UNIVERSITY OF VERONA

Department of COMPUTER SCIENCE

Master's Degree in
COMPUTER SCIENCE AND ENGINEERING

Master Thesis

Towards Process Comprehension of Industrial Control Systems: a Framework for Analyzing Industrial Systems

Candidate:
Marco OLIANI
VR457249

Supervisor:
Prof. **Massimo MERRO**

Co-supervisor:
Prof. **Ruggero LANOTTE**
University of Insubria

Academic Year 2022/2023

"If you spend more on coffee than on IT security, you will be hacked. What's more, you deserve to be hacked"
(Richard Clarke)

Abstract

Bla bla bla

Contents

Chapter 1

Introduction

LOREM ipsum dolor bla bla bla. Ma dove metto l'abstract? Prova di interlinea che direi posso anche andare bene, ma bisogna poi vedere il tutto come si incastra alla fine, in modo da ottenere un bel risultato alla vista.

1.1 Contribution

The primary objective of this thesis was originally to validate a specific methodology designed to attain process comprehension in an Industrial Control System (ICS) when applied to a real-world scenario. This methodology, which adopts a blackbox approach, involves dynamic analysis of the physical process and network communications of the system. The ultimate goal of this methodology is to accurately derive a comprehensive understanding of the industrial process.

To accomplish this objective, a framework developed by the authors of the methodology is utilized. This framework incorporates a series of analysis steps that are employed to facilitate the application of the methodology. The entire framework and methodology are then applied to a specially implemented virtualized testbed, providing a controlled environment for testing and evaluation purposes and to facilitate the process of achieving process comprehension of an Industrial Control System.

As the thesis progressed, it quickly became evident that both the framework and the methodology employed had certain limitations. In response, an extensive restructuring of the framework was conducted, aimed at enhancing and expanding its existing features while introducing new ones.

Hence, the main contribution of the thesis is now to enhance the original methodology by refining the framework and reassessing the approach for each analysis step. The aim is to achieve a more complete and exhaustive process comprehension. Subsequently, the improved methodology will be tested on a different and more complex case study, distinct from the virtualized testbed, to validate its effectiveness in a real-world scenario.

1.2 Outline

The thesis is structured as follows:

Chapter 2: provides a background on Industrial Control Systems, (ICSs) describing their structure, components, and some of the network communication protocols used;

Chapter 3: following an introductory section that provides a brief overview of the existing literature on process comprehension in industrial control systems, this chapter focuses on a specific paper that outlines a methodology to attaining process comprehension of an industrial system by employing dynamic blackbox analysis;

Chapter 4: outlines a proposal to improve and extend the methodology outlined in the previous chapter;

Chapter 5: presents the case study on which the proposed methodology will be applied;

Chapter 6: shows how the proposed methodology is applied to the case study illustrated above;

Chapter 7: outlines final conclusions and future work.

Chapter 2

Background on Industrial Control Systems

INDUSTRIAL CONTROL SYSTEMS (ICSs) are information systems used to control industrial processes such as manufacturing, product handling, production, and distribution [1]. ICSs are often found in critical infrastructure facilities such as power plants, oil and gas refineries, and chemical plant ICSs are different from traditional IT systems in several key ways.

Firstly, ICSs are designed to control physical processes, whereas IT systems are designed to process and store data. This means that ICSs have different requirements for availability, reliability, and performance. Secondly, ICSs are typically deployed in environments that are harsh and have limited resources, such as extreme temperatures and limited power. Thirdly, the protocols hardware and software used in ICSs are often proprietary.

ICSs are becoming increasingly connected to the internet and other networks, which has led to increased concerns about their security. Industrial systems were not originally designed with security in mind, and many of them have known vulnerabilities that could be exploited by attackers. Additionally, the use of legacy systems and equipment can make it difficult to implement security measures. As a result, ICSs are increasingly seen as a potential target for cyber attacks, which could have serious consequences for the safe and reliable operation of critical infrastructure: some notorious examples of cyber attacks are *(i)* the **STUXnet** worm [2], which purpose was to sabotage the nuclear centrifuges of the enrichment plant at the Natanz nuclear facility in Iran; *(ii)* **Industroyer** [3], also referred as *Crashoverride*, responsible for the attack on the Ukrainian power grid on December 17, 2016; *(iii)* the attack on February, 2021 to a water treatment plant in Oldsmar, Florida [4], where the level of sodium hydroxide was intentionally increased to a level approximately 100 times higher than normal.

The increasing connectivity of ICSs and the associated security risks have led to a growing interest in the field of ICS security. Researchers and practitioners are working to develop new security technologies, standards, and best practices to protect ICSs from cyber attacks. This includes efforts to improve the security of ICS networks and devices, as well as the development of new monitoring and detection techniques to identify and respond to cyber attacks.

Table 2.1 summarizes the differences between traditional IT and ICSs [5]:

	Traditional IT	**ICSs**
Focus	Data	Asset
Update Frequency	High	Low
Priority	Confidentiality Integrity Availability	Availability Integrity Confidentiality
Operating System	Standardized	Proprietary
Protocols	Standardized	Proprietary
Attacker Motivation	Monetization	Disruption

Table 2.1: *differences between Information Tecnology (IT) and Industrial Control Systems (ICSs)*

2.1 Industrial Control Systems Architecture

In the past, there has been a clear division between *Information Technology* (IT) and *Operational Technology* (OT), both at the technical and organizational levels. Each domain has maintained its own distinct technology stacks, protocols, and standards. However, with the emergence of Industry 4.0 and the rapid expansion of industrial automation, which heavily relies on IT tools for monitoring and controlling critical infrastructures, the boundary between IT and OT has started to blur. This trend has paved the way for greater integration between these two domains, thus improving productivity and process quality.

General ICS architecture consists in **six levels** each representing a functionality: this architecture comprising the OT and IT parts is represented in Figure 2.1 [6][5], according to the *Purdue Enterprise Reference Architecture* (PERA), or simply **Purdue Model**:

- Level 0 (**Processes**, or **Field I/O Devices**): contains **field devices**.
- Level 1 (**Intelligent Devices**, or **Controller Network**): includes **local or remote controllers** that sense, monitor and control the physical process, such as **PLCs** (2.2.2.1) and **RTUs** (2.2.2.2). Controllers interface directly to the field devices reading data from sensors and sending commands to actuators.
- Level 2 (**Control Systems**, or **Area Control**): contains computer systems used to supervising and monitoring the physical process: they provide a **Human-Machine Interface** (*HMI*, 2.2.3.2) and *Engineering Workstations* (EW) for operator control.
- Level 3 (**Manufactoring/Site Operations**, or **Operations/Control**): comprises systems used to manage the production workflow for plant-wide control: they collate informations from the previous levels and store them in Data Historian servers.

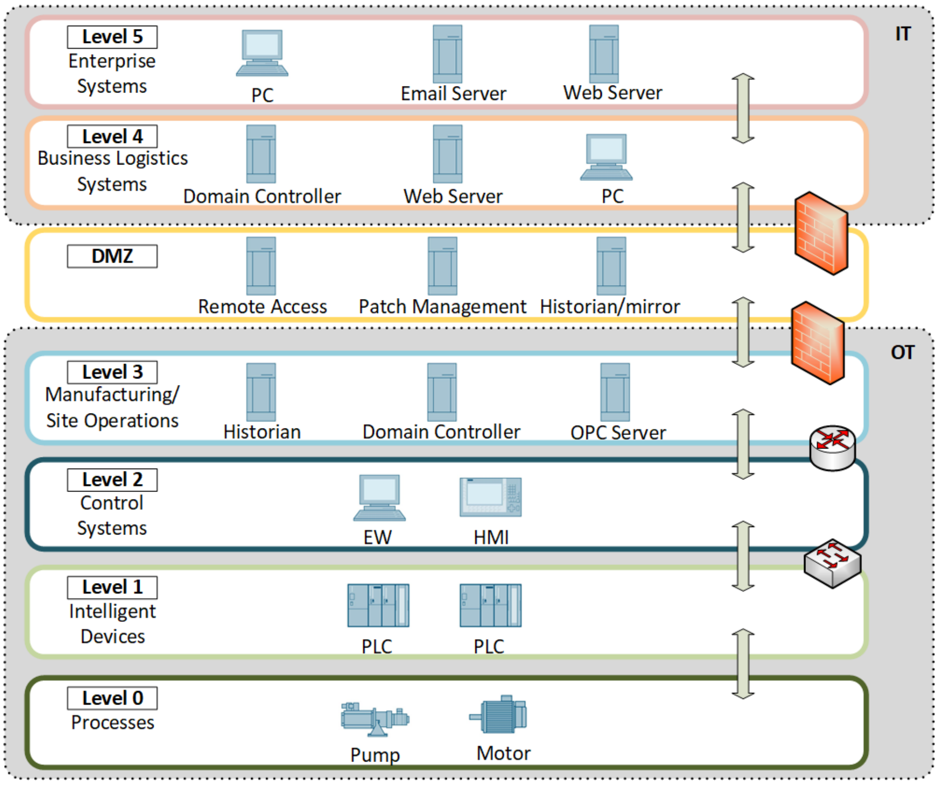


Figure 2.1: ICS architecture schema

- **Industrial Demilitarized Zone** (DMZ): intermediate level that connects the *Operational Technology* (OT) part (levels 0-3) with the *Information Technology* (IT) part of the system (levels 4 and 5). Communication takes place indirectly through services such as *proxy servers* and *remote access servers*, which act as intermediaries between the two environments.

- Level 4 (**Business Logistics Systems**, or **Business Planning/Logistics**): collect and aggregates data from the Manufactoring/Site Operations level overseeing the IT-related activities to generate **reporting** to the Enterprise System layer. At this layer we can find application and e-mail servers, and *Enterprise Resource Planning* (ERP) systems.

- Level 5 (**Enterprise Systems**): represents the enterprise network, used for the business-to-business activities and for business-to-client purpose services. At Enteprise Systems level are typical IT services such as mail servers, web servers and all the systems used to manage the ongoing process.

As previously discussed, the gap between IT and OT is steadily narrowing. Nowadays, it is increasingly common to encounter IT elements within the OT realm. For example, desktop PCs are now frequently found in OT environments, and industrial devices are interconnected using standard IT communication protocols like TCP and UDP.

2.2 Operational Technology Networks

Operational Technology primarily encompasses the **tangible aspects** of Industrial Control Systems and directly interfaces with the physical processes of the monitored systems. Its main purpose is to **manage and control the procedures** involved in creating and correcting physical value in various equipment.
This section will focus on the key aspects and components of Operational Technology network, with specific reference to the first four levels of the Purdue model previously seen.

2.2.1 Field I/O Devices Layer

This level concerns all aspects related to the physical environment and the physical elements that are part of it, which have the ability to actively influence the environment.
These physical elements are represented by **Field Devices**, i.e., **sensors** and **actuators** used to collect data from the process and control it: sensors are the elements responsible for reading specific values related to the physical environment (e.g., the level of a liquid), while actuators change its behavior and characteristics (e.g., opening or closing a valve to make the liquid flow).
Examples of field devices include temperature sensors, pressure sensors, valves and pumps.

2.2.2 Controller Network Layer

Controller Network layer includes devices that handle data from and to the *Field I/O Devices* layer. This kind of device is capable of gathering data from sensors, updating its internal state, and activating actuators (for example opening or close a pump that controls the level of a tank), making decisions based on a customized program, known as its control logic.
Commonly found within this layer are *Programmable Logic Controllers* (PLCs) and *Remote Terminal Units* (RTUs): in the upcoming sections, we will examine these elements in detail.

2.2.2.1 Programmable Logic Controllers

A *Programmable Logic Controller* (PLC) is a **small and specialized industrial computer** having the capability of controlling complex industrial and manifacturing processes [7].

Compared to relay systems and personal computers, PLCs are optimized for control tasks and industrial environments: they are rugged and designed to withdraw harsh conditions such as dust, vibrations, humidity and temperature: they have more reliability than personal computers, which are more prone to crash, and they are more compact a require less maintenance than a relay system.
Furthermore, I/O interfaces are already on the controller, so PLCs are easier to expand with additional I/O modules (if in a rack format) to manage more inputs and ouputs, without reconfiguring hardware as in relay systems when a reconfiguration occours.

PLCs are more *user-friendly*: they are not intended (only) for computer programmers, but designed for engineers with a limited knowledge in programming languages: control program can be entered with a simple and intuitive language based on logic and switching operations instead of a general-purpose programming language (*i.e.* C, C++, ...).

PLC Architecture The basic hardware architecture of a PLC consists of these elements [8]:

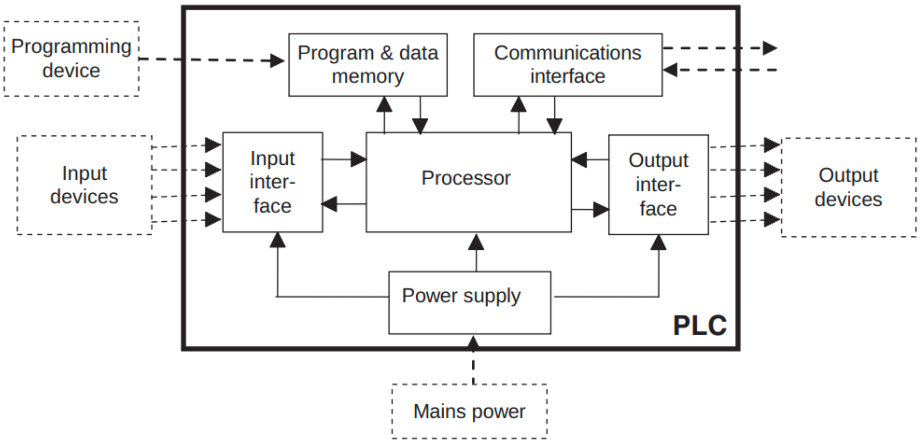


Figure 2.2: *PLC architecture*

- **Processor unit (CPU)**: contains the microprocessor. This unit interpretes the input signals from I/O modules, executes the control program stored in the Memory Unit and sends the output signals to the I/O Modules. The processor unit also sends data to the Communication interface, for the communication with additional devices.
- **Power supply unit:** converts AC voltage to low DC voltage.
- **Programming device:** is used to store the required program into the memory unit.
- **Memory Unit:** consists in RAM memory and ROM memory. RAM memory is used for storing data from inputs, ROM memory for storing operating system, firmware and user program to be executed by the CPU.
- **I/O modules:** provide interface between sensors and final control elements (actuators).
- **Communications interface:** used to send and receive data on a network from/to other PLCs.

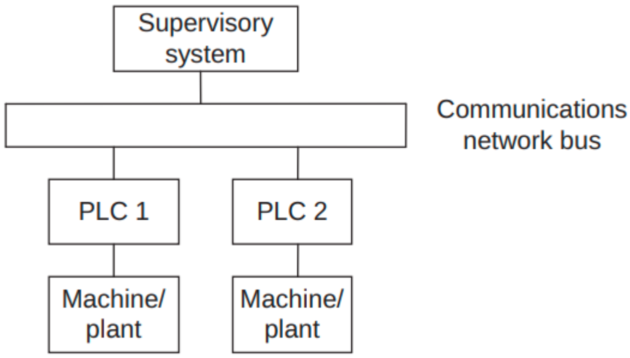


Figure 2.3: *PLC communication schema*

PLC Programming Two different programs are executed in a PLC: the **operating system** and the **user program**.

The operating system tasks include executing the user program, managing memory areas and the *process image table* (memory registers where inputs from sensors and outputs for actuators are stored).

The user program needs to be uploaded on the PLC via the programming device and runs on the process image table in *scan cycles*: each scan is made up of three phases [9]:

1. reading inputs from the process images table
2. execution of the control code and computing the physical process evolution
3. writing output to the process image table to have an effect on the physical process. At the end of the cycle, the process image table is refreshed by the CPU

Standard PLCs **programming languages** are basically of two types: **textuals** and **graphicals**. Textual languages include languages such as *Instruction List* (IL) and *Structured Text* (ST), while *Ladder Diagrams* (LD), *Function Block Diagram* (FBD) and *Sequential Function Chart* (SFC) belong to the graphical languages.

Graphical languages are more simple and immediate comparing to the textual ones and are preferred by programmers because of their features and simplicity, in particular the **Ladder Logic programming** (see Figure 2.4 for a comparison).

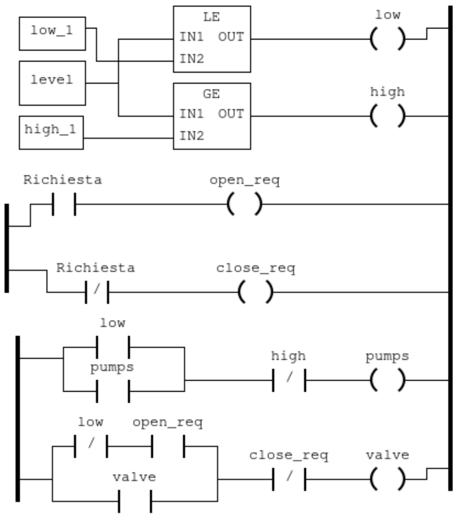
```
PROGRAM PLC1
  VAR
    level AT %IW0 : INT;
    Richiesta AT %QX0.2 : BOOL;
    request AT %IW1 : INT;
    pumps AT %QX0.0 : BOOL;
    valve AT %QX0.1 : BOOL;
    low AT %MX0.0 : BOOL;
    high AT %MX0.1 : BOOL;
    open_req AT %MX0.3 : BOOL;
    close_req AT %MX0.4 : BOOL;
    low_1 AT %MW0 : INT := 40;
    high_1 AT %MW1 : INT := 80;
  END_VAR
  VAR
    LE3_OUT : BOOL;
    GE7_OUT : BOOL;
  END_VAR

  LE3_OUT := LE(level, low_1);
  low := LE3_OUT;
  GE7_OUT := GE(level, high_1);
  high := GE7_OUT;
  open_req := Richiesta;
  close_req := NOT(Richiesta);
  pumps := NOT(high) AND (low OR pumps);
  valve := NOT(close_req) AND (open_req AND NOT(low) OR valve);
END_PROGRAM


CONFIGURATION Config0

  RESOURCE Res0 ON PLC
    TASK task0(INTERVAL := T#20ms,PRIORITY := 0);
    PROGRAM instance0 WITH task0 : PLC1;
  END_RESOURCE
END_CONFIGURATION
```

(a) *Example of ST programming*



(b) *Example of Ladder Logic*

Figure 2.4: *Comparison between ST language and Ladder Logic*

PLC Security PLCs were originally designed to operate as closed systems, not connected and exposed to the outside world via communication networks: the question of the safety of these systems, therefore, was not a primary aspect. The advent of Internet has brought undoubted advantages, but has introduced problems relating to the safety and protection of PLCs from external attacks and vulnerabilities.

Indeed, a variety of different communication protocols used in ICSs are designed to be efficient in communications, but do not provide any security measure i.e. confidentiality, authentication and data integrity, which makes these protocols vulnerable against many of the IT classic attacks such as *Replay Attack* or *Man in the Middle Attack*.

Countermeasures to enhance security in PLC systems may include [10]:

- protocol modifications implementing **data integrity**, **authentication** and **protection** against *Replay Attacks*
- use of *Intrusion Detection and Prevention Systems* (IDP)
- creation of *Demilitarized Zones* (DMZ) on the network

In addition to this, keeping the process network and Internet separated, limiting the use of USB devices among users to reduce the risks of infections, and using strong account management and maintenance policies are best practices to prevent attacks and threats and to avoid potential damages.

2.2.2.2 Remote Terminal Units

Remote Terminal Units (RTUs) are computers with radio interfacing similar to PLCs: they transmit telemetry data to the control center or to the PLCs and use messages from the master supervisory system to control connected objects [11].

The purpose of RTUs is to operate efficiently in remote and isolated locations by utilizing wireless connections. In contrast, PLCs are designed for local use and rely on high-speed wired connections. This key difference allows RTUs to conserve energy by operating in low-power mode for extended periods using batteries or solar panels. As a result, RTUs consume less energy than PLCs, making them a more sustainable and cost-effective option for remote operations.

Industries that require RTUs often operate in areas without reliable access to the power grid or require monitoring and control substations in remote locations. These include telecommunications, railways, and utilities that manage critical infrastructure such as power grids, pipelines, and water treatment facilities. The advanced technology of RTUs allows these industries to maintain essential services, even in challenging environments or under adverse weather conditions.

2.2.3 Area Control Layer

The Area Control layer encompasses hardware and software systems useful for supervising, monitoring and controlling the physical process, driving the behavior of the entire infrastructure. The layer includes systems such as *Supervisory Control and Data Acquisition* (SCADA), *Distributed Control Systems* (DCSs), that perform SCADA functions but are usually deployed locally, and engineer workstations.

2.2.3.1 Supervisory Control And Data Acquisition

Supervisory Control And Data Acquisition (**SCADA**) is a system of software and hardware elements that allows industrial organizations to [12]:

- Control industrial processes locally or at remote locations;
- Monitor, gather, and process real-time data;
- Directly interact with devices such as sensors, valves, pumps, motors, and more through human-machine interface (HMI) software;
- Record and aggregate events to send to historian server.

The SCADA software processes, distributes, and displays the data, helping operators and other employees analyze the data and make important decisions.

2.2.3.2 Human-Machine Interface

The *Human-Machine Interface* (HMI) is the hardware and software interface that operators use to monitor the processes and interact with the ICS. A HMI shows the operator and authorized users information about system status and history; it also allows them to configure parameters on the ICS such as set points and, send commands and make control decisions [13].
The HMI can be in the form of a physical panel, with buttons and indicator lights, or PC software as shown in Figure 2.5.

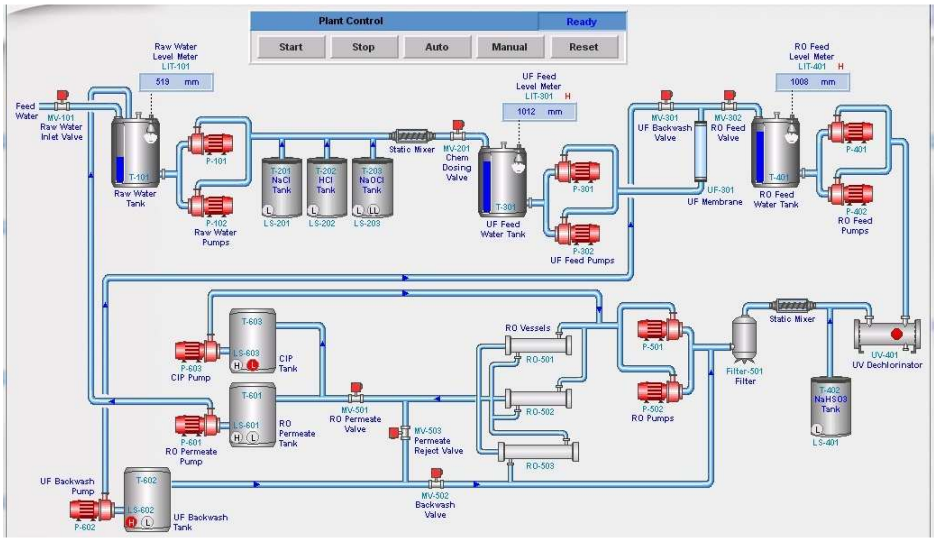


Figure 2.5: *Example of HMI for a water treatment plant*

2.2.4 Operations/Control Layer

Within this zone, there are specialized OT devices that are utilized to manage production workflows on the shop floor [14]. These devices include:

- *Manufacturing Operations Management* (MOM) systems, which are responsible for overseeing production operations.
- *Manufacturing Execution Systems* (MES), which collect real-time data to optimize production processes.
- *Data Historians*, which store process data and, in modern solutions, analyze it within its contextual framework.

2.2.5 Demilitarized Zone

This zone comprises security systems like firewalls, proxies, *Intrusion Detection and Prevention systems* (IDP) and *Security Information and Event Management* (SIEM) systems which are implemented to mitigate the risk of lateral threat movement between IT and OT domains. With the rise of automation, the need for bidirectional data flows between OT and IT systems has increased. The convergence of IT and OT in this layer can offer organizations a competitive edge. However, it's important to note that adopting a flat network approach in this context can potentially heighten cyber risks for the organization.

2.2.6 Industrial Protocols

Industrial Protocols are the networks that are used to connect the different components of the ICS and allow them to communicate with each other. Industrial Protocols can include wired and wireless networks, such as Ethernet/IP, Modbus, DNP3, Profinet and others.
As mentioned at the beginning of this Chapter, industrial systems differ from classical IT systems in the purpose for which they are designed: controlling physical processes the former, processing and storing data the latter. For this reason, ICSs require different communication protocols than traditional IT systems for real time communications and data transfer.

A wide variety of industrial protocols exists: this is because originally each vendor developed and used its own proprietary protocol. However, these protocols were often incompatible with each other, resulting in devices from different vendors being unable to communicate with each other.
To solve this problem, standards were defined with a view to allowing these otherwise incompatible device to intercommunicates.
Among all the various protocols, some have risen to prominence as widely accepted standards. These *de facto* protocols are commonly utilized in industrial systems due to their proven reliability and effectiveness. In the following sections, we will provide a brief overview of some of the most prevalent and widely used protocols in the industry.

