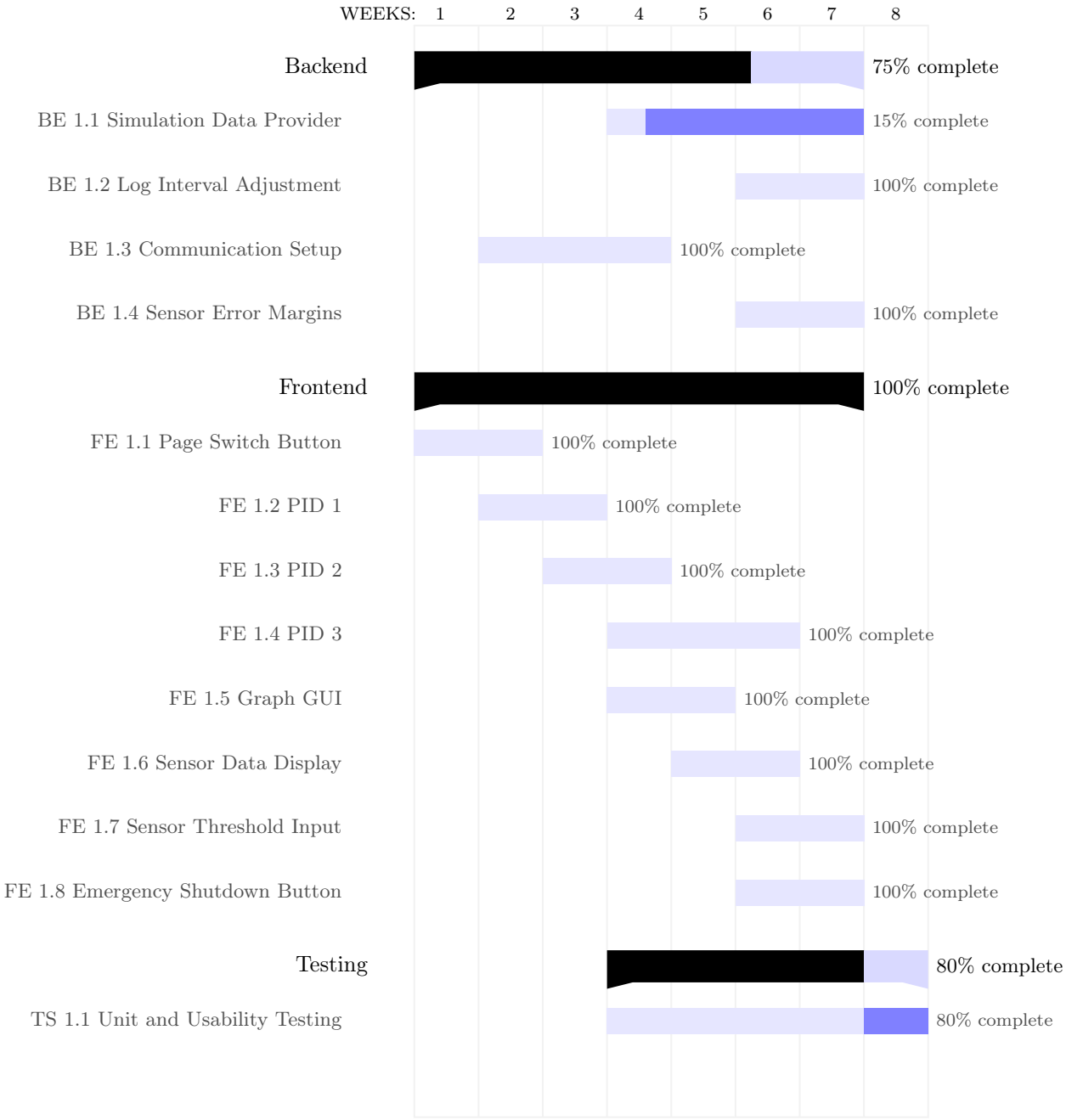
[9] *(PDF) The OMG Modelling Language (SYSML)*, en. [Online]. Available: https://www.researchgate.net/publication/260359652_The_OMG_Modelling_Language_SYSML (visited on 04/10/2025).

