

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

1	Introduction	1
1.1	Contribution	1
1.2	Outline	2
2	Background on Industrial Control Systems	3
2.1	Industrial Control Systems Architecture	5
2.2	Operational Technology Networks	7
2.2.1	Field I/O Devices Layer	7
2.2.2	Controller Network Layer	8
2.2.2.1	Programmable Logic Controllers	8
2.2.2.2	Remote Terminal Units	12
2.2.3	Area Control Layer	13
2.2.3.1	Supervisory Control And Data Acquisition	13
2.2.3.2	Human-Machine Interface	13
2.2.4	Operations/Control Layer	14
2.2.5	Demilitarized Zone	14
2.2.6	Industrial Protocols	15
2.2.6.1	Modbus	16
2.2.6.2	EtherNet/IP	19
2.2.6.3	Common Industrial Protocol (CIP)	21
3	State of the Art	23
3.1	Literature on Process Comprehension	24

3.2	Ceccato et al.'s black-box dynamic analysis for water-tank systems	26
3.2.1	Testbed	27
3.2.2	Scanning of the System and Data Pre-processing	31
3.2.3	Graphs and Statistical Analysis	33
3.2.4	Invariant Inference and Analysis	34
3.2.5	Business Process Mining and Analysis	36
3.2.6	Application	38
3.2.7	Limitations	44
4	A Substantial Improvement to Ceccato et al.'s Framework	55
4.1	The Proposed New Framework	56
4.1.1	Framework Structure	58
4.1.2	Python Libraries and External Tools	60
4.2	Analysis Phases	61
4.2.1	A Little Testbed: Stage 1 of iTrust SWaT System	61
4.2.2	Phase 0: Network Analysis	62
4.2.2.1	Extracting Data from PCAP Files	62
4.2.2.2	Network Information	64
4.2.3	Phase 1: Data Pre-processing	68
4.2.3.1	Subsystem Selection	69
4.2.3.2	Dataset Enrichment	71
4.2.3.3	Datasets Merging	78
4.2.3.4	Preliminary Analysis of the Obtained Subsystem	80
4.2.4	Phase 2: Graphs and Statistical Analysis	83
4.2.5	Phase 3: Invariant Inference and Analysis	89
4.2.5.1	Revised Daikon Output	89
4.2.5.2	Types of Analysis	93
4.2.6	Phase 4: Business Process Analysis	99
4.2.6.1	Process Mining of the Physical Process	100
4.2.6.2	Network Communications	105

5 Case study: the iTrust SWaT System	107
5.1 Architecture	107
5.1.1 Physical Process	108
5.1.2 Control and Communication Network	110
5.2 Datasets	112
5.2.1 Our Case Study: the 2015 Dataset	113
6 Our Framework at Work on the iTrust SWaT System	117
6.1 Preliminary Operations	118
6.2 Planning the Analysis Strategy	118
6.3 Reverse Engineering of the iTrust SWaT System	120
6.3.1 Reverse Engineering of PLC1 and PLC2	122
6.3.1.1 Pre-processing - Preliminary Analysis . . .	122
6.3.1.2 Graphs and Statistical Analysis	126
6.3.1.3 Invariant Inference and Analysis	129
6.3.1.4 Business Process Analysis	129
6.3.2 Reverse Engineering of PLC2 and PLC3	129
6.3.2.1 Pre-processing	129
6.3.2.2 Graphs and Statistical Analysis	129
6.3.2.3 Invariant Inference and Analysis	129
6.3.2.4 Business Process Analysis	129
6.3.3 Reverse Engineering of PLC3 and PLC4	129
6.3.3.1 Pre-processing	129
6.3.3.2 Graphs and Statistical Analysis	129
6.3.3.3 Invariant Inference and Analysis	129
6.3.3.4 Business Process Analysis	129
7 Conclusions	131
7.1 Discussions	131
7.2 Future work	131
List of Figures	133
List of Tables	135

Listings	137
-----------------	------------

References	139
-------------------	------------

Introduction

¹ **L**OREM ipsum dolor bla bla bla. Ma dove metto l'abstract? Prova di
² interlinea che direi posso anche andare bene, ma bisogna poi vedere
³ il tutto come si incasca alla fine, in modo da ottenere un bel risultato alla
⁴ vista.

⁵ 1.1 Contribution

⁶ The primary objective of this thesis was originally to validate a specific
⁷ methodology designed to attain process comprehension in an Industrial
⁸ Control System (ICS) when applied to a real-world scenario. This method-
⁹ ology, which adopts a blackbox approach, involves dynamic analysis of
¹⁰ the physical process and network communications of the system. The ul-
¹¹ timate goal of this methodology is to accurately derive a comprehensive
¹² understanding of the industrial process.

¹³ To accomplish this objective, a framework developed by the authors of
¹⁴ the methodology is utilized. This framework incorporates a series of anal-
¹⁵ ysis steps that are employed to facilitate the application of the methodol-
¹⁶ ogy. The entire framework and methodology are then applied to a spe-
¹⁷ cially implemented virtualized testbed, providing a controlled environ-
¹⁸ ment for testing and evaluation purposes and to facilitate the process of
¹⁹ achieving process comprehension of an Industrial Control System.

²⁰ As the thesis progressed, it quickly became evident that both the frame-
²¹ work and the methodology employed had certain limitations. In response,
²² an extensive restructuring of the framework was conducted, aimed at en-
²³ hancing and expanding its existing features while introducing new ones.

²⁴ Hence, the main contribution of the thesis is now to enhance the origi-
²⁵ nal methodology by refining the framework and reassessing the approach
²⁶ for each analysis step. The aim is to achieve a more complete and exhaus-
²⁷ tive process comprehension. Subsequently, the improved methodology
²⁸ will be tested on a different and more complex case study, distinct from the
²⁹ virtualized testbed, to validate its effectiveness in a real-world scenario.

³⁰ 1.2 Outline

³¹ The thesis is structured as follows:

³² **Chapter 2:** provides a background on Industrial Control Systems, (ICSS)
³³ describing their structure, components, and some of the network
³⁴ communication protocols used;

³⁵ **Chapter 3:** following an introductory section that provides a brief overview
³⁶ of the existing literature on process comprehension in industrial con-
³⁷ trol systems, this chapter focuses on a specific paper that outlines
³⁸ a methodology to attaining process comprehension of an industrial
³⁹ system by employing dynamic blackbox analysis;

⁴⁰ **Chapter 4:** outlines a proposal to improve and extend the methodology
⁴¹ outlined in the previous chapter;

⁴² **Chapter 5:** presents the case study on which the proposed methodology
⁴³ will be applied;

⁴⁴ **Chapter 6:** shows how the proposed methodology is applied to the case
⁴⁵ study illustrated above;

⁴⁶ **Chapter 7:** outlines final conclusions and future work.

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

⁶⁷ purpose was to sabotage the nuclear centrifuges of the enrichment plant
⁶⁸ at the Natanz nuclear facility in Iran; (ii) **Industroyer** [3], also referred
⁶⁹ as *Crashoverride*, responsible for the attack on the Ukrainian power grid
⁷⁰ on December 17, 2016; (iii) the attack on February, 2021 to a water treat-
⁷¹ ment plant in Oldsmar, Florida [4], where the level of sodium hydroxide
⁷² was intentionally increased to a level approximately 100 times higher than
⁷³ normal.

⁷⁴

⁷⁵ The increasing connectivity of ICSs and the associated security risks have
⁷⁶ led to a growing interest in the field of ICS security. Researchers and prac-
⁷⁷ titioners are working to develop new security technologies, standards, and
⁷⁸ best practices to protect ICSs from cyber attacks. This includes efforts to
⁷⁹ improve the security of ICS networks and devices, as well as the develop-
⁸⁰ ment of new monitoring and detection techniques to identify and respond
⁸¹ to cyber attacks.

⁸²

⁸³ 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 Technology (IT) and Industrial Control Systems (ICSs)

84 2.1 Industrial Control Systems Architecture

85 In the past, there has been a clear division between *Information Technology* (IT)
86 and *Operational Technology* (OT), both at the technical and organiza-
87 tional levels. Each domain has maintained its own distinct technology
88 stacks, protocols, and standards. However, with the emergence of Indus-
89 try 4.0 and the rapid expansion of industrial automation, which heavily
90 relies on IT tools for monitoring and controlling critical infrastructures,
91 the boundary between IT and OT has started to blur. This trend has paved
92 the way for greater integration between these two domains, thus improv-
93 ing productivity and process quality.

94

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

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

2.1 Industrial Control Systems Architecture

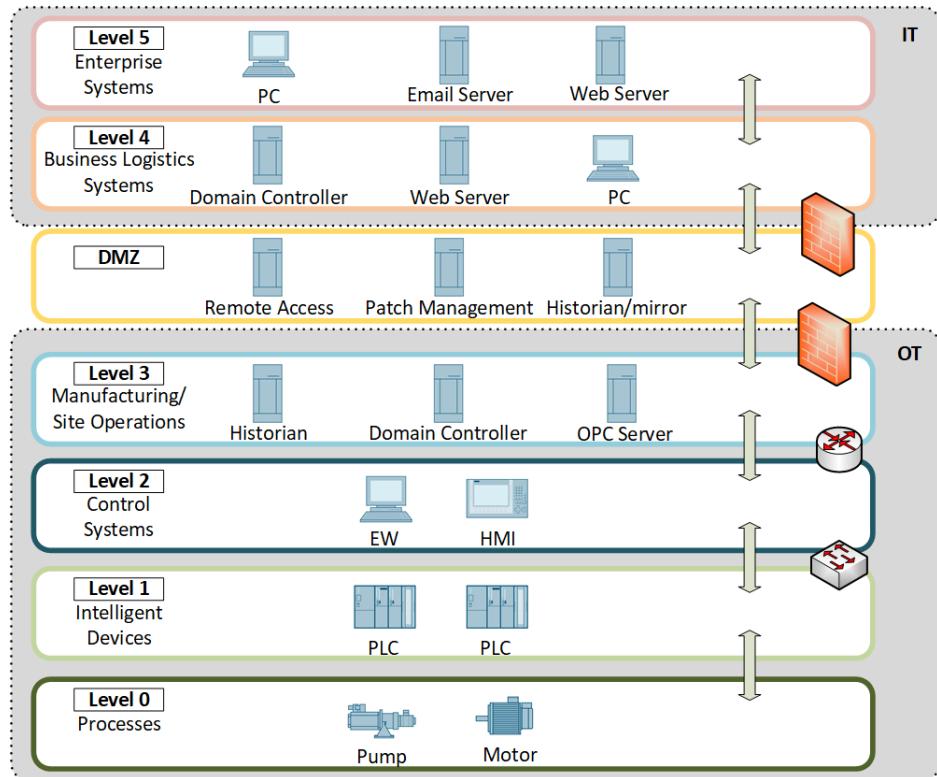


Figure 2.1: ICS architecture schema

- **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 Manufacturing/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 pur-

pose services. At Enterprise 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.

¹⁵⁵ 2.2.2 Controller Network Layer

¹⁵⁶ *Controller Network* layer includes devices that handle data from and to
¹⁵⁷ the *Field I/O Devices* layer. This kind of device is capable of gathering data
¹⁵⁸ from sensors, updating its internal state, and activating actuators (for ex-
¹⁵⁹ ample opening or close a pump that controls the level of a tank), making
¹⁶⁰ decisions based on a customized program, known as its control logic.
¹⁶¹ Commonly found within this layer are *Programmable Logic Controllers* (PLCs)
¹⁶² and *Remote Terminal Units* (RTUs): in the upcoming sections, we will ex-
¹⁶³ amine these elements in detail.

¹⁶⁴ 2.2.2.1 Programmable Logic Controllers

¹⁶⁵ A *Programmable Logic Controller* (PLC) is a **small and specialized in-**
¹⁶⁶ **dustrial computer** having the capability of controlling complex industrial
¹⁶⁷ and manufacturing processes [7].

¹⁶⁸ Compared to relay systems and personal computers, PLCs are opti-
¹⁶⁹ mized for control tasks and industrial environments: they are rugged and
¹⁷⁰ designed to withdraw harsh conditions such as dust, vibrations, humid-
¹⁷¹ ity and temperature: they have more reliability than personal computers,
¹⁷² which are more prone to crash, and they are more compact a require less
¹⁷³ maintenance than a relay system.

¹⁷⁴ Furthermore, I/O interfaces are already on the controller, so PLCs are eas-
¹⁷⁵ ier to expand with additional I/O modules (if in a rack format) to manage
¹⁷⁶ more inputs and ouputs, without reconfiguring hardware as in relay sys-
¹⁷⁷ tems when a reconfiguration occurs.

¹⁷⁸ PLCs are more *user-friendly*: they are not intended (only) for computer
¹⁷⁹ programmers, but designed for engineers with a limited knowledge in
¹⁸⁰ programming languages: control program can be entered with a simple
¹⁸¹ and intuitive language based on logic and switching operations instead of
¹⁸² a general-purpose programming language (*i.e.* C, C++, ...).

- 183 **PLC Architecture** The basic hardware architecture of a PLC consists of
184 these elements [8]:

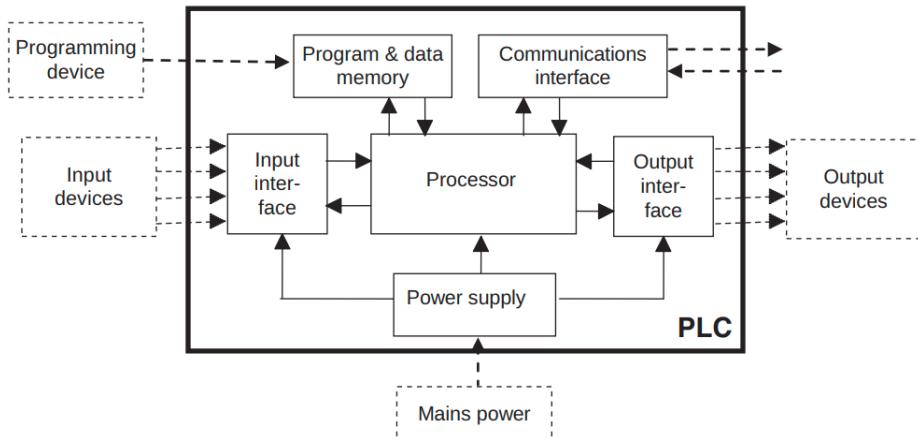


Figure 2.2: PLC architecture

- 185 • **Processor unit (CPU):** contains the microprocessor. This unit inter-
186 pretes the input signals from I/O modules, executes the control pro-
187 gram stored in the Memory Unit and sends the output signals to the
188 I/O Modules. The processor unit also sends data to the Communi-
189 cation interface, for the communication with additional devices.
- 190 • **Power supply unit:** converts AC voltage to low DC voltage.
- 191 • **Programming device:** is used to store the required program into the
192 memory unit.
- 193 • **Memory Unit:** consists in RAM memory and ROM memory. RAM
194 memory is used for storing data from inputs, ROM memory for stor-
195 ing operating system, firmware and user program to be executed by
196 the CPU.
- 197 • **I/O modules:** provide interface between sensors and final control
198 elements (actuators).
- 199 • **Communications interface:** used to send and receive data on a net-
200 work from/to other PLCs.

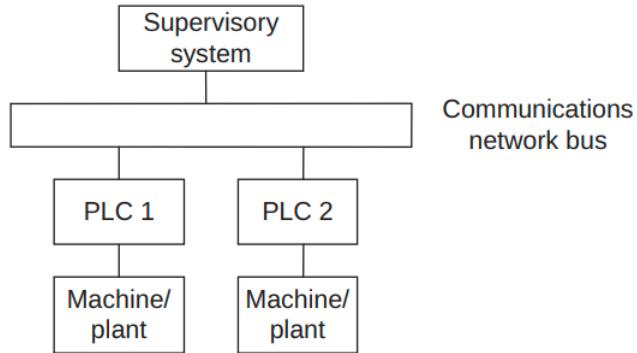


Figure 2.3: PLC communication schema

²⁰¹ **PLC Programming** Two different programs are executed in a PLC: the
²⁰² **operating system** and the **user program**.

²⁰³ The operating system tasks include executing the user program, man-
²⁰⁴ aging memory areas and the *process image table* (memory registers where
²⁰⁵ inputs from sensors and outputs for actuators are stored).

²⁰⁶ The user program needs to be uploaded on the PLC via the program-
²⁰⁷ ming device and runs on the process image table in *scan cycles*: each scan
²⁰⁸ is made up of three phases [9]:

- ²⁰⁹ 1. reading inputs from the process images table
- ²¹⁰ 2. execution of the control code and computing the physical process evolution
- ²¹² 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

²¹⁵ Standard PLCs **programming languages** are basically of two types:
²¹⁶ **textuals** and **graphicals**. Textual languages include languages such as
²¹⁷ *Instruction List* (IL) and *Structured Text* (ST), while *Ladder Diagrams* (LD),
²¹⁸ *Function Block Diagram* (FBD) and *Sequential Function Chart* (SFC) belong
²¹⁹ to the graphical languages.

