

**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

This thesis focuses on enhancing the *process comprehension* of an Industrial Control System (ICS) by employing a dynamic black-box analysis approach to analyze the system's physical process. The proposed analysis methodology involves multiple steps and utilizes a custom tool capable of extracting valuable insights about the behavior of the industrial system from both physical process logs and network traffic logs. The tool and methodology will be validated through a case study based on a real-world scenario.

The ultimate goal of this analysis is to gain a deep understanding of the industrial system under examination, with the aim of uncovering as much knowledge as possible. By comprehensively studying the system's behavior and characteristics, potential attackers can develop targeted and covert attack strategies.



# Contents

<b>1</b>	<b>Introduction</b>	<b>1</b>
1.1	Contribution . . . . .	2
1.2	Outline . . . . .	3
<b>2</b>	<b>Background on Industrial Control Systems</b>	<b>5</b>
2.1	Industrial Control Systems Architecture . . . . .	7
2.2	Operational Technology Networks . . . . .	9
2.2.1	Field I/O Devices Layer . . . . .	9
2.2.2	Controller Network Layer . . . . .	10
2.2.2.1	Programmable Logic Controllers . . . . .	10
2.2.2.2	Remote Terminal Units . . . . .	14
2.2.3	Area Control Layer . . . . .	15
2.2.3.1	Supervisory Control And Data Acquisition . . . . .	15
2.2.3.2	Human-Machine Interface . . . . .	15
2.2.4	Operations/Control Layer . . . . .	16
2.2.5	Demilitarized Zone . . . . .	16
2.2.6	Industrial Protocols . . . . .	17
2.2.6.1	Modbus . . . . .	18
2.2.6.2	EtherNet/IP . . . . .	21
2.2.6.3	Common Industrial Protocol (CIP) . . . . .	23
<b>3</b>	<b>State of the Art</b>	<b>25</b>
3.1	Literature on Process Comprehension . . . . .	26

3.2	Ceccato et al.'s black-box dynamic analysis for water-tank systems . . . . .	28
3.2.1	Testbed . . . . .	29
3.2.2	Scanning of the System and Data Pre-processing . . . . .	33
3.2.3	Graphs and Statistical Analysis . . . . .	35
3.2.4	Invariant Inference and Analysis . . . . .	36
3.2.5	Business Process Mining and Analysis . . . . .	38
3.2.6	Application . . . . .	40
3.2.7	Limitations . . . . .	46
<b>4</b>	<b>Extending and Generalizing Ceccato et al.'s Framework</b>	<b>57</b>
4.1	The Proposed New Framework . . . . .	58
4.1.1	Framework Structure . . . . .	61
4.1.2	Python Libraries and External Tools . . . . .	62
4.2	Analysis Phases . . . . .	63
4.2.1	A Little Testbed: Stage 1 of iTrust SWaT System . . . . .	64
4.2.2	Phase 0: Network Analysis . . . . .	65
4.2.2.1	Extracting Data from PCAP Files . . . . .	66
4.2.2.2	Network Information . . . . .	68
4.2.3	Phase 1: Data Pre-processing . . . . .	71
4.2.3.1	Subsystem Selection . . . . .	72
4.2.3.2	Dataset Enrichment . . . . .	75
4.2.3.3	Datasets Merging . . . . .	82
4.2.3.4	Preliminary Analysis of the Obtained Sub-system . . . . .	84
4.2.4	Phase 2: Graphs and Statistical Analysis . . . . .	87
4.2.5	Phase 3: Invariant Inference and Analysis . . . . .	90
4.2.5.1	Revised Daikon Output . . . . .	91
4.2.5.2	Types of Analysis . . . . .	95
4.2.6	Phase 4: Business Process Analysis . . . . .	100
4.2.6.1	Process Mining of the Physical Process . . . . .	101
4.2.6.2	Network Communications . . . . .	105

<b>5 Case study: the iTrust SWaT System</b>	<b>107</b>
5.1 Architecture . . . . .	107
5.1.1 Physical Process . . . . .	108
5.1.2 Control and Communication Network . . . . .	110
5.2 Datasets . . . . .	112
5.2.1 Our Case Study: the 2015 Dataset . . . . .	113
<b>6 Our Framework at Work on the iTrust SWaT System</b>	<b>117</b>
6.1 Preliminary Operations . . . . .	118
6.2 Planning the Analysis Strategy . . . . .	118
6.3 Reverse Engineering of the iTrust SWaT System . . . . .	120
6.3.1 Reverse Engineering of PLC1 and PLC2 . . . . .	122
6.3.1.1 Pre-processing - Preliminary Analysis . . .	122
6.3.1.2 Graphs and Statistical Analysis . . . . .	126
6.3.1.3 Invariant Inference and Analysis . . . . .	129
6.3.1.4 Business Process Mining and Analysis . .	133
6.3.1.5 Properties . . . . .	135
6.3.2 Reverse Engineering of PLC2 and PLC3 . . . . .	137
6.3.2.1 Pre-processing - Preliminary Analysis . .	137
6.3.2.2 Graphs and Statistical Analysis . . . . .	141
6.3.2.3 Invariant Inference and Analysis . . . . .	145
6.3.2.4 Business Process Mining and Analysis . .	148
6.3.2.5 Properties . . . . .	148
6.3.3 Reverse Engineering of PLC3 and PLC4 . . . . .	148
6.3.3.1 Pre-processing - Preliminary Analysis . .	148
6.3.3.2 Graphs and Statistical Analysis . . . . .	148
6.3.3.3 Invariant Inference and Analysis . . . . .	148
6.3.3.4 Business Process Mining and Analysis . .	148
6.3.3.5 Properties . . . . .	148
<b>7 Conclusion and Future Work</b>	<b>149</b>
<b>List of Figures</b>	<b>153</b>

<b>List of Tables</b>	<b>157</b>
<b>Listings</b>	<b>159</b>
<b>References</b>	<b>161</b>

## Introduction

1 THE advent of Industry 4.0 has sparked significant transformations in  
2 T the realm of Industrial Control Systems (ICS). In recent years, there  
3 has been a rapid expansion in the interconnectivity and convergence of  
4 IT (*Information Technology*) and OT (*Operational Technology*) systems. While  
5 this integration offers undeniable benefits in terms of operational efficiency  
6 and effectiveness of ICSs, it has also exposed these systems to the preva-  
7 lent vulnerabilities commonly associated with IT environments. Conse-  
8 quently, there has been a parallel rise in **cyber-physical attacks**, which  
9 originates in the cyber environment and target the physical processes of  
10 these systems. These attacks aim to manipulate or disrupt the normal op-  
11 eration of ICSs, posing a considerable risk to their integrity and function-  
12 ality.

13 To execute a successful cyber-physical attack, an attacker must possess  
14 a comprehensive understanding of the target system's characteristics and  
15 behavior, commonly referred to as ***process comprehension***. This entails ac-  
16 quiring detailed knowledge about the underlying physical processes, con-  
17 trol mechanisms, communication protocols, and interdependencies within  
18 the system. By attaining such insights, the attacker can effectively identify  
19 vulnerabilities, exploit weaknesses, and manipulate the system in a tar-  
20 geted and stealthy manner.

## **21 1.1 Contribution**

**22** The original purpose of this thesis was to validate a specific methodology  
**23** designed to attain *process comprehension* in an Industrial Control System  
**24** (ICS) when applied to a real-world scenario. This methodology, which  
**25** adopts a **blackbox approach**, involves dynamic analysis of the physical  
**26** process and network communications of the system. The primary goal  
**27** of this methodology is to obtain a comprehensive understanding of the  
**28** industrial process.

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

**36** As the thesis work advanced, it became progressively apparent that  
**37** both the methodology and tools employed required revision and expansion.  
**38** In response, an extensive revamping of the framework was conducted,  
**39** aimed at enhancing and expanding its existing features while introducing  
**40** new ones.

**41** Therefore, the **primary contribution** of this thesis is to **enhance the**  
**42** **original methodology** by refining the existing framework, reassessing the  
**43** approach for each step of the analysis, and incorporating new features.  
**44** The objective is to achieve a more comprehensive and thorough process  
**45** comprehension of the industrial system under investigation. This entails  
**46** expanding the methodology to gather a richer set of information, improv-  
**47** ing the accuracy of the analysis, and incorporating additional techniques  
**48** to uncover hidden patterns and relationships within the system.

**49** Furthermore, the enhanced methodology will be validated through the  
**50** application to a different and larger case study, distinct from the virtu-  
**51** alized testbed used previously. This real-world scenario will provide a

- 52 robust validation of the methodology's effectiveness in a practical set-  
53 ting, ensuring its applicability and reliability in diverse industrial envi-  
54 ronments.

## 55 1.2 Outline

56 The thesis is structured as follows:

- 57 **Chapter 2:** provides a background on Industrial Control Systems, (ICSSs)  
58 describing their structure, components, and some of the network  
59 communication protocols used;
- 60 **Chapter 3:** following an introductory section that provides a brief overview  
61 of the existing literature on process comprehension in industrial con-  
62 trol systems, this chapter focuses on a specific paper that outlines  
63 a methodology to attaining process comprehension of an industrial  
64 system by employing dynamic blackbox analysis;
- 65 **Chapter 4:** outlines a proposal to improve and extend the methodology  
66 outlined in the previous chapter;
- 67 **Chapter 5:** presents the case study on which the proposed methodology  
68 will be applied;
- 69 **Chapter 6:** shows how the proposed methodology is applied to the case  
70 study illustrated above;
- 71 **Chapter 7:** outlines final conclusions and future work.



## Background on Industrial Control Systems

<sup>72</sup> INDUSTRIAL CONTROL SYSTEMS (ICSs) are information systems used to  
<sup>73</sup> control industrial processes such as manufacturing, product handling,  
<sup>74</sup> production, and distribution [1]. ICSs are often found in critical infrastruc-  
<sup>75</sup> ture facilities such as power plants, oil and gas refineries, and chemical  
<sup>76</sup> plant ICSs are different from traditional IT systems in several key ways.

<sup>77</sup> Firstly, ICSs are designed to control physical processes, whereas IT sys-  
<sup>78</sup> tems are designed to process and store data. This means that ICSs have dif-  
<sup>79</sup> ferent requirements for availability, reliability, and performance. Secondly,  
<sup>80</sup> ICSs are typically deployed in environments that are harsh and have lim-  
<sup>81</sup> ited resources, such as extreme temperatures and limited power. Thirdly,  
<sup>82</sup> the protocols hardware and software used in ICSs are often proprietary.

<sup>83</sup> ICSs are becoming increasingly connected to the internet and other net-  
<sup>84</sup> works, which has led to increased concerns about their security. Industrial  
<sup>85</sup> systems were not originally designed with security in mind, and many of  
<sup>86</sup> them have known vulnerabilities that could be exploited by attackers. Ad-  
<sup>87</sup> ditionally, the use of legacy systems and equipment can make it difficult  
<sup>88</sup> to implement security measures. As a result, ICSs are increasingly seen  
<sup>89</sup> as a potential target for cyber attacks, which could have serious conse-  
<sup>90</sup> quences for the safe and reliable operation of critical infrastructure: some  
<sup>91</sup> notorious examples of cyber attacks are (*i*) the **STUXnet** worm [2], which

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

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

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

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

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

## <sup>109</sup> 2.1 Industrial Control Systems Architecture

<sup>110</sup> In the past, there has been a clear division between *Information Technology* (IT) and *Operational Technology* (OT), both at the technical and organizational levels. Each domain has maintained its own distinct technology stacks, protocols, and standards. However, with the emergence of Industry 4.0 and the rapid expansion of industrial automation, which heavily relies on IT tools for monitoring and controlling critical infrastructures, the boundary between IT and OT has started to blur. This trend has paved the way for greater integration between these two domains, thus improving productivity and process quality.

<sup>119</sup>

<sup>120</sup> General ICS architecture consists in **six levels** each representing a functionality: this architecture comprising the OT and IT parts is represented in Figure 2.1 [6][5], according to the *Purdue Enterprise Reference Architecture* (PERA), or simply **Purdue Model**:

<sup>124</sup>

- Level 0 (**Processes, or Field I/O Devices**): contains **field devices**.

<sup>125</sup>

- Level 1 (**Intelligent Devices, or Controller Network**): includes **local or remote controllers** that sense, monitor and control the physical process, such as **PLCs** (2.2.2.1) and **RTUs** (2.2.2.2). Controllers interface directly to the field devices reading data from sensors and sending commands to actuators.

<sup>130</sup>

- Level 2 (**Control Systems, or Area Control**): contains computer systems used to supervising and monitoring the physical process: they provide a **Human-Machine Interface** (HMI, 2.2.3.2) and *Engineering Workstations* (EW) for operator control.

<sup>134</sup>

- Level 3 (**Manufacturing/Site Operations, or Operations/Control**): comprises systems used to manage the production workflow for plant-wide control: they collate informations from the previous levels and store them in Data Historian servers.

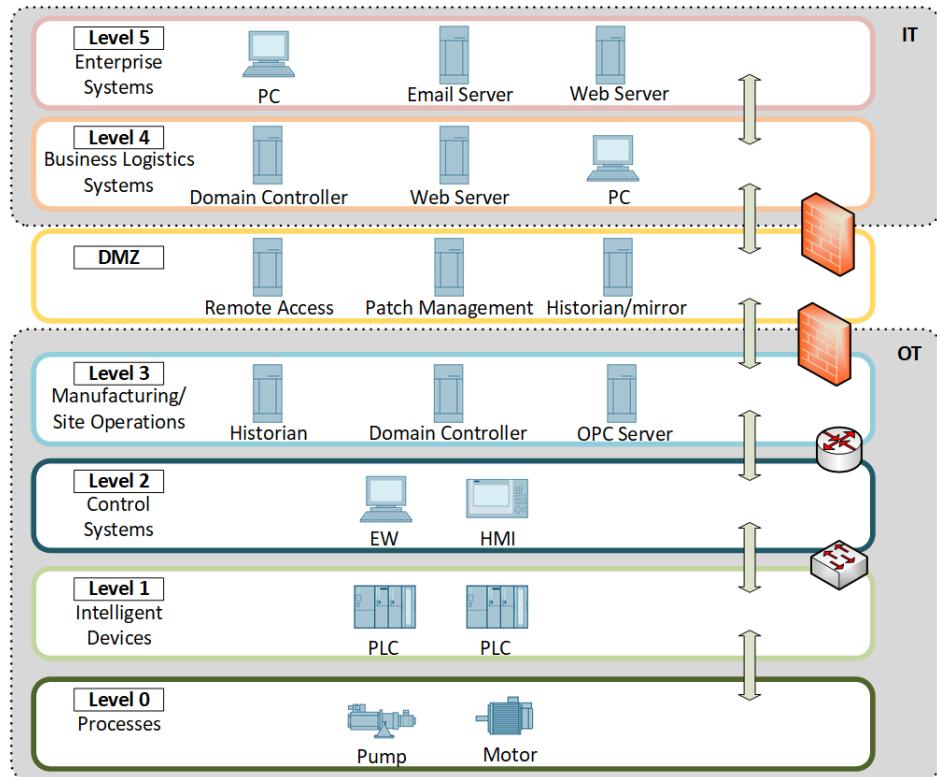


Figure 2.1: ICS architecture schema

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

### <sup>180</sup> 2.2.2 Controller Network Layer

<sup>181</sup>      *Controller Network* layer includes devices that handle data from and to  
<sup>182</sup> the *Field I/O Devices* layer. This kind of device is capable of gathering data  
<sup>183</sup> from sensors, updating its internal state, and activating actuators (for ex-  
<sup>184</sup> ample opening or close a pump that controls the level of a tank), making  
<sup>185</sup> decisions based on a customized program, known as its control logic.  
<sup>186</sup> Commonly found within this layer are *Programmable Logic Controllers* (PLCs)  
<sup>187</sup> and *Remote Terminal Units* (RTUs): in the upcoming sections, we will ex-  
<sup>188</sup> amine these elements in detail.

#### <sup>189</sup> 2.2.2.1 Programmable Logic Controllers

<sup>190</sup>      A *Programmable Logic Controller* (PLC) is a **small and specialized in-**  
<sup>191</sup> **dustrial computer** having the capability of controlling complex industrial  
<sup>192</sup> and manufacturing processes [7].

<sup>193</sup>      Compared to relay systems and personal computers, PLCs are opti-  
<sup>194</sup> mized for control tasks and industrial environments: they are rugged and  
<sup>195</sup> designed to withdraw harsh conditions such as dust, vibrations, humid-  
<sup>196</sup> ity and temperature: they have more reliability than personal computers,  
<sup>197</sup> which are more prone to crash, and they are more compact a require less  
<sup>198</sup> maintenance than a relay system.