[10] F. Wilking, C. Sauer, B. Schleich, and S. Wartzack, "SysML 4 Digital Twins – Utilization of System Models for the Design and Operation of Digital Twins," en, *Proceedings of the Design Society*, vol. 2, pp. 1815–1824, May 2022, ISSN: 2732-527X. DOI: 10.1017/pds.2022.184. [Online]. Available: https://www.cambridge.org/core/journals/proceedings-of-the-design-society/article/sysml-4-digital-twins-utilization-of-system-models-for-the-design-and-operation-of-digital-twins/2B934D8367F74EA1E2CB6789096EA268 (visited on 04/09/2025).

Appendices

Appendix A

Planning



Appendix B

User Manual

A user manual has also been done, in regards to the functionality of the dashboard for the Digital Twin. The step by step demonstration of the functionality as follows. Please keep in mind that PID functionality can not be shown here due to confidentiality issues. Such as interacting with the elements in the PID and hovering over the elements (which is also majority of the application).

When you launch the dashboard, you will initially be disconnected from the PLC. As shown below.

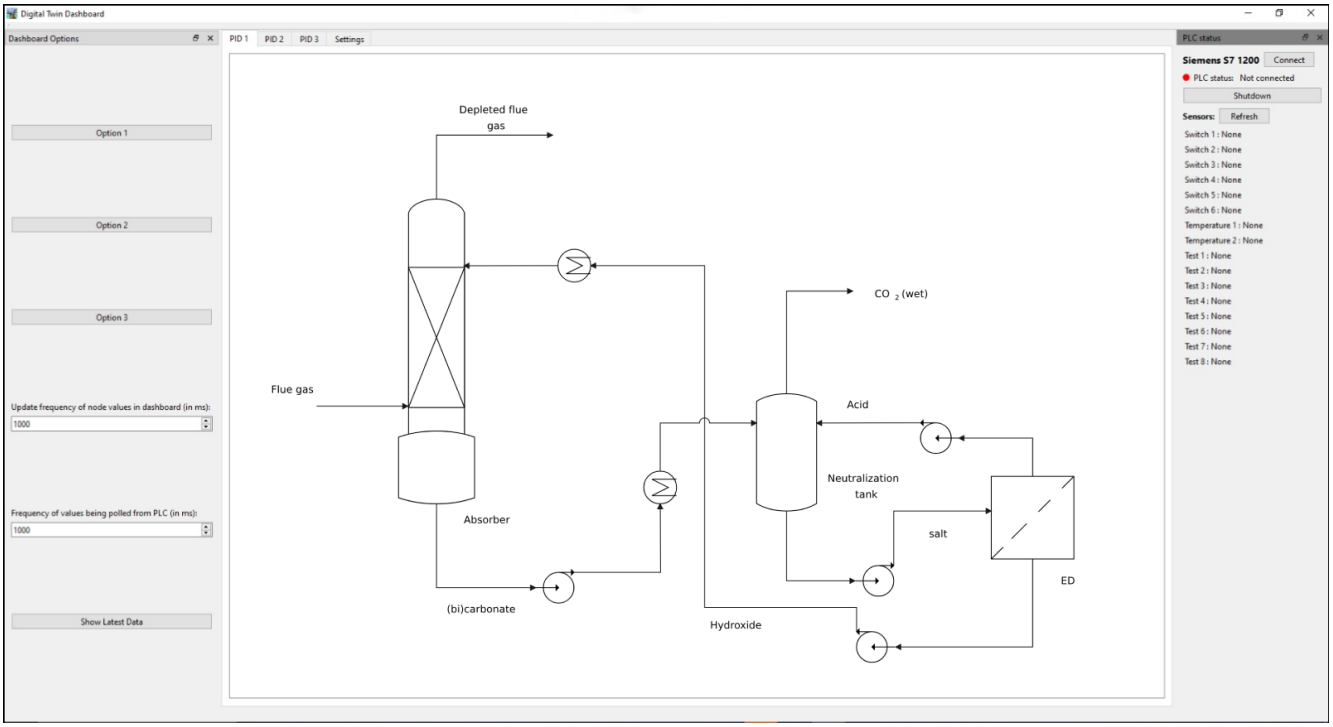


FIGURE B.1: Dashboard launched and not connected to the PLC

You can press on the Connect button on the right widget, next to the name of the PLC. If you can't connect, it will give the following error.