2.2.6.1 Modbus

Modbus is a serial communication protocol developed by Modicon (now Schneider Electric) in 1979 for use with its PLCs [15] and designed expressly for industrial use: it facilitates interoperability of different devices connected to the same network (sensors, PLCs, HMIs, ...) and it is also often used to connect RTUs to SCADA acquisition systems.

Modbus is the most widely used communication protocol among industrial systems because it has several advantages:

- simplicity of implementation and debugging
- it moves raw bits and words, letting the individual vendor to represent the data as it prefers
- it is, nowadays, an **open** and ***royalty-free*** protocol: there is no need to sustain licensing costs for implementation and use by industrial device vendors

Modbus is a **request/response** (or *master/slave*) protocol: this makes it independent of the transport layer used.

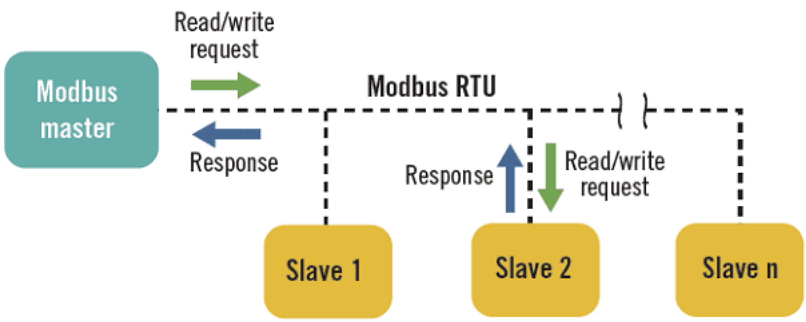


Figure 2.6: *Modbus Request/Response schema*

In this kind of architecture, a single device (master) can send requests to other devices (slaves), either individually or in broadcast: these slave devices (usually peripherals such as actuators) will respond to the master by providing data or performing the action requested by the master using the Modbus protocol. Slave devices cannot generate requests to the master [16].

There are several variants of Modbus, of which the most popular and widely used are Modbus RTU (used in serial port connections) and Modbus TCP (which instead uses TCP/IP as the transport layer). Modbus TCP embeds a standard Modbus frame in a TCP frame (see Figure 2.7): both masters and slaves listen and receive data via TCP port 502.

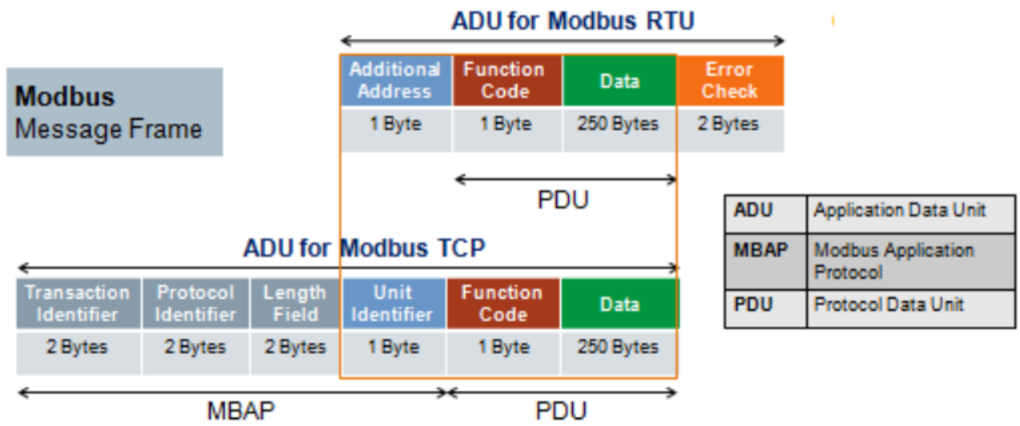


Figure 2.7: *Modbus RTU frame and Modbus TCP frame*

Modbus registers Modbus provides four object types, which map the data accessed by master and slave to the PLC memory:

- *Coil*: binary type, read/write accessible by both masters and slaves
- *Discrete Input*: binary type, accessible in read-only mode by masters and in read/write mode by slaves
- *Analog Input*: 16 bits in size (word), are accessible in read-only mode by masters and in read/write mode by slaves
- *Holding Register*: 16 bits in size (word), accessible in read/write mode by both masters and slaves. Holding Registers are the most commonly used registers for output and as general memory registers.

Modbus Function Codes *Modbus Function Codes* are specific codes used by the Modbus master within a request frame (see Figure 2.7) to tell the Modbus slave device which register type to access and which action to perform on it.

Two types of Function Codes exists: for data access and for diagnostic Function Codes list for data access are listed in Table 2.2:

Function Code	Description
FC01	Read Coils
FC02	Read Discrete Input
FC03	Read Holding Registers
FC04	Read Analog Input Registers
FC05	Write/Force Single Coil
FC06	Write/Force Single Holding Register
FC15	Write/Force Multiple Coils
FC16	Write/Force Multiple Holding Registers

Table 2.2: *Modbus Function Codes list*

Modbus Security Issues Despite its simplicity and widespread use, the Modbus protocol does not have any security feature, which exposes it to vulnerabilities and attacks.

Data in Modbus are transmitted unencrypted (*lack of confidentiality*), with no data integrity controls (*lack of integrity*) and authentication checks (*lack of authentication*), in addition to the *lack of session*. Hence, the protocol is vulnerable to a variety of attacks, such as Denial of Services (DoS), buffer overflows and reconnaissance activities.

The easiest attack to bring to the Modbus protocol, however, is **packet sniffing**: since, as mentioned earlier, network traffic is unencrypted and the data transmitted is in cleartext, it is sufficient to use a packet sniffer to capture the network traffic, read the packets and thus gather informations about the system such as ip addresses, function codes of requests and to modify the operation of the devices.

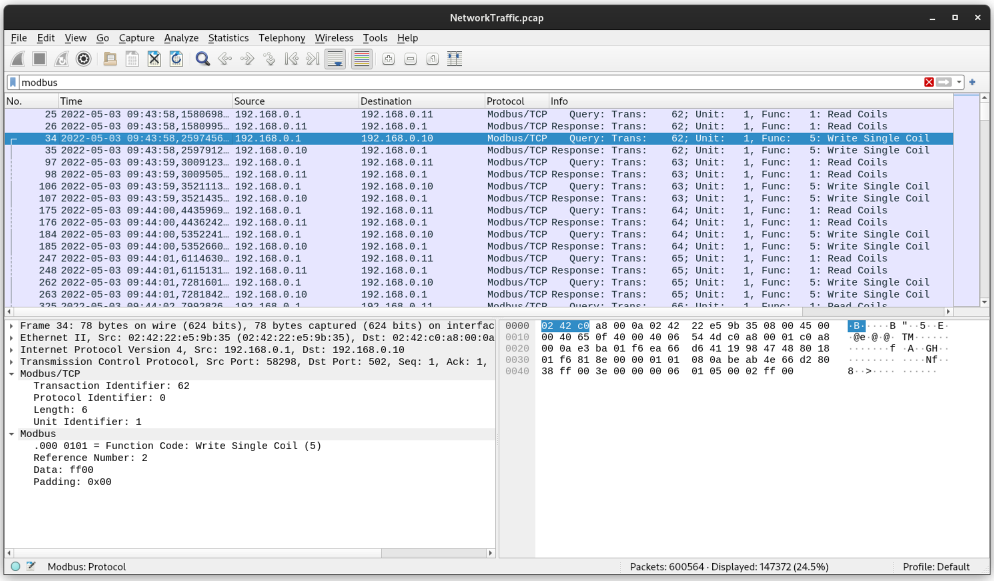


Figure 2.8: *Example of packet sniffing on the Modbus protocol*

To make the Modbus protocol more secure, an encapsulated version was developed within the *Transport Security Layer* (TLS) cryptographic protocol, also using mutual authentication. This version of the Modbus protocol is called **Secure Modbus** or **Modbus TLS**. In addition to this, Secure Modbus also includes X.509-type certificates to define permissions and authorisations [17].

2.2.6.2 EtherNet/IP

EtherNet/IP (where IP stands for *Industrial Protocol*) is an open industrial protocol that allows the *Common Industrial Protocol* (CIP) to run on a typical Ethernet network [18]. It is supported by ODVA [19].

EtherNet/IP uses the major Ethernet standards, such as IEEE 802.3 and the TCP/IP suite, and implements the CIP protocol stack at the upper layers of the OSI stack (see Figure 2.9). It is furthermore compatible with the main Internet standard protocols, such as SNMP, HTTP, FTP and DHCP,

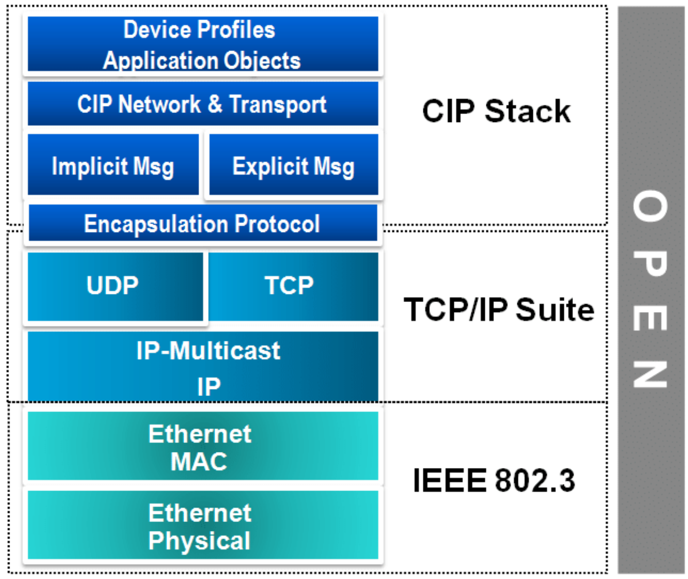


Figure 2.9: OSI model for EtherNet/IP stack

and other industrial protocols for data access and exchange such as *Open Platform Communication* (OPC).

Physical and Data Link layer The use of the IEEE 802.3 standard allows EtherNet/IP to flexibly adopt different network topologies (star, linear, ring, etc.) over different connections (copper, fibre optic, wireless, etc.), as well as the possibility to choose the speed of network devices.
IEEE 802.3 in addition defines at Data Link layer the *Carrier Sense Multiple Access - Collision Detection* (CSMA/CD) protocol, which controls access to the communication channel and prevents collisions.

Transport layer At the transport level, EtherNet/IP encapsulates messages from the CIP stack into an Ethernet message, so that messages can be transmitted from one node to another on the network using the TCP/IP protocol. EtherNet/IP uses two forms of messaging, as defined by CIP standard [18][20]:

- **unconnected messaging:** used during the connection establishment phase and for infrequent, low priority, explicit messages. Unconnected messaging uses TCP/IP to transmit messages across the network asking for connection resource each time from the *Unconnected Message Manager* (UCMM).

- **connected messaging:** used for frequent message transactions or for real-time I/O data transfers. Connection resources are reserved and configured using communications services available via the UCMM.

EtherNet/IP has two types of message connection [18]:

  - **explicit messaging:** *point-to-point* connections to facilitate *request-response* transactions between two nodes. These connections use TCP/IP service on port 44818 to transmit messages over Ethernet.
  - **implicit messaging:** this kind of connection moves application-specific **real-time I/O data** at regular intervals. It uses multicast *producer-consumer* model in contrast to the traditional *source-destination* model and UDP/IP service (which has lower protocol overhead and smaller packet size than TCP/IP) on port 2222 to transfer data over Ethernet.

Session, Presentation and Application layer At the upper layers, EtherNet/IP implements the CIP protocol stack. We will discuss this protocol more in detail in Section 2.2.6.3.

2.2.6.3 Common Industrial Protocol (CIP)

The *Common Industrial Protocol* (CIP) is an open industrial automation protocol supported by ODVA. It is a **media independent** (or *transport independent*) protocol using a *producer-consumer* communication model and providing a **unified architecture** throughout the manufacturing enterprise [21][22].
CIP has been adapted in different types of network:

- **EtherNet/IP**, adaptation to *Transmission Control Protocol* (TCP) technologies
- **ControlNet**, adaptation to *Concurrent Time Domain Multiple Access* (CTDMA) technologies
- **DeviceNet**, adaptation to *Controller Area Network* (CAN) technologies
- **CompoNet**, adaptation to *Time Division Multiple Access* (TDMA) technologies

CIP objects CIP is a *strictly object oriented* protocol at the upper layers: each object of CIP has **attributes** (data), **services** (commands), **connections**, and **behaviors** (relationship between values and services of attributes) which are defined in the **CIP object library**. The object library supports many common automation devices and functions, such as analog and digital I/O, valves, motion systems, sensors, and actuators. So if the same object is implemented in two or more devices, it will behave the same way in each device [23].

Security [24] In EtherNet/IP implementation, security issues are the same as in traditional Ethernet, such as network traffic sniffing and spoofing. The use of the UDP protocol also exposes CIP to transmission route manipulation attacks using the *Internet Group Management Protocol* (IGMP) and malicious traffic injection.

Regardless of the implementation used, it is recommended that certain basic measures be implemented on the CIP network to ensure a high level of security, such as *integrity*, *authentication* and *authorization*.

Chapter 3

State of the Art

IN COVENTIONAL IT SYSTEMS, the objective of an attacker is to comprehend the behavior of a program using diverse techniques in order to launch attacks that alter its execution flow, functionalities, or bypass limitations imposed by software licensing. These attack techniques involve an initial examination of the program, consisting of *static analysis* (i.e., analyzing the software without running it) and *dynamic analysis* (i.e., analyzing the program while it is running).
The outcome of these two investigative techniques is the ***reverse engineering*** of the software, which serves the purpose of identifying vulnerabilities or bugs and subsequently strategizing an attack.

In the context of OT systems, the notion of *reverse engineering* is not limited to its conventional definition, but also includes the concept of **process comprehension**. This term, introduced by Green et al. [25], refers to gaining a comprehensive understanding of the underlying physical process.

There is limited literature available concerning the gathering and analysis of information related to the comprehension and operation of an Industrial Control System (ICS). In Section 3.1, we will provide a brief overview of the existing literature on this topic, and in the subsequent sections, we will specifically focus on one of the presented papers.

3.1 Literature on Process Comprehension

Keliris and Maniatikos The first approach presented in this section is by Keliris and Maniatakos [26]: they present a methodology for automating the reverse engineering of ICS binaries based on a *modular framework* (called ICSREF) that can reverse binaries compiled with CODESYS [27], one of the most popular and widely used PLC compilers, irrespective of the language used.

Yuan et al. Yuan et al. [28] propose a *data-driven* approach to discovering cyber-physical systems process behavior from data directly: to achieve this goal, they have implemented a framework whose purpose is to identify physical systems and transition logic inference, and to seek to understand the mechanisms underlying these processes, making furthermore predictions concerning their state trajectories based on the discovered models.

Feng et al. Feng et al. [29] developed a framework that can generate system *invariant rules* based on machine learning and data mining techniques from ICS operational data log. These invariants are then selected by systems engineers to derive IDS systems from them.
The experiment results on two different testbeds, the *Water Distribution system* (WaDi) and the *Secure Water Treatment system* (SWaT), both located at the iTrust - Center for Research in Cyber Security at the University of Singapore of Technology and Design [30], show that under the same false positive rate invariant-based IDSs have a higher efficiency in detecting anomalies than IDS systems based on a residual error-based model.

Pal et al. Pal et al. [31] work is somewhat related to Feng et al.'s: this paper describes a data-driven approach to identifying invariants automatically using *association rules mining* [32] with the aim of generating invariants sometimes hidden from the design layout. The study has the same objective of Feng et al.'s and uses too the iTrust SwaT System as testbed.

Currently this technique is limited to only pair wise sensors and actuators: for more accurate invariants generation, the technique adopted must be capable of deriving valid constrains across multiple sensors and actuators.

Winnicki et al. Winnicki et al. [33] instead propose a different approach to process comprehension based on the *attacker's perspective* and not limited to mere *Denial of Service* (DoS): their approach is to discover the dynamic behavior of the system, in a semi-automated and process-aware way, through *probing*, that is, slightly perturbing the cyber physical system and observing how it reacts to changes and how it returns to its original state. The difficulty and challenge for the attacker is to perturb the system in such a way as to achieve an observable change, but at the same time avoid this change being seen as a system anomaly by the IDSs.

Green et al. Green et al. [25] also adopt an approach based on the attacker's perspective: this approach consists of two practical examples in a *Man in the Middle* (MitM) scenario to obtain, correlate, and understand all the types of information an attacker might need to plan an attack to alter the process while avoiding detection.

The paper shows *step-by-step* how to perform a ICS **reconnaissance**, a phase specifically designed to gather extensive intelligence on multiple fronts, including human factors, network and protocol information, details about the manufacturing process, industrial applications, and potential vulnerabilities. The primary goal is to accumulate a wealth of information to enhance understanding and awareness in these areas [34]).
Reconnaissance phase is fundamental to process comprehension and thus to the execution of MitM attacks.

Ceccato et al. Ceccato et al. [9] propose a methodology based on a *black box dynamic analysis* of an ICS using a reverse engineering tool to derive from the scans performed on the memory registers of the ex-

posed PLCs and the Modbus protocol network scans an approximate model of the physical process. This model is obtained by inferring statistical properties, business process and system invariants from data logs.
The proposed methodology was tested on a non-trivial case study, using a virtualized testbed inspired by an industrial water treatment plant.

In the next section we will examine this latest work in more detail, which will be the basis for my work and thus the subsequent chapters of this thesis.

3.2 Ceccato et al.'s black-box dynamic analyisis for water-tank systems

As previously mentioned, the paper introduces a methodology that relies on black box dynamic analysis of an Industrial Control System (ICS) and more particularly of its OT network. This methodology involves identifying potential Programmable Logic Controllers (PLCs) within the network and scanning the memory registers of these identified controllers. The purpose of this process is to obtain an approximate model of the controlled physical process.

The primary goal of this black box analysis is to establish a correlation between the different memory registers of the targeted PLCs and fundamental concepts of an OT network such as sensor values (i.e., measurements), actuator commands, setpoints (i.e., range of values of a physical variable), network communications, among others.
To accomplish this, the various types of memory registers are analyzed, and attempts are made to determine the nature of the data they might contain.
The second goal is to establish a relationship between the dynamic evolution of these fundamental concepts.

To accomplish this, Ceccato et al. have developed a prototype tool [35] that facilitates the reverse engineering of the physical system. This tool goes through four distinct phases:

1. **scanning of the system and data pre-processing**: this phase involves gathering data to generate data logs for the registers of PLCs and for Modbus network communications.

2. **graphs and statistical analysis**: The collected data is utilized to provide insights into the memory registers associated with the Modbus protocol by leveraging graphs and statistical data. This analysis approach offers valuable information about the characteristics and patterns of the memory registers.

3. **invariants inference and analysis**: generates system invariants, which are used to identify specific patterns and regularities within the system. Additionally, this phase provides users with the capability to view invariants related to a particular sensor or actuator.

4. **business process mining and analysis**: Using event logs, this phase involves reconstructing the business process that depicts how a process is executed. This step enables a thorough understanding of the sequence of events that occur in the system and how they are interrelated, ultimately leading to a comprehensive overview of the business process.

Figure 3.1 presents a schematic representation of the stages and the workflow associated with this work, specifying tools and technologies used. In the subsequent sections of this chapter, we will provide a detailed exploration of each of these phases, offering a comprehensive understanding of the entire process.

3.2.1 Testbed

Before delving into the description of the methodology's different phases, let's first examine the testbed utilized to evaluate this approach. The testbed

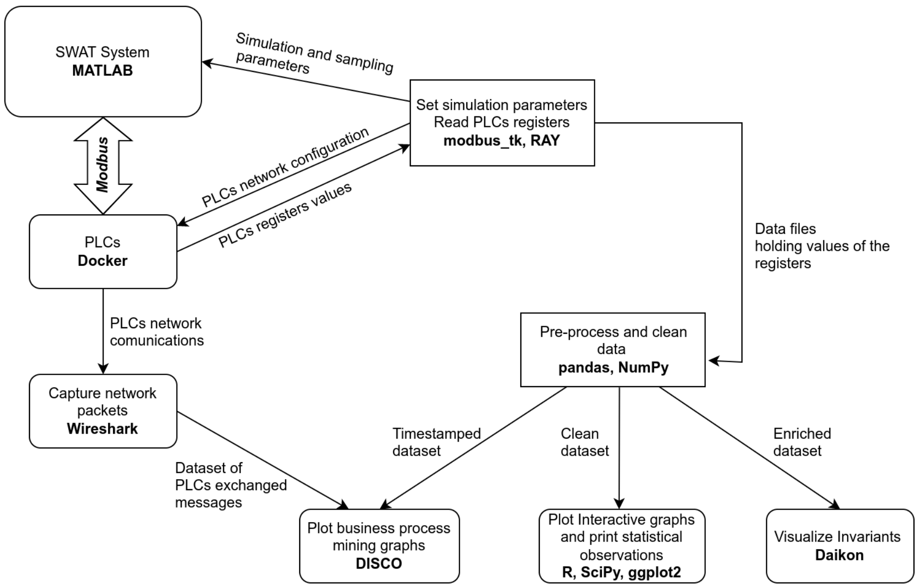


Figure 3.1: *Workflow of Ceccato et al.'s stages and operations with used tools*

employed for testing purposes is a (very) simplified rendition of the iTrust SWaT system [36], as implemented by Lanotte et al. [37]. Figure 3.2 provides a graphical representation of the testbed. This simplified version comprises three stages, each governed by a dedicated PLC.

Stage 1 During the initial stage, a **tank** referred to as T-201 with a capacity of 80 gallons is filled with raw water using the P-101 pump. Connected to the T-201 tank, the MV-301 motorized valve flushes out the accumulated water from the tank, directing it to the next stage. Initially, the water flows from the T-201 tank to the *filtration unit* (which is not specifically identified by any sensor), and subsequently to a **second tank** denoted as T-202, with a capacity of 20 gallons.

Stage 2 At the second stage, the water stored in tank T-202 flows into the *reverse osmosis unit* (RO), which serves as both a valve and a continuous water extractor. The purpose of the RO unit is to reduce organic impurities present in the water. Subsequently, the water flows from the *RO unit* to the third and final stage of the system.

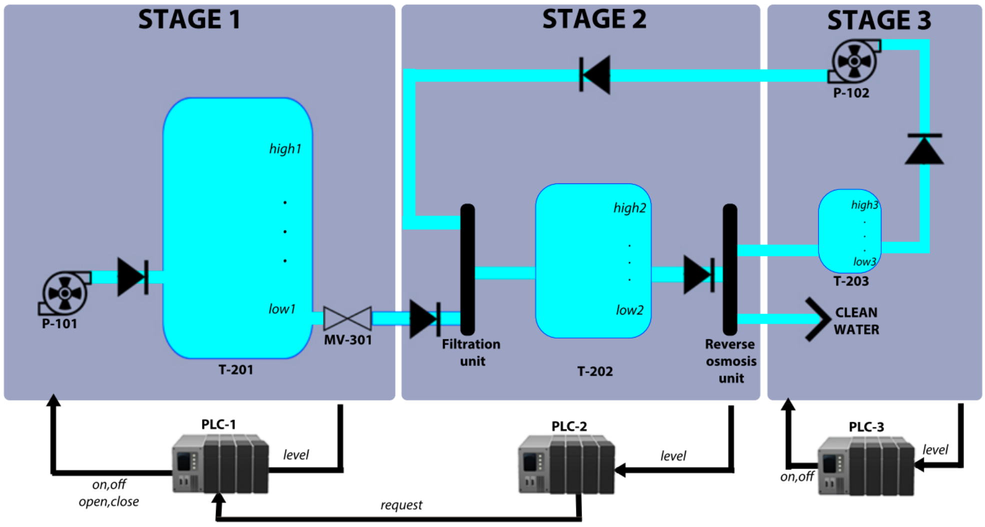


Figure 3.2: *The simplified SWaT system used for running Ceccato et al. methodology*

Stage 3 At the third stage, the water coming from the *RO unit* undergoes division based on whether it meets the required standards. If the water is deemed clean and meets the standards, it is directed into the distribution system. However, if the water fails to meet the standards, it is redirected to a *backwash tank* identified as T-203, which has a capacity of one gallon. The water stored in this tank is then pumped back to the stage 2 *filtration unit* using pump P-102.

As previously mentioned, each stage of the system is handled via a dedicated PLC, namely PLC1, PLC2, and PLC3, which are responsible for controlling their respective stages. Let's briefly explore the behavior of each PLC:

PLC1 PLC1 monitors the level of tank T-201 and distinguishes three different cases based on the level readings:

1. when the level of tank T-201 reaches the defined *low setpoint low1* (which is hardcoded in a specific memory register), PLC1 **opens pump P-101** and **closes valve MV-301**. This configuration allows the tank to be filled with water;

2. if the level of T-201 reaches the *high setpoint high1* (which is also hardcoded in a specific memory register), then the pump **P-101 is closed**;

3. in cases where the level of T-201 is between the *low setpoint low1* and the *high setpoint high1*, PLC1 waits for a request from PLC2 to open or close the valve MV-301. If a request to open the valve MV-301 is received, water will flow from T-201 to T-202. However, if no request is received, the valve remains closed. In both situations, the pump P-101 remains closed.