220 Graphical languages are more simple and immediate comparing to the
 221 textual ones and are preferred by programmers because of their features
 222 and simplicity, in particular the **Ladder Logic programming** (see Figure
 223 2.4 for a comparison).

```

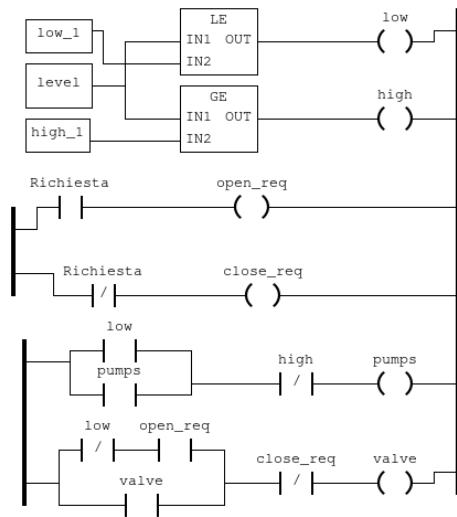
PROGRAM PLC1
VAR
  level AT %IW0 : INT;
  Richiesta AT %QX0..2 : BOOL;
  request AT %IW1 : INT;
  pumps AT %QX0..0 : BOOL;
  valve AT %QX0..1 : BOOL;
  low AT %MW0..0 : BOOL;
  high AT %MW0..1 : BOOL;
  open_req AT %MW0..3 : BOOL;
  close_req AT %MW0..4 : BOOL;
  low_1 AT %MW0 : INT := 40;
  high_1 AT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

224 **PLC Security** PLCs were originally designed to operate as closed sys-
 225 tems, not connected and exposed to the outside world via communication
 226 networks: the question of the safety of these systems, therefore, was not
 227 a primary aspect. The advent of Internet has brought undoubted advan-
 228 tages, but has introduced problems relating to the safety and protection of
 229 PLCs from external attacks and vulnerabilities.

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

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

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

240 In addition to this, keeping the process network and Internet sepa-
241 rated, limiting the use of USB devices among users to reduce the risks of
242 infections, and using strong account management and maintenance poli-
243 cies are best practices to prevent attacks and threats and to avoid potential
244 damages.

245 **2.2.2.2 Remote Terminal Units**

246 *Remote Terminal Units* (RTUs) are computers with radio interfacing sim-
247 ilar to PLCs: they transmit telemetry data to the control center or to the
248 PLCs and use messages from the master supervisory system to control
249 connected objects [11].

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

257 Industries that require RTUs often operate in areas without reliable ac-
258 cess to the power grid or require monitoring and control substations in re-
259 mote locations. These include telecommunications, railways, and utilities
260 that manage critical infrastructure such as power grids, pipelines, and wa-
261 ter treatment facilities. The advanced technology of RTUs allows these in-
262 dustries to maintain essential services, even in challenging environments
263 or under adverse weather conditions.

²⁶⁴ 2.2.3 Area Control Layer

²⁶⁵ The Area Control layer encompasses hardware and software systems
²⁶⁶ useful for supervising, monitoring and controlling the physical process,
²⁶⁷ driving the behavior of the entire infrastructure. The layer includes sys-
²⁶⁸ tems such as *Supervisory Control and Data Acquisition* (SCADA), *Distributed*
²⁶⁹ *Control Systems* (DCSs), that perform SCADA functions but are usually de-
²⁷⁰ ployed locally, and engineer workstations.

²⁷¹ 2.2.3.1 Supervisory Control And Data Acquisition

²⁷² *Supervisory Control And Data Acquisition (SCADA)* is a system of soft-
²⁷³ ware and hardware elements that allows industrial organizations to [12]:

- ²⁷⁴ • Control industrial processes locally or at remote locations;
- ²⁷⁵ • Monitor, gather, and process real-time data;
- ²⁷⁶ • Directly interact with devices such as sensors, valves, pumps, mo-
²⁷⁷ tors, and more through human-machine interface (HMI) software;
- ²⁷⁸ • Record and aggregate events to send to historian server.

²⁷⁹ The SCADA software processes, distributes, and displays the data, help-
²⁸⁰ ing operators and other employees analyze the data and make important
²⁸¹ decisions.

²⁸² 2.2.3.2 Human-Machine Interface

²⁸³ The *Human-Machine Interface* (HMI) is the hardware and software in-
²⁸⁴ terface that operators use to monitor the processes and interact with the
²⁸⁵ ICS. A HMI shows the operator and authorized users information about
²⁸⁶ system status and history; it also allows them to configure parameters on
²⁸⁷ the ICS such as set points and, send commands and make control deci-
²⁸⁸ sions [13].

²⁸⁹ The HMI can be in the form of a physical panel, with buttons and indicator
²⁹⁰ lights, or PC software as shown in Figure 2.5.

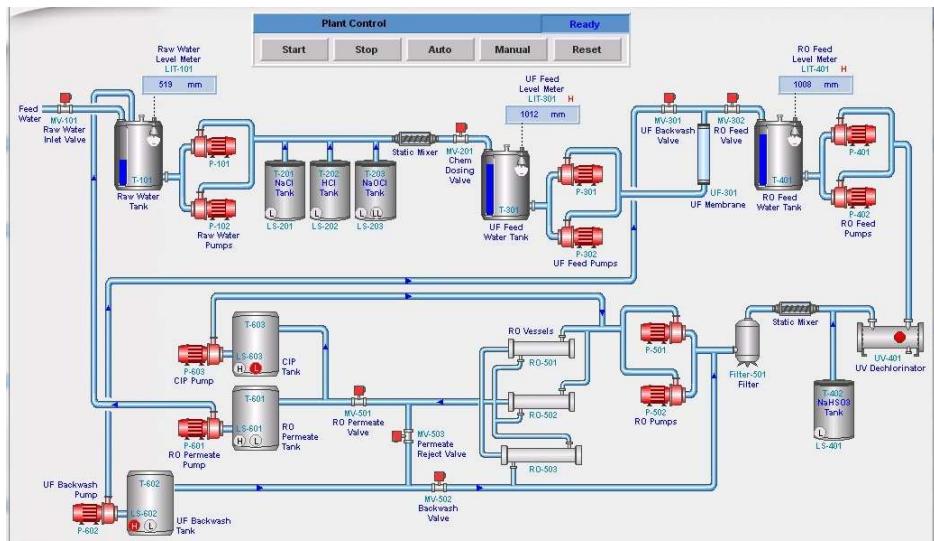


Figure 2.5: Example of HMI for a water treatment plant

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

306 of automation, the need for bidirectional data flows between OT and IT
307 systems has increased. The convergence of IT and OT in this layer can offer
308 organizations a competitive edge. However, it's important to note that
309 adopting a flat network approach in this context can potentially heighten
310 cyber risks for the organization.

311 2.2.6 Industrial Protocols

312 *Industrial Protocols* are the networks that are used to connect the dif-
313 ferent components of the ICS and allow them to communicate with each
314 other. Industrial Protocols can include wired and wireless networks, such
315 as Ethernet/IP, Modbus, DNP3, Profinet and others.

316 As mentioned at the beginning of this Chapter, industrial systems differ
317 from classical IT systems in the purpose for which they are designed: con-
318 trolling physical processes the former, processing and storing data the lat-
319 ter. For this reason, ICSs require different communication protocols than
320 traditional IT systems for real time communications and data transfer.

321 A wide variety of industrial protocols exists: this is because originally
322 each vendor developed and used its own proprietary protocol. How-
323 ever, these protocols were often incompatible with each other, resulting
324 in devices from different vendors being unable to communicate with each
325 other.

326 To solve this problem, standards were defined with a view to allowing
327 these otherwise incompatible device to intercommunicates.

328 Among all the various protocols, some have risen to prominence as widely
329 accepted standards. These *de facto* protocols are commonly utilized in in-
330 dustrial systems due to their proven reliability and effectiveness. In the
331 following sections, we will provide a brief overview of some of the most
332 prevalent and widely used protocols in the industry.

333 2.2.6.1 Modbus

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

339 Modbus is the most widely used communication protocol among in-
 340 dustrial systems because it has several advantages:

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

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

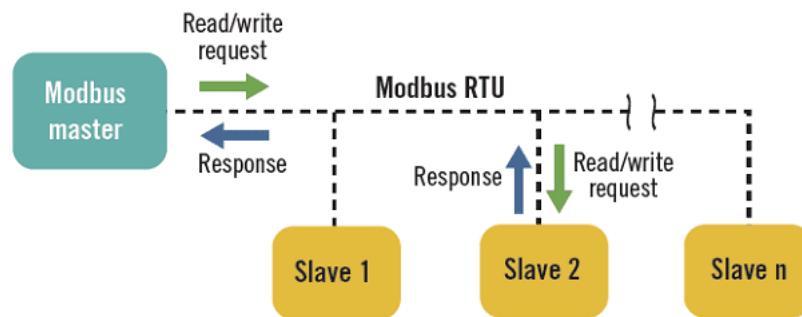


Figure 2.6: Modbus Request/Response schema

349 In this kind of architecture, a single device (master) can send requests
 350 to other devices (slaves), either individually or in broadcast: these slave
 351 devices (usually peripherals such as actuators) will respond to the master

352 by providing data or performing the action requested by the master using
 353 the Modbus protocol. Slave devices cannot generate requests to the master
 354 [16].

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

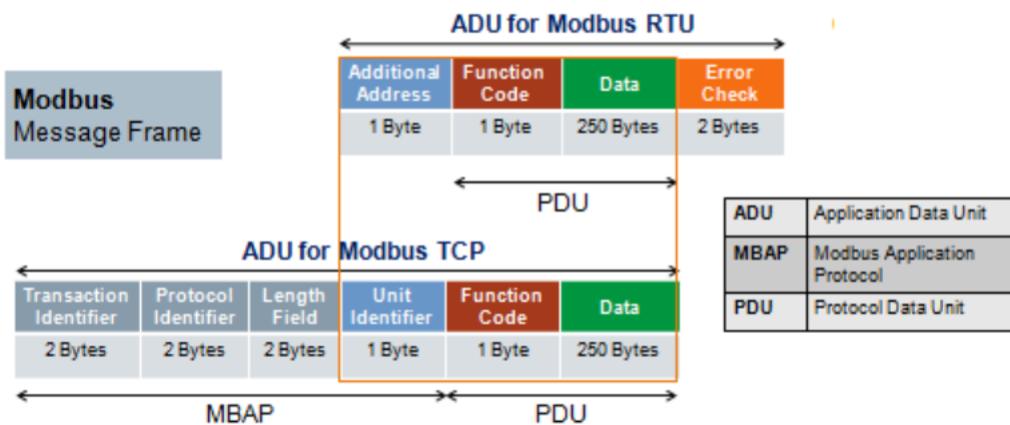


Figure 2.7: Modbus RTU frame and Modbus TCP frame

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

- 362 • *Coil*: binary type, read/write accessible by both masters and slaves
- 363 • *Discrete Input*: binary type, accessible in read-only mode by masters
 364 and in read/write mode by slaves
- 365 • *Analog Input*: 16 bits in size (word), are accessible in read-only mode
 366 by masters and in read/write mode by slaves
- 367 • *Holding Register*: 16 bits in size (word), accessible in read/write mode
 368 by both masters and slaves. Holding Registers are the most com-
 369 monly used registers for output and as general memory registers.

³⁷⁰ **Modbus Function Codes** *Modbus Function Codes* are specific codes used
³⁷¹ by the Modbus master within a request frame (see Figure 2.7) to tell the
³⁷² Modbus slave device which register type to access and which action to
³⁷³ perform on it.

³⁷⁴ Two types of Function Codes exists: for data access and for diagnostic
³⁷⁵ 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

³⁷⁶ **Modbus Security Issues** Despite its simplicity and widespread use, the
³⁷⁷ Modbus protocol does not have any security feature, which exposes it to
³⁷⁸ vulnerabilities and attacks.

³⁷⁹ Data in Modbus are transmitted unencrypted (*lack of confidentiality*),
³⁸⁰ with no data integrity controls (*lack of integrity*) and authentication checks
³⁸¹ (*lack of authentication*), in addition to the *lack of session*. Hence, the protocol
³⁸² is vulnerable to a variety of attacks, such as Denial of Services (DoS), buffer
³⁸³ overflows and reconnaissance activities.

³⁸⁴ The easiest attack to bring to the Modbus protocol, however, is **packet**
³⁸⁵ **sniffing**: since, as mentioned earlier, network traffic is unencrypted and
³⁸⁶ the data transmitted is in cleartext, it is sufficient to use a packet sniffer to
³⁸⁷ capture the network traffic, read the packets and thus gather informations

388 about the system such as ip addresses, function codes of requests and to
 389 modify the operation of the devices.

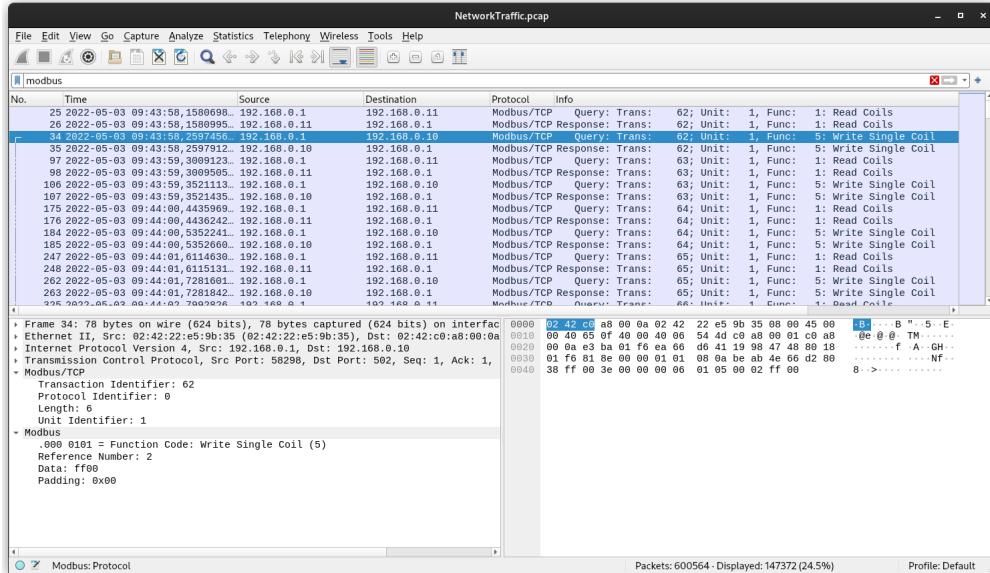


Figure 2.8: Example of packet sniffing on the Modbus protocol

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

396 2.2.6.2 EtherNet/IP

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

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

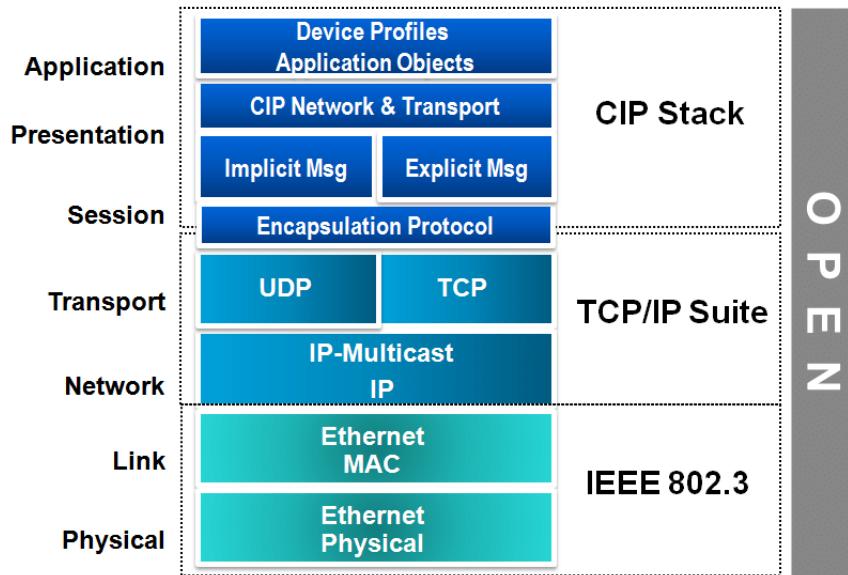


Figure 2.9: OSI model for EtherNet/IP stack

⁴⁰⁴ and other industrial protocols for data access and exchange such as *Open*
⁴⁰⁵ *Platform Communication* (OPC).

⁴⁰⁶ **Physical and Data Link layer** The use of the IEEE 802.3 standard allows
⁴⁰⁷ EtherNet/IP to flexibly adopt different network topologies (star, linear,
⁴⁰⁸ ring, etc.) over different connections (copper, fibre optic, wireless, etc.), as
⁴⁰⁹ well as the possibility to choose the speed of network devices.
⁴¹⁰ IEEE 802.3 in addition defines at Data Link layer the *Carrier Sense Multiple*
⁴¹¹ *Access - Collision Detection* (CSMA/CD) protocol, which controls access to
⁴¹² the communication channel and prevents collisions.

⁴¹³ **Transport layer** At the transport level, EtherNet/IP encapsulates mes-
⁴¹⁴ sages from the CIP stack into an Ethernet message, so that messages can
⁴¹⁵ be transmitted from one node to another on the network using the TCP/IP
⁴¹⁶ protocol. EtherNet/IP uses two forms of messaging, as defined by CIP
⁴¹⁷ standard [18][20]:

- ⁴¹⁸ • **unconnected messaging:** used during the connection establishment
⁴¹⁹ phase and for infrequent, low priority, explicit messages. Uncon-

420 nected messaging uses TCP/IP to transmit messages across the net-
421 work asking for connection resource each time from the *Unconnected*
422 *Message Manager* (UCMM).

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

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

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

437 **Session, Presentation and Application layer** At the upper layers, Ether-
438 Net/IP implements the CIP protocol stack. We will discuss this protocol
439 more in detail in Section 2.2.6.3.

440 **2.2.6.3 Common Industrial Protocol (CIP)**

441 The *Common Industrial Protocol* (CIP) is an open industrial automation
442 protocol supported by ODVA. It is a **media independent** (or *transport in-*
443 *dependent*) protocol using a *producer-consumer* communication model and
444 providing a **unified architecture** throughout the manufacturing enterprise
445 [21][22].

446 CIP has been adapted in different types of network:

- 447 • **EtherNet/IP**, adaptation to *Transmission Control Protocol* (TCP) tech-
448 nologies
- 449 • **ControlNet**, adaptation to *Concurrent Time Domain Multiple Access*
450 (CTDMA) technologies
- 451 • **DeviceNet**, adaptation to *Controller Area Network* (CAN) technolo-
452 gies
- 453 • **CompoNet**, adaptation to *Time Division Multiple Access* (TDMA) tech-
454 nologies

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

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

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

State of the Art

471 IN COVENTIONAL IT SYSTEMS, the objective of an attacker is to comprehend
472 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).

478 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.

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

485 There is limited literature available concerning the gathering and analysis
486 of information related to the comprehension and operation of an Industrial
487 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
488 will specifically focus on one of the presented papers.

490 3.1 Literature on Process Comprehension

491 **Keliris and Maniatikos** The first approach presented in this section is by
492 Keliris and Maniatikos [26]: they present a methodology for au-
493 tomating the reverse engineering of ICS binaries based on a *modular*
494 *framework* (called ICSREF) that can reverse binaries compiled with
495 CODESYS [27], one of the most popular and widely used PLC com-
496 pilers, irrespective of the language used.

497 **Yuan et al.** Yuan et al. [28] propose a *data-driven* approach to discover-
498 ing cyber-physical systems process behavior from data directly: to
499 achieve this goal, they have implemented a framework whose pur-
500 pose is to identify physical systems and transition logic inference,
501 and to seek to understand the mechanisms underlying these pro-
502 cesses, making furthermore predictions concerning their state trajec-
503 tories based on the discovered models.

504 **Feng et al.** Feng et al. [29] developed a framework that can generate sys-
505 tem *invariant rules* based on machine learning and data mining tech-
506 niques from ICS operational data log. These invariants are then se-
507 lected by systems engineers to derive IDS systems from them.

508 The experiment results on two different testbeds, the *Water Distri-*
509 *bution system* (WaDi) and the *Secure Water Treatment system* (SWaT),
510 both located at the iTrust - Center for Research in Cyber Security at
511 the University of Singapore of Technology and Design [30], show
512 that under the same false positive rate invariant-based IDSs have a
513 higher efficiency in detecting anomalies than IDS systems based on
514 a residual error-based model.

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

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

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

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

540 The paper shows *step-by-step* how to perform a **ICS reconnaissance**, a
541 phase specifically designed to gather extensive intelligence on mul-
542 tiple fronts, including human factors, network and protocol infor-
543 mation, details about the manufacturing process, industrial applica-
544 tions, and potential vulnerabilities. The primary goal is to accumu-
545 late a wealth of information to enhance understanding and aware-
546 ness in these areas [34]).

547 Reconnaissance phase is fundamental to process comprehension and
548 thus to the execution of MitM attacks.

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

26 3.2 Ceccato et al.'s black-box dynamic analysis for water-tank systems

552 posed PLCs and the Modbus protocol network scans an approximate
553 model of the physical process. This model is obtained by inferring
554 statistical properties, business process and system invariants from
555 data logs.

556 The proposed methodology was tested on a non-trivial case study,
557 using a virtualized testbed inspired by an industrial water treatment
558 plant.

559 In the next section we will examine this latest work in more detail,
560 which will be the basis for my work and thus the subsequent chap-
561 ters of this thesis.

562 3.2 Ceccato et al.'s black-box dynamic analysis 563 for water-tank systems

564 As previously mentioned, the paper introduces a methodology that re-
565 lies on black box dynamic analysis of an Industrial Control System (ICS)
566 and more particularly of its OT network. This methodology involves iden-
567 tifying potential Programmable Logic Controllers (PLCs) within the net-
568 work and scanning the memory registers of these identified controllers.
569 The purpose of this process is to obtain an approximate model of the con-
570 trolled physical process.

571 The primary goal of this black box analysis is to establish a correlation
572 between the different memory registers of the targeted PLCs and funda-
573 mental concepts of an OT network such as sensor values (i.e., measure-
574 ments), actuator commands, setpoints (i.e., range of values of a physical
575 variable), network communications, among others.

576 To accomplish this, the various types of memory registers are analyzed,
577 and attempts are made to determine the nature of the data they might
578 contain.

579 The second goal is to establish a relationship between the dynamic evolu-
580 tion of these fundamental concepts.

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

- 584 1. **scanning of the system and data pre-processing:** this phase involves
585 gathering data to generate data logs for the registers of PLCs and for
586 Modbus network communications.
- 587 2. **graphs and statistical analysis:** The collected data is utilized to pro-
588 vide insights into the memory registers associated with the Modbus
589 protocol by leveraging graphs and statistical data. This analysis ap-
590 proach offers valuable information about the characteristics and pat-
591 terns of the memory registers.
- 592 3. **invariants inference and analysis:** generates system invariants, which
593 are used to identify specific patterns and regularities within the sys-
594 tem. Additionally, this phase provides users with the capability to
595 view invariants related to a particular sensor or actuator.
- 596 4. **business process mining and analysis:** Using event logs, this phase
597 involves reconstructing the business process that depicts how a pro-
598 cess is executed. This step enables a thorough understanding of the
599 sequence of events that occur in the system and how they are in-
600 terrelated, ultimately leading to a comprehensive overview of the
601 business process.

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

607 3.2.1 Testbed

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

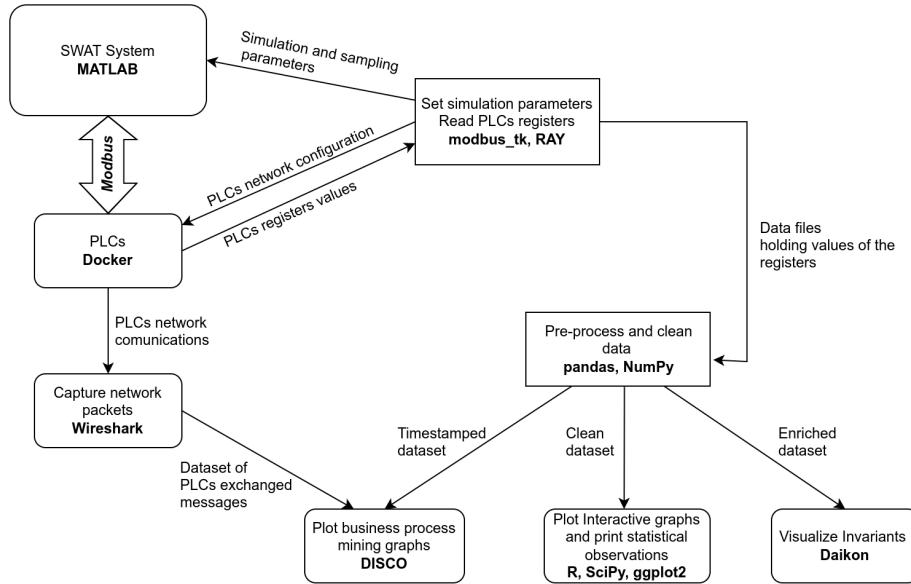


Figure 3.1: Workflow of Ceccato et al.'s stages and operations with used tools

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

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

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

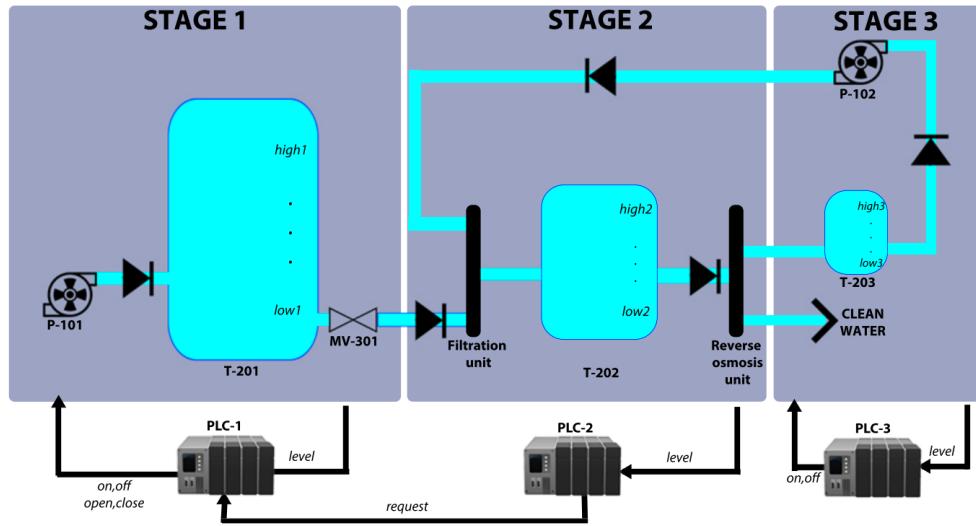


Figure 3.2: The simplified SWaT system used for running Ceccato et al. methodology

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

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

637 **PLC1** PLC1 monitors the level of tank T-201 and distinguishes three dif-
 638 ferent cases based on the level readings:

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

30 3.2 Ceccato et al.'s black-box dynamic analysis for water-tank systems

- 643 2. if the level of T-201 reaches the *high setpoint high1* (which is also
644 hardcoded in a specific memory register), then the pump **P-101**
645 **is closed**;
646 3. in cases where the level of T-201 is between the *low setpoint low1*
647 and the *high setpoint high1*, PLC1 waits for a request from PLC2
648 to open or close the valve MV-301. If a request to open the valve
649 MV-301 is received, water will flow from T-201 to T-202. How-
650 ever, if no request is received, the valve remains closed. In both
651 situations, the pump P-101 remains closed.

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

- 654 1. when the water level in tank T-202 reaches *low setpoint low2* (also
655 hardcoded in the memory registers), PLC2 sends a request to
656 PLC1 through a Modbus channel to **open valve MV-301**. This
657 request is made in order to allow the water to flow from tank T-
658 201 to tank T-202. The transmission channel between the PLCs
659 is established by copying a boolean value from a memory reg-
660 ister of PLC2 to a corresponding register of PLC1.
661 2. when the water level in tank T-202 reaches the *high setpoint high2*
662 value (also hardcoded in the memory registers), PLC2 sends a
663 **close request to PLC1 for valve MV-301**. This request prompts
664 PLC1 to close the valve, stopping the flow of water from tank
665 T-201 to tank T-202.
666 3. In cases where the water level in tank T-202 is between the low
667 and high setpoints, the valve MV-301 remains in its current state
668 (open or closed) while the tank is either filling or emptying.

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

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

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

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

678

3.2.2 Scanning of the System and Data Pre-processing

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

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

```
698       "127.0.0.1/8502/2022-05-03 12_10_00.591": {  
699         "DiscreteInputRegisters": {"%IX0.0": "0"},  
700         "InputRegisters": {"%IW0": "53"},  
701         "HoldingOutputRegisters": {"%QW0": "0"},  
702         "MemoryRegisters": {"%MW0": "40", "%MW1": "80"},
```

32 3.2 Ceccato et al.'s black-box dynamic analysis for water-tank systems

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

Listing 3.1: Example of registers capture

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

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

```
712     Time,Source,Destination,Protocol,Length,Function Code,  
713     ↳ Destination Port,Source Port,Data,Frame length on the  
714     ↳ wire,Bit Value,Request Frame,Reference Number,Info  
715     2022-05-03 11:43:58.158,IP_PLC1,IP_PLC2,Modbus/TCP,76,Read  
716     ↳ Coils,46106,502,,76,TRUE,25,,,"Response: Trans: 62;  
717     ↳ Unit: 1, Func: 1: Read Coils"
```

Listing 3.2: Example of raw network capture

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

729

¹Not 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).

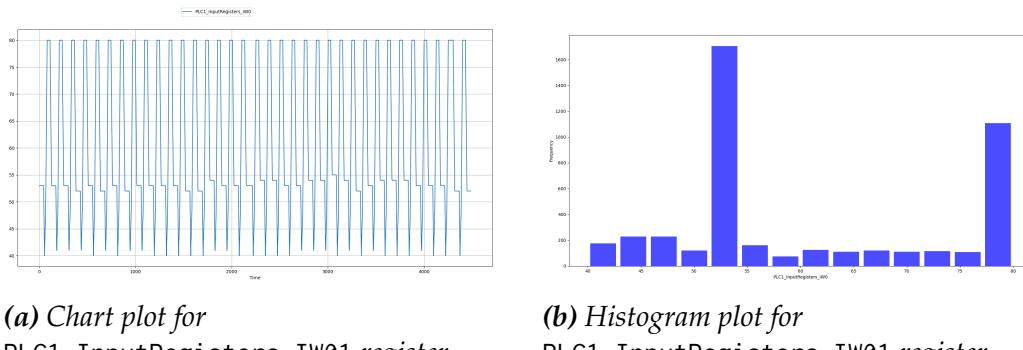


Figure 3.3: Output graphs from graph analysis

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-**

34 3.2 Ceccato et al.'s black-box dynamic analysis for water-tank systems

750 **tistical informations** about the register entered as command-line input.

751 These informations include:

- 752 • the mean, median, standard deviation, maximum value and mini-
753 mum value
- 754 • two tests for the statistical distribution: *Chi-squared* test for unifor-
755 mity and *Shapiro-Wilk* test for normality, as shown in Listing 3.3:

```
756     Chi-squared test for uniformity
757     Distance      pvalue      Uniform?
758     12488.340    0.00000000    NO
759
760     Shapiro-Wilk test for normality
761     Test statistic   pvalue      Normal?
762     0.844        0.00000000    NO
763
764     Stats of PLC1_InputRegisters_IW0
765     Sample mean = 60.8881; Stddev = 13.0164; max = 80; min =
766     ↪ 40 for 4488 values
```

Listing 3.3: Statistical data for PLC1_InputRegisters_IW0 register

767 3.2.4 Invariant Inference and Analysis

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

775 Daikon uses *machine learning* techniques applied to arbitrary data with
776 the possibility of setting custom conditions for analysis by using a spe-
777 cific file [47] with a *.spinfo* extension (see Listing 3.4). The framework is
778 designed to find the invariants of a program, with various supported pro-
779 gramming languages, starting from the direct execution of the program

780 itself or passing as input the execution run (typically a file in CSV format);
 781 the authors of the paper tried to apply it by analogy also to the execution
 782 runs of a cyber physical system, to extract the invariants of this system.

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

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

786 Therefore, Daikon is fed with the enriched dataset obtained in the pre-
 787 processing phase²: a simple bash script launches Daikon (optionally spec-
 788 ifying the desired condition for analysis in the *.spininfo* file), which output is
 789 simply redirected to a text file containing the general invariants of the sys-
 790 tem (i.e., valid regardless of any custom condition specified), those gener-
 791 ated based on the custom condition in the *.spininfo* file, and those generated
 792 based on the negation of the condition (see Listing 3.5 below).

```
793 =====
794 aprogram.point:::POINT
795 PLC2_MemoryRegisters_MW1 == PLC3_MemoryRegisters_MW1
796 PLC1_MemoryRegisters_MW0 == 40.0
797 PLC1_MemoryRegisters_MW1 == 80.0
798 PLC1_Coils_QX00 one of { 0.0, 1.0 }
799 [...]
800 =====
801 aprogram.point:::POINT; condition="PLC1_InputRegisters_IW0
802   ↪ > 60"
803 PLC1_InputRegisters_IW0 > PLC1_MemoryRegisters_MW0
804 PLC1_InputRegisters_IW0 > PLC1_Min_safety
805 PLC1_MemoryRegisters_MW0 < prev_PLC1_InputRegisters_IW0
806 [...]
807 =====
808 aprogram.point:::POINT; condition="not(
809   ↪ PLC1_InputRegisters_IW0 > 60)"
810 PLC1_InputRegisters_IW0 < PLC1_MemoryRegisters_MW1
811 PLC1_InputRegisters_IW0 < PLC1_Max_safety
```

²In 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

36 3.2 Ceccato et al.'s black-box dynamic analysis for water-tank systems