You can make the details show extended, for more information such as shown below.

When you solve the connection error, or if there are no issues to begin with; when you click connect. You will be connected as follows.

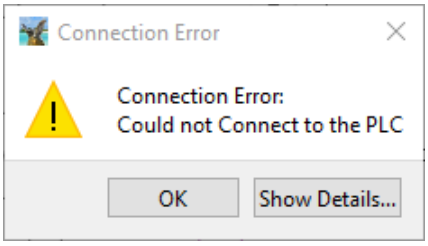


FIGURE B.2: Error connecting to the PLC

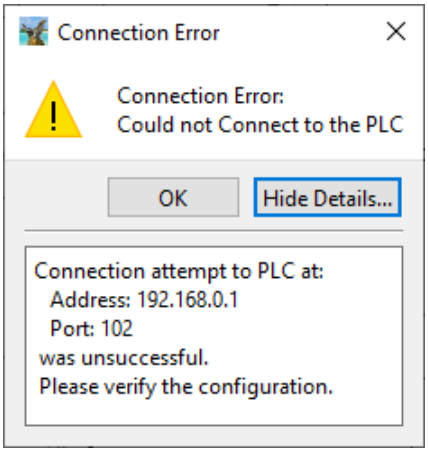


FIGURE B.3: Error connecting to the PLC expanded

After you are connected, you can move through the PIDs 1-2-3 as you click on them on the UI. You can hover over and interact with the PID elements. The PID pages can not be shown in the report due to confidentiality as previously stated. When you interact with a sensor, you can choose to calibrate it.

As seen on the left side of the widget you can choose where to get the data from (Mock which is random data generation for, PLC for real-time data or Simulation which is not implemented due to being outside of the scope, agreed by the client)

