

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	Invariants 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	60
4.2.1	Phase 1: Data Pre-processing	60
4.2.1.1	Subsystem Selection	62
4.2.1.2	Dataset Enrichment	64
4.2.1.3	Datasets Merging	71
4.2.1.4	Brief Analysis of the Obtained Subsystem	72
4.2.2	Phase 2: Graphs and Statistical Analysis	76
4.2.3	Phase 3: Invariant Inference and Analysis	82
4.2.3.1	Revised Daikon Output	82
4.2.3.2	Types of Analysis	86
4.2.4	Phase 4: Business Process Analysis	92
4.2.5	Phase 5: Network Analysis	92
5	Case study: the iTrust SWaT System	93
5.1	Architecture	93
5.1.1	Physical Process	94
5.1.2	Control and Communication Network	96
5.2	Datasets	98
5.2.1	Our Case Study: the 2015 Dataset	99

6 Our framework at work: reverse engineering of the SWaT system	103
6.1 Pre-processing	103
6.2 Graph Analysis	103
6.2.1 Conjectures About the System	103
6.3 Invariants Analysis	103
6.3.1 Actuators Detection	103
6.3.2 Daikon Analysis and Results Comparing	103
6.4 Extra information on the Physics	103
6.5 Business Process Analysis	103
7 Conclusions	105
7.1 Discussions	105
7.2 Future work	105
List of Figures	107
List of Tables	109
Listings	111
References	113

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

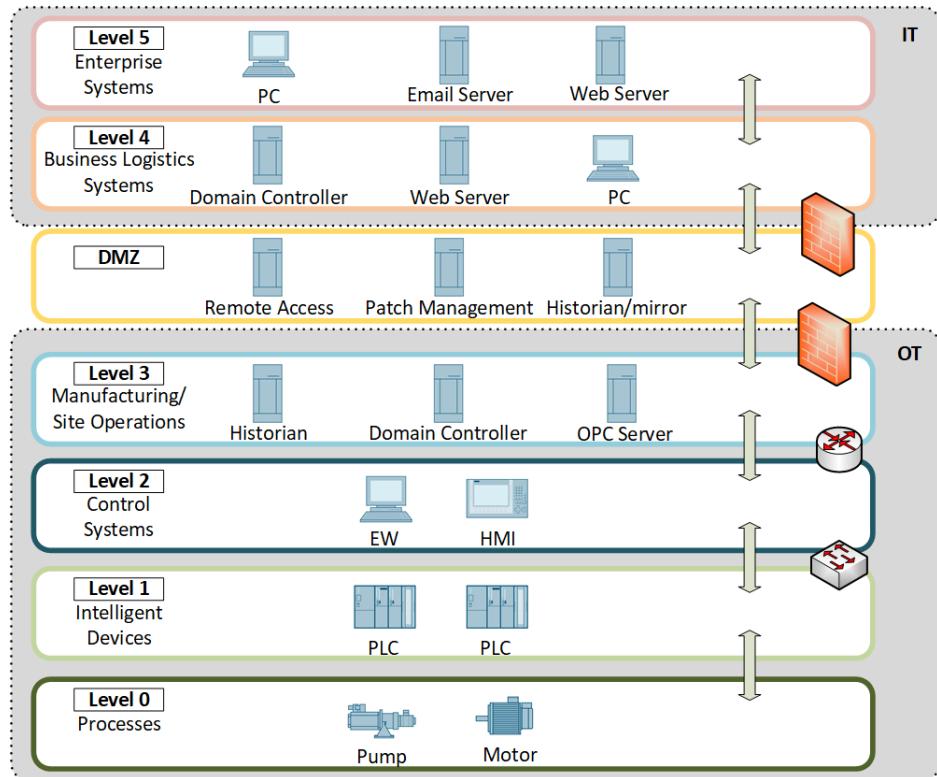


Figure 2.1: ICS architecture schema

- **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]:

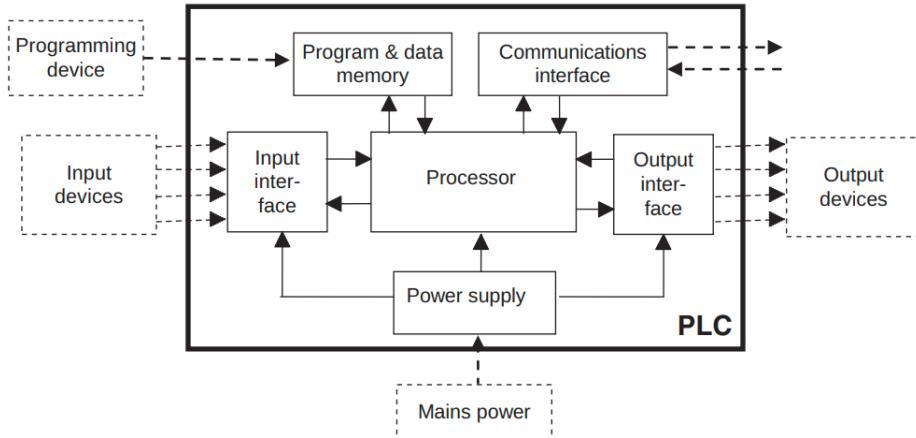


Figure 2.2: PLC architecture

- 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.