PLC2 PLC2 monitors the level of tank T-202 and adjusts its behavior based on the water level. There are three cases to consider:

1. when the water level in tank T-202 reaches *low setpoint low2* (also hardcoded in the memory registers), PLC2 sends a request to PLC1 through a Modbus channel to **open valve MV-301**. This request is made in order to allow the water to flow from tank T-201 to tank T-202. The transmission channel between the PLCs is established by copying a boolean value from a memory register of PLC2 to a corresponding register of PLC1.

2. when the water level in tank T-202 reaches the *high setpoint high2* value (also hardcoded in the memory registers), PLC2 sends a **close request to PLC1 for valve MV-301**. This request prompts PLC1 to close the valve, stopping the flow of water from tank T-201 to tank T-202.

3. In cases where the water level in tank T-202 is between the low and high setpoints, the valve MV-301 remains in its current state (open or closed) while the tank is either filling or emptying.

PLC3 PLC3 monitors the level of the T-203 backwash tank and adjusts its behavior accordingly. There are two cases to consider:

1. If the water level in the backwash tank reaches the *low setpoint*

low3, **PLC3 sets pump P103 to off**. This allows the backwash tank to be filled.

2. If the water level in the backwash tank reaches the *high setpoint high3*, **PLC3 opens pump P103**. This action triggers the pumping of the entire content of the backwash tank back to the filter unit of T-202.

3.2.2 Scanning of the System and Data Pre-processing

Scanning tool The Ceccato et al. scanning tool extends and generalizes a project I did [38] for the "*Network Security*" and "*Cyber Security for IoT*" courses taught by Professors Massimo Merro and Mariano Ceccato, respectively, in the 2020/21 academic year. The original project involved, in its first part, the recognition within a network of potential PLCs listening on the standard Modbus TCP port 502 using the Nmap module for Python, obtaining the corresponding IP addresses: then a (sequential) scan of a given range of the memory registers of the found PLCs was performed to collect the register data. The data thus collected were saved to a file in *JavaScript Object Notation* (JSON) format for later use in the second part of my project.

The scanning tool by Ceccato et. al works in a similar way, but extends what originally did by trying to discover other ports on which the Modbus protocol might be listening (since in many realities Modbus runs on different ports than the standard one, according to the concept of *security by obscurity*) and, most importantly, by **parallelizing and distributing the scan** of PLC memory registers through the Ray module [39], specifying moreover the desired granularity of the capture. An example of raw data capture can be seen at Listing 3.1:

```
"127.0.0.1/8502/2022-05-03 12_10_00.591": {
  "DiscreteInputRegisters": {"%IX0.0": "0"},
  "InputRegisters": {"%IW0": "53"},
  "HoldingOutputRegisters": {"%QW0": "0"},
  "MemoryRegisters": {"%MW0": "40","%MW1": "80"},
```

```
    "Coils": {"%QX0.0": "0"}}
```

Listing 3.1: *Example of registers capture*

The captured data includes PLC's IP address, Modbus port and timestamp (first line), type and name of registers with their values read from the scan (subsequent lines).

The tool furthermore offers the possibility, in parallel to the memory registers scan, of **sniffing network traffic** related to the Modbus protocol using the *Man in the Middle* (MitM) technique on the supervisory control network using a Python wrapper for tshark/Wireshark [40] [41]. An example of raw data obtained with this sniffing can be seen in Listing 3.2:

```
Time,Source,Destination,Protocol,Length,Function Code,
  ↪ Destination Port,Source Port,Data,Frame length on the
  ↪  wire,Bit Value,Request Frame,Reference Number,Info
2022-05-03 11:43:58.158,IP_PLC1,IP_PLC2,Modbus/TCP,76,Read
  ↪  Coils,46106,502,,76,TRUE,25,,"Response: Trans: 62;
  ↪ Unit: 1, Func: 1: Read Coils"
```

Listing 3.2: *Example of raw network capture*

Data Pre-processing The data collected by scanning the memory registers of the PLCs are then reprocessed by a Python script and converted in order to create a distinct raw dataset in *Comma Separated Value* format (CSV) for each PLC, containing the memory register values associated with the corresponding controller registers. These datasets are reprocessed again through the Python modules for **pandas** [42] and **NumPy** [43] by another script to first perform a **data cleanup**, removing all unused registers, **merged** into a single dataset, and finally **enriched** with additional data, such as the **previous value** of all registers and the the **measurement slope**, that is, the trend of the water level in the system tanks along the system cycles.[1]. See 3.2.7 for more detail.

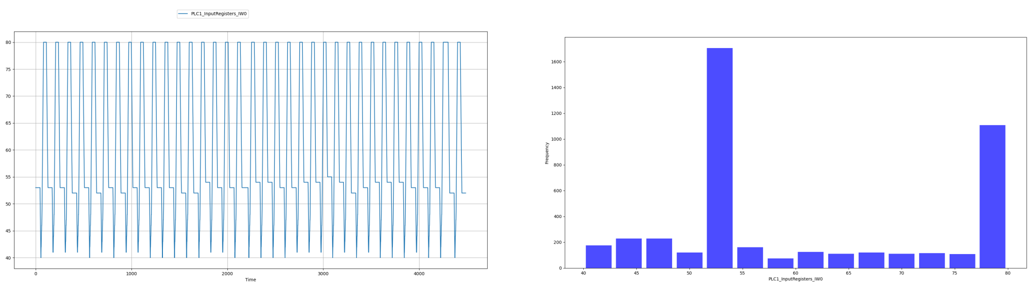
[1]Not all additional data are calculated and entered automatically by the tool: some are manually inserted.

This process leads to the creation of two copies of the full dataset: one enriched with the additional data, but not timestamped, which will be used for the invariant analysis; the other unenriched, but timestamped, which will be used for business process mining.

3.2.3 Graphs and Statistical Analysis

The paper mentions the presence of a *mild graph analysis*, performed using the framework **R** [44] for statistical analysis at the time of data gathering to find any uncovered patterns, trends and identify measurements and/or actuator commands through the analysis of registers holding mutable values.

There is actually no trace of this within the tool: *graph analysis* and *statistical analysis* of the data contained in the PLC memory registers are instead performed using the **matplotlib libraries** and statistical algorithms made available by the **SciPy libraries** [45], through two separate Python scripts (see Figure 3.3).



***(a)** Chart plot for* `PLC1_InputRegisters_IW01` *register*

***(b)** Histogram plot for* `PLC1_InputRegisters_IW01` *register*

***Figure 3.3:** Output graphs from graph analysis*

The first script plots the charts, one at the time, of certain registers entered by the user from the command line, plots in which one can see the trend of the data and get a first basic idea of what that particular register contains (a measurement, an actuation, a hardcoded setpoint, ...) and possibly the trend; the second script, instead, shows **a histogram and sta-**

tistical informations about the register entered as command-line input. These informations include:

- the mean, median, standard deviation, maximum value and minimum value
- two tests for the statistical distribution: *Chi-squared* test for uniformity and *Shapiro-Wilk* test for normality, as shown in Listing 3.3:

```
Chi-squared test for uniformity
Distance       pvalue    Uniform?
12488.340   0.00000000    NO

Shapiro-Wilk test for normality
Test statistic     pvalue    Normal?
0.844   0.00000000    NO

Stats of PLC1_InputRegisters_IW0
Sample mean = 60.8881; Stddev = 13.0164; max = 80; min =
 ↪ 40 for 4488 values
```

Listing 3.3: *Statistical data for* `PLC1_InputRegisters_IW0` *register*

3.2.4 Invariant Inference and Analysis

For invariant analysis Ceccato et al. rely on **Daikon** [46], a framework to **dynamically detect likely invariants** within a program. An *invariant* is a property that holds at one or more points in a program, properties that are not normally made explicit in the code, but within assert statements, documentation and formal specifications: invariants are useful in understanding the behavior of a program (in our case, of the cyber physical system).

Daikon uses *machine learning* techniques applied to arbitrary data with the possibility of setting custom conditions for analysis by using a specific file [47] with a *.spinfo* extension (see Listing 3.4). The framework is designed to find the invariants of a program, with various supported programming languages, starting from the direct execution of the program itself or passing as input the execution run (typically a file in CSV format): the authors of the paper tried to apply it by analogy also to the execution runs of a cyber physical system, to extract the invariants of this system.

```
PPT_NAME aprogram.point:::POINT
VAR1 > VAR2
VAR1 == VAR3 && VAR1 != VAR4
```

Listing 3.4: *Generic example of a .spinfo file for customizing rules in Daikon*

Therefore, Daikon is fed with the enriched dataset obtained in the pre-processing phase[2]: a simple bash script launches Daikon (optionally specifying the desired condition for analysis in the *.spinfo* file), which output is simply redirected to a text file containing the general invariants of the system (i.e., valid regardless of any custom condition specified), those generated based on the custom condition in the *.spinfo* file, and those generated based on the negation of the condition (see Listing 3.5 below).

```
=====================================================
aprogram.point:::POINT
PLC2_MemoryRegisters_MW1 == PLC3_MemoryRegisters_MW1
PLC1_MemoryRegisters_MW0 == 40.0
PLC1_MemoryRegisters_MW1 == 80.0
PLC1_Coils_QX00 one of { 0.0, 1.0 }
[...]
=====================================================
aprogram.point:::POINT;condition="PLC1_InputRegisters_IW0
 ↪ > 60"
PLC1_InputRegisters_IW0 > PLC1_MemoryRegisters_MW0
PLC1_InputRegisters_IW0 > PLC1_Min_safety
PLC1_MemoryRegisters_MW0 < prev_PLC1_InputRegisters_IW0
[...]
=====================================================
aprogram.point:::POINT;condition="not(
 ↪ PLC1_InputRegisters_IW0 > 60)"
PLC1_InputRegisters_IW0 < PLC1_MemoryRegisters_MW1
PLC1_InputRegisters_IW0 < PLC1_Max_safety
```

[2] In the paper, timestamped dataset is explicitly mentioned as input: from the tests performed, Daikon seems to ignore timestamps, hence it is indifferent whether the dataset is timestamped or not

```
PLC1_MemoryRegisters_MW1 > prev_PLC1_InputRegisters_IW0
[...]
```

Listing 3.5: *The three sections of Daikon analysis outcomes*

When the analysis is finished, the user is asked to enter the name of a registry to view its related invariants.

Some examples of invariants derived from the enriched dataset may be:

- measurements bounded by some setpoint;
- actuators state changes occourred in the proximity of setpoints or, vice versa, proximity of setpoints upon the occurrence of an actuator state change;
- state invariants of some actuators correspond to a specific trend in the evolution of the measurements (ascending, descending, or stable) or, vice versa, the measurements trend corresponds to a specific state invariant of some actuators.

3.2.5 Businness Process Mining and Analysis

Process mining is the analysis of operational processes based on the event log [48]: the aim of this analysis is to **extract useful informations** from the event data to **reconstruct and understand the behavior** of the business process and how it was actually performed.

In the considered system, process mining begins by analyzing the event logs derived from scanning the memory registers of the PLCs and monitoring the network communications associated with the Modbus protocol, as detailed in Subsection 3.2.2. These event logs serve as the *execution trace* of the system. A Java program is utilized to extract and consolidate information from these event logs, resulting in a CSV format file that captures the relevant data.
This file is fed to **Disco** [49], a commercial process mining tool, which

generates an *activity diagram* similar to UML Activity Diagram and whose nodes represent the activities while the edges represent the relations between these activities. In Figure 3.4 we can see an example of this diagram referred to PLC2 of the testbed: nodes represent the trend of register associated with measurement, actuator state changes, and communications between PLCs involving these state changes, while edges represent transitions with their associated time duration and frequency.

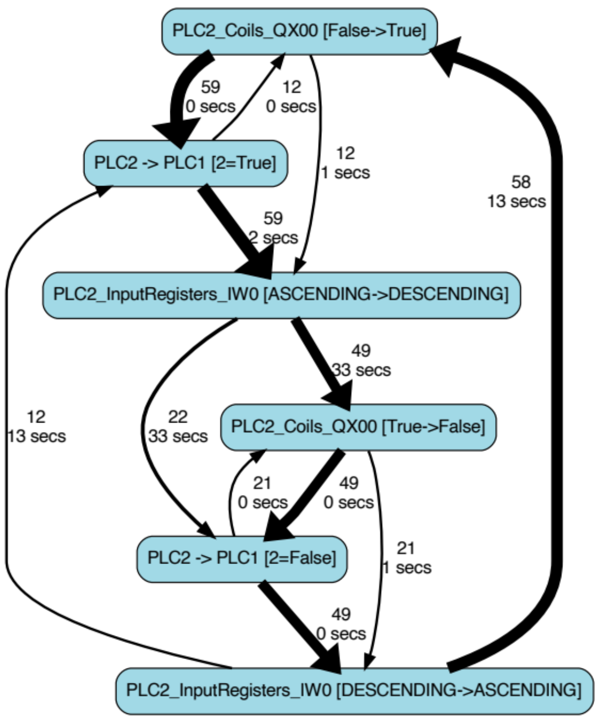


Figure 3.4: *An example of Disco generated activity diagram for PLC2*

The *business process* obtained in this way provides an **overview of the system** and makes it possible to **make conjectures** about its behavior, particularly between changes in actuator state and measurement trends (i.e., a given change in state of some actuators corresponds to a specific measurement trend and vice versa), and with the possibility of **establishing causality** between Modbus communications and state changes within the physical system.

3.2.6 Application

In this section we will see how the black box analysis presented above in its various phases is applied in practice, using the testbed described in Subsection 3.2.1. The methodology supports a ***top-down* approach**: that is, we start with an overview of the industrial process and then gradually refine our understanding of the process by descending to a higher and higher level of detail based on the results of the previous analyses and focusing on the most interesting parts of the system for further in-depth analysis.

Data Collection and Pre-processing According to what is described in the paper, the data gathering process lasted six hours, with a granularity of one data point per second (a full system cycle takes approximately 30 minutes). Each datapoint consists of 168 attributes (55 registers plus a special register concerning the tank slope of each PLC) after the enrichment. In addition, IP addresses are automatically replaced by an abstract name identified by the prefix PLC followed by a progressive integer (PLC1, PLC2, PLC3), in order to make reading easier.

Graphs and Statistical Analysis Graphs and Statistical Analysis revealed three properties regarding the contents of the registers:

Property 1: `PLC1_MemoryRegisters_MW0`, `PLC1_MemoryRegisters_MW1`, `PLC2_MemoryRegisters_MW0`, `PLC2_MemoryRegisters_MW1`, `PLC3_MemoryRegisters_MW0` and `PLC3_MemoryRegisters_MW1`

registers contain constant integer values (40, 80, 10, 20, 0, 10 respectively)[3]. The authors speculate that they may be (relative) hardcoded **setpoints**.

[3] From my tests on the original tool and dataset, the `PLC3_MemoryRegisters_MW0` register is deleted during the *pre-processing* phase, as it is recognized as an unused register because of the constant value "0" it takes on. This leads me to assume that the properties are derived from a human read of the dataset prior to the *pre-processing* phase.

Property 2: `PLC1_Coils_QX01`, `PLC1_Coils_QX02`, `PLC2_Coils_QX01`, `PLC2_Coils_QX02`, `PLC3_Coils_QX01` and `PLC3_Coils_QX03` contain mutable binary (Boolean) values. The authors speculate that these registers can be associated with the **actuators** of the system.

Property 3: `PLC1_InputRegisters_IW0`, `PLC2_InputRegisters_IW0` and `PLC3_InputRegisters_IW0` registers contain mutable values.

Property 3 suggests that those registers might contain **values related to measurements**: it is therefore necessary to investigate further to see if the conjecture (referred to as *Conjecture 1* in the paper) is correct.

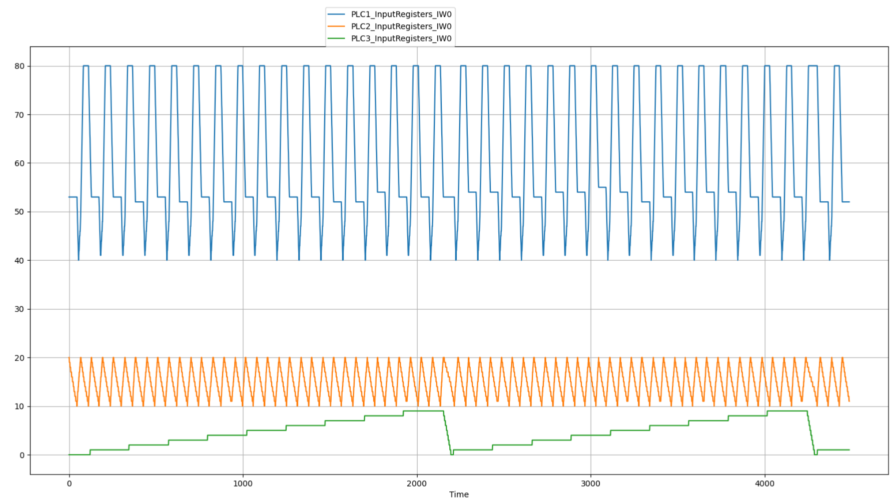


Figure 3.5: *Execution traces of InputRegisters_IW0 on the three PLCs*

The graph analysis of the `InputRegisters_IW0` registers of the three PLCs (summarized in Figure 3.5 with a single plot) not only seems to confirm the conjecture, but also allows the measurements to be correlated with the contents of the `MemoryRegisters_MW0` and `MemoryRegisters_MW1` registers to the measurements, which may well represent the **relative setpoints of the measurements**. Hence, we have *Conjecture 2* described in the paper referring to the relative setpoints:

Conjecture 2:

- the relative setpoints for `PLC1_InputRegisters_IW0` are 40 and 80;

- the relative setpoints for `PLC2_InputRegisters_IW0` are 10 and 20;
- the relative setpoints for `PLC3_InputRegisters_IW0` 0 and 9.

Further confirmation of this conjecture may come from statistical analysis. Indeed, in the example in Listing 3.1, some statistical data are given for the register `PLC1_InputRegisters_IW0`, including the maximum value and the minimum value: these values are, in fact, 80 and 40 respectively.

Business Process Mining and Analysis With Business Process Mining, the authors aim to **visualize and highlight relevant system behaviors** by relating PLC states and Modbus commands.

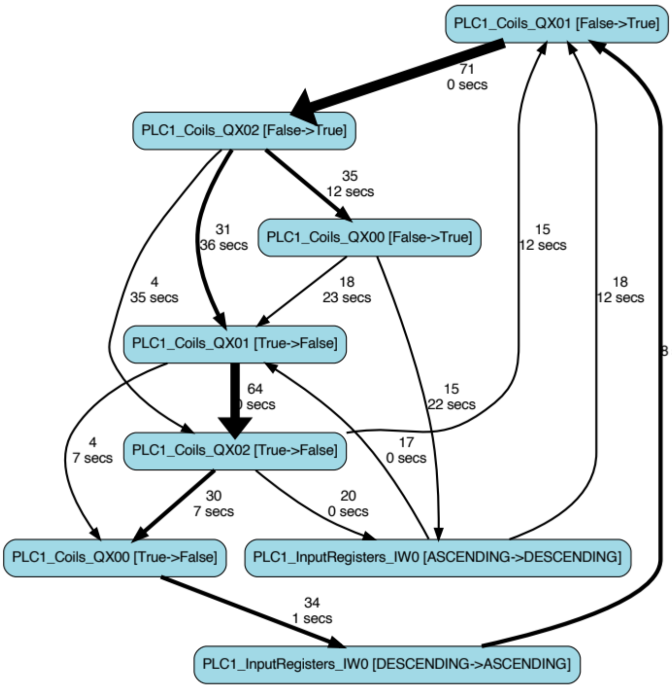
Through analysis of the activity diagrams shown in Figure 3.6, drawn through Disco, they derive the following properties and conjectures:

Property 4: PLC2 sends messages to PLC1 (see Figure 3.6b) which are recorded to `PLC1_Coils_QX02`.

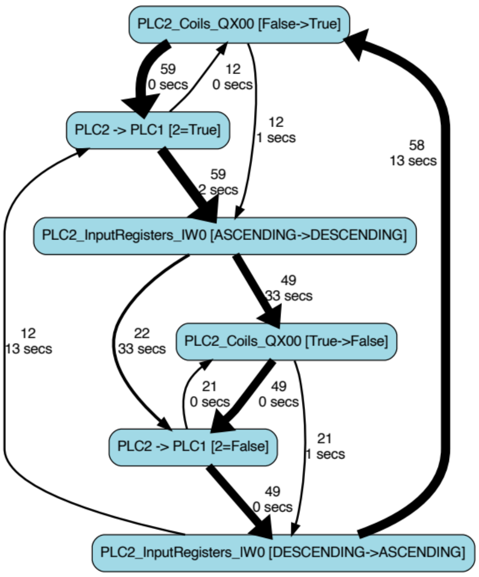
Conjecture 3: `PLC2_Coils_QX00` determines the trend in tank T-202 (Figure 3.6b).
When this register is set to *True*, the input register `PLC2_InputRegisters_IW0` related to the tank controlled by PLC2 starts an **ascending trend**; vice versa, when the coil register is set to *False*, the input register starts a **descending trend**.

Conjecture 4: If `PLC1_Coils_QX00` change his value to True, trend in tank T-201, related to `PLC1_InputRegisters_IW0` and controlled by PLC1, become **ascending** (see Figure 3.6a)

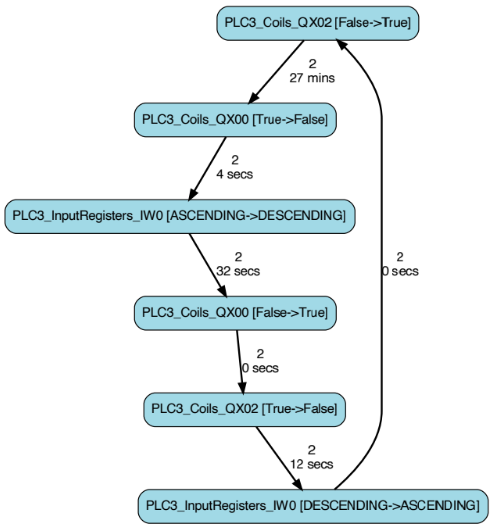
Conjecture 5: `PLC3_Coils_QX00` starts a **decreasing trend** in tank T-203, related to `PLC3_InputRegisters_IW0` and controlled by PLC3, whereas `PLC3_Coils_QX02` starts an **increasing trend** on the tank (see Figure 3.6c)



***(a)** States in PLC1*



***(b)** States and Modbus command in PLC2*



***(c)** States in PLC3*

***Figure 3.6:** Business process with states and Modbus commands for the three PLCs*

Invariant Inference and Analysis The last phase of the analysis of the example industrial system is invariant analysis, performed through Daikon framework. At this stage, an attempt will be made to confirm what has been seen previously and to derive new properties of the system based on the results of the Daikon analysis.

To get gradually more and more accurate results, the authors presumably performed more than one analysis with Daikon, including certain rules within the *splitter information file* (see Section 3.2.4 and Listing 3.4) based on specific conditions placed on the measurements, for example, the level of water contained in a tank. Given moreover the massive amount of invariants generated by Daikon's output, it is not easy to identify and correlate those that are actually useful for analysis: this must be done manually.

However, it was possible to have confirmation of the conjectures made in the previous stages of the analysis: starting with the setpoints, analyzing the output of the invariants returned by Daikon[4] reveals that

```
PLC1_InputRegisters_IW0 >= PLC1_MemoryRegisters_MW0 == 40.0
PLC1_InputRegisters_IW0 <= PLC1_MemoryRegisters_MW1 == 80.0
PLC2_InputRegisters_IW0 >= PLC2_MemoryRegisters_MW0 == 10.0
PLC2_InputRegisters_IW0 <= PLC2_MemoryRegisters_MW1 == 20.0
PLC3_InputRegisters_IW0 >= PLC3_MemoryRegisters_MW0 == 0.0
PLC3_InputRegisters_IW0 <= PLC3_MemoryRegisters_MW1 == 9.0
```

i.e., that the `MemoryRegisters_MW0` and `MemoryRegisters_MW1` registers of each PLC contain the **absolute minimum and maximum setpoints**, respectively (*Property 5*).

There is also a confirmation regarding *Property 4*: from the computed invariants it can be seen that

[4] The invariants shown here are a manual summary and derivation of those actually returned in output by Daikon. We will discuss this more in Section 3.2.7

`PLC1_Coils_QX01 == PLC1_Coils_QX02 == PLC2_Coils_QX00`

and from this derive that there is a **communication channel between PLC2 and PLC1**, where the value of `PLC2_Coils_QX00` is copied to `PLC1_Coils_QX01` and `PLC1_Coils_QX02` (*Property 6*).

Regarding the **relationships between actuator state changes and measurement trends**, invariant analysis yields the results summarized in the following rules:

Property 7: Tank T-202 level *increases* iff `PLC1_Coils_QX01 == True`. Otherwise, if `PLC1_Coils_QX01 == False` will be *non-increasing*.

This is because if the coil is *True* the condition `PLC2_InputRegisters_IW0 == PLC2_MemoryRegisters_MW0 == 20.0 && PLC2_slope > 0` is verified. On the opposite hand, if the coil is *False*, the condition `PLC2_InputRegisters_IW0 == PLC2_MemoryRegisters_MW0 == 20.0 && PLC2_slope <= 0` is verified. The *slope* is increasing if > 0, decreasing if < 0, stable otherwise.