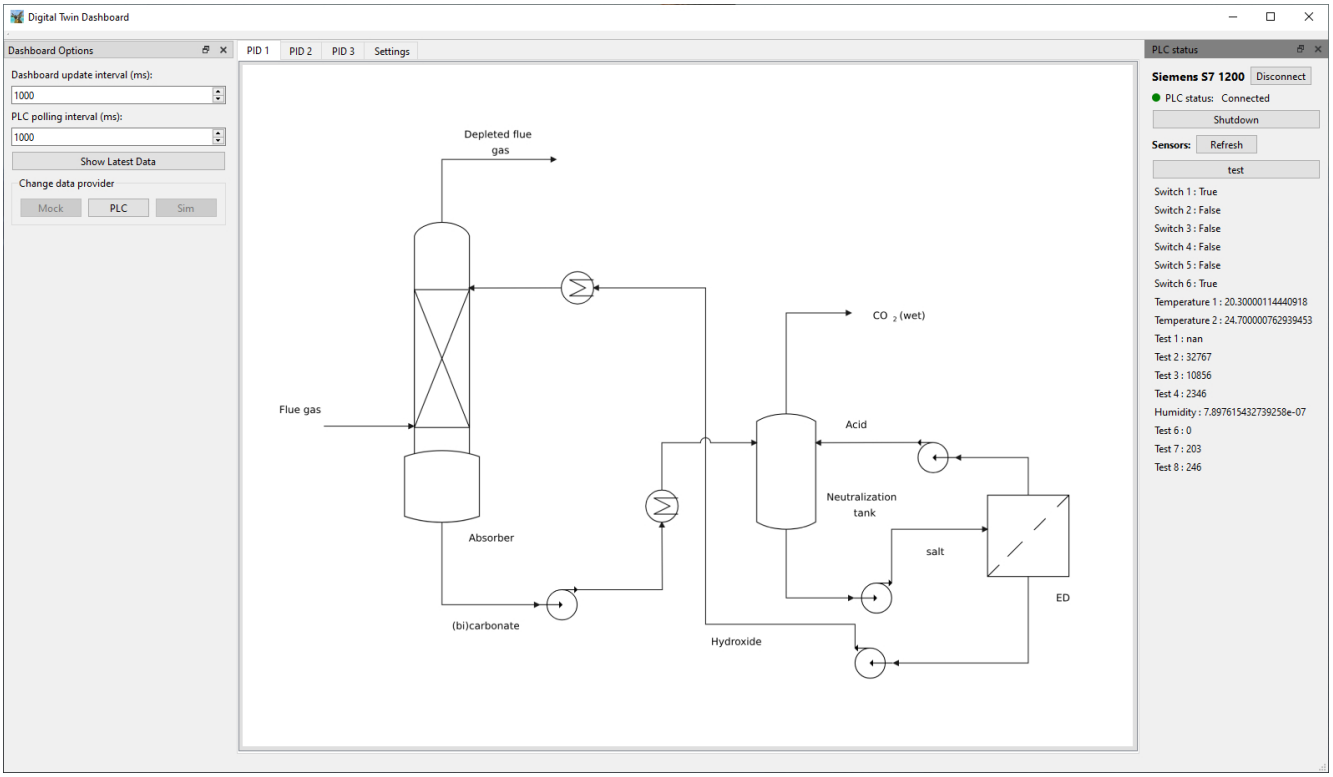


FIGURE B.4: Dashboard launched and not connected to the PLC

As one of the core functionalities, you can also shutdown the CCS plant. When you press shutdown as seen on the widget on the right side, you will get the following pop-up.

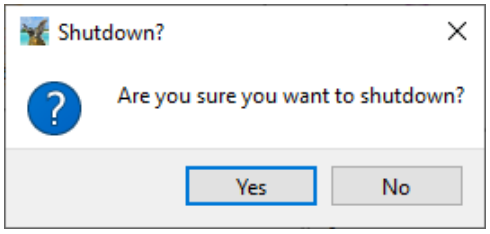


FIGURE B.5: Shutdown pop-up for confirmation

Clicking on yes will result in shutting down of the plant.

Within the dashboard you can press on the sensors and open the graphs related to them. These graphs open on a separate window as seen below. You can also calibrate the sensor as necessary.

The widgets have been made that you can resize them, close and change their locations according to your own preferences.