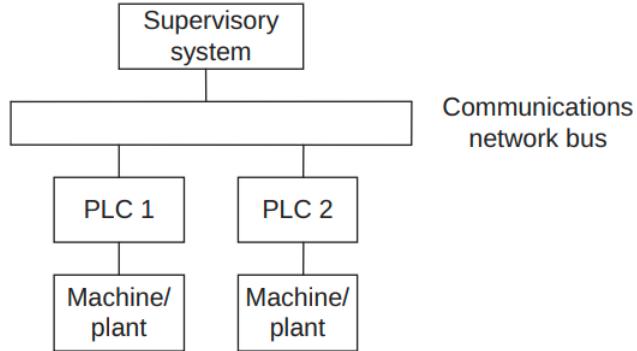


Figure 2.3: PLC communication schema

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).

```

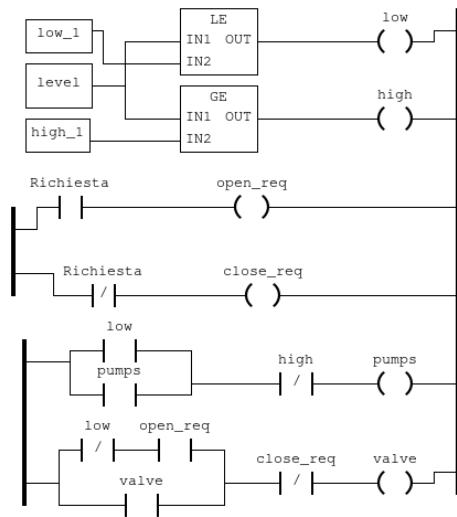
PROGRAM PLC1
VAR
  level AT %IW0 : INT;
  Richiesta AT %QX0..2 : BOOL;
  request AT %IW1 : INT;
  pumps AT %QX0..0 : BOOL;
  valve AT %QX0..1 : BOOL;
  low AT %MW0..0 : BOOL;
  high AT %MW0..1 : BOOL;
  open_req AT %MW0..3 : BOOL;
  close_req AT %MW0..4 : BOOL;
  low_1 AT %MW0 : INT := 40;
  high_1 AT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

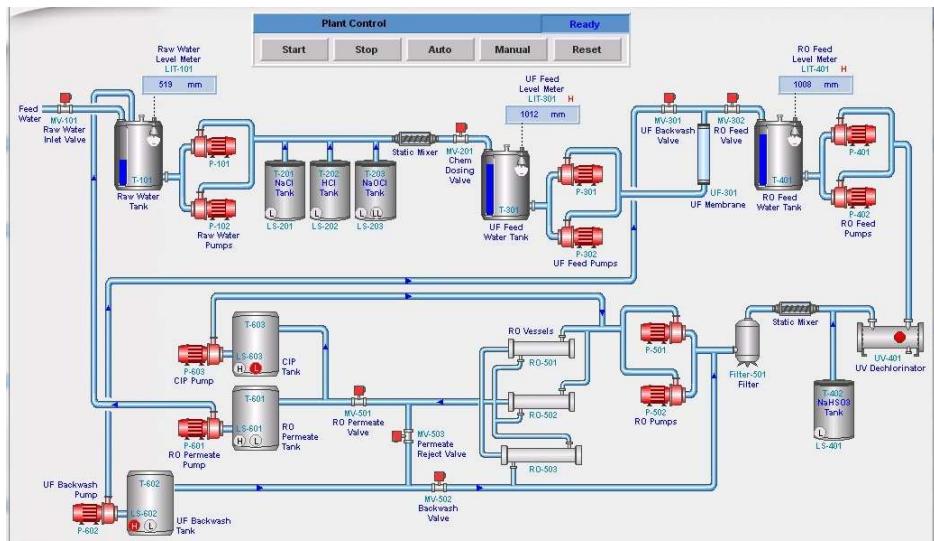


Figure 2.5: Example of HMI for a water treatment plant

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.

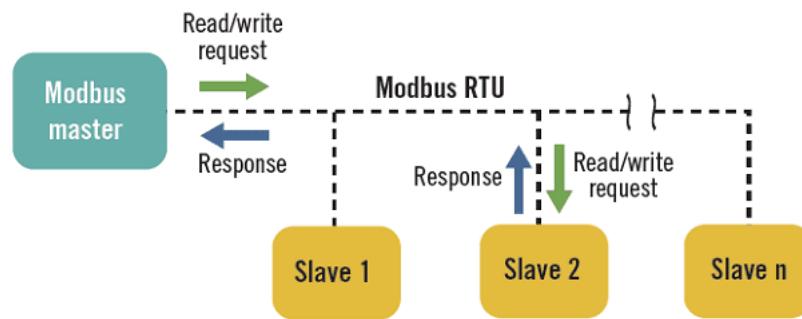


Figure 2.6: Modbus Request/Response schema

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.

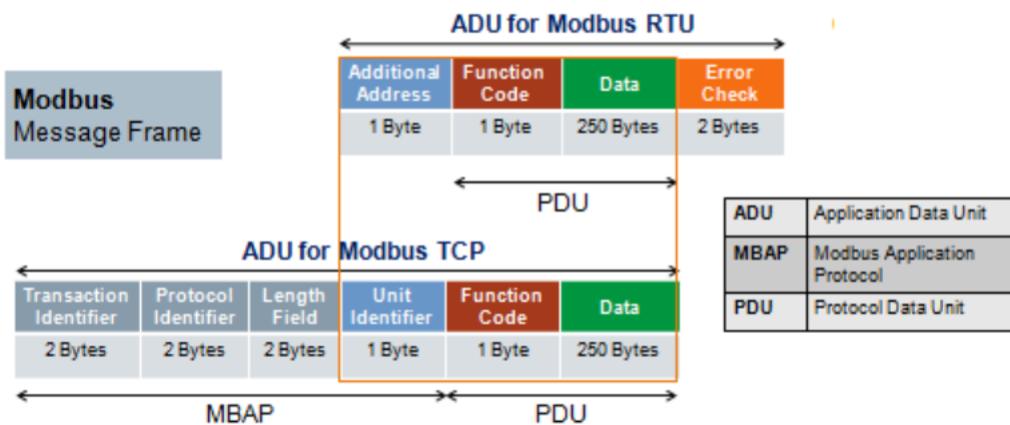


Figure 2.7: Modbus RTU frame and Modbus TCP frame

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.

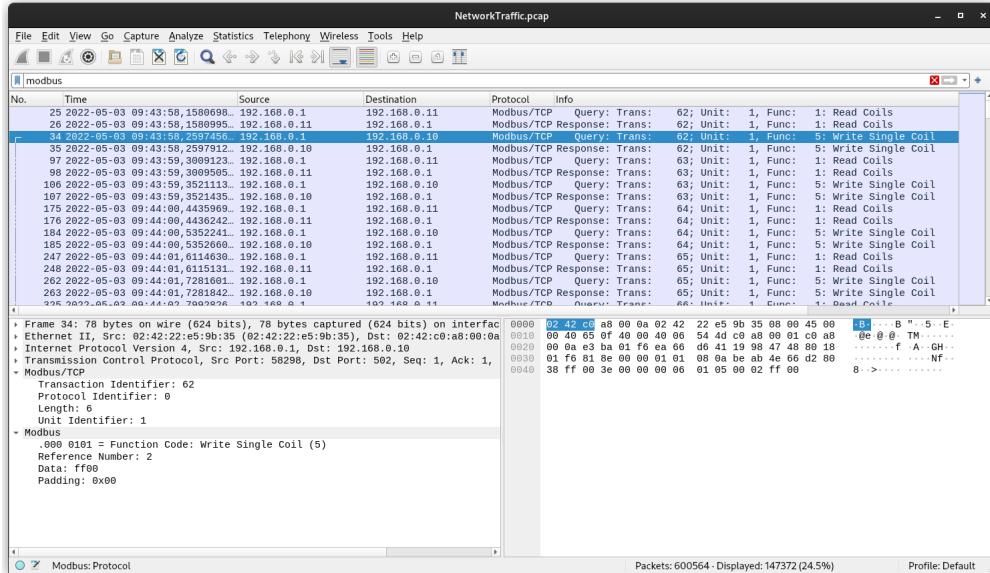


Figure 2.8: Example of packet sniffing on the Modbus protocol

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,

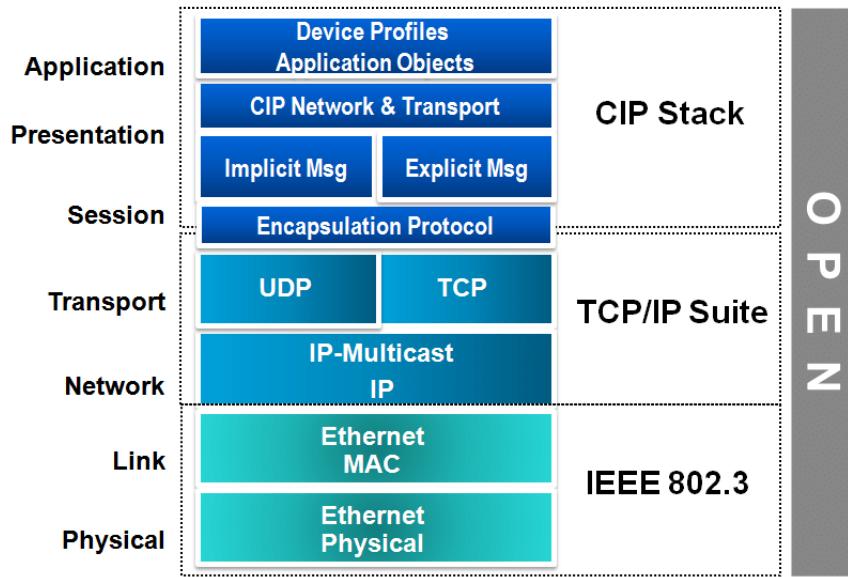


Figure 2.9: OSI model for EtherNet/IP stack

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

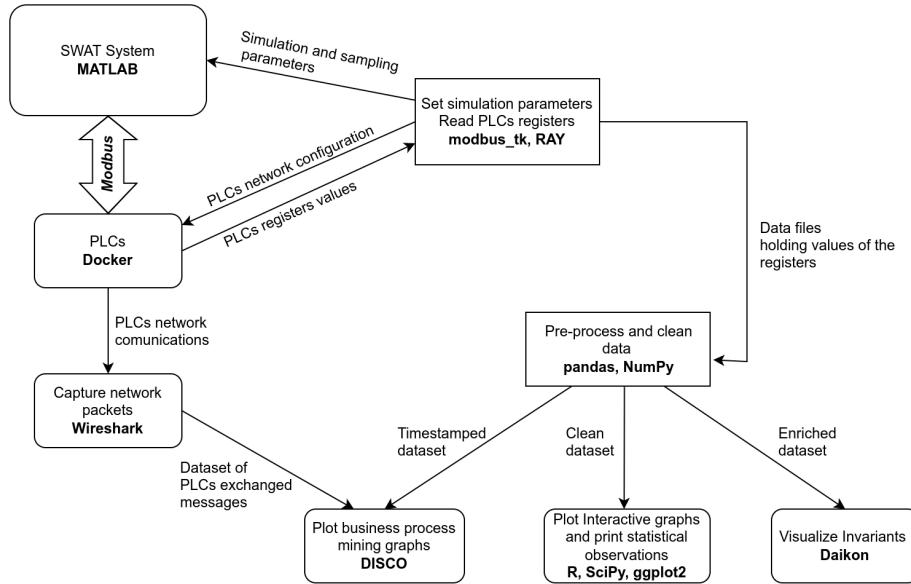


Figure 3.1: Workflow of Ceccato et al.'s stages and operations with used tools

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.

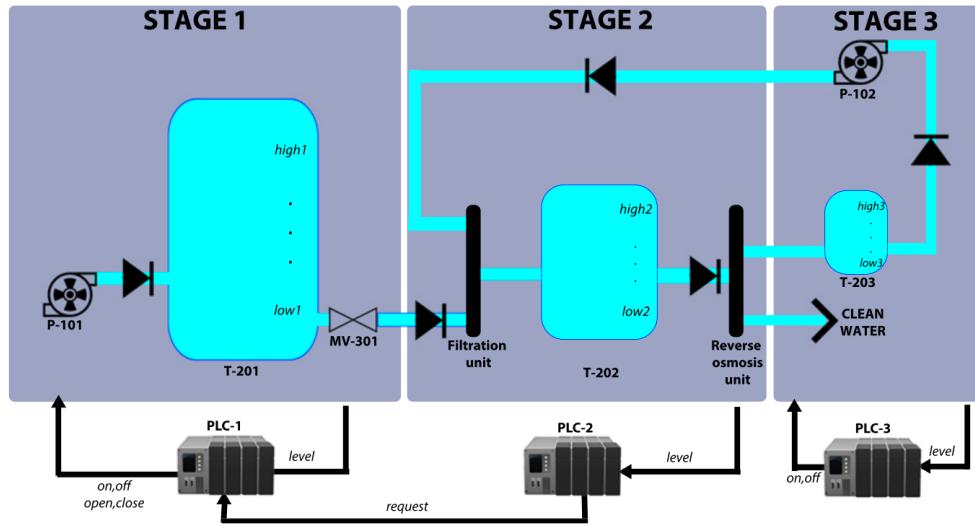


Figure 3.2: The simplified SWaT system used for running Ceccato et al. methodology

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).

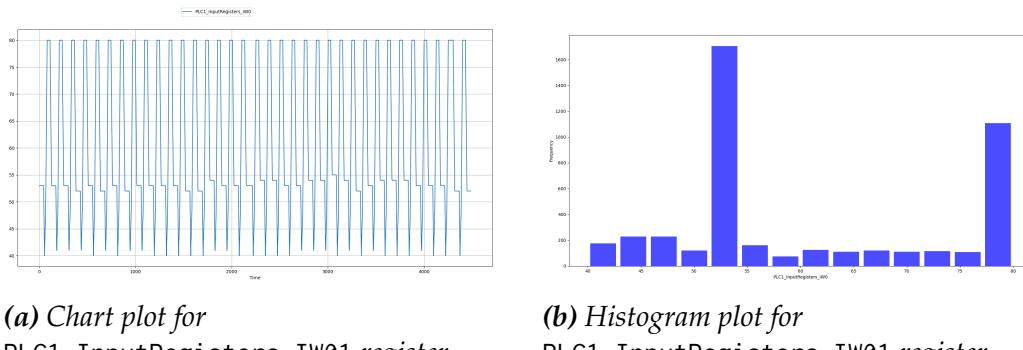


Figure 3.3: Output graphs from graph analysis

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 Invariants 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.

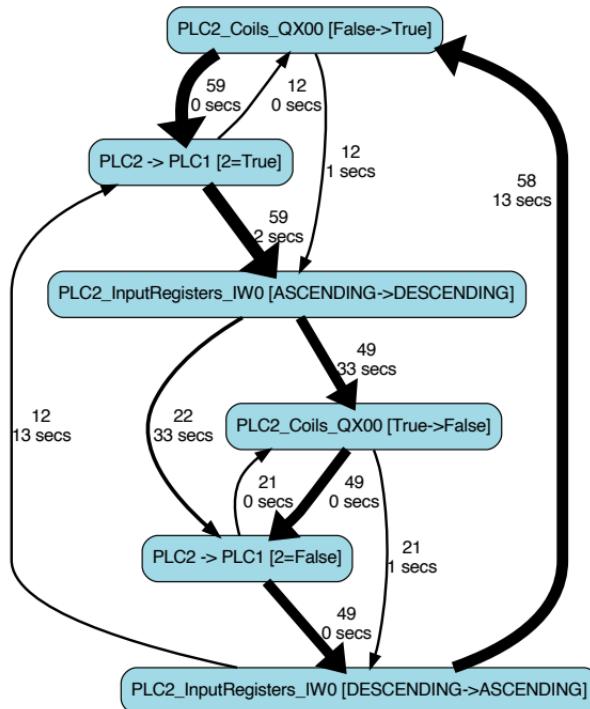


Figure 3.4: An example of Disco generated activity diagram for PLC2

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.

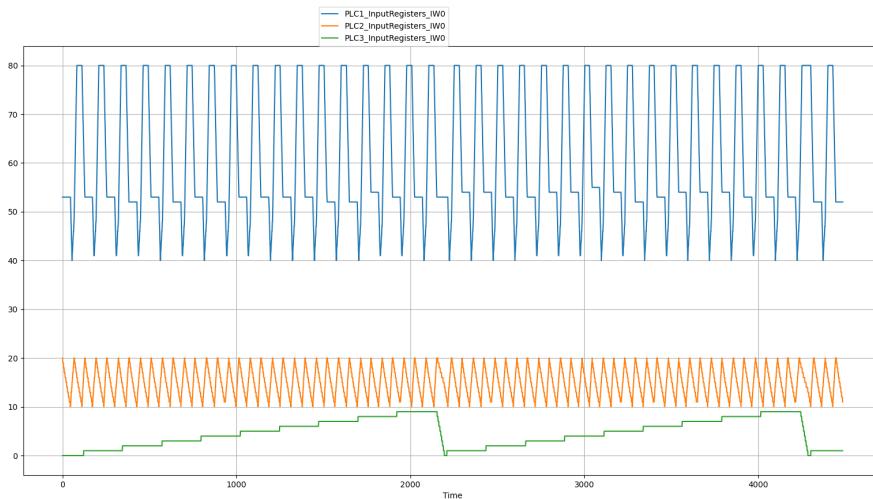


Figure 3.5: Execution traces of InputRegisters_IW0 on the three PLCs

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)

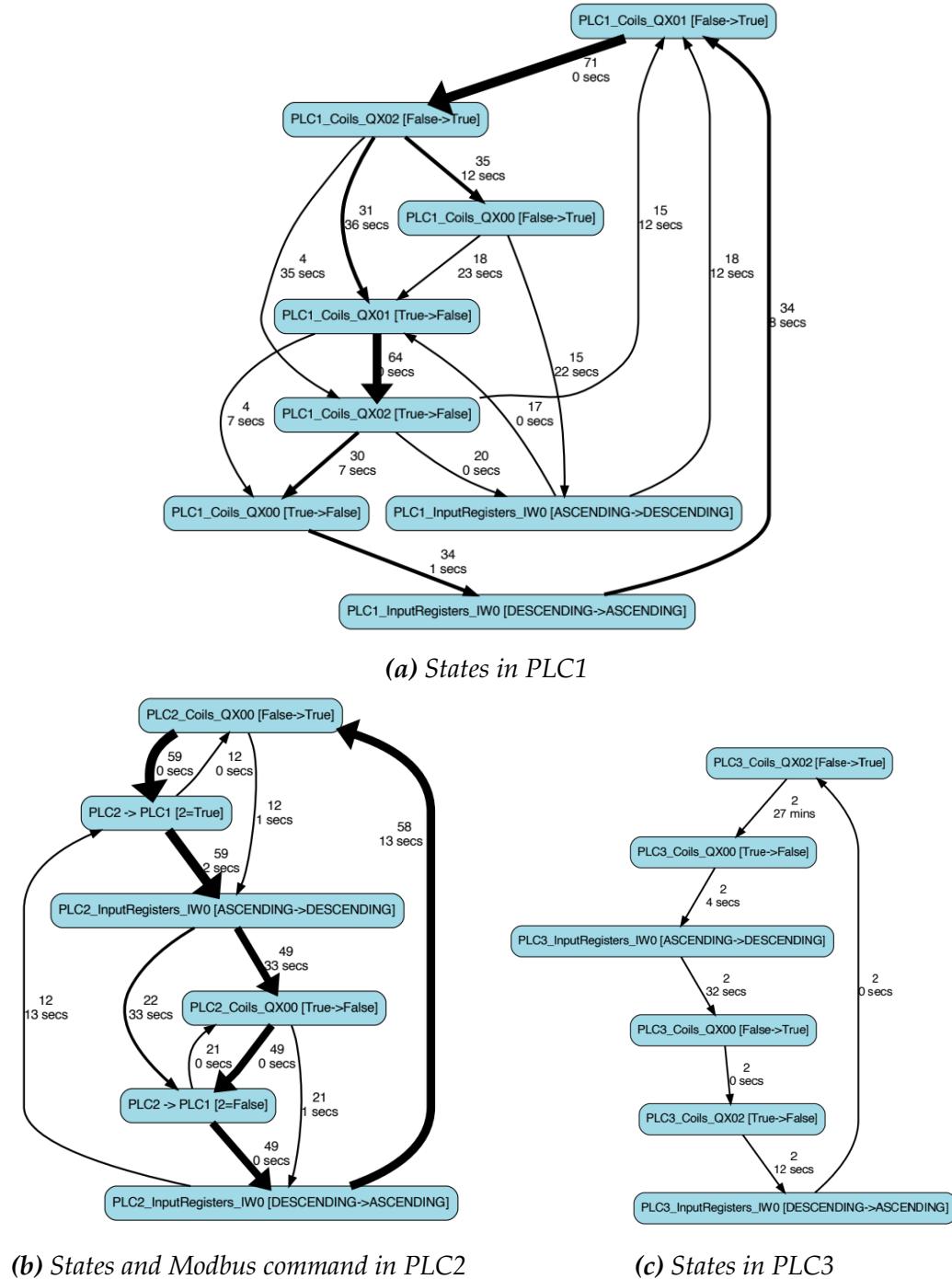


Figure 3.6: Business process with states and Modbus commands for the three PLCs

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.

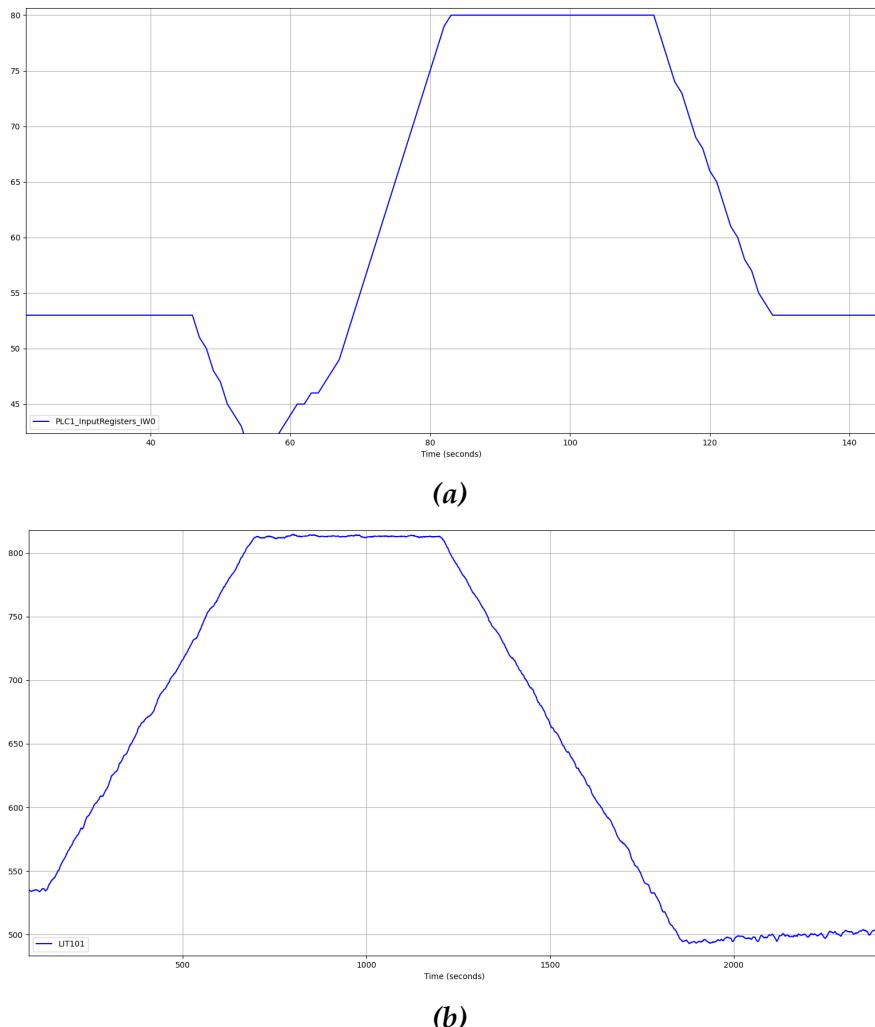


Figure 3.7: Water physics compared: simulated physics in the Simulink model (a) and physics in a real system (iTrust SWaT) (b). Fluctuations in the tank level in (b), almost completely absent in (a), can be appreciated.

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.

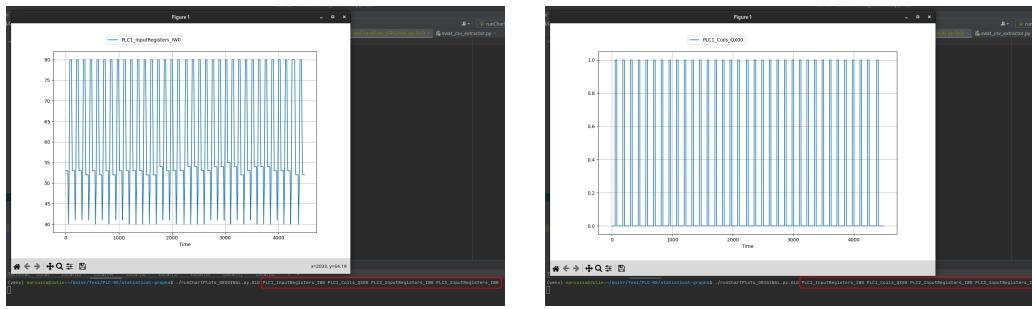
1087 In short, there is no possibility to analyze only a subsystem and thus
1088 make the analysis faster and more understandable. Moreover, a data
1089 gathering can take up to days, and the analyst/attacker may need to
1090 make an analysis of the system isolating precise time ranges, ignor-
1091 ing 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 In general, however, little statistical information is provided.

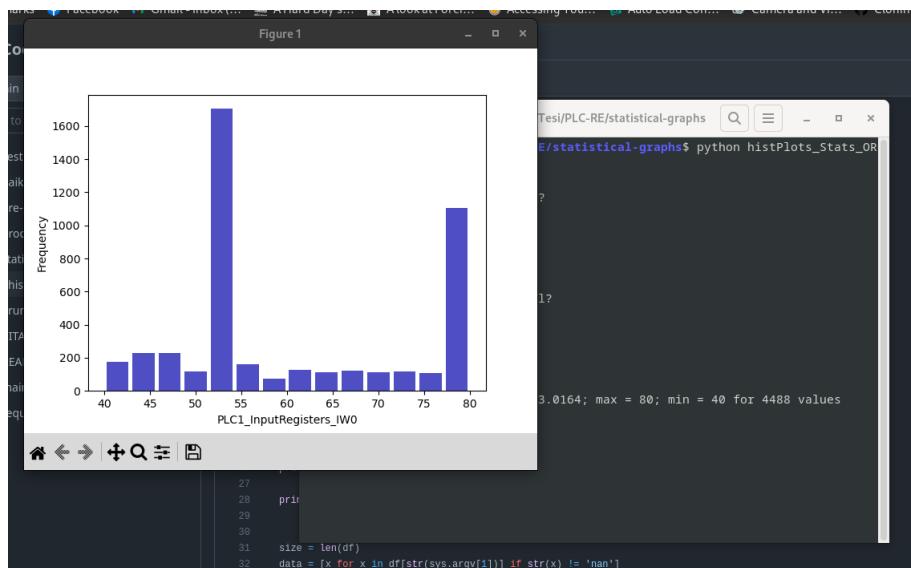


Figure 3.9: Histogram plot overshadowing statistical information shown on the terminal window in the background

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 of the resulting output: the software in fact returns a very long list 1181 of invariants, one invariant per line, many of no use and without 1182 correlating invariants that may have common features or deriving 1183 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 methodology**
1208 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 inserting
1211 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 Phase 1: Data Pre-processing

1365 *Data Pre-processing phase* is probably the most delicate and significant
1366 one: depending on how large the industrial system to be analyzed is, the

1367 data collected, and how it is enriched using the additional attributes, the
1368 subsequent system analysis will provide more or less accurate outcomes.

1369 The previous tool has several limitations, particularly at this stage. It
1370 does not allow for the isolation of a subsystem, either in terms of time or
1371 the number of PLCs to be analyzed. The system is considered as a whole
1372 without the ability to focus on specific subsystems. Additionally, many of
1373 the additional attributes had to be manually added, and for the ones en-
1374 tered automatically, there is no way to specify the register type to associate
1375 them with.

1376 The combination of these limitations, along with the presence of hard-
1377 coded references to attributes and registers in the tool's code, makes the
1378 analysis of the system more challenging. Furthermore, it compromises the
1379 accuracy and reliability of the obtained results in terms of both quantity
1380 and quality.

1381 In the proposed framework, these issues have been addressed by in-
1382 corporating new features. Firstly, the framework allows for the selection
1383 of a subsystem from the command line based on both temporal criteria
1384 and the specific PLCs to be included. This enables more focused and tar-
1385 geted analysis. Additionally, we have revamped the process of enriching
1386 the resulting dataset by eliminating manual entry of additional attributes.
1387 Instead, users now have the flexibility to determine the type of additional
1388 attribute to associate with a specific register.

1389 Furthermore, after the pre-processing stage, a preliminary analysis can be
1390 conducted on the resulting dataset. This analysis aims to identify the reg-
1391 isters that are associated with actuators, measurements, and hardcoded
1392 setpoints or constants. It provides insights into the dataset and helps in
1393 refining the enrichment step. The parameters for this analysis can be con-
1394 figured in the *config.ini* file, allowing for customization and fine-tuning of
1395 the process.

1396

1397 In the upcoming sections, we will delve into a more comprehensive ex-
1398 amination of the achievements made in this framework.

¹³⁹⁹ **4.2.1.1 Subsystem Selection**

¹⁴⁰⁰ In the previous tool, the datasets for each individual PLC in CSV for-
¹⁴⁰¹ mat were required to be placed in a specific directory that was hardcoded
¹⁴⁰² in the script. The script would then merge and enrich these datasets to
¹⁴⁰³ generate a single output dataset representing the complete process trace
¹⁴⁰⁴ of the industrial system. However, the script did not provide options to
¹⁴⁰⁵ select specific PLCs for analysis or define a temporal range for analysis.
¹⁴⁰⁶ This lack of flexibility made the analysis more complex, especially when
¹⁴⁰⁷ dealing with *transient states* (i.e., general states in which the industrial sys-
¹⁴⁰⁸ tem is still initializing before actually reaching full operation) or when fo-
¹⁴⁰⁹ cusing on specific parts of the industrial system during certain periods of
¹⁴¹⁰ interest. The fixed dataset structure also may increase the number of vari-
¹⁴¹¹ ables that could be analyzed.
¹⁴¹² Furthermore, the previous tool did not allow for specifying an output CSV
¹⁴¹³ file to save the resulting dataset. Each dataset creation and enrichment op-
¹⁴¹⁴ eration would overwrite the previous file, making it inconvenient for com-
¹⁴¹⁵ parisons between different execution traces unless the files were manually
¹⁴¹⁶ renamed.

¹⁴¹⁷ The proposed framework addressed these issues by introducing im-
¹⁴¹⁸ provements. First of all, in the general *config.ini* file there are some general
¹⁴¹⁹ default settings about paths, and among them the one concerning the di-
¹⁴²⁰ rectory where to place the datasets of the individual PLCs to be processed.
¹⁴²¹ In addition to this option, there are other ones that define further aspects
¹⁴²² related to the operations performed in this phase. Listing 4.2 shows the
¹⁴²³ settings in question:

```
1424 [PATHS]
1425 root_dir = /home/marcuzzo/UniVr/Tesi
1426 project_dir = %(root_dir)s/PLC-RE
1427 net_csv_path = %(root_dir)s/datasets_SWaT/2015/Network_CSV
1428
1429 [PREPROC]
1430 raw_dataset_directory = datasets_SWaT/2015 # Directory
1431   ↪ containing datasets
```

```
1432     dataset_file = PLC_SWaT_Dataset.csv # Default output  
1433     ↪ dataset  
1434     granularity = 10 # slope granularity  
1435     number_of_rows = 20000 # Seconds to consider  
1436     skip_rows = 100000 # Skip seconds from beginning
```

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

1437 At the same time, the user has the option to specify these settings via the
1438 command line using the new Python script called `mergeDatasets.py`, lo-
1439 cated in the pre-processing directory of the project. Any options provided
1440 through the command line will override the default settings specified in
1441 the `config.ini` file. These options are:

- 1442 • **-s or --skiprows:** initial transient period (expressed in seconds) to
1443 be skipped. This option is useful in case the system has an initial
1444 transient or the analyzer wishes to start the analysis from a specific
1445 point in the dataset;
- 1446 • **-n or --nrows:** time interval under analysis, expressed in terms of the
1447 number of rows in the dataset.
1448 This option makes a **selection** on the data of the dataset;
- 1449 • **-p or --plcs:** PLCs to be merged and enriched. The user can specify
1450 the desired PLCs by indicating the CSV file names of the associated
1451 datasets with no limitations on number.
1452 This option makes a **projection** on the data of the dataset.
- 1453 • **-d or --directory:** performs the merge and enrichment of all CSV files
1454 contained in the directory specified by user, overriding the default
1455 setting in `config.ini`. It is in fact the old functionality of the previous
1456 tool, maintained here to give the user more flexibility and conve-
1457 nience in case he wants to perform the analysis on the whole system.
1458 This is also the default behavior in case the `-p` option is not specified.
- 1459 • **-o or --output:** specifies the name of the file in which the obtained
1460 dataset will be saved. It must necessarily be a file in CSV format.

- 1461 • **-g** or **--granularity**: specifies a granularity (expressed in seconds)
 1462 that will be used to calculate the measurement slope during the dataset
 1463 enrichment phase. We will discuss this later in Section 4.2.1.2.

1464 **4.2.1.2 Dataset Enrichment**

1465 After a step in which a function is applied to each PLC-related dataset
 1466 to eliminate its unused registers within the system¹, the **dataset enrichment**
 1467 **operation** is performed.

1468 This operation differs from the previous version not only in the fact that it
 1469 is performed on each individual dataset and not on the resulting dataset,
 1470 but also in the additional attributes: not only are they greater in number,
 1471 but they are automatically calculated and inserted by the `mergeDatasets.py`
 1472 script into the dataset and, most importantly, it is possible to decide through
 1473 the parameters in the `config.ini` configuration file under the [DATASET] sec-
 1474 tion to which registers these attributes should be assigned.

1475 In Listing 4.3 we can see the list of additional attributes and how they
 1476 should be associated with the registers of the dataset:

```
1477 [DATASET]
1478 timestamp_col = Timestamp
1479 max_prefix = max_
1480 min_prefix = min_
1481 max_min_cols_list = lit|ait|dpit
1482 prev_cols_prefix = prev_
1483 prev_cols_list = mv[0-9]{3}|p[0-9]{3}
1484 trend_cols_prefix = trend_
1485 trend_cols_list = lit
1486 trend_period = 150
1487 slope_cols_prefix = slope_
1488 slope_cols_list = lit
```

Listing 4.3: config.ini parameters for dataset enriching

1489 Following is a brief explanation of the parameters just seen:

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

1490 **timestap_col** indicates the name of the column that contains the data
1491 timestamps. This parameter is used not only in this phase, but is
1492 also referred to in the Process Mining phase. In the previous work,
1493 this parameter was hardcoded and not configurable (and thus caus-
1494 ing errors if the system being analyzed changed)

1495 **max_prefix, min_prefix, max_min_cols_list** refer to any relative maximum
1496 or minimum values (*relative setpoints*) of one or more measures and
1497 that can be found and inserted as new columns within the dataset.
1498 The first two parameters indicate the prefix to be used in the column
1499 names affected by this additional attribute, while the third specifies
1500 of which type of registers we want to know the maximum and/or
1501 minimum value reached (several options can be specified using the
1502 logical operator | - or).

1503 If, for example, we want to know the maximum value of the regis-
1504 ters associated with the tanks, indicated in the iTrust SWaT system
1505 by the prefix LIT, we only need to specify the necessary parameter in
1506 the *config.ini* file, so `max_min_cols_list = lit`.

1507 The result will be to have in the dataset thus enriched a new column
1508 named `max_P1_LIT101`.

1509 **prev_cols_prefix, prev_cols_list** refer to the values at the previous time
1510 instant of the registers specified in `prev_cols_list`. It is possible to
1511 specify registers using *regex*, as in the example shown. It may be use-
1512 ful in some cases to have this value available to check, for example,
1513 when a change of state of a single given actuator occurs. The behav-
1514 ior of these parameters is the same as described in the point above.

1515 **slope_cols_prefix, slope_cols_list** are related to the calculation of the
1516 slope of a specific register that contains numeric values (usually a
1517 measure), that is, its trend. Slope calculation makes little sense on
1518 booleans. The slope can be **ascending** (if its value is greater than
1519 zero), **descending** (if less than zero) or **stable** (if approximately equal
1520 to zero). We will delve into the details of slope calculation in the fol-

1521 lowing paragraph, as it pertains to the attributes `trend_cols_prefix`,
 1522 `trend_cols_list`, and `trend_period`.

1523 Initially, the parameters for registers to be associated with each addi-
 1524 tional attribute may be left blank, as we may not have prior knowledge
 1525 about the system and are unsure about which registers correspond to actu-
 1526 ators, measurements, or other attributes. This information can be obtained
 1527 from the brief analysis that follows the merging of datasets. The analysis,
 1528 performed based on user's choice, provides indications on potential sen-
 1529 sors, actuators, and other relevant information. These indications help the
 1530 user set the desired values in the `config.ini` file and refine the enrichment
 1531 process by re-launching the `mergeDatasets.py` script.

1532 **Slope Calculation** The *slope* is an attribute that represents the **trend** of
 1533 the measurement being considered. It is particularly useful, in our con-
 1534 text, during the inference and invariant analysis phase to gather informa-
 1535 tion about the trend under specific conditions. The slope can generally be
 1536 classified as **increasing** ($slope > 0$), **decreasing** ($slope < 0$), or **stable** ($slope$
 1537 $= 0$).

1538 Normally, the slope is calculated through a simple mathematical formula:
 1539 given an interval a, b relative to the measurement l , the slope is given by
 1540 the difference of these two values divided by the amount of time t that the
 1541 measurement takes to reach b from a :

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

1542 In the proposed framework, similar to the previous tool, this time inter-
 1543 val (the granularity) can be adjusted to be either long or short. The choice
 1544 of granularity depends on the desired accuracy of the slope calculation. A
 1545 lower granularity will provide a slope that closely reflects the actual mea-
 1546 surement trend, while a higher granularity will result in flatter slope data.
 1547 Each time interval within which the measurement is divided corresponds
 1548 to a slope value. These slopes are calculated and added as additional at-

1549 tributes in the dataset. Later on, these slope values are used to determine
1550 the trend of the measurement in specific situations or conditions.

1551 Calculating the slope directly from the raw measurement data can be
1552 a suitable approach for systems where the measurements are not heavily
1553 influenced by **perturbations**. Perturbations, such as liquid oscillations in a
1554 tank during filling and emptying phases, can lead to fluctuating readings
1555 of the level. In such cases, maintaining a low granularity can provide a
1556 more accurate calculation of the overall trend that closely aligns with the
1557 actual measurement trend. The tanks of Ceccato et al.'s testbed are an ex-
1558 ample where this holds true.

1559 However, if perturbations significantly affect the measurement readings,
1560 calculating the slope on individual time intervals may result in an inaccur-
1561 ate trend definition, irrespective of the chosen granularity. In such cases,
1562 the fluctuating nature of the measurements due to perturbations can intro-
1563 duce errors in the slope calculation, making it less reliable as an indicator
1564 of the actual trend.

1565 Figure 4.1 demonstrates this assertion: the measurement, in blue, refers
1566 to the P1_LIT101 tank of the iTrust SWaT system; in red, the slope calcu-
1567 lation related to the measurement with three different granularities: 30
1568 (Figure 4.1a), 60 (Figure 4.1b) and 120 seconds (Figure 4.1c). It is notice-
1569 able that as the granularity increases, the slope values flatten. Moreover,
1570 in the time interval between seconds 1800 and 4200, the level of P1_LIT101
1571 exhibits a predominantly increasing trend, yet the calculated slope values
1572 fluctuate between positive and negative. Consequently, during the invari-
1573 ant analysis, the overall increasing trend may not be detected, resulting in
1574 a loss of information.

1575
1576 The previous tool did not take into account the possibility of having strongly
1577 perturbed data, which presented a challenge that we needed to address in
1578 the development of the proposed framework.

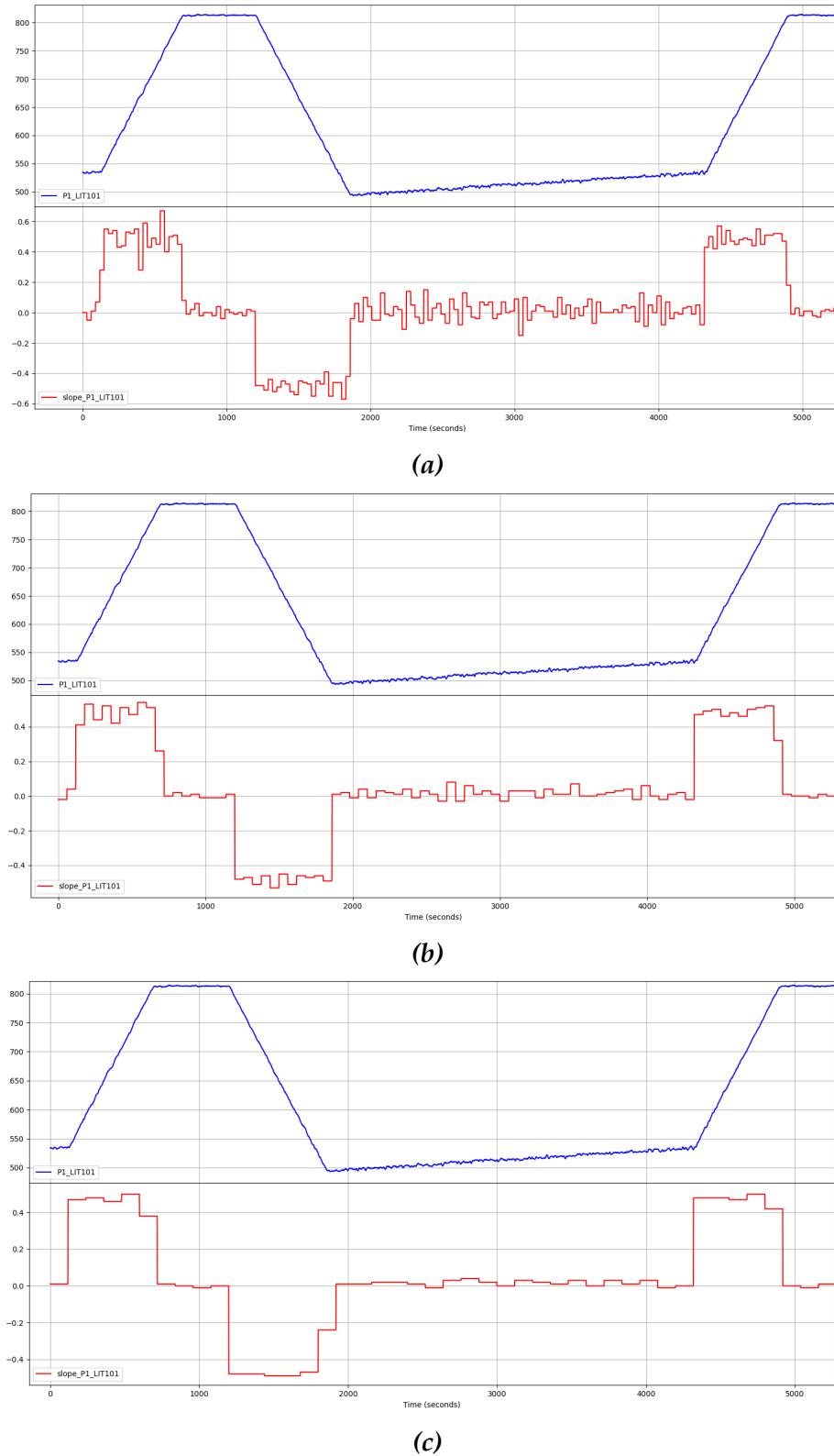


Figure 4.1: Slope comparison with granularity 30 (a), 60 (b) and 120 seconds (c)

1579 The solution to this problem involves applying techniques to reduce
1580 the "noise" in the data, aiming to achieve a more linear trend in the mea-
1581 surement curve. By minimizing the effects of perturbations, we can calcu-
1582 late slopes more accurately.

1583 There are various methods available for smoothing out noise in the data.
1584 In our framework, we focused on two commonly used approaches found
1585 in the literature: **polynomial regression** and **seasonal decomposition**. In
1586 addition to these two methods, we also explored the use of a **line simpli-
1587 fication algorithm**.

1588

1589 *Polynomial regression* [58] is a technique that allows us to create a filter to
1590 reduce the impact of noise on the data. By fitting a polynomial function
1591 to the measurements, we can obtain a smoother curve that captures the
1592 underlying trend while minimizing the effects of perturbations.

1593 *Seasonal decomposition* [59], specifically the part related to trending, is an-
1594 other method we explored. It involves decomposing the time series into
1595 different components, such as trend, seasonality, and residual. By isolat-
1596 ing the trend component, we can obtain a cleaner representation of the
1597 underlying pattern in the data.

1598 *Line simplification algorithms* [60] aim to reduce the complexity of a polyline
1599 or curve by approximating it with a simplified version composed of fewer
1600 points. By selectively removing redundant or less significant points, line
1601 simplification algorithms help reduce storage space and computational re-
1602 quirements while preserving the overall shape and characteristics of the
1603 original line.

1604 Regarding polynomial regression, we evaluated the use of the **Savitzky-**
1605 **Golay filter** [61] as a smoothing technique. For seasonal decomposition,
1606 we explored the **Seasonal-Trend decomposition using LOESS** (STL) method
1607 [62]. For the line simplification algorithm, we specifically considered the
1608 **Ramer-Douglas-Peucker** (RDP) algorithm [63].

1609

1610 Figure 4.2 shows a quick graphical comparison of these techniques com-
1611 pared with the original data. The solution adopted is the *STL decomposition*

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

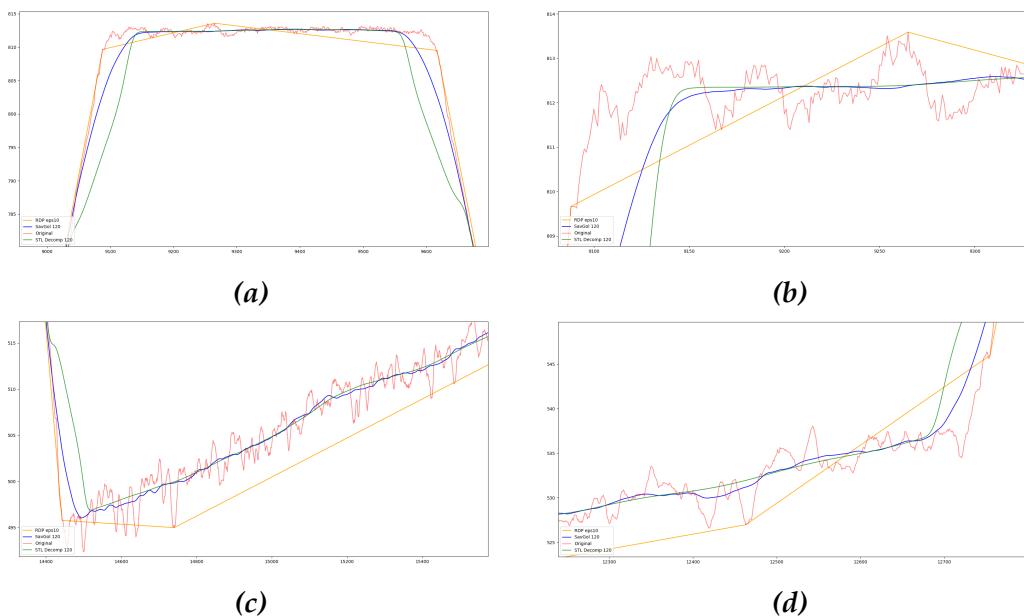


Figure 4.2: Savitzky-Golay filter (blue line), STL decomposition (green) and RDP algorithm (orange) comparison

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

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

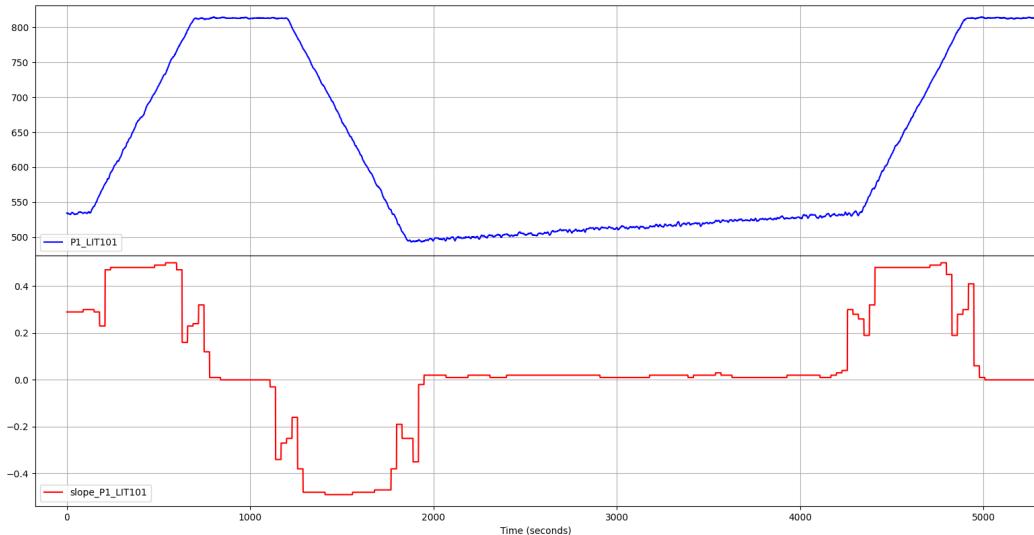


Figure 4.3: Slope after the application of the STL decomposition

1628 During the slope calculation, the analysis will be performed on the data
 1629 from the additional measurement trend attributes specified in the `trend_cols_list`
 1630 directive of the configuration file, rather than on the original unfiltered
 1631 data.

1632 To ensure proper interpretation by Daikon, the decimal values repre-
 1633 senting the calculated slopes are converted into **three numerical values**:
 1634 -1, 0, and 1. These values correspond to *decreasing* (if the slope is less than
 1635 zero), *stable* (if it is equal to zero), and *increasing* (if it is greater than zero)
 1636 trends, respectively. Figure 4.4 displays the modified slopes along with
 1637 the curve obtained from the STL decomposition:

1638 4.2.1.3 Datasets Merging

1639 During this step, the datasets of the individual PLCs are merged, re-
 1640 sulting in two separate datasets. The first dataset is enriched with addi-
 1641 tional attributes but excludes the timestamp column. This dataset is in-
 1642 tended for inference and invariant analysis. The second dataset does not
 1643 contain any additional data and is specifically used in the process mining
 1644 phase.

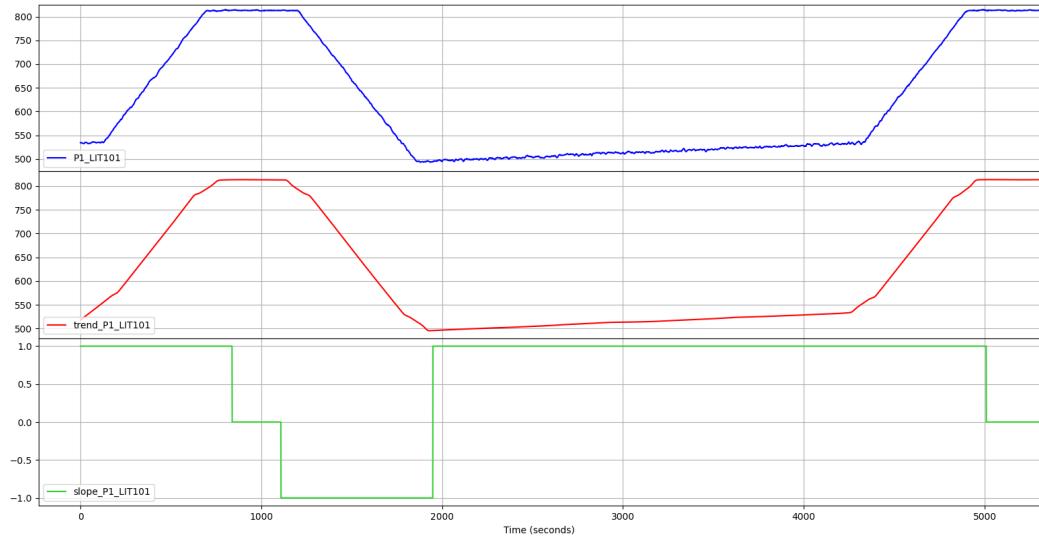


Figure 4.4: The new slope representation (green line) and the smoothed measurement data obtained with the STL decomposition (red)

1645 By default, the enriched dataset will be saved in CSV format in the
 1646 `$(project-dir)/daikon/Daikon_Invariants` directory. The other dataset,
 1647 without additional data, will be saved in the `$(project-dir)/process-mining/data`
 1648 directory. It's worth noting that both paths can be configured in the
 1649 `config.ini` file. The dataset name can be specified in the `config.ini` file or
 1650 through the `-o` command-line option. When generating the dataset for
 1651 process mining, the script will automatically add a `_TS` suffix to the file-
 1652 name to indicate that it includes the timestamp. This flexibility allows the
 1653 user to provide a different filename for each output, preventing overwrit-
 1654 ing of previous datasets. It enables the user to save the execution trace of
 1655 the selected subsystem separately and utilize them in subsequent analysis
 1656 phases.

1657 4.2.1.4 Brief Analysis of the Obtained Subsystem

1658 After merging the datasets, the user has the option to perform an **op-**
 1659 **tional analysis** of the resulting dataset to extract preliminary data. This
 1660 analysis aims to gather basic information about the (sub)system and po-
 1661 tentially refine the enrichment process. If the user chooses to proceed with

the analysis, the `mergeDatasets.py` script invokes another Python script located in the `$(project-dir)/pre-processing` directory called `system_info.py`. Relying on an analysis based on a combination of Daikon and Pandas this script performs a quick analysis of the dataset allowing to **estimate**, albeit approximately, the **type of registers** (sensors, actuators, ...), also identifying possible maximum and minimum values of measurements and hardcoded setpoints. Furthermore, leveraging the use of the additional attribute `prev_`, the `system_info.py` script is capable of deriving measurement values corresponding to state changes of individual actuators. This allows for the identification of specific measurements associated with the activation or deactivation of certain actuators within the system.

As the last information we have duration of actuator states for each cycle of the system: this information can be useful for making assumptions and conjectures about the behavior of an actuator in a specific state or, by observing the duration values of each cycle, highlighting anomalies in the system.

Listing 4.4 shows an example of this brief analysis related to PLC1 of the iTrust SWaT system (for brevity, only one measurement is reported in the analysis of actuator state changes):