<sup>199</sup> Furthermore, I/O interfaces are already on the controller, so PLCs are eas-  
<sup>200</sup> ier to expand with additional I/O modules (if in a rack format) to manage  
<sup>201</sup> more inputs and ouputs, without reconfiguring hardware as in relay sys-  
<sup>202</sup> tems when a reconfiguration occurs.

<sup>203</sup>      PLCs are more *user-friendly*: they are not intended (only) for computer  
<sup>204</sup> programmers, but designed for engineers with a limited knowledge in  
<sup>205</sup> programming languages: control program can be entered with a simple  
<sup>206</sup> and intuitive language based on logic and switching operations instead of  
<sup>207</sup> a general-purpose programming language (*i.e.* C, C++, ...).

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

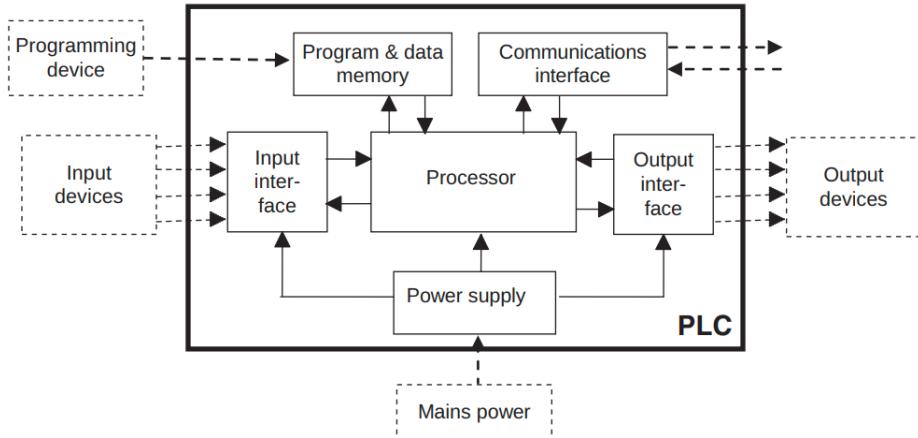
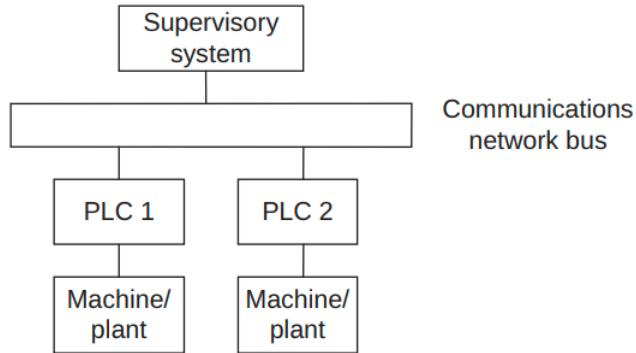


Figure 2.2: PLC architecture

- 210 • **Processor unit (CPU):** contains the microprocessor. This unit inter-  
211 pretes the input signals from I/O modules, executes the control pro-  
212 gram stored in the Memory Unit and sends the output signals to the  
213 I/O Modules. The processor unit also sends data to the Communi-  
214 cation interface, for the communication with additional devices.
- 215 • **Power supply unit:** converts AC voltage to low DC voltage.
- 216 • **Programming device:** is used to store the required program into the  
217 memory unit.
- 218 • **Memory Unit:** consists in RAM memory and ROM memory. RAM  
219 memory is used for storing data from inputs, ROM memory for stor-  
220 ing operating system, firmware and user program to be executed by  
221 the CPU.
- 222 • **I/O modules:** provide interface between sensors and final control  
223 elements (actuators).
- 224 • **Communications interface:** used to send and receive data on a net-  
225 work from/to other PLCs.



*Figure 2.3: PLC communication schema*

<sup>226</sup> **PLC Programming** Two different programs are executed in a PLC: the  
<sup>227</sup> **operating system** and the **user program**.

<sup>228</sup> The operating system tasks include executing the user program, man-  
<sup>229</sup> aging memory areas and the *process image table* (memory registers where  
<sup>230</sup> inputs from sensors and outputs for actuators are stored).

<sup>231</sup> The user program needs to be uploaded on the PLC via the program-  
<sup>232</sup> ming device and runs on the process image table in *scan cycles*: each scan  
<sup>233</sup> is made up of three phases [9]:

- <sup>234</sup> 1. reading inputs from the process images table
- <sup>235</sup> 2. execution of the control code and computing the physical process evolution
- <sup>237</sup> 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

<sup>240</sup> Standard PLCs **programming languages** are basically of two types:  
<sup>241</sup> **textuals** and **graphicals**. Textual languages include languages such as  
<sup>242</sup> *Instruction List* (IL) and *Structured Text* (ST), while *Ladder Diagrams* (LD),  
<sup>243</sup> *Function Block Diagram* (FBD) and *Sequential Function Chart* (SFC) belong  
<sup>244</sup> to the graphical languages.

245 Graphical languages are more simple and immediate comparing to the  
 246 textual ones and are preferred by programmers because of their features  
 247 and simplicity, in particular the **Ladder Logic programming** (see Figure  
 248 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

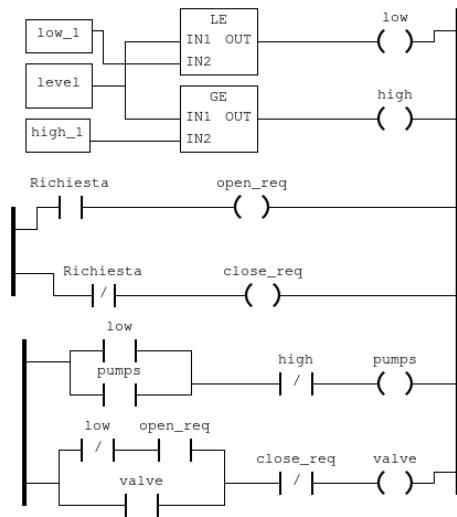
LE3_OUT := LE(level, low_1);
low := LE3_OUT;
GE7_OUT := GE(level, high_1);
high := GE7_OUT;
open_req := Richiesta;
close_req := NOT(Richiesta);
pumps := NOT(high) AND (low OR pumps);
valve := NOT(close_req) AND (open_req AND NOT(low) OR valve);

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

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

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

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

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

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

270     **2.2.2.2 Remote Terminal Units**

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

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

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

### 289 2.2.3 Area Control Layer

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

296 2.2.3.1 Supervisory Control And Data Acquisition

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

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

304 The SCADA software processes, distributes, and displays the data, help-  
305 ing operators and other employees analyze the data and make important  
306 decisions.

307 2.2.3.2 Human-Machine Interface

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

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

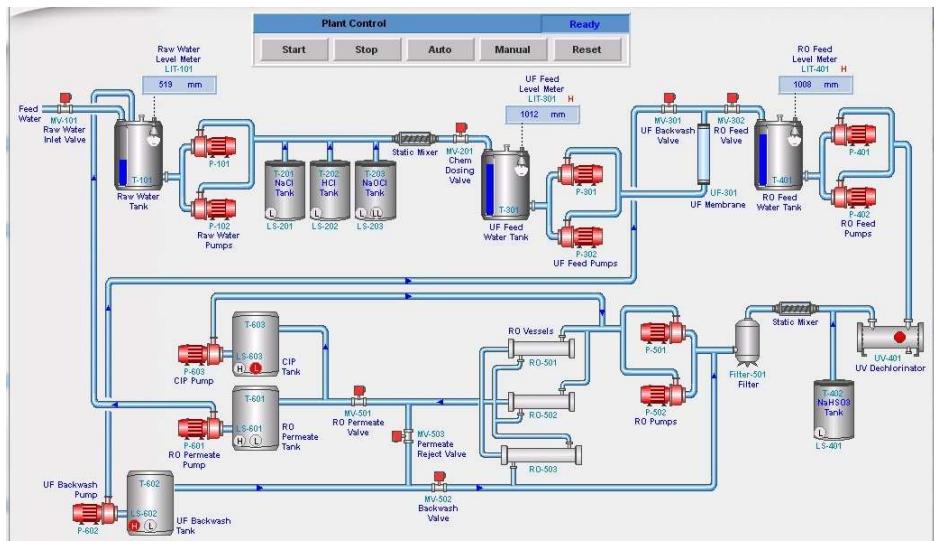


Figure 2.5: Example of HMI for a water treatment plant

#### 316 2.2.4 Operations/Control Layer

317 Within this zone, there are specialized OT devices that are utilized to  
 318 manage production workflows on the shop floor [14]. These devices in-  
 319 clude:

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

#### 326 2.2.5 Demilitarized Zone

327 This zone comprises security systems like firewalls, proxies, *Intrusion*  
 328 *Detection and Prevention systems* (IDP) and *Security Information and Event*  
 329 *Management* (SIEM) systems which are implemented to mitigate the risk  
 330 of lateral threat movement between IT and OT domains. With the rise

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

### 336 2.2.6 Industrial Protocols

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

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

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

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

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

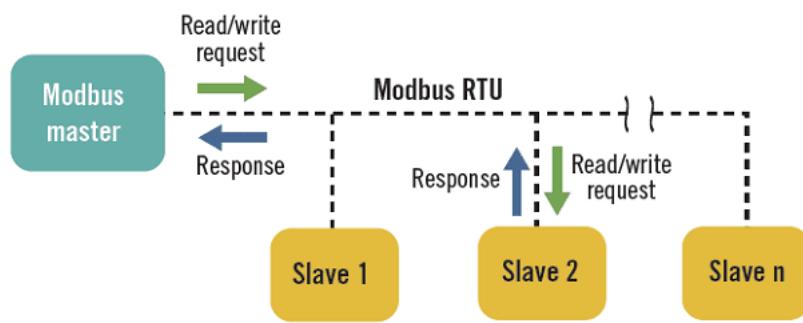
<sup>358</sup> **2.2.6.1 Modbus**

<sup>359</sup> *Modbus* is a serial communication protocol developed by Modicon (now  
<sup>360</sup> Schneider Electric) in 1979 for use with its PLCs [15] and designed ex-  
<sup>361</sup> pressly for industrial use: it facilitates interoperability of different devices  
<sup>362</sup> connected to the same network (sensors, PLCs, HMIs, ...) and it is also  
<sup>363</sup> often used to connect RTUs to SCADA acquisition systems.

<sup>364</sup> Modbus is the most widely used communication protocol among in-  
<sup>365</sup> dustrial systems because it has several advantages:

- <sup>366</sup> • simplicity of implementation and debugging
- <sup>367</sup> • it moves raw bits and words, letting the individual vendor to repre-  
<sup>368</sup> sent the data as it prefers
- <sup>369</sup> • it is, nowadays, an **open** and *royalty-free* protocol: there is no need  
<sup>370</sup> to sustain licensing costs for implementation and use by industrial  
<sup>371</sup> device vendors

<sup>372</sup> Modbus is a **request/response** (or *master/slave*) protocol: this makes it  
<sup>373</sup> independent of the transport layer used.



*Figure 2.6: Modbus Request/Response schema*

<sup>374</sup> In this kind of architecture, a single device (master) can send requests  
<sup>375</sup> to other devices (slaves), either individually or in broadcast: these slave  
<sup>376</sup> devices (usually peripherals such as actuators) will respond to the master

377 by providing data or performing the action requested by the master using  
378 the Modbus protocol. Slave devices cannot generate requests to the master  
379 [16].

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

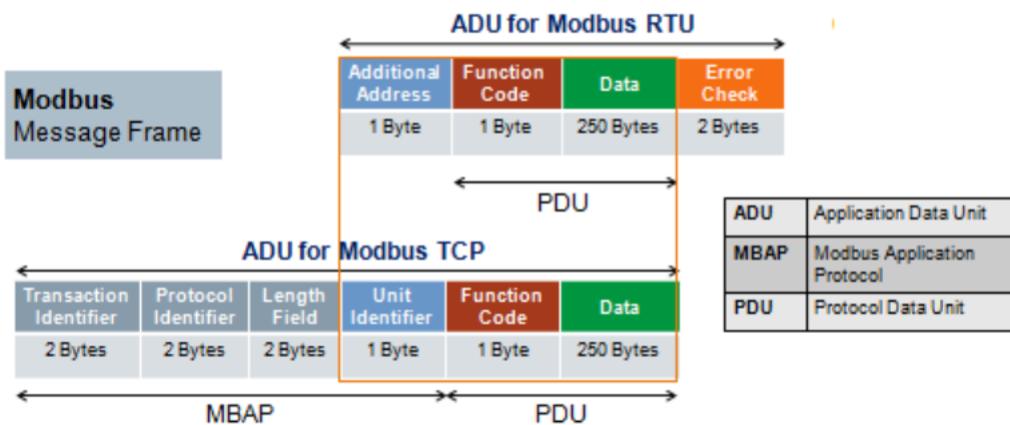


Figure 2.7: Modbus RTU frame and Modbus TCP frame

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

- 387 • *Coil*: binary type, read/write accessible by both masters and slaves
- 388 • *Discrete Input*: binary type, accessible in read-only mode by masters  
389 and in read/write mode by slaves
- 390 • *Analog Input*: 16 bits in size (word), are accessible in read-only mode  
391 by masters and in read/write mode by slaves
- 392 • *Holding Register*: 16 bits in size (word), accessible in read/write mode  
393 by both masters and slaves. Holding Registers are the most com-  
394 monly used registers for output and as general memory registers.

<sup>395</sup> **Modbus Function Codes** *Modbus Function Codes* are specific codes used  
<sup>396</sup> by the Modbus master within a request frame (see Figure 2.7) to tell the  
<sup>397</sup> Modbus slave device which register type to access and which action to  
<sup>398</sup> perform on it.

<sup>399</sup> Two types of Function Codes exists: for data access and for diagnostic  
<sup>400</sup> 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

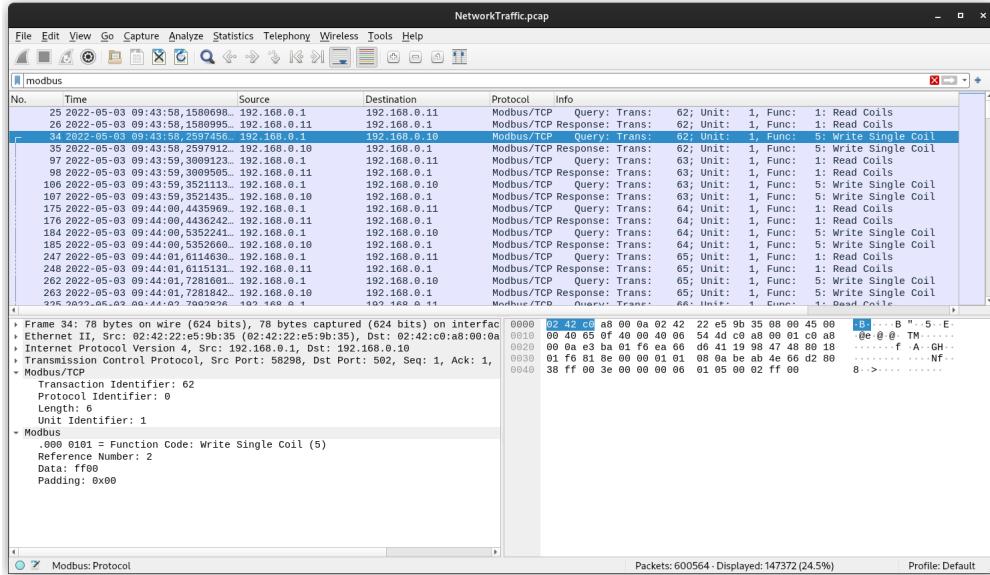
*Table 2.2: Modbus Function Codes list*

<sup>401</sup> **Modbus Security Issues** Despite its simplicity and widespread use, the  
<sup>402</sup> Modbus protocol does not have any security feature, which exposes it to  
<sup>403</sup> vulnerabilities and attacks.

<sup>404</sup> Data in Modbus are transmitted unencrypted (*lack of confidentiality*), with  
<sup>405</sup> no data integrity controls (*lack of integrity*) and authentication checks (*lack*  
<sup>406</sup> *of authentication*), in addition to the *lack of session*. Hence, the protocol is  
<sup>407</sup> vulnerable to a variety of attacks, such as Denial of Services (DoS), buffer  
<sup>408</sup> overflows and reconnaissance activities.