Property 8: Tank T-201 level *increases* iff `PLC1_Coils_QX00 == True`. On the other hand, if `PLC1_Coils_QX00 == False` and if `PLC1_Coils_QX01 == True` the level will be *non-decreasing*.

Property 9: Tank T-203 level *decreases* iff `PLC3_Coils_QX00 == True`. It will be *non-decreasing* if `PLC1_Coils_QX00 == False`.

The last two properties concern the **relationship between actuator state changes and the setpoints**: it is intended to check what happens to the actuators when the water level reaches one of these setpoints. From the analysis of the relevant invariants, the following properties are derived:

Property 10: Tank T-201 reaches the upper absolute setpoint when `PLC1_Coils_QX00` changes its state from *True* to *False*. If the coil changes from *False* to *True*, the tank reaches its absolute lower setpoint.

Property 11: Tank T-203 reaches the upper absolute setpoint when `PLC3_Coils_QX00` changes its state from *True* to *False*. If the coil changes from *False* to *True*, the tank reaches its absolute lower setpoint.

3.2.7 Limitations

The methodology proposed by Ceccato et al. is certainly valid and offers a good starting point for approaching the reverse engineering of an industrial control system from the attacker's perspective, while also providing a tool to perform this task.

The limitations of this approach, however, all lie in the tool mentioned above and also in the testbed described in Section 3.2.1. In this section we will explain which are the criticisms of each phase, while in Chapter 4 we will formulate proposals to improve and make this methodology more efficient.

General Criticism There are several critical aspects associated with the application of this approach: the primary one concerns the fact that the proposed tool seems to be built specifically for the testbed used and that it is not applicable to other contexts, even to the same type of industrial control system (water treatment systems, in this case).

What severely limits the analysis performed with the tool implemented by Ceccato et al. is the use of *ad hoc* solutions and *a posteriori* interventions done manually on the datasets after the data gathering process: we will discuss this last aspect in more detail later.
Moreover, there is the presence of many *hardcoded* variables and conditions within the scripts: this makes the system unconfigurable and unable to properly perform the various stages of the analysis as errors can occur due to incorrect data and mismatches with the system under analysis.
Having considered, furthermore, only the Modbus protocol for network communications between the PLCs is another major limiting factor and

does not help the methodology to be adaptable to different systems communicating with different protocols (sometimes even multiple ones on the same system).

Let us now look at the limitations and critical aspects of each phase.

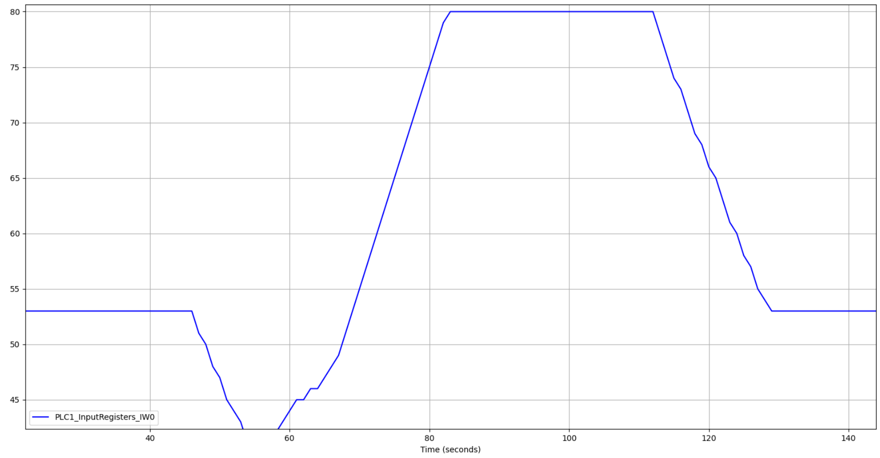
Testbed The testbed environment used by Ceccato et. al is entirely simulated, from the physical system to the control system. The PLCs were built with **OpenPLC** [50] in a Docker environment [51], while the physics part was built through **Simulink** [52].

OpenPLC is an open source cross-platform software that simulates the hardware and software functionality of a physical PLC and also offers a complete editor for PLC program development with support for all standard languages: *Ladder Logic* (LD), *Function Block Diagram* (FBD), *Instruction List* (IL), *Structured Text* (ST), and *Sequential Function Chart* (SFC). It is for sure an excellent choice for creating a zero-cost industrial or home automation and *Internet of Things* (IoT) system that is easy to manage via a dedicated, comprehensive and functional web interface. In spite of these undoubted merits, however, there are (at the moment) **very few supported protocols**: the main one and also referred to in the official documentation is **Modbus**, while the other protocol is DNP3.

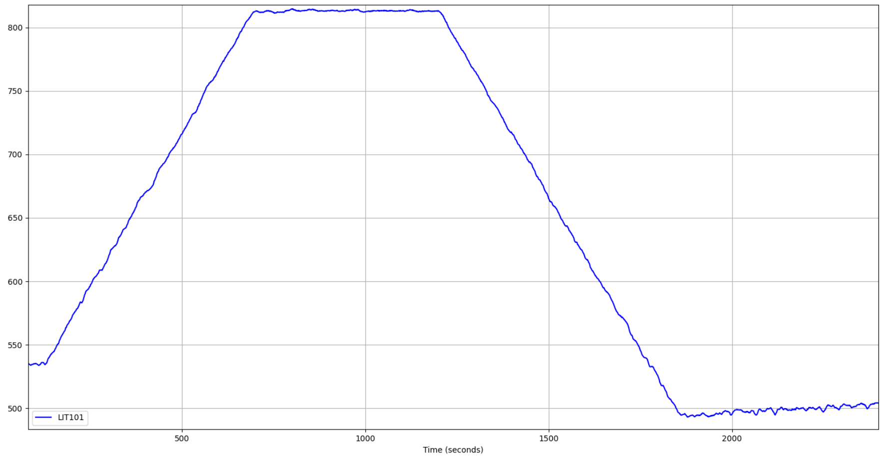
First limitation The biggest problem with the testbed, however, is not with the controller part, but with the **physical part**: first of all, it must be said that although this is something purely demonstrative even though it is fully functional, the implemented Simulink model is really **oversimplified** compared to the iTrust SWaT system, which itself is a scaled-down version of a real water treatment plant. In fact, in the entire system there are only three actuators, two of which are connected to the same tank and controlled by the same PLC, and sensors related only to the water level in the system's tanks: in a real system there are many more *field devices*, which can monitor and control other aspects of the system beyond the mere contents of the

tanks. Consider, for example, measuring and controlling the chemicals in the water, the pressure of the liquid in the filter unit, or more simply the amount of water flow at a given point or time.
All these must be considered and represent a number of additional variables that makes analysis and consequently reverse engineering of the system more difficult.

Second limitation The second critical aspect concerns the **simulation of the physics of the liquid** inside the tanks: Simulink does not consider the fact that inside a tank that is filling (emptying) the liquid in it undergoes **fluctuations** which cause the level sensor not to see the water level constantly increasing (decreasing) or at most being stable at each point of detection. Figure 3.7 exemplifies more clearly with an example the concept just expressed: these oscillations cause a **perturbation** in the data.
This issue leads to the difficulty, on a real physical system, of **correctly calculating the trend of a measurement** by using the slope attribute: if this was obtained with a too low granularity, the trend will be oscillating between increasing and decreasing even when in reality this would be in general increasing (decreasing) or stable; on the other hand, if the slope was obtained with a too high granularity there is a loss of information and the trend may be "flattened" with respect to reality.
In the present case, the slope in the Simulink model was calculated statically with a (very) low granularity, 5 and 6 seconds according to the Properties 7 and 9 described in the original paper: an averagely careful reader will have already guessed that this granularity is inapplicable to the real system in Figure 3.7b. As we will later see, we need to **operate on the data perturbations** to be able to obtain a suitable granularity and a correct calculation of the slope and consequently of the measurement trend.



(a)



(b)

Figure 3.7: *Water physics compared: simulated physics in the Simulink model (a) and physics in a real system (iTrust SWaT) (b). Fluctuations in the tank level in (b), almost completely absent in (a), can be appreciated.*

Pre-processing In the pre-processing phase, the authors make use of a Python script to merge all the datasets of the individual PLCs into a single dataset, remove the (supposedly) unused registers, and finally enrich the obtained dataset with additional attributes. These attributes, as seen in 3.2.2, are:

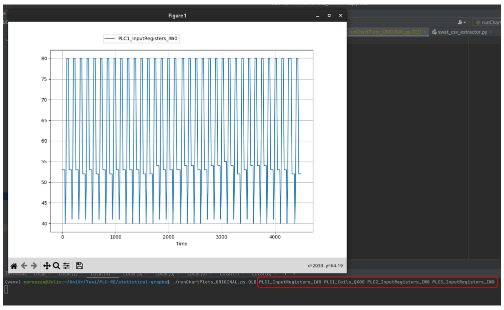
- the **previous value** of all registers;

- some **additional relative setpoints** named `PLCx_Max_safety` and `PLCx_Min_safety` (where x is the PLC number), which represent a kind of alert on reaching the maximum and minimum water levels of the tanks;
- the **measurement slope**.

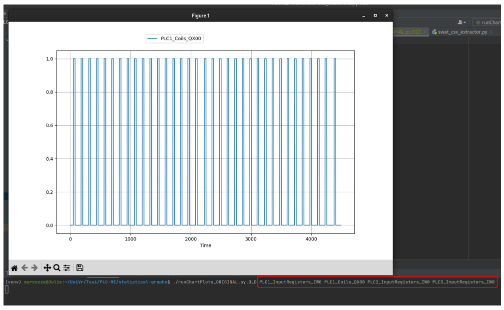
First limitation Merging the datasets of all individual PLCs into a single dataset representing the entire system can be a sound practice if the system to be anlized is (very) small as is the testbed analyzed here, consisting of a few PLCs and especially a few registers. If, however, the complexity of the system increases, this type of merging can become counterproductive and make it difficult to analyze and understand the data obtained in subsequent steps.
In short, there is no possibility to analyze only a subsystem and thus make the analysis faster and more understandable. Moreover, a data gathering can take up to days, and the analyst/attacker may need to make an analysis of the system isolating precise time ranges, ignoring everything that happens before and/or after: all of this, with the tool we have seen, cannot be done.

Second limitation Regarding the additional attributes, looking at the code of the script that performs the enrichment, we observed that **some attributes were manually inserted** after the merging phase: we are referring in particular to the attributes `PLCx_Max_safety` and `PLCx_Min_safety`, whose references were moreover hardcoded into the script, and the *slope* whose calculation method we mentioned in the previous paragraph about the testbed limitations.
In the end, only the attribute *prev* related to the value at the previous point of the detection is inserted automatically for all registers, moreover without the possibility to choose whether this attribute should be extended to all registers or only to a part.

Graphs and Statistical Analysis Describing the behavior of graphical analysis in Section 3.2.3 we had already mentioned that only one register plot at a time was shown and not, for example, a single window containing the charts of all registers entered by the user as input from the command line, such as in Figure 3.5. Figure 3.8 shows the actual behavior of graphical analysis: note that although we have specified four registers (highlighted in red in the figures) as command-line parameters, only one at a time is shown and it is necessary to close the current chart in order to display the next one.



(a) *Chart for* `PLC1_InputRegisters_IW0`



(b) *Chart for* `PLC1_Coils_QX00`

Figure 3.8: *Behavior of the Graph Analysis on the Ceccato et al.'s tool*

First limitation While displaying charts for individual registers still provides useful information about the system such as the distinction between actuators and measurements and the general trend of the latter, single display does not allow one to catch, or at least makes it difficult, the relationship that exists between actuators and measurements, where it exists, because a view of the system as a whole is missing.

In this way, the risk is to make conjectures about the behavior of the system that may prove to be at least imprecises, if not inaccurates.

Second limitation On the other hand, regarding the statistical analysis, two observations need to be made: the first is that for the given system, I personally was unable to appreciate the usefulness of the generated histogram in Figure 3.3b, as it does not provide any particular new information that has not already been obtained from the graphical analysis (except maybe something marginal); the second observation pertains to the presentation of statistical information obtained from the histogram plot. In certain cases, the histogram plot itself can overshadow the displayed statistical information. These statistics are actually shown on the terminal from which the script is executed. However, to an inattentive or unfamiliar user, these statistics may be mistaken for debugging output or warnings, as they coincide with the display of the histogram plot window, which takes the focus (see Figure 3.9).

In general, however, little statistical information is provided.

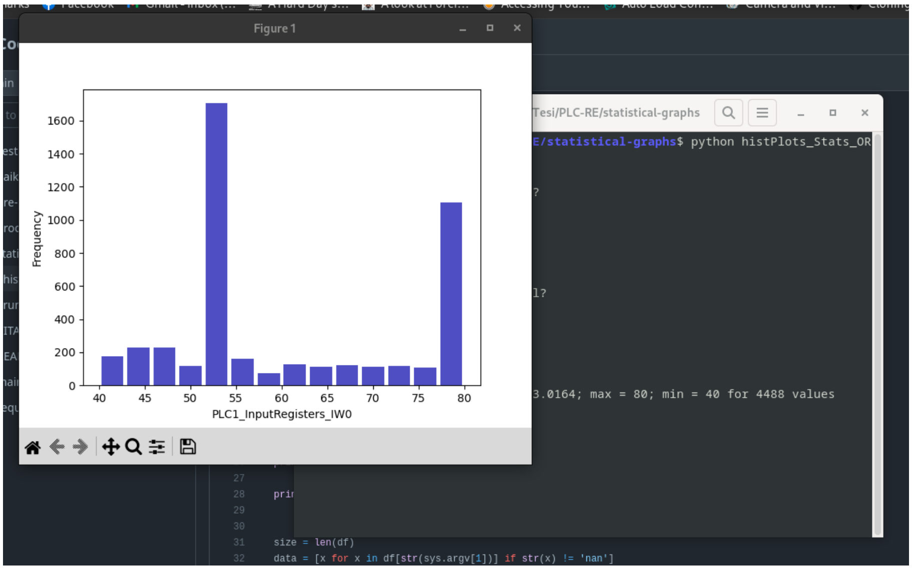


Figure 3.9: *Histogram plot overshadowing statistical information shown on the terminal window in the background*

In general, however, little statistical information is provided.

Business Process Mining and Analysis Concerning the data mining, this is a purely ad *hoc solution*, designed to work under special conditions: first, the timestamped dataset of the physical process and the one obtained after the packet sniffing operation of Modbus traffic on the network need to be synchronized and have the same granularity, in this case one event per second.

It is relatively easy, therefore, to find correspondences between Modbus commands sent over the network and events occurring on the physical system, such as state changes in actuations, due in part to the fact that the number of communications over the network is really small (see Section 3.2.1).

<u>First limitation</u> In a real system, network communications are much more numerous and involve many more devices even in the same second: finding the exact correspondence with what is happening in the cyber physical system becomes much more difficult.
Since this is, as mentioned, an *ad hoc* solution, only the Modbus protocol is being considered: as widely used as this industrial protocol is, other protocols that are widely used [53] such as EtherNet/IP (see Section 2.2.6.2) or Profinet should be considered in order to extend the analysis to other industrial systems that use a different communication network.

<u>Second limitation</u> The other limiting aspect of the business process mining phase is the **process mining software** used to generate the activity diagram. As mentioned in Section 3.2.5, the process mining software used by Ceccato et al. is **Disco**: this is commercial software, with an academic license lasting only 30 days (although free of charge), released for Windows and MacOS operating systems only, which makes its use under Linux systems impossible except by using emulation environments such as Wine.
For what is my personal vision and training as a computer scientist, it would have been preferable to use a *cross-platform*, *freely licensed open source* software alternative to Disco: one such software could have been **ProM Tools** [54], a framework for process mining very similar to Disco in functionality, but fitting the criteria just described, or use Python libraries such as **PM4PY** [55], which offer ready-to-use algorithms suitable for various process mining needs.

Invariants Inference and Analysis The limitation in this case is principally Daikon: this software is designed to compute the invariants of a software from its live execution or from a file containing its execution flow, not to find the invariants of a cyber physical system. Since there are currently no better consolidated alternatives for inferring invariants, however, an attempt was still made to use Daikon as best as possible.

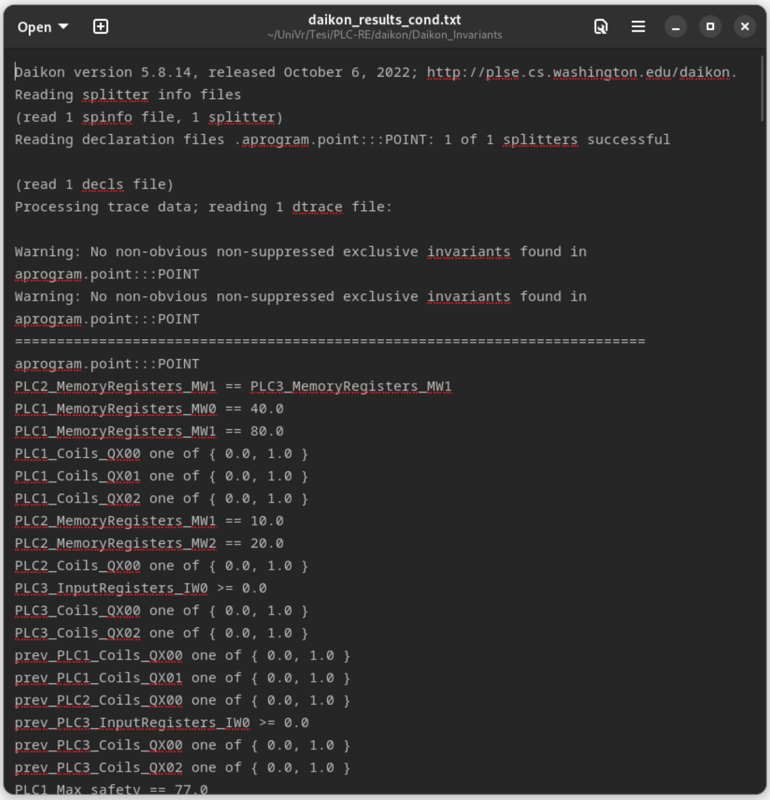


Figure 3.10: *Example of Daikon's output*

First limitation The biggest problem with Daikon applied to the computation of invariants of an industrial system is the difficult reading of the resulting output: the software in fact returns a very long list of invariants, one invariant per line, many of no use and without correlating invariants that may have common features or deriving additional information from them. The process of screening and recognizing the significant invariants, as well as the correlation between them, must be done by a human: certainly not an easy task given the volume of invariants one could theoretically be faced with (hundreds and hundreds of invariants). An example of Daikon's output can be seen in Figure 3.10.

Second limitation The bash script used in this phase of the analysis does not help at all in deriving significant invariants more easily: it merely launches Daikon and saves its output to a text file by simply redirecting the stdout to file. No data reprocessing is done during this step. In addition, if a condition is to be specified to Daikon before performing the analysis, it is necessary each time to edit the .spinfo file by manually entering the desired rule, an inconvenient operation when multiple analyses are to be performed with different conditions each time.

Table 3.1 provides a summary of the limitations discussed regarding the Ceccato et al. framework.

Phase	**Limitations**
Testbed	- Oversimplified model compared to iTrust SWaT system - Physics of the liquid not considered: this causes data perturbation
Pre-processing	- It is not possible to select a subsystem by (groups of) PLCs or by time range - Some additional attributes are manually inserted in the dataset
Graphical/Statistical Analysis	- Only one chart at the time is displayed: difficulty in capturing the relationship between actuators and sensors - Statistical Analysis provides little

	information
Business Process Analysis	- Ad hoc solution designed to work under special conditions - Use of commercial software for process mining
Invariant Analysis	- Reading output is challenging - Script for analysis merely launches Daikon without reprocessing outcomes

Table 3.1: *Summary table of Ceccato et al. framework limitations*

Chapter 4

A Substantial Improvement to Ceccato et al.'s Framework

In Chapter 3, we presented the state of the art of *process comprehension* of an Industrial Control System (ICS) with a focus on the methodology proposed by Ceccato et al. [9][Section 3.2], explaining what it consists of, its practical application on a testbed, and most importantly highlighting its limitations and critical issues (see Section 3.2.7).

In this chapter we will present **a proposal to improve the methodology** presented in the previous chapter, addressing most of the critical issues (or at least trying to do so) mentioned above by almost completely rewriting the original framework, enhancing its functionalities and inserting new ones where possible, while preserving its general structure and approach. The system analysis will in fact consist of the same four steps as in the original methodology (Data Pre-processing, Graph and Statistical Analysis, Business Process Mining and Invariants Inference), but each of them will be deeply revised in order to provide a richer, clearer and more complete process comprehension of the industrial system to be analyzed and its behavior.

As it may have already been noted, my proposals do not involve improving the data gathering phase: this is due simply to the fact that the novel framework will not be tested on the same case study used by Ceccato et, al. (Section 3.2.1), but on a different case study, the ITrust SWaT system [36], of which (some) datasets containing the execution trace of the physical system and the network traffic scan are already provided by iTrust itself. For more details about this case study, the reader is referred to Chapter 5.

4.1 The Proposed New Framework

In our version of the framework we decided to follow a few design choices:

1. it must be implemented in a **single programming language**;
2. it must be **independent of the system** to be analyzed;
3. It must provide greater **flexibility and ease of use** for the user at every stage.

In the following, we discuss these three features in more detail.

Single Programming Language The original tool was implemented using various programming languages in each of the different phases: from Python up to Java, passing through Bash scripting.
In our opinion, this heterogeneity makes it more difficult and less intuitive for the user to operate on the tool: moreover, the use of multiple technologies makes it more difficult to maintain the code and add new features, particularly if only a single person is managing the code (he/she might be proficient in one language, but little of the others).

For these reasons, we decided to use a single programming language, to ensure homogeneity to the framework and ease of use and maintenance of the code for anyone who wants to manage it in the future: we chose to use Python, because of its simplicity and easy readability combined with its versatility and powerfulness: moreover, Python can count on a massive number of available libraries and packages that meet all kinds of needs.

System Independence One of the biggest limitations of Ceccato et al.'s tool that we highlighted in Section 3.2.7 is the fact that it is **highly dependent on the testbed used**: that is, it is *not* possible to configure any of the tool's parameters to analyze different industrial systems. To overcome this issue and make my framework independent of the system to be analyzed, also eliminating all references to hardcoded variables and values present in the previous tool, we decided to use a **general configuration file**, named *config.ini*, in which the user can, at will, customize all the parameters necessary to perform the analysis of the targeted system.

Flexibility and Ease on Use The lack of flexibility and ease of use in a tool can be a significant disadvantage, limiting its effectiveness and making it challenging for the user to get the desired outcomes. The original tool suffered from these limitations, with users having to run scripts from the command line, with little to no options or parameters available to customize the analysis. As a result, the tool was not user-friendly and lacked the flexibility to adapt to specific user needs.

To settle these issues, I enhanced the command-line interface in the novel framework by adding new options and parameters. These new features provide the user with greater flexibility, enabling to specify parameters and options that allow for more in-depth analysis and focused results analyzing data more effectively and efficiently. With these enhancements, the framework has become more user-friendly, reducing the learning curve and making it more accessible to a wider range of users.
This, in turn, makes the framework more valuable and useful, increasing its adoption and effectiveness across a range of industrial control systems and applications.

Moreover, with new options and parameters users no longer have to rely solely on the command line interface, which can be challenging and intimidating for those with limited technical expertise. Instead, users can now access a range of customizable options and parameters, making the tool more intuitive and user-friendly.

Overall, the enhancements made to the framework represent a significant step forward in making it more effective, efficient, and user-friendly.

4.1.1 Framework Structure

The proposed framework follows a similar structure to the original tool, with a division into five main directories representing different phases of the analysis: **data pre-processing**, **graphs and statistical analysis**, **process mining**, and **invariant analysis**. A new phase is added compared to the original, concerning the **analysis of the network traffic**. These directories contain the corresponding Python scripts responsible for performing the analysis, along with any necessary subdirectories and input/output files to ensure the proper functioning of the framework.

```
.
├── config.ini
├── daikon
│    ├── Daikon_Invariants
│    ├── daikonAnalysis.py
│    └── runDaikon.py
├── network-analysis
│    ├── data
│    ├── networkAnalysis.py
│    ├── export_pcap_data.py
│    └── swat_csv_extractor.py
├── pre-processing
│    ├── mergeDatasets.py
│    └── system_info.py
├── process-mining
│    ├── data
│    └── process_mining.py
│
```

```
    └── statistical-graphs
        ├── histPlots_Stats.py
        └── runChartSubPlots.py
```

Listing 4.1: *Novel Framework structure and Python scripts*

Ahead of these directories there is the most important part, that allows the framework to be independent of the industrial control system being analyzed: the *config.ini* file. Here the user can configure general parameters and options, such as paths to read from or write files to, or related to individual analysis phases.
The file is divided into sections, each covering a different aspect of the configuration: each section contains user-customizable parameters that will then be called within the Python scripts that constitute the framework. Sections of *config.ini* are:

- **[PATHS]:** defines general paths such as the project root directory;
- **[PREPROC]:** includes parameters needed for the **pre-processing phase**, like the directory containing the raw datasets of the individual PLCs and *granularity* for the slope calculation. Granularity is the time interval over which the slope is calculated;
- **[DATASET]:** defines settings and parameters used during the **dataset enrichment** stage, for example the additional attributes;
- **[DAIKON]:** defines parameters needed for **invariant analysis** with Daikon, e.g. directories and files containing the outcomes of the analysis;
- **[MINING]:** contains parameters used during the **process mining** phase, such as data directory;
- **[NETWORK]:** includes specific settings for extracting the data obtained from the packet sniffing phase on the ICS network and converting it to CSV format. It also defines the **network protocols** that are to be analyzed.