```
812     PLC1_MemoryRegisters_MW1 > prev_PLC1_InputRegisters_IW0  
813     [...]
```

Listing 3.5: The three sections of Daikon analysis outcomes

814 When the analysis is finished, the user is asked to enter the name of a reg-
815 istry to view its related invariants.

816

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

- 818 • measurements bounded by some setpoint;
- 819 • actuators state changes occurred in the proximity of setpoints or,
820 vice versa, proximity of setpoints upon the occurrence of an actuator
821 state change;
- 822 • state invariants of some actuators correspond to a specific trend in
823 the evolution of the measurements (ascending, descending, or sta-
824 ble) or, vice versa, the measurements trend corresponds to a specific
825 state invariant of some actuators.

826 3.2.5 Business Process Mining and Analysis

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

831

832 In the considered system, process mining begins by analyzing the event
833 logs derived from scanning the memory registers of the PLCs and moni-
834 toring the network communications associated with the Modbus protocol,
835 as detailed in Subsection 3.2.2. These event logs serve as the *execution trace*
836 of the system. A Java program is utilized to extract and consolidate infor-
837 mation from these event logs, resulting in a CSV format file that captures
838 the relevant data.

839 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.

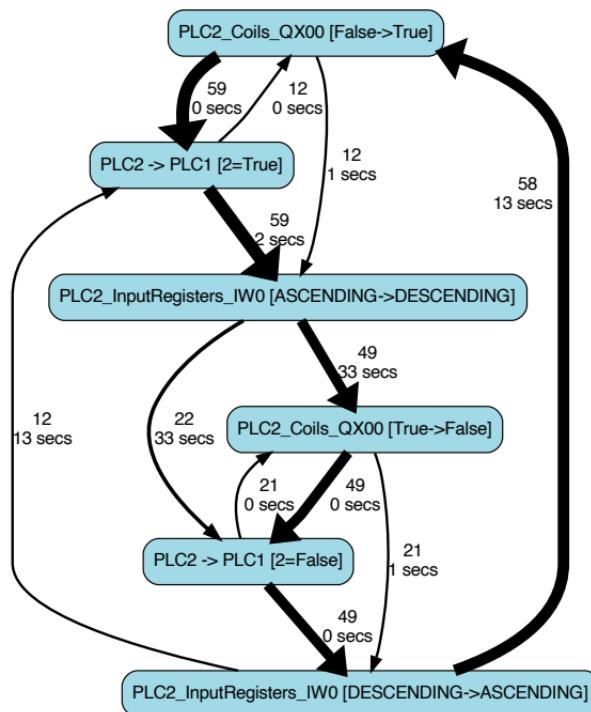


Figure 3.4: An example of Disco generated activity diagram for PLC2

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.

854 3.2.6 Application

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

863 **Data Collection and Pre-processing** According to what is described in
864 the paper, the data gathering process lasted six hours, with a granular-
865 ity of one data point per second (a full system cycle takes approximately
866 30 minutes). Each datapoint consists of 168 attributes (55 registers plus
867 a special register concerning the tank slope of each PLC) after the en-
868 richment. In addition, IP addresses are automatically replaced by an ab-
869 stract name identified by the prefix PLC followed by a progressive integer
870 (PLC1, PLC2, PLC3), in order to make reading easier.

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

873 **Property 1:** PLC1_MemoryRegisters_MW0, PLC1_MemoryRegisters_MW1,
874 PLC2_MemoryRegisters_MW0, PLC2_MemoryRegisters_MW1,
875 PLC3_MemoryRegisters_MW0 and PLC3_MemoryRegisters_MW1
876 registers contain constant integer values (40, 80, 10, 20, 0, 10 respec-
877 tively)³. The authors speculate that they may be (relative) hardcoded
878 **setpoints**.

³From 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.

879 **Property 2:** PLC1_Coils_QX01, PLC1_Coils_QX02, PLC2_Coils_QX01,
 880 PLC2_Coils_QX02, PLC3_Coils_QX01 and PLC3_Coils_QX03 contain mu-
 881 table binary (Boolean) values. The authors speculate that these reg-
 882 isters can be associated with the **actuators** of the system.

883 **Property 3:** PLC1_InputRegisters_IW0, PLC2_InputRegisters_IW0 and
 884 PLC3_InputRegisters_IW0 registers contain mutable values.

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

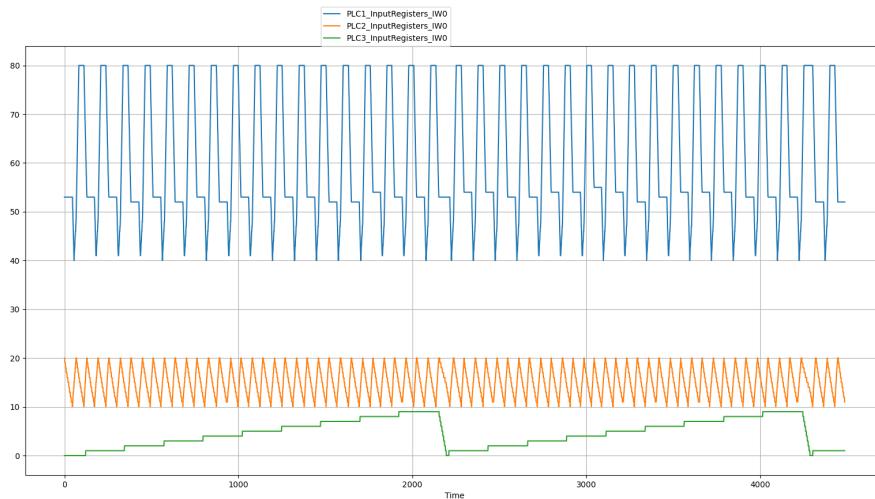


Figure 3.5: Execution traces of InputRegisters_IW0 on the three PLCs

888 The graph analysis of the InputRegisters_IW0 registers of the three
 889 PLCs (summarized in Figure 3.5 with a single plot) not only seems to con-
 890 firm the conjecture, but also allows the measurements to be correlated with
 891 the contents of the MemoryRegisters_MW0 and MemoryRegisters_MW1 regis-
 892 ters to the measurements, which may well represent the **relative setpoints**
 893 **of the measurements**. Hence, we have *Conjecture 2* described in the paper
 894 referring to the relative setpoints:

895

896 **Conjecture 2:**

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

40 3.2 Ceccato et al.'s black-box dynamic analysis for water-tank systems

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

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

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

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

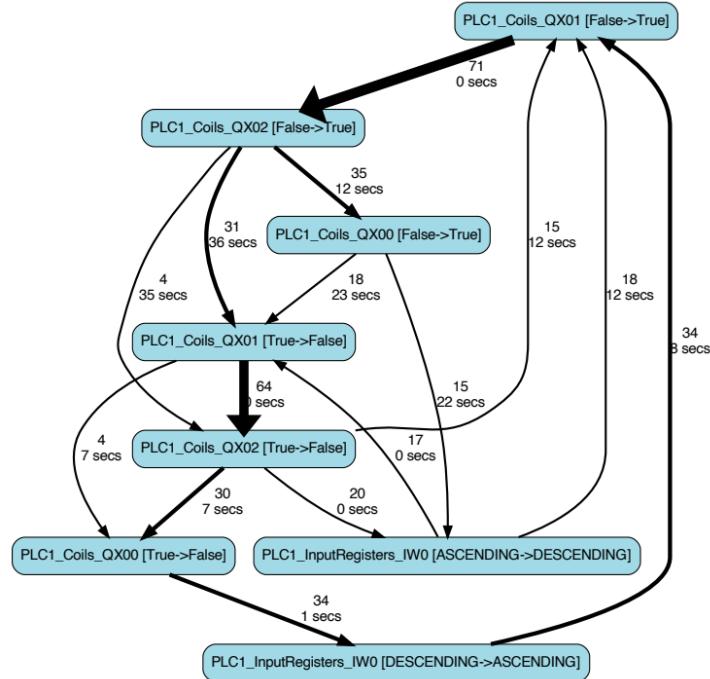
909 **Property 4:** PLC2 sends messages to PLC1 (see Figure 3.6b) which are
910 recorded to PLC1_Coils_QX02.

911 **Conjecture 3:** PLC2_Coils_QX00 determines the trend in tank T-202 (Figure
912 3.6b).

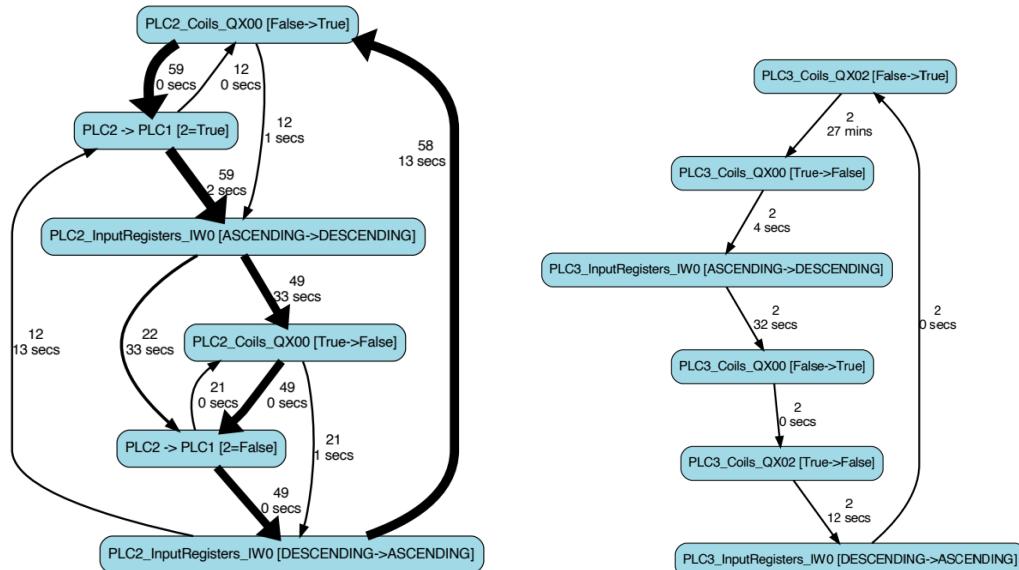
913 When this register is set to *True*, the input register PLC2_InputRegisters_IW0
914 related to the tank controlled by PLC2 starts an **ascending trend**; vice
915 versa, when the coil register is set to *False*, the input register starts a
916 **descending trend**.

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

920 **Conjecture 5:** PLC3_Coils_QX00 starts a **decreasing trend** in tank T-203, re-
921 lated to PLC3_InputRegisters_IW0 and controlled by PLC3, whereas
922 PLC3_Coils_QX02 starts an **increasing trend** on the tank (see Figure
923 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

42 3.2 Ceccato et al.'s black-box dynamic analysis for water-tank systems

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

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

937 However, it was possible to have confirmation of the conjectures made
938 in the previous stages of the analysis: starting with the setpoints, analyz-
939 ing the output of the invariants returned by Daikon⁴ reveals that

940
941 PLC1_InputRegisters_IW0 >= PLC1_MemoryRegisters_MW0 == 40.0
942 PLC1_InputRegisters_IW0 <= PLC1_MemoryRegisters_MW1 == 80.0
943 PLC2_InputRegisters_IW0 >= PLC2_MemoryRegisters_MW0 == 10.0
944 PLC2_InputRegisters_IW0 <= PLC2_MemoryRegisters_MW1 == 20.0
945 PLC3_InputRegisters_IW0 >= PLC3_MemoryRegisters_MW0 == 0.0
946 PLC3_InputRegisters_IW0 <= PLC3_MemoryRegisters_MW1 == 9.0
947
948 i.e., that the MemoryRegisters_MW0 and MemoryRegisters_MW1 registers of
949 each PLC contain the **absolute minimum and maximum setpoints**, re-
950 spectively (*Property 5*).

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

⁴The 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

953
954 PLC1_Coils_QX01 == PLC1_Coils_QX02 == PLC2_Coils_QX00
955
956 and from this derive that there is a **communication channel between PLC2**
957 **and PLC1**, where the value of PLC2_Coils_QX00 is copied to PLC1_Coils_QX01
958 and PLC1_Coils_QX02 (*Property 6*).

959 Regarding the **relationships between actuator state changes and mea-**
960 **surement trends**, invariant analysis yields the results summarized in the
961 following rules:

962 *Property 7:* Tank T-202 level *increases* iff PLC1_Coils_QX01 == True. Oth-
963 erwise, if PLC1_Coils_QX01 == False will be *non-increasing*.

964 This is because if the coil is *True* the condition

965 PLC2_InputRegisters_IW0 == PLC2_MemoryRegisters_MW0 == 20.0 && PLC2_slope > 0
966 is verified. On the opposite hand, if the coil is *False*, the condition
967 PLC2_InputRegisters_IW0 == PLC2_MemoryRegisters_MW0 == 20.0 && PLC2_slope <= 0 is verified. The
968 *slope* is increasing if > 0, decreasing if < 0, stable otherwise.

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

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

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

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

44 3.2 Ceccato et al.'s black-box dynamic analysis for water-tank systems

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

984 3.2.7 Limitations

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

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

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

999 What severely limits the analysis performed with the tool implemented
1000 by Ceccato et al. is the use of *ad hoc* solutions and *a posteriori* interventions
1001 done manually on the datasets after the data gathering process: we will
1002 discuss this last aspect in more detail later.

1003 Moreover, there is the presence of many *hardcoded* variables and condi-
1004 tions within the scripts: this makes the system unconfigurable and unable
1005 to properly perform the various stages of the analysis as errors can occur
1006 due to incorrect data and mismatches with the system under analysis.

1007 Having considered, furthermore, only the Modbus protocol for network
1008 communications between the PLCs is another major limiting factor and

1009 does not help the methodology to be adaptable to different systems com-
1010 municating with different protocols (sometimes even multiple ones on the
1011 same system).

1012 Let us now look at the limitations and critical aspects of each phase.

1013 **Testbed** The testbed environment used by Ceccato et. al is entirely simu-
1014 lated, from the physical system to the control system. The PLCs were built
1015 with **OpenPLC** [50] in a Docker environment [51], while the physics part
1016 was built through **Simulink** [52].

1017 OpenPLC is an open source cross-platform software that simulates the
1018 hardware and software functionality of a physical PLC and also offers a
1019 complete editor for PLC program development with support for all stan-
1020 dard languages: *Ladder Logic* (LD), *Function Block Diagram* (FBD), *Instruc-*
1021 *tion List* (IL), *Structured Text* (ST), and *Sequential Function Chart* (SFC).

1022 It is for sure an excellent choice for creating a zero-cost industrial or home
1023 automation and *Internet of Things* (IoT) system that is easy to manage via a
1024 dedicated, comprehensive and functional web interface. In spite of these
1025 undoubted merits, however, there are (at the moment) **very few supported**
1026 **protocols**: the main one and also referred to in the official documentation
1027 is **Modbus**, while the other protocol is DNP3.

1028 **First limitation** The biggest problem with the testbed, however, is not
1029 with the controller part, but with the **physical part**: first of all, it
1030 must be said that although this is something purely demonstrative
1031 even though it is fully functional, the implemented Simulink model
1032 is really **oversimplified** compared to the iTrust SWaT system, which
1033 itself is a scaled-down version of a real water treatment plant. In
1034 fact, in the entire system there are only three actuators, two of which
1035 are connected to the same tank and controlled by the same PLC, and
1036 sensors related only to the water level in the system's tanks: in a
1037 real system there are many more *field devices*, which can monitor and
1038 control other aspects of the system beyond the mere contents of the

46 3.2 Ceccato et al.'s black-box dynamic analysis for water-tank systems

1039 tanks. Consider, for example, measuring and controlling the chemi-
1040 cals in the water, the pressure of the liquid in the filter unit, or more
1041 simply the amount of water flow at a given point or time.

1042 All these must be considered and represent a number of additional
1043 variables that makes analysis and consequently reverse engineering
1044 of the system more difficult.

1045 ***Second limitation*** The second critical aspect concerns the **simulation of**
1046 **the physics of the liquid** inside the tanks: Simulink does not con-
1047 sider the fact that inside a tank that is filling (emptying) the liquid
1048 in it undergoes **fluctuations** which cause the level sensor not to see
1049 the water level constantly increasing (decreasing) or at most being
1050 stable at each point of detection. Figure 3.7 exemplifies more clearly
1051 with an example the concept just expressed: these oscillations cause
1052 a **perturbation** in the data.

1053 This issue leads to the difficulty, on a real physical system, of **cor-**
1054 **rectly calculating the trend of a measurement** by using the slope
1055 attribute: if this was obtained with a too low granularity, the trend
1056 will be oscillating between increasing and decreasing even when in
1057 reality this would be in general increasing (decreasing) or stable; on
1058 the other hand, if the slope was obtained with a too high granularity
1059 there is a loss of information and the trend may be "flattened" with
1060 respect to reality.

1061 In the present case, the slope in the Simulink model was calculated
1062 statically with a (very) low granularity, 5 and 6 seconds according
1063 to the Properties 7 and 9 described in the original paper: an aver-
1064 agely careful reader will have already guessed that this granularity
1065 is inapplicable to the real system in Figure 3.7b. As we will later see,
1066 we need to **operate on the data perturbations** to be able to obtain a
1067 suitable granularity and a correct calculation of the slope and conse-
1068 quently of the measurement trend.

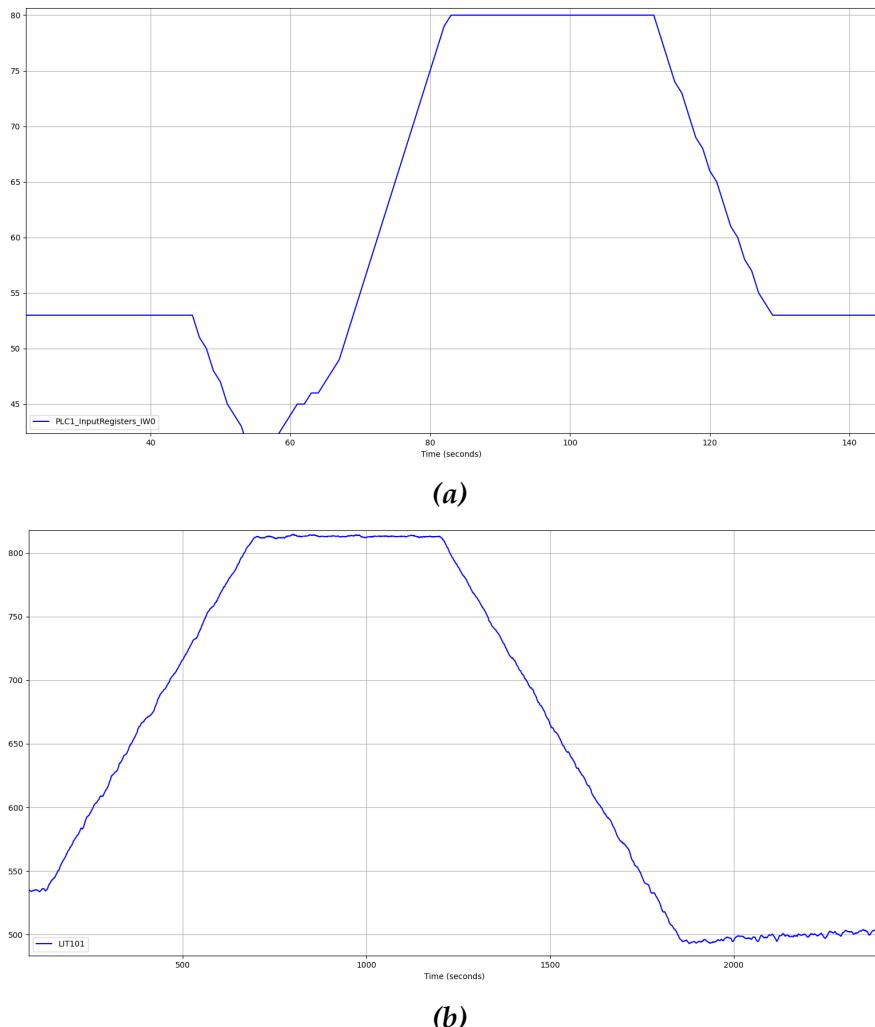


Figure 3.7: Water physics compared: simulated physics in the Simulink model (a) and physics in a real system (iTrust SWaT) (b). Fluctuations in the tank level in (b), almost completely absent in (a), can be appreciated.

1069 **Pre-processing** In the pre-processing phase, the authors make use of a
 1070 Python script to merge all the datasets of the individual PLCs into a single
 1071 dataset, remove the (supposedly) unused registers, and finally enrich the
 1072 obtained dataset with additional attributes. These attributes, as seen in
 1073 3.2.2, are:

- 1074 • the **previous value** of all registers;

48 3.2 Ceccato et al.'s black-box dynamic analysis for water-tank systems

- 1075 • some **additional relative setpoints** named PLC x _Max_safety and
1076 PLC x _Min_safety (where x is the PLC number), which represent a
1077 kind of alert on reaching the maximum and minimum water levels
1078 of the tanks;
- 1079 • the **measurement slope**.

1080 ***First limitation*** Merging the datasets of all individual PLCs into a single
1081 dataset representing the entire system can be a sound practice if the
1082 system to be analyzed is (very) small as is the testbed analyzed here,
1083 consisting of a few PLCs and especially a few registers. If, however,
1084 the complexity of the system increases, this type of merging can be-
1085 come counterproductive and make it difficult to analyze and under-
1086 stand the data obtained in subsequent steps.

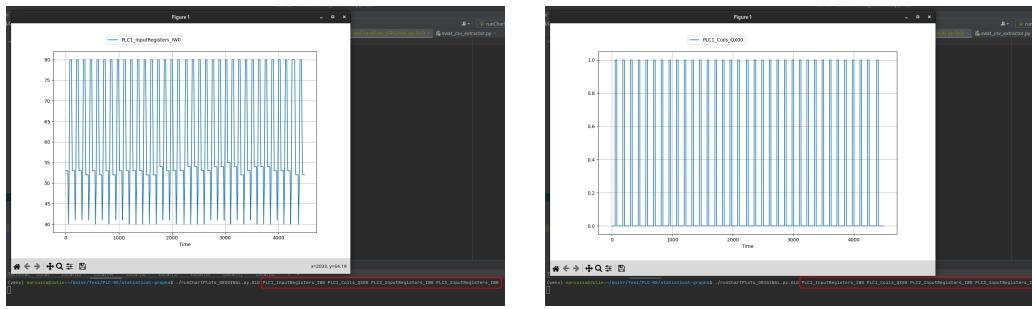
1087 In short, there is no possibility to analyze only a subsystem and thus
1088 make the analysis faster and more understandable. Moreover, a data
1089 gathering can take up to days, and the analyst/attacker may need to
1090 make an analysis of the system isolating precise time ranges, ignor-
1091 ing everything that happens before and/or after: all of this, with the
1092 tool we have seen, cannot be done.

1093 ***Second limitation*** Regarding the additional attributes, looking at the code
1094 of the script that performs the enrichment, we observed that **some at-**
1095 **tributes were manually inserted** after the merging phase: we are re-
1096 ferring in particular to the attributes PLC x _Max_safety and PLC x _Min_safety,
1097 whose references were moreover hardcoded into the script, and the
1098 *slope* whose calculation method we mentioned in the previous para-
1099 graph about the testbed limitations.

1100 In the end, only the attribute *prev* related to the value at the previous
1101 point of the detection is inserted automatically for all registers, more-
1102 over without the possibility to choose whether this attribute should
1103 be extended to all registers or only to a part.

1104 **Graphs and Statistical Analysis** Describing the behavior of graphical
1105 analysis in Section 3.2.3 we had already mentioned that only one register

1106 plot at a time was shown and not, for example, a single window containing
 1107 the charts of all registers entered by the user as input from the com-
 1108 mand line, such as in Figure 3.5. Figure 3.8 shows the actual behavior
 1109 of graphical analysis: note that although we have specified four registers
 1110 (highlighted in red in the figures) as command-line parameters, only one
 1111 at a time is shown and it is necessary to close the current chart in order to
 1112 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

1113 ***First limitation*** While displaying charts for individual registers still pro-
 1114 vides useful information about the system such as the distinction
 1115 between actuators and measurements and the general trend of the
 1116 latter, single display does not allow one to catch, or at least makes it
 1117 difficult, the relationship that exists between actuators and measure-
 1118 ments, where it exists, because a view of the system as a whole is
 1119 missing.

1120 In this way, the risk is to make conjectures about the behavior of the
 1121 system that may prove to be at least imprecise, if not inaccurate.

1122 ***Second limitation*** On the other hand, regarding the statistical analysis,
 1123 two observations need to be made: the first is that for the given sys-
 1124 tem, I personally was unable to appreciate the usefulness of the gen-
 1125 erated histogram in Figure 3.3b, as it does not provide any particular
 1126 new information that has not already been obtained from the graph-
 1127 ical analysis (except maybe something marginal); the second obser-

50 3.2 Ceccato et al.'s black-box dynamic analysis for water-tank systems

1128 vation pertains to the presentation of statistical information obtained
1129 from the histogram plot. In certain cases, the histogram plot itself
1130 can overshadow the displayed statistical information. These statis-
1131 tics are actually shown on the terminal from which the script is exe-
1132 cuted. However, to an inattentive or unfamiliar user, these statistics
1133 may be mistaken for debugging output or warnings, as they coin-
1134 cide with the display of the histogram plot window, which takes the
1135 focus (see Figure 3.9).

1136

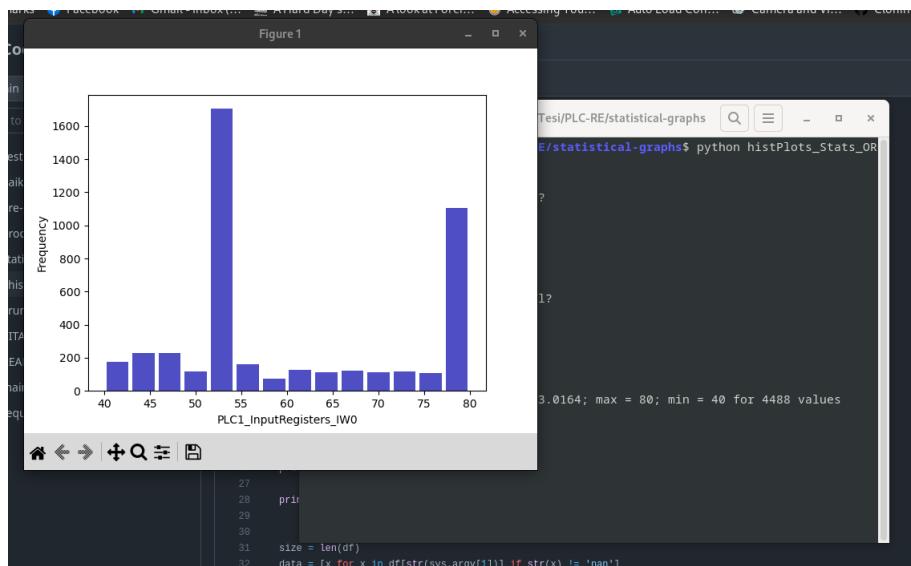


Figure 3.9: Histogram plot overshadowing statistical information shown on the terminal window in the background

1137

In general, however, little statistical information is provided.

1138 **Business Process Mining and Analysis** Concerning the data mining,
1139 this is a purely *ad hoc solution*, designed to work under special conditions:
1140 first, the timestamped dataset of the physical process and the one obtained
1141 after the packet sniffing operation of Modbus traffic on the network need
1142 to be synchronized and have the same granularity, in this case one event
1143 per second.

1144 It is relatively easy, therefore, to find correspondences between Modbus

1145 commands sent over the network and events occurring on the physical
1146 system, such as state changes in actuations, due in part to the fact that the
1147 number of communications over the network is really small (see Section
1148 3.2.1).

1149 ***First limitation*** In a real system, network communications are much more
1150 numerous and involve many more devices even in the same second:
1151 finding the exact correspondence with what is happening in the cy-
1152 ber physical system becomes much more difficult.

1153 Since this is, as mentioned, an *ad hoc* solution, only the Modbus pro-
1154 tocol is being considered: as widely used as this industrial protocol
1155 is, other protocols that are widely used [53] such as EtherNet/IP (see
1156 Section 2.2.6.2) or Profinet should be considered in order to extend
1157 the analysis to other industrial systems that use a different commu-
1158 nication network.

1159 ***Second limitation*** The other limiting aspect of the business process min-
1160 ing phase is the **process mining software** used to generate the ac-
1161 tivity diagram. As mentioned in Section 3.2.5, the process mining
1162 software used by Ceccato et al. is **Disco**: this is commercial soft-
1163 ware, with an academic license lasting only 30 days (although free of
1164 charge), released for Windows and MacOS operating systems only,
1165 which makes its use under Linux systems impossible except by us-
1166 ing emulation environments such as Wine.

1167 For what is my personal vision and training as a computer scientist,
1168 it would have been preferable to use a *cross-platform, freely licensed*
1169 *open source* software alternative to Disco: one such software could
1170 have been **ProM Tools** [54], a framework for process mining very
1171 similar to Disco in functionality, but fitting the criteria just described,
1172 or use Python libraries such as **PM4PY** [55], which offer ready-to-use
1173 algorithms suitable for various process mining needs.

52 3.2 Ceccato et al.'s black-box dynamic analysis for water-tank systems

1174 **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.