<sup>409</sup> The easiest attack to bring to the Modbus protocol, however, is **packet**  
<sup>410</sup> **sniffing** (Figure 2.8): since, as mentioned earlier, network traffic is un-  
<sup>411</sup> encrypted and the data transmitted is in cleartext, it is sufficient to use  
<sup>412</sup> a packet sniffer to capture the network traffic, read the packets and thus

- 413 gather informations about the system such as ip addresses, function codes  
 414 of requests and to modify the operation of the devices.



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

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

### 421 2.2.6.2 EtherNet/IP

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

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

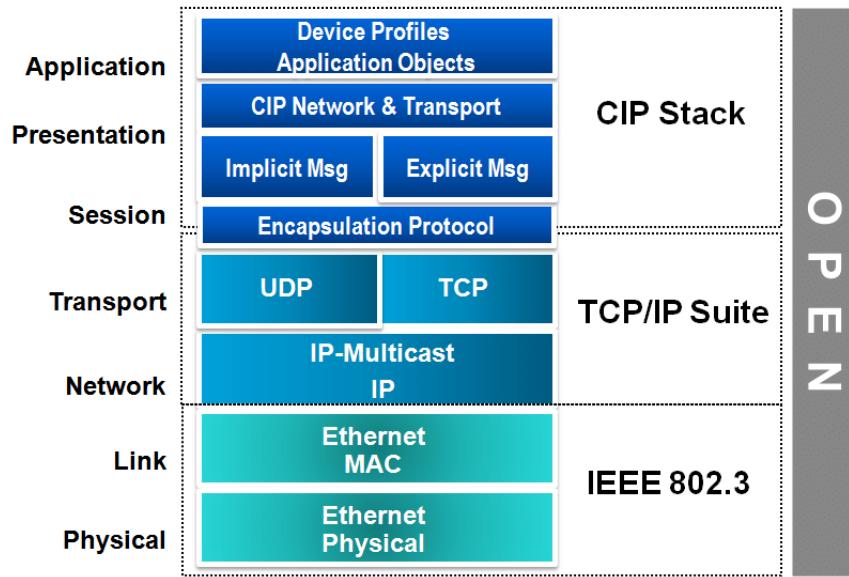


Figure 2.9: OSI model for EtherNet/IP stack

<sup>429</sup> and other industrial protocols for data access and exchange such as *Open*  
<sup>430</sup> *Platform Communication* (OPC).

<sup>431</sup> **Physical and Data Link layer** The use of the IEEE 802.3 standard allows  
<sup>432</sup> EtherNet/IP to flexibly adopt different network topologies (star, linear,  
<sup>433</sup> ring, etc.) over different connections (copper, fibre optic, wireless, etc.), as  
<sup>434</sup> well as the possibility to choose the speed of network devices.  
<sup>435</sup> IEEE 802.3 in addition defines at Data Link layer the *Carrier Sense Multiple*  
<sup>436</sup> *Access - Collision Detection* (CSMA/CD) protocol, which controls access to  
<sup>437</sup> the communication channel and prevents collisions.

<sup>438</sup> **Transport layer** At the transport level, EtherNet/IP encapsulates mes-  
<sup>439</sup> sages from the CIP stack into an Ethernet message, so that messages can  
<sup>440</sup> be transmitted from one node to another on the network using the TCP/IP  
<sup>441</sup> protocol. EtherNet/IP uses two forms of messaging, as defined by CIP  
<sup>442</sup> standard [18][20]:

- <sup>443</sup> • **unconnected messaging:** used during the connection establishment  
<sup>444</sup> phase and for infrequent, low priority, explicit messages. Uncon-

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

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

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

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

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

#### 465        2.2.6.3 Common Industrial Protocol (CIP)

466        The *Common Industrial Protocol* (CIP) is an open industrial automation  
467        protocol supported by ODVA. It is a **media independent** (or *transport in-*  
468        *dependent*) protocol using a *producer-consumer* communication model and  
469        providing a **unified architecture** throughout the manufacturing enterprise  
470        [21][22].

471        CIP has been adapted in different types of network:

- 472     • **EtherNet/IP**, adaptation to *Transmission Control Protocol* (TCP) tech-  
473         nologies
- 474     • **ControlNet**, adaptation to *Concurrent Time Domain Multiple Access*  
475         (CTDMA) technologies
- 476     • **DeviceNet**, adaptation to *Controller Area Network* (CAN) technolo-  
477         gies
- 478     • **CompoNet**, adaptation to *Time Division Multiple Access* (TDMA) tech-  
479         nologies

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

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

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

## State of the Art

496 IN COVENTIONAL IT SYSTEMS, the objective of an attacker is to comprehend  
497 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).

503 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.

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

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

### 515 3.1 Literature on Process Comprehension

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

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

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

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

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

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

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

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

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

572 Reconnaissance phase is fundamental to process comprehension and  
573 thus to the execution of MitM attacks.

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

## 28 3.2 Ceccato et al.'s black-box dynamic analysis for water-tank systems

---

577           posed PLCs and the Modbus protocol network scans an approximate  
578           model of the physical process. This model is obtained by inferring  
579           statistical properties, business process and system invariants from  
580           data logs.

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

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

## 587 3.2 Ceccato et al.'s black-box dynamic analysis 588           for water-tank systems

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

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

601           To accomplish this, the various types of memory registers are analyzed,  
602           and attempts are made to determine the nature of the data they might  
603           contain.

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

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

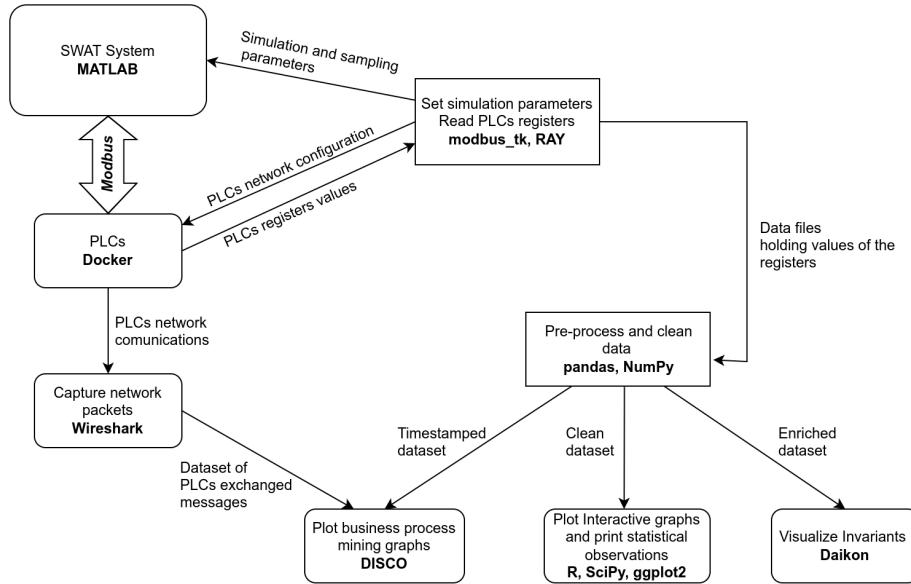
- 609 1. **scanning of the system and data pre-processing:** this phase involves  
610 gathering data to generate data logs for the registers of PLCs and for  
611 Modbus network communications.
- 612 2. **graphs and statistical analysis:** The collected data is utilized to pro-  
613 vide insights into the memory registers associated with the Modbus  
614 protocol by leveraging graphs and statistical data. This analysis ap-  
615 proach offers valuable information about the characteristics and pat-  
616 terns of the memory registers.
- 617 3. **invariants inference and analysis:** generates system invariants, which  
618 are used to identify specific patterns and regularities within the sys-  
619 tem. Additionally, this phase provides users with the capability to  
620 view invariants related to a particular sensor or actuator.
- 621 4. **business process mining and analysis:** Using event logs, this phase  
622 involves reconstructing the business process that depicts how a pro-  
623 cess is executed. This step enables a thorough understanding of the  
624 sequence of events that occur in the system and how they are in-  
625 terrelated, ultimately leading to a comprehensive overview of the  
626 business process.

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

### 632 3.2.1 Testbed

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

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



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

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

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

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

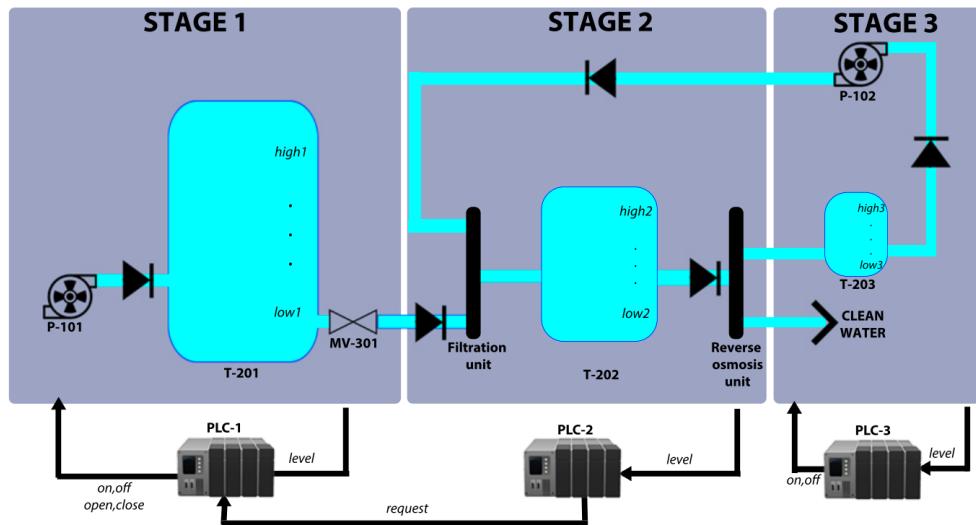


Figure 3.2: The simplified SWaT system used for running Ceccato et al. methodology

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

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

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

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

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

---

- 668           2. if the level of T-201 reaches the *high setpoint high1* (which is also  
669           hardcoded in a specific memory register), then the pump **P-101**  
670           **is closed**;  
671           3. in cases where the level of T-201 is between the *low setpoint low1*  
672           and the *high setpoint high1*, PLC1 waits for a request from PLC2  
673           to open or close the valve MV-301. If a request to open the valve  
674           MV-301 is received, water will flow from T-201 to T-202. How-  
675           ever, if no request is received, the valve remains closed. In both  
676           situations, the pump P-101 remains closed.

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

- 679           1. when the water level in tank T-202 reaches *low setpoint low2* (also  
680           hardcoded in the memory registers), PLC2 sends a request to  
681           PLC1 through a Modbus channel to **open valve MV-301**. This  
682           request is made in order to allow the water to flow from tank T-  
683           201 to tank T-202. The transmission channel between the PLCs  
684           is established by copying a boolean value from a memory reg-  
685           ister of PLC2 to a corresponding register of PLC1.  
686           2. when the water level in tank T-202 reaches the *high setpoint high2*  
687           value (also hardcoded in the memory registers), PLC2 sends a  
688           **close request to PLC1 for valve MV-301**. This request prompts  
689           PLC1 to close the valve, stopping the flow of water from tank  
690           T-201 to tank T-202.  
691           3. In cases where the water level in tank T-202 is between the low  
692           and high setpoints, the valve MV-301 remains in its current state  
693           (open or closed) while the tank is either filling or emptying.

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

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

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

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

### 703 3.2.2 Scanning of the System and Data Pre-processing

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

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

```
723     "127.0.0.1/8502/2022-05-03 12_10_00.591": {  
724         "DiscreteInputRegisters": {"%IX0.0": "0"},  
725         "InputRegisters": {"%IW0": "53"},  
726         "HoldingOutputRegisters": {"%QW0": "0"},  
727         "MemoryRegisters": {"%MW0": "40", "%MW1": "80"},
```

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

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

*Listing 3.1: Example of registers capture*

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

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

```
737     Time,Source,Destination,Protocol,Length,Function Code,  
738     ↳ Destination Port,Source Port,Data,Frame length on the  
739     ↳ wire,Bit Value,Request Frame,Reference Number,Info  
740     2022-05-03 11:43:58.158,IP_PLC1,IP_PLC2,Modbus/TCP,76,Read  
741     ↳ Coils,46106,502,,76,TRUE,25,,,"Response: Trans: 62;  
742     ↳ Unit: 1, Func: 1: Read Coils"
```

*Listing 3.2: Example of raw network capture*

743 **Data Pre-processing** The data collected by scanning the memory regis-  
744 ters of the PLCs are then reprocessed by a Python script and converted  
745 in order to create a distinct raw dataset in *Comma Separated Value* for-  
746 mat (CSV) for each PLC, containing the memory register values associ-  
747 ated with the corresponding controller registers. These datasets are repro-  
748 cessed again through the Python modules for **pandas** [42] and **NumPy** [43]  
749 by another script to first perform a **data cleanup**, removing all unused reg-  
750 isters, **merged** into a single dataset, and finally **enriched** with additional  
751 data, such as the **previous value** of all registers and the the **measurement**  
752 **slope**, that is, the trend of the water level in the system tanks along the  
753 system cycles<sup>1</sup>. See 3.2.7 for more detail.

754

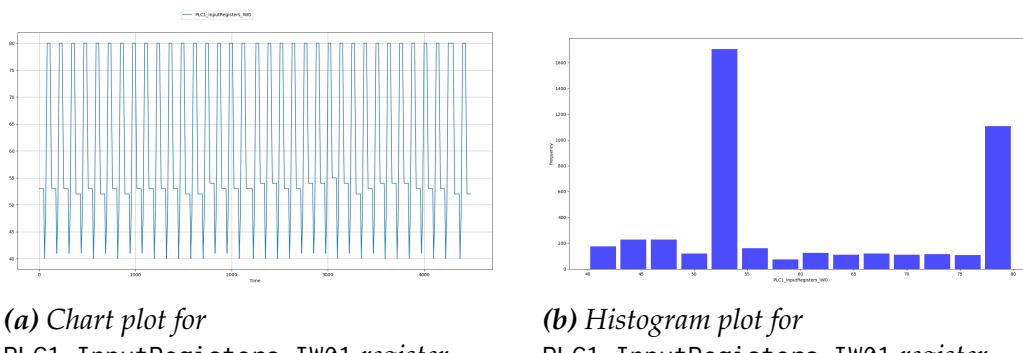
<sup>1</sup>Not all additional data are calculated and entered automatically by the tool: some are manually inserted.

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

### 3.2.3 Graphs and Statistical Analysis

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

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



*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-**

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

---

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

776 These informations include:

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

```
781     Chi-squared test for uniformity
782     Distance      pvalue      Uniform?
783     12488.340    0.00000000    NO
784
785     Shapiro-Wilk test for normality
786     Test statistic   pvalue      Normal?
787     0.844        0.00000000    NO
788
789     Stats of PLC1_InputRegisters_IW0
790     Sample mean = 60.8881; Stddev = 13.0164; max = 80; min =
791     ↪ 40 for 4488 values
```

*Listing 3.3: Statistical data for PLC1\_InputRegisters\_IW0 register*

### 792 3.2.4 Invariant Inference and Analysis

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

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

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

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

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

811 Therefore, Daikon is fed with the enriched dataset obtained in the pre-  
 812 processing phase<sup>2</sup>: a simple bash script launches Daikon (optionally spec-  
 813 ifying the desired condition for analysis in the *.spininfo* file), which output is  
 814 simply redirected to a text file containing the general invariants of the sys-  
 815 tem (i.e., valid regardless of any custom condition specified), those gener-  
 816 ated based on the custom condition in the *.spininfo* file, and those generated  
 817 based on the negation of the condition (see Listing 3.5 below).

```
818 =====
819 aprogram.point:::POINT
820 PLC2_MemoryRegisters_MW1 == PLC3_MemoryRegisters_MW1
821 PLC1_MemoryRegisters_MW0 == 40.0
822 PLC1_MemoryRegisters_MW1 == 80.0
823 PLC1_Coils_QX00 one of { 0.0, 1.0 }
824 [...]
825 =====
826 aprogram.point:::POINT; condition="PLC1_InputRegisters_IW0
827   ↪ > 60"
828 PLC1_InputRegisters_IW0 > PLC1_MemoryRegisters_MW0
829 PLC1_InputRegisters_IW0 > PLC1_Min_safety
830 PLC1_MemoryRegisters_MW0 < prev_PLC1_InputRegisters_IW0
831 [...]
832 =====
833 aprogram.point:::POINT; condition="not(
834   ↪ PLC1_InputRegisters_IW0 > 60)"
835 PLC1_InputRegisters_IW0 < PLC1_MemoryRegisters_MW1
836 PLC1_InputRegisters_IW0 < PLC1_Max_safety
```

---

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

### 38 3.2 Ceccato et al.'s black-box dynamic analysis for water-tank systems

---

```
837     PLC1_MemoryRegisters_MW1 > prev_PLC1_InputRegisters_IW0  
838     [...]
```

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

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

841  
842 Some examples of invariants derived from the enriched dataset may be:

- 843     • measurements bounded by some setpoint;  
844     • actuators state changes occurred in the proximity of setpoints or,  
845       vice versa, proximity of setpoints upon the occurrence of an actuator  
846       state change;  
847     • state invariants of some actuators correspond to a specific trend in  
848       the evolution of the measurements (ascending, descending, or sta-  
849       ble) or, vice versa, the measurements trend corresponds to a specific  
850       state invariant of some actuators.