```
1681      Do you want to perform a brief analysis of the dataset? [y
1682      ↵ /n]: y
1683
1684      Actuators:
1685      P1_MV101 [0.0, 1.0, 2.0]
1686      P1_P101 [1.0, 2.0]
1687
1688      Sensors:
1689      P1_FIT101 {'max_lvl': 2.7, 'min_lvl': 0.0}
1690      P1_LIT101 {'max_lvl': 815.1, 'min_lvl': 489.6}
1691
1692      Hardcoded setpoints or spare actuators:
1693      P1_P102 [1.0]
1694
1695      Actuator state changes:
1696          P1_LIT101  P1_MV101  prev_P1_MV101
```

1697	669	800.7170	0	2
1698	1850	499.0203	0	1
1699	4876	800.5992	0	2
1700	6052	498.9026	0	1
1701	9071	800.7170	0	2
1702	10260	499.1381	0	1
1703	13268	801.3058	0	2
1704	14435	498.4315	0	1
1705	17423	801.4628	0	2
1706	18603	498.1567	0	1
1707				
1708	P1_LIT101	P1_MV101	prev_P1_MV101	
1709	677	805.0741	1	0
1710	4885	805.7414	1	0
1711	9079	805.7806	1	0
1712	13276	805.1133	1	0
1713	17432	804.4068	1	0
1714				
1715	P1_LIT101	P1_MV101	prev_P1_MV101	
1716	1858	495.4483	2	0
1717	6060	497.9998	2	0
1718	10269	495.9586	2	0
1719	14443	495.8016	2	0
1720	18611	494.5847	2	0
1721				
1722	P1_LIT101	P1_P101	prev_P1_P101	
1723	118	536.0356	1	2
1724	4322	533.3272	1	2
1725	8537	542.1591	1	2
1726	12721	534.8581	1	2
1727	16883	540.5890	1	2
1728				
1729	P1_LIT101	P1_P101	prev_P1_P101	
1730	1190	813.0031	2	1
1731	5395	813.0031	2	1
1732	9597	811.8256	2	1
1733	13776	812.7283	2	1
1734	17938	813.3171	2	1
1735				

```

1736     Actuator state durations:
1737     P1_MV101 == 0.0
1738     9   9   10   9   9   10   9   9   10   9
1739
1740     P1_MV101 == 1.0
1741     1174   1168   1182   1160   1172
1742
1743     P1_MV101 == 2.0
1744     669   3019   3012   3000   2981
1745
1746     P1_P101 == 1.0
1747     1073   1074   1061   1056   1056
1748
1749     P1_P101 == 2.0
1750     118   3133   3143   3125   3108

```

Listing 4.4: Example of preliminar system analysis

From these results we can draw the following conjectures:

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

1767 value 1 (OFF) to value 2 (ON) when the level of P1_LIT101 is ap-
 1768 proximately 813, and it changes from value 2 (ON) to 1 (OFF) when
 1769 the level of P1_LIT101 is around 535. We can deduce that P1_P101 is
 1770 responsible for emptying the tank;

- 1771 • Regarding the actuators states duration, the very short duration of
 1772 P1_MV101 in state 0 is observable: since we are unable, at this pre-
 1773 liminary stage, to make assumptions about this, we will try to un-
 1774 derstand the behavior of the system in this actuator state in the next
 1775 stages of the analysis.

1776 The information obtained here can be used, as mentioned above, to
 1777 refine the enrichment of the dataset by setting directives in the [DATASET]
 1778 section of the *config.ini* file, should this be empty or only partially set, or to
 1779 make the first conjectures about the system, as we have just seen.

1780 The *system_info.py* file can also run in standalone mode if needed:
 1781 it takes as command-line arguments the dataset to be analyzed, a list of
 1782 actuators, and a list of sensors. For analysis related to state changes, the
 1783 dataset must mandatorily be of the enriched type.

1784 4.2.2 Phase 2: Graphs and Statistical Analysis

1785 The new *graph analysis* arises from the need to give the user an overview
 1786 of the (sub)system obtained in the previous pre-processing phase, identi-
 1787 fying more easily the typology of the registers and grasping more effec-
 1788 tively the relationships and the dynamics that may exist between the reg-
 1789 isters controlled by one or more PLCs, confirming the initial conjectures if
 1790 the brief analysis described in the previous section has been performed, or
 1791 making new ones thanks to the visual graph support.

1792 In Ceccato et al.'s framework, as mentioned in Section 3.2.7, it was only
 1793 possible to view the chart of one register at a time. While this allowed for
 1794 the identification or hypothesis of the register type, it made it challenging
 1795 to establish relationships with other components of the system and derive

conjectures about their behavior. To address this limitation, there was a need for a new tool that could provide more information in a more accessible manner.

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

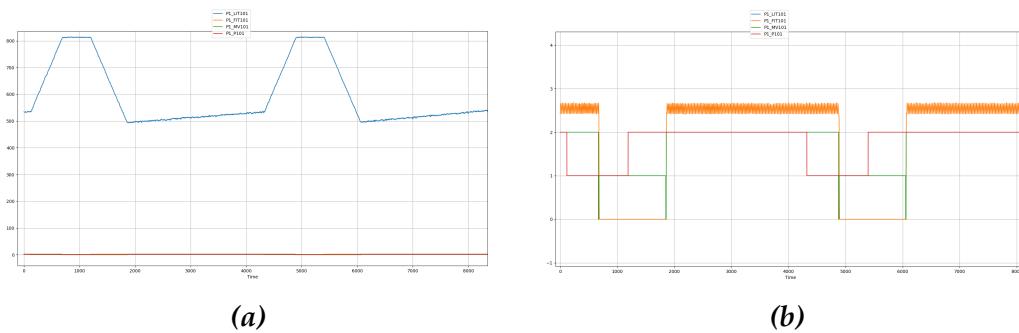
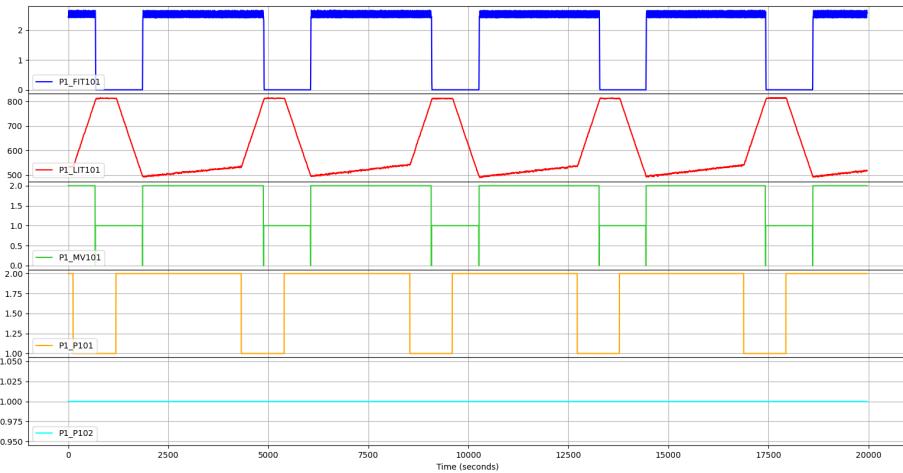


Figure 4.5: Plotting registers on the same y-axis

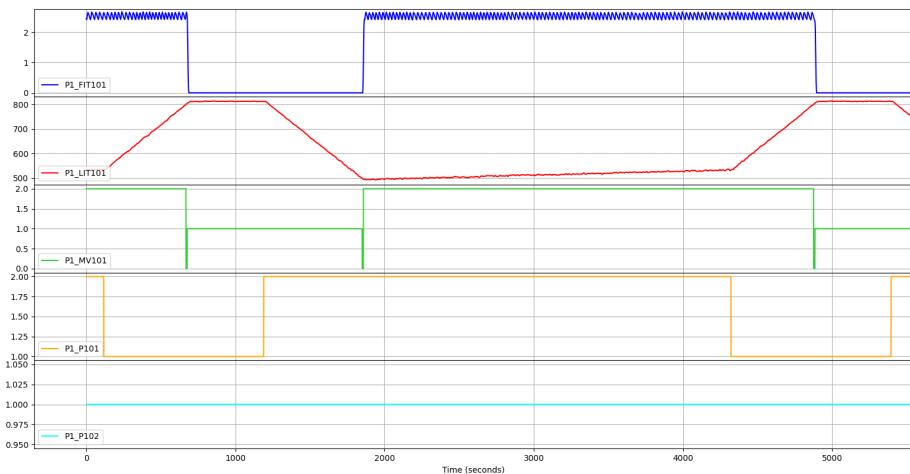
Figure 4.5a highlights the main issue with this approach, which is the use of the same y-axis for all the charts, representing the values of individual registers. When the range between register values is wide, it can result in some charts appearing as a single flat line or becoming indistinguishable,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

The solution to this issue is simple and effective: the use of **subplots**. Basically, each register corresponds to a subplot of the graph that shares the time axis (the x-axis) with the other subplots, but keeps the y-axis of the values of each register independent. This maintains the readability and comprehensibility of the charts, while simultaneously being able to immediately grasp the relationships between them. In addition, by sharing the time axis, it is possible to zoom in on a particular area of one of the charts and automatically the other ones will be zoomed in as well, thus

not losing any information and no connection between registers. Figure 4.6 illustrates more clearly what has just been explained: the charts refer to the PLC1 registers of the iTrust SWaT system.



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



(b) Zooming on a particular zone of the charts

Figure 4.6: Example of the new graph analysis

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

- 1825 • the data presented in the P1_FIT101 chart provides support for as-
1826 sociating this register with a **measurement**. Furthermore, the chart
1827 indicates a close correlation between the trend of this register and
1828 the periods in which P1_LIT101 exhibits an upward trend and the
1829 evolution of P1_MV101. The values of P1_FIT101, ranging from ap-
1830 proximately 0 to 2.5, are too limited to conclude that this register
1831 represents the level sensor of a tank. The overall trend also aligns
1832 with this interpretation, suggesting that it is more likely a **pressure**
1833 **or flow sensor**;

- 1834 • based on the information provided above, it appears that P1_LIT101
1835 indeed **represents the level sensor of a tank**, which was initially
1836 assumed in the brief analysis. The hypothesis is further reinforced
1837 by several factors. Firstly, the wide range of values observed in this
1838 register supports the notion of it being a level sensor. Additionally,
1839 the graphical representation of the level trend provides valuable in-
1840 sights. It shows an initial sharp rise, followed by a period of stabi-
1841 lization referred to as a *plateau*, and subsequently, a similarly steep
1842 descent. Finally, there is a new phase of slower level growth com-
1843 pared to the previous one. These patterns lend further support to
1844 the interpretation of P1_LIT101 as a sensor specifically measuring the
1845 level of a tank;

- 1846 • at the start of the ascending trend of P1_LIT101, the value of P1_MV101
1847 is 2. Conversely, during the *plateau* phase observed when the mea-
1848 surement is approximately 800 and during the descending trend,
1849 P1_MV101 has a value of 1. These observations provide further confir-
1850 mation of the initial hypothesis: the value 1 represents the **OFF state**
1851 of the sensor, while 2 represents the **ON state**. Moreover, P1_MV101
1852 can be identified as the actuator responsible for causing the rising
1853 level in P1_LIT101;

- 1854 • during the brief analysis, we noticed the short duration of P1_MV101
1855 in state 0, but we were unable to speculate on the underlying rea-

1856 son. However, further analysis of the graph reveals that P1_MV101
1857 transitions and remains in the 0 state, acting as a type of "transient"
1858 between states 1 and 2. It can be inferred that this period represents
1859 the actual time required for the actuator to change its state.

- 1860 • we can observe the behavior of P1_P101 in relation to P1_LIT101 and
1861 P1_MV101 to understand its role. At the beginning of the descend-
1862 ing trend of P1_LIT101, after the plateau phase, P1_P101 assumes a
1863 value of 2. It then changes back to value 1 at the point of the (pre-
1864 sumed) tank's level increase. Although this fact alone may not be
1865 entirely clear, when comparing the behaviors of P1_P101, P1_LIT101,
1866 and P1_MV101, certain patterns emerge. When P1_P101 and P1_MV101
1867 are both in state 1, the water level remains stable. However, when
1868 P1_P101 changes to state 2, the level of the measurement drops rapidly.
1869 Subsequently, when P1_MV101 also transitions to state 2, the level
1870 slowly starts to rise again, with a sudden increase occurring when
1871 P1_P101 changes from 2 back to 1. From these observations, it can be
1872 inferred that P1_P101 serves as the **actuator responsible for empty-**
1873 **ing the presumed tank**. State 1 represents the **OFF state**, while state
1874 2 represents the **ON state** of this actuator.
- 1875 • it appears that P1_P102 plays no active role in the system as its state
1876 remains constant at 1 throughout. Considering this information, it
1877 seems unlikely that P1_P102 could serve as a relative setpoint. There-
1878 fore, according to the brief analysis, it can be inferred that P1_P102 is
1879 most likely a **spare actuator** that is not actively utilized in the sys-
1880 tem. A *spare actuator* is indeed a secondary actuator that is intended
1881 to remain idle unless needed as a backup or replacement when the
1882 primary actuator is unavailable or needs to be taken out of service.

1883 As the reader has probably already noticed, the majority of the graphs
1884 presented in the previous sections and chapters were generated using the
1885 new graph analysis script, specifically the `runChartsSubPlots.py` script.
1886 This Python script is located in the `$(project-dir)/statistical-graphs`

1887 directory and utilizes the *matplotlib* libraries to generate the graph plots,
1888 similar to the previous tool.

1889 The script accepts the following command-line parameters:

- 1890 • **-f or --filename:** specifies the CVS format dataset to read data from.
1891 The dataset must be within the directory containing the enriched
1892 datasets for the invariant analysis phase. If subsequent parameters
1893 are not specified, the script will display all registers in the dataset,
1894 excluding any additional attributes;
- 1895 • **-r or --registers:** specifies one or more specific registers to be dis-
1896 played;
- 1897 • **-a or --addregisters:** adds one or more registers to the default visual-
1898 ization. This option is useful in case an additional attribute such as
1899 slope is to be analyzed;
- 1900 • **-e or --excluderegisters:** excludes one or more specific registers from
1901 the default visualization. This option is useful to avoid displaying
1902 hardcoded setpoints or spare registers.

1903 This script, like the previous ones, is designed to provide the maximum
1904 flexibility and ease of use for the user, combined with greater power and
1905 effectiveness in deriving useful information about the analyzed system.

1906 **Statistical Analysis** After careful consideration, we made the decision
1907 not to include the statistical analysis aspect of the previous tool in the
1908 framework. We found that there was no practical use for it. Instead, we
1909 integrated the relevant statistical information into the brief analysis con-
1910 ducted after the pre-processing phase. Additionally, we deemed the his-
1911 togram to have limited utility and considered it outdated in comparison
1912 to the new graph analysis approach we implemented.

1913 Although we decided not to include the statistical analysis aspect in the
1914 framework, the Python script `histPlots_Stats.py` from the original tool

¹⁹¹⁵ remains in the directory. This script is essentially unchanged from the ver-
¹⁹¹⁶ sion developed by Ceccato et al. and can be used in the future if the need
¹⁹¹⁷ arises.

¹⁹¹⁸ 4.2.3 Phase 3: Invariant Inference and Analysis

¹⁹¹⁹ The phase of invariant inference and analysis has undergone a redesign
¹⁹²⁰ and improvement to offer the user a more comprehensive and easier ap-
¹⁹²¹ proach to identify invariants. This has been achieved through the applica-
¹⁹²² tion of new criteria to analyze and reorganize the Daikon analysis results.
¹⁹²³ The outcome of this is a more compact presentation of information that
¹⁹²⁴ highlights the possible relationships among invariants.
¹⁹²⁵ The new design not only enables the identification of undiscovered aspects
¹⁹²⁶ of the system behavior but also confirms the hypotheses made during the
¹⁹²⁷ earlier stages of analysis. This step is now semi-automated, unlike before,
¹⁹²⁸ and allows for the analysis of invariants on individual actuator states and
¹⁹²⁹ their combinations.

¹⁹³⁰ 4.2.3.1 Revised Daikon Output

¹⁹³¹ To streamline the process of identifying invariants quickly and effi-
¹⁹³² ciently, it is necessary to revise the output generated by standard Daikon
¹⁹³³ analysis. The goal is to create a more compact and readable format for the
¹⁹³⁴ output.

¹⁹³⁵
¹⁹³⁶ The current Daikon results basically consist of three sections, referred as
¹⁹³⁷ *program point sections* [64]:

- ¹⁹³⁸ 1. the first section containing generic invariants, i.e., valid regardless of
¹⁹³⁹ whether a condition is specified for the analysis;
- ¹⁹⁴⁰ 2. the second section containing invariants obtained by specifying a
¹⁹⁴¹ condition for the analysis in the *.spinfo* file, if any;

1942 3. a third section containing the invariants that are obtained from the
 1943 negation of the condition potentially specified in the *.spinfo* file.

1944 In each section only a single invariant per row is shown, without relat-
 1945 ing it in any way to the others: this makes it difficult to identify significant
 1946 invariants and any invariant chain that might provide much more infor-
 1947 mation about the behavior of the system than the single invariant.

1948 A brief example of the structure and format of this output related to PLC1
 1949 of the iTrust SWaT system is shown in Listing 4.5, where a condition was
 1950 specified on the measurement P1_LIT101 and on actuator P1_MV101:

```

1951     aprogram.point:::POINT
1952     P1_P102 == prev_P1_P102
1953     P1_FIT101 >= 0.0
1954     P1_MV101 one of { 0.0, 1.0, 2.0 }
1955     P1_P101 one of { 1.0, 2.0 }
1956     P1_P102 == 1.0
1957     max_P1_LIT101 == 816.0
1958     min_P1_LIT101 == 489.0
1959     slope_P1_LIT101 one of { -1.0, 0.0, 1.0 }
1960     [...]
1961     P1_LIT101 > P1_MV101
1962     P1_LIT101 > P1_P101
1963     P1_LIT101 > P1_P102
1964     P1_LIT101 < max_P1_LIT101
1965     P1_LIT101 > min_P1_LIT101
1966     [...]
1967     P1_MV101 < min_P1_LIT101
1968     P1_MV101 < trend_P1_LIT101
1969     P1_P101 >= P1_P102
1970     P1_P101 < max_P1_LIT101
1971     [...]
1972     =====
1973     aprogram.point:::POINT; condition="P1_MV101 == 2.0 &&
1974     ↪ P1_LIT101 < max_P1_LIT101 - 16 && P1_LIT101 >
1975     ↪ min_P1_LIT101 + 15"
1976     P1_MV101 == prev_P1_MV101
1977     P1_P102 == slope_P1_LIT101
1978     P1_MV101 == 2.0

```

```

1979      P1_FIT101 > P1_MV101
1980      P1_FIT101 > P1_P101
1981      P1_FIT101 > P1_P102
1982      P1_FIT101 > prev_P1_P101
1983      P1_MV101 >= P1_P101
1984      P1_MV101 >= prev_P1_P101
1985      P1_P101 <= prev_P1_P101
1986      =====
1987      aprogram.point::POINT; condition="not(P1_MV101 == 2.0 &&
1988          ↪ P1_LIT101 < max_P1_LIT101 - 16 && P1_LIT101 >
1989          ↪ min_P1_LIT101 + 15)"
1990      P1_P101 >= prev_P1_P101
1991      Exiting Daikon.

```

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

1992 In the presented framework, the output is simplified to **two sections**:
 1993 the *general section* and the section related to the *user-specified condition*. The
 1994 section related to the negated condition is eliminated as it is not relevant in
 1995 this context and could lead to potential misinterpretation. Moreover, the
 1996 relationships between invariants will be emphasized by utilizing **transi-**
 1997 **tive closures**: transitive closure of a relation R is another relation, typically
 1998 denoted R^+ that adds to R all those elements that, while not necessarily
 1999 related directly to each other, can be reached by a *chain* of elements related
 2000 to each other. In other words, the transitive closure of R is the smallest (in
 2001 set theory sense) transitive relation R such that $R \subset R^+$ [65].

2002 To implement the transitive closure of the invariants generated by Daikon,
 2003 we initially categorized the invariants in each section by type based on
 2004 their mathematical relation ($==$, $>$, $<$, $>=$, $<=$, $!=$), excluding any in-
 2005 variant related to additional attributes except for the slope. For each type
 2006 of invariant, we constructed a **graph** using the NetworkX library, where
 2007 registers were represented as nodes, and arcs were created to connect reg-
 2008 isters that shared a common endpoint in the invariant, applying the transi-
 2009 tive property. To reconstruct the individual invariant chains, we employed
 2010 a straightforward approach known as *Depth-first Search* (DFS) on each of
 2011 the graphs. This method allowed us to traverse the graphs and obtain the

2012 desired outcome, identifying the invariant chains. Figure 4.7 shows some
 2013 examples of these graphs:

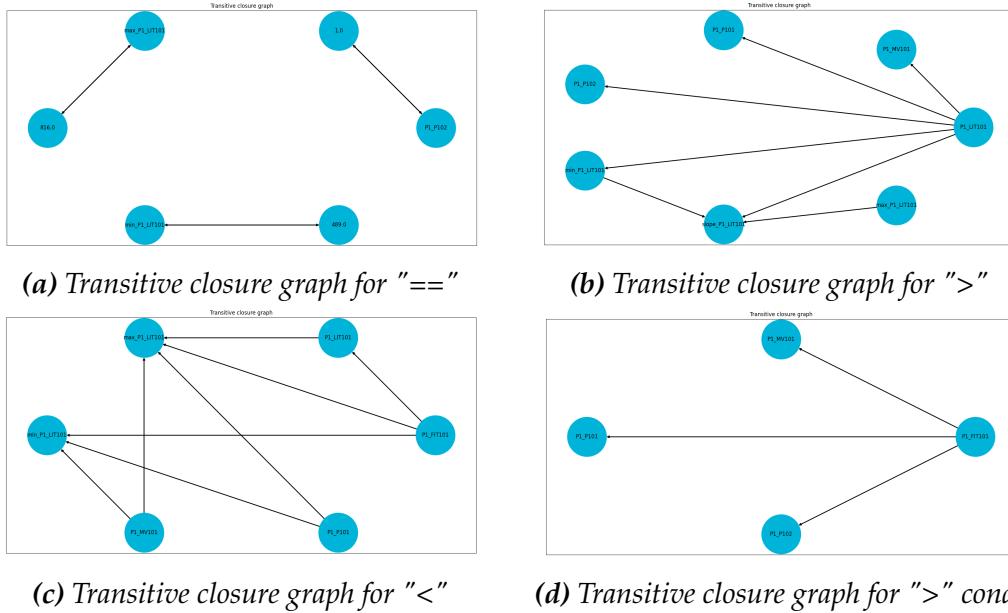


Figure 4.7: Example of transitive closure graphs for invariants in PLC1 of iTrust SWaT system

2014 At the end of this process, still applied to PLC1 of the iTrust SWaT sys-
 2015 tem and with the same analysis condition, we get the following complete
 2016 output:

```

2017 1 =====
2018 2 Generic
2019 3 =====
2020 4 P1_MV101 one of { 0.0, 1.0, 2.0 }
2021 5 P1_P101 one of { 1.0, 2.0 }
2022 6 slope_P1_LIT101 one of { -1.0, 0.0, 1.0 }
2023 7 P1_FIT101 != P1_P101, P1_P102
2024 8 P1_P102 == 1.0
2025 9 max_P1_LIT101 == 816.0
2026 10 min_P1_LIT101 == 489.0
2027 11 P1_LIT101 > P1_MV101
2028 12 P1_LIT101 > P1_P101
2029 13 P1_LIT101 > P1_P102
  
```

```

2030 14 P1_LIT101 > min_P1_LIT101 > slope_P1_LIT101
2031 15 P1_FIT101 >= 0.0
2032 16 P1_P101 >= P1_P102 >= slope_P1_LIT101
2033 17 =====
2034 18 =====
2035 19 P1_MV101 == 2.0 && P1_LIT101 < max_P1_LIT101 - 16 &&
2036   ↢ P1_LIT101 > min_P1_LIT101 + 15
2037 20 =====
2038 21 slope_P1_LIT101 == P1_P102
2039 22 P1_MV101 == 2.0
2040 23 P1_FIT101 > P1_MV101
2041 24 P1_FIT101 > P1_P101
2042 25 P1_FIT101 > P1_P102
2043 26 P1_MV101 >= P1_P101

```

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

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

4.2.3.2 Types of Analysis

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

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

The second type of analysis considers the current system configuration and examines the **actual states of the actuators**. This analysis takes into account the interplay and combined effects of multiple actuators on the selected measurement. By incorporating the real-time states of the actu-

2061 ators, this semi-automated analysis offers a more accurate assessment of
2062 the system behavior.

2063 These two types of analysis will be handled by the Python script
2064 `daikonAnalysis.py`, contained in the default directory `$(project_dir)/daikon`.
2065 The script accepts the following command-line arguments:

- 2066 • **-f or --filename**: specifies the enriched dataset, in CSV format, from
2067 which to read the data. The dataset must be located within the di-
2068 rectory containing the enriched datasets specified in the `config.ini` file
2069 (by default `$(project_dir)/daikon/Daikon_Invariants`);
- 2070 • **-s or --simpleanalysis**: performs the analysis on the states of indi-
2071 vidual actuators;
- 2072 • **-c or --customanalysis**: performs the analysis on combinations of ac-
2073 tual states of the actuators;
- 2074 • **-u or --uppermargin**: defines a percentage margin on the maximum
2075 value of the measurement;
- 2076 • **-l or --lowermargin**: defines a percentage margin on the minimum
2077 value of the measurement;

2078 The selection of one or both types of analysis is possible. Additionally,
2079 the last two parameters can be used to set a condition on the value of the
2080 measurement. This condition is designed to bypass the transient periods
2081 that occur during actuator state changes and the actual trend changes at
2082 the maximum and minimum values of the measurement.

2083 This approach proves particularly beneficial for the first type of analysis,
2084 as it enables more accurate data on the trends of the measurement. By
2085 excluding the transient periods, the analysis can focus on the stable and
2086 meaningful trends, providing improved insights into the behavior of the
2087 system.

2088 **Analysis on single actuator states** Analysis on the states of individual
 2089 actuators is the simplest: after the user is prompted to input the measure-
 2090 ment, chosen from a list of likely available measurements, the script rec-
 2091ognizes the likely actuators and the relative states of each, using the same
 2092 mixed Daikon/Pandas technique adopted in the brief analysis during the
 2093 pre-processing phase.

2094 For each actuator and each state it assumes, a single Daikon analysis is
 2095 performed, eventually placing the condition on the maximum and mini-
 2096 mum level of the measurement.

2097 The result of these analyses are saved in the form of text files in a directory
 2098 having the name corresponding to the analyzed actuator and contained in
 2099 the default parent directory `$(project_dir)/daikon/Daikon_Invariants/results`:
 2100 each file generated by the analysis is identified by the name of the actuator,
 2101 the state and the condition, if any, on the measurement (see Figure 4.8).

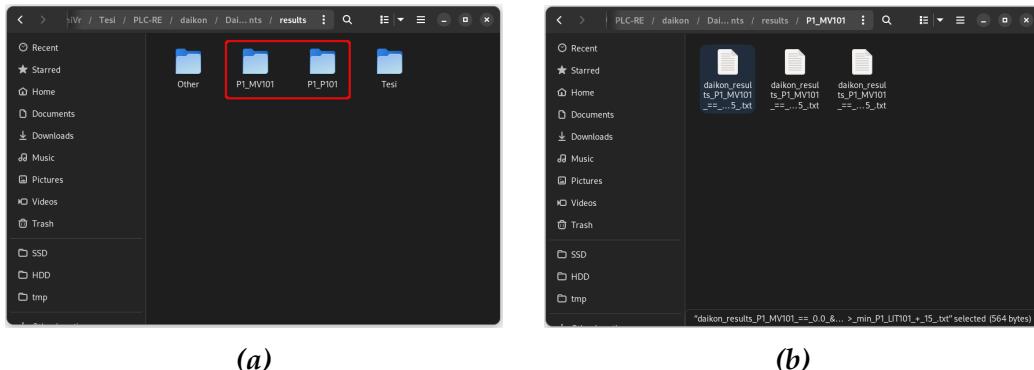


Figure 4.8: Directory (a) and outcome files (b) for the single actuator states analysis

2102 Listing 4.6 provides an illustrative example of the analysis results. In
 2103 this case, we focus on the actuator `P1_MV101` in state 2. By examining the
 2104 generic invariants, we can deduce that `P1_P101` is a probable actuator with
 2105 binary values, as indicated in line 5. However, upon closer inspection
 2106 in line 8, we discover that `P1_P102` consistently maintains a value of 1
 2107 throughout, thus confirming its role as a spare actuator rather than a set-
 2108 point. Furthermore, lines 9 and 10 present the maximum and minimum
 2109 values achieved by `P1_LIT101`, the register presumed to be connected to

2110 the tank.

2111 The condition-generated invariants provide the most interesting insights.

2112 In particular, at line 21, we observe that when P1_MV101 is set to 2, the
2113 slope of P1_LIT101 is 1, indicating an increasing trend. This confirms our
2114 previous assumption that P1_MV101 represents the actuator's ON state and
2115 is responsible for filling the tank (P1_LIT101).

2116 Furthermore, at line 23, we discover that P1_FIT101 is greater than P1_MV101.

2117 This implies that when the actuator assumes the value 2, the sensor P1_FIT101
2118 registers a measurement. In contrast, when P1_MV101 is 1, P1_FIT101 re-
2119 mains at 0, signifying no recorded readings. This finding reinforces our
2120 initial hypothesis that P1_FIT101 may serve as a pressure or flow sensor.

2121 **Analysis of the Current System Configuration** The analysis of the cur-
2122 rent system configuration based on the actual states of the actuators is
2123 more complex, but at the same time offers more interesting outcomes as
2124 it provides better evidence of actuator behavior in relation to the selected
2125 measurement.

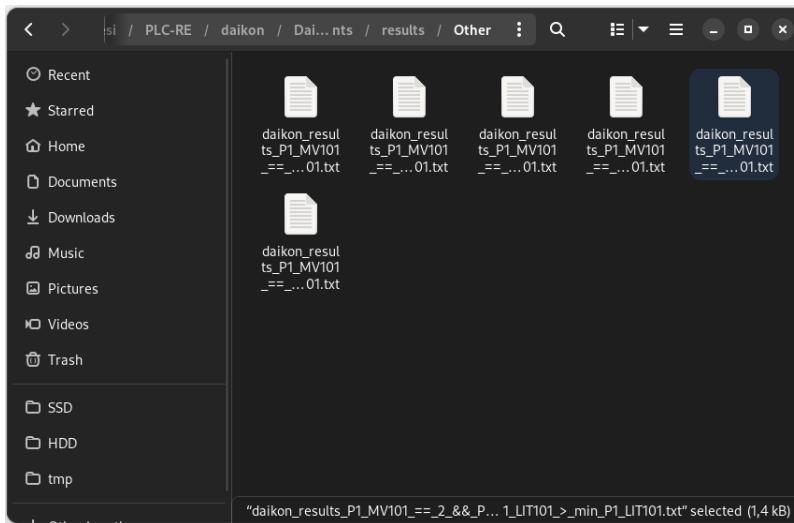


Figure 4.9: Daikon outcome files for system configuration analysis. Each file represents a single system state

2126 For this analysis, the script automatically identifies the configurations
2127 of the actuators that represent different system states (e.g., P1_MV101 ==

2128 2, P1_P101 == 1). Daikon analysis is then performed for each of these
 2129 configurations. The user is prompted to select the measurement attribute
 2130 and, if desired, specific actuators for studying their configurations. If no
 2131 actuators are selected, all previously detected actuators will be considered.
 2132 The analysis results are saved in text format within a designated directory,
 2133 located alongside the previous analysis outputs. The filenames of these
 2134 result files follow a specific naming convention based on the analysis rule
 2135 applied (see Figure 4.9).

2136

2137 An example of the obtained outcomes can be seen in Listing 4.7:

```

2138 1 =====
2139 2 Generic
2140 3 =====
2141 4 P1_MV101 <= P1_P101 ==> P1_FIT101 >= 0.0
2142 5 P1_MV101 <= P1_P101 ==> P1_MV101 one of { 0.0, 1.0, 2.0
2143 6   ↪ }
2144 7 P1_MV101 <= P1_P101 ==> P1_P101 one of { 1.0, 2.0 }
2145 8 P1_MV101 <= P1_P101 ==> slope_P1_LIT101 one of { -1.0,
2146 9   ↪ 0.0, 1.0 }
2147 10 P1_MV101 > P1_P101 ==> P1_FIT101 > P1_MV101
2148 11 P1_MV101 > P1_P101 ==> P1_FIT101 > P1_P101
2149 12 P1_MV101 > P1_P101 ==> P1_FIT101 > P1_P102
2150 13 P1_MV101 > P1_P101 ==> P1_FIT101 > slope_P1_LIT101
2151 14 P1_MV101 > P1_P101 ==> P1_MV101 == 2.0
2152 15 P1_MV101 > P1_P101 ==> P1_MV101 > P1_P102
2153 16 P1_MV101 > P1_P101 ==> P1_MV101 > slope_P1_LIT101
2154 17 P1_MV101 > P1_P101 ==> P1_P101 == 1.0
2155 18 P1_MV101 > P1_P101 ==> P1_P101 == P1_P102
2156 19 P1_MV101 > P1_P101 ==> P1_P101 == slope_P1_LIT101
2157 20 P1_MV101 > P1_P101 ==> slope_P1_LIT101 == 1.0
2158 21 [...]
2159 22 =====
2160 23 P1_MV101 == 2 && P1_P101 == 1 && P1_LIT101 < max_P1_LIT101
2161 24   ↪ && P1_LIT101 > min_P1_LIT101
2162 25 =====
2163 26 slope_P1_LIT101 == P1_P102 == P1_P101 == 1.0

```