```
daikon_results_cond.txt
~/UniVr/Tesi/PLC-RE/daikon/Daikon_Invariants

Daikon version 5.8.14, released October 6, 2022; http://plse.cs.washington.edu/daikon.
Reading splitter info files
(read 1 spinfo file, 1 splitter)
Reading declaration files .aprogram.point:::POINT: 1 of 1 splitters successful

(read 1 decls file)
Processing trace data; reading 1 dtrace file:

Warning: No non-obvious non-suppressed exclusive invariants found in
aprogram.point:::POINT
Warning: No non-obvious non-suppressed exclusive invariants found in
aprogram.point:::POINT
=====
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 }
PLC1\_Coils\_QX01 one of { 0.0, 1.0 }
PLC1\_Coils\_QX02 one of { 0.0, 1.0 }
PLC2\_MemoryRegisters\_MW1 == 10.0
PLC2\_MemoryRegisters\_MW2 == 20.0
PLC2\_Coils\_QX00 one of { 0.0, 1.0 }
PLC3\_InputRegisters\_IW0 >= 0.0
PLC3\_Coils\_QX00 one of { 0.0, 1.0 }
PLC3\_Coils\_QX02 one of { 0.0, 1.0 }
prev\_PLC1\_Coils\_QX00 one of { 0.0, 1.0 }
prev\_PLC1\_Coils\_QX01 one of { 0.0, 1.0 }
prev\_PLC2\_Coils\_QX00 one of { 0.0, 1.0 }
prev\_PLC3\_InputRegisters\_IW0 >= 0.0
prev\_PLC3\_Coils\_QX00 one of { 0.0, 1.0 }
prev\_PLC3\_Coils\_QX02 one of { 0.0, 1.0 }
PLC1\_Max\_safety == 77.0
```

Figure 3.10: Example of Daikon's output

1180 **First limitation** The biggest problem with Daikon applied to the computation of invariants of an industrial system is the difficult reading 1181 of the resulting output: the software in fact returns a very long list 1182 of invariants, one invariant per line, many of no use and without 1183 correlating invariants that may have common features or deriving 1184

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	<ul style="list-style-type: none"> - Oversimplified model compared to iTrust SWaT system - Physics of the liquid not considered: this causes data perturbation
Pre-processing	<ul style="list-style-type: none"> - 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	<ul style="list-style-type: none"> - Only one chart at the time is displayed: difficulty in capturing the relationship between actuators and sensors - Statistical Analysis provides little

54 3.2 Ceccato et al.'s black-box dynamic analysis for water-tank systems

	information
Business Process Analysis	<ul style="list-style-type: none">- Ad hoc solution designed to work under special conditions- Use of commercial software for process mining
Invariant Analysis	<ul style="list-style-type: none">- Reading output is challenging- Script for analysis merely launches Daikon without reprocessing outcomes

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

A Substantial Improvement to Ceccato et al.'s Framework

1202 IN Chapter 3, we presented the state of the art of *process comprehension*
1203 of an Industrial Control System (ICS) with a focus on the methodology
1204 proposed by Ceccato et al. [9][Section 3.2], explaining what it consists of,
1205 its practical application on a testbed, and most importantly highlighting
1206 its limitations and critical issues (see Section 3.2.7).

1207 In this chapter we will present a **proposal to improve the method-**
1208 **ology** presented in the previous chapter, addressing most of the critical
1209 issues (or at least trying to do so) mentioned above by almost completely
1210 rewriting the original framework, enhancing its functionalities and insert-
1211 ing new ones where possible, while preserving its general structure and
1212 approach. The system analysis will in fact consist of the same four steps
1213 as in the original methodology (Data Pre-processing, Graph and Statistical
1214 Analysis, Business Process Mining and Invariants Inference), but each of
1215 them will be deeply revised in order to provide a richer, clearer and more
1216 complete process comprehension of the industrial system to be analyzed
1217 and its behavior.

1218 As it may have already been noted, my proposals do not involve im-
1219 proving the data gathering phase: this is due simply to the fact that the
1220 novel framework will not be tested on the same case study used by Cec-

1221 cato et, al. (Section 3.2.1), but on a different case study, the iTrust SWaT
1222 system [36], of which (some) datasets containing the execution trace of
1223 the physical system and the network traffic scan are already provided by
1224 iTrust itself. For more details about this case study, the reader is referred
1225 to Chapter 5.

1226 4.1 The Proposed New Framework

1227 In our version of the framework we decided to follow a few design
1228 choices:

- 1229 1. it must be implemented in a **single programming language**;
- 1230 2. it must be **independent of the system** to be analyzed;
- 1231 3. It must provide greater **flexibility and ease of use** for the user at
1232 every stage.

1233 In the following, we discuss these three features in more detail.

1234 **Single Programming Language** The original tool was implemented us-
1235 ing various programming languages in each of the different phases:
1236 from Python up to Java, passing through Bash scripting.
1237 In our opinion, this heterogeneity makes it more difficult and less
1238 intuitive for the user to operate on the tool: moreover, the use of
1239 multiple technologies makes it more difficult to maintain the code
1240 and add new features, particularly if only a single person is manag-
1241 ing the code (he/she might be proficient in one language, but little
1242 of the others).

1243 For these reasons, we decided to use a single programming language,
1244 to ensure homogeneity to the framework and ease of use and main-
1245 tenance of the code for anyone who wants to manage it in the future:
1246 we chose to use Python, because of its simplicity and easy readability
1247 combined with its versatility and powerfulness: moreover, Python

1248 can count on a massive number of available libraries and packages
1249 that meet all kinds of needs.

1250 **System Independence** One of the biggest limitations of Ceccato et al.'s
1251 tool that we highlighted in Section 3.2.7 is the fact that it is **highly**
1252 **dependent on the testbed used**: that is, it is *not* possible to configure
1253 any of the tool's parameters to analyze different industrial systems.
1254 To overcome this issue and make my framework independent of the
1255 system to be analyzed, also eliminating all references to hardcoded
1256 variables and values present in the previous tool, we decided to use a
1257 **general configuration file**, named *config.ini*, in which the user can, at
1258 will, customize all the parameters necessary to perform the analysis
1259 of the targeted system.

1260 **Flexibility and Ease on Use** The lack of flexibility and ease of use in a tool
1261 can be a significant disadvantage, limiting its effectiveness and mak-
1262 ing it challenging for the user to get the desired outcomes. The orig-
1263 inal tool suffered from these limitations, with users having to run
1264 scripts from the command line, with little to no options or param-
1265 eters available to customize the analysis. As a result, the tool was
1266 not user-friendly and lacked the flexibility to adapt to specific user
1267 needs.

1268 To settle these issues, I enhanced the command-line interface in the
1269 novel framework by adding new options and parameters. These
1270 new features provide the user with greater flexibility, enabling to
1271 specify parameters and options that allow for more in-depth anal-
1272 ysis and focused results analyzing data more effectively and effi-
1273 ciently. With these enhancements, the framework has become more
1274 user-friendly, reducing the learning curve and making it more acces-
1275 sible to a wider range of users.

1276 This, in turn, makes the framework more valuable and useful, in-
1277 creasing its adoption and effectiveness across a range of industrial
1278 control systems and applications.

1279 Moreover, with new options and parameters users no longer have to
 1280 rely solely on the command line interface, which can be challenging
 1281 and intimidating for those with limited technical expertise. Instead,
 1282 users can now access a range of customizable options and parame-
 1283 ters, making the tool more intuitive and user-friendly.

1284 Overall, the enhancements made to the framework represent a significant
 1285 step forward in making it more effective, efficient, and user-friendly.

1286 4.1.1 Framework Structure

1287 The proposed framework follows a similar structure to the original
 1288 tool, with a division into five main directories representing different phases
 1289 of the analysis: **data pre-processing, graphs and statistical analysis, pro-**
1290 cess mining, and invariant analysis. A new phase is added compared to
 1291 the original, concerning the **analysis of the network traffic**. These directo-
 1292 ries contain the corresponding Python scripts responsible for performing
 1293 the analysis, along with any necessary subdirectories and input/output
 1294 files to ensure the proper functioning of the framework.

```
1295 .
1296   |-- config.ini
1297   |-- daikon
1298     |-- Daikon_Invariants
1299     |-- daikonAnalysis.py
1300     |-- runDaikon.py
1301   |-- network-analysis
1302     |-- data
1303     |-- networkAnalysis.py
1304     |-- export_pcap_data.py
1305     |-- swat_csv_extractor.py
1306   |-- pre-processing
1307     |-- mergeDatasets.py
1308     |-- system_info.py
1309   |-- process-mining
1310     |-- data
1311     |-- process_mining.py
```

```
1312     L-- statistical-graphs  
1313         |-- histPlots_Stats.py  
1314         |-- runChartSubPlots.py
```

Listing 4.1: Novel Framework structure and Python scripts

1315 Ahead of these directories there is the most important part, that allows
1316 the framework to be independent of the industrial control system being
1317 analyzed: the *config.ini* file. Here the user can configure general parame-
1318 ters and options, such as paths to read from or write files to, or related to
1319 individual analysis phases.

1320 The file is divided into sections, each covering a different aspect of the con-
1321 figuration: each section contains user-customizable parameters that will
1322 then be called within the Python scripts that constitute the framework.

1323 Sections of *config.ini* are:

- 1324 • **[PATHS]**: defines general paths such as the project root directory;
- 1325 • **[PREPROC]**: includes parameters needed for the **pre-processing phase**,
1326 like the directory containing the raw datasets of the individual PLCs
1327 and *granularity* for the slope calculation. Granularity is the time in-
1328 terval over which the slope is calculated;
- 1329 • **[DATASET]**: defines settings and parameters used during the **dataset**
1330 **enrichment** stage, for example the additional attributes;
- 1331 • **[DAIKON]**: defines parameters needed for **invariant analysis** with
1332 Daikon, e.g. directories and files containing the outcomes of the anal-
1333 ysis;
- 1334 • **[MINING]**: contains parameters used during the **process mining**
1335 phase, such as data directory;
- 1336 • **[NETWORK]**: includes specific settings for extracting the data ob-
1337 tained from the packet sniffing phase on the ICS network and con-
1338 verting it to CSV format. It also defines the **network protocols** that
1339 are to be analyzed.

1340 4.1.2 Python Libraries and External Tools

1341 As the framework has been developed entirely in Python, the objec-
1342 tive was to minimize reliance on external tools and instead integrate vari-
1343 ous functionalities within the framework itself. The aim was to make the
1344 framework independent from external software. The only remaining ex-
1345 ternal tool from the Ceccato et al. tool is Daikon. This choice was made
1346 because there is currently no better alternative or Python package avail-
1347 able that offers the same functionalities as Daikon.

1348 Instead, the framework extensively utilizes Python libraries for han-
1349 dling various functionalities and input data. The core libraries on which
1350 the framework relies are:

- 1351 • **Pandas**, also used in the previous tool for dataset management, but
1352 whose use here has been deepened and extended
- 1353 • **NumPy**, often used together with Pandas to perform some opera-
1354 tions to support it;
- 1355 • **MatPlotLib**, for managing and plotting graphical analysis;
- 1356 • other scientific libraries such as **SciPy**, **StatsModel** [56] and **Net-**
1357 **workX** [57], for mathematical, statistical and analysis operations on
1358 the data;
- 1359 • **GraphViz**, for the creation of activity diagrams in the process mining
1360 phase.

1361 Having now seen the structure of the framework, in the next sections we
1362 will go into more detail describing our proposal.

1363 **4.2 Analysis Phases**

1364 **4.2.1 A Little Testbed: Stage 1 of iTrust SWaT System**

1365 Before we proceed with presenting the analysis steps of the proposed
1366 framework, let us introduce the testbed that will serve as an illustration for
1367 practical examples, demonstrating the effectiveness of our methodology
1368 and the potential of the framework. This testbed corresponds to Stage 1 of
1369 the iTrust SWaT (Secure Water Treatment) system [36]. The selection of this
1370 testbed is intentional, as it serves as a precursor to the comprehensive case
1371 study we will be addressing in the upcoming chapters. The iTrust SWaT
1372 system offers elements of greater complexity compared to the individual
1373 stages of the Ceccato et al. testbed.

1374 The testbed comprises several components, including:

- 1375 • a **PLC** responsible for monitoring and controlling the operations within
1376 the stage;
- 1377 • a **tank**, which serves as the main element of interest;
- 1378 • **two sensors** that provide readings of the water level within the tank
1379 and the incoming flow;
- 1380 • **three actuators**, namely a valve and two pumps. These actuators
1381 regulate the level within the tank by controlling the inflow and out-
1382 flow of the liquid.

1383 Despite its moderate complexity, this testbed provides an ideal plat-
1384 form for presenting straightforward and concise examples of the frame-
1385 work's behavior. It enables us to effectively demonstrate the potential
1386 of the framework and facilitate a deeper understanding of its underlying
1387 methodology.

1388 4.2.2 Phase 0: Network Analysis

1389 The objective of the network analysis presented in this section is to offer
1390 users valuable information regarding the communication process within
1391 an industrial control system and a broader perspective on network com-
1392 munications, delving into previously unexplored aspects of the system.
1393 These additional dimensions provide a deeper understanding of the sys-
1394 tem's behavior and characteristics. This analysis aims to provide users
1395 with an overview of the communication between Programmable Logic
1396 Controllers (PLCs) at level 1, as well as the communication between PLCs
1397 and devices at higher levels such as HMIs and Historian servers (see Sec-
1398 tion 2.1 for ICSs architecture). The analysis focuses on industrial protocols
1399 used and the information exchanged.

1400 By reconstructing the network communication structure using data ob-
1401 tained from the network traffic sniffing process, users can gain a compre-
1402 hensive understanding of the behavior of the underlying industrial sys-
1403 tem. This knowledge can then be utilized to **plan a strategy** for analyzing
1404 the physical processes within the system.

1405 4.2.2.1 Extracting Data from PCAP Files

1406 The initial step involves extracting the desired information from the
1407 PCAP files that contain the captured network traffic. This includes details
1408 such as the source IP address, destination IP address, protocol used, and
1409 the type of request made (e.g., Read/Write, Request/Response). The ex-
1410 tracted data is then converted into a more convenient CSV format. This
1411 extracted data serves as the foundation for the subsequent phase.

1412 In the latter part of this phase, the extracted data is utilized to generate
1413 the network schema. The network schema provides a visual representa-
1414 tion of the connections and relationships within the network, showcasing
1415 the communication patterns between different components. This schema
1416 helps in understanding the overall structure and behavior of the industrial
1417 control system.

1418 To accomplish the extraction of data from the PCAP files, a Python
1419 script called `export_pcap_data.py` is employed. This script, originally de-
1420 signed for the business process phase, is located in the directory
1421 `$(project_dir)/network-analysis` and accepts the following options as
1422 command-line arguments:

- 1423 • **-f or --filename:** allows the user to specify a single PCAP file to be
1424 passed as input to the script. The user can provide the complete file
1425 path of the PCAP file as an argument;
- 1426 • **-m or --mergefiles:** enables the merging of multiple PCAP files. In
1427 this scenario, the files should be located within the directory speci-
1428 fied by the `pcap_dir` directive in the `config.ini` configuration file and
1429 the user does not have to provide the path to each PCAP file;
- 1430 • **-d or --mergedir:** allows for specifying the directory that contains the
1431 PCAP files to be automatically imported into the script and merged.
1432 This ensures that all the PCAP files within the specified directory will
1433 be processed by the script without the need for manual selection or
1434 input.
- 1435 • **-s or --singledir:** operates differently from the previous option men-
1436 tioned. This option enables the extraction of data from each individ-
1437 ual PCAP file within the specified directory. The extracted data is
1438 then saved in separate CSV datasets, which are stored in the direc-
1439 tory specified by the `split_dir` directive in the `config.ini` file. This
1440 functionality proves useful when dealing with exceptionally large
1441 PCAP files, where merging them together for export might consume
1442 significant time and resources. By utilizing this option, the extrac-
1443 tion procedure becomes lighter and more manageable. The extracted
1444 data in separate CSV files can be utilized in the later stages of the
1445 Network Analysis process;
- 1446 • **-t or --timerange:** this functionality enables users to specify a specific
1447 time period within the PCAP files from which they wish to extract
1448 relevant information.

1449 Unless the `-s` option is explicitly specified, the results of data extraction
1450 and export will be saved to a single CSV dataset within the
1451 `$(project_dir)/network-analysis/data` directory. The default file name
1452 for this output file is determined by the `pcap_export_output` directive spec-
1453 ified in the `config.ini` file. In addition, by utilizing the protocols and
1454 `ws_<protocol>_field` directives, user can configure the network protocols
1455 to be searched within the PCAP files. Furthermore, user can specify the
1456 relevant Tshark/Wireshark fields to extract for the specified protocols set
1457 in the `protocols` directive.

1458 After obtaining the extracted data, it is possible to proceed with the
1459 second part of the network analysis.

1460 4.2.2.2 Network Information

1461 During this stage, the exported CSV data is processed to derive val-
1462 uable information regarding network communications and the structure of
1463 the network itself. The objective is to identify and establish relationships
1464 between IP addresses present on the network, thereby determining the
1465 sources and destinations of communications. Furthermore, the analysis
1466 detects the protocols used for each communication and quantifies the var-
1467 ious types of requests made.

1468 This information is then transformed into a **graph representation of**
1469 **the network** (or subnetwork, if specified). In this graph, devices are repre-
1470 sented as nodes labeled with their IP addresses, while edges represent the
1471 incoming and outgoing communications of these devices, along with the
1472 corresponding information.

1473 To ensure comprehensibility, the analysis also provides users with **textual**
1474 **information** containing the same details as the graph representation. This
1475 text-based information serves as an alternative for cases where the graph
1476 may become complex to interpret, particularly when numerous edges con-
1477 nect nodes and result in a high volume of network requests.

1478 This textual information is saved to another CSV file, enabling offline ref-

1479 erence or potential future utilization. By having this file available, users
1480 can access the network analysis results in a structured format for further
1481 analysis or documentation purposes.

1482 The Python script `networkAnalysis.py` in the `$(project_dir)/network-analysis`
1483 directory manages this phase of the analysis. The script can be executed
1484 with the following parameters:

- 1485 • **-f or --filename:** used to specify the CSV dataset containing the net-
1486 work data exported in the previous step. The dataset should be lo-
1487 cated in the directory `$(project_dir)/network-analysis/data`;
- 1488 • **-D or --directory:** used to specify the directory that contains the CSV
1489 datasets obtained using the `-s` option of the Python script `export_pcap_data.py`.
1490 By passing this parameter, the script will automatically merge the
1491 datasets and proceed with the analysis of the data contained within
1492 them;
- 1493 • **-s or --srcaddr:** allows for specifying the source IP address for which
1494 you wish to display the incoming and outgoing communications. By
1495 providing the source IP address as an argument, the script will focus
1496 on showcasing the communications associated with that particular
1497 IP address;
- 1498 • **-d or --dstaddr:** enables the user to specify the destination IP address
1499 for which you want to display the incoming and outgoing communi-
1500 cations. By providing the destination IP address as an argument, the
1501 script will concentrate on presenting the communications associated
1502 with that specific IP address.

1503 The parameters related to IP addresses, including source and destina-
1504 tion, are optional. It is possible to specify either one of them individually.
1505 For instance, if the user specifies only the source IP address, the script will
1506 display the network nodes with which it communicates on the outgoing
1507 side, along with the corresponding generated traffic. Similarly, if only the

1508 destination IP address is specified, the script will showcase the network
 1509 nodes communicating with it on the incoming side, along with the relevant traffic data.
 1510

1511 During the analysis, the script identifies and displays the IP addresses
 1512 present in the network as output for the user's reference. This allows the
 1513 user to select specific IP addresses from the command line for a more focused
 1514 analysis, such as choosing a subnet of interest. Additionally, the
 1515 script detects and tracks distinct communications between pairs of PLCs,
 1516 keeping a record of the number of these communications.

1517 The result of the analysis is a graph representation of the network (or sub-
 1518 network) to be analyzed. An example of such a graph can be seen in Figure
 1519 4.1.

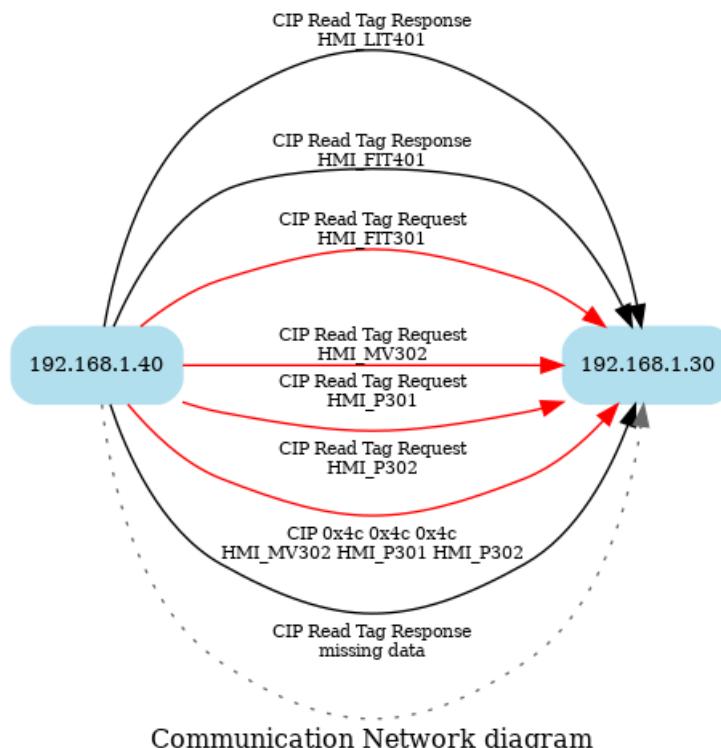


Figure 4.1: Network Communications between 192.168.1.40 (Source) and 192.168.1.30 (Destination)

1520 The graph illustrates the network communications between the source

1521 IP address 192.168.1.40 and the destination IP address 192.168.1.30. Each
 1522 arrow represents a communication between these IP addresses, showcasing
 1523 the flow and direction of the interactions. The red arrows indicate re-
 1524 quests initiated by the source IP address towards the destination IP, while
 1525 the black arrows represent responses sent by the source IP in response
 1526 to previous requests made by the destination IP. The gray dotted arrows
 1527 represent responses for which the corresponding request is missing or un-
 1528 available for some reason. Overall, the graph distinguishes the different
 1529 types of communications and provides insights into the request-response
 1530 dynamics between the source and destination IP addresses. The graph is
 1531 automatically generated and saved within the
 1532 `$(project_dir)/network-analysis/data` directory.

1533 In terms of the textual output, we can observe how the same data is
 1534 represented. The communications exchanged between the two PLCs are
 1535 displayed more prominently, allowing for a clearer understanding. Unlike
 1536 the graph, the textual representation includes a column on the right-hand
 1537 side, indicating the number of communications for each type of request.
 1538 This provides a more distinct perspective on the network behavior within
 1539 an industrial control system that utilizes, in this case, the CIP protocol for
 1540 its communications.

src	dst	protocol	service_detail	register	
192.168.1.40	192.168.1.30	CIP	Read Tag Response	HMI_LIT401	11249
				HMI_FIT401	10539
			Read Tag Request	HMI_FIT301	8031
				HMI_MV302	7209
				HMI_P301	7115
				HMI_P302	7040
			0x4c 0x4c 0x4c	HMI_MV302 HMI_P301 HMI_P302	1
			Read Tag Response	missing data	1

Figure 4.2: Network Communications between 192.168.1.40 (Source) and 192.168.1.30 (Destination) in textual mode

1541 As previously mentioned, the textual data is stored in a CSV dataset
 1542 located in the `$(project_dir)/network-analysis/data` directory.

1543 4.2.3 Phase 1: Data Pre-processing

1544 *Data Pre-processing phase* is probably the most delicate and significant
1545 one: depending on how large the industrial system to be analyzed is, the
1546 data collected, and how it is enriched using the additional attributes, the
1547 subsequent system analysis will provide more or less accurate outcomes.

1548 The previous tool has several limitations, particularly at this stage. It
1549 does not allow for the isolation of a subsystem, either in terms of time or
1550 the number of PLCs to be analyzed. The system is considered as a whole
1551 without the ability to focus on specific subsystems. Additionally, many of
1552 the additional attributes had to be manually added, and for the ones en-
1553 tered automatically, there is no way to specify the register type to associate
1554 them with.

1555 The combination of these limitations, along with the presence of hard-
1556 coded references to attributes and registers in the tool's code, makes the
1557 analysis of the system more challenging. Furthermore, it compromises the
1558 accuracy and reliability of the obtained results in terms of both quantity
1559 and quality.

1560 In the proposed framework, these issues have been addressed by in-
1561 corporating new features. Firstly, the framework allows for the selection
1562 of a subsystem from the command line based on both temporal criteria
1563 and the specific PLCs to be included. This enables more focused and tar-
1564 getted analysis. Additionally, we have revamped the process of enriching
1565 the resulting dataset by eliminating manual entry of additional attributes.
1566 Instead, users now have the flexibility to determine the type of additional
1567 attribute to associate with a specific register.

1568 Furthermore, after the pre-processing stage, a preliminary analysis can be
1569 conducted on the resulting dataset. This analysis aims to identify the reg-
1570 isters that are associated with actuators, measurements, and hardcoded
1571 setpoints or constants. It provides insights into the dataset and helps in
1572 refining the enrichment step. The parameters for this analysis can be con-
1573 figured in the *config.ini* file, allowing for customization and fine-tuning of

1574 the process.

1575

1576 In the upcoming sections, we will delve into a more comprehensive ex-
1577 amination of the achievements made in this framework.

1578 **4.2.3.1 Subsystem Selection**

1579 In the previous tool, the datasets for each individual PLC in CSV for-
1580 mat were required to be placed in a specific directory that was hardcoded
1581 in the script. The script would then merge and enrich these datasets to
1582 generate a single output dataset representing the complete process trace
1583 of the industrial system. However, the script did not provide options to
1584 select specific PLCs for analysis or define a temporal range for analysis.
1585 This lack of flexibility made the analysis more complex, especially when
1586 dealing with *transient states* (i.e., general states in which the industrial sys-
1587 tem is still initializing before actually reaching full operation) or when fo-
1588 cusing on specific parts of the industrial system during certain periods of
1589 interest. The fixed dataset structure also may increase the number of vari-
1590 ables that could be analyzed.

1591 Furthermore, the previous tool did not allow for specifying an output CSV
1592 file to save the resulting dataset. Each dataset creation and enrichment op-
1593 eration would overwrite the previous file, making it inconvenient for com-
1594 parisons between different execution traces unless the files were manually
1595 renamed.

1596 The proposed framework addressed these issues by introducing im-
1597 provements. First of all, in the general *config.ini* file there are some general
1598 default settings about paths, and among them the one concerning the di-
1599 rectory where to place the datasets of the individual PLCs to be processed.
1600 In addition to this option, there are other ones that define further aspects
1601 related to the operations performed in this phase. Listing 4.2 shows the
1602 settings in question:

1603 [PATHS]
1604 root_dir = /home/marcuzzo/UniVr/Tesi

```

1605     project_dir = %(root_dir)s/PLC-RE
1606     net_csv_path = %(root_dir)s/datasets_SWaT/2015/Network_CSV
1607
1608     [PREPROC]
1609     raw_dataset_directory = datasets_SWaT/2015 # Directory
1610         ↪ containing datasets
1611     dataset_file = PLC_SWaT_Dataset.csv # Default output
1612         ↪ dataset
1613     granularity = 10 # slope granularity
1614     number_of_rows = 20000 # Seconds to consider
1615     skip_rows = 100000 # Skip seconds from beginning

```

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

1616 At the same time, the user has the option to specify these settings via the
1617 command line using the new Python script called `mergeDatasets.py`, lo-
1618 cated in the pre-processing directory of the project. Any options provided
1619 through the command line will override the default settings specified in
1620 the `config.ini` file. These options are:

- 1621 • **-s or --skiprows:** initial transient period (expressed in seconds) to
1622 be skipped. This option is useful in case the system has an initial
1623 transient or the analyzer wishes to start the analysis from a specific
1624 point in the dataset;
- 1625 • **-n or --nrows:** time interval under analysis, expressed in terms of the
1626 number of rows in the dataset.
1627 This option makes a **selection** on the data of the dataset;
- 1628 • **-p or --plcs:** PLCs to be merged and enriched. The user can specify
1629 the desired PLCs by indicating the CSV file names of the associated
1630 datasets with no limitations on number.
1631 This option makes a **projection** on the data of the dataset.
- 1632 • **-d or --directory:** performs the merge and enrichment of all CSV files
1633 contained in the directory specified by user, overriding the default
1634 setting in `config.ini`. It is in fact the old functionality of the previous

1635 tool, maintained here to give the user more flexibility and convenience
1636 in case he wants to perform the analysis on the whole system.
1637 This is also the default behavior in case the `-p` option is not specified.

- 1638 • **-o or --output:** specifies the name of the file in which the obtained
1639 dataset will be saved. It must necessarily be a file in CSV format.
- 1640 • **-g or --granularity:** specifies a granularity (expressed in seconds)
1641 that will be used to calculate the measurement slope during the dataset
1642 enrichment phase. We will discuss this later in Section 4.2.3.2.

1643 4.2.3.2 Dataset Enrichment

1644 After a step in which a function is applied to each PLC-related dataset
1645 to eliminate its unused registers within the system¹, the **dataset enrichment**
1646 **operation** is performed.

1647 This operation differs from the previous version not only in the fact that it
1648 is performed on each individual dataset and not on the resulting dataset,
1649 but also in the additional attributes: not only are they greater in number,
1650 but they are automatically calculated and inserted by the `mergeDatasets.py`
1651 script into the dataset and, most importantly, it is possible to decide through
1652 the parameters in the `config.ini` configuration file under the [DATASET] sec-
1653 tion to which registers these attributes should be assigned.

1654 In Listing 4.3 we can see the list of additional attributes and how they
1655 should be associated with the registers of the dataset:

```
1656 [DATASET]
1657 timestamp_col = Timestamp
1658 max_prefix = max_
1659 min_prefix = min_
1660 max_min_cols_list = lit|ait|dpit
1661 prev_cols_prefix = prev_
1662 prev_cols_list = mv[0-9]{3}|p[0-9]{3}
1663 trend_cols_prefix = trend_
```

¹This 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