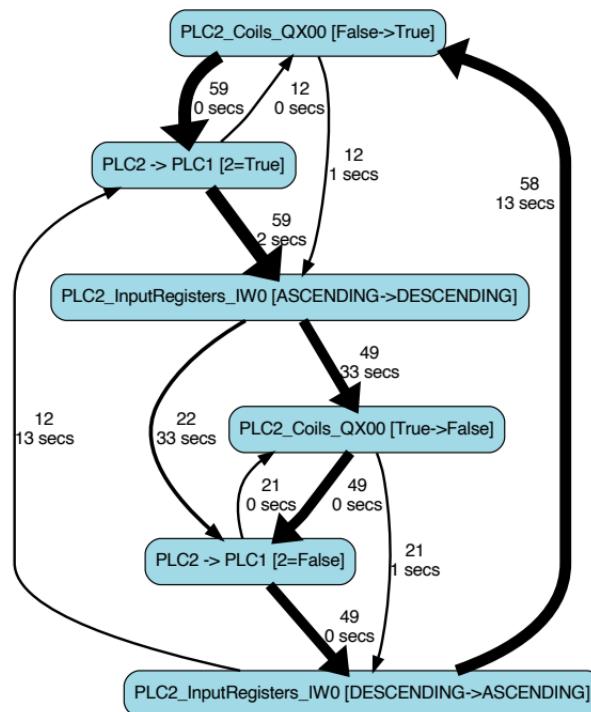
#### 851 3.2.5 Business Process Mining and Analysis

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

856  
857 In the considered system, process mining begins by analyzing the event  
858 logs derived from scanning the memory registers of the PLCs and moni-  
859 toring the network communications associated with the Modbus protocol,  
860 as detailed in Subsection 3.2.2. These event logs serve as the *execution trace*  
861 of the system. A Java program is utilized to extract and consolidate infor-  
862 mation from these event logs, resulting in a CSV format file that captures  
863 the relevant data.

864 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.

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

---

### 879 3.2.6 Application

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

888 **Data Collection and Pre-processing** According to what is described in  
889 the paper, the data gathering process lasted six hours, with a granular-  
890 ity of one data point per second (a full system cycle takes approximately  
891 30 minutes). Each datapoint consists of 168 attributes (55 registers plus  
892 a special register concerning the tank slope of each PLC) after the en-  
893 richment. In addition, IP addresses are automatically replaced by an ab-  
894 stract name identified by the prefix PLC followed by a progressive integer  
895 (PLC1, PLC2, PLC3), in order to make reading easier.

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

898 **Property 1:** PLC1\_MemoryRegisters\_MW0, PLC1\_MemoryRegisters\_MW1,  
899 PLC2\_MemoryRegisters\_MW0, PLC2\_MemoryRegisters\_MW1,  
900 PLC3\_MemoryRegisters\_MW0 and PLC3\_MemoryRegisters\_MW1  
901 registers contain constant integer values (40, 80, 10, 20, 0, 10 respec-  
902 tively)<sup>3</sup>. The authors speculate that they may be (relative) hardcoded  
903 **setpoints**.

---

<sup>3</sup>From my tests on the original tool and dataset, the PLC3\_MemoryRegisters\_MW0 register is deleted during the *pre-processing* phase, as it is recognized as an unused register because of the constant value "0" it takes on. This leads me to assume that the properties are derived from a human read of the dataset prior to the *pre-processing* phase.

904     **Property 2:** PLC1\_Coils\_QX01, PLC1\_Coils\_QX02, PLC2\_Coils\_QX01,  
 905       PLC2\_Coils\_QX02, PLC3\_Coils\_QX01 and PLC3\_Coils\_QX03 contain mu-  
 906       table binary (Boolean) values. The authors speculate that these reg-  
 907       isters can be associated with the **actuators** of the system.

908     **Property 3:** PLC1\_InputRegisters\_IW0, PLC2\_InputRegisters\_IW0 and  
 909       PLC3\_InputRegisters\_IW0 registers contain mutable values.

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

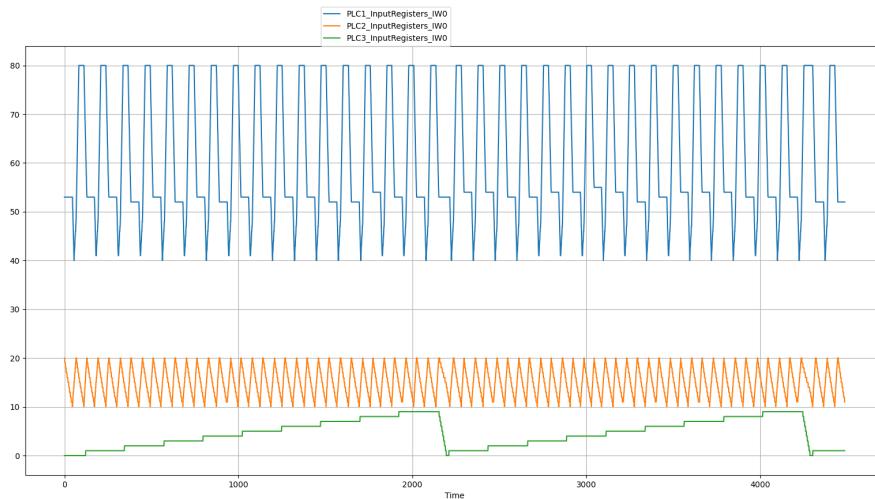


Figure 3.5: Execution traces of *InputRegisters\_IW0* on the three PLCs

913       The graph analysis of the *InputRegisters\_IW0* registers of the three  
 914       PLCs (summarized in Figure 3.5 with a single plot) not only seems to con-  
 915       firm the conjecture, but also allows the measurements to be correlated with  
 916       the contents of the *MemoryRegisters\_MW0* and *MemoryRegisters\_MW1* regis-  
 917       ters to the measurements, which may well represent the **relative setpoints**  
 918       **of the measurements**. Hence, we have *Conjecture 2* described in the paper  
 919       referring to the relative setpoints:

920

921     **Conjecture 2:**

922       - the relative setpoints for *PLC1\_InputRegisters\_IW0* are 40 and 80;

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

---

- 923 - the relative setpoints for PLC2\_InputRegisters\_IW0 are 10 and 20;  
924 - the relative setpoints for PLC3\_InputRegisters\_IW0 0 and 9.

925 Further confirmation of this conjecture may come from statistical anal-  
926 ysis. Indeed, in the example in Listing 3.1, some statistical data are given  
927 for the register PLC1\_InputRegisters\_IW0, including the maximum value  
928 and the minimum value: these values are, in fact, 80 and 40 respectively.

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

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

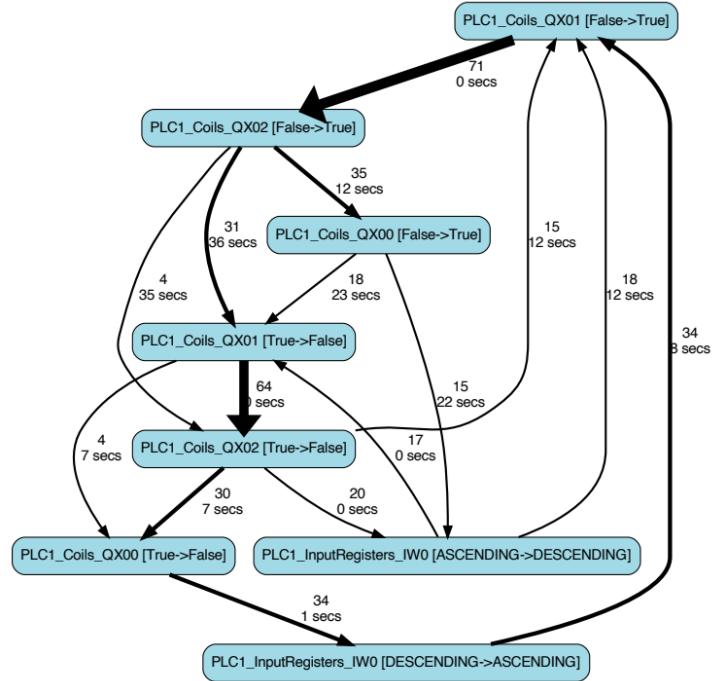
934 **Property 4:** PLC2 sends messages to PLC1 (see Figure 3.6b) which are  
935 recorded to PLC1\_Coils\_QX02.

936 **Conjecture 3:** PLC2\_Coils\_QX00 determines the trend in tank T-202 (Figure  
937 3.6b).

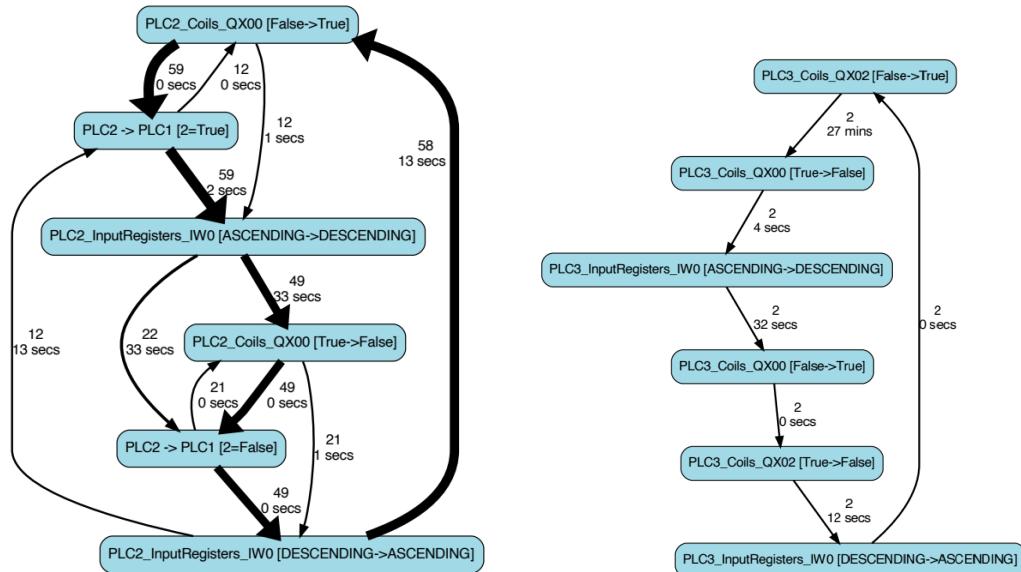
938 When this register is set to *True*, the input register PLC2\_InputRegisters\_IW0  
939 related to the tank controlled by PLC2 starts an **ascending trend**; vice  
940 versa, when the coil register is set to *False*, the input register starts a  
941 **descending trend**.

942 **Conjecture 4:** If PLC1\_Coils\_QX00 change his value to True, trend in tank  
943 T-201, related to PLC1\_InputRegisters\_IW0 and controlled by PLC1,  
944 become **ascending** (see Figure 3.6a)

945 **Conjecture 5:** PLC3\_Coils\_QX00 starts a **decreasing trend** in tank T-203, re-  
946 lated to PLC3\_InputRegisters\_IW0 and controlled by PLC3, whereas  
947 PLC3\_Coils\_QX02 starts an **increasing trend** on the tank (see Figure  
948 3.6c)



**(a) States in PLC1**



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

(c) States in PLC3

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

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

---

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

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

962 However, it was possible to have confirmation of the conjectures made  
963 in the previous stages of the analysis: starting with the setpoints, analyz-  
964 ing the output of the invariants returned by Daikon<sup>4</sup> reveals that

965  
966 PLC1\_InputRegisters\_IW0 >= PLC1\_MemoryRegisters\_MW0 == 40.0  
967 PLC1\_InputRegisters\_IW0 <= PLC1\_MemoryRegisters\_MW1 == 80.0  
968 PLC2\_InputRegisters\_IW0 >= PLC2\_MemoryRegisters\_MW0 == 10.0  
969 PLC2\_InputRegisters\_IW0 <= PLC2\_MemoryRegisters\_MW1 == 20.0  
970 PLC3\_InputRegisters\_IW0 >= PLC3\_MemoryRegisters\_MW0 == 0.0  
971 PLC3\_InputRegisters\_IW0 <= PLC3\_MemoryRegisters\_MW1 == 9.0  
972  
973 i.e., that the MemoryRegisters\_MW0 and MemoryRegisters\_MW1 registers of  
974 each PLC contain the **absolute minimum and maximum setpoints**, re-  
975 spectively (*Property 5*).

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

---

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

978

979 PLC1\_Coils\_QX01 == PLC1\_Coils\_QX02 == PLC2\_Coils\_QX00

980

981 and from this derive that there is a **communication channel between PLC2**  
982 **and PLC1**, where the value of PLC2\_Coils\_QX00 is copied to PLC1\_Coils\_QX01  
983 and PLC1\_Coils\_QX02 (*Property 6*).

984 Regarding the **relationships between actuator state changes and mea-**  
985 **surement trends**, invariant analysis yields the results summarized in the  
986 following rules:

987 **Property 7:** Tank T-202 level *increases* iff PLC1\_Coils\_QX01 == True. Oth-  
988 erwise, if PLC1\_Coils\_QX01 == False will be *non-increasing*.

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

990 PLC2\_InputRegisters\_IW0 == PLC2\_MemoryRegisters\_MW0 == 20.0 &amp;&amp; PLC2\_slope &gt; 0

991 is verified. On the opposite hand, if the coil is *False*, the condition992 PLC2\_InputRegisters\_IW0 == PLC2\_MemoryRegisters\_MW0 == 20.0 && PLC2\_slope <= 0 is verified. The  
993 *slope* is increasing if > 0, decreasing if < 0, stable otherwise.

994 **Property 8:** Tank T-201 level *increases* iff PLC1\_Coils\_QX00 == True. On the  
995 other hand, if PLC1\_Coils\_QX00 == False and if PLC1\_Coils\_QX01 ==  
996 *True* the level will be *non-decreasing*.

997 **Property 9:** Tank T-203 level *decreases* iff PLC3\_Coils\_QX00 == True. It will  
998 be *non-decreasing* if PLC1\_Coils\_QX00 == False.

999 The last two properties concern the **relationship between actuator state**  
1000 **changes and the setpoints**: it is intended to check what happens to the  
1001 actuators when the water level reaches one of these setpoints. From the  
1002 analysis of the relevant invariants, the following properties are derived:

1003 **Property 10:** Tank T-201 reaches the upper absolute setpoint when  
1004 PLC1\_Coils\_QX00 changes its state from *True* to *False*. If the coil changes  
1005 from *False* to *True*, the tank reaches its absolute lower setpoint.

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

---

1006 **Property 11:** Tank T-203 reaches the upper absolute setpoint when  
1007 PLC3\_Coils\_QX00 changes its state from *True* to *False*. If the coil changes  
1008 from *False* to *True*, the tank reaches its absolute lower setpoint.

### 1009 3.2.7 Limitations

1010 The methodology proposed by Ceccato et al. is certainly valid and  
1011 offers a good starting point for approaching the reverse engineering of  
1012 an industrial control system from the attacker's perspective, while also  
1013 providing a tool to perform this task.

1014 The limitations of this approach, however, all lie in the tool mentioned  
1015 above and also in the testbed described in Section 3.2.1. In this section  
1016 we will explain which are the criticisms of each phase, while in Chapter 4  
1017 we will formulate proposals to improve and make this methodology more  
1018 efficient.

1019 **General Criticism** There are several critical aspects associated with the  
1020 application of this approach: the primary one concerns the fact that the  
1021 proposed tool seems to be built specifically for the testbed used and that  
1022 it is not applicable to other contexts, even to the same type of industrial  
1023 control system (water treatment systems, in this case).