4.1.2 Python Libraries and External Tools

As the framework has been developed entirely in Python, the objective was to minimize reliance on external tools and instead integrate various functionalities within the framework itself. The aim was to make the framework independent from external software. The only remaining external tool from the Ceccato et al. tool is Daikon. This choice was made because there is currently no better alternative or Python package available that offers the same functionalities as Daikon.

Instead, the framework extensively utilizes Python libraries for handling various functionalities and input data. The core libraries on which the framework relies are:

- **Pandas**, also used in the previous tool for dataset management, but whose use here has been deepened and extended
- **NumPy**, often used together with Pandas to perform some operations to support it;
- **MatPlotLib**, for managing and plotting graphical analysis;
- other scientific libraries such as **SciPy**, **StatsModel** [56] and **NetworkX** [57], for mathematical, statistical and analysis operations on the data;
- **GraphViz**, for the creation of activity diagrams in the process mining phase.

Having now seen the structure of the framework, in the next sections we will go into more detail describing our proposal.

4.2 Analysis Phases

4.2.1 Phase 1: Data Pre-processing

Data Pre-processing phase is probably the most delicate and significant one: depending on how large the industrial system to be analyzed is, the data collected, and how it is enriched using the additional attributes, the subsequent system analysis will provide more or less accurate outcomes.

The previous tool has several limitations, particularly at this stage. It does not allow for the isolation of a subsystem, either in terms of time or the number of PLCs to be analyzed. The system is considered as a whole without the ability to focus on specific subsystems. Additionally, many of the additional attributes had to be manually added, and for the ones entered automatically, there is no way to specify the register type to associate them with.

The combination of these limitations, along with the presence of hard-coded references to attributes and registers in the tool's code, makes the analysis of the system more challenging. Furthermore, it compromises the accuracy and reliability of the obtained results in terms of both quantity and quality.

In the proposed framework, these issues have been addressed by incorporating new features. Firstly, the framework allows for the selection of a subsystem from the command line based on both temporal criteria and the specific PLCs to be included. This enables more focused and targeted analysis. Additionally, we have revamped the process of enriching the resulting dataset by eliminating manual entry of additional attributes. Instead, users now have the flexibility to determine the type of additional attribute to associate with a specific register.

Furthermore, after the pre-processing stage, a preliminary analysis can be conducted on the resulting dataset. This analysis aims to identify the registers that are associated with actuators, measurements, and hardcoded setpoints or constants. It provides insights into the dataset and helps in refining the enrichment step. The parameters for this analysis can be configured in the *config.ini* file, allowing for customization and fine-tuning of the process.

In the upcoming sections, we will delve into a more comprehensive examination of the achievements made in this framework.

4.2.1.1 Subsystem Selection

In the previous tool, the datasets for each individual PLC in CSV format were required to be placed in a specific directory that was hardcoded in the script. The script would then merge and enrich these datasets to generate a single output dataset representing the complete process trace of the industrial system. However, the script did not provide options to select specific PLCs for analysis or define a temporal range for analysis. This lack of flexibility made the analysis more complex, especially when dealing with *transient states* (i.e., general statea in which the industrial system is still initializing before actually reaching full operation) or when focusing on specific parts of the industrial system during certain periods of interest. The fixed dataset structure also may increase the number of variables that could be analyzed.
Furthermore, the previous tool did not allow for specifying an output CSV file to save the resulting dataset. Each dataset creation and enrichment operation would overwrite the previous file, making it inconvenient for comparisons between different execution traces unless the files were manually renamed.

The proposed framework addressed these issues by introducing improvements. First of all, in the general *config.ini* file there are some general default settings about paths, and among them the one concerning the directory where to place the datasets of the individual PLCs to be processed. In addition to this option, there are other ones that define further aspects related to the operations performed in this phase. Listing 4.2 shows the settings in question:

```
[PATHS]
root_dir = /home/marcuzzo/UniVr/Tesi
project_dir = %(root_dir)s/PLC-RE
net_csv_path = %(root_dir)s/datasets_SWaT/2015/Network_CSV

[PREPROC]
raw_dataset_directory = datasets_SWaT/2015 # Directory
  ↪ containg datasets
```

```
dataset_file = PLC_SWaT_Dataset.csv # Default output dataset
granularity = 10  # slope granularity
number_of_rows = 20000  # Seconds to consider
skip_rows = 100000  # Skip seconds from beginning
```

Listing 4.2: *Paths and parameters for the Pre-processing phase in config.ini file*

At the same time, the user has the option to specify these settings via the command line using the new Python script called `mergeDatasets.py`, located in the `pre-processing` directory of the project. Any options provided through the command line will override the default settings specified in the *config.ini* file. These options are:

- **-s** or **--skiprows:** initial transient period (expressed in seconds) to be skipped. This option is useful in case the system has an initial transient or the analyzer wishes to start the analysis from a specific point in the dataset;
- **-n** or **--nrows:** time interval under analysis, expressed in terms of the number of rows in the dataset.
  This option makes a **selection** on the data of the dataset;
- **-p** or **--plcs:** PLCs to be merged and enriched. The user can specify the desired PLCs by indicating the CSV file names of the associated datasets with no limitations on number.
  This option makes a **projection** on the data of the dataset.
- **-d** or **--directory:** performs the merge and enrichment of all CSV files contained in the directory specified by user, overriding the default setting in *config.ini*. It is in fact the old functionality of the previous tool, maintained here to give the user more flexibility and convenience in case he wants to perform the analysis on the whole system. This is also the default behavior in case the `-p` option is not specified.
- **-o** or **--output:** specifies the name of the file in which the obtained dataset will be saved. It must necessarily be a file in CSV format.

- **-g** or **--granularity:** specifies a granularity (expressed in seconds) that will be used to calculate the measurement slope during the dataset enrichment phase. We will discuss this later in Section 4.2.1.2.

4.2.1.2 Dataset Enrichment

After a step in which a function is applied to each PLC-related dataset to eliminate its unused registers within the system [1], the **dataset enrichment operation** is performed.
This operation differs from the previous version not only in the fact that it is performed on each individual dataset and not on the resulting dataset, but also in the additional attributes: not only are they greater in number, but they are automatically calculated and inserted by the `mergeDatasets.py` script into the dataset and, most importantly, it is possible to decide through the parameters in the *config.ini* configuration file under the `[DATASET]` section to which registers these attributes should be assigned.
In Listing 4.3 we can see the list of additional attributes and how they should be associated with the registers of the dataset:

```
[DATASET]
timestamp_col = Timestamp
max_prefix = max_
min_prefix = min_
max_min_cols_list = lit|ait|dpit
prev_cols_prefix = prev_
prev_cols_list = mv[0-9]{3}|p[0-9]{3}
trend_cols_prefix = trend_
trend_cols_list = lit
trend_period = 150
slope_cols_prefix = slope_
slope_cols_list = lit
```

Listing 4.3: `config.ini` *parameters for dataset enriching*

Following is a brief explanation of the parameters just seen:

[1] This is especially true if the Modbus register scan has been performed, in which ranges of registers are scanned: it is assumed that unused registers have constant value zero

`timestap_col` indicates the name of the column that contains the data timestamps. This parameter is used not only in this phase, but is also referred to in the Process Mining phase. In the previous work, this parameter was hardcoded and not configurable (and thus causing errors if the system being analyzed changed)

`max_prefix`, `min_prefix`, `max_min_cols_list` refer to any relative maximum or minimum values (*relative setpoints*) of one or more measures and that can be found and inserted as new columns within the dataset. The first two parameters indicate the prefix to be used in the column names affected by this additional attribute, while the third specifies of which type of registers we want to know the maximum and/or minimum value reached (several options can be specified using the logical operator `|` - or).
If, for example, we want to know the maximum value of the registers associated with the tanks, indicated in the iTrust SWaT system by the prefix `LIT`, we only need to specify the necessary parameter in the *config.ini* file, so `max_min_cols_list = lit`.
The result will be to have in the dataset thus enriched a new column named `max_P1_LIT101`.

`prev_cols_prefix`, `prev_cols_list` refer to the values at the previous time instant of the registers specified in `prev_cols_list`. It is possible to specify registers using *regex*, as in the example shown. It may be useful in some cases to have this value available to check, for example, when a change of state of a single given actuator occurs. The behavior of these parameters is the same as described in the point above.

`slope_cols_prefix`, `slope_cols_list` are related to the calculation of the slope of a specific register that contains numeric values (usually a measure), that is, its trend. Slope calculation makes little sense on booleans. The slope can be **ascending** (if its value is greater than zero), **descending** (if less than zero) or **stable** (if approximately equal to zero). We will delve into the details of slope calculation in the following paragraph, as it pertains to the attributes `trend_cols_prefix`, `trend_cols_list`, and `trend_period`.

Initially, the parameters for registers to be associated with each additional attribute may be left blank, as we may not have prior knowledge about the system and are unsure about which registers correspond to actuators, measurements, or other attributes. This information can be obtained from the brief analysis that follows the merging of datasets. The analysis, performed based on user's choice, provides indications on potential sensors, actuators, and other relevant information. These indications help the user set the desired values in the *config.ini* file and refine the enrichment process by re-launching the `mergeDatasets.py` script.

Slope Calculation The *slope* is an attribute that represents the **trend** of the measurement being considered. It is particularly useful, in our context, during the inference and invariant analysis phase to gather information about the trend under specific conditions. The slope can generally be classified as **increasing** (slope > 0), **decreasing** (slope < 0), or **stable** (slope = 0).
Normally, the slope is calculated through a simple mathematical formula: given an interval a, b relative to the measurement l, the slope is given by the difference of these two values divided by the amount of time t that the measurement takes to reach b from a:

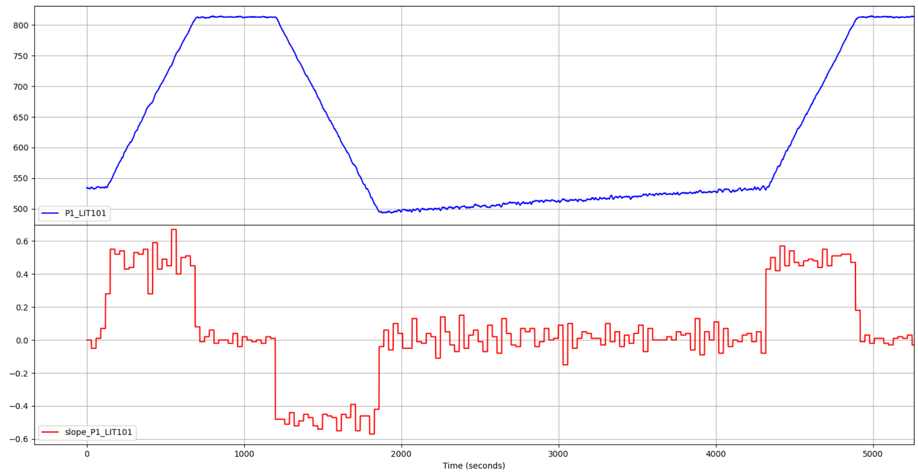
$$slope = \frac{l(b) - l(a)}{t(b) - t(a)}$$

In the proposed framework, similar to the previous tool, this time interval (the granularity) can be adjusted to be either long or short. The choice of granularity depends on the desired accuracy of the slope calculation. A lower granularity will provide a slope that closely reflects the actual measurement trend, while a higher granularity will result in flatter slope data. Each time interval within which the measurement is divided corresponds to a slope value. These slopes are calculated and added as additional attributes in the dataset. Later on, these slope values are used to determine the trend of the measurement in specific situations or conditions.

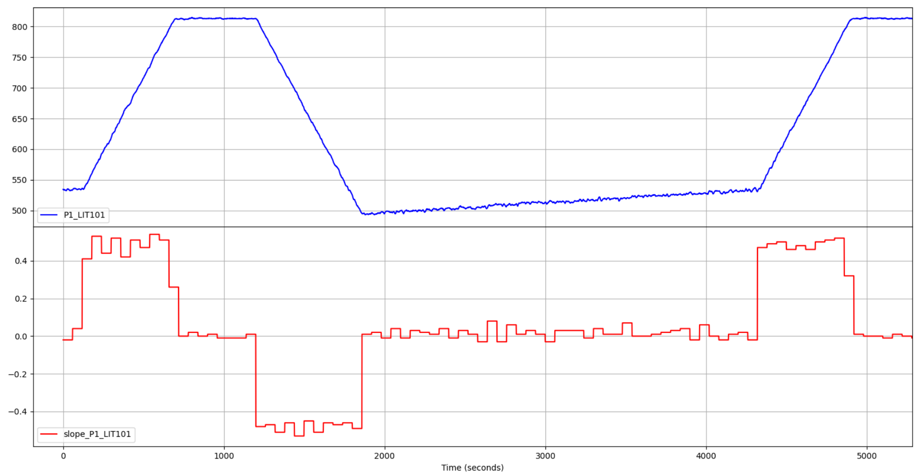
Calculating the slope directly from the raw measurement data can be a suitable approach for systems where the measurements are not heavily influenced by **perturbations**. Perturbations, such as liquid oscillations in a tank during filling and emptying phases, can lead to fluctuating readings of the level. In such cases, maintaining a low granularity can provide a more accurate calculation of the overall trend that closely aligns with the actual measurement trend. The tanks of Ceccato et al.'s testbed are an example where this holds true.
However, if perturbations significantly affect the measurement readings, calculating the slope on individual time intervals may result in an inaccurate trend definition, irrespective of the chosen granularity. In such cases, the fluctuating nature of the measurements due to perturbations can introduce errors in the slope calculation, making it less reliable as an indicator of the actual trend.

Figure 4.1 demonstrates this assertion: the measurement, in blue, refers to the `P1_LIT101` tank of the iTrust SWaT system; in red, the slope calculation related to the measurement with three different granularities: 30 (Figure 4.1a), 60 (Figure 4.1b) and 120 seconds (Figure 4.1c). It is noticeable that as the granularity increases, the slope values flatten. Moreover, in the time interval between seconds 1800 and 4200, the level of `P1_LIT101` exhibits a predominantly increasing trend, yet the calculated slope values fluctuate between positive and negative. Consequently, during the invariant analysis, the overall increasing trend may not be detected, resulting in a loss of information.

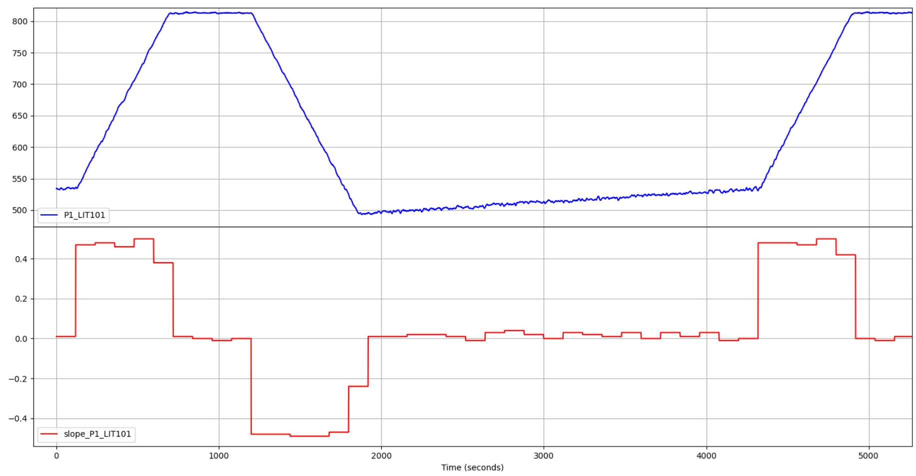
The previous tool did not take into account the possibility of having strongly perturbed data, which presented a challenge that we needed to address in the development of the proposed framework.



(a)



(b)



(c)

Figure 4.1: *Slope comparison with granularity 30 (a), 60 (b) and 120 seconds (c)*

The solution to this problem involves applying techniques to reduce the "noise" in the data, aiming to achieve a more linear trend in the measurement curve. By minimizing the effects of perturbations, we can calculate slopes more accurately.

There are various methods available for smoothing out noise in the data. In our framework, we focused on two commonly used approaches found in the literature: **polynomial regression** and **seasonal decomposition**. In addition to these two methods, we also explored the use of a **line simplification algorithm**.

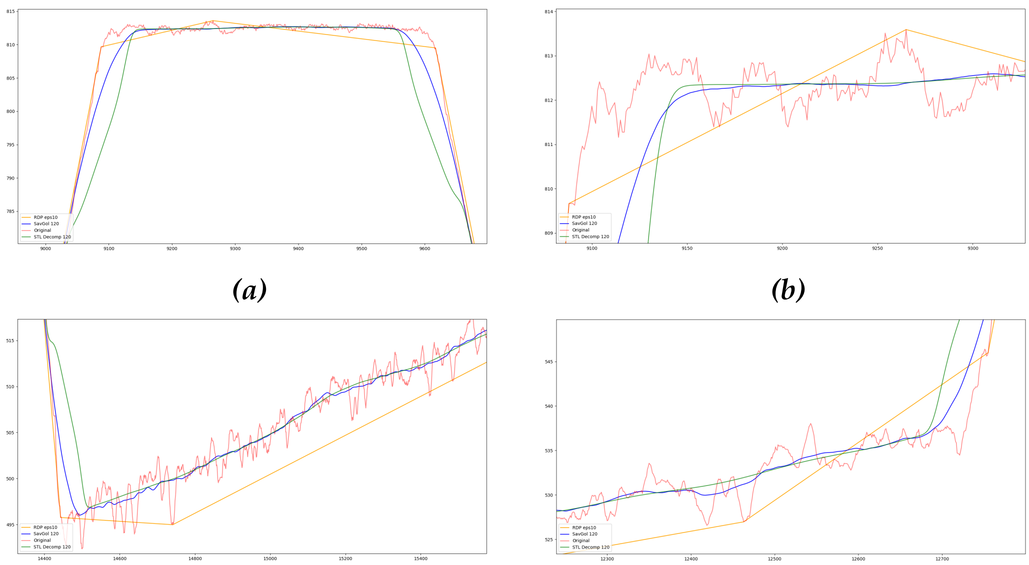
Polynomial regression [58] is a technique that allows us to create a filter to reduce the impact of noise on the data. By fitting a polynomial function to the measurements, we can obtain a smoother curve that captures the underlying trend while minimizing the effects of perturbations.

Seasonal decomposition [59], specifically the part related to trending, is another method we explored. It involves decomposing the time series into different components, such as trend, seasonality, and residual. By isolating the trend component, we can obtain a cleaner representation of the underlying pattern in the data.

Line simplification algorithms [60] aim to reduce the complexity of a polyline or curve by approximating it with a simplified version composed of fewer points. By selectively removing redundant or less significant points, line simplification algorithms help reduce storage space and computational requirements while preserving the overall shape and characteristics of the original line.

Regarding polynomial regression, we evaluated the use of the **Savitzky-Golay filter** [61] as a smoothing technique. For seasonal decomposition, we explored the **Seasonal-Trend decomposition using LOESS** (STL) method [62]. For the line simplification algorithm, we specifically considered the **Ramer-Douglas-Peucker** (RDP) algorithm [63].

Figure 4.2 shows a quick graphical comparison of these techniques compared with the original data. The solution adopted is the *STL decomposition* method, which effectively reduces noise compared to the Savitzky-Golay filter. However, it should be noted that this method may introduce some delay in certain parts of the data, as is typically observed in similar algorithms. Despite its apparent effectiveness, the RDP algorithm fails to accurately approximate sections where the measurement level remains relatively stable. Consequently, it yields incorrect slope estimations, causing a loss of valuable information about the system.



(a) *(b)* *(c)* *(d)*

Figure 4.2: *Savitzky-Golay filter (blue line), STL decomposition (green) and RDP algorithm (orange) comparison*

By applying the STL decomposition, we observe a notable enhancement in slope calculation even when using a low granularity. Figure 4.3 demonstrates that, with the same granularity as shown in Figure 4.1a, the slope values, albeit exhibiting fluctuations, consistently align with the underlying trend of the data curve. The introduced lag resulting from the decomposition's periodicity is responsible for the observed delay.

The periodicity, which defines the sampling time window for decomposition and the level of noise smoothing, can be configured using the `trend_period` directive in the *config.ini* file.

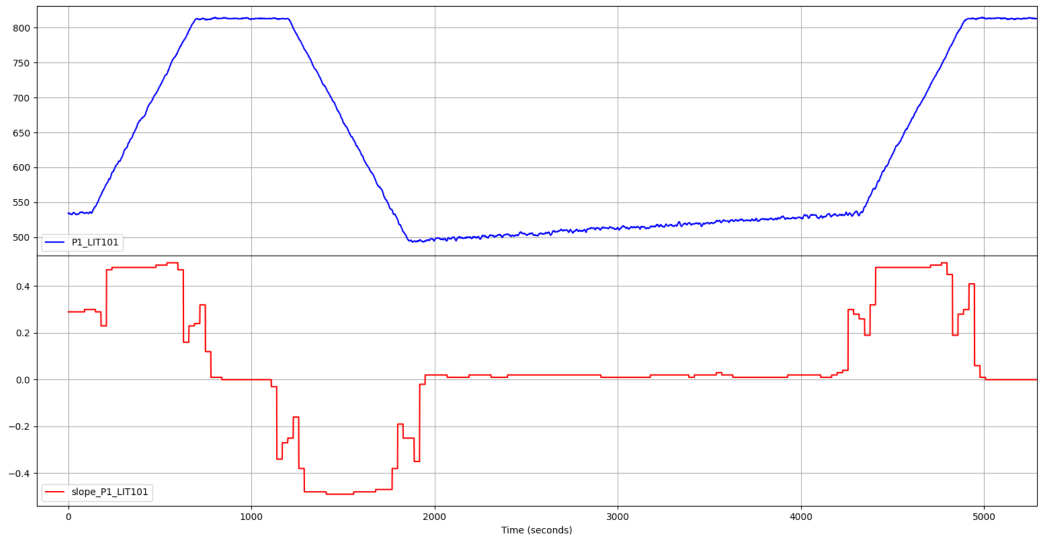


Figure 4.3: *Slope after the application of the STL decomposition*

During the slope calculation, the analysis will be performed on the data from the additional measurement trend attributes specified in the `trend_cols_list` directive of the configuration file, rather than on the original unfiltered data.

To ensure proper interpretation by Daikon, the decimal values representing the calculated slopes are converted into **three numerical values**: -1, 0, and 1. These values correspond to *decreasing* (if the slope is less than zero), *stable* (if it is equal to zero), and *increasing* (if it is greater than zero) trends, respectively. Figure 4.4 displays the modified slopes along with the curve obtained from the STL decomposition:

4.2.1.3 Datasets Merging

During this step, the datasets of the individual PLCs are merged, resulting in two separate datasets. The first dataset is enriched with additional attributes but excludes the timestamp column. This dataset is intended for inference and invariant analysis. The second dataset does not contain any additional data and is specifically used in the process mining phase.

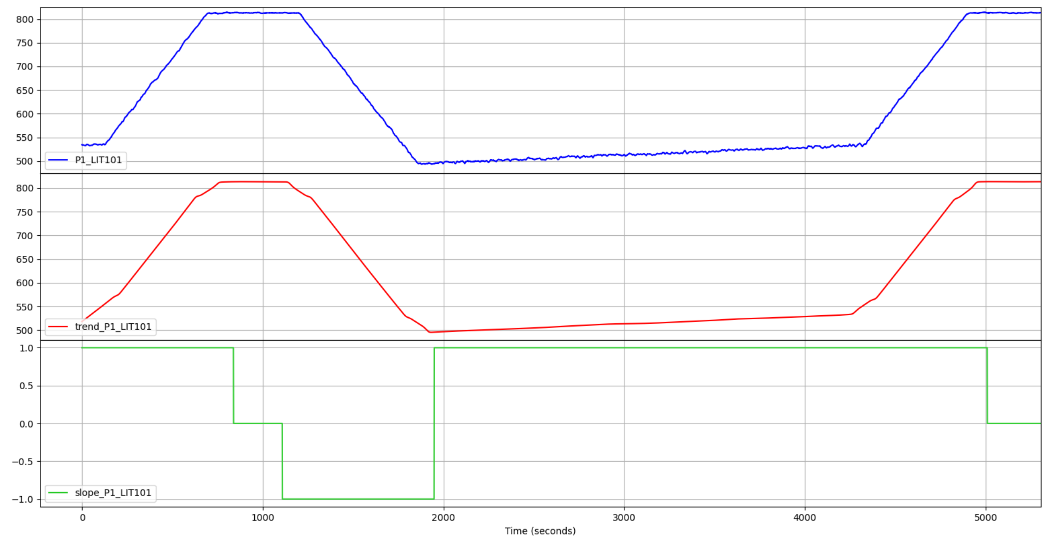


Figure 4.4: *The new slope representation (green line) and the smoothed measurement data obtaind with the STL decomposition (red)*

By default, the enriched dataset will be saved in CSV format in the `$(project-dir)/daikon/Daikon_Invariants` directory. The other dataset, without additional data, will be saved in the `$(project-dir)/process-mining/data` directory. It's worth noting that both paths can be configured in the *config.ini* file. The dataset name can be specified in the *config.ini* file or through the `-o` command-line option. When generating the dataset for process mining, the script will automatically add a `_TS` suffix to the filename to indicate that it includes the timestamp. This flexibility allows the user to provide a different filename for each output, preventing overwriting of previous datasets. It enables the user to save the execution trace of the selected subsystem separately and utilize them in subsequent analysis phases.