```

1664     trend_cols_list = lit
1665     trend_period = 150
1666     slope_cols_prefix = slope_
1667     slope_cols_list = lit

```

Listing 4.3: config.ini parameters for dataset enriching

1668 Following is a brief explanation of the parameters just seen:

1669 **timestamp_col** indicates the name of the column that contains the data
 1670 timestamps. This parameter is used not only in this phase, but is
 1671 also referred to in the Process Mining phase. In the previous work,
 1672 this parameter was hardcoded and not configurable (and thus caus-
 1673 ing errors if the system being analyzed changed)

1674 **max_prefix, min_prefix, max_min_cols_list** refer to any relative maximum
 1675 or minimum values (*relative setpoints*) of one or more measures and
 1676 that can be found and inserted as new columns within the dataset.
 1677 The first two parameters indicate the prefix to be used in the column
 1678 names affected by this additional attribute, while the third specifies
 1679 of which type of registers we want to know the maximum and/or
 1680 minimum value reached (several options can be specified using the
 1681 logical operator | - or).

1682 If, for example, we want to know the maximum value of the regis-
 1683 ters associated with the tanks, indicated in the iTrust SWaT system
 1684 by the prefix LIT, we only need to specify the necessary parameter in
 1685 the *config.ini* file, so `max_min_cols_list = lit`.

1686 The result will be to have in the dataset thus enriched a new column
 1687 named `max_P1_LIT101`.

1688 **prev_cols_prefix, prev_cols_list** refer to the values at the previous time
 1689 instant of the registers specified in `prev_cols_list`. It is possible to
 1690 specify registers using *regex*, as in the example shown. It may be use-
 1691 ful in some cases to have this value available to check, for example,
 1692 when a change of state of a single given actuator occurs. The behav-
 1693 ior of these parameters is the same as described in the point above.

1694 **slope_cols_prefix, slope_cols_list** are related to the calculation of the
1695 slope of a specific register that contains numeric values (usually a
1696 measure), that is, its trend. Slope calculation makes little sense on
1697 booleans. The slope can be **ascending** (if its value is greater than
1698 zero), **descending** (if less than zero) or **stable** (if approximately equal
1699 to zero). We will delve into the details of slope calculation in the fol-
1700 lowing paragraph, as it pertains to the attributes **trend_cols_prefix**,
1701 **trend_cols_list**, and **trend_period**.

1702 Initially, the parameters for registers to be associated with each addi-
1703 tional attribute may be left blank, as we may not have prior knowledge
1704 about the system and are unsure about which registers correspond to ac-
1705 tuators, measurements, or other attributes. This information can be ob-
1706 tained from the preliminary analysis that follows the merging of datasets.
1707 The analysis, performed based on user's choice, provides indications on
1708 potential sensors, actuators, and other relevant information. These indica-
1709 tions help the user set the desired values in the *config.ini* file and refine the
1710 enrichment process by re-launching the *mergeDatasets.py* script.

1711 **Slope Calculation** The *slope* is an attribute that represents the **trend** of
1712 the measurement being considered. It is particularly useful, in our con-
1713 text, during the inference and invariant analysis phase to gather informa-
1714 tion about the trend under specific conditions. The slope can generally be
1715 classified as **increasing** ($slope > 0$), **decreasing** ($slope < 0$), or **stable** ($slope$
1716 = 0).

1717 Normally, the slope is calculated through a simple mathematical formula:
1718 given an interval a, b relative to the measurement l , the slope is given by
1719 the difference of these two values divided by the amount of time t that the
1720 measurement takes to reach b from a :

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

1721 In the proposed framework, similar to the previous tool, this time inter-
1722 val (the granularity) can be adjusted to be either long or short. The choice

1723 of granularity depends on the desired accuracy of the slope calculation. A
1724 lower granularity will provide a slope that closely reflects the actual mea-
1725 surement trend, while a higher granularity will result in flatter slope data.
1726 Each time interval within which the measurement is divided corresponds
1727 to a slope value. These slopes are calculated and added as additional at-
1728 tributes in the dataset. Later on, these slope values are used to determine
1729 the trend of the measurement in specific situations or conditions.

1730 Calculating the slope directly from the raw measurement data can be
1731 a suitable approach for systems where the measurements are not heavily
1732 influenced by **perturbations**. Perturbations, such as liquid oscillations in a
1733 tank during filling and emptying phases, can lead to fluctuating readings
1734 of the level. In such cases, maintaining a low granularity can provide a
1735 more accurate calculation of the overall trend that closely aligns with the
1736 actual measurement trend. The tanks of Ceccato et al.'s testbed are an ex-
1737 ample where this holds true.

1738 However, if perturbations significantly affect the measurement readings,
1739 calculating the slope on individual time intervals may result in an inaccu-
1740 rate trend definition, irrespective of the chosen granularity. In such cases,
1741 the fluctuating nature of the measurements due to perturbations can intro-
1742 duce errors in the slope calculation, making it less reliable as an indicator
1743 of the actual trend.

1744 Figure 4.3 demonstrates this assertion: the measurement, in blue, refers
1745 to the P1_LIT101 tank of the iTrust SWaT system; in red, the slope calcu-
1746 lation related to the measurement with three different granularities: 30
1747 (Figure 4.3a), 60 (Figure 4.3b) and 120 seconds (Figure 4.3c). It is notice-
1748 able that as the granularity increases, the slope values flatten. Moreover,
1749 in the time interval between seconds 1800 and 4200, the level of P1_LIT101
1750 exhibits a predominantly increasing trend, yet the calculated slope values
1751 fluctuate between positive and negative. Consequently, during the invari-
1752 ant analysis, the overall increasing trend may not be detected, resulting in
1753 a loss of information.

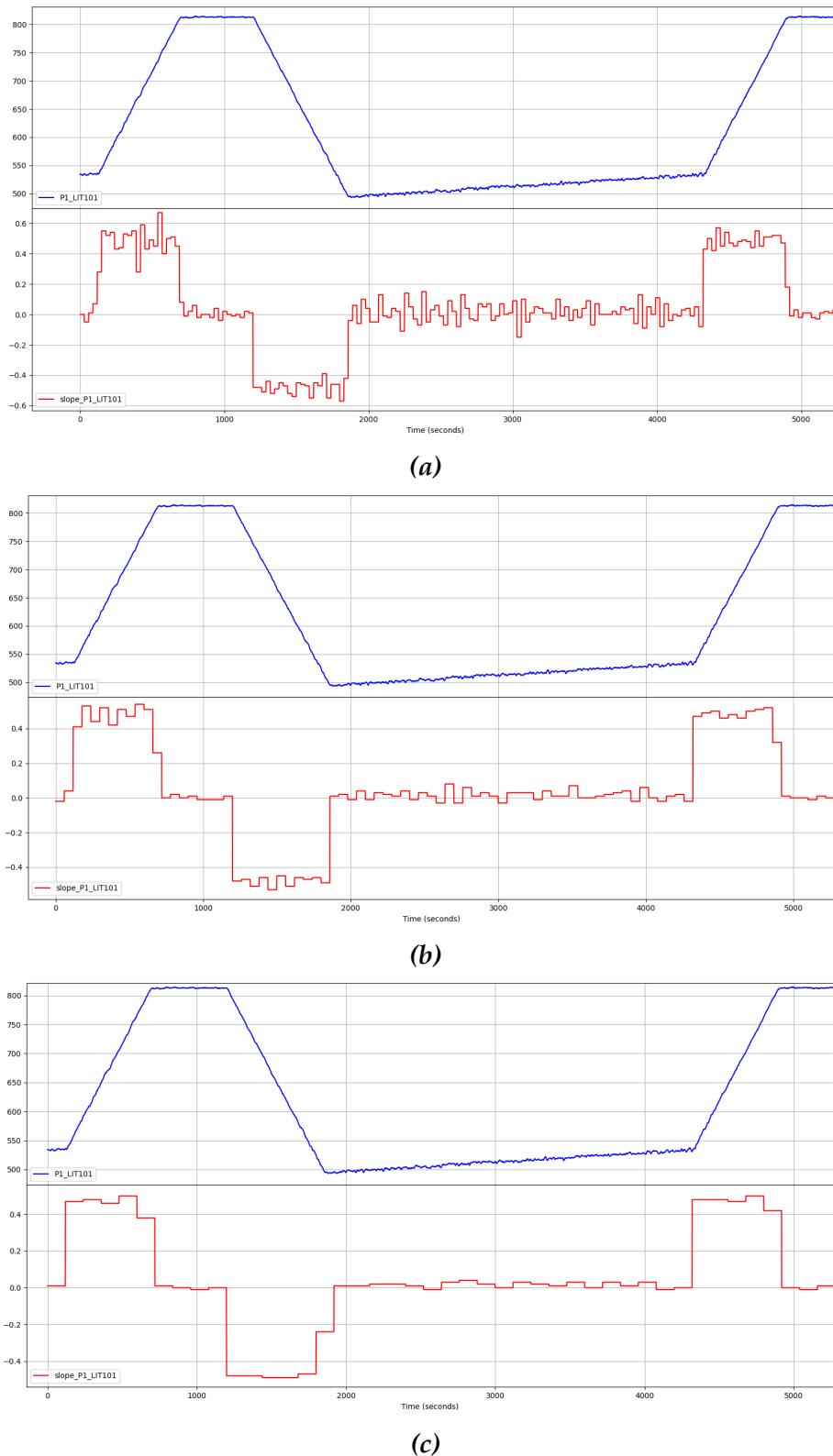


Figure 4.3: Slope comparison with granularity 30 (a), 60 (b) and 120 seconds (c)

1755 The previous tool did not take into account the possibility of having strongly
1756 perturbed data, which presented a challenge that we needed to address in
1757 the development of the proposed framework.

1758 The solution to this problem involves applying techniques to reduce
1759 the "noise" in the data, aiming to achieve a more linear trend in the mea-
1760 surement curve. By minimizing the effects of perturbations, we can calcu-
1761 late slopes more accurately.

1762 There are various methods available for smoothing out noise in the data.
1763 In our framework, we focused on two commonly used approaches found
1764 in the literature: **polynomial regression** and **seasonal decomposition**. In
1765 addition to these two methods, we also explored the use of a **line simpli-
1766 fication algorithm**.

1767
1768 *Polynomial regression* [58] is a technique that allows us to create a filter to
1769 reduce the impact of noise on the data. By fitting a polynomial function
1770 to the measurements, we can obtain a smoother curve that captures the
1771 underlying trend while minimizing the effects of perturbations.

1772 *Seasonal decomposition* [59], specifically the part related to trending, is an-
1773 other method we explored. It involves decomposing the time series into
1774 different components, such as trend, seasonality, and residual. By isolat-
1775 ing the trend component, we can obtain a cleaner representation of the
1776 underlying pattern in the data.

1777 *Line simplification algorithms* [60] aim to reduce the complexity of a polyline
1778 or curve by approximating it with a simplified version composed of fewer
1779 points. By selectively removing redundant or less significant points, line
1780 simplification algorithms help reduce storage space and computational re-
1781 quirements while preserving the overall shape and characteristics of the
1782 original line.

1783 Regarding polynomial regression, we evaluated the use of the **Savitzky-**
1784 **Golay filter** [61] as a smoothing technique. For seasonal decomposition,
1785 we explored the **Seasonal-Trend decomposition using LOESS (STL)** method
1786 [62]. For the line simplification algorithm, we specifically considered the

1787 **Ramer-Douglas-Peucker (RDP) algorithm [63].**

1788

1789 Figure 4.4 shows a quick graphical comparison of these techniques com-
 1790 pared with the original data. The solution adopted is the *STL decomposition*
 1791 method, which effectively reduces noise compared to the Savitzky-Golay
 1792 filter. However, it should be noted that this method may introduce some
 1793 delay in certain parts of the data, as is typically observed in similar algo-
 1794 rithms. Despite its apparent effectiveness, the RDP algorithm fails to ac-
 1795 curately approximate sections where the measurement level remains rela-
 1796 tively stable. Consequently, it yields incorrect slope estimations, causing a
 1797 loss of valuable information about the system.

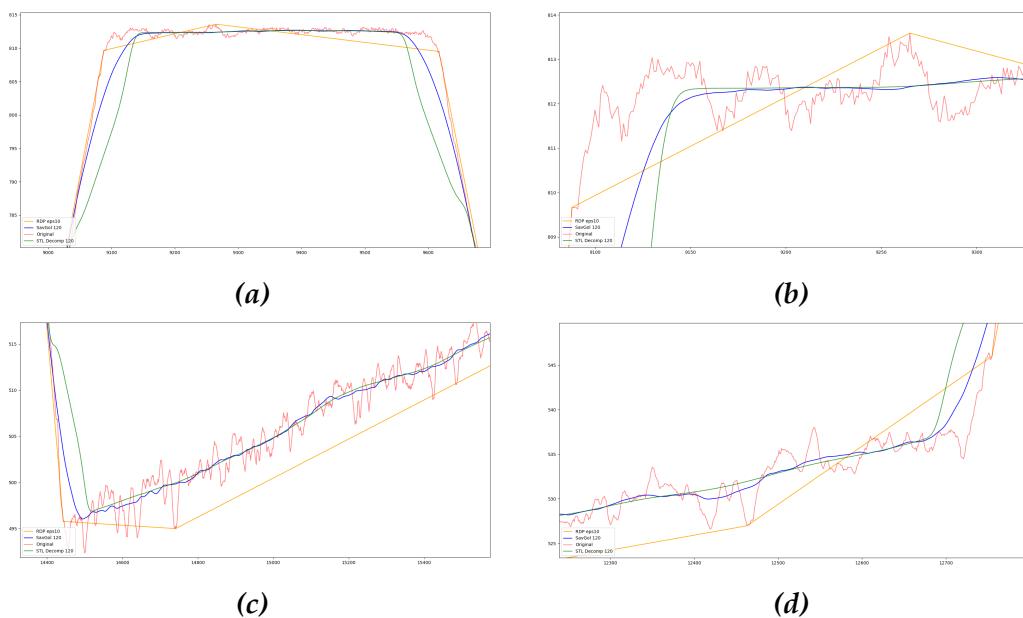


Figure 4.4: Savitzky-Golay filter (blue line), STL decomposition (green) and RDP algorithm (orange) comparison

1798

1799 By applying the STL decomposition, we observe a notable enhance-
 1800 ment in slope calculation even when using a low granularity. Figure 4.5
 1801 demonstrates that, with the same granularity as shown in Figure 4.3a, the
 1802 slope values, albeit exhibiting fluctuations, consistently align with the un-
 derlying trend of the data curve. The introduced lag resulting from the

1803 decomposition's periodicity is responsible for the observed delay.

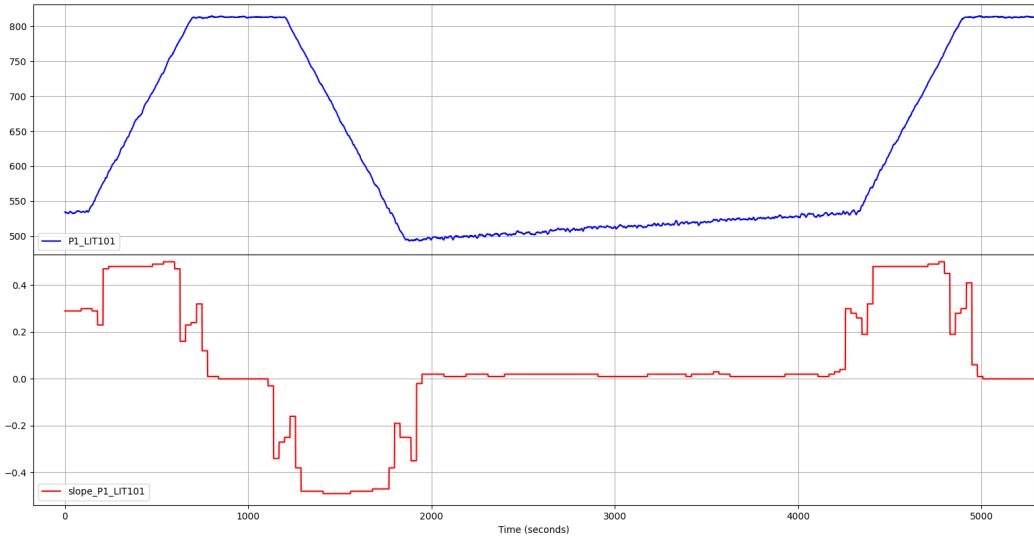


Figure 4.5: Slope after the application of the STL decomposition

1804 The periodicity, which defines the sampling time window for decom-
 1805 position and the level of noise smoothing, can be configured using the
 1806 `trend_period` directive in the `config.ini` file.
 1807 During the slope calculation, the analysis will be performed on the data
 1808 from the additional measurement trend attributes specified in the `trend_cols_list`
 1809 directive of the configuration file, rather than on the original unfiltered
 1810 data.

1811 To ensure proper interpretation by Daikon, the decimal values repre-
 1812 senting the calculated slopes are converted into **three numerical values**:
 1813 -1, 0, and 1. These values correspond to *decreasing* (if the slope is less than
 1814 zero), *stable* (if it is equal to zero), and *increasing* (if it is greater than zero)
 1815 trends, respectively. Figure 4.6 displays the modified slopes along with
 1816 the curve obtained from the STL decomposition:

1817 4.2.3.3 Datasets Merging

1818 During this step, the datasets of the individual PLCs are merged, re-
 1819 sulting in two separate datasets. The first dataset is enriched with addi-

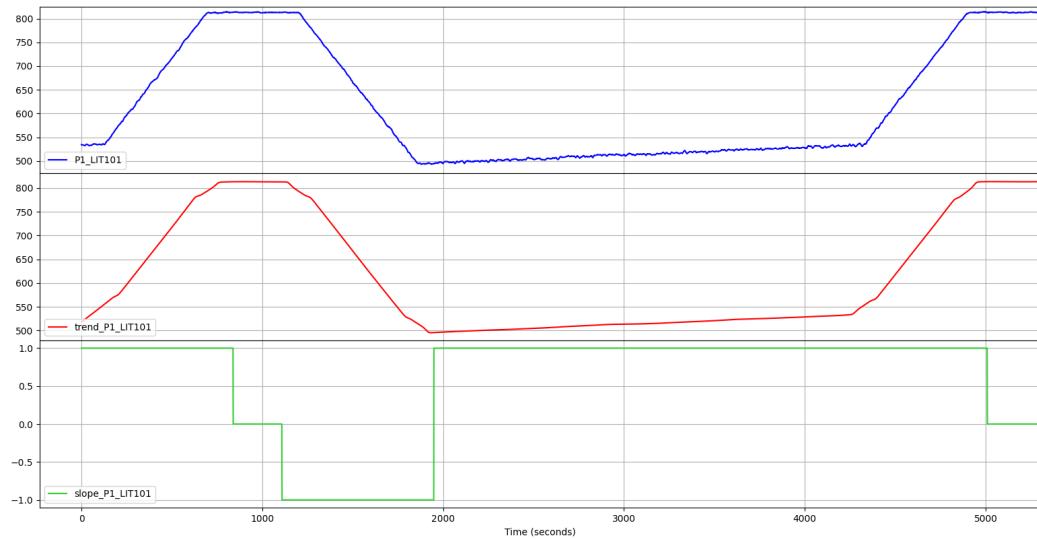


Figure 4.6: The new slope representation (green line) and the smoothed measurement data obtained with the STL decomposition (red)

1820 tional attributes but excludes the timestamp column. This dataset is in-
 1821 tended for inference and invariant analysis. The second dataset does not
 1822 contain any additional data and is specifically used in the process mining
 1823 phase.

1824 By default, the enriched dataset will be saved in CSV format in the
 1825 `$(project-dir)/daikon/Daikon_Invariants` directory. The other dataset,
 1826 without additional data, will be saved in the `$(project-dir)/process-mining/data`
 1827 directory. It's worth noting that both paths can be configured in the
 1828 `config.ini` file. The dataset name can be specified in the `config.ini` file or
 1829 through the `-o` command-line option. When generating the dataset for
 1830 process mining, the script will automatically add a `_TS` suffix to the file-
 1831 name to indicate that it includes the timestamp. This flexibility allows the
 1832 user to provide a different filename for each output, preventing overwrit-
 1833 ing of previous datasets. It enables the user to save the execution trace of
 1834 the selected subsystem separately and utilize them in subsequent analysis
 1835 phases.

1836 4.2.3.4 Preliminar Analysis of the Obtained Subsystem

1837 After merging the datasets, the user has the option to perform an **optional analysis** of the resulting dataset to extract preliminary data. This
 1838 analysis aims to gather basic information about the (sub)system and po-
 1839 tentially refine the enrichment process. If the user chooses to proceed with
 1840 the analysis, the `mergeDatasets.py` script invokes another Python script lo-
 1841 cated in the `$(project-dir)/pre-processing` directory called `system_info.py`.
 1842 Relying on an analysis based on a combination of Daikon and Pandas this
 1843 script performs a quick analysis of the dataset allowing to **estimate**, al-
 1844beit approximately, the **type of registers** (sensors, actuators, ...), also iden-
 1845tifying possible maximum and minimum values of measurements and
 1846 hardcoded setpoints. Furthermore, leveraging the use of the additional
 1847 attribute `prev_`, the `system_info.py` script is capable of deriving measure-
 1848ment values corresponding to state changes of individual actuators. This
 1849 allows for the identification of specific measurements associated with the
 1850 activation or deactivation of certain actuators within the system.
 1851 As the last information we have duration of actuator states for each cy-
 1852cle of the system: this information can be useful for making assumptions
 1853 and conjectures about the behavior of an actuator in a specific state or, by
 1854 observing the duration values of each cycle, highlighting anomalies in the
 1855 system.
 1856 Listing 4.4 shows an example of this brief analysis related to PLC1 of the
 1857 iTrust SWaT system (for brevity, only one measurement is reported in the
 1858 analysis of actuator state changes):

```
1860      Do you want to perform a brief analysis of the dataset? [y
1861      ↵ /n]: y
1862
1863      Actuators:
1864      P1_MV101 [0.0, 1.0, 2.0]
1865      P1_P101 [1.0, 2.0]
1866
1867      Sensors:
1868      P1_FIT101 {'max_lvl': 2.7, 'min_lvl': 0.0}
1869      P1_LIT101 {'max_lvl': 815.1, 'min_lvl': 489.6}
```

```
1870
1871     Hardcoded setpoints or spare actuators:
1872     P1_P102 [1.0]
1873
1874     Actuator state changes:
1875         P1_LIT101   P1_MV101   prev_P1_MV101
1876         669       800.7170    0           2
1877         1850      499.0203    0           1
1878         4876      800.5992    0           2
1879         6052      498.9026    0           1
1880         9071      800.7170    0           2
1881         10260     499.1381    0           1
1882         13268     801.3058    0           2
1883         14435     498.4315    0           1
1884         17423     801.4628    0           2
1885         18603     498.1567    0           1
1886
1887         P1_LIT101   P1_MV101   prev_P1_MV101
1888         677       805.0741    1           0
1889         4885      805.7414    1           0
1890         9079      805.7806    1           0
1891         13276     805.1133    1           0
1892         17432      804.4068   1           0
1893
1894         P1_LIT101   P1_MV101   prev_P1_MV101
1895         1858      495.4483    2           0
1896         6060      497.9998    2           0
1897         10269     495.9586    2           0
1898         14443     495.8016    2           0
1899         18611     494.5847    2           0
1900
1901         P1_LIT101   P1_P101   prev_P1_P101
1902         118       536.0356    1           2
1903         4322      533.3272    1           2
1904         8537      542.1591    1           2
1905         12721     534.8581    1           2
1906         16883     540.5890    1           2
1907
1908         P1_LIT101   P1_P101   prev_P1_P101
```