1024 What severely limits the analysis performed with the tool implemented  
1025 by Ceccato et al. is the use of *ad hoc* solutions and *a posteriori* interventions  
1026 done manually on the datasets after the data gathering process: we will  
1027 discuss this last aspect in more detail later.

1028 Moreover, there is the presence of many *hardcoded* variables and condi-  
1029 tions within the scripts: this makes the system unconfigurable and unable  
1030 to properly perform the various stages of the analysis as errors can occur  
1031 due to incorrect data and mismatches with the system under analysis.

1032 Having considered, furthermore, only the Modbus protocol for network  
1033 communications between the PLCs is another major limiting factor and

1034 does not help the methodology to be adaptable to different systems com-  
1035 municating with different protocols (sometimes even multiple ones on the  
1036 same system).

1037 Let us now look at the limitations and critical aspects of each phase.

1038 **Testbed** The testbed environment used by Ceccato et. al is entirely simu-  
1039 lated, from the physical system to the control system. The PLCs were built  
1040 with **OpenPLC** [50] in a Docker environment [51], while the physics part  
1041 was built through **Simulink** [52].

1042 OpenPLC is an open source cross-platform software that simulates the  
1043 hardware and software functionality of a physical PLC and also offers a  
1044 complete editor for PLC program development with support for all stan-  
1045 dard languages: *Ladder Logic* (LD), *Function Block Diagram* (FBD), *Instruc-*  
1046 *tion List* (IL), *Structured Text* (ST), and *Sequential Function Chart* (SFC).

1047 It is for sure an excellent choice for creating a zero-cost industrial or home  
1048 automation and *Internet of Things* (IoT) system that is easy to manage via a  
1049 dedicated, comprehensive and functional web interface. In spite of these  
1050 undoubted merits, however, there are (at the moment) **very few supported**  
1051 **protocols**: the main one and also referred to in the official documentation  
1052 is **Modbus**, while the other protocol is DNP3.

1053 **First limitation** The biggest problem with the testbed, however, is not  
1054 with the controller part, but with the **physical part**: first of all, it  
1055 must be said that although this is something purely demonstrative  
1056 even though it is fully functional, the implemented Simulink model  
1057 is really **oversimplified** compared to other testbeds. In fact, in the  
1058 entire system there are only three actuators, two of which are con-  
1059 nected to the same tank and controlled by the same PLC, and sensors  
1060 related only to the water level in the system's tanks: in a real sys-  
1061 tem there are many more *field devices*, which can monitor and control  
1062 other aspects of the system beyond the mere contents of the tanks.  
1063 Consider, for example, measuring and controlling the chemicals in

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

---

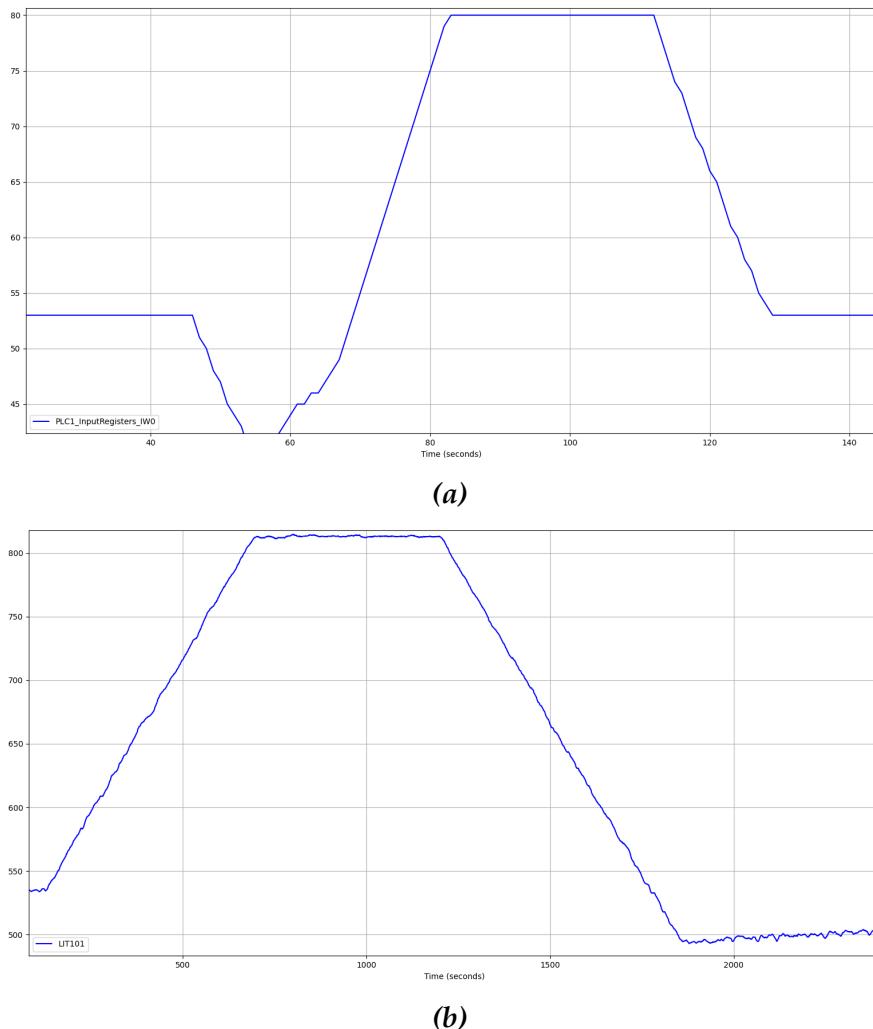
1064 the water, the pressure of the liquid in the filter unit, or more simply  
1065 the amount of water flow at a given point or time.

1066 All these must be considered and represent a number of additional  
1067 variables that makes analysis and consequently reverse engineering  
1068 of the system more difficult.

1069 ***Second limitation*** The second critical aspect concerns the **simulation of**  
1070 **the physics of the liquid** inside the tanks: Simulink does not con-  
1071 sider the fact that inside a tank that is filling (emptying) the liquid  
1072 in it undergoes **fluctuations** which cause the level sensor not to see  
1073 the water level constantly increasing (decreasing) or at most being  
1074 stable at each point of detection. Figure 3.7 exemplifies more clearly  
1075 with an example the concept just expressed: these oscillations cause  
1076 a **perturbation** in the data.

1077 This issue leads to the difficulty, on a real physical system, of **cor-**  
1078 **rectly calculating the trend of a measurement** by using the slope  
1079 attribute: if this was obtained with a too low granularity, the trend  
1080 will be oscillating between increasing and decreasing even when in  
1081 reality this would be in general increasing (decreasing) or stable; on  
1082 the other hand, if the slope was obtained with a too high granularity  
1083 there is a loss of information and the trend may be "flattened" with  
1084 respect to reality.

1085 In the present case, the slope in the Simulink model was calculated  
1086 statically with a (very) low granularity, 5 and 6 seconds according  
1087 to the Properties 7 and 9 described in the original paper: an aver-  
1088 agely careful reader will have already guessed that this granularity  
1089 is inapplicable to the real system in Figure 3.7b. As we will later see,  
1090 we need to **operate on the data perturbations** to be able to obtain a  
1091 suitable granularity and a correct calculation of the slope and conse-  
1092 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.

1093 **Pre-processing** In the pre-processing phase, the authors make use of a  
 1094 Python script to merge all the datasets of the individual PLCs into a single  
 1095 dataset, remove the (supposedly) unused registers, and finally enrich the  
 1096 obtained dataset with additional attributes. These attributes, as seen in  
 1097 3.2.2, are:

- 1098 • the **previous value** of all registers;

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

---

- 1099 • some **additional relative setpoints** named PLC $x$ \_Max\_safety and  
1100 PLC $x$ \_Min\_safety (where  $x$  is the PLC number), which represent a  
1101 kind of alert on reaching the maximum and minimum water levels  
1102 of the tanks;
- 1103 • the **measurement slope**.

1104 ***First limitation*** Merging the datasets of all individual PLCs into a single  
1105 dataset representing the entire system can be a sound practice if the  
1106 system to be analyzed is (very) small as is the testbed analyzed here,  
1107 consisting of a few PLCs and especially a few registers. If, however,  
1108 the complexity of the system increases, this type of merging can be-  
1109 come counterproductive and make it difficult to analyze and under-  
1110 stand the data obtained in subsequent steps.

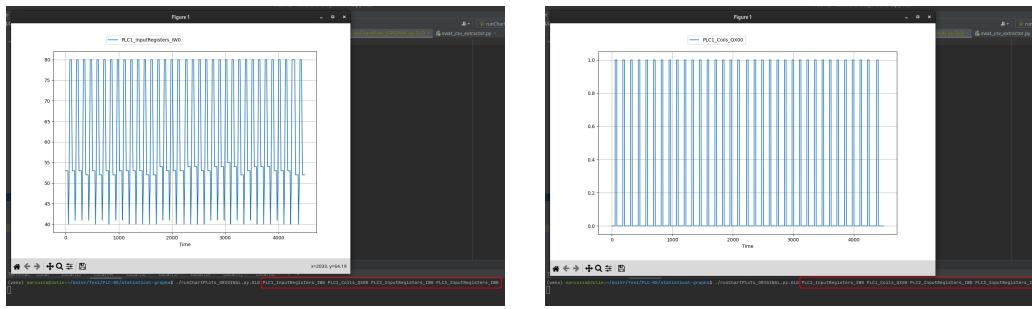
1111 In short, there is no possibility to analyze only a subsystem and thus  
1112 make the analysis faster and more understandable. Moreover, a data  
1113 gathering can take up to days, and the analyst/attacker may need to  
1114 make an analysis of the system isolating precise time ranges, ignor-  
1115 ing everything that happens before and/or after: all of this, with the  
1116 tool we have seen, cannot be done.

1117 ***Second limitation*** Regarding the additional attributes, looking at the code  
1118 of the script that performs the enrichment, we observed that **some at-**  
1119 **tributes were manually inserted** after the merging phase: we are re-  
1120 ferring in particular to the attributes PLC $x$ \_Max\_safety and PLC $x$ \_Min\_safety,  
1121 whose references were moreover hardcoded into the script, and the  
1122 *slope* whose calculation method we mentioned in the previous para-  
1123 graph about the testbed limitations.

1124 In the end, only the attribute *prev* related to the value at the previous  
1125 point of the detection is inserted automatically for all registers, more-  
1126 over without the possibility to choose whether this attribute should  
1127 be extended to all registers or only to a part.

1128 **Graphs and Statistical Analysis** Describing the behavior of graphical  
1129 analysis in Section 3.2.3 we had already mentioned that only one register

1130 plot at a time was shown and not, for example, a single window containing  
 1131 the charts of all registers entered by the user as input from the com-  
 1132 mand line, such as in Figure 3.5. Figure 3.8 shows the actual behavior  
 1133 of graphical analysis: note that although we have specified four registers  
 1134 (highlighted in red in the figures) as command-line parameters, only one  
 1135 at a time is shown and it is necessary to close the current chart in order to  
 1136 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

1137 ***First limitation*** While displaying charts for individual registers still pro-  
 1138 vides useful information about the system such as the distinction  
 1139 between actuators and measurements and the general trend of the  
 1140 latter, single display does not allow one to catch, or at least makes it  
 1141 difficult, the relationship that exists between actuators and measure-  
 1142 ments, where it exists, because a view of the system as a whole is  
 1143 missing.

1144 In this way, the risk is to make conjectures about the behavior of the  
 1145 system that may prove to be at least imprecise, if not inaccurate.

1146 ***Second limitation*** On the other hand, regarding the statistical analysis,  
 1147 two observations need to be made: the first is that for the given sys-  
 1148 tem, I personally was unable to appreciate the usefulness of the gen-  
 1149 erated histogram in Figure 3.3b, as it does not provide any particular  
 1150 new information that has not already been obtained from the graph-  
 1151 ical analysis (except maybe something marginal); the second obser-

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

1152 vation pertains to the presentation of statistical information obtained  
1153 from the histogram plot. In certain cases, the histogram plot itself  
1154 can overshadow the displayed statistical information. These statis-  
1155 tics are actually shown on the terminal from which the script is exe-  
1156 cuted. However, to an inattentive or unfamiliar user, these statistics  
1157 may be mistaken for debugging output or warnings, as they coin-  
1158 cide with the display of the histogram plot window, which takes the  
1159 focus (see Figure 3.9).

1160

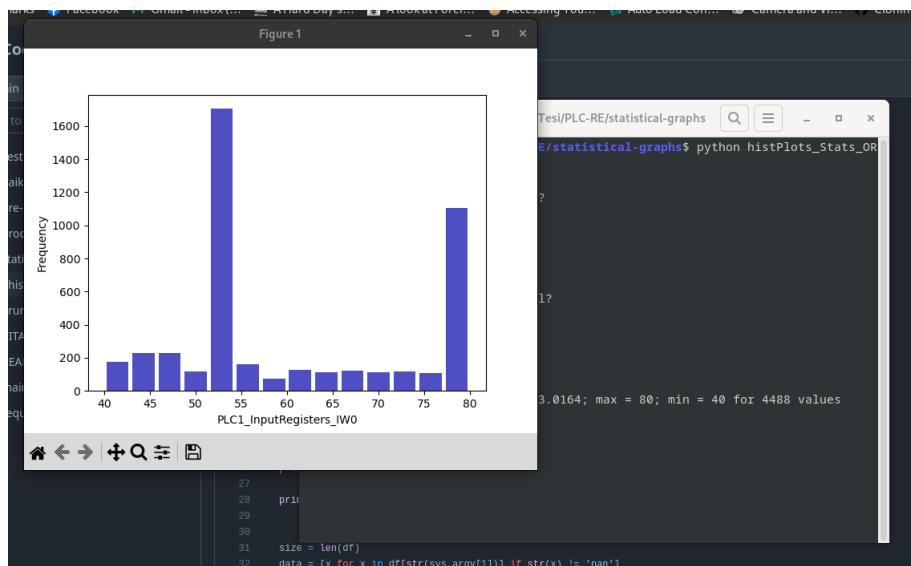


Figure 3.9: Histogram plot overshadowing statistical information shown on the terminal window in the background

1161

In general, however, little statistical information is provided.

1162 **Business Process Mining and Analysis** Concerning the data mining,  
1163 this is a purely *ad hoc solution*, designed to work under special conditions:  
1164 first, the timestamped dataset of the physical process and the one obtained  
1165 after the packet sniffing operation of Modbus traffic on the network need  
1166 to be synchronized and have the same granularity, in this case one event  
1167 per second.

1168 It is relatively easy, therefore, to find correspondences between Modbus

1169 commands sent over the network and events occurring on the physical  
1170 system, such as state changes in actuators, due in part to the fact that the  
1171 number of communications over the network is really small (see Section  
1172 3.2.1).

1173 ***First limitation*** In a real system, network communications are much more  
1174 numerous and involve many more devices even in the same second:  
1175 finding the exact correspondence with what is happening in the cy-  
1176 ber physical system becomes much more difficult.

1177 Since this is, as mentioned, an *ad hoc* solution, only the Modbus pro-  
1178 tocol is being considered: as widely used as this industrial protocol  
1179 is, other protocols that are widely used [53] such as EtherNet/IP (see  
1180 Section 2.2.6.2) or Profinet should be considered in order to extend  
1181 the analysis to other industrial systems that use a different commu-  
1182 nication network.

1183 ***Second limitation*** The other limiting aspect of the business process min-  
1184 ing phase is the **process mining software** used to generate the ac-  
1185 tivity diagram. As mentioned in Section 3.2.5, the process mining  
1186 software used by Ceccato et al. is **Disco**: this is commercial soft-  
1187 ware, with an academic license lasting only 30 days (although free of  
1188 charge), released for Windows and MacOS operating systems only,  
1189 which makes its use under Linux systems impossible except by us-  
1190 ing emulation environments such as Wine.

1191 For what is my personal vision and training as a computer scientist,  
1192 it would have been preferable to use a *cross-platform, freely licensed*  
1193 *open source* software alternative to Disco: one such software could  
1194 have been **ProM Tools** [54], a framework for process mining very  
1195 similar to Disco in functionality, but fitting the criteria just described,  
1196 or use Python libraries such as **PM4PY** [55], which offer ready-to-use  
1197 algorithms suitable for various process mining needs.

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

1198 **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

1204 **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 1205 of invariants, one invariant per line, many of no use and without 1206 correlating invariants that may have common features or deriving 1207 1208

1209 additional information from them. The process of screening and rec-  
1210ognizing the significant invariants, as well as the correlation between  
1211them, must be done by a human: certainly not an easy task given the  
1212volume of invariants one could theoretically be faced with (hundreds  
1213and hundreds of invariants). An example of Daikon's output can be  
1214seen in Figure 3.10.