4.2.1.4 Brief Analysis of the Obtained Subsystem

After merging the datasets, the user has the option to perform an **optional analysis** of the resulting dataset to extract preliminary data. This analysis aims to gather basic information about the (sub)system and potentially refine the enrichment process. If the user chooses to proceed with

the analysis, the `mergeDatasets.py` script invokes another Python script located in the `$(project-dir)/pre-processing` directory called `system_info.py`. Relying on an analysis based on a combination of Daikon and Pandas this script performs a quick analysis of the dataset allowing to **estimate**, albeit approximately, the **type of registers** (sensors, actuators, ...), also identifying possible maximum and minimum values of measurements and hardcoded setpoints. Furthermore, leveraging the use of the additional attribute `prev_`, the `system_info.py` script is capable of deriving measurement values corresponding to state changes of individual actuators. This allows for the identification of specific measurements associated with the activation or deactivation of certain actuators within the system.
As the last information we have duration of actuator states for each cycle of the system: this information can be useful for making assumptions and conjectures about the behavior of an actuator in a specific state or, by observing the duration values of each cycle, highlighting anomalies in the system.
Listing 4.4 shows an example of this brief analysis related to PLC1 of the iTrust SWaT system (for brevity, only one measurement is reported in the analysis of actuator state changes):

```
Do you want to perform a brief analysis of the dataset? [y
  ↪ /n]: y

Actuators:
P1_MV101 [0.0, 1.0, 2.0]
P1_P101 [1.0, 2.0]

Sensors:
P1_FIT101 {'max_lvl': 2.7, 'min_lvl': 0.0}
P1_LIT101 {'max_lvl': 815.1, 'min_lvl': 489.6}

Hardcoded setpoints or spare actuators:
P1_P102 [1.0]

Actuator state changes:
       P1_LIT101  P1_MV101  prev_P1_MV101
```

```
669      800.7170         0              2
1850     499.0203         0              1
4876     800.5992         0              2
6052     498.9026         0              1
9071     800.7170         0              2
10260    499.1381         0              1
13268    801.3058         0              2
14435    498.4315         0              1
17423    801.4628         0              2
18603    498.1567         0              1

P1_LIT101  P1_MV101  prev_P1_MV101
677      805.0741         1              0
4885     805.7414         1              0
9079     805.7806         1              0
13276    805.1133         1              0
17432    804.4068         1              0

P1_LIT101  P1_MV101  prev_P1_MV101
1858     495.4483         2              0
6060     497.9998         2              0
10269    495.9586         2              0
14443    495.8016         2              0
18611    494.5847         2              0

       P1_LIT101  P1_P101  prev_P1_P101
118      536.0356        1             2
4322     533.3272        1             2
8537     542.1591        1             2
12721    534.8581        1             2
16883    540.5890        1             2

P1_LIT101  P1_P101  prev_P1_P101
1190     813.0031        2             1
5395     813.0031        2             1
9597     811.8256        2             1
13776    812.7283        2             1
17938    813.3171        2             1
```

```
Actuator state durations:
P1_MV101 == 0.0
9  9  10  9  9  10  9  9  10  9

P1_MV101 == 1.0
1174  1168  1182  1160  1172

P1_MV101 == 2.0
669  3019  3012  3000  2981

P1_P101 == 1.0
1073  1074  1061  1056  1056

P1_P101 == 2.0
118  3133  3143  3125  3108
```

Listing 4.4: *Example of preliminar system analysis*

From these results we can draw the following conjectures:

- the **probable actuators** are `P1_MV101` and `P1_P101`. `P1_MV101` has three states identified by the values 0, 1, and 2, suggesting it is a multi-state actuator. `P1_P101` has two states identified by the values 1 and 2, indicating a binary actuator;
- there are **two probable measures**: `P1_FIT101` and `P1_LIT101`. `P1_FIT101` has values ranging from 0 to 2.7, `P1_LIT101` has values ranging from 489.6 to 815.1. Based on this information, a further conjecture can be made that `P1_LIT101` represents a tank level measurement;
- apparently there is a probable **spare actuator**, `P1_P102`, whose value is always 1. No related *hardcoded setpoints* were found. From this information, another speculation can be made that the value 1 represents the **OFF state** for binary actuators, while the value 2 represents the **ON state**;
- from the analysis of state changes we can derive some **relative setpoints**. For example, we observe that `P1_P101` changes state from

value 1 (OFF) to value 2 (ON) when the level of P1_LIT101 is approximately 813, and it changes from value 2 (ON) to 1 (OFF) when the level of P1_LIT101 is around 535. We can deduce that P1_P101 is responsible for emptying the tank;

- Regarding the actuators states duration, the very short duration of P1_MV101 in state 0 is observable: since we are unable, at this preliminary stage, to make assumptions about this, we will try to understand the behavior of the system in this actuator state in the next stages of the analysis.

The information obtained here can be used, as mentioned above, to refine the enrichment of the dataset by setting directives in the [DATASET] section of the *config.ini* file, should this be empty or only partially set, or to make the first conjectures about the system, as we have just seen.

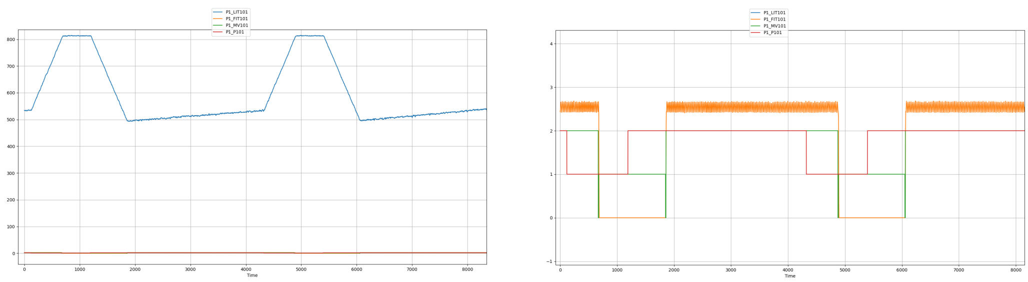
The system_info.py file can also run in standalone mode if needed: it takes as command-line arguments the dataset to be analyzed, a list of actuators, and a list of sensors. For analysis related to state changes, the dataset must mandatorily be of the enriched type.

4.2.2 Phase 2: Graphs and Statistical Analysis

The new *graph analysis* arises from the need to give the user an overview of the (sub)system obtained in the previous pre-processing phase, identifying more easily the typology of the registers and grasping more effectively the relationships and the dynamics that may exist between the registers controlled by one or more PLCs, confirming the initial conjectures if the brief analysis described in the previous section has been performed, or making new ones thanks to the visual graph support.

In Ceccato et al.'s framework, as mentioned in Section 3.2.7, it was only possible to view the chart of one register at a time. While this allowed for the identification or hypothesis of the register type, it made it challenging to establish relationships with other components of the system and derive conjectures about their behavior. To address this limitation, there was a need for a new tool that could provide more information in a more accessible manner.

Initially, we considered adopting an approach similar to Figure 3.5, where all the graphs are displayed within a single plot. However, we soon realized that this solution was not feasible and could not be adopted. Figure 4.5 helps to understand why.

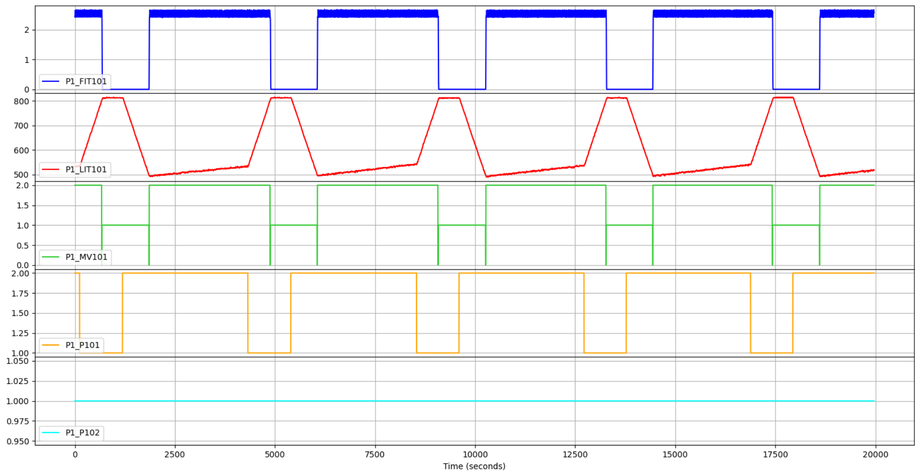


(a) *(b)*

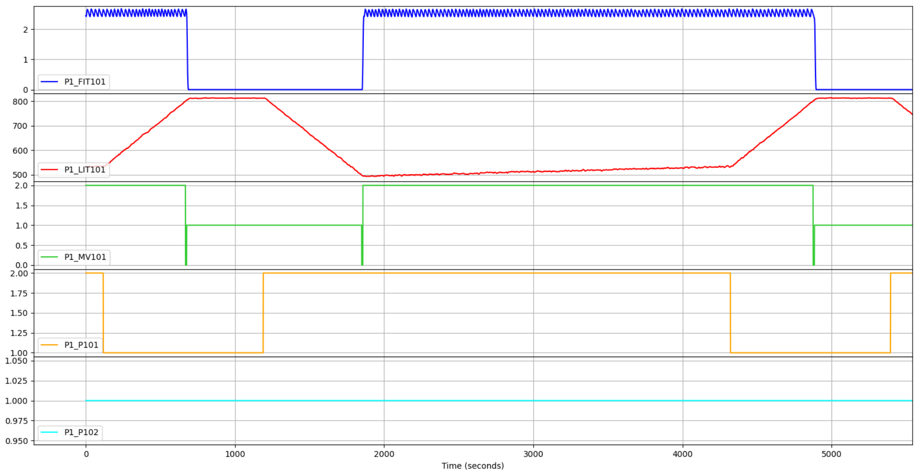
Figure 4.5: *Plotting registers on the same y-axis*

Figure 4.5a highlights the main issue with this approach, which is the use of the same y-axis for all the charts, representing the values of individual registers. When the range between register values is wide, it can result in some charts appearing as a single flat line or becoming indistinguishable, making them difficult to read. Additionally, as shown in Figure 4.5b, when registers have similar values, the graphs can become confusing and harder to interpret.

The solution to this issue is simple and effective: the use of **subplots**. Basically, each register corresponds to a subplot of the graph that shares the time axis (the x-axis) with the other subplots, but keeps the y-axis of the values of each register independent. This maintains the readability and comprehensibility of the charts, while simultaneously being able to immediately grasp the relationships between them. In addition, by sharing the time axis, it is possible to zoom in on a particular area of one of the charts and automatically the other ones will be zoomed in as well, thus not losing any information and no connection between registers. Figure 4.6 illustrates more clearly what has just been explained: the charts refer to the PLC1 registers of the iTrust SWaT system.



***(a)** Example of plotting charts of a PLC registers using subplots*



***(b)** Zooming on a particular zone of the charts*

***Figure 4.6:** Example of the new graph analysis*

To demonstrate the behavior and effectiveness of the new graph analysis process, we observe in particular, Figure 4.6b: we can already have some validation on the conjectures made in the brief analysis from the previous step:

- the data presented in the P1_FIT101 chart provides support for associating this register with a **measurement**. Furthermore, the chart indicates a close correlation between the trend of this register and the periods in which P1_LIT101 exhibits an upward trend and the evolution of P1_MV101. The values of P1_FIT101, ranging from approximately 0 to 2.5, are too limited to conclude that this register represents the level sensor of a tank. The overall trend also aligns with this interpretation, suggesting that it is more likely a **pressure or flow sensor**;

- based on the information provided above, it appears that P1_LIT101 indeed **represents the level sensor of a tank**, which was initially assumed in the brief analysis. The hypothesis is further reinforced by several factors. Firstly, the wide range of values observed in this register supports the notion of it being a level sensor. Additionally, the graphical representation of the level trend provides valuable insights. It shows an initial sharp rise, followed by a period of stabilization referred to as a *plateau*, and subsequently, a similarly steep descent. Finally, there is a new phase of slower level growth compared to the previous one. These patterns lend further support to the interpretation of P1_LIT101 as a sensor specifically measuring the level of a tank;

- at the start of the ascending trend of P1_LIT101, the value of P1_MV101 is 2. Conversely, during the *plateau* phase observed when the measurement is approximately 800 and during the descending trend, P1_MV101 has a value of 1. These observations provide further confirmation of the initial hypothesis: the value 1 represents the **OFF state** of the sensor, while 2 represents the **ON state**. Moreover, P1_MV101 can be identified as the actuator responsible for causing the rising level in P1_LIT101;

- during the brief analysis, we noticed the short duration of P1_MV101 in state 0, but we were unable to speculate on the underlying reason. However, further analysis of the graph reveals that P1_MV101 transitions and remains in the 0 state, acting as a type of "transient" between states 1 and 2. It can be inferred that this period represents the actual time required for the actuator to change its state.

- we can observe the behavior of P1_P101 in relation to P1_LIT101 and P1_MV101 to understand its role. At the beginning of the descending trend of P1_LIT101, after the plateau phase, P1_P101 assumes a value of 2. It then changes back to value 1 at the point of the (presumed) tank's level increase. Although this fact alone may not be entirely clear, when comparing the behaviors of P1_P101, P1_LIT101, and P1_MV101, certain patterns emerge. When P1_P101 and P1_MV101 are both in state 1, the water level remains stable. However, when P1_P101 changes to state 2, the level of the measurement drops rapidly. Subsequently, when P1_MV101 also transitions to state 2, the level slowly starts to rise again, with a sudden increase occurring when P1_P101 changes from 2 back to 1. From these observations, it can be inferred that P1_P101 serves as the **actuator responsible for emptying the presumed tank**. State 1 represents the **OFF state**, while state 2 represents the **ON state** of this actuator.

- it appears that P1_P102 plays no active role in the system as its state remains constant at 1 throughout. Considering this information, it seems unlikely that P1_P102 could serve as a relative setpoint. Therefore, according to the brief analysis, it can be inferred that P1_P102 is most likely a **spare actuator** that is not actively utilized in the system. A *spare actuator* is indeed a secondary actuator that is intended to remains idle unless needed as a backup or replacement when the primary actuator is unavailable or needs to be taken out of service.

As the reader has probably already noticed, the majority of the graphs presented in the previous sections and chapters were generated using the new graph analysis script, specifically the runChartsSubPlots.py script. This Python script is located in the $(project-dir)/statistical-graphs

directory and utilizes the *matplotlib* libraries to generate the graph plots, similar to the previous tool.
The script accepts the following command-line parameters:

- **-f** or **--filename**: specifies the CVS format dataset to read data from. The dataset must be within the directory containing the enriched datasets for the invariant analysis phase. If subsequent parameters are not specified, the script will display all registers in the dataset, excluding any additional attributes;

- **-r** or **--registers**: specifies one or more specific registers to be displayed;

- **-a** or **--addregisters**: adds one or more registers to the default visualization. This option is useful in case an additional attribute such as slope is to be analyzed;

- **-e** or **--excluderegisters**: excludes one or more specific registers from the default visualization. This option is useful to avoid displaying hardcoded setpoints or spare registers.

This script, like the previous ones, is designed to provide the maximum flexibility and ease of use for the user, combined with greater power and effectiveness in deriving useful information about the analyzed system.

Statistical Analysis After careful consideration, we made the decision not to include the statistical analysis aspect of the previous tool in the framework. We found that there was no practical use for it. Instead, we integrated the relevant statistical information into the brief analysis conducted after the pre-processing phase. Additionally, we deemed the histogram to have limited utility and considered it outdated in comparison to the new graph analysis approach we implemented.
Although we decided not to include the statistical analysis aspect in the framework, the Python script `histPlots_Stats.py` from the original tool

remains in the directory. This script is essentially unchanged from the version developed by Ceccato et al. and can be used in the future if the need arises.

4.2.3 Phase 3: Invariant Inference and Analysis

The phase of invariant inference and analysis has undergone a redesign and improvement to offer the user a more comprehensive and easier approach to identify invariants. This has been achieved through the application of new criteria to analyze and reorganize the Daikon analysis results. The outcome of this is a more compact presentation of information that highlights the possible relationships among invariants.
The new design not only enables the identification of undiscovered aspects of the system behavior but also confirms the hypotheses made during the earlier stages of analysis. This step is now semi-automated, unlike before, and allows for the analysis of invariants on individual actuator states and their combinations.

4.2.3.1 Revised Daikon Output

To streamline the process of identifying invariants quickly and efficiently, it is necessary to revise the output generated by standard Daikon analysis. The goal is to create a more compact and readable format for the output.

The current Daikon results basically consist of three sections, referred as *program point sections* [64]:

1. the first section containing generic invariants, i.e., valid regardless of whether a condition is specified for the analysis;

2. the second section containing invariants obtained by specifying a condition for the analysis in the *.spinfo* file, if any;

3. a third section containing the invariants that are obtained from the negation of the condition potentially specified in the *.spinfo* file.

In each section only a single invariant per row is shown, without relating it in any way to the others: this makes it difficult to identify significant invariants and any invariant chain that might provide much more information about the behavior of the system than the single invariant. A brief example of the structure and format of this output related to `PLC1` of the iTrust SWaT system is shown in Listing 4.5, where a condition was specified on the measurement `P1_LIT101` and on actuator `P1_MV101`:

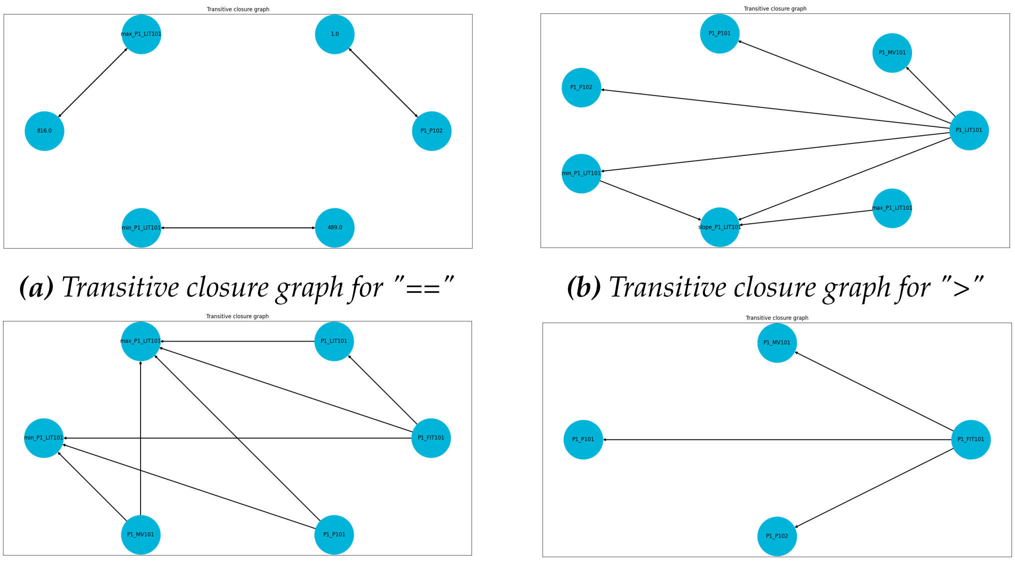
```
aprogram.point:::POINT
P1_P102 == prev_P1_P102
P1_FIT101 >= 0.0
P1_MV101 one of { 0.0, 1.0, 2.0 }
P1_P101 one of { 1.0, 2.0 }
P1_P102 == 1.0
max_P1_LIT101 == 816.0
min_P1_LIT101 == 489.0
slope_P1_LIT101 one of { -1.0, 0.0, 1.0 }
[...]
P1_LIT101 > P1_MV101
P1_LIT101 > P1_P101
P1_LIT101 > P1_P102
P1_LIT101 < max_P1_LIT101
P1_LIT101 > min_P1_LIT101
[...]
P1_MV101 < min_P1_LIT101
P1_MV101 < trend_P1_LIT101
P1_P101 >= P1_P102
P1_P101 < max_P1_LIT101
[...]
===============================================
aprogram.point:::POINT;condition="P1_MV101 == 2.0 &&
  ↪ P1_LIT101 < max_P1_LIT101 - 16 && P1_LIT101 >
  ↪ min_P1_LIT101 + 15"
P1_MV101 == prev_P1_MV101
P1_P102 == slope_P1_LIT101
P1_MV101 == 2.0
```

```
P1_FIT101 > P1_MV101
P1_FIT101 > P1_P101
P1_FIT101 > P1_P102
P1_FIT101 > prev_P1_P101
P1_MV101 >= P1_P101
P1_MV101 >= prev_P1_P101
P1_P101 <= prev_P1_P101
===============================================
aprogram.point:::POINT;condition="not(P1_MV101 == 2.0 &&
 ↪ P1_LIT101 < max_P1_LIT101 - 16 && P1_LIT101 >
 ↪ min_P1_LIT101 + 15)"
P1_P101 >= prev_P1_P101
Exiting Daikon.
```

Listing 4.5: *Standard Daikon output for* PLC1 *of the iTrust SWaT system*

In the presented framework, the output is simplified to **two sections**: the *general section* and the section related to the *user-specified condition*. The section related to the negated condition is eliminated as it is not relevant in this context and could lead to potential misinterpretation. Moreover, the relationships between invariants will be emphasized by utilizing **transitive closures**: transitive closure of a relation R is another relation, typically denoted R^+ that adds to R all those elements that, while not necessarily related directly to each other, can be reached by a *chain* of elements related to each other. In other words, the transitive closure of R is the smallest (in set theory sense) transitive relation R such that $R \subset R^+$ [65].

To implement the transitive closure of the invariants generated by Daikon, we initially categorized the invariants in each section by type based on their mathematical relation ($==$, $>$, $<$, $>=$, $<=$, $!=$), excluding any invariant related to additional attributes except for the slope. For each type of invariant, we constructed a **graph** using the NetworkX library, where registers were represented as nodes, and arcs were created to connect registers that shared a common endpoint in the invariant, applying the transitive property. To reconstruct the individual invariant chains, we employed a straightforward approach known as *Depth-first Search* (DFS) on each of the graphs. This method allowed us to traverse the graphs and obtain the desired outcome, identifying the invariant chains. Figure 4.7 shows some examples of these graphs:



***(a)** Transitive closure graph for "=="*

***(b)** Transitive closure graph for ">"*

***(c)** Transitive closure graph for "<"*

***(d)** Transitive closure graph for ">" cond.*

***Figure 4.7:** Example of transitive closure graphs for invariants in PLC1 of iTrust SWaT system*

At the end of this process, still applied to `PLC1` of the iTrust SWaT system and with the same analysis condition, we get the following complete output:

```
===========================
Generic
===========================
P1_MV101 one of { 0.0, 1.0, 2.0 }
P1_P101 one of { 1.0, 2.0 }
slope_P1_LIT101 one of { -1.0, 0.0, 1.0 }
P1_FIT101 != P1_P101, P1_P102
P1_P102 == 1.0
max_P1_LIT101 == 816.0
min_P1_LIT101 == 489.0
P1_LIT101 > P1_MV101
P1_LIT101 > P1_P101
P1_LIT101 > P1_P102
```

```
14  P1_LIT101 > min_P1_LIT101 > slope_P1_LIT101
15  P1_FIT101 >= 0.0
16  P1_P101 >= P1_P102 >= slope_P1_LIT101
17
18  ===========================
19  P1_MV101 == 2.0 && P1_LIT101 < max_P1_LIT101 - 16 &&
      ↪ P1_LIT101 > min_P1_LIT101 + 15
20  ===========================
21  slope_P1_LIT101 == P1_P102
22  P1_MV101 == 2.0
23  P1_FIT101 > P1_MV101
24  P1_FIT101 > P1_P101
25  P1_FIT101 > P1_P102
26  P1_MV101 >= P1_P101
```

Listing 4.6: *Revised Daikon output with transitive closures for* `PLC1` *of the iTrust SWaT system*

Transitive closures can be appreciated in lines 7, 14 and 16 of Listing 4.6. In general, the output has been reduced in the number of effective rows (19 versus the 61 in the output of Listing 4.5, making it certainly better to read and identify significant invariants) and the invariant chains make it more immediate to grasp the relationships between registers.

4.2.3.2 Types of Analysis

In contrast to Ceccato et al.'s solution, which involves manual individual analyses, our proposal is to introduce **two types of semi-automated analysis**.

The first type focuses on analyzing **all states for each individual actuator**. This automated analysis aims to provide comprehensive insights into the behavior of each actuator in relation to a specific measurement selected by the user.

The second type of analysis considers the current system configuration and examines the **actual states of the actuators**. This analysis takes into account the interplay and combined effects of multiple actuators on the selected measurement. By incorporating the real-time states of the actuators, this semi-automated analysis offers a more accurate assessment of the system behavior.