```

1909      1190    813.0031      2      1
1910      5395    813.0031      2      1
1911      9597    811.8256      2      1
1912     13776    812.7283      2      1
1913     17938    813.3171      2      1
1914
1915      Actuator state durations:
1916      P1_MV101 == 0.0
1917      9   9   10   9   9   10   9   9   10   9
1918
1919      P1_MV101 == 1.0
1920      1174   1168   1182   1160   1172
1921
1922      P1_MV101 == 2.0
1923      669   3019   3012   3000   2981
1924
1925      P1_P101 == 1.0
1926      1073   1074   1061   1056   1056
1927
1928      P1_P101 == 2.0
1929      118   3133   3143   3125   3108

```

Listing 4.4: Example of preliminar system analysis

1930 From these results we can draw the following conjectures:

- 1931 • the **probable actuators** are P1_MV101 and P1_P101. P1_MV101 has three
1932 states identified by the values 0, 1, and 2, suggesting it is a multi-
1933 state actuator. P1_P101 has two states identified by the values 1 and
1934 2, indicating a binary actuator;
- 1935 • there are **two probable measures**: P1_FIT101 and P1_LIT101. P1_FIT101
1936 has values ranging from 0 to 2.7, P1_LIT101 has values ranging from
1937 489.6 to 815.1. Based on this information, a further conjecture can be
1938 made that P1_LIT101 represents a tank level measurement;
- 1939 • apparently there is a probable **spare actuator**, P1_P102, whose value
1940 is always 1. No related *hardcoded setpoints* were found. From this
1941 information, another speculation can be made that the value 1 repre-

1942 sents the **OFF state** for binary actuators, while the value 2 represents
1943 the **ON state**;

- 1944 • from the analysis of state changes we can derive some **relative set-**
1945 **points**. For example, we observe that P1_P101 changes state from
1946 value 1 (OFF) to value 2 (ON) when the level of P1_LIT101 is ap-
1947 proximately 813, and it changes from value 2 (ON) to 1 (OFF) when
1948 the level of P1_LIT101 is around 535. We can deduce that P1_P101 is
1949 responsible for emptying the tank;
- 1950 • Regarding the actuators states duration, the very short duration of
1951 P1_MV101 in state 0 is observable: since we are unable, at this pre-
1952 liminary stage, to make assumptions about this, we will try to un-
1953 derstand the behavior of the system in this actuator state in the next
1954 stages of the analysis.

1955 The information obtained here can be used, as mentioned above, to
1956 refine the enrichment of the dataset by setting directives in the [DATASET]
1957 section of the *config.ini* file, should this be empty or only partially set, or to
1958 make the first conjectures about the system, as we have just seen.

1959 The *system_info.py* file can also run in standalone mode if needed:
1960 it takes as command-line arguments the dataset to be analyzed, a list of
1961 actuators, and a list of sensors. For analysis related to state changes, the
1962 dataset must mandatorily be of the enriched type.

1963 4.2.4 Phase 2: Graphs and Statistical Analysis

1964 The new *graph analysis* arises from the need to give the user an overview
1965 of the (sub)system obtained in the previous pre-processing phase, identi-
1966 fying more easily the typology of the registers and grasping more effec-
1967 tively the relationships and the dynamics that may exist between the reg-
1968 isters controlled by one or more PLCs, confirming the initial conjectures
1969 if the preliminary analysis described in the previous section has been per-
1970 formed, 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 4.7 helps to understand why.

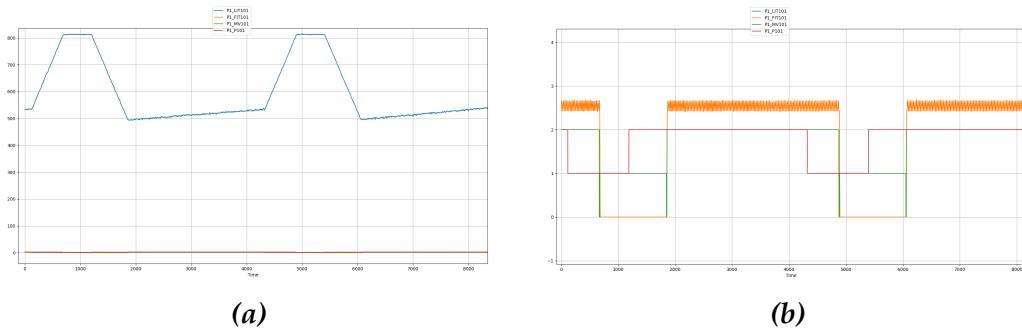
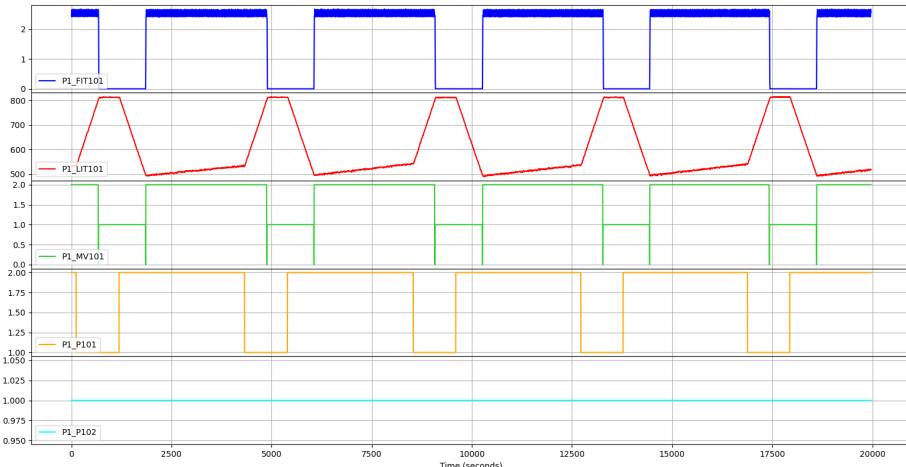


Figure 4.7: Plotting registers on the same y-axis

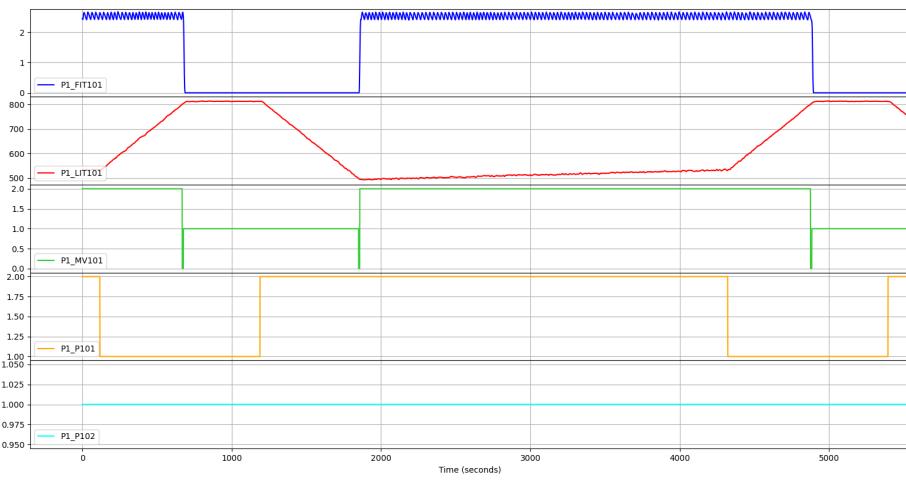
Figure 4.7a highlights the main issue with this approach, which is the use of the same y-axis for all the charts, representing the values of individual registers. When the range between register values is wide, it can result in some charts appearing as a single flat line or becoming indistinguishable, making them difficult to read. Additionally, as shown in Figure 4.7b, 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 4.8 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 4.8: Example of the new graph analysis

2000 To demonstrate the behavior and effectiveness of the new graph anal-
2001 ysis process, we observe in particular, Figure 4.8b: we can already have
2002 some validation on the conjectures made in the preliminar analysis from
2003 the previous step:

- 2004 • the data presented in the P1_FIT101 chart provides support for as-
2005 sociating this register with a **measurement**. Furthermore, the chart
2006 indicates a close correlation between the trend of this register and
2007 the periods in which P1_LIT101 exhibits an upward trend and the
2008 evolution of P1_MV101. The values of P1_FIT101, ranging from ap-
2009 proximately 0 to 2.5, are too limited to conclude that this register
2010 represents the level sensor of a tank. The overall trend also aligns
2011 with this interpretation, suggesting that it is more likely a **pressure**
2012 or **flow sensor**;
- 2013 • based on the information provided above, it appears that P1_LIT101
2014 indeed **represents the level sensor of a tank**, which was initially
2015 assumed in the preliminar analysis. The hypothesis is further rein-
2016 forced by several factors. Firstly, the wide range of values observed
2017 in this register supports the notion of it being a level sensor. Ad-
2018 ditionally, the graphical representation of the level trend provides
2019 valuable insights. It shows an initial sharp rise, followed by a period
2020 of stabilization referred to as a *plateau*, and subsequently, a similarly
2021 steep descent. Finally, there is a new phase of slower level growth
2022 compared to the previous one. These patterns lend further support
2023 to the interpretation of P1_LIT101 as a sensor specifically measuring
2024 the level of a tank;
- 2025 • at the start of the ascending trend of P1_LIT101, the value of P1_MV101
2026 is 2. Conversely, during the *plateau* phase observed when the mea-
2027 surement is approximately 800 and during the descending trend,
2028 P1_MV101 has a value of 1. These observations provide further confir-
2029 mation of the initial hypothesis: the value 1 represents the **OFF state**
2030 of the sensor, while 2 represents the **ON state**. Moreover, P1_MV101

2031 can be identified as the actuator responsible for causing the rising
2032 level in P1_LIT101;

- 2033 • during the preliminar analysis, we noticed the short duration of P1_MV101
2034 in state 0, but we were unable to speculate on the underlying rea-
2035 son. However, further analysis of the graph reveals that P1_MV101
2036 transitions and remains in the 0 state, acting as a type of "transient"
2037 between states 1 and 2. It can be inferred that this period represents
2038 the actual time required for the actuator to change its state.
- 2039 • we can observe the behavior of P1_P101 in relation to P1_LIT101 and
2040 P1_MV101 to understand its role. At the beginning of the descend-
2041 ing trend of P1_LIT101, after the plateau phase, P1_P101 assumes a
2042 value of 2. It then changes back to value 1 at the point of the (pre-
2043 sumed) tank's level increase. Although this fact alone may not be
2044 entirely clear, when comparing the behaviors of P1_P101, P1_LIT101,
2045 and P1_MV101, certain patterns emerge. When P1_P101 and P1_MV101
2046 are both in state 1, the water level remains stable. However, when
2047 P1_P101 changes to state 2, the level of the measurement drops rapidly.
2048 Subsequently, when P1_MV101 also transitions to state 2, the level
2049 slowly starts to rise again, with a sudden increase occurring when
2050 P1_P101 changes from 2 back to 1. From these observations, it can be
2051 inferred that P1_P101 serves as the **actuator responsible for empty-**
2052 **ing the presumed tank**. State 1 represents the **OFF state**, while state
2053 2 represents the **ON state** of this actuator.
- 2054 • it appears that P1_P102 plays no active role in the system as its state
2055 remains constant at 1 throughout. Considering this information, it
2056 seems unlikely that P1_P102 could serve as a relative setpoint. There-
2057 fore, according to the preliminar analysis, it can be inferred that P1_P102
2058 is most likely a **spare actuator** that is not actively utilized in the sys-
2059 tem. A *spare actuator* is indeed a secondary actuator that is intended
2060 to remains idle unless needed as a backup or replacement when the
2061 primary actuator is unavailable or needs to be taken out of service.

2062 As the reader has probably already noticed, the majority of the graphs
2063 presented in the previous sections and chapters were generated using the
2064 new graph analysis script, specifically the `runChartsSubPlots.py` script.
2065 This Python script is located in the `$(project-dir)/statistical-graphs`
2066 directory and utilizes the `matplotlib` libraries to generate the graph plots,
2067 similar to the previous tool.

2068 The script accepts the following command-line parameters:

- 2069 • **-f or --filename:** specifies the CVS format dataset to read data from.
2070 The dataset must be within the directory containing the enriched
2071 datasets for the invariant analysis phase. If subsequent parameters
2072 are not specified, the script will display all registers in the dataset,
2073 excluding any additional attributes;
- 2074 • **-r or --registers:** specifies one or more specific registers to be dis-
2075 played;
- 2076 • **-a or --addregisters:** adds one or more registers to the default visual-
2077 ization. This option is useful in case an additional attribute such as
2078 slope is to be analyzed;
- 2079 • **-e or --excluderegisters:** excludes one or more specific registers from
2080 the default visualization. This option is useful to avoid displaying
2081 hardcoded setpoints or spare registers.

2082 This script, like the previous ones, is designed to provide the maximum
2083 flexibility and ease of use for the user, combined with greater power and
2084 effectiveness in deriving useful information about the analyzed system.

2085 **Statistical Analysis** After careful consideration, we made the decision
2086 not to include the statistical analysis aspect of the previous tool in the
2087 framework. We found that there was no practical use for it. Instead, we
2088 integrated the relevant statistical information into the preliminary analysis
2089 conducted after the pre-processing phase. Additionally, we deemed the

2090 histogram to have limited utility and considered it outdated in comparison
2091 to the new graph analysis approach we implemented.

2092 Although we decided not to include the statistical analysis aspect in the
2093 framework, the Python script `histPlots_Stats.py` from the original tool
2094 remains in the directory. This script is essentially unchanged from the ver-
2095 sion developed by Ceccato et al. and can be used in the future if the need
2096 arises.

2097 4.2.5 Phase 3: Invariant Inference and Analysis

2098 The phase of invariant inference and analysis has undergone a redesign
2099 and improvement to offer the user a more comprehensive and easier ap-
2100 proach to identify invariants. This has been achieved through the applica-
2101 tion of new criteria to analyze and reorganize the Daikon analysis results.
2102 The outcome of this is a more compact presentation of information that
2103 highlights the possible relationships among invariants.

2104 The new design not only enables the identification of undiscovered aspects
2105 of the system behavior but also confirms the hypotheses made during the
2106 earlier stages of analysis. This step is now semi-automated, unlike before,
2107 and allows for the analysis of invariants on individual actuator states and
2108 their combinations.

2109 4.2.5.1 Revised Daikon Output

2110 To streamline the process of identifying invariants quickly and effi-
2111 ciently, it is necessary to revise the output generated by standard Daikon
2112 analysis. The goal is to create a more compact and readable format for the
2113 output.

2114
2115 The current Daikon results basically consist of three sections, referred as
2116 *program point sections* [64]:

- 2117 1. the first section containing generic invariants, i.e., valid regardless of
2118 whether a condition is specified for the analysis;

- 2119 2. the second section containing invariants obtained by specifying a
 2120 condition for the analysis in the *.spinfo* file, if any;
- 2121 3. a third section containing the invariants that are obtained from the
 2122 negation of the condition potentially specified in the *.spinfo* file.

2123 In each section only a single invariant per row is shown, without relating
 2124 it in any way to the others: this makes it difficult to identify significant
 2125 invariants and any invariant chain that might provide much more information
 2126 about the behavior of the system than the single invariant.

2127 A brief example of the structure and format of this output related to PLC1
 2128 of the iTrust SWaT system is shown in Listing 4.5, where a condition was
 2129 specified on the measurement P1_LIT101 and on actuator P1_MV101:

```

2130     aprogram.point:::POINT
2131     P1_P102 == prev_P1_P102
2132     P1_FIT101 >= 0.0
2133     P1_MV101 one of { 0.0, 1.0, 2.0 }
2134     P1_P101 one of { 1.0, 2.0 }
2135     P1_P102 == 1.0
2136     max_P1_LIT101 == 816.0
2137     min_P1_LIT101 == 489.0
2138     slope_P1_LIT101 one of { -1.0, 0.0, 1.0 }
2139     [...]
2140     P1_LIT101 > P1_MV101
2141     P1_LIT101 > P1_P101
2142     P1_LIT101 > P1_P102
2143     P1_LIT101 < max_P1_LIT101
2144     P1_LIT101 > min_P1_LIT101
2145     [...]
2146     P1_MV101 < min_P1_LIT101
2147     P1_MV101 < trend_P1_LIT101
2148     P1_P101 >= P1_P102
2149     P1_P101 < max_P1_LIT101
2150     [...]
2151     =====
2152     aprogram.point:::POINT; condition="P1_MV101 == 2.0 &&
2153         ↳ P1_LIT101 < max_P1_LIT101 - 16 && P1_LIT101 >
2154         ↳ min_P1_LIT101 + 15"

```

```

2155     P1_MV101 == prev_P1_MV101
2156     P1_P102 == slope_P1_LIT101
2157     P1_MV101 == 2.0
2158     P1_FIT101 > P1_MV101
2159     P1_FIT101 > P1_P101
2160     P1_FIT101 > P1_P102
2161     P1_FIT101 > prev_P1_P101
2162     P1_MV101 >= P1_P101
2163     P1_MV101 >= prev_P1_P101
2164     P1_P101 <= prev_P1_P101
2165     =====
2166    aprogram.point:::POINT;condition="not(P1_MV101 == 2.0 &&
2167     ↪ P1_LIT101 < max_P1_LIT101 - 16 && P1_LIT101 >
2168     ↪ min_P1_LIT101 + 15)"
2169     P1_P101 >= prev_P1_P101
2170     Exiting Daikon.

```

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

2171 In the presented framework, the output is simplified to **two sections**:
2172 the *general section* and the section related to the *user-specified condition*. The
2173 section related to the negated condition is eliminated as it is not relevant in
2174 this context and could lead to potential misinterpretation. Moreover, the
2175 relationships between invariants will be emphasized by utilizing **transi-**
2176 **tive closures**: transitive closure of a relation R is another relation, typically
2177 denoted R^+ that adds to R all those elements that, while not necessarily
2178 related directly to each other, can be reached by a *chain* of elements related
2179 to each other. In other words, the transitive closure of R is the smallest (in
2180 set theory sense) transitive relation R such that $R \subset R^+$ [65].

2181 To implement the transitive closure of the invariants generated by Daikon,
2182 we initially categorized the invariants in each section by type based on
2183 their mathematical relation ($==, >, <, >=, <=, !=$), excluding any in-
2184 variant related to additional attributes except for the slope. For each type
2185 of invariant, we constructed a **graph** using the NetworkX library, where
2186 registers were represented as nodes, and arcs were created to connect reg-
2187 isters that shared a common endpoint in the invariant, applying the transi-

tive 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 4.9 shows some examples of these graphs:

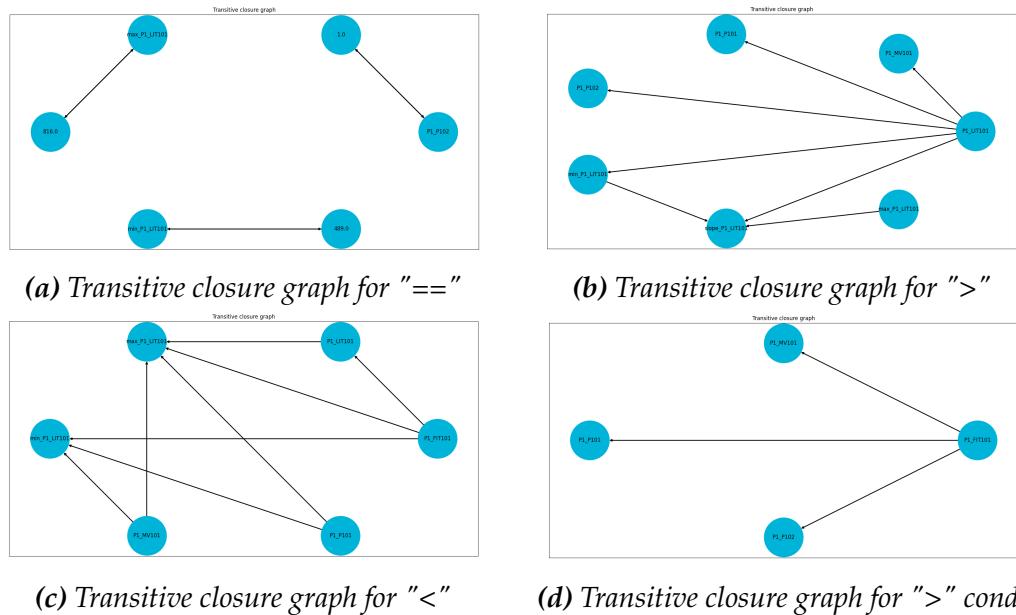


Figure 4.9: Example of transitive closure graphs for invariants in PLC1 of iTrust SWaT system

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:

```

2196 1 =====
2197 2 Generic
2198 3 =====
2199 4 P1_MV101 one of { 0.0, 1.0, 2.0 }
2200 5 P1_P101 one of { 1.0, 2.0 }
2201 6 slope_P1_LIT101 one of { -1.0, 0.0, 1.0 }
2202 7 P1_FIT101 != P1_P101, P1_P102
2203 8 P1_P102 == 1.0
2204 9 max_P1_LIT101 == 816.0
2205 10 min_P1_LIT101 == 489.0
  
```

```

2206 11 P1_LIT101 > P1_MV101
2207 12 P1_LIT101 > P1_P101
2208 13 P1_LIT101 > P1_P102
2209 14 P1_LIT101 > min_P1_LIT101 > slope_P1_LIT101
2210 15 P1_FIT101 >= 0.0
2211 16 P1_P101 >= P1_P102 >= slope_P1_LIT101
2212 17 =====
2213 18 =====
2214 19 P1_MV101 == 2.0 && P1_LIT101 < max_P1_LIT101 - 16 &&
2215   ↪ P1_LIT101 > min_P1_LIT101 + 15
2216 20 =====
2217 21 slope_P1_LIT101 == P1_P102
2218 22 P1_MV101 == 2.0
2219 23 P1_FIT101 > P1_MV101
2220 24 P1_FIT101 > P1_P101
2221 25 P1_FIT101 > P1_P102
2222 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.

4.2.5.2 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

2238 account the interplay and combined effects of multiple actuators on the
2239 selected measurement. By incorporating the real-time states of the actu-
2240 ators, this semi-automated analysis offers a more accurate assessment of
2241 the system behavior.

2242 These two types of analysis will be handled by the Python script
2243 `daikonAnalysis.py`, contained in the default directory `$(project_dir)/daikon`.
2244 The script accepts the following command-line arguments:

- 2245 • **-f or --filename**: specifies the enriched dataset, in CSV format, from
2246 which to read the data. The dataset must be located within the di-
2247 rectory containing the enriched datasets specified in the `config.ini` file
2248 (by default `$(project_dir)/daikon/Daikon_Invariants`);
- 2249 • **-s or --simpleanalysis**: performs the analysis on the states of indi-
2250 vidual actuators;
- 2251 • **-c or --customanalysis**: performs the analysis on combinations of ac-
2252 tual states of the actuators;
- 2253 • **-u or --uppermargin**: defines a percentage margin on the maximum
2254 value of the measurement;
- 2255 • **-l or --lowermargin**: defines a percentage margin on the minimum
2256 value of the measurement;

2257 The selection of one or both types of analysis is possible. Additionally,
2258 the last two parameters can be used to set a condition on the value of the
2259 measurement. This condition is designed to bypass the transient periods
2260 that occur during actuator state changes and the actual trend changes at
2261 the maximum and minimum values of the measurement.

2262 This approach proves particularly beneficial for the first type of analysis,
2263 as it enables more accurate data on the trends of the measurement. By
2264 excluding the transient periods, the analysis can focus on the stable and
2265 meaningful trends, providing improved insights into the behavior of the
2266 system.

2267 **Analysis on single actuator states** Analysis on the states of individual
 2268 actuators is the simplest: after the user is prompted to input the measure-
 2269 ment, chosen from a list of likely available measurements, the script rec-
 2270ognizes the likely actuators and the relative states of each, using the same
 2271 mixed Daikon/Pandas technique adopted in the preliminary analysis dur-
 2272 ing the pre-processing phase.
 2273 For each actuator and each state it assumes, a single Daikon analysis is
 2274 performed, eventually placing the condition on the maximum and mini-
 2275 mum level of the measurement.
 2276 The result of these analyses are saved in the form of text files in a directory
 2277 having the name corresponding to the analyzed actuator and contained in
 2278 the default parent directory `$(project_dir)/daikon/Daikon_Invariants/results`:
 2279 each file generated by the analysis is identified by the name of the actuator,
 2280 the state and the condition, if any, on the measurement (see Figure 4.10).

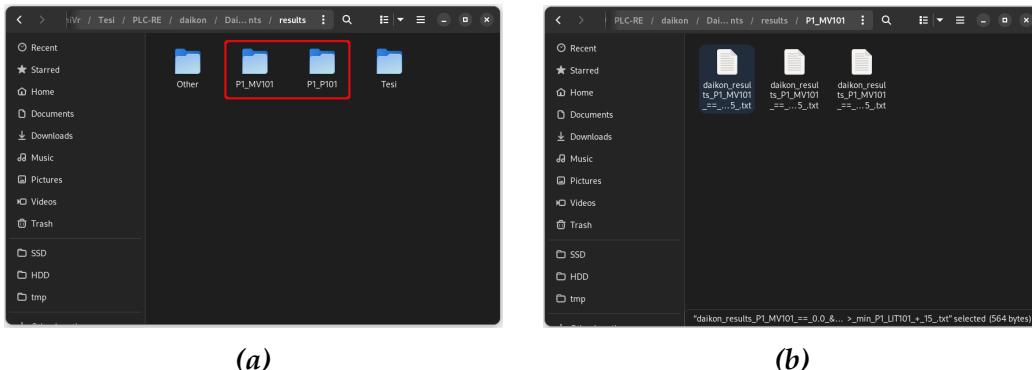


Figure 4.10: Directory (a) and outcome files (b) for the single actuator states analysis

2281 Listing 4.6 provides an illustrative example of the analysis results. In
 2282 this case, we focus on the actuator `P1_MV101` in state 2. By examining the
 2283 generic invariants, we can deduce that `P1_P101` is a probable actuator with
 2284 binary values, as indicated in line 5. However, upon closer inspection
 2285 in line 8, we discover that `P1_P102` consistently maintains a value of 1
 2286 throughout, thus confirming its role as a spare actuator rather than a set-
 2287 point. Furthermore, lines 9 and 10 present the maximum and minimum
 2288 values achieved by `P1_LIT101`, the register presumed to be connected to

2289 the tank.

2290 The condition-generated invariants provide the most interesting insights.

2291 In particular, at line 21, we observe that when P1_MV101 is set to 2, the
2292 slope of P1_LIT101 is 1, indicating an increasing trend. This confirms our
2293 previous assumption that P1_MV101 represents the actuator's ON state and
2294 is responsible for filling the tank (P1_LIT101).

2295 Furthermore, at line 23, we discover that P1_FIT101 is greater than P1_MV101.
2296 This implies that when the actuator assumes the value 2, the sensor P1_FIT101
2297 registers a measurement. In contrast, when P1_MV101 is 1, P1_FIT101 re-
2298 mains at 0, signifying no recorded readings. This finding reinforces our
2299 initial hypothesis that P1_FIT101 may serve as a pressure or flow sensor.

2300 **Analysis of the Current System Configuration** The analysis of the cur-
2301 rent system configuration based on the actual states of the actuators is
2302 more complex, but at the same time offers more interesting outcomes as
2303 it provides better evidence of actuator behavior in relation to the selected
2304 measurement.

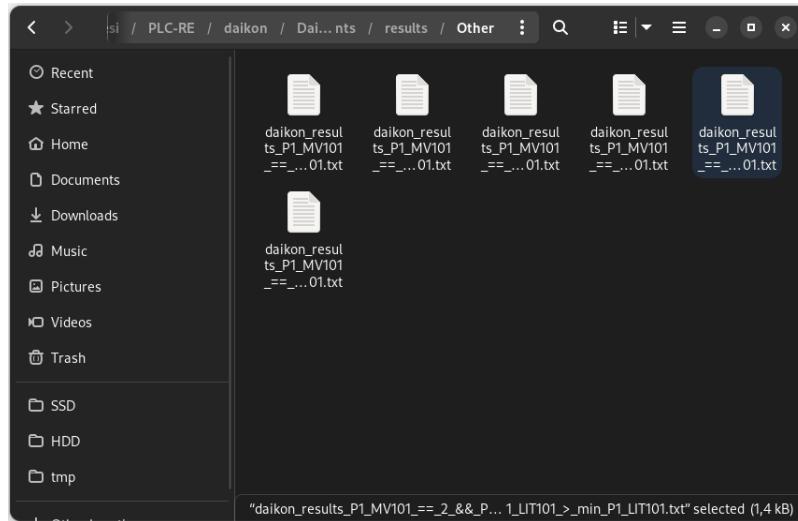


Figure 4.11: Daikon outcome files for system configuration analysis. Each file represents a single system state

2305 For this analysis, the script automatically identifies the configurations
2306 of the actuators that represent different system states (e.g., P1_MV101 ==

2307 2, P1_P101 == 1). Daikon analysis is then performed for each of these
 2308 configurations. The user is prompted to select the measurement attribute
 2309 and, if desired, specific actuators for studying their configurations. If no
 2310 actuators are selected, all previously detected actuators will be considered.
 2311 The analysis results are saved in text format within a designated directory,
 2312 located alongside the previous analysis outputs. The filenames of these
 2313 result files follow a specific naming convention based on the analysis rule
 2314 applied (see Figure 4.11).