1215 **Second limitation** The bash script used in this phase of the analysis does  
1216not help at all in deriving significant invariants more easily: it merely  
1217launches Daikon and saves its output to a text file by simply redirect-  
1218ing the stdout to file. No data reprocessing is done during this step.  
1219In addition, if a condition is to be specified to Daikon before perform-  
1220ing the analysis, it is necessary each time to edit the .spinfo file by  
1221manually entering the desired rule, an inconvenient operation when  
1222multiple analyses are to be performed with different conditions each  
1223time.

1224 Table 3.1 provides a summary of the limitations discussed regarding the  
1225 Ceccato et al. framework:

Phase	Limitations
Testbed	<ul style="list-style-type: none"><li>- Oversimplified model compared other testbeds</li><li>- Physics of the liquid not considered: this causes data perturbation.</li></ul>
Pre-processing	<ul style="list-style-type: none"><li>- It is not possible to select a subsystem by (groups of) PLCs or by time range</li><li>- Some additional attributes are manually inserted into dataset.</li></ul>
Graphical/Statistical Analysis	<ul style="list-style-type: none"><li>- Only one chart at the time is displayed: difficulty in capturing the relationship between actuators and sensors.</li><li>- Statistical Analysis provides little information</li></ul>

## **56 3.2 Ceccato et al.'s black-box dynamic analysis for water-tank systems**

---

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

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

## Extending and Generalizing Ceccato et al.'s Framework

1226 IN Chapter 3, we presented the state of the art of *process comprehension*  
1227 of an Industrial Control System (ICS) with a focus on the methodology  
1228 proposed by Ceccato et al. [9][Section 3.2], explaining what it consists of,  
1229 its practical application on a testbed, and most importantly highlighting  
1230 its limitations and critical issues (see Section 3.2.7).

1231 In this chapter we will present a **proposal to improve the methodology**  
1232 seen in the previous chapter, addressing most of the critical issues (or  
1233 at least trying to do so) described in Section 3.2.7 by almost completely  
1234 rewriting the original framework, enhancing its functionalities and in-  
1235 serting new ones where possible, while preserving its general structure  
1236 and approach. Indeed, the system analysis will encompass the same **four**  
1237 **phases** as the original methodology (Data Pre-processing, Graphs and Sta-  
1238 tistical Analysis, Business Process Mining and Analysis, and Invariant In-  
1239 ference and Analysis). In addition to these phases, a **fifth phase** dedicated  
1240 exclusively to **network traffic analysis** will be introduced. Each phase of  
1241 the original methodology will undergo thorough revision to enhance the  
1242 understanding of the industrial system being analyzed and its behavior,  
1243 aiming to provide a more complete and coherent process comprehension.  
1244 This revision aims to enrich the analysis process, making it more robust

1245 and transparent.

1246 Please note that our proposals do not include improvements to the data  
1247 gathering phase. This decision is solely because the new framework will  
1248 not be tested on the same case study used by Ceccato et al. (Section 3.2.1),  
1249 but rather on a different case study known as the iTrust SWaT system [36].  
1250 iTrust has already provided some datasets that contain the execution trace  
1251 of the physical system and the network traffic scan for this case study. In  
1252 contrast to the case study conducted by Ceccato et al., the iTrust SWaT  
1253 system is not simulated but rather a **real-world system**. This distinction  
1254 is important as it introduces additional complexities and considerations  
1255 when analyzing and interpreting the data.

1256 **Outline** The upcoming sections provide an outline of what to expect:

- 1257 • **Section 4.1** will introduce a **novel framework** that we have devel-  
1258 oped to address the limitations of Ceccato et al.'s framework. Sec-  
1259 tion will provide a brief overview of the framework's features and  
1260 structure;
- 1261 • in **Section 4.2** the **features of our framework** will be demonstrated  
1262 through their application to the different steps of the methodology.  
1263 Practical examples will be used to illustrate the functionality of the  
1264 framework. To facilitate this, we will utilize a more complex and  
1265 detailed testbed compared to the one employed by Ceccato et al.

## 1266 4.1 The Proposed New Framework

1267 In our version of the framework we decided to follow a few design  
1268 choices:

- 1269 1. it must be implemented in a **single programming language**;
- 1270 2. it must be **as independent as possible of the system** to be analyzed;

1271 3. It must provide greater **flexibility and ease of use** at every stage.

1272 In the following, we discuss these three features in more detail.

1273 **Single Programming Language** The implementation of Ceccato et al.'s tool  
1274 involved the use of different programming languages for each of the  
1275 phases, ranging from Python to Java, and even including Bash script-  
1276 ing.

1277 However, we believe that this heterogeneity introduces challenges  
1278 in terms of user-friendliness and intuitiveness. It becomes more dif-  
1279 ficult for users to navigate and operate the tool effectively. Fur-  
1280 thermore, the utilization of multiple technologies complicates code  
1281 maintenance and the addition of new features, especially when man-  
1282 aged by a single person who may have expertise in only one lan-  
1283 guage while being less familiar with others.

1284 Considering these factors, we have made the decision to adopt a  
1285 single programming language for the framework to ensure homo-  
1286 geneity, simplify usability, and facilitate code maintenance for fu-  
1287 ture users. Python has been chosen as the language of choice due  
1288 to its simplicity, readability, versatility, and powerfulness. Addition-  
1289 ally, Python benefits from a vast ecosystem of libraries and packages  
1290 that cater to various requirements, making it a suitable choice for our  
1291 framework.

1292 **System Independence** One of the significant limitations we identified in  
1293 Ceccato et al.'s tool, as highlighted in Section 3.2.7, is its **strong de-**  
1294 **pendence on the specific testbed** being used. This means that the  
1295 tool lacks the flexibility to configure parameters for analyzing differ-  
1296 ent industrial systems.

1297 To address this limitation and achieve system independence, we have  
1298 made a crucial improvement in our framework. We have introduced  
1299 a comprehensive configuration file called *config.ini* that allows users  
1300 to customize all the necessary parameters for analyzing the targeted

1301 system. This general configuration file eliminates any references to  
1302 hardcoded variables or values found in the Ceccato et al.'s tool, pro-  
1303 viding users with the flexibility to tailor the analysis according to  
1304 their specific requirements.

1305 **Flexibility and Ease on Use** The lack of flexibility and ease of use in a tool  
1306 can be a significant disadvantage, as it hampers its effectiveness and  
1307 makes it difficult for users to achieve their desired outcomes. Cec-  
1308 cato et al.'s tool was affected by these limitations, as it required users  
1309 to run scripts from the command line without sufficient options or  
1310 parameters for customizing the analysis. Consequently, the tool fell  
1311 short in terms of user-friendliness and failed to provide the neces-  
1312 sary flexibility to accommodate specific user requirements.

1313 To settle these issues, we enhanced the command-line interface in the  
1314 proposed framework by adding new options and parameters. These  
1315 new features provide the user with greater flexibility, enabling to  
1316 specify parameters and options that allow for more in-depth anal-  
1317 ysis and focused results analyzing data more effectively and effi-  
1318 ciently. With these enhancements, the framework has become more  
1319 user-friendly, reducing the learning curve and making it more acces-  
1320 sible to a wider range of users.

1321 This, in turn, makes the framework more valuable and useful, in-  
1322 creasing its adoption and effectiveness across a range of industrial  
1323 control systems and applications.

1324 Moreover, with new options and parameters users can now access a  
1325 range of customizable options and parameters, making the tool more  
1326 intuitive and user-friendly.

1327 Overall, the enhancements made to the framework represent a significant  
1328 step forward in making it more effective, efficient, and user-friendly.

### 1329 4.1.1 Framework Structure

1330 The proposed framework follows a similar structure to the original  
 1331 tool, with a division into five main directories representing different phases  
 1332 of the analysis: **data pre-processing, graphs and statistical analysis, pro-**  
**1333 cess mining, and invariant analysis.** A new phase is added compared to  
 1334 the original, concerning the **analysis of the network traffic.** These directo-  
 1335 ries contain the corresponding Python scripts responsible for performing  
 1336 the analysis, along with any necessary subdirectories and input/output  
 1337 files to ensure the proper functioning of the framework.

```
1338 .
1339   |-- config.ini
1340   |-- daikon
1341     |-- Daikon_Invariants
1342     |-- daikonAnalysis.py
1343     |-- runDaikon.py
1344   |-- network-analysis
1345     |-- data
1346     |-- networkAnalysis.py
1347     |-- export_pcap_data.py
1348     |-- swat_csv_extractor.py
1349   |-- pre-processing
1350     |-- mergeDatasets.py
1351     |-- system_info.py
1352   |-- process-mining
1353     |-- data
1354     |-- process_mining.py
1355   |-- statistical-graphs
1356     |-- histPlots_Stats.py
1357     |-- runChartSubPlots.py
```

*Listing 4.1: Novel Framework structure and Python scripts*

1358 The *config.ini* file is located in the root directory of the framework. This  
 1359 file holds significant importance as it grants the framework independence  
 1360 from the industrial control system being analyzed. Within this file, users  
 1361 have the opportunity to configure various general parameters and op-  
 1362 tions. These include specifying file paths for reading or writing files, as

1363 well as options related to specific analysis phases.

1364 The file is organized into sections, with each section dedicated to a spe-  
1365 cific aspect of the configuration. These sections contain user-customizable  
1366 parameters that are later referenced within the Python scripts comprising  
1367 the framework. Sections of *config.ini* are:

- 1368 • [PATHS]: defines general paths such as the project root directory;
- 1369 • [PREPROC]: includes parameters needed for the **pre-processing phase**,  
1370 like the directory containing the raw datasets of the individual PLCs  
1371 and *granularity* for the slope calculation. The granulariy is given in  
1372 terms of the time interval over which the slope is calculated;
- 1373 • [DATASET]: defines settings and parameters used during the **dataset**  
1374 **enrichment** stage, for example the additional attributes;
- 1375 • [DAIKON]: defines parameters needed for **invariant analysis** with  
1376 Daikon, e.g. directories and files containing the outcomes of the anal-  
1377 ysis;
- 1378 • [MINING]: contains parameters used during the **process mining**  
1379 phase, such as data directory;
- 1380 • [NETWORK]: includes specific settings for extracting the data ob-  
1381 tained from the packet sniffing phase on the ICS network and con-  
1382 verting it to CSV format. It also defines the **network protocols** that  
1383 are to be analyzed.

#### 1384 4.1.2 Python Libraries and External Tools

1385 As the framework has been developed entirely in Python, the objec-  
1386 tive was to minimize reliance on external tools and instead integrate vari-  
1387 ous functionalities within the framework itself. The aim was to make the  
1388 framework independent from external software. The only remaining ex-  
1389 ternal tool from the Ceccato et al. tool is Daikon. This choice was made

1390 because there is currently no better alternative or Python package avail-  
1391 able that offers the same functionalities as Daikon.

1392 Instead, the framework extensively utilizes Python libraries for han-  
1393 dling various functionalities and input data. The core libraries on which  
1394 the framework relies are:

- 1395 • **Pandas**, also used in the Ceccato et al.'s tool for dataset management,  
1396 but whose use here has been deepened and extended
- 1397 • **NumPy**, often used together with Pandas to perform some opera-  
1398 tions to support it;
- 1399 • **MatPlotLib**, for managing and plotting graphical analysis;
- 1400 • other scientific libraries such as **SciPy**, **StatsModel** [56] and **Net-**  
1401 **workX** [57], for mathematical, statistical and analysis operations on  
1402 the data;
- 1403 • **GraphViz**, for the creation of activity diagrams in the process mining  
1404 phase.

1405 Having now seen the structure of the framework, in the next sections we  
1406 will go into more detail describing our proposal.

## 1407 4.2 Analysis Phases

1408 In this section, the behavior of the proposed framework throughout the  
1409 various analysis phases of the methodology will be illustrated. Here is a  
1410 brief **outline** of the content covered in this section:

- 1411 • **Section 4.2.1:** this section will present the **testbed** used to demon-  
1412 strate examples of the framework's application;
- 1413 • **Section 4.2.2:** the introduction of the new **network traffic analysis**  
1414 phase (*Phase 0*) will be discussed. This phase aims to understand the  
1415 structure of the industrial system's network and analyze its commu-  
1416 nication patterns;

- **Section 4.2.3:** the **pre-processing** phase (*Phase 1*) will be presented, consisting of two parts: dataset merging and enrichment, followed by a preliminary analysis of the resulting dataset;
  - **Section 4.2.4:** this section focuses on the **Graphs and Statistical Analysis** phase (*Phase 2*). It will demonstrate the extraction of valuable information from the system by analyzing graphs that represent the behavior of registers over time;
  - **Section 4.2.5:** the **Invariant Inference** phase (*Phase 3*) will be explored. This phase involves deriving system information based on discovered invariants through the use of Daikon. Two examples of semi-automatic analysis will be showcased;
  - **Section 4.2.6:** the **Business Process Mining** phase (*Phase 4*) will be covered. This phase aims to gain an overview of the system's behavior and extract additional information through process mining techniques.

### <sup>1432</sup> 4.2.1 A Little Testbed: Stage 1 of iTrust SWaT System

1433 Before we proceed with presenting the analysis steps of the proposed  
1434 framework, let us introduce the testbed that will serve as an illustration for  
1435 practical examples, demonstrating the effectiveness of our methodology  
1436 and the potential of the framework. This testbed corresponds to Stage 1 of  
1437 the iTrust SWaT (Secure Water Treatment) system [36]. The selection of this  
1438 testbed is intentional, as it serves as a precursor to the comprehensive case  
1439 study we will be addressing in the upcoming chapters. The iTrust SWaT  
1440 system offers elements of greater complexity compared to the individual  
1441 stages of the Ceccato et al. testbed.

1442 The testbed comprises several components, including:

- a **tank**, which serves as the main element of interest;

- 1444 • a PLC responsible for monitoring and controlling the operations within  
1445 the stage;
- 1446 • **two sensors** that provide readings of the water level within the tank  
1447 and the incoming flow. These sensors are identified as LIT101 and  
1448 FIT101 in the PLC registers;
- 1449 • **three actuators**, namely a valve and two pumps. These actuators  
1450 regulate the level within the tank by controlling the inflow and out-  
1451 flow of the liquid. These sensors are identified as MV101 (valve), P101  
1452 and P101 (pumps) in the PLC registers.

1453 Despite its moderate complexity, this testbed provides an ideal plat-  
1454 form for presenting straightforward and concise examples of the frame-  
1455 work's behavior. It enables us to effectively demonstrate the potential  
1456 of the framework and facilitate a deeper understanding of its underlying  
1457 methodology.

### 1458 4.2.2 Phase 0: Network Analysis

1459 The objective of the network analysis presented in this section is to offer  
1460 users valuable information regarding the communication process within  
1461 an industrial control system and a broader perspective on network com-  
1462 munications, delving into previously unexplored aspects of the system.  
1463 These additional dimensions provide a deeper understanding of the sys-  
1464 tem's behavior and characteristics. This analysis aims to provide users  
1465 with an overview of the communication between PLCs at level 1 (see Fig-  
1466 ure 2.1), as well as the communication between PLCs and devices at higher  
1467 levels such as HMIs and Historian servers (see Section 2.1 for ICSs archi-  
1468 tecture). This also allows us to have a better understanding of the system  
1469 architecture and network topology. The analysis focuses on industrial pro-  
1470 tocols used and the information exchanged.

1471 By reconstructing the network communication structure using data ob-  
1472 tained from the network traffic sniffing process, users can gain a compre-  
1473 hensive understanding of the behavior of the underlying industrial sys-

1474 tem. This knowledge can then be utilized to **plan a strategy** for analyzing  
1475 the physical processes within the system.

1476 **4.2.2.1 Extracting Data from PCAP Files**

1477 The initial step involves extracting the desired information from the  
1478 PCAP files that contain the captured network traffic. This includes details  
1479 such as the source IP address, destination IP address, protocol used, and  
1480 the type of request made (e.g., Read/Write, Request/Response). The ex-  
1481 tracted data is then converted into a more convenient CSV format. This  
1482 extracted data serves as the foundation for the subsequent phase.

1483 In the latter part of this phase, the extracted data is utilized to generate  
1484 the network schema. The network schema provides a visual representa-  
1485 tion of the connections and relationships within the network, showcasing  
1486 the communication patterns between different components. This schema  
1487 helps in understanding the overall structure and behavior of the industrial  
1488 control system.