```
2165 25 P1_MV101 == 2.0  
2166 26 P1_FIT101 > P1_MV101  
2167 27 P1_FIT101 > P1_P101
```

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

2168 In contrast to Listing 4.6, the current analysis reveals the presence of im-
2169 plications in the general invariants section, which were previously absent
2170 (remaining generic invariants omitted for brevity). These implications of-
2171 fer valuable insights, as demonstrated in this case. For instance, the invari-
2172 ant stated in line 18 informs us that if the value of $P1_MV101$ is greater than
2173 $P1_P101$, then the slope's value is 1, indicating an increasing tank level.
2174 This finding further corroborates our initial understanding that the state
2175 $P1_MV101 == 2$ represents the ON state for the actuator responsible for fill-
2176 ing the tank, namely $P1_LIT101$. Upon comparing the results of the other
2177 analyses, we will uncover that $P1_P101$ serves the purpose of emptying the
2178 tank, with its ON and OFF states denoted as 2 and 1, respectively.
2179 Furthermore, the invariant presented in line 8 reveals that when $P1_MV101$
2180 is in state 2 and $P1_P101$ is in state 1, the value of $P1_FIT101$ is greater
2181 than 2. Consequently, it can be inferred that the associated sensor is mea-
2182 suring something in relation to the tank represented by $P1_LIT101$.
2183 The aforementioned observations are further supported by the invariants
2184 associated with the analysis condition. As indicated in line 24, the slope is
2185 indeed equal to 1, confirming an upward trend. Additionally, at line 26, it
2186 is evident that $P1_FIT101$ assumes values greater than 2 when $P1_MV101$ is
2187 equal to 2.

2188 **Refining the Analysis** In certain situations, the outcomes provided by
2189 the semi-automated analyses may not meet the user's expectations. For
2190 instance, the clarity of the slope value may be insufficient, or the user may
2191 wish to delve deeper into a specific aspect of the system to uncover ad-
2192 dditional invariants that were not previously identified. In such cases, the
2193 user has the option to conduct a more targeted and specific invariant anal-
2194 ysis using the Python script `runDaikon.py`, which enables precise investi-

2195 gations of the system.

2196 The script, located in the default directory \$(project_dir)/daikon, can be
2197 executed with three command-line parameters:

2198 • **-f or --filename:** specifies the CSV format *enriched* dataset to read
2199 data from. Even in this case, the dataset must be located within the
2200 directory containing the enriched datasets;

2201 • **-c or --condition:** specifies the condition for the analysis, which will
2202 be automatically overwritten in the .sinfo file. It is possible to spec-
2203 ify more than one condition, but it is strongly recommended to use
2204 the logical operator && to avoid undesired behaviors in Daikon out-
2205 comes;

2206 • **-r or --register:** specifies the directory where to save the text file with
2207 the outcomes.

2208 During the execution of this analysis, a single output file is generated, con-
2209 taining the discovered invariants. The user can specify the directory where
2210 this file should be saved using the -r command-line option. By default, the
2211 file is stored in the directory \$(project_dir)/daikon/Daikon_Invariants/results.
2212 The output file serves as a record of the identified invariants and can be
2213 examined at a later time.

2214 In conclusion, the integration of these two analysis types, along with
2215 the ability to conduct more refined analysis in the future and the enhanced
2216 output format of Daikon, significantly enhances the completeness, clarity,
2217 and effectiveness of this stage compared to the previous framework.

2218 **4.2.4 Phase 4: Business Process Analysis**

2219 **4.2.5 Phase 5: Network Analysis**

Case study: the iTrust SWaT System

2220 **H**AVING introduced the innovative framework and highlighted its po-
2221 tential in the preceding chapter, we now turn our attention to the
2222 case study where we will apply this framework. As previously mentioned
2223 in Chapter 4 and demonstrated through various examples in the same
2224 chapter, our focus will be on the **iTrust SWaT system** [36], developed by
2225 the iTrust – Center for Research in Cyber Security of University of Singa-
2226 pore for Technology and Design [30]. The acronym SWaT represents *Secure*
2227 *Water Treatment*.

2228 The iTrust SWaT system is a testbed that replicates on a small scale
2229 a real water treatment plant arises to support research in the area of cy-
2230 ber security of industrial control systems and has been operational since
2231 March 2015: it is still being used by students at the University of Singapore
2232 for educational and training purposes and is available to organizations to
2233 train their operators on cyber physical incidents.

2234 5.1 Architecture

2235 In contrast to the virtualized testbed discussed in Section 3.2.1 by Cec-
2236 cato et al., the iTrust SWaT system is composed entirely of physical hard-
2237 ware components. It encompasses various elements, starting from field

2238 devices and extending to PLCs, HMI, SCADA workstations, and the SCADA
2239 server (also referred to as the *historian*). The historian is responsible for
2240 recording data from field devices for further analysis. In the upcoming
2241 sections, we will delve deeper into the architecture of the physical process
2242 and the communication network.

2243 **5.1.1 Physical Process**

2244 The physical process of the SWaT consists of six stages, denoted P1
2245 through P6. These stages are [66][67]:

- 2246 1. **taking in raw water:** feeds unfiltered water into the system
- 2247 2. **chemical dosing:** adds chemicals to water useful for initial pretreat-
2248 ment
- 2249 3. **Ultra Filtration (UF) system:** the water is filtered through a semi-
2250 permeable membrane (ultrafiltration membrane) using the liquid pres-
2251 sure, effectively capturing impurities and suspended solids, as well
2252 as removing bacteria, viruses, and other pathogens present in the
2253 water.
- 2254 4. **dechlorination:** removes residual chlorine from disinfected water
2255 using ultraviolet lamps
- 2256 5. **Reverse Osmosis (RO):** performs further filtration of the water
- 2257 6. **backwash process:** cleans the membranes in UF using the water pro-
2258 duced by RO

2259 Figure 5.1 shows a graphical representation of the architecture and the six
2260 stages of the SWaT system.

2261 The SWaT system incorporates an array of sensors that play a crucial
2262 role in monitoring the system's operations and ensuring their safe. These
2263 sensors are responsible for continuously collecting data and providing
2264 valuable insights into the functioning of the system. These sensors are:

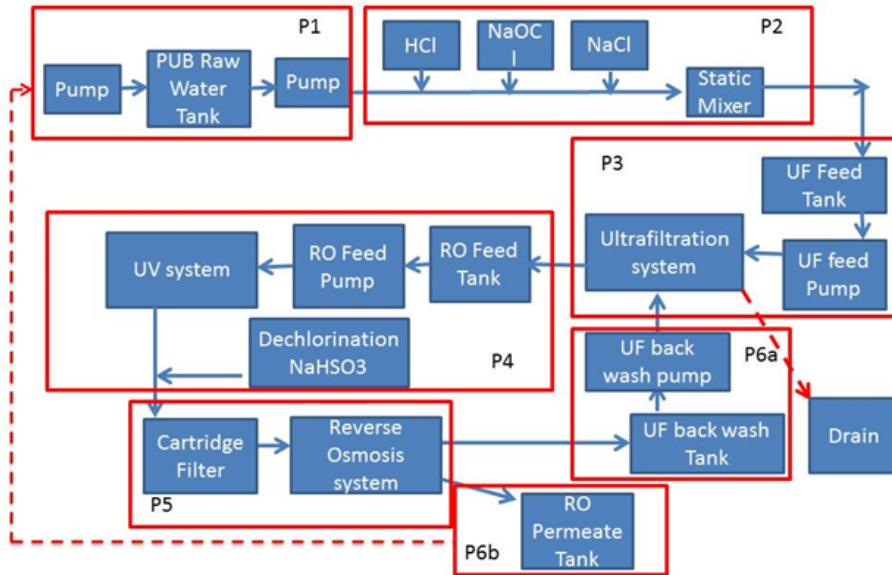


Figure 5.1: SWaT architecture

- Level Indication Transmitter (measured in mm)
- Flow Indication Transmitter (m³/hr)
- Analyser Indicator Transmitter
 - Conductivity ($\mu\text{S}/\text{cm}$)
 - pH
 - Oxidation Reduction Potential (mV)
- Differential Pressure Indicator Transmitter (kPa)
- Pressure Indicator Transmitter (kPa)

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

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

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	DPT-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

2280 5.1.2 Control and Communication Network

2281 The SWaT system's network architecture follows the principles of lay-
 2282 ering and zoning, which enable segmentation and control of traffic within
 2283 the network.

2284

2285 Five layers are present starting from the highest to the lowest:

- 2286 • **Layer 3.5** – Demilitarized Zone (DMZ)
- 2287 • **Layer 3** – Operation Management (Historian)
- 2288 • **Layer 2** – Supervisory Control (Touch Panel, Engineering Worksta-
 2289 tion, HMI Control Clients)
- 2290 • **Layer 1** – Plant Control Network (PLCs) (Star Network)
- 2291 • **Layer 0** – Process (Actuator/Sensors and Input/output modules)
 2292 (Ring Network)

2293 PLCs at Layer 1 communicate with their respective sensors and actuators
 2294 at Layer 0 through a conventional ring network topology based on Ether-
 2295 net/IP, to ensure that the system can tolerate the loss of a single link

2296 without any adverse impact on data or control functionality.
 2297 PLCs between the different process stages at Layer 1 communicate with
 2298 each other through a star network topology using the CIP protocol on Eth-
 2299 erNet/IP, previously discussed in Section 2.2.6.3.

2300 Regarding zoning, the SWaT system is divided into three zones, each
 2301 containing one or more layers. These zones are, in descending order of
 2302 security level:

- 2303 • **Plant Control Network, or Control System:** includes layers from 0
 2304 to 2
- 2305 • **DMZ:** includes Layer 3.5
- 2306 • **Plant Network:** includes Layer 3

2307 Figure 5.3 provides a clearer visualization of the zoning and layer division
 2308 within the network architecture of the SWaT system. This diagram high-
 2309 lights the distinct zones and their corresponding layers, offering a com-
 2310 prehensive overview of the system's network structure.

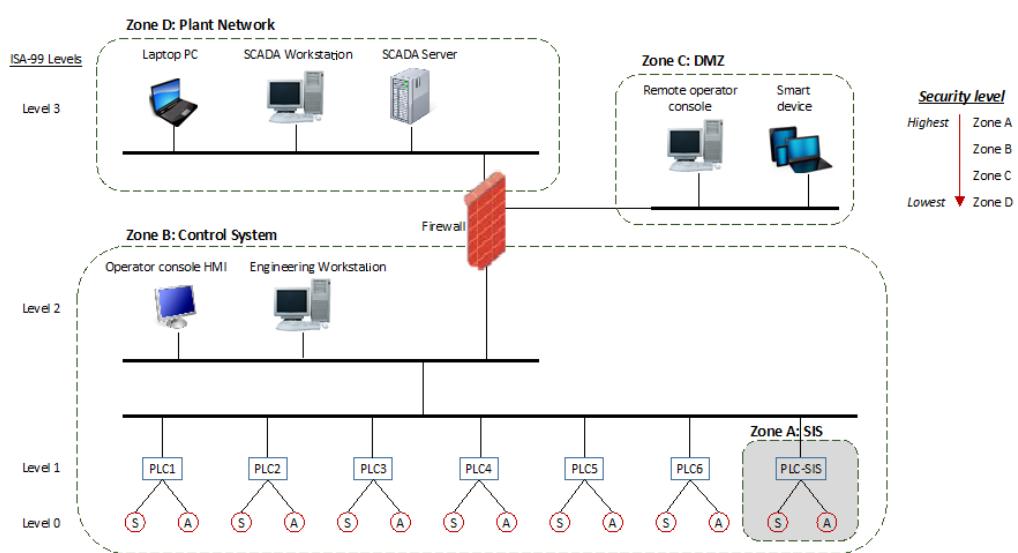


Figure 5.3: SWaT network architecture

2311 A specific IP address is associated with each device: in Table 5.1 we
2312 report the addresses for the PLCs, historian, and Touch Panel in the *Plant*
2313 *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

2314 5.2 Datasets

2315 To facilitate the study and testing of technologies related to cyber security
2316 in Industrial Control Systems and critical infrastructure, iTrust offers
2317 researchers worldwide the opportunity to access a range of datasets [68].
2318 These datasets consist of a collection of data obtained from the SWaT sys-
2319 tem, encompassing information on both the physical processes and net-
2320 work communications. The data is organized into different years, and
2321 researchers can request access to these datasets for their analysis and ex-
2322 perimentation purposes.

2323 **Physical Process Datasets** The datasets containing information about
2324 the physical processes are provided in CSV format files. These files en-
2325 compass data collected during different time intervals, which can vary
2326 from a few hours to entire days. The granularity of the data is typically

2327 at a one-second interval, although there may be some exceptions. The col-
2328 lected data primarily consists of timestamped sensor measurements and
2329 actuator status values for each PLC, describing the physical properties of
2330 the testbed in operational mode.

2331 **Network Communications Datasets** Network communications are typi-
2332 cally available in the form of *Packet Capture* format (PCAP) files. These files
2333 contain captures of communication network traffic, allowing researchers
2334 to analyze and examine the network interactions. In some instances, CSV
2335 files are provided instead of PCAP files, featuring different characteristics
2336 for the collected data.

2337 5.2.1 Our Case Study: the 2015 Dataset

2338 The dataset selected as a case study to apply the framework discussed
2339 in the previous chapter is specifically the dataset from the year 2015 [69].
2340 The main reason for this choice is the unique characteristics found in the
2341 physical process dataset that are not present in datasets from subsequent
2342 years.

2343 **Physical Process Data** The data collection process lasted 11 consecutive
2344 days, 24 hours per day. During the first 7 days, the system operated nor-
2345 mally without any recorded attacks. However, attacks were observed dur-
2346 ing the remaining 4 days. The collected data reflects the impact of these
2347 attacks, leading to the creation of two separate CSV files: one containing
2348 the recorded data of SWaT during the system's regular operations, and
2349 the other containing data recorded during the days of the attacks. To en-
2350 sure accurate information about the system, the dataset pertaining to the
2351 normal operations, which spans seven days, was chosen for analysis.

2352 Data collection occurs at a frequency of one data point per second,
2353 with the assumption that significant attacks cannot occur within a shorter
2354 time frame. Additionally, the firmware of the PLCs remains unchanged
2355 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).

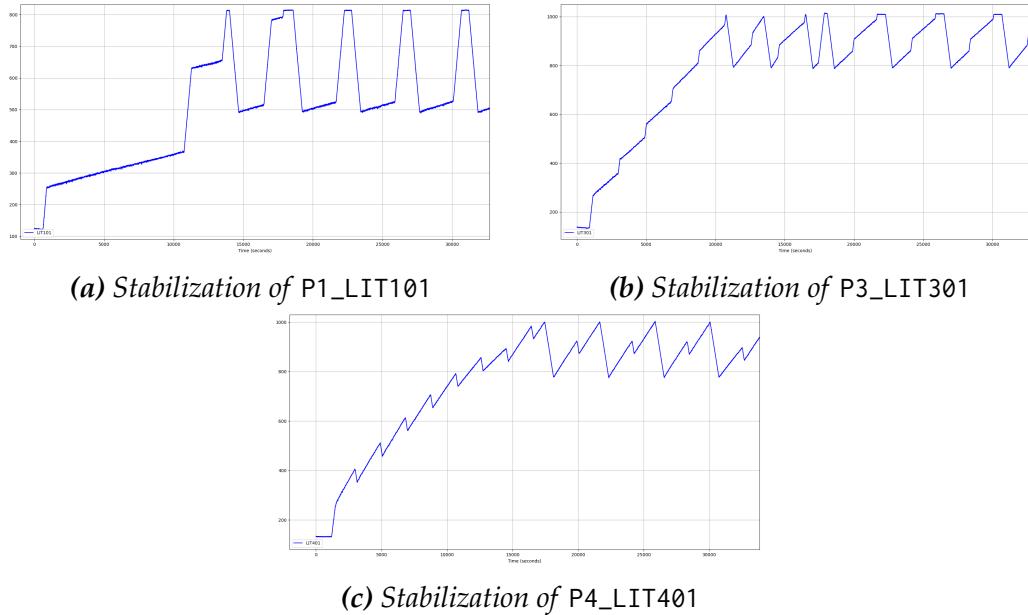


Figure 5.4: SWaT stabilization

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

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

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

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

Table 5.2: SWaT network traffic data

2366 Unfortunately, the data provided are partial as it only includes read-
 2367 ings from sensors and does not include information on actuator status.

2368
 2369 Do not mislead by the word *Modbus* in the captured data fields: this is
 2370 most likely a feature of the sniffing appliance, which may have encapsu-
 2371 lated EtherNet/IP-CIP protocol packets in Modbus frames, as described
 2372 by T. A. Snide in [70].

2373 To provide evidence that the protocol being used is CIP over EtherNet/IP,
 2374 we can examine the "Source / Destination port" field in the captured data.
 2375 The presence of TCP port 44818 as the destination port aligns with the
 2376 TCP port used for transporting **explicit messages** in the EtherNet/IP pro-
 2377 tocol, as explained in Section 2.2.6.2. Additionally, the "Application name",
 2378 "Modbus Function Code" and "Modbus Function Description" fields fur-
 2379 ther support this conclusion. The "Application name" explicitly states
 2380 "CIP_read_tag_service," while the "Modbus Function Code" field contains
 2381 a decimal value of 76, which does not correspond to any known Modbus
 2382 Function Calls. When converted to hexadecimal, this value is 4C, match-
 2383 ing the CIP protocol read tag request code as indicated in the "Modbus
 2384 Function Description" field.

2385 In summary, the datasets for subsequent years were not selected due

2386 to certain limitations. In the case of the year 2017, the physical system
2387 data had a wide granularity, resulting in significant information loss. Ad-
2388 ditionally, in some cases, the datasets were considered "spurious" as there
2389 was no clear differentiation between data obtained during normal SWaT
2390 operations and data associated with attacks.

2391 Regarding network traffic data, there were instances where the data was
2392 either missing or fragmented into large PCAP files weighing several gi-
2393 gabytes (more than 6 GB each!), despite containing only a few minutes'
2394 worth of data. This made it impractical to manage such files given the
2395 available resources.

2396 Despite their large quantity, the CSV files associated with network traffic
2397 for the year 2015 are comparatively smaller in size, just slightly over 100
2398 MB each. These files cover a more extensive time period, which facilitates
2399 easier management and analysis. Prioritizing the physical process dataset
2400 as crucial, I have selected the year 2015 as a case study, even though the
2401 network data may be incomplete.

Our framework at work: reverse engineering of the SWaT system

6.1 Pre-processing

6.2 Graph Analysis

6.2.1 Conjectures About the System

6.3 Invariants Analysis

6.3.1 Actuators Detection

6.3.2 Daikon Analysis and Results Comparing

6.4 Extra information on the Physics

6.5 Business Process Analysis

Chapter

7

Conclusions

7.1 Discussions

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	Slope comparison with granularity 30 (a), 60 (b) and 120 seconds (c)	68
4.2	Savitzky-Golay filter (blue line), STL decomposition (green) and RDP algorithm (orange) comparison	70
4.3	Slope after the application of the STL decomposition	71
4.4	The new slope representation (green line) and the smoothed measurement data obtaind with the STL decomposition (red)	72
4.5	Plotting registers on the same y-axis	77
4.6	Example of the new graph analysis	78
4.7	Example of transitive closure graphs for invariants in PLC1 of iTrust SWaT system	85
4.8	Directory (a) and outcome files (b) for the single actuator states analysis	88
4.9	Daikon outcome files for system configuration analysis. Each file represents a single system state	89
5.1	SWaT architecture	95
5.2	Sensors and actuators associated with each PLC	96
5.3	SWaT network architecture	97
5.4	SWaT stabilization	100

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	98
5.2	SWaT network traffic data	101

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	62
4.3	<i>config.ini</i> parameters for dataset enriching	64
4.4	Example of preliminar system analysis	73
4.5	Standard Daikon output for PLC1 of the iTrust SWaT system	83
4.6	Revised Daikon output with transitive closures for PLC1 of the iTrust SWaT system	85
4.7	Daikon outcomes for the system configuration P1_MV101 == 2, P1_P101 == 1 on P1_LIT101	90

References

- [1] NIST. *ICS defintion*. URL: https://csrc.nist.gov/glossary/term/industrial_control_system (visited on 2023-04-06).
- [2] Wikipedia. *Stuxnet*. URL: <https://en.wikipedia.org/wiki/Stuxnet> (visited on 2023-05-18).
- [3] Wikipedia. *Industroyer*. URL: <https://en.wikipedia.org/wiki/Industroyer> (visited on 2023-05-18).
- [4] F. Robles and N. Perlroth. “‘Dangerous Stuff’: Hackers Tried to Poison Water Supply of Florida Town”. In: *The New York Times* (2021-02-08). URL: <https://www.nytimes.com/2021/02/08/us/oldsmar-florida-water-supply-hack.html> (visited on 2023-05-18).
- [5] A. Akbarzadeh. “Dependency based risk analysis in Cyber-Physical Systems”. PhD thesis. Norwegian University of Science, Technology - Faculty of Information Technology, and Electrical Engineering, 2023. ISBN: 978-82-326-6986-8 (electronic version).
- [6] G. M. Makrakis, C. Kolias, G. Kambourakis, C. Rieger, and J. Benjamin. “Industrial and Critical Infrastructure Security: Technical Analysis of Real-Life Security Incidents”. In: *IEEE Access* (2021-12-06). DOI: <https://dx.doi.org/10.1109/ACCESS.2021.3133348>.
- [7] NIST. *PLC defintion*. URL: https://csrc.nist.gov/glossary/term/programmable_logic_controller (visited on 2023-04-06).

- [8] W. Bolton. *Programmable Logic Controllers, 6th edition*. Newnes, 2015, pp. 7–9.
- [9] M. Ceccato, Y. Driouich, R. Lanotte, M. Lucchese, and M. Merro. “Towards Reverse Engineering of Industrial Physical Processes”. In: CPS4CIP@ESORICSAt 2022 (Copenhagen, Denmark, Sept. 30, 2022). 2022, 273–290. DOI: https://doi.org/10.1007/978-3-031-25460-4_15.
- [10] H. S. G. Pussewalage, P. S. Ranaweera, and V. Oleshchuk. “PLC security and critical infrastructure protection”. In: 2013 IEEE 8th International Conference on Industrial and Information Systems (Dec. 2013). 2013. DOI: <https://dx.doi.org/10.1109/ICIInfs.2013.6731959>.
- [11] Wikipedia. *Remote Terminal Unit*. URL: https://en.wikipedia.org/wiki/Remote_terminal_unit (visited on 2023-04-08).
- [12] What is SCADA? URL: <https://www.inductiveautomation.com/resources/article/what-is-scada> (visited on 2023-04-05).
- [13] NIST. *HMI defintion*. URL: https://csrc.nist.gov/glossary/term/human_machine_interface (visited on 2023-04-11).
- [14] Zscaler. *What Is the Purdue Model for ICS Security?* URL: <https://www.zscaler.com/resources/security-terms-glossary/what-is-purdue-model-ics-security> (visited on 2023-05-17).
- [15] Schneider Electrics. *What is Modbus and how does it work?* URL: <https://www.se.com/us/en/faqs/FA168406/> (visited on 2023-04-13).
- [16] O. Morando. *Hacking: Modbus - Cyber security OT/ICS*. URL: <https://blog.omarmorando.com/hacking/ot-ics-hacking/hacking-modbus> (visited on 2023-04-15).
- [17] Modbus.org. “MODBUS/TCP Security”. In: pp. 7–8. URL: https://modbus.org/docs/MB-TCP-Security-v21_2018-07-24.pdf (visited on 2023-04-16).

- [18] ODVA Inc. "EtherNet/IP - CIP on Ethernet Technology". In: URL: https://www.odva.org/wp-content/uploads/2021/05/PUB00138R7_Tech-Series-EtherNetIP.pdf (visited on 2023-04-18).
- [19] *Odva, Inc.* URL: <https://www.odva.org> (visited on 2023-04-21).
- [20] *Introduction to EtherNet/IP Technology*. URL: https://scadahacker.com/library/Documents/ICS_Protocols/Intro%20to%20EthernetIP%20Technology.pdf (visited on 2023-04-19).
- [21] ODVA Inc. *Common Industrial Protocol*. URL: <https://www.odva.org/technology-standards/key-technologies/common-industrial-protocol-cip/> (visited on 2022-12-26).
- [22] Wikipedia. *Common Industrial Protocol*. URL: https://en.wikipedia.org/wiki/Common_Industrial_Protocol (visited on 2023-04-21).
- [23] *What is the Common Industrial Protocol (CIP)?* URL: <https://www.motioncontroltips.com/what-is-the-common-industrial-protocol-cip/> (visited on 2023-04-21).
- [24] *CIP (Common Industrial Protocol): CIP messages, device types, implementation and security in CIP*. URL: <https://resources.infosecinstitute.com/topic/cip/> (visited on 2023-04-21).
- [25] B. Green, M. Krotofil, and A. Abbasi. "On the Significance of Process Comprehension for Conducting Targeted ICS Attacks". In: Workshop on Cyber-Physical Systems Security and PrivaCy (Nov. 2017). ACM, 2017, pp. 57–67. DOI: <https://doi.org/10.1145/3140241.3140254>.
- [26] A. Keliris and M. Maniatakos. "ICSREF: A Framework for Automated Reverse Engineering of Industrial Control Systems Binaries". In: 26th Annual Network and Distributed System Security Symposium (NDSS) (Feb. 2019). 2019. DOI: <https://doi.org/10.48550/arXiv.1812.03478>.
- [27] Wikipedia. *CODESYS*. URL: <https://en.wikipedia.org/wiki/CODESYS> (visited on 2023-05-18).

- [28] Y. Yuan, X. Tang, W. Zhou, W. Pan, X. Li, H. T. Zhang, H. Ding, and J. Goncalves. "Data driven discovery of cyber physical systems". In: *Nature Communications* 10 (2019-10-25). URL: <https://www.nature.com/articles/s41467-019-12490-1> (visited on 2023-04-20).
- [29] C. Feng, V.R. Palleti, A. Mathur, and D. Chana. "A 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-Douglas-Peucker_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-Modbus-Integration-Hanover-Fair_0408.pdf (visited on 2023-05-17).