These two types of analysis will be handled by the Python script `daikonAnalysis.py`, contained in the default directory `$(project_dir)/daikon`. The script accepts the following command-line arguments:

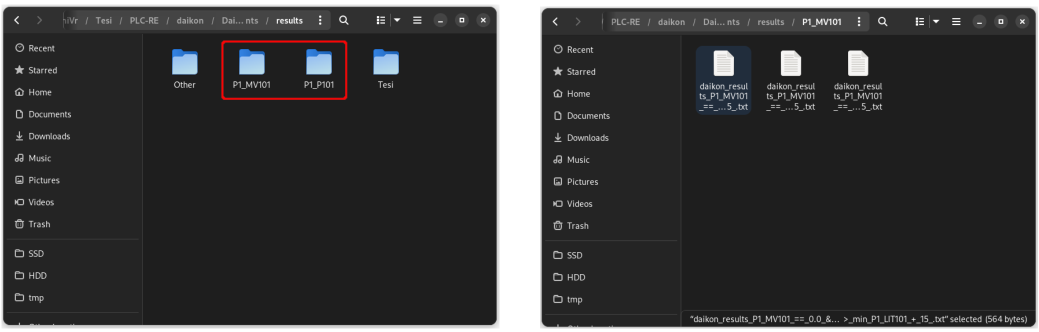
- **-f** or **--filename**: specifies the enriched dataset, in CSV format, from which to read the data. The dataset must be located within the directory containing the enriched datasets specified in the *config.ini* file (by default `$(project_dir)/daikon/Daikon_Invariants`);
- **-s** or **--simpleanalysis**: performs the analysis on the states of individual actuators;
- **-c** or **--customanalysis**: performs the analysis on combinations of actual states of the actuators;
- **-u** or **--uppermargin**: defines a percentage margin on the maximum value of the measurement;
- **-l** or **--lowermargin**: defines a percentage margin on the minimum value of the measurement;

The selection of one or both types of analysis is possible. Additionally, the last two parameters can be used to set a condition on the value of the measurement. This condition is designed to bypass the transient periods that occur during actuator state changes and the actual trend changes at the maximum and minimum values of the measurement.
This approach proves particularly beneficial for the first type of analysis, as it enables more accurate data on the trends of the measurement. By excluding the transient periods, the analysis can focus on the stable and meaningful trends, providing improved insights into the behavior of the system.

Analysis on single actuator states Analysis on the states of individual actuators is the simplest: after the user is prompted to input the measurement, chosen from a list of likely available measurements, the script recognizes the likely actuators and the relative states of each, using the same mixed Daikon/Pandas technique adopted in the brief analysis during the pre-processing phase.

For each actuator and each state it assumes, a single Daikon analysis is performed, eventually placing the condition on the maximum and minimum level of the measurement.

The result of these analyses are saved in the form of text files in a directory having the name corresponding to the analyzed actuator and contained in the default parent directory `$(project_dir)/daikon/Daikon_Invariants/results`: each file generated by the analysis is identified by the name of the actuator, the state and the condition, if any, on the measurement (see Figure 4.8).



(a) *(b)*

Figure 4.8: *Directory (a) and outcome files (b) for the single actuator states analysis*

Listing 4.6 provides an illustrative example of the analysis results. In this case, we focus on the actuator `P1_MV101` in state 2. By examining the generic invariants, we can deduce that `P1_P101` is a probable actuator with binary values, as indicated in line 5. However, upon closer inspection in line 8, we discover that `P1_P102` consistently maintains a value of 1 throughout, thus confirming its role as a spare actuator rather than a setpoint. Furthermore, lines 9 and 10 present the maximum and minimum values achieved by `P1_LIT101`, the register presumed to be connected to

the tank.
The condition-generated invariants provide the most interesting insights. In particular, at line 21, we observe that when `P1_MV101` is set to 2, the slope of `P1_LIT101` is 1, indicating an increasing trend. This confirms our previous assumption that `P1_MV101` represents the actuator's ON state and is responsible for filling the tank (`P1_LIT101`).
Furthermore, at line 23, we discover that `P1_FIT101` is greater than `P1_MV101`. This implies that when the actuator assumes the value 2, the sensor `P1_FIT101` registers a measurement. In contrast, when `P1_MV101` is 1, `P1_FIT101` remains at 0, signifying no recorded readings. This finding reinforces our initial hypothesis that `P1_FIT101` may serve as a pressure or flow sensor.

Analysis of the Current System Configuration The analysis of the current system configuration based on the actual states of the actuators is more complex, but at the same time offers more interesting outcomes as it provides better evidence of actuator behavior in relation to the selected measurement.

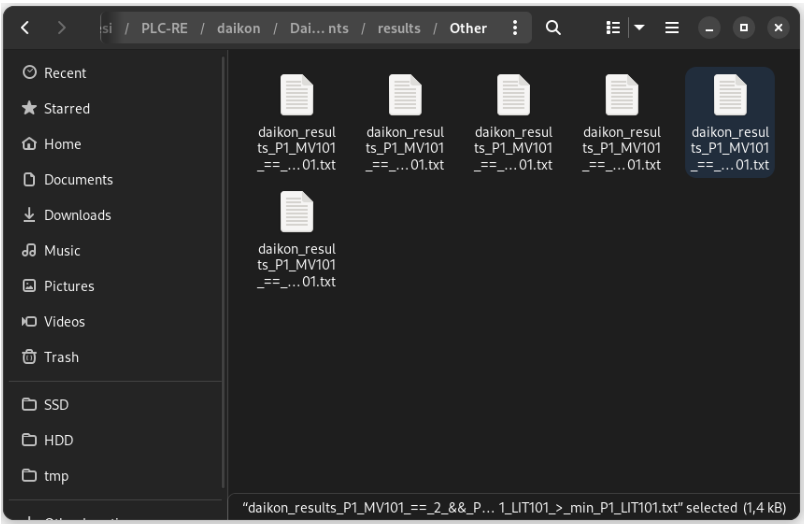


Figure 4.9: *Daikon outcome files for system configuration analysis. Each file represents a single system state*

For this analysis, the script automatically identifies the configurations of the actuators that represent different system states (e.g., `P1_MV101 ==`

`2, P1_P101 == 1)`. Daikon analysis is then performed for each of these configurations. The user is prompted to select the measurement attribute and, if desired, specific actuators for studying their configurations. If no actuators are selected, all previously detected actuators will be considered. The analysis results are saved in text format within a designated directory, located alongside the previous analysis outputs. The filenames of these result files follow a specific naming convention based on the analysis rule applied (see Figure 4.9).

An example of the obtained outcomes can be seen in Listing 4.7:

```
===========================
Generic
===========================
P1_MV101 <= P1_P101  ==>  P1_FIT101 >= 0.0
P1_MV101 <= P1_P101  ==>  P1_MV101 one of { 0.0, 1.0, 2.0 }
P1_MV101 <= P1_P101  ==>  P1_P101 one of { 1.0, 2.0 }
P1_MV101 <= P1_P101  ==>  slope_P1_LIT101 one of { -1.0, 0.0, 1.0 }
P1_MV101 > P1_P101  ==>  P1_FIT101 > P1_MV101
P1_MV101 > P1_P101  ==>  P1_FIT101 > P1_P101
P1_MV101 > P1_P101  ==>  P1_FIT101 > P1_P102
P1_MV101 > P1_P101  ==>  P1_FIT101 > slope_P1_LIT101
P1_MV101 > P1_P101  ==>  P1_MV101 == 2.0
P1_MV101 > P1_P101  ==>  P1_MV101 > P1_P102
P1_MV101 > P1_P101  ==>  P1_MV101 > slope_P1_LIT101
P1_MV101 > P1_P101  ==>  P1_P101 == 1.0
P1_MV101 > P1_P101  ==>  P1_P101 == P1_P102
P1_MV101 > P1_P101  ==>  P1_P101 == slope_P1_LIT101
P1_MV101 > P1_P101  ==>  slope_P1_LIT101 == 1.0
[...]

===========================
P1_MV101 == 2 && P1_P101 == 1 && P1_LIT101 < max_P1_LIT101 && P1_LIT101 > min_P1_LIT101
===========================
slope_P1_LIT101 == P1_P102 == P1_P101 == 1.0
```

```
25 P1_MV101 == 2.0
26 P1_FIT101 > P1_MV101
27 P1_FIT101 > P1_P101
```

Listing 4.7: *Daikon outcomes for the system configuration* `P1_MV101 == 2, P1_P101 == 1` *on* `P1_LIT101`

In contrast to Listing 4.6, the current analysis reveals the presence of implications in the general invariants section, which were previously absent (remaining generic invariants omitted for brevity). These implications offer valuable insights, as demonstrated in this case. For instance, the invariant stated in line 18 informs us that if the value of `P1_MV101` in greater than `P1_P101`, then the slope's value is 1, indicating an increasing tank level. This finding further corroborates our initial understanding that the state `P1_MV101 == 2` represents the ON state for the actuator responsible for filling the tank, namely `P1_LIT101`. Upon comparing the results of the other analyses, we will uncover that `P1_P101` serves the purpose of emptying the tank, with its ON and OFF states denoted as 2 and 1, respectively.
Furthermore, the invariant presented in line 8 reveals that when `P1_MV101` is in state 2 and `P1_P101` is in state 1, the value of `P1_FIT101` ins greater than 2. Consequently, it can be inferred that the associated sensor is measuring something in relation to the tank represented by `P1_LIT101`.
The aforementioned observations are further supported by the invariants associated with the analysis condition. As indicated in line 24, the slope is indeed equal to 1, confirming an upward trend. Additionally, at line 26, it is evident that `P1_FIT101` assumes values greater than 2 when `P1_MV101` is equal to 2.

Refining the Analysis In certain situations, the outcomes provided by the semi-automated analyses may not meet the user's expectations. For instance, the clarity of the slope value may be insufficient, or the user may wish to delve deeper into a specific aspect of the system to uncover additional invariants that were not previously identified. In such cases, the user has the option to conduct a more targeted and specific invariant analysis using the Python script `runDaikon.py`, which enables precise investi-

gations of the system.
The script, located in the default directory `$(project_dir)/daikon`, can be executed with three command-line parameters:

- **-f** or **--filename**: specifies the CSV format *enriched* dataset to read data from. Even in this case, the dataset must be located within the directory containing the enriched datasets;
- **-c** or **--condition**: specifies the condition for the analysis, which will be automatically overwritten in the *.spinfo* file. It is possible to specify more than one condition, but it is strongly recommended to use the logical operator `&&` to avoid undesired behaviors in Daikon outcomes;
- **-r** or **--register**: specifies the directory where to save the text file with the outcomes.

During the execution of this analysis, a single output file is generated, containing the discovered invariants. The user can specify the directory where this file should be saved using the `-r` command-line option. By default, the file is stored in the directory `$(project_dir)/daikon/Daikon_Invariants/results`. The output file serves as a record of the identified invariants and can be examined at a later time.

In conclusion, the integration of these two analysis types, along with the ability to conduct more refined analysis in the future and the enhanced output format of Daikon, significantly enhances the completeness, clarity, and effectiveness of this stage compared to the previous framework.

4.2.4 Phase 4: Businness Process Analysis

We have made significant revisions to the Business Process Analysis we are presenting. Instead of relying on the previous Java solution and proprietary process mining software Disco, we have adopted a **new integrated solution** created in Python from scratch. The new solution utilizes the Graphviz libraries to generate the corresponding activity diagram.

In this updated Business Process, greater emphasis is placed on process mining related to the physical system. Our goal is to extract as much information as possible from the dataset, enabling us to promptly visualize the system's behavior and its various states. This approach allows us to validate the conjectures and hypotheses formulated in the previous phases, and potentially uncover hidden patterns that were previously undisclosed.
On the other hand, the aspect related to network communications was reconsidered to enable operation with multiple protocols, expanding beyond the limitations of Modbus. Unfortunately, despite our intentions, we were unable to implement this modification due to reasons that will be elaborated on later.

Now, let's examine the key aspects of this new phase in greater detail.

4.2.4.1 Process Mining of the Physical Process

The mining of the physical process is performed by the Python script called `processMining.py`, located in the default directory `$(project_dir)/process-mining`. This script accepts the following parameters from the command line:

- **-f** or **--filename:** specifies the CSV format *timestamped* dataset to be mined from. The dataset is obtained from the pre-processing stage and is located in the default directory `$(project_dir)/process-mining/data`;

- **-a** or **--actuators:** specifies one ore more actuators whose combinations of states are to be analyzed. If this parameter is not provided, all actuators in the subsystem will be considered;

- **-s** or **--sensors:** specifies one or more measurements for which the trend will be calculated based on actuator state changes. If this parameter is omitted, all available measurements will be considered;

- **-t** or **--tolerance:** specifies the tolerance to be taken into account during the trend calculation;

- **-g** or **--graph:** shows the resulting activity diagram.

The script processes the dataset in a sequential manner, analyzing each actuator state change. It calculates the duration in seconds of the system state, as well as the trend and slope of the specified measurement(s). Additionally, the script stores the next system state and the measurement values corresponding to the state change points, allowing for the identification of relative setpoints. These results are gradually collected and stored in a dictionary, where the keys represent the system states based on the actuator configurations, and the values represent the measured values mentioned above. The dictionary is then saved as a JSON file, which can be accessed by the user in the `$(project_dir)/process-mining/data` directory. The name of the file can be specified in the configuration file *config.ini*.

An example of the JSON file obtained in this step is shown in Listing 4.8: the JSON file showcases the structure of the data obtained during the process.


```
{
  "P1_MV101 == 2, P1_P101 == 2": {
    "start_value_P1_LIT101": [
    534.9366,
    495.4483,
    497.9998,
    495.9586,
    495.8016
    ],
    "end_value_P1_LIT101": [
    536.0356,
    532.7384,
    541.7273,
    534.4656,
    540.8245
    ],
```

```
      "slope_P1_LIT101": [
      0,
      0.015,
      0.018,
      0.016,
      0.018
      ],
      "trend_P1_LIT101": [
      "STBL",
      "ASC",
      "ASC",
      "ASC",
      "ASC"
      ],
      "time": [
      119,
      2464,
      2477,
      2452,
      2440
      ],
      "next_state": [
      "P1_MV101 == 2, P1_P101 == 1",
      "P1_MV101 == 2, P1_P101 == 1",
      "P1_MV101 == 2, P1_P101 == 1",
      "P1_MV101 == 2, P1_P101 == 1",
      "P1_MV101 == 2, P1_P101 == 1"
      ]
    },
    "P1_MV101 == 2, P1_P101 == 1": {
      ...
    }
    "P1_MV101 == 0, P1_P101 == 1": {
      ...
    }

    ...
  }
```

Listing 4.8: *Example of the data contained in the produced JSON file*

The collected data is now utilized to generate the **system activity diagram**. This diagram represents an *oriented* graph where the nodes correspond to the system states determined by the actuator configurations. In addition to the state information, the nodes also display the trend (ascending, descending, or stable) and the slope of the reference measurement.
The edges in the diagram depict the values of the measurements at the time of the state change. These measurement values represent the setpoints for each measurement. Setpoints are calculated on the average of the values measured for that specific state. Additionally, the diagram incorporates on the edges the average duration of each system state.
Each data point on the arcs is accompanied by a *standard deviation value*. This value provides information about the occurrence of a specific state within the system's cycle, indicating whether it appears multiple times with varying values and time durations. A low standard deviation suggests that the state is likely to occur only once within each cycle. Conversely, a high standard deviation indicates that the state may occur multiple times within the cycle.

An example of the activity diagram generated by the `processMining.py` script is presented in Figure 4.10. This diagram illustrates the system's behavior by depicting transitions between different states. Nodes in the diagram represent specific system states, while arrows or edges indicate the flow between these states.
The diagram reveals important aspects of the system, such as its cyclicality and the emphasis on the temporal sequence of actuator states. These insights might not have been clearly evident in earlier stages of the analysis. The diagram provides a visual representation that allows us to appreciate the recurring patterns and understand how the system evolves over time. By observing the transitions and relationships between different states, we gain a better understanding of the dynamics and behavior of the system.

It is important to acknowledge that despite considering the tolerance set for trend and slope calculation, the accuracy of the related data may vary. For instance, there could be instances where multiple trends exist

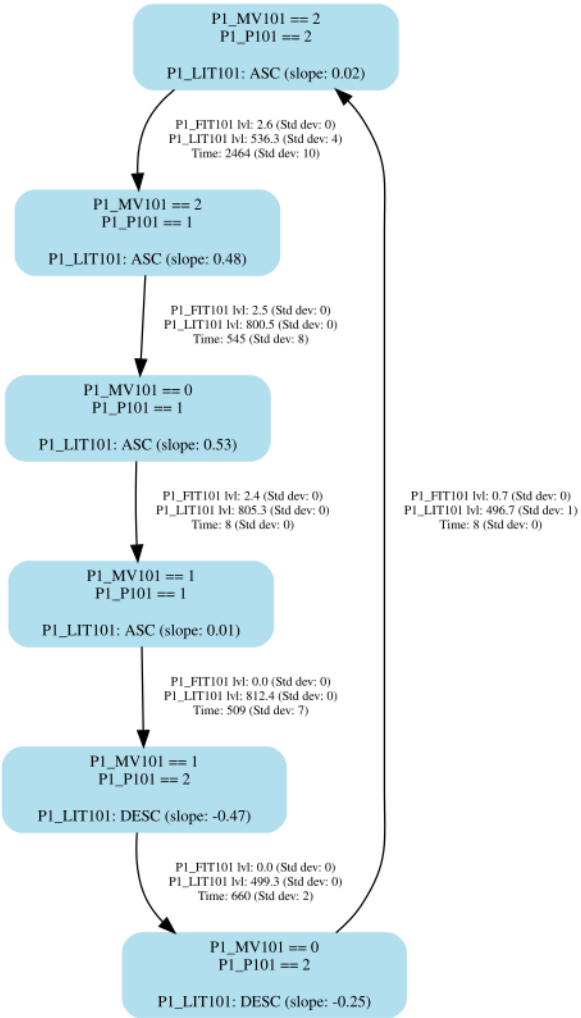


Figure 4.10: *Activity diagram for PLC1 of the iTrust SWaT system*

within the same node, or a trend may be stable instead of increasing or decreasing. These discrepancies arise due to data perturbations, which were discussed in Section 4.2.1.2. Additionally, smoothing techniques were not employed in this analysis. Therefore, it becomes the responsibility of the user to correctly interpret the outcomes of this step, taking into account the information presented in the process mining diagram and the findings from previous analyses. By considering these factors together, the user can make informed interpretations and draw accurate conclusions from the results.

4.2.4.2 Network Communications

The incorporation of network communications into Business Process Analysis has been reconsidered to shift away from a *single-protocol* solution based on Modbus. Instead, the focus is on adopting a solution that can handle **multiple protocols**, even at the same time.
The main concept was to develop a new Python script that would extract and process data from PCAP files obtained through network traffic sniffing. The intention was to export this data to a CSV file, which would then be used as input for the process mining script, `processMining.py`, and integrated with the physical system data to derive commands to the actuators.

However, due to the specific case study discussed in Chapter 5 (and especially in Section 5.2.1), **it was not feasible to implement this solution at the time**. The network data available for analysis either lacked references to the reads or writes performed by the actuators or, if such references existed, they were not associated with a suitable physical process dataset for the intended purposes. As a result, the integration of network data with the physical system data could not be realized in the given context.

Nevertheless, a thorough study was conducted on the available PCAP files to explore potential methodologies that could be implemented as future work. The analysis of the network data revealed that different protocols exhibit distinct behaviors when commanding changes in actuator states. Consequently, the previous approach proposed by Ceccato et al. becomes **impractical** when attempting to detect system state changes through network commands sent to the actuators.
However, considering that system state changes have already been identified through the process mining of the physical process, and with the corresponding event timestamps available, we propose an alternative approach to Ceccato et al.'s method. Instead of seeking the correspondence between network events and physical process events at the same instant, we suggest reversing the perspective. By focusing on the correspondence between a given event occurring in the physical process at a specific mo-

ment and the corresponding event in the network data at the same moment, it should be possible to achieve a similar, if not superior, outcome compared to the previous solution.

This *inverse approach* aims to leverage the existing knowledge of system state changes obtained through the mining of the physical process. By aligning these physical events with their corresponding events in the network data, a more effective analysis can be conducted.

Although the integration of network data within Business Process Analysis was not feasible, the progress made thus far has resulted in the development of a **novel form of network analysis**. This approach, unlike the work conducted by Ceccato et al., offers a broader perspective on network communications and delves into previously unexplored aspects of the system. These additional dimensions provide a deeper understanding of the system's behavior and characteristics.

In the upcoming section, we will delve further into this new network analysis methodology, discussing its unique features and the insights it can offer. By expanding the scope of our analysis, we aim to uncover valuable information that was not extensively explored in previous studies conducted by Ceccato et al.

4.2.5 Phase 5: Network Analysis

Chapter 5

Case study: the iTrust SWaT System

HAVING introduced the innovative framework and highlighted its potential in the preceding chapter, we now turn our attention to the case study where we will apply this framework. As previously mentioned in Chapter 4 and demonstrated through various examples in the same chapter, our focus will be on the **iTrust SWaT system** [36], developed by the iTrust – Center for Research in Cyber Security of University of Singapore for Technology and Design [30]. The acronym SWaT represents ***S**ecure **Wa**ter **T**reatment*.

The iTrust SWaT system is a testbed that replicates on a small scale a real water treatment plant arises to support research in the area of cyber security of industrial control systems and has been operational since March 2015: it is still being used by students at the University of Singapore for educational and training purposes and is available to organizations to train their operators on cyber physical incidents.

5.1 Architecture

In contrast to the virtualized testbed discussed in Section 3.2.1 by Ceccato et al., the iTrust SWaT system is composed entirely of physical hardware components. It encompasses various elements, starting from field devices and extending to PLCs, HMI, SCADA workstations, and the SCADA server (also referred to as the *historian*). The historian is responsible for recording data from field devices for further analysis. In the upcoming sections, we will delve deeper into the architecture of the physical process and the communication network.

5.1.1 Physical Process

The physical process of the SWaT consists of six stages, denoted P1 through P6. These stages are [66][67]:

1. **taking in raw water:** feeds unfiltered water into the system
2. **chemical dosing:** adds chemicals to water useful for initial pretreatment
3. **Ultra Filtration (UF) system:** the water is filtered through a semi-permeable membrane (ultrafiltration membrane) using the liquid pressure, effectively capturing impurities and suspended solids, as well as removing bacteria, viruses, and other pathogens present in the water.
4. **dechlorination:** removes residual chlorine from disinfected water using ultraviolet lamps
5. **Reverse Osmosis (RO):** performs further filtration of the water
6. **backwash process:** cleans the membranes in UF using the water produced by RO

Figure 5.1 shows a graphical representation of the architecture and the six stages of the SWaT system.

The SWaT system incorporates an array of sensors that play a crucial role in monitoring the system's operations and ensuring their safe. These sensors are responsible for continuously collecting data and providing valuable insights into the functioning of the system. These sensors are:

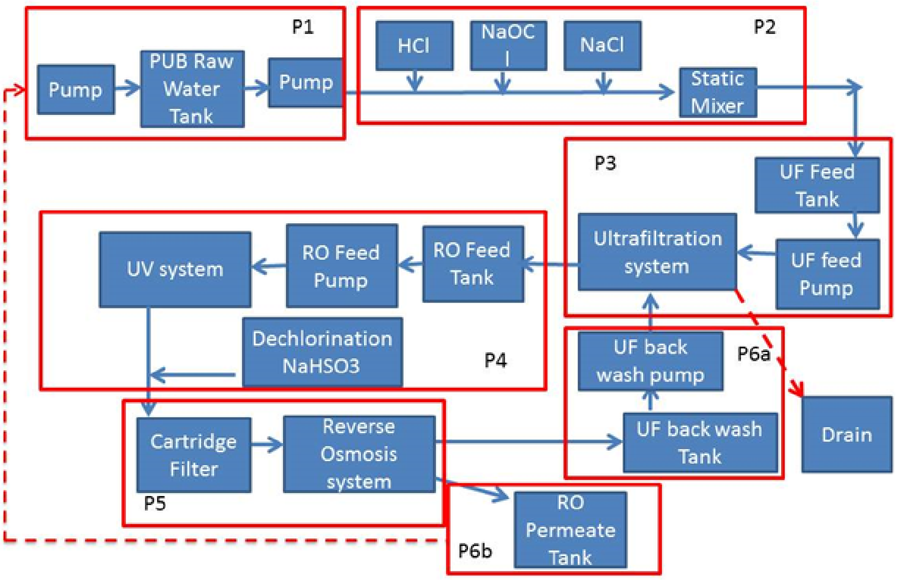


Figure 5.1: SWaT architecture

- Level Indication Transmitter (measured in mm)
- Flow Indication Transmitter (m3/hr)
- Analyser Indicator Transmitter
    - o Conductivity (µS/cm)
    - o pH
    - o Oxidation Reduction Potential (mV)
- Differential Pressure Indicator Transmitter (kPa)
- Pressure Indicator Transmitter (kPa)

The sensors and actuators associated with each PLC are shown in Figure 5.2.

Sensors and actuators are mapped to tags by the communication protocol used (see 5.1.2): a tag can be addressed via string descriptor defined by the system designer (e.g. MV101, to indicate motorized valve number 1 at stage 1) or by referring directly to the analog/digital pins of the PLC I/O unit [67].