1489 To accomplish the extraction of data from the PCAP files, a Python  
1490 script called `export_pcap_data.py` is employed. This script, originally de-  
1491 signed for the business process phase, is located in the directory  
1492 `$(project_dir)/network-analysis` and accepts the following options as  
1493 command-line arguments:

- 1494 • **-f or --filename:** allows the user to specify a single PCAP file to be  
1495 passed as input to the script. The user can provide the complete file  
1496 path of the PCAP file as an argument;
- 1497 • **-m or --mergefiles:** enables the merging of multiple PCAP files. In  
1498 this scenario, the files should be located within the directory speci-  
1499 fied by the `pcap_dir` directive in the `config.ini` configuration file and  
1500 the user does not have to provide the path to each PCAP file;
- 1501 • **-d or --mergedir:** allows for specifying the directory that contains the  
1502 PCAP files to be automatically imported into the script and merged.

1503        This ensures that all the PCAP files within the specified directory will  
1504        be processed by the script without the need for manual selection or  
1505        input.

- 1506        • **-s or --singledir:** operates differently from the previous option men-  
1507        tioned. This option enables the extraction of data from each individ-  
1508        ual PCAP file within the specified directory. The extracted data is  
1509        then saved in separate CSV datasets, which are stored in the direc-  
1510        tory specified by the `split_dir` directive in the `config.ini` file. This  
1511        functionality proves useful when dealing with exceptionally large  
1512        PCAP files, where merging them together for export might consume  
1513        significant time and resources. By utilizing this option, the extrac-  
1514        tion procedure becomes lighter and more manageable. The extracted  
1515        data in separate CSV files can be utilized in the later stages of the  
1516        Network Analysis process;
- 1517        • **-t or --timerange:** this functionality enables users to specify a specific  
1518        time period within the PCAP files from which they wish to extract  
1519        relevant information.

1520        Unless the `-s` option is explicitly specified, the results of data extraction  
1521        and export will be saved to a single CSV dataset within the  
1522        `$(project_dir)/network-analysis/data` directory. The default file name  
1523        for this output file is determined by the `pcap_export_output` directive spec-  
1524        ified in the `config.ini` file. In addition, by utilizing the `protocols` and  
1525        `ws_<protocol>_field` directives, user can configure the network protocols  
1526        to be searched within the PCAP files. Furthermore, user can specify the  
1527        relevant Tshark/Wireshark fields to extract for the specified protocols set  
1528        in the `protocols` directive.

1529        After obtaining the extracted data, it is possible to proceed with the  
1530        second part of the network analysis.

1531    **4.2.2.2 Network Information**

1532       During this stage, the exported CSV data is processed to derive valua-  
1533       able information regarding network communications and the structure of  
1534       the network itself. The objective is to identify and establish relationships  
1535       between IP addresses present on the network, thereby determining the  
1536       sources and destinations of communications. Furthermore, the analysis  
1537       detects the protocols used for each communication and quantifies the var-  
1538       ious types of requests made.

1539       This information is then transformed into a **graph representation of**  
1540       **the network** (or subnetwork, if specified). In this graph, devices are repre-  
1541       sented as nodes labeled with their IP addresses, while edges represent the  
1542       incoming and outgoing communications of these devices, along with the  
1543       corresponding information.

1544       To ensure comprehensibility, the analysis also provides users with **textual**  
1545       **information** containing the same details as the graph representation. This  
1546       text-based information serves as an alternative for cases where the graph  
1547       may become complex to interpret, particularly when numerous edges con-  
1548       nect nodes and result in a high volume of network requests.

1549       This textual information is saved to another CSV file, enabling offline ref-  
1550       erence or potential future utilization. By having this file available, users  
1551       can access the network analysis results in a structured format for further  
1552       analysis or documentation purposes.

1553       The Python script `networkAnalysis.py` in the `$(project_dir)/network-analysis`  
1554       directory manages this phase of the analysis. The script can be executed  
1555       with the following parameters:

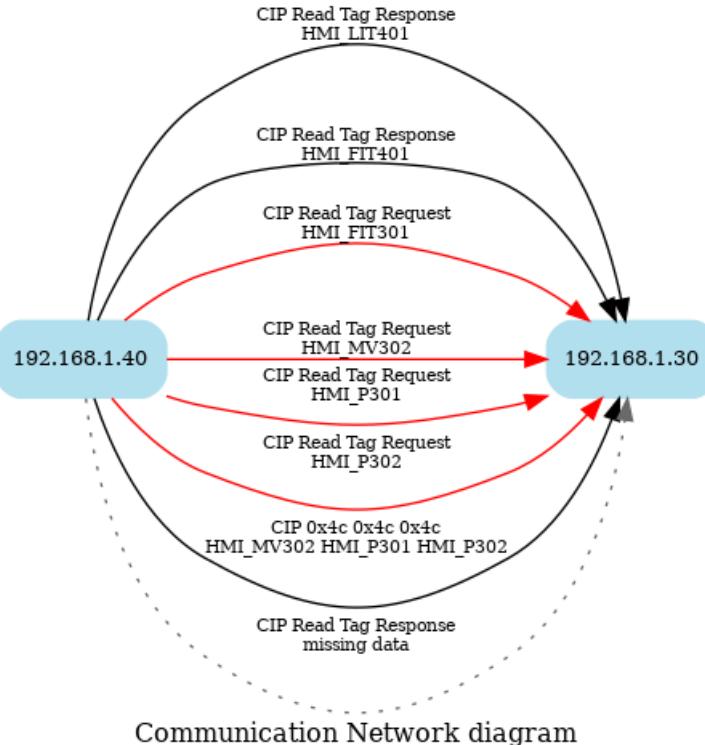
- 1556       • **-f** or **--filename**: used to specify the CSV dataset containing the net-  
1557       work data exported in the previous step. The dataset should be lo-  
1558       cated in the directory `$(project_dir)/network-analysis/data`;
- 1559       • **-D** or **--directory**: used to specify the directory that contains the CSV  
1560       datasets obtained using the `-s` option of the Python script `export_pcap_data.py`.

1561 By passing this parameter, the script will automatically merge the  
1562 datasets and proceed with the analysis of the data contained within  
1563 them;

- 1564 • **-s or --srcaddr:** allows for specifying the source IP address for which  
1565 you wish to display the incoming and outgoing communications. By  
1566 providing the source IP address as an argument, the script will focus  
1567 on showcasing the communications associated with that particular  
1568 IP address;
- 1569 • **-d or --dstaddr:** enables the user to specify the destination IP address  
1570 for which you want to display the incoming and outgoing communi-  
1571 cations. By providing the destination IP address as an argument, the  
1572 script will concentrate on presenting the communications associated  
1573 with that specific IP address.

1574 The parameters related to IP addresses, including source and destina-  
1575 tion, are optional. It is possible to specify either one of them individually.  
1576 For instance, if the user specifies only the source IP address, the script will  
1577 display the network nodes with which it communicates on the outgoing  
1578 side, along with the corresponding generated traffic. Similarly, if only the  
1579 destination IP address is specified, the script will showcase the network  
1580 nodes communicating with it on the incoming side, along with the rele-  
1581 vant traffic data.

1582 During the analysis, the script identifies and displays the IP addresses  
1583 present in the network as output for the user's reference. This allows the  
1584 user to select specific IP addresses from the command line for a more fo-  
1585 cused analysis, such as choosing a subnet of interest. Additionally, the  
1586 script detects and tracks distinct communications between pairs of PLCs,  
1587 keeping a record of the number of these communications.  
1588 The result of the analysis is a graph representation of the network (or sub-  
1589 network) to be analyzed. An example of such a graph can be seen in Figure  
1590 4.1.



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

1591        The graph illustrates the network communications between the source  
 1592        IP address 192.168.1.40 and the destination IP address 192.168.1.30. Each  
 1593        arrow represents a communication between these IP addresses, showcas-  
 1594        ing the flow and direction of the interactions. The red arrows indicate re-  
 1595        quests initiated by the source IP address towards the destination IP, while  
 1596        the black arrows represent responses sent by the source IP in response  
 1597        to previous requests made by the destination IP. The gray dotted arrows  
 1598        represent responses for which the corresponding request is missing or un-  
 1599        available for some reason. Overall, the graph distinguishes the different  
 1600        types of communications and provides insights into the request-response  
 1601        dynamics between the source and destination IP addresses. The graph is  
 1602        automatically generated and saved within the  
 1603        `$(project_dir)/network-analysis/data` directory.

1604 In terms of the textual output, we can observe how the same data is  
 1605 represented. The communications exchanged between the two PLCs are  
 1606 displayed more prominently, allowing for a clearer understanding. Unlike  
 1607 the graph, the textual representation includes a column on the right-hand  
 1608 side, indicating the number of communications for each type of request.  
 1609 This provides a more distinct perspective on the network behavior within  
 1610 an industrial control system that utilizes, in this case, the CIP protocol for  
 1611 its communications.

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

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

1612 As previously mentioned, the textual data is stored in a CSV dataset  
 1613 located in the \$(project\_dir)/network-analysis/data directory.

### 1614 4.2.3 Phase 1: Data Pre-processing

1615 *Data Pre-processing phase* is probably the most delicate and significant  
 1616 one: depending on how large the industrial system to be analyzed is, the  
 1617 data collected, and how it is enriched using the additional attributes, the  
 1618 subsequent system analysis will provide more or less accurate outcomes.

1619 The Ceccato et al.'s tool has several limitations, particularly at this  
 1620 stage. It does not allow for the isolation of a subsystem, either in terms  
 1621 of time or the number of PLCs to be analyzed. The system is considered as  
 1622 a whole without the ability to focus on specific subsystems. Additionally,  
 1623 many of the additional attributes had to be manually added, and for the  
 1624 ones entered automatically, there is no way to specify the register type to

1625 associate them with.

1626 The combination of these limitations, along with the presence of hard-  
1627 coded references to attributes and registers in the tool's code, makes the  
1628 analysis of the system more challenging. Furthermore, it compromises the  
1629 accuracy and reliability of the obtained results in terms of both quantity  
1630 and quality.

1631 In the proposed framework, these issues have been addressed by in-  
1632 corporating new features.

1633 *Firstly*, the framework allows for the **selection of a subsystem from the**  
1634 **command line** based on both temporal criteria and the specific PLCs to be  
1635 included. This enables more focused and targeted analysis.

1636 *Secondly*, we have **revamped the process of enriching** the resulting dataset  
1637 by eliminating manual entry of additional attributes. Instead, users now  
1638 have the flexibility to determine the type of additional attribute to asso-  
1639 ciate with a specific register.

1640 *Thirdly*, after the pre-processing stage, a **preliminary analysis** can be con-  
1641 ducted on the resulting dataset. This analysis aims to identify the reg-  
1642 isters that are associated with actuators, measurements, and hardcoded  
1643 setpoints or constants. It provides insights into the dataset and helps in  
1644 refining the enrichment step. The parameters for this analysis can be con-  
1645 figured in the *config.ini* file, allowing for customization and fine-tuning of  
1646 the process.

1647

1648 In the upcoming sections, we will delve into a more comprehensive ex-  
1649 amination of the achievements made in this phase.

#### 1650 4.2.3.1 Subsystem Selection

1651 In Ceccato et al.'s tool, the datasets for each individual PLC in CSV for-  
1652 mat were required to be placed in a specific directory that was hardcoded  
1653 in the script. The script would then merge and enrich these datasets to  
1654 generate a single output dataset representing the complete process trace  
1655 of the industrial system. However, the script did not provide options to

1656 select specific PLCs for analysis or define a temporal range for analysis.  
 1657 This lack of flexibility made the analysis more complex, especially when  
 1658 dealing with *transient states* (i.e., general states in which the industrial sys-  
 1659 tem is still initializing before actually reaching full operation) or when fo-  
 1660 cusing on specific parts of the industrial system during certain periods of  
 1661 interest. The fixed dataset structure also may increase the number of vari-  
 1662 ables that could be analyzed.  
 1663 Furthermore, the tool in question did *not* allow for specifying an output  
 1664 CSV file to save the resulting dataset. Each dataset creation and enrich-  
 1665 ment operation would overwrite the previous file, making it inconvenient  
 1666 for comparisons between different execution traces unless the files were  
 1667 manually renamed.

1668 The proposed framework addresses these issues by introducing im-  
 1669 provements. First of all, in the general *config.ini* file there are some general  
 1670 default settings about paths, and among them the one concerning the di-  
 1671 rectory where to place the datasets to be processed. In addition to this  
 1672 option, there are other ones that define further aspects related to the oper-  
 1673 ations performed in this phase. Listing 4.2 shows the settings in question:

```
1674 [PATHS]
1675 root_dir = /home/marcuzzo/UniVr/Tesi
1676 project_dir = %(root_dir)s/PLC-RE
1677 net_csv_path = %(root_dir)s/datasets_SWaT/2015/Network_CSV
1678
1679 [PREPROC]
1680 raw_dataset_directory = datasets_SWaT/2015 # Directory
1681     ↪ containing datasets
1682 dataset_file = PLC_SWaT_Dataset.csv # Default output
1683     ↪ dataset
1684 granularity = 10 # slope granularity
1685 number_of_rows = 20000 # Seconds to consider
1686 skip_rows = 100000 # Skip seconds from beginning
```

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

1687 At the same time, the user has the option to specify these settings via the  
 1688 command line using the new Python script called *mergeDatasets.py*, lo-

1689 cated in the pre-processing directory of the project. Any options provided  
1690 through the command line will override the default settings specified in  
1691 the *config.ini* file. These options are:

- 1692 • **-s or --skiprows:** initial transient period (expressed in seconds) to  
1693 be skipped. This option is useful in case the system has an initial  
1694 transient or the analyzer wishes to start the analysis from a specific  
1695 point in the dataset;
- 1696 • **-n or --nrows:** time interval under analysis, expressed in terms of the  
1697 number of rows in the dataset.  
1698 This option makes a **selection** on the data of the dataset;
- 1699 • **-p or --plcs:** PLCs to be merged and enriched. The user can specify  
1700 the desired PLCs by indicating the CSV file names of the associated  
1701 datasets with no limitations on number.  
1702 This option makes a **projection** on the data of the dataset.
- 1703 • **-d or --directory:** performs the merge and enrichment of all CSV files  
1704 contained in the directory specified by user, overriding the default  
1705 setting in *config.ini*. It is in fact the old functionality of the Ceccato  
1706 et al.'s tool, maintained here to give the user more flexibility and  
1707 convenience in case he wants to perform the analysis on the whole  
1708 system. This is also the default behavior in case the -p option is not  
1709 specified.
- 1710 • **-o or --output:** specifies the name of the file in which the obtained  
1711 dataset will be saved. It must necessarily be a file in CSV format.
- 1712 • **-g or --granularity:** specifies a granularity (expressed in seconds)  
1713 that will be used to calculate the measurement slope during the dataset  
1714 enrichment phase. We will discuss this later in Section 4.2.3.2.

1715    **4.2.3.2 Dataset Enrichment**

1716    After a step in which a function is applied to each PLC-related dataset  
 1717    to eliminate its unused registers within the system<sup>1</sup>, the **dataset enrichment operation** is performed.  
 1718

1719    This operation brings about several distinctions compared to the previous  
 1720    version. Firstly, it is performed on each individual dataset instead  
 1721    of the resulting dataset. Additionally, there are a greater number of attributes,  
 1722    which are automatically calculated and added to the dataset by  
 1723    the `mergeDatasets.py` script. Importantly, the configuration file `config.ini`  
 1724    under the [DATASET] section allows users to determine which registers  
 1725    should be assigned to these attributes.

1726    In Listing 4.3 we can see the list of additional attributes and how they  
 1727    should be associated with the registers of the dataset:

```
1728 [DATASET]
1729 timestamp_col = Timestamp
1730 max_prefix = max_
1731 min_prefix = min_
1732 max_min_cols_list = lit|ait|dpit
1733 prev_cols_prefix = prev_
1734 prev_cols_list = mv[0-9]{3}|p[0-9]{3}
1735 trend_cols_prefix = trend_
1736 trend_cols_list = lit
1737 trend_period = 150
1738 slope_cols_prefix = slope_
1739 slope_cols_list = lit
```

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

1740    In the following, we report a brief explanation of the parameters just seen:

1741    **timestamp\_col** denotes the name of the column in the dataset that holds  
 1742    the timestamp data. This parameter is significant not only in the  
 1743    current phase but also in the Process Mining phase. In the Ceccato et

---

<sup>1</sup>This becomes particularly relevant when conducting a Modbus register scan, where ranges of registers are examined. In this process, it is assumed that any unused registers hold a constant value of zero.