2315

2316 An example of the obtained outcomes can be seen in Listing 4.7:

```

2317 1 =====
2318 2 Generic
2319 3 =====
2320 4 P1_MV101 <= P1_P101 ==> P1_FIT101 >= 0.0
2321 5 P1_MV101 <= P1_P101 ==> P1_MV101 one of { 0.0, 1.0, 2.0
2322 6   ↪ }
2323 7 P1_MV101 <= P1_P101 ==> P1_P101 one of { 1.0, 2.0 }
2324 8 P1_MV101 <= P1_P101 ==> slope_P1_LIT101 one of { -1.0,
2325 9   ↪ 0.0, 1.0 }
2326 10 P1_MV101 > P1_P101 ==> P1_FIT101 > P1_MV101
2327 11 P1_MV101 > P1_P101 ==> P1_FIT101 > P1_P101
2328 12 P1_MV101 > P1_P101 ==> P1_FIT101 > P1_P102
2329 13 P1_MV101 > P1_P101 ==> P1_FIT101 > slope_P1_LIT101
2330 14 P1_MV101 > P1_P101 ==> P1_MV101 == 2.0
2331 15 P1_MV101 > P1_P101 ==> P1_MV101 > P1_P102
2332 16 P1_MV101 > P1_P101 ==> P1_MV101 > slope_P1_LIT101
2333 17 P1_MV101 > P1_P101 ==> P1_P101 == 1.0
2334 18 P1_MV101 > P1_P101 ==> P1_P101 == P1_P102
2335 19 P1_MV101 > P1_P101 ==> P1_P101 == slope_P1_LIT101
2336 20 P1_MV101 > P1_P101 ==> slope_P1_LIT101 == 1.0
2337 21 [...]
2338 22 =====
2339 23 P1_MV101 == 2 && P1_P101 == 1 && P1_LIT101 < max_P1_LIT101
2340 24   ↪ && P1_LIT101 > min_P1_LIT101
2341 25 =====
2342 26 slope_P1_LIT101 == P1_P102 == P1_P101 == 1.0

```

```

2344 25 P1_MV101 == 2.0
2345 26 P1_FIT101 > P1_MV101
2346 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 $P1_MV101$ is 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 $P1_FIT101$ is 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-

2374 gations of the system.

2375 The script, located in the default directory \$(project_dir)/daikon, can be
2376 executed with three command-line parameters:

- 2377 • **-f or --filename:** specifies the CSV format *enriched* dataset to read
2378 data from. Even in this case, the dataset must be located within the
2379 directory containing the enriched datasets;
- 2380 • **-c or --condition:** specifies the condition for the analysis, which will
2381 be automatically overwritten in the *.sinfo* file. It is possible to spec-
2382 ify more than one condition, but it is strongly recommended to use
2383 the logical operator && to avoid undesired behaviors in Daikon out-
2384 comes;
- 2385 • **-r or --register:** specifies the directory where to save the text file with
2386 the outcomes.

2387 During the execution of this analysis, a single output file is generated, con-
2388 taining the discovered invariants. The user can specify the directory where
2389 this file should be saved using the -r command-line option. By default, the
2390 file is stored in the directory \$(project_dir)/daikon/Daikon_Invariants/results.
2391 The output file serves as a record of the identified invariants and can be
2392 examined at a later time.

2393 In conclusion, the integration of these two analysis types, along with
2394 the ability to conduct more refined analysis in the future and the enhanced
2395 output format of Daikon, significantly enhances the completeness, clarity,
2396 and effectiveness of this stage compared to the previous framework.

2397 4.2.6 Phase 4: Business Process Analysis

2398 We have made significant revisions to the Business Process Analysis
2399 we are presenting. Instead of relying on the previous Java solution and
2400 proprietary process mining software Disco, we have adopted a **new inte-**
2401 **grated solution** created in Python from scratch. The new solution utilizes

2402 the Graphviz libraries to generate the corresponding activity diagram.

2403

2404 In this updated Business Process, greater emphasis is placed on process
2405 mining related to the physical system. Our goal is to extract as much
2406 information as possible from the dataset, enabling us to promptly visu-
2407 alize the system's behavior and its various states. This approach allows
2408 us to validate the conjectures and hypotheses formulated in the previ-
2409 ous phases, and potentially uncover hidden patterns that were previously
2410 undisclosed.

2411 On the other hand, the aspect related to network communications was
2412 reconsidered to enable operation with multiple protocols, expanding be-
2413 yond the limitations of Modbus. Unfortunately, despite our intentions, we
2414 were unable to implement this modification due to reasons that will be
2415 elaborated on later.

2416

2417 Now, let's examine the key aspects of this new phase in greater detail.

2418 **4.2.6.1 Process Mining of the Physical Process**

2419 The mining of the physical process is performed by the Python script
2420 called `processMining.py`, located in the default directory `$(project_dir)/process-mining`.
2421 This script accepts the following parameters from the command line:

- 2422 • **-f or --filename:** specifies the CSV format *timestamped* dataset to be
2423 mined from. The dataset is obtained from the pre-processing stage
2424 and is located in the default directory `$(project_dir)/process-mining/data`;
- 2425 • **-a or --actuators:** specifies one ore more actuators whose combina-
2426 tions of states are to be analyzed. If this parameter is not provided,
2427 all actuators in the subsystem will be considered;
- 2428 • **-s or --sensors:** specifies one or more measurements for which the
2429 trend will be calculated based on actuator state changes. If this pa-
2430 rameter is omitted, all available measurements will be considered;

- 2431 • **-t or --tolerance:** specifies the tolerance to be taken into account dur-
2432 ing the trend calculation;
- 2433 • **-g or --graph:** shows the resulting activity diagram.

2434 The script processes the dataset in a sequential manner, analyzing each
2435 actuator state change. It calculates the duration in seconds of the system
2436 state, as well as the trend and slope of the specified measurement(s). Addi-
2437 tionally, the script stores the next system state and the measurement values
2438 corresponding to the state change points, allowing for the identification of
2439 relative setpoints. These results are gradually collected and stored in a dic-
2440 tionary, where the keys represent the system states based on the actuator
2441 configurations, and the values represent the measured values mentioned
2442 above. The dictionary is then saved as a JSON file, which can be accessed
2443 by the user in the `$(project_dir)/process-mining/data` directory. The
2444 name of the file can be specified in the configuration file `config.ini`.

2445
2446 An example of the JSON file obtained in this step is shown in Listing 4.8:
2447 the JSON file showcases the structure of the data obtained during the pro-
2448 cess.

```
2449 {  
2450     "P1_MV101 == 2, P1_P101 == 2": {  
2451         "start_value_P1_LIT101": [  
2452             534.9366,  
2453             495.4483,  
2454             497.9998,  
2455             495.9586,  
2456             495.8016  
2457         ],  
2458         "end_value_P1_LIT101": [  
2459             536.0356,  
2460             532.7384,  
2461             541.7273,  
2462             534.4656,  
2463             540.8245  
2464         ],  
2465     }  
2466 }
```

```

2465     "slope_P1_LIT101": [
2466         0,
2467         0.015,
2468         0.018,
2469         0.016,
2470         0.018
2471     ],
2472     "trend_P1_LIT101": [
2473         "STBL",
2474         "ASC",
2475         "ASC",
2476         "ASC",
2477         "ASC"
2478     ],
2479     "time": [
2480         119,
2481         2464,
2482         2477,
2483         2452,
2484         2440
2485     ],
2486     "next_state": [
2487         "P1_MV101 == 2, P1_P101 == 1",
2488         "P1_MV101 == 2, P1_P101 == 1",
2489         "P1_MV101 == 2, P1_P101 == 1",
2490         "P1_MV101 == 2, P1_P101 == 1",
2491         "P1_MV101 == 2, P1_P101 == 1"
2492     ]
2493 },
2494 "P1_MV101 == 2, P1_P101 == 1": {
2495     ...
2496 }
2497 "P1_MV101 == 0, P1_P101 == 1": {
2498     ...
2499 }
2500 ...
2501 }
2502 }
```

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

2503 The collected data is now utilized to generate the **system activity dia-**
2504 **gram.** This diagram represents an *oriented* graph where the nodes corre-
2505 spond to the system states determined by the actuator configurations. In
2506 addition to the state information, the nodes also display the trend (ascend-
2507 ing, descending, or stable) and the slope of the reference measurement.
2508 The edges in the diagram depict the values of the measurements at the
2509 time of the state change. These measurement values represent the set-
2510 points for each measurement. Setpoints are calculated on the average of
2511 the values measured for that specific state. Additionally, the diagram in-
2512 corporates on the edges the average duration of each system state.
2513 Each data point on the arcs is accompanied by a *standard deviation value*.
2514 This value provides information about the occurrence of a specific state
2515 within the system's cycle, indicating whether it appears multiple times
2516 with varying values and time durations. A low standard deviation sug-
2517 gests that the state is likely to occur only once within each cycle. Con-
2518 versely, a high standard deviation indicates that the state may occur mul-
2519 tiple times within the cycle.

2520 An example of the activity diagram generated by the processMining.py
2521 script is presented in Figure 4.12. This diagram illustrates the system's
2522 behavior by depicting transitions between different states. Nodes in the
2523 diagram represent specific system states, while arrows or edges indicate
2524 the flow between these states.

2525 The diagram reveals important aspects of the system, such as its cyclicity
2526 and the emphasis on the temporal sequence of actuator states. These in-
2527 sights might not have been clearly evident in earlier stages of the analysis.
2528 The diagram provides a visual representation that allows us to appreciate
2529 the recurring patterns and understand how the system evolves over time.
2530 By observing the transitions and relationships between different states, we
2531 gain a better understanding of the dynamics and behavior of the system.

2532 It is important to acknowledge that despite considering the tolerance
2533 set for trend and slope calculation, the accuracy of the related data may
2534 vary. For instance, there could be instances where multiple trends exist

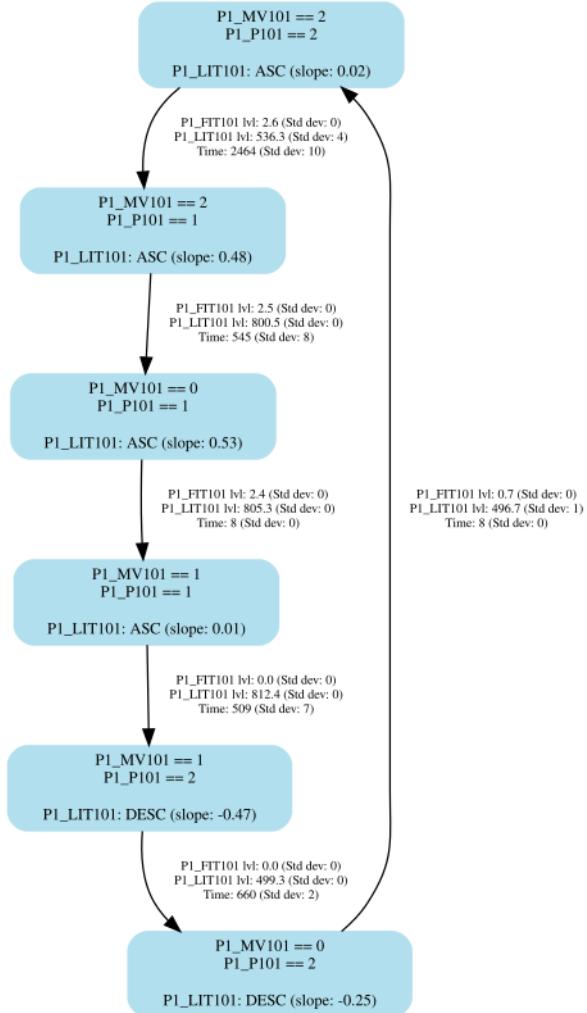


Figure 4.12: Activity diagram for PLC1 of the iTrust SWaT system

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 4.2.3.2. 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.

2544 **4.2.6.2 Network Communications**

2545 The incorporation of network communications into Business Process
2546 Analysis has been reconsidered to shift away from a *single-protocol* solu-
2547 tion based on Modbus. Instead, the focus is on adopting a solution that
2548 can handle **multiple protocols**, even at the same time.

2549 The main concept was to develop a new Python script that would extract
2550 and process data from PCAP files obtained through network traffic sniff-
2551 ing. The intention was to export this data to a CSV file, which would then
2552 be used as input for the process mining script, `processMining.py`, and inte-
2553 grated with the physical system data to derive commands to the actuators.

2554 The analysis of the network data revealed that different protocols ex-
2555 hibit distinct behaviors when commanding changes in actuator states. Con-
2556 sequently, the previous approach proposed by Ceccato et al. becomes **im-**
2557 **practical** when attempting to detect system state changes through network
2558 commands sent to the actuators.

2559 However, considering that system state changes have already been iden-
2560 tified through the process mining of the physical process, and with the
2561 corresponding event timestamps available, we propose an alternative ap-
2562 proach to Ceccato et al.'s method. Instead of seeking the correspondence
2563 between network events and physical process events at the same instant,
2564 we suggest reversing the perspective. By focusing on the correspondence
2565 between a given event occurring in the physical process at a specific mo-
2566 ment and the corresponding event in the network data at the same mo-
2567 ment, it should be possible to achieve a similar, if not superior, outcome
2568 compared to the previous solution.

2569 This *inverse approach* aims to leverage the existing knowledge of system
2570 state changes obtained through the mining of the physical process. By
2571 aligning these physical events with their corresponding events in the net-
2572 work data, a more effective analysis can be conducted.

Case study: the iTrust SWaT System

2573 HAVING introduced the innovative framework and highlighted its po-
2574 tential in the preceding chapter, we now turn our attention to the
2575 case study where we will apply this framework. As previously mentioned
2576 in Chapter 4 and demonstrated through various examples in the same
2577 chapter, our focus will be on the **iTrust SWaT system** [36], developed by
2578 the iTrust – Center for Research in Cyber Security of University of Singa-
2579 pore for Technology and Design [30]. The acronym SWaT represents *Secure*
2580 *Water Treatment*.

2581 The iTrust SWaT system is a testbed that replicates on a small scale
2582 a real water treatment plant arises to support research in the area of cy-
2583 ber security of industrial control systems and has been operational since
2584 March 2015: it is still being used by students at the University of Singapore
2585 for educational and training purposes and is available to organizations to
2586 train their operators on cyber physical incidents.

2587 5.1 Architecture

2588 In contrast to the virtualized testbed discussed in Section 3.2.1 by Cec-
2589 cato et al., the iTrust SWaT system is composed entirely of physical hard-
2590 ware components. It encompasses various elements, starting from field

2591 devices and extending to PLCs, HMI, SCADA workstations, and the SCADA
2592 server (also referred to as the *historian*). The historian is responsible for
2593 recording data from field devices for further analysis. In the upcoming
2594 sections, we will delve deeper into the architecture of the physical process
2595 and the communication network.

2596 **5.1.1 Physical Process**

2597 The physical process of the SWaT consists of six stages, denoted P1
2598 through P6. These stages are [66][67]:

- 2599 P1. **taking in raw water:** feeds unfiltered water into the system
- 2600 P2. **chemical dosing:** adds chemicals to water useful for initial pretreat-
2601 ment;
- 2602 P3. **Ultra Filtration (UF) system:** the water is filtered through a semi-
2603 permeable membrane (ultrafiltration membrane) using the liquid pres-
2604 sure, effectively capturing impurities and suspended solids, as well
2605 as removing bacteria, viruses, and other pathogens present in the
2606 water;
- 2607 P4. **dechlorination:** removes residual chlorine from disinfected water
2608 using ultraviolet lamps;
- 2609 P5. **Reverse Osmosis (RO):** performs further filtration of the water;
- 2610 P6. **backwash process:** cleans the membranes in UF using the water pro-
2611 duced by RO.

2612 Figure 5.1 shows a graphical representation of the architecture and the six
2613 stages of the SWaT system.

2614 The SWaT system incorporates an array of sensors that play a crucial
2615 role in monitoring the system's operations and ensuring their safe. These
2616 sensors are responsible for continuously collecting data and providing
2617 valuable insights into the functioning of the system. These sensors are:

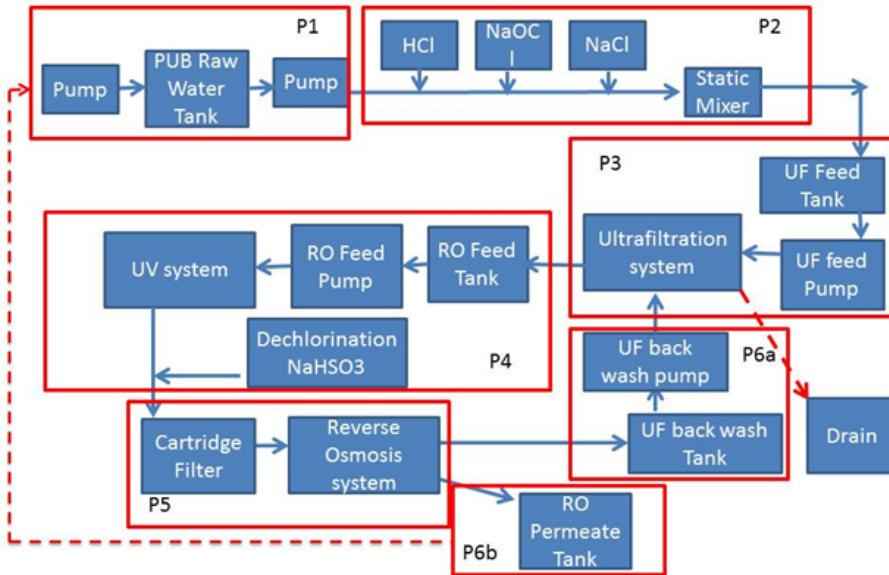


Figure 5.1: SWaT architecture

- 2618 • Level Indication Transmitter (measured in mm)
- 2619 • Flow Indication Transmitter (m³/hr)
- 2620 • Analyser Indicator Transmitter
 - 2621 o Conductivity ($\mu\text{S}/\text{cm}$)
 - 2622 o pH
 - 2623 o Oxidation Reduction Potential (mV)
- 2624 • Differential Pressure Indicator Transmitter (kPa)
- 2625 • Pressure Indicator Transmitter (kPa)

2626 The sensors and actuators associated with each PLC are shown in Figure
2627 5.2.

2628 Sensors and actuators are mapped to tags by the communication protocol
2629 used (see 5.1.2): a tag can be addressed via string descriptor defined by
2630 the system designer (e.g. MV101, to indicate motorized valve number 1 at
2631 stage 1) or by referring directly to the analog/digital pins of the PLC I/O
2632 unit [67].

Raw Water	Pre-Treatment	Ultra-Filtration	De-Chlorination	Reverse Osmosis	RO Product
P-101 Stopped	P-201 Stopped	P-301 Stopped	P-401 Stopped	P-501 Stopped	P-601 Stopped
P-102 Stopped	P-202 Stopped	P-302 Stopped	P-402 Stopped	P-502 Stopped	P-602 Stopped
MV-101 Closed	P-203 Stopped	MV-301 Closed	P-403 Stopped	MV-501 Closed	P-603 Stopped
LIT-101 520 mm	P-204 Stopped	MV-302 Closed	P-404 Stopped	MV-502 Closed	LS-601 Normal
FIT-101 0.00 m³/h LL	P-205 Stopped	MV-303 Closed	UV-401 Stopped	MV-503 Closed	LS-602 HIGH
FIT-201 0.00 m³/h LL	P-206 Stopped	MV-304 Closed	LS-401 Normal	MV-504 Closed	LS-603 LOW
	P-207 Stopped	PSH-301 Normal	LIT-401 1008 mm H	PSL-501 Normal	FIT-601 0.00 m³/h LL
	P-208 Stopped	DPSH-301 Normal	FIT-401 0.00 m³/h LL	PSH-501 Normal	
	MV-201 Closed	LIT-301 1012 mm H	AIT-401 0.17 ppm	AUT-501 6.89	
	LS-201 Normal	FIT-301 0.00 m³/h	AIT-402 275.70 mV	AUT-502 204.20 mV	
	LS-202 Normal	DPII-301 0.95 kPa LL		AUT-503 264.23 µS/cm H	
	LS-203 Normal			AUT-504 14.27 µS/cm H	
	AIT-201 142.18 µS/cm L			FIT-501 0.00 m³/h L	
	AIT-202 7.20 H			FIT-502 0.00 m³/h HH	
	AIT-203 293.59 mV L			FIT-503 0.00 m³/h HH	
				FIT-504 0.00 m³/h LL	
				PIT-501 2.64 kPa LL	
				PIT-502 0.00 kPa H	
				PIT-503 0.00 kPa	

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.

2637

2638 Five layers are present starting from the highest to the lowest:

2639

- 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).

2646 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

2647

2648

2649 without any adverse impact on data or control functionality.
 2650 PLCs between the different process stages at Layer 1 communicate with
 2651 each other through a star network topology using the CIP protocol on Eth-
 2652 erNet/IP, previously discussed in Section 2.2.6.3.

2653 Regarding zoning, the SWaT system is divided into three zones, each
 2654 containing one or more layers. These zones are, in descending order of
 2655 security level:

- 2656 • **Plant Control Network, or Control System:** includes layers from 0
 2657 to 2;
- 2658 • **DMZ:** includes Layer 3.5;
- 2659 • **Plant Network:** includes Layer 3;

2660 Figure 5.3 provides a clearer visualization of the zoning and layer division
 2661 within the network architecture of the SWaT system. This diagram high-
 2662 lights the distinct zones and their corresponding layers, offering a com-
 2663 prehensive overview of the system's network structure.

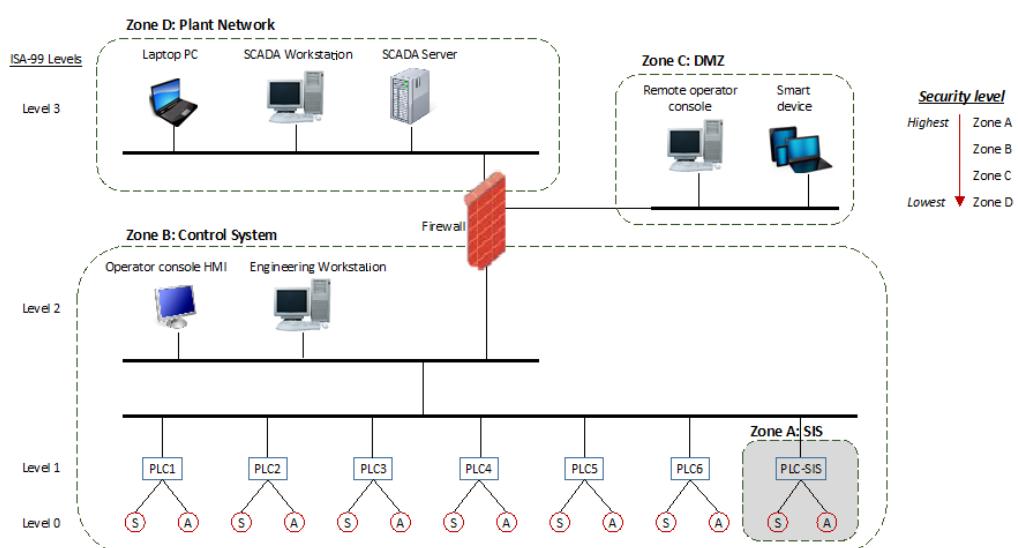


Figure 5.3: SWaT network architecture

2664 A specific IP address is associated with each device: in Table 5.1 we
2665 report the addresses for the PLCs, historian, and Touch Panel in the *Plant*
2666 *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

2667 5.2 Datasets

2668 To facilitate the study and testing of technologies related to cyber security
2669 in Industrial Control Systems and critical infrastructure, iTrust offers
2670 researchers worldwide the opportunity to access a range of datasets [68].
2671 These datasets consist of a collection of data obtained from the SWaT sys-
2672 tem, encompassing information on both the physical processes and net-
2673 work communications. The data is organized into different years, and
2674 researchers can request access to these datasets for their analysis and ex-
2675 perimentation purposes.

2676 **Physical Process Datasets** The datasets containing information about
2677 the physical processes are provided in CSV format files. These files en-
2678 compass data collected during different time intervals, which can vary
2679 from a few hours to entire days. The granularity of the data is typically

2680 at a one-second interval, although there may be some exceptions. The col-
2681 lected data primarily consists of timestamped sensor measurements and
2682 actuator status values for each PLC, describing the physical properties of
2683 the testbed in operational mode.

2684 **Network Communications Datasets** Network communications are typi-
2685 cally available in the form of *Packet Capture* format (PCAP) files. These files
2686 contain captures of communication network traffic, allowing researchers
2687 to analyze and examine the network interactions. In some instances, CSV
2688 files are provided instead of PCAP files, featuring different characteristics
2689 for the collected data.

2690 5.2.1 Our Case Study: the 2015 Dataset

2691 The dataset selected as a case study to apply the framework discussed
2692 in the previous chapter is specifically the dataset from the year 2015 [69].
2693 The main reason for this choice is the unique characteristics found in the
2694 physical process dataset that are not present in datasets from subsequent
2695 years.

2696 **Physical Process Data** The data collection process lasted 11 consecutive
2697 days, 24 hours per day. During the first 7 days, the system operated nor-
2698 mally without any recorded attacks. However, attacks were observed dur-
2699 ing the remaining 4 days. The collected data reflects the impact of these
2700 attacks, leading to the creation of two separate CSV files: one containing
2701 the recorded data of SWaT during the system's regular operations, and
2702 the other containing data recorded during the days of the attacks. To en-
2703 sure accurate information about the system, the dataset pertaining to the
2704 normal operations, which spans seven days, was chosen for analysis.

2705 Data collection occurs at a frequency of one data point per second,
2706 with the assumption that significant attacks cannot occur within a shorter
2707 time frame. Additionally, the firmware of the PLCs remains unchanged
2708 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).

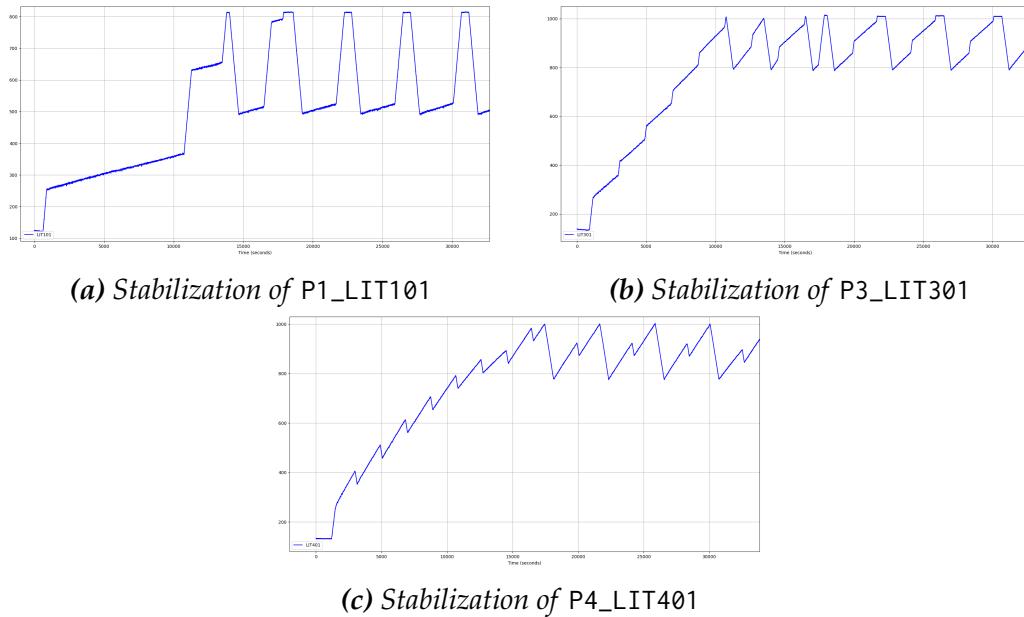


Figure 5.4: SWaT stabilization

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

2715 **Network Traffic** The network traffic was collected using an appliance
2716 from a well-known network hardware manufacturer and was made avail-
2717 able only in CSV format and not PCAP format. Table 5.2 shows some of
2718 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

2719 Unfortunately, the data provided are partial as it only includes read-
 2720 ings from sensors and does not include information on actuator status.

2721
 2722 Do not mislead by the word *Modbus* in the captured data fields: this is
 2723 most likely a feature of the sniffing appliance, which may have encapsu-
 2724 lated EtherNet/IP-CIP protocol packets in Modbus frames, as described
 2725 by T. A. Snide in [70].

2726 To provide evidence that the protocol being used is CIP over EtherNet/IP,
 2727 we can examine the "Source / Destination port" field in the captured data.
 2728 The presence of TCP port 44818 as the destination port aligns with the
 2729 TCP port used for transporting **explicit messages** in the EtherNet/IP pro-
 2730 tocol, as explained in Section 2.2.6.2. Additionally, the "Application name",
 2731 "Modbus Function Code" and "Modbus Function Description" fields fur-
 2732 ther support this conclusion. The "Application name" explicitly states
 2733 "CIP_read_tag_service," while the "Modbus Function Code" field contains
 2734 a decimal value of 76, which does not correspond to any known Modbus
 2735 Function Calls. When converted to hexadecimal, this value is 4C, match-
 2736 ing the CIP protocol read tag request code as indicated in the "Modbus
 2737 Function Description" field.

2738 In summary, the datasets for subsequent years were not selected due

2739 to certain limitations. In the case of the year 2017, the physical system
2740 data had a wide granularity, resulting in significant information loss. Ad-
2741 ditionally, in some cases, the datasets were considered "spurious" as there
2742 was no clear differentiation between data obtained during normal SWaT
2743 operations and data associated with attacks.

2744 Regarding network traffic data, there were instances where the data was
2745 either missing or fragmented into large PCAP files weighing several gi-
2746 gabytes (more than 6 GB each!), despite containing only a few minutes'
2747 worth of data. This made it impractical to manage such files given the
2748 available resources.

2749 Despite their large quantity, the CSV files associated with network traffic
2750 for the year 2015 are comparatively smaller in size, just slightly over 100
2751 MB each. These files cover a more extensive time period, which facilitates
2752 easier management and analysis. Prioritizing the physical process dataset
2753 as crucial, I have selected the year 2015 as a case study, even though the
2754 network data may be incomplete.

Our Framework at Work: Reverse Engineering of the iTrust SWaT System

2755 **I**N THIS chapter, our main objective is to apply the framework and method-
2756 ology introduced in Chapter 4 to the case study of the iTrust SWaT sys-
2757 tem, as illustrated in Chapter 5. The purpose of this analysis is to assess
2758 the effectiveness and potential of the proposed framework within the con-
2759 text of a system that closely replicates a real-world water treatment plant,
2760 albeit on a smaller scale.

2761 Due to the complexity of the system and the limited space available
2762 in this thesis, we will not conduct a comprehensive analysis and reverse
2763 engineering of the entire system. Instead, we will focus on specific parts
2764 for analysis. We leave it to the reader or those interested in utilizing the
2765 proposed methodology and framework to complete the analysis, should
2766 they choose to do so.

2767 By focusing on selective components and leaving room for further ex-
2768 ploration, we strike a balance between providing valuable insights and
2769 acknowledging the potential for additional research. This approach em-
2770 powers the reader and interested individuals to explore the iTrust SWaT
2771 system further and leverage the proposed methodology and framework
2772 for a more comprehensive analysis.

2773 6.1 Preliminary Operations

2774 Prior to beginning the actual analysis, several preliminary manual op-
2775 erations need to be conducted on the physical process dataset utilized as
2776 a case study, specifically the SWaT system dataset for the year 2015 as out-
2777 lined in Section 5.2.1. To simulate the data-capture process performed by
2778 Ceccato et al. using their scanning tool, the original dataset in XLSX format
2779 (proprietary to Microsoft Excel) was divided into multiple datasets in CSV
2780 format. Each of these datasets corresponds to the individual stages of the
2781 SWaT system and contains the respective registers. These resulting files
2782 were then saved in the directory specified by the `raw_dataset_directory`
2783 directive in the framework configuration file, `config.ini`, ready to be used
2784 in the pre-processing phase.
2785 Furthermore, the headers were manually renamed by adding a prefix from
2786 P1_ to P6_ to each register's name. This prefix indicates the stages, ensur-
2787 ing that each register is easily identifiable and linked to its corresponding
2788 stage.

2789 6.2 Planning the Analysis Strategy

2790 The complexity of the system being analyzed necessitates the adoption
2791 of a deliberate strategy for the analysis. It is not feasible to rely on trial and
2792 error or attempt every possible combination between stages. The former
2793 approach may overlook crucial relationships between PLCs or between
2794 registers, while the latter may result in excessive and unproductive efforts
2795 if the specific portion of the system being analyzed lacks significant infor-
2796 mation or relationships.

2797 A sound analysis strategy helps us focus on the important parts of the
2798 system, improving the quality of the analysis and leading to better process
2799 comprehension. By prioritizing our attention, we can gain a deeper under-
2800 standing of the crucial components, resulting in more informed decision-
2801 making and a comprehensive understanding of the overall processes.

2802 To define this strategy, a potential starting point could involve analyzing
2803 network traffic to determine the communication patterns and participants
2804 within the system. This can be accomplished by utilizing the techniques
2805 discussed in Section 4.2.2 on Network Analysis. By applying the
2806 Python script described in that section to the data extracted from the net-
2807 work traffic dataset debated in Section 5.2.1, we can generate a (simplified)
2808 network graph, as illustrated in Figure 6.1.

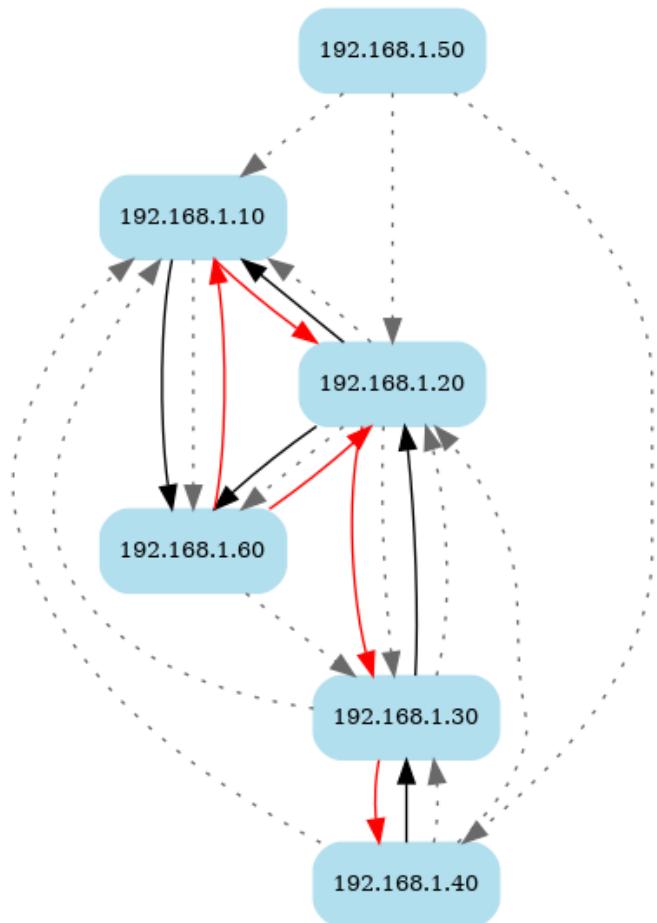


Figure 6.1: Simplified graph of the iTrust SWaT system network

2809 The graph clearly illustrates the structure of communications between
2810 the PLCs. Referring back to Table 5.1, which displays the IP address - PLC
2811 associations, we can observe that PLCs 1 through 4 communicate directly
2812 and sequentially with each other in a Request/Response communication

2813 pattern (represented by red and black arrows, respectively). Additionally,
2814 PLC6 communicates with both PLC1 and PLC2. On the other hand, the
2815 gray dotted arrows indicate communications for which we have knowl-
2816 edge of a response, but the corresponding request is unknown. For the
2817 purposes of our analysis strategy, we will not consider these communica-
2818 tions within this context.

2819 Based on our observations, the analysis strategy we will adopt involves
2820 considering sequential pairs of PLCs to effectively capture the relation-
2821 ships and implications between registers. Therefore, the PLC pairs we
2822 will focus on are PLC1-2, PLC2-3, and PLC3-4.

2823 **6.3 Reverse Engineering of the iTrust SWaT Sys-
2824 tem**

2825 Before we delve into the analysis, it is important to provide some pre-
2826 liminary remarks.

2827 Firstly, the analysis will be structured as a schematic analysis due to
2828 space constraints, which prevent us from presenting the extensive infer-
2829 ences and reasoning regarding the system in full detail. However, the
2830 general procedure of the methodology and how to reason about the data
2831 obtained from the framework have already been demonstrated through
2832 examples on the PLC1 of the SWaT system in Chapter 4. We encourage
2833 readers to refer to those examples for a more comprehensive understand-
2834 ing. In this analysis, our focus will be on illustrating the conjectures and
2835 properties that arise from the analysis, utilizing tables and the outputs
2836 generated during the analysis.

2837 The second premise addresses the process of defining the subsystems
2838 to be analyzed, which were obtained during the pre-processing phase,
2839 during the merge phase of the individual datasets. Apart from the pro-
2840 jection determined by the considered PLCs, a time-based selection of the

2841 analysis period has also been performed (see Section 4.2.3.1). This selec-
2842 tion spans a duration of 20,000 seconds, which is equivalent to approxi-
2843 mately five and a half hours or roughly five system cycles. The analysis
2844 begins at 100,000 seconds, which corresponds to approximately 27 hours
2845 from the start of the available data. This deliberate selection aims to ex-
2846 clude the initial transient period during which the SWaT system is initial-
2847 ized. We believe that this time range is more than sufficient for accurately
2848 defining the characteristics of the SWaT system components.

2849 The third premise introduces some conventions regarding the PLC reg-
2850 isters that will be utilized during the analysis. These conventions aid in
2851 identifying the nature and function of the registers. Specifically:

- 2852 1. Registers containing discrete values, such as binary or ternary val-
2853 ues, are likely to represent actuators.
- 2854 2. Registers with continuous values are identified as measurements.
- 2855 3. Actuators with three states are recognized as valves, while those
2856 with two states are considered pumps.
- 2857 4. Sensors with a wide range between the maximum and minimum val-
2858 ues are identified as level sensors for tanks. Sensors with a narrow
2859 range may represent other types of measurements such as flow or
2860 pressure sensors.
- 2861 5. Registers with a single constant value are recognized as either rel-
2862 ative setpoints or backup actuators. The distinction is made based
2863 on the value contained in the register. Binary values or values corre-
2864 sponding to the states of the main actuators indicate backup actua-
2865 tors, while different values (e.g., 20, 40, 80, etc.) suggest the presence
2866 of relative setpoints.
- 2867 6. Registers with similar names, such as P1_LIT101 and P3_LIT301, or
2868 P2_MV201 and P3_MV302, indicate the same type of register, such as a
2869 valve, pump, level sensor, or other category.

2870 By following these conventions, it becomes easier to interpret and ana-
 2871 lyze the functionality of the PLC registers within the system.

2872 6.3.1 Reverse Engineering of PLC1 and PLC2

2873 The initial focus of analysis will be on the pair comprising PLC1 and
 2874 PLC2. Let's delve into the main features of this subsystem by examining
 2875 the outcomes obtained from applying the framework to it.

2876 6.3.1.1 Pre-processing - Preliminary Analysis

2877 Listing 6.1 shows the outcomes obtained from automatic recognition of
 2878 sensors and actuators:

```
2879 1 Actuators:  

  2 P1_MV101 [0.0, 1.0, 2.0]  

  3 P1_P101 [1.0, 2.0]  

  4 P2_MV201 [0.0, 1.0, 2.0]  

  5 P2_P203 [1.0, 2.0]  

  6 P2_P205 [1.0, 2.0]  

  7  

  8 Sensors:  

  9 P1_FIT101 {'max_lvl': 2.7, 'min_lvl': 0.0}  

  10 P1_LIT101 {'max_lvl': 815.1, 'min_lvl': 489.6}  

  11 P2_AIT201 {'max_lvl': 256.5, 'min_lvl': 252.9}  

  12 P2_AIT202 {'max_lvl': 8.4, 'min_lvl': 8.3}  

  13 P2_AIT203 {'max_lvl': 342.8, 'min_lvl': 320.0}  

  14 P2_FIT201 {'max_lvl': 2.5, 'min_lvl': 0.0}  

  15  

  16 Hardcoded setpoints or spare actuators:  

  17 P1_P102 [1.0]  

  18 P2_P201 [1.0]  

  19 P2_P202 [1.0]  

  20 P2_P204 [1.0]  

  21 P2_P206 [1.0]
```

Listing 6.1: Preliminary analysis outcomes for sensors and actuators of PLC1-2

2900 Based on the output presented in the listing, we have compiled a preliminary
 2901 set of conjectures and properties for the subsystem in question. These
 2902 findings are summarized in Table 6.1:

#	Conjecture / Property	Reason
1	P1_LIT101, P1_FIT101, P2_FIT201, P2_AIT201, P2_AIT202, P2_AIT203 are measurements.	Registers contain continuous numerical values.
2	P1_LIT101 is identified as the level sensor specifically associated with a tank	Sensor has a wide range of values (minimum 489, maximum value of 815).
3	P1_MV101, P1_P101, P2_MV201, P2_P203 and P2_P205 are actuators .	Registers contain discrete numerical values.
4	P1_MV101 and P2_MV201 assume three distinct states, represented by the values 0, 1, and 2. They can be identified as values .	Obvious from the output and from point 3 of the third premise in Section 6.3
5	P1_P101, P2_P203 and P2_P205 assume two distinct states, represented by the values 1, and 2. They can be identified as pumps .	Obvious from the output and from point 3 of the third premise in Section 6.3
6	P1_102, P2_P201, P2_P202, P2_P204 and P2_P206 are spare actuators .	Registers contain constant numerical values.
7	A spare actuator in state 1 is considered to be OFF .	Assumption on the actuator states.

Table 6.1: Likely measurements and actuators conjectures for PLC1-2

2903 Continuing with the examination of the outcomes from the preliminary
 2904 analysis, Listing 6.2 shows the state durations of P1_MV101 and P2_MV201:

```
2905 1 Actuator state durations:
2906 2 P1_MV101 == 0.0
2907 3   9   9   10   9   9   10   9   9   10   9
2908 4
2909 5 P1_MV101 == 1.0
```

```

2910   6  1174  1168  1182  1160  1172
2911   7
2912   8 P1_MV101 == 2.0
2913   9  669   3019  3012  3000  2981
2914  10
2915  11 P2_MV201 == 0.0
2916  12  8     8     9     9     8     9     9     9     9
2917  13
2918  14 P2_MV201 == 1.0
2919  15  1057  1057  1045  1038  1039
2920  16
2921  17 P2_MV201 == 2.0
2922  18  120   3135  3144  3127  3109

```

Listing 6.2: Time duration of the states of actuators MV101 and MV201 of PLC1-2

2923 Based on these data, we can derive a conjecture about the state 0 of the
 2924 valves (Table 6.2):

#	Conjecture / Property	Reason
8	The state 0 of a valve is a transient state .	The duration of state 0 is only a few seconds. However, the subsequent states, have a significantly longer duration.

Table 6.2: Conjecture on the state 0 of actuators

2925 In the analysis of actuator state changes relative to measurements, we
 2926 referred to examples from Section 4.2.3.4 that demonstrated the behavior
 2927 of P1_P101. Applying the same reasoning also to P1_MV101 we can derive
 2928 the conjectures described in Table 6.3:

#	Conjecture / Property	Reason
9	State 1 of the pump P1_P101 corresponds to the OFF state of the actuator; state 2 of P1_P101 corresponds to the ON state of the actuator.	Derived from Conjecture 7.

10	The <i>relative setpoints</i> of P1_P101 are approximately 535 (minimum) when state change to 1 and approximately 813 (maximum) when state change to 2.	Observation derived from output
11	P1_P101 is responsible for emptying the tank represented by P1_LIT101.	Derived from Conjecture 9 and 10
12	Pumps are responsible for water outflow .	Corollary to Conjecture 11
13	State 1 of the valve P1_MV101 corresponds to the OFF state of the actuator; state 2 of P1_MV101 corresponds to the ON state of the actuator.	Derived from Conjecture 7.
14	The <i>relative setpoints</i> of P1_MV101 are approximately 500 (minimum) when state change to 2 and approximately 800 (maximum) when state change to 1.	Observation derived from output
15	P1_MV101 is responsible for filling the tank represented by P1_LIT101.	Derived from Conjecture 13 and 14
16	Valves are responsible for water intake .	Corollary to Conjecture 15

Table 6.3: Conjecture on valves and pumps

2929 Based on the conjecture, we can speculate that the level of the tank monitored by register P1_LIT101 is regulated by valve P1_MV101 at the inlet and
 2930 valve P1_P101 at the outlet. However, the role of sensor P1_FIT101 is still
 2931 uncertain, although in Section 4.2.3.4 we assumed it to be a flow or pressure
 2932 sensor, and requires further investigation.
 2933

2934 Regarding the elements controlled by PLC2, the analysis does not reveal the presence of another tank. Therefore, we cannot determine the
 2935 exact role of sensors P2_AIT20x and P2_FIT201 at this point.
 2936
 2937 However, there is a similarity observed between the relative setpoints of

2938 P1_P101 and those of P2_MV201, P2_P203, and P2_P205. These registers ex-
 2939 hibit very similar values during state changes, suggesting a potential rela-
 2940 tionship or similar control behavior between them.

2941 **6.3.1.2 Graphs and Statistical Analysis**

2942 Figure 6.2 illustrates the graphical representation of the registers in
 2943 PLC1-2 and their respective trends.

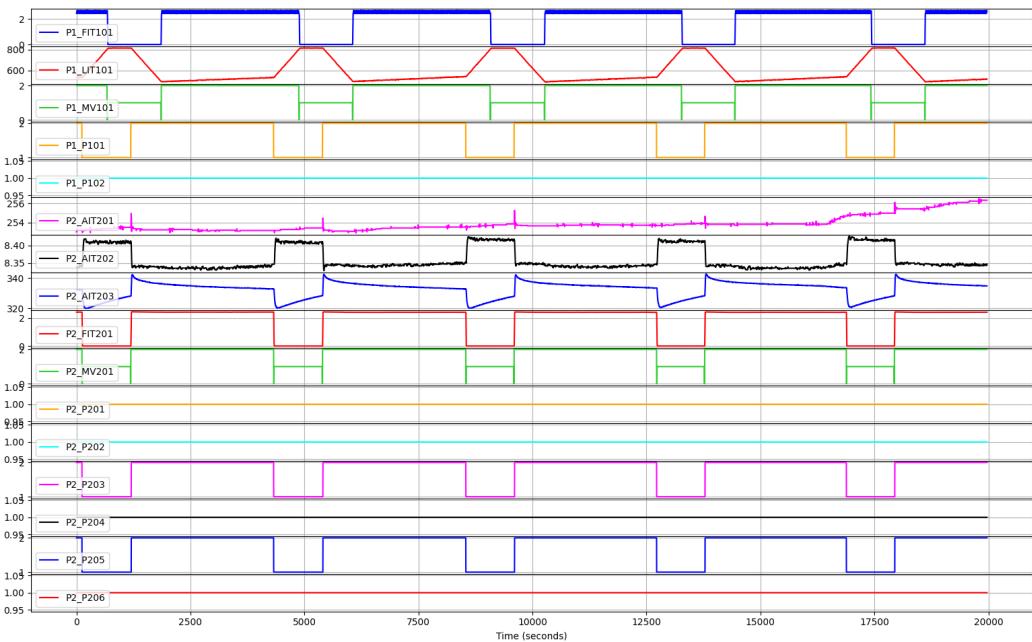


Figure 6.2: Chart of PLC1-2

2944 The image provides further support for Conjecture 6 and 7: P1_P102, P2_P201,
 2945 P2_P202, P2_P204, and P2_P206 are **backup actuators**, specifically **pumps**.
 2946 These actuators consistently maintain a state of 1 and do not seem to have
 2947 any significant impact on the measurements. This confirmation also sup-
 2948 ports the inference made in Conjecture 9 that state 1 of the pumps corre-
 2949 sponds to the **OFF state**, while state 2 corresponds to the **ON state**. As a
 2950 result, we can exclude these registers from further analysis.
 2951 Figure 6.3 provides a clearer representation of the subsystem after remov-
 2952 ing the spare actuators.

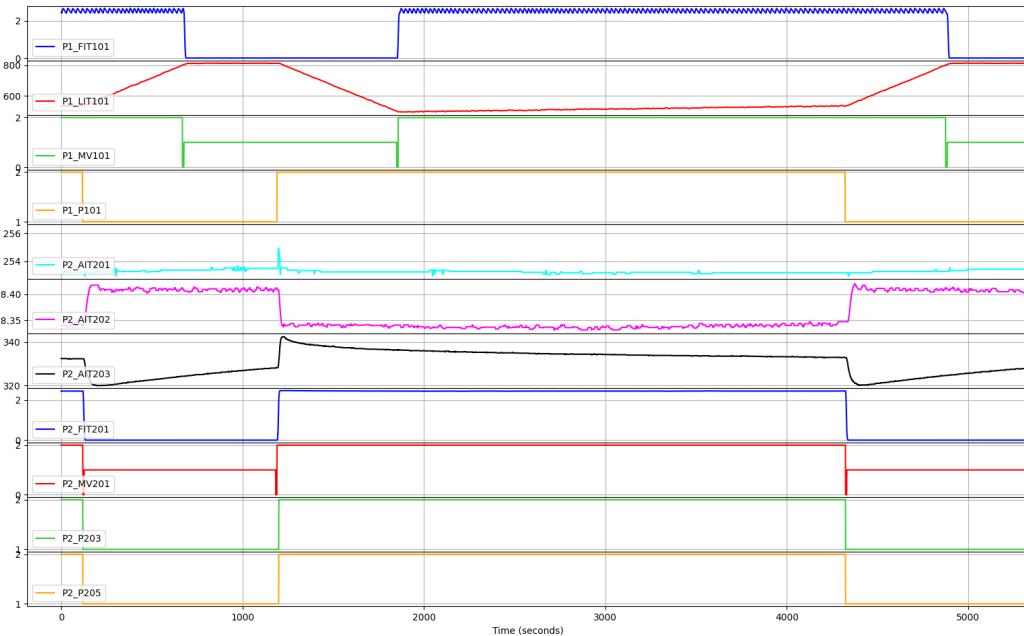


Figure 6.3: Chart of PLC1-2 without spare actuators (particular)

2953 Conjecture 1 and 2, which involve the identification of the tank level
 2954 sensor and measurements, are confirmed by the charts. Additionally, Con-
 2955 jecture 11 and 12 regarding pump behavior and Conjecture 15 and 16 re-
 2956 garding valve behavior in relation to the tank are also validated. Further-
 2957 more, Conjecture 8, which suggests that valves have a 0 state, is also con-
 2958 firmed.

2959 Looking at the plot, we can make further conjectures based on the ob-
 2960 served trends and patterns (Table 6.4):

#	Conjecture / Property	Reason
17	The trend of P1_FIT101 follows the trend of P1_MV101	P1_FIT101 takes a value of 0 when the valve P1_MV101 is OFF, and positive values (> 2) when the valve is ON.
18	P1_FIT101 is a flow sensor related to the inlet water flow at P1_LIT101.	Corollary to Conjecture 17
19	P2_FIT201 is also a likely flow sensor .	By analogy with P1_FIT101 (point 6 of the third premise in Section 6.3)

20	There is a correlation between the trend of P1_P101 and the trend of P2_MV201.	Observation derived from the plot
21	P2_MV201 is responsible for filling a tank that is <i>not</i> part of this subsystem.	Sensors P2_AIT202 and P2_AIT203 do not exhibit an increasing trend when the valve is ON, and there is no apparent correspondence between the trend of the valve and the trend of P1_LIT101.
22	P2_AIT201 is a sensor that measures some property of the water .	Sensor does not show a cyclic pattern in its measurements.
23	P2_AIT202 is a sensor that measures some property of the water .	Very narrow range between maximum and minimum values (point 4 of the third premise in Section 6.3)
24	The trend of P2_AIT202 and P2_AIT203 follows the trend of P2_P20x pumps	Observation derived from the plot

Table 6.4: Conjecture on the charts

2961 Currently, the specific role of P2_AIT203 sensors and the correlation be-
 2962 tween P2_P20x pumps and P2_AIT202 and P2_AIT203 sensors cannot be
 2963 determined. These aspects require further analysis and investigation as
 2964 they exhibit a cyclic pattern that seems to be associated with the behavior
 2965 of the pumps.

²⁹⁶⁶ **6.3.1.3 Invariant Inference and Analysis**

²⁹⁶⁷ **6.3.1.4 Business Process Analysis**

²⁹⁶⁸ **6.3.2 Reverse Engineering of PLC2 and PLC3**

²⁹⁶⁹ **6.3.2.1 Pre-processing**

²⁹⁷⁰ **6.3.2.2 Graphs and Statistical Analysis**

²⁹⁷¹ **6.3.2.3 Invariant Inference and Analysis**

²⁹⁷² **6.3.2.4 Business Process Analysis**

²⁹⁷³ **6.3.3 Reverse Engineering of PLC3 and PLC4**

²⁹⁷⁴ **6.3.3.1 Pre-processing**

²⁹⁷⁵ **6.3.3.2 Graphs and Statistical Analysis**

²⁹⁷⁶ **6.3.3.3 Invariant Inference and Analysis**

²⁹⁷⁷ **6.3.3.4 Business Process Analysis**

Chapter

7

Conclusions

²⁹⁷⁸ 7.1 Discussions

²⁹⁷⁹ 7.2 Future work

List of Figures

2.1	ICS architecture schema	6
2.2	PLC architecture	9
2.3	PLC communication schema	10
2.4	Comparison between ST language and Ladder Logic	11
2.5	Example of HMI for a water treatment plant	14
2.6	Modbus Request/Response schema	16
2.7	Modbus RTU frame and Modbus TCP frame	17
2.8	Example of packet sniffing on the Modbus protocol	19
2.9	OSI model for EtherNet/IP stack	20
3.1	Workflow of Ceccato et al.'s stages and operations with used tools	28
3.2	The simplified SWaT system used for running Ceccato et al. methodology	29
3.3	Output graphs from graph analysis	33
3.4	An example of Disco generated activity diagram for PLC2 .	37
3.5	Execution traces of InputRegisters_IW0 on the three PLCs .	39
3.6	Business process with states and Modbus commands for the three PLCs	41
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.	47

3.8	Behavior of the Graph Analysis on the Ceccato et al.'s tool	49
3.9	Histogram plot overshadowing statistical information shown on the terminal window in the background	50
3.10	Example of Daikon's output	52
4.1	Network Communications between 192.168.1.40 (Source) and 192.168.1.30 (Destination)	66
4.2	Network Communications between 192.168.1.40 (Source) and 192.168.1.30 (Destination) in textual mode	67
4.3	Slope comparison with granularity 30 (a), 60 (b) and 120 seconds (c)	75
4.4	Savitzky-Golay filter (blue line), STL decomposition (green) and RDP algorithm (orange) comparison	77
4.5	Slope after the application of the STL decomposition	78
4.6	The new slope representation (green line) and the smoothed measurement data obtaind with the STL decomposition (red)	79
4.7	Plotting registers on the same y-axis	84
4.8	Example of the new graph analysis	85
4.9	Example of transitive closure graphs for invariants in PLC1 of iTrust SWaT system	92
4.10	Directory (a) and outcome files (b) for the single actuator states analysis	95
4.11	Daikon outcome files for system configuration analysis. Each file represents a single system state	96
4.12	Activity diagram for PLC1 of the iTrust SWaT system	104
5.1	SWaT architecture	109
5.2	Sensors and actuators associated with each PLC	110
5.3	SWaT network architecture	111
5.4	SWaT stabilization	114
6.1	Simplified graph of the iTrust SWaT system network	119
6.2	Chart of PLC1-2	126
6.3	Chart of PLC1-2 without spare actuators (particular)	127

List of Tables

2.1	differences between Information Technology (IT) and Industrial Control Systems (ICSs)	4
2.2	Modbus Function Codes list	18
3.1	Summary table of Ceccato et al. framework limitations	54
5.1	main IP addresses of the six PLCs and SCADA in SWaT	112
5.2	SWaT network traffic data	115
6.1	Likely measurements and actuators conjectures for PLC1-2	123
6.2	Conjecture on the state 0 of actuators	124
6.3	Conjecture on valves and pumps	125
6.4	Conjecture on the charts	128

Listings

3.1	Example of registers capture	31
3.2	Example of raw network capture	32
3.3	Statistical data for PLC1_InputRegisters_IW0 register	34
3.4	Generic example of a .spinfo file for customizing rules in Daikon	35
3.5	The three sections of Daikon analysis outcomes	35
4.1	Novel Framework structure and Python scripts	58
4.2	Paths and parameters for the Pre-processing phase in <i>config.ini</i> file	69
4.3	<i>config.ini</i> parameters for dataset enriching	71
4.4	Example of preliminar system analysis	80
4.5	Standard Daikon output for PLC1 of the iTrust SWaT system	90
4.6	Revised Daikon output with transitive closures for PLC1 of the iTrust SWaT system	92
4.7	Daikon outcomes for the system configuration P1_MV101 == 2, P1_P101 == 1 on P1_LIT101	97
4.8	Example of the data contained in the produced JSON file	101
6.1	Preliminary analysis outcomes for sensors and actuators of PLC1-2	122
6.2	Time duration of the states of actuators MV101 and MV201 of PLC1-2	123

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.motioncontrolltips.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 Systematic 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 algorithms 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).