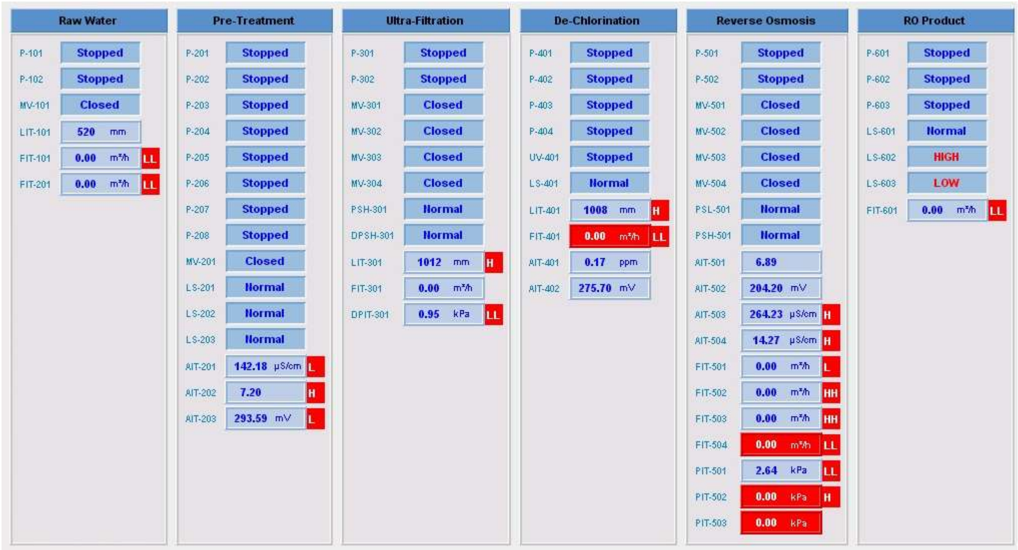


Figure 5.2: *Sensors and actuators associated with each PLC*

5.1.2 Control and Communication Network

The SWaT system's network architecture follows the principles of layering and zoning, which enable segmentation and control of traffic within the network.

Five layers are present starting from the highest to the lowest:

- **Layer 3.5** – Demilitarized Zone (DMZ)
- **Layer 3** – Operation Management (Historian)
- **Layer 2** – Supervisory Control (Touch Panel, Engineering Workstation, HMI Control Clients)
- **Layer 1** – Plant Control Network (PLCs) (Star Network)
- **Layer 0** – Process (Actuator/Sensors and Input/output modules) (Ring Network)

PLCs at Layer 1 communicate with their respective sensors and actuators at Layer 0 through a conventional ring network topology based on EtherNet/IP, to ensure that the system can tolerate the loss of a single link without any adverse impact on data or control functionality. PLCs between the different process stages at Layer 1 communicate with each other through a star network topology using the CIP protocol on EtherNet/IP, previously discussed in Section 2.2.6.3.

Regarding zoning, the SWaT system is divided into three zones, each containing one or more layers. These zones are, in descending order of security level:

- **Plant Control Network**, or **Control System:** includes layers from 0 to 2
- **DMZ:** includes Layer 3.5
- **Plant Network:** includes Layer 3

Figure 5.3 provides a clearer visualization of the zoning and layer division within the network architecture of the SWaT system. This diagram highlights the distinct zones and their corresponding layers, offering a comprehensive overview of the system's network structure.

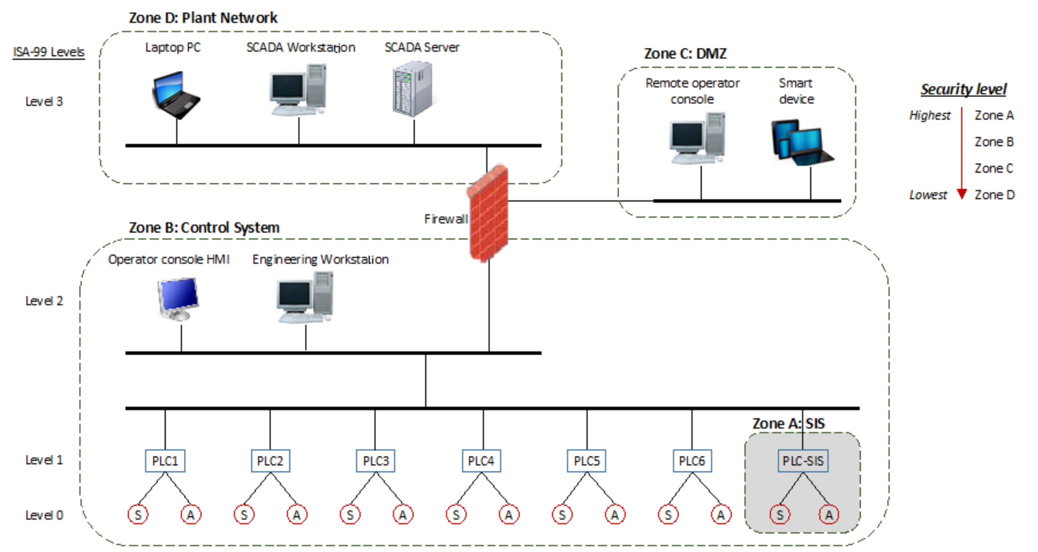


Figure 5.3: SWaT network architecture

A specific IP address is associated with each device: in Table 5.1 we report the addresses for the PLCs, historian, and Touch Panel in the *Plant Control Network* (PCN) zone:

IP Address	Device
192.168.1.10	PLC1
192.168.1.20	PLC2
192.168.1.30	PLC3
192.168.1.40	PLC4
192.168.1.50	PLC5
192.168.1.60	PLC6
192.168.1.100	Touch Panel (PCN)
192.168.1.200	Historian Server
192.168.1.201	Engineering Workstation

Table 5.1: *main IP addresses of the six PLCs and SCADA in SWaT*

5.2 Datasets

To facilitate the study and testing of technologies related to cyber security in Industrial Control Systems and critical infrastructure, iTrust offers researchers worldwide the opportunity to access a range of datasets [68]. These datasets consist of a collection of data obtained from the SWaT system, encompassing information on both the physical processes and network communications. The data is organized into different years, and researchers can request access to these datasets for their analysis and experimentation purposes.

Physical Process Datasets The datasets containing information about the physical processes are provided in CSV format files. These files encompass data collected during different time intervals, which can vary from a few hours to entire days. The granularity of the data is typically at a one-second interval, although there may be some exceptions. The collected data primarily consists of timestamped sensor measurements and actuator status values for each PLC, describing the physical properties of the testbed in operational mode.

Network Communications Datasets Network communications are typically available in the form of *Packet Capture* format (PCAP) files. These files contain captures of communication network traffic, allowing researchers to analyze and examine the network interactions. In some instances, CSV files are provided instead of PCAP files, featuring different characteristics for the collected data.

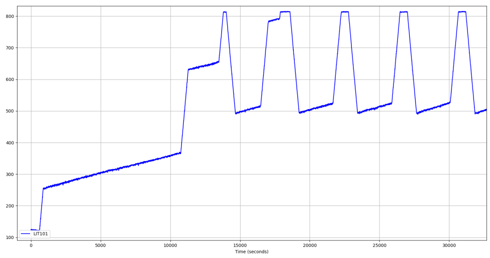
5.2.1 Our Case Study: the 2015 Dataset

The dataset selected as a case study to apply the framework discussed in the previous chapter is specifically the dataset from the year 2015 [69]. The main reason for this choice is the unique characteristics found in the physical process dataset that are not present in datasets from subsequent years.

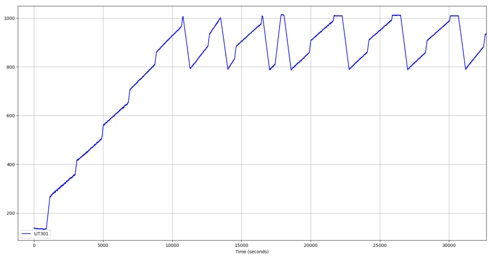
Physical Process Data The data collection process lasted 11 consecutive days, 24 hours per day. During the first 7 days, the system operated normally without any recorded attacks. However, attacks were observed during the remaining 4 days. The collected data reflects the impact of these attacks, leading to the creation of two separate CSV files: one containing the recorded data of SWaT during the system's regular operations, and the other containing data recorded during the days of the attacks. To ensure accurate information about the system, the dataset pertaining to the normal operations, which spans seven days, was chosen for analysis.

Data collection occurs at a frequency of one data point per second, with the assumption that significant attacks cannot occur within a shorter time frame. Additionally, the firmware of the PLCs remains unchanged throughout the data collection period.

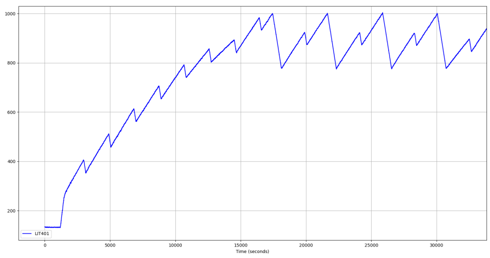
At the beginning of data gathering the tanks are empty and the system must be initialized in order to then reach full operation: it typically takes around five hours for all tanks to be fully filled and for the system to stabilize and reach the appropriate operational state (see Figure 5.4).



***(a)** Stabilization of* `P1_LIT101`



***(b)** Stabilization of* `P3_LIT301`



***(c)** Stabilization of* `P4_LIT401`

***Figure 5.4:** SWaT stabilization*

In total, the dataset consists of thousand and thousand of samples related to 51 attributes.

Network Traffic The network traffic was collected using an appliance from a well-known network hardware manufacturer and was made available only in CSV format and not PCAP format. Table 5.2 shows some of the main data captured:

Category	Description
Date	Date of Log
Time	Time of Log
Origin	IP of server
Source IP	IP address of source

Destination IP	IP address of destination
Protocol	Network protocol
Application Name	Name of application
Modbus Function Code	Function Code
Modbus Function Description	Description of function
Modbus Transaction ID	Transaction ID
SCADA Tag	Sensor or actuator ID
Modbus Value	Value transmitted
Service / Destination Port	Port number of destination IP
Source Port	Port number of source IP

Table 5.2: *SWaT network traffic data*

Unfortunately, the data provided are partial as it only includes readings from sensors and does not include information on actuator status.

Do not mislead by the word *Modbus* in the captured data fields: this is most likely a feature of the sniffing appliance, which may have encapsulated EtherNet/IP-CIP protocol packets in Modbus frames, as described by T. A. Snide in [70].
To provide evidence that the protocol being used is CIP over EtherNet/IP, we can examine the "Source / Destination port" field in the captured data. The presence of TCP port 44818 as the destination port aligns with the TCP port used for transporting **explicit messages** in the EtherNet/IP protocol, as explained in Section 2.2.6.2. Additionally, the "Application name", "Modbus Function Code" and "Modbus Function Description" fields further support this conclusion. The "Application name" explicitly states "CIP_read_tag_service," while the "Modbus Function Code" field contains a decimal value of 76, which does not correspond to any known Modbus Function Calls. When converted to hexadecimal, this value is `4C`, matching the CIP protocol read tag request code as indicated in the "Modbus Function Description" field.

In summary, the datasets for subsequent years were not selected due to certain limitations. In the case of the year 2017, the physical system data had a wide granularity, resulting in significant information loss. Additionally, in some cases, the datasets were considered "spurious" as there was no clear differentiation between data obtained during normal SWaT operations and data associated with attacks.

Regarding network traffic data, there were instances where the data was either missing or fragmented into large PCAP files weighing several gigabytes (more than 6 GB each!), despite containing only a few minutes' worth of data. This made it impractical to manage such files given the available resources.

Despite their large quantity, the CSV files associated with network traffic for the year 2015 are comparatively smaller in size, just slightly over 100 MB each. These files cover a more extensive time period, which facilitates easier management and analysis. Prioritizing the physical process dataset as crucial, I have selected the year 2015 as a case study, even though the network data may be incomplete.

Chapter 6

Our framework at work: reverse engineering of the SWaT system

6.1 Pre-processing

6.2 Graph Analysis

6.2.1 Conjectures About the System

6.3 Invariant Analysis

6.3.1 Actuators Detection

6.3.2 Daikon Analysis and Results Comparing

6.4 Extra information on the Physics

6.5 Businness Process Analysis

Chapter 7

Conclusions

7.1 Discussions

7.2 Future work

List of Figures

List of Tables

Listings

References

[1] NIST. *ICS defintion*. URL: https://csrc.nist.gov/glossary/term/industrial_control_system (visited on 2023-04-06).

[2] Wikipedia. *Stuxnet*. URL: https://en.wikipedia.org/wiki/Stuxnet (visited on 2023-05-18).

[3] Wikipedia. *Industroyer*. URL: https://en.wikipedia.org/wiki/Industroyer (visited on 2023-05-18).

[4] F. Robles and N. Perlroth. "'Dangerous Stuff': Hackers Tried to Poison Water Supply of Florida Town". In: *The New York Times* (2021-02-08). URL: https://www.nytimes.com/2021/02/08/us/oldsmar-florida-water-supply-hack.html (visited on 2023-05-18).

[5] A. Akbarzadeh. "Dependency based risk analysis in Cyber-Physical Systems". PhD thesis. Norwegian University of Science, Technology - Faculty of Information Technology, and Electrical Engineering, 2023. ISBN: 978-82-326-6986-8 (electronic version).

[6] G. M. Makrakis, C. Kolias, G. Kambourakis, C. Rieger, and J. Benjamin. "Industrial and Critical Infrastructure Security: Technical Analysis of Real-Life Security Incidents". In: *IEEE Access* (2021-12-06). DOI: https://dx.doi.org/10.1109/ACCESS.2021.3133348.

[7] NIST. *PLC defintion*. URL: https://csrc.nist.gov/glossary/term/programmable_logic_controller (visited on 2023-04-06).

[8] W. Bolton. *Programmable Logic Controllers, 6th edition*. Newnes, 2015, pp. 7–9.

[9] M. Ceccato, Y. Driouich, R. Lanotte, M. Lucchese, and M. Merro. "Towards Reverse Engineering of Industrial Physical Processes". In: CPS4CIP@ESORICSAt 2022 (Copenhagen, Denmark, Sept. 30, 2022). 2022, 273–290. DOI: `https://doi.org/10.1007/978-3-031-25460-4_15`.

[10] H. S. G. Pussewalage, P. S. Ranaweera, and V. Oleshchuk. "PLC security and critical infrastructure protection". In: 2013 IEEE 8th International Conference on Industrial and Information Systems (Dec. 2013). 2013. DOI: `https://dx.doi.org/10.1109/ICIInfS.2013.6731959`.

[11] Wikipedia. *Remote Terminal Unit*. URL: `https://en.wikipedia.org/wiki/Remote_terminal_unit` (visited on 2023-04-08).

[12] *What is SCADA?* URL: `https://www.inductiveautomation.com/resources/article/what-is-scada` (visited on 2023-04-05).

[13] NIST. *HMI defintion*. URL: `https://csrc.nist.gov/glossary/term/human_machine_interface` (visited on 2023-04-11).

[14] Zscaler. *What Is the Purdue Model for ICS Security?* URL: `https://www.zscaler.com/resources/security-terms-glossary/what-is-purdue-model-ics-security` (visited on 2023-05-17).

[15] Schneider Electrics. *What is Modbus and how does it work?* URL: `https://www.se.com/us/en/faqs/FA168406/` (visited on 2023-04-13).

[16] O. Morando. *Hacking: Modbus - Cyber security OT/ICS*. URL: `https://blog.omarmorando.com/hacking/ot-ics-hacking/hacking-modbus` (visited on 2023-04-15).

[17] Modbus.org. "MODBUS/TCP Security". In: pp. 7–8. URL: `https://modbus.org/docs/MB-TCP-Security-v21_2018-07-24.pdf` (visited on 2023-04-16).

[18] ODVA Inc. "EtherNet/IP - CIP on Ethernet Technology". In: URL: https://www.odva.org/wp-content/uploads/2021/05/PUB00138R7_Tech-Series-EtherNetIP.pdf (visited on 2023-04-18).

[19] *Odva, Inc.* URL: https://www.odva.org (visited on 2023-04-21).

[20] *Introduction to EtherNet/IP Technology*. URL: https://scadahacker.com/library/Documents/ICS_Protocols/Intro%20to%20EthernetIP%20Technology.pdf (visited on 2023-04-19).

[21] ODVA Inc. *Common Industrial Protocol*. URL: https://www.odva.org/technology-standards/key-technologies/common-industrial-protocol-cip/ (visited on 2022-12-26).

[22] Wikipedia. *Common Industrial Protocol*. URL: https://en.wikipedia.org/wiki/Common_Industrial_Protocol (visited on 2023-04-21).

[23] *What is the Common Industrial Protocol (CIP)?* URL: https://www.motioncontroltips.com/what-is-the-common-industrial-protocol-cip/ (visited on 2023-04-21).

[24] *CIP (Common Industrial Protocol): CIP messages, device types, implementation and security in CIP*. URL: https://resources.infosecinstitute.com/topic/cip/ (visited on 2023-04-21).

[25] B. Green, M. Krotofil, and A. Abbasi. "On the Significance of Process Comprehension for Conducting Targeted ICS Attacks". In: Workshop on Cyber-Physical Systems Security and PrivaCy (Nov. 2017). ACM, 2017, pp. 57–67. DOI: https://doi.org/10.1145/3140241.3140254.

[26] A. Keliris and M. Maniatakos. "ICSREF: A Framework for Automated Reverse Engineering of Industrial Control Systems Binaries". In: 26th Annual Network and Distributed System Security Symposium (NDSS) (Feb. 2019). 2019. DOI: https://doi.org/10.48550/arXiv.1812.03478.

[27] Wikipedia. *CODESYS*. URL: https://en.wikipedia.org/wiki/CODESYS (visited on 2023-05-18).

[28] Y. Yuan, X. Tang, W. Zhou, W. Pan, X. Li, H. T. Zhang, H. Ding, and J. Goncalves. "Data driven discovery of cyber physical systems". In: *Nature Communications 10* (2019-10-25). URL: https://www.nature.com/articles/s41467-019-12490-1 (visited on 2023-04-20).

[29] C. Feng, V.R. Palleti, A. Mathur, and D. Chana. "A Sysematic Framework to Generate Invariants for Anomaly Detection in Industrial Control Systems". In: NDSS Symposium 2019 (San Diego, CA, USA, Feb. 24–27, 2019). 2019. DOI: https://dx.doi.org/10.14722/ndss.2019.23265.

[30] Singapore University of Technology and Design. *iTrust - Center for Research in Cyber Security*. URL: https://itrust.sutd.edu.sg/ (visited on 2023-05-15).

[31] K. Pal, A. Adepu, and J. Goh. "Effectiveness of Association Rules Mining for Invariants Generation in Cyber-Physical Systems". In: 2017 IEEE 18th International Symposium on High Assurance Systems Engineering (HASE) (Jan. 2017). 2017. DOI: https://doi.org/10.1109/HASE.2017.21.

[32] *Association rules mining*. URL: https://en.wikipedia.org/wiki/Association_rule_learning (visited on 2023-04-21).

[33] K. Pal, A. Adepu, and J. Goh. "Cyber-Physical System Discovery: Reverse Engineering Physical Processes". In: 3rd ACM Workshop on Cyber-Physical System Security (Apr. 2017). 2017, pp. 3–14. DOI: https://doi.org/10.1145/3055186.3055195.

[34] B. Perelman. "Reconnaissance in Industrial Networks: What You Don't See Can Hurt You". In: *SecurityWeek* (2017-04-04). URL: https://www.securityweek.com/reconnaissance-industrial-networks-what-you-dont-see-can-hurt-you/ (visited on 2023-05-18).

[35] *Ceccato et al. reverse engineering prototype tool*. URL: https://github.com/SeUniVr/PLC-RE (visited on 2023-04-23).

[36] iTrust Singapore University of Technology and Design. *Secure Water Treatment*. URL: https://itrust.sutd.edu.sg/testbeds/secure-water-treatment-swat/ (visited on 2023-05-15).

[37] R. Lanotte, M. Merro, and A. Munteanu. "Industrial Control Systems Security via Runtime Enforcement". In: *ACM Transactions on Privacy and Security* 26.4 (2023-02), pp. 1–41. DOI: https://doi.org/10.1145/3546579.

[38] M. Oliani. *AttackPLC - Project for Network Security teaching*. URL: https://github.com/marcooliani/AttackPLC (visited on 2023-04-23).

[39] *Ray - Productionizing and scaling Python ML workloads simply*. URL: https://www.ray.io/ (visited on 2023-04-25).

[40] *Tshark*. URL: https://tshark.dev/ (visited on 2023-04-25).

[41] *Wireshark*. URL: https://www.wireshark.org/ (visited on 2023-04-25).

[42] *pandas - Python Data Analysis library*. URL: https://pandas.pydata.org/ (visited on 2023-04-25).

[43] *NumPy - The fundamental package for scientific computing with Python*. URL: https://numpy.org/ (visited on 2023-04-25).

[44] *R project for statistical computing*. URL: https://www.r-project.org/ (visited on 2023-04-25).

[45] *SciPy - Fundamental alghorithms for scientific computing with Python*. URL: https://scipy.org/ (visited on 2023-04-25).

[46] *The Daikon dynamic invariant detector*. URL: http://plse.cs.washington.edu/daikon/ (visited on 2023-04-25).

[47] *Enhancing Daikon output*. URL: https://plse.cs.washington.edu/daikon/download/doc/daikon/Enhancing-Daikon-output.html#Splitter-info-file-format (visited on 2023-04-25).

[48] *Process mining*. URL: https://en.wikipedia.org/wiki/Process_mining (visited on 2023-04-26).

[49] *Fluxicon Disco*. URL: https://fluxicon.com/disco/ (visited on 2023-04-26).

[50] *OpenPLC - Open source PLC software*. URL: https://openplcproject.com/ (visited on 2023-04-23).

[51] *Docker*. URL: https://www.docker.com/ (visited on 2023-04-26).

[52] MathWorks. *Simulink*. URL: https://it.mathworks.com/products/simulink.html (visited on 2023-04-26).

[53] T. Carlsson. "Industrial network market shares 2023". In: *HMS Networks* (2023-05-05). URL: https://www.hms-networks.com/news-and-insights/news-from-hms/2023/05/05/industrial-network-market-shares-2023 (visited on 2023-05-18).

[54] *ProM Tools - The Process Mining Framework*. URL: https://promtools.org/ (visited on 2023-04-26).

[55] Fraunhofer Institute for Applied Information Technology. *pm4py - Process Mining for Python*. URL: https://promtools.org/ (visited on 2023-04-26).

[56] statsmodels. *statsmodels - statistical models, hypothesis tests, and data exploration*. URL: https://www.statsmodels.org/stable/index.html (visited on 2023-05-05).

[57] NetworkX. *NetworkX - Network Analysis in Python*. URL: https://networkx.org/ (visited on 2023-05-05).

[58] Wikipedia. *Polynomial regression*. URL: https://en.wikipedia.org/wiki/Polynomial_regression (visited on 2023-05-21).

[59] Wikipedia. *Decomposition of time series*. URL: https://en.wikipedia.org/wiki/Decomposition_of_time_series (visited on 2023-05-21).

[60] Martin Fleischmann. "Line simplification algorithms". In: (2020-04-27). URL: https://martinfleischmann.net/line-simplification-algorithms/ (visited on 2023-05-21).

[61] Wikipedia. *Savitzky–Golay filter*. URL: https://en.wikipedia.org/wiki/Savitzky%E2%80%93Golay_filter (visited on 2023-05-06).

[62] Otext. *STL decomposition*. URL: https://otexts.com/fpp2/stl.html (visited on 2023-05-06).

[63] Wikipedia. *Ramer-Douglas-Peucker algorithm*. URL: https://en.wikipedia.org/wiki/Ramer%E2%80%93Douglas%E2%80%93Peucker_algorithm (visited on 2023-05-21).

[64] *The Daikon Invariant Detector User Manual*. URL: https://plse.cs.washington.edu/daikon/download/doc/daikon.html#Program-point-sections (visited on 2023-05-25).

[65] Wikipedia. *Transitive closures*. URL: https://en.wikipedia.org/wiki/Transitive_closure (visited on 2023-05-11).

[66] iTrust Singapore University of Technology and Design. *Secure Water Treatment (SWaT) Testbed*. URL: https://itrust.sutd.edu.sg/wp-content/uploads/2021/07/SWaT_technical_details-160720-v4.4.pdf (visited on 2023-05-15).

[67] A. Mathur and N. O. Tippenhauer. "SWaT: a water treatment testbed for research and training on ICS security". In: 2016 International Workshop on Cyber-physical Systems for Smart Water Networks (CySWater) (Apr. 2016). 2016. DOI: http://dx.doi.org/10.1109/CySWater.2016.7469060.

[68] iTrust Singapore University of Technology and Design. *Datasets*. URL: https://itrust.sutd.edu.sg/itrust-labs_datasets/ (visited on 2023-05-15).

[69] S. Adepu, J. Goh, K.N. Junejo, and A. Mathur. "A Dataset to Support Research in the Design of Secure Water Treatment Systems". In: The 11th International Conference on Critical Information Infrastructures Security (France, Oct. 2016). 2016.

[70] T. A. Snide. *CIP-Modbus Integration*. 2008. URL: https://www.modbus.org/docs/CIP%20Modbus%20Integration%20Hanover%20Fair_0408.pdf (visited on 2023-05-17).