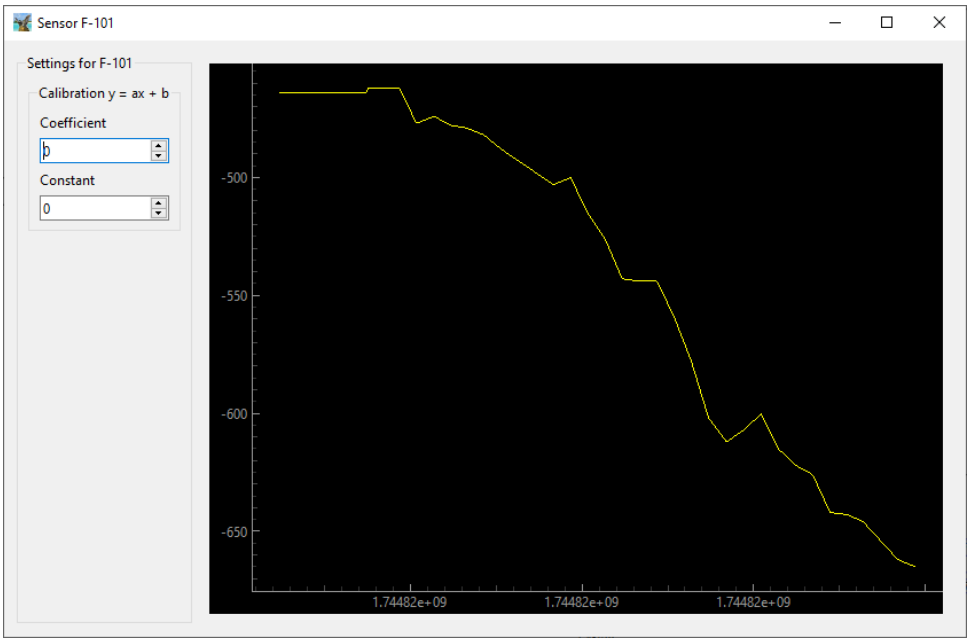


FIGURE B.6: Graph pop-up with calibration option for the sensor

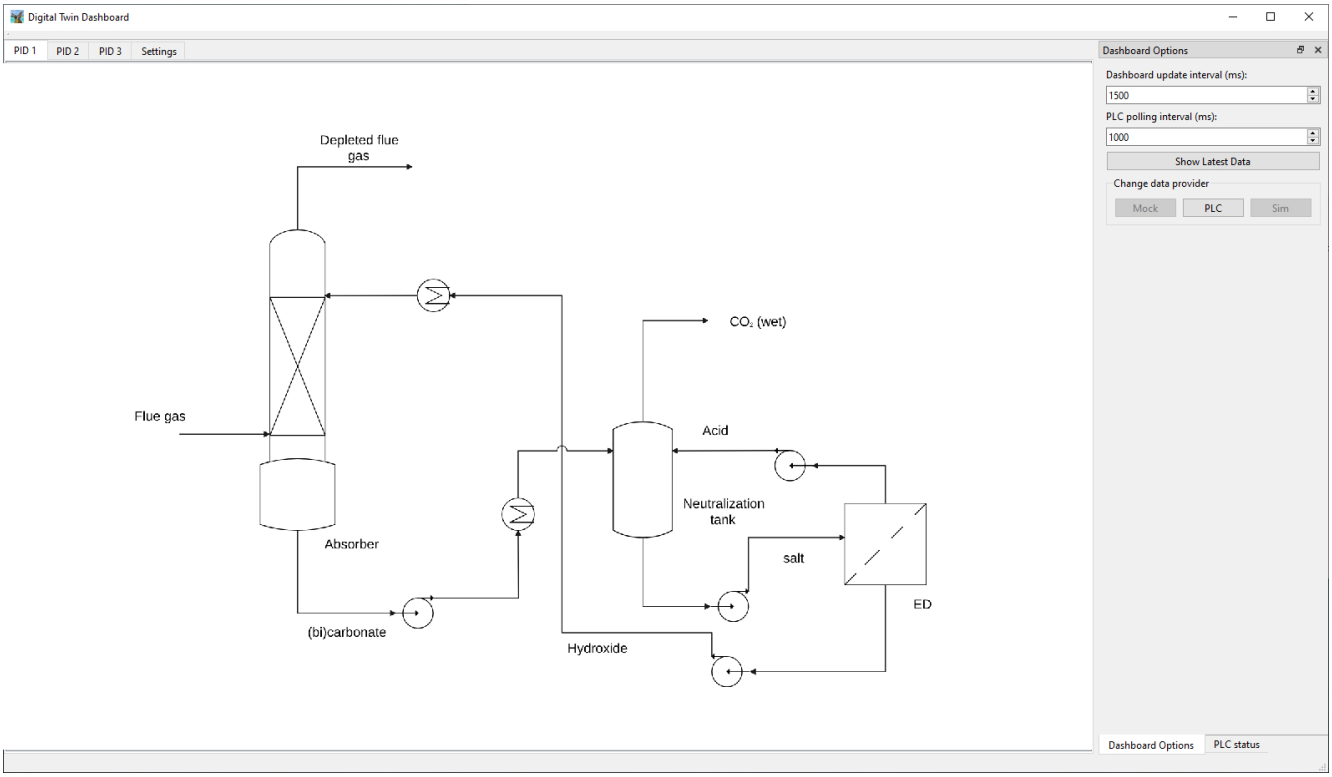


FIGURE B.7: One widget closed, one widget moved place (as an example of user QoL improvement)