1744 al.'s work, this parameter was hardcoded and lacked configurability,  
1745 leading to errors if the analyzed system changed.

1746 **max\_prefix, min\_prefix, max\_min\_cols\_list** refer to any relative maximum  
1747 or minimum values (*relative setpoints*) of one or more measures and  
1748 that can be found and inserted as new columns within the dataset.  
1749 The first two parameters indicate the prefix to be used in the column  
1750 names affected by this additional attribute, while the third specifies  
1751 of which type of registers we want to know the maximum and/or  
1752 minimum value reached (several options can be specified using the  
1753 logical operator | - or).

1754 If, for example, we want to know the maximum value of the regis-  
1755 ters associated with the tanks, indicated in the iTrust SWaT system  
1756 by the prefix LIT, we only need to specify the necessary parameter in  
1757 the *config.ini* file, so `max_min_cols_list = lit`.

1758 The result will be to have in the dataset thus enriched a new column  
1759 named `max_LIT101`.

1760 **prev\_cols\_prefix, prev\_cols\_list** refer to the values at the previous time  
1761 instant of the registers specified in `prev_cols_list`. It is possible to  
1762 specify registers using *regex*, as in the example shown. It may be use-  
1763 ful in some cases to have this value available to check, for example,  
1764 when a change of state of a single given actuator occurs. The behav-  
1765 ior of these parameters is the same as described in the point above.

1766 **slope\_cols\_prefix, slope\_cols\_list** are related to the calculation of the  
1767 slope of a specific register that contains numeric values (usually a  
1768 measure), that is, its trend. Slope calculation makes little sense on  
1769 booleans. The slope can be **ascending** (if its value is greater than  
1770 zero), **descending** (if less than zero) or **stable** (if approximately equal  
1771 to zero). We will delve into the details of slope calculation in the fol-  
1772 lowing paragraph, as it pertains to the attributes `trend_cols_prefix`,  
1773 `trend_cols_list`, and `trend_period`.

1774 Initially, the parameters for registers to be associated with each addi-

1775 tional attribute may be left blank, as we may not have prior knowledge  
1776 about the system and are unsure about which registers correspond to ac-  
1777 tuators, measurements, or other attributes. This information can be ob-  
1778 tained from the preliminary analysis that follows the merging of datasets.  
1779 The analysis, performed based on user's choice, provides indications on  
1780 potential sensors, actuators, and other relevant information. These indica-  
1781 tions help the user set the desired values in the *config.ini* file and refine the  
1782 enrichment process by re-launching the `mergeDatasets.py` script.

1783 **Slope Calculation** The *slope* is an attribute that represents the **trend** of  
1784 the measurement being considered. It is particularly useful, in our con-  
1785 text, during the inference and invariant analysis phase to gather informa-  
1786 tion about the trend under specific conditions. The slope can generally be  
1787 classified as **increasing** ( $slope > 0$ ), **decreasing** ( $slope < 0$ ), or **stable** ( $slope$   
1788  $= 0$ ).

1789 Normally, the slope is calculated through a simple mathematical formula:  
1790 given an interval  $a, b$  relative to the measurement  $l$ , the slope is given by  
1791 the difference of these two values divided by the amount of time  $t$  that the  
1792 measurement takes to reach  $b$  from  $a$ :

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

1793 In the proposed framework, similar to the Ceccato et al.'s tool, this  
1794 time interval (the granularity) can be adjusted to be either long or short.  
1795 The choice of granularity depends on the desired accuracy of the slope  
1796 calculation. A lower granularity will provide a slope that closely reflects  
1797 the actual measurement trend, while a higher granularity will result in  
1798 flatter slope data. Each time interval within which the measurement is  
1799 divided corresponds to a slope value. These slopes are calculated and  
1800 added as additional attributes in the dataset. Later on, these slope values  
1801 are used to determine the trend of the measurement in specific situations  
1802 or conditions.

1803 Calculating the slope directly from the raw measurement data can be  
1804 a suitable approach for systems where the measurements are *not* heavily  
1805 influenced by **perturbations**. Perturbations, such as liquid oscillations in  
1806 a tank during filling and emptying phases, can lead to fluctuating read-  
1807 ings of the level. In such cases, maintaining a low granularity can provide  
1808 a more accurate calculation of the overall trend that closely aligns with  
1809 the actual measurement trend. This situation occurs, for instance, in the  
1810 testbed utilized by Ceccato et al.

1811

1812 However, if perturbations significantly affect the measurement readings,  
1813 calculating the slope on individual time intervals may result in an inaccur-  
1814 ate trend definition, irrespective of the chosen granularity. In such cases,  
1815 the fluctuating nature of the measurements due to perturbations can intro-  
1816 duce errors in the slope calculation, making it less reliable as an indicator  
1817 of the actual trend.

1818

1819 Figure 4.3 demonstrates this assertion: the measurement, in blue, refers  
1820 to the LIT101 tank of our testbed; in red, the slope calculation related to  
1821 the measurement with three different granularities: 30 (Figure 4.3a), 60  
1822 (Figure 4.3b) and 120 seconds (Figure 4.3c). It is noticeable that as the gran-  
1823 ularity increases, the slope values flatten. Moreover, in the time interval  
1824 between seconds 1800 and 4200, the level of LIT101 exhibits a predomi-  
1825 nantly increasing trend, yet the calculated slope values fluctuate between  
1826 positive and negative. Consequently, during the invariant analysis, the  
1827 overall increasing trend may not be detected, resulting in a loss of infor-  
1828 mation.

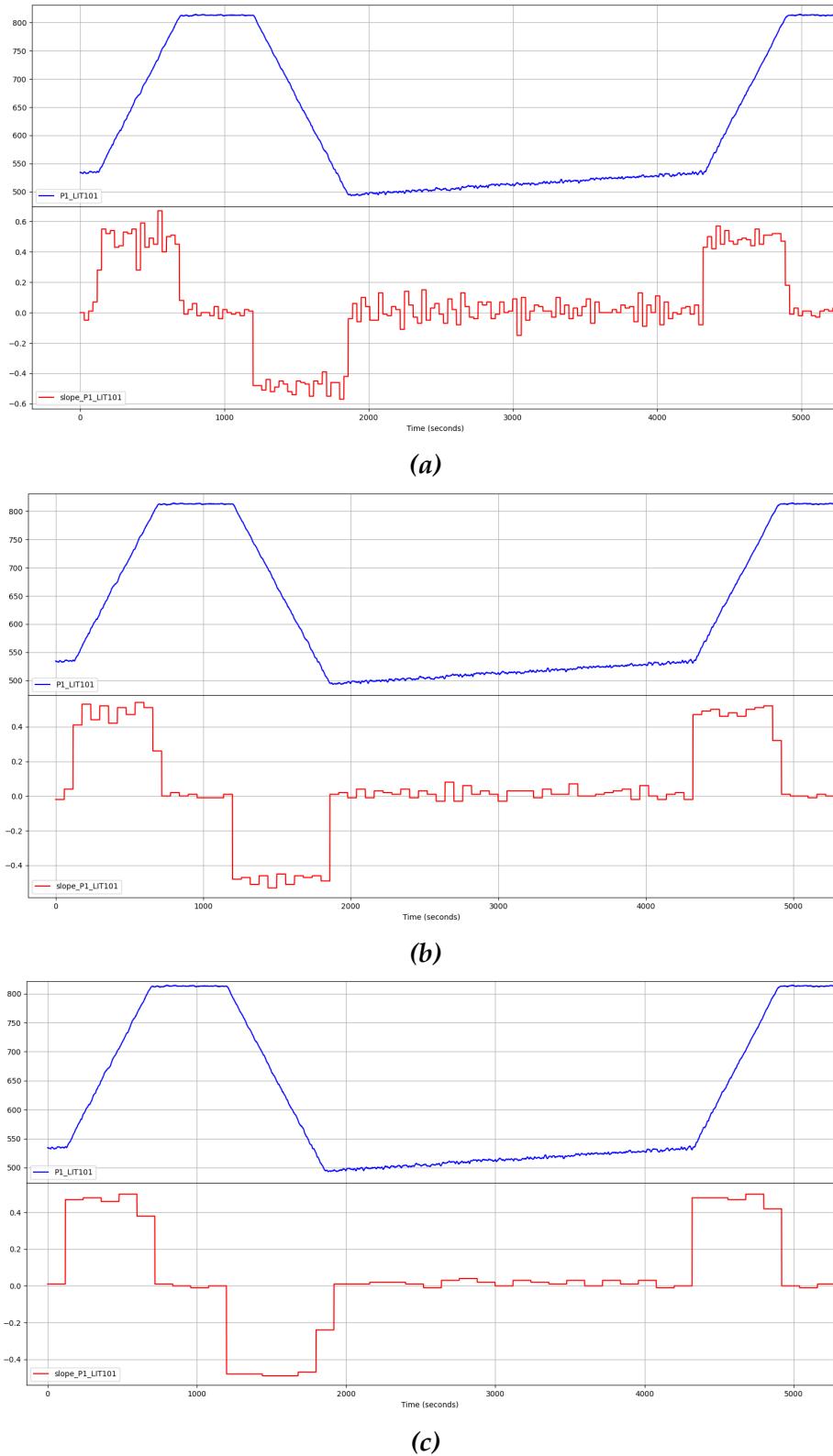


Figure 4.3: Slope comparison with granularity 30 (a), 60 (b) and 120 seconds (c)

1829 Ceccato et al.'s tool did not take into account the possibility of having  
1830 strongly perturbed data, which presented a challenge that we needed to  
1831 address in the development of the proposed framework.

1832 The solution to this problem involves applying techniques to reduce  
1833 the "noise" in the data, aiming to achieve a more linear trend in the mea-  
1834 surement curve. By minimizing the effects of perturbations, we can calcu-  
1835 late slopes more accurately.

1836 There are various methods available for smoothing out noise in the data.  
1837 In our framework, we focused on two commonly used approaches found  
1838 in the literature: **polynomial regression** and **seasonal decomposition**. In  
1839 addition to these two methods, we also explored the use of a **line simpli-  
1840 fication algorithm**.

1841  
1842 *Polynomial regression* [58] is a technique that allows us to create a filter to  
1843 reduce the impact of noise on the data. By fitting a polynomial function  
1844 to the measurements, we can obtain a smoother curve that captures the  
1845 underlying trend while minimizing the effects of perturbations.

1846 *Seasonal decomposition* [59], specifically the part related to trending, is an-  
1847 other method we explored. It involves decomposing the time series into  
1848 different components, such as trend, seasonality, and residual. By isolat-  
1849 ing the trend component, we can obtain a cleaner representation of the  
1850 underlying pattern in the data.

1851 *Line simplification algorithms* [60] aim to reduce the complexity of a polyline  
1852 or curve by approximating it with a simplified version composed of fewer  
1853 points. By selectively removing redundant or less significant points, line  
1854 simplification algorithms help reduce storage space and computational re-  
1855 quirements while preserving the overall shape and characteristics of the  
1856 original line.

1857 Regarding polynomial regression, we evaluated the use of the **Savitzky-**  
1858 **Golay filter** [61] as a smoothing technique. For seasonal decomposition,  
1859 we explored the **Seasonal-Trend decomposition using LOESS** (STL) method  
1860 [62]. For the line simplification algorithm, we specifically considered the

1861 **Ramer-Douglas-Peucker (RDP) algorithm [63].**

1862

1863 Figure 4.4 shows a quick graphical comparison of these techniques com-  
 1864 pared with the original data. The solution adopted is the *STL decomposition*  
 1865 method, which effectively reduces noise compared to the Savitzky-Golay  
 1866 filter. However, it should be noted that this method may introduce some  
 1867 delay in certain parts of the data, as is typically observed in similar algo-  
 1868 rithms. Despite its apparent effectiveness, the RDP algorithm fails to ac-  
 1869 curately approximate sections where the measurement level remains rela-  
 1870 tively stable. Consequently, it yields incorrect slope estimations, causing a  
 1871 loss of valuable information about the system.

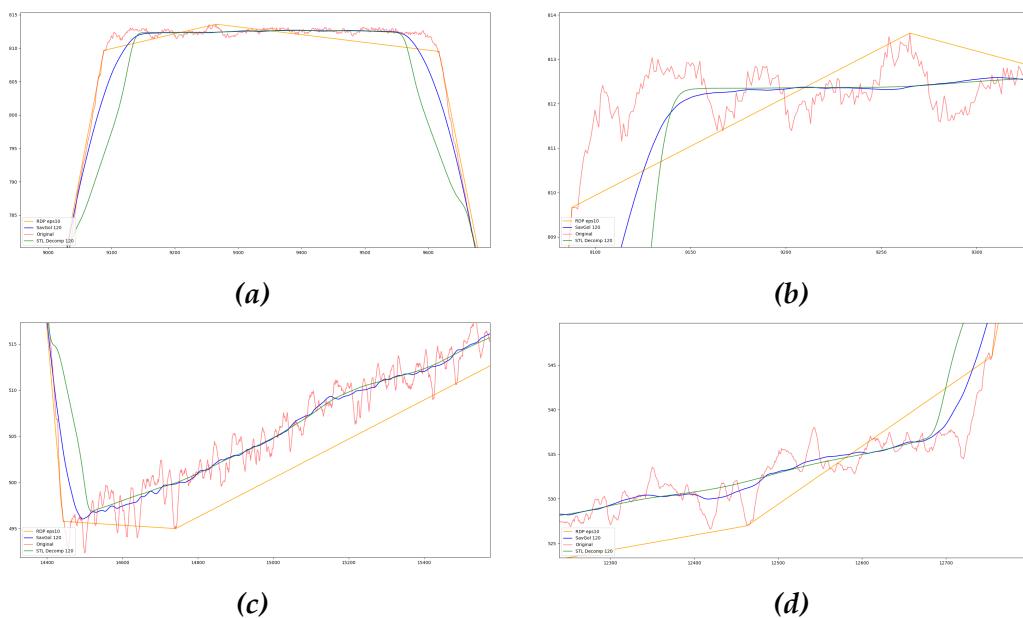
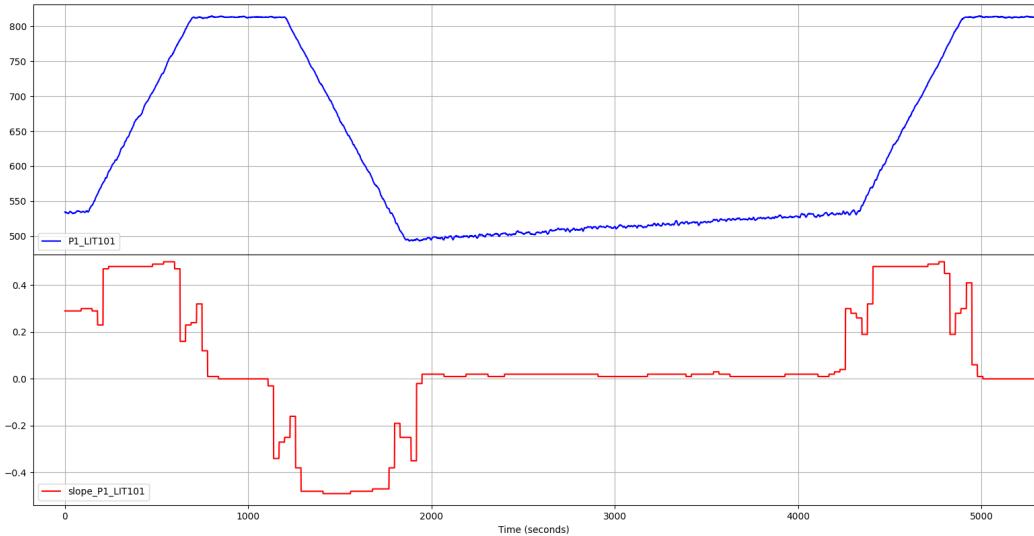


Figure 4.4: Savitzky-Golay filter (blue line), STL decomposition (green) and RDP algorithm (orange) comparison

1872

1873 By applying the STL decomposition, we observe a notable enhance-  
 1874 ment in slope calculation even when using a low granularity. Figure 4.5  
 1875 demonstrates that, with the same granularity as shown in Figure 4.3a, the  
 1876 slope values, albeit exhibiting fluctuations, consistently align with the un-  
 derlying trend of the data curve. The introduced lag resulting from the

1877 decomposition's periodicity is responsible for the observed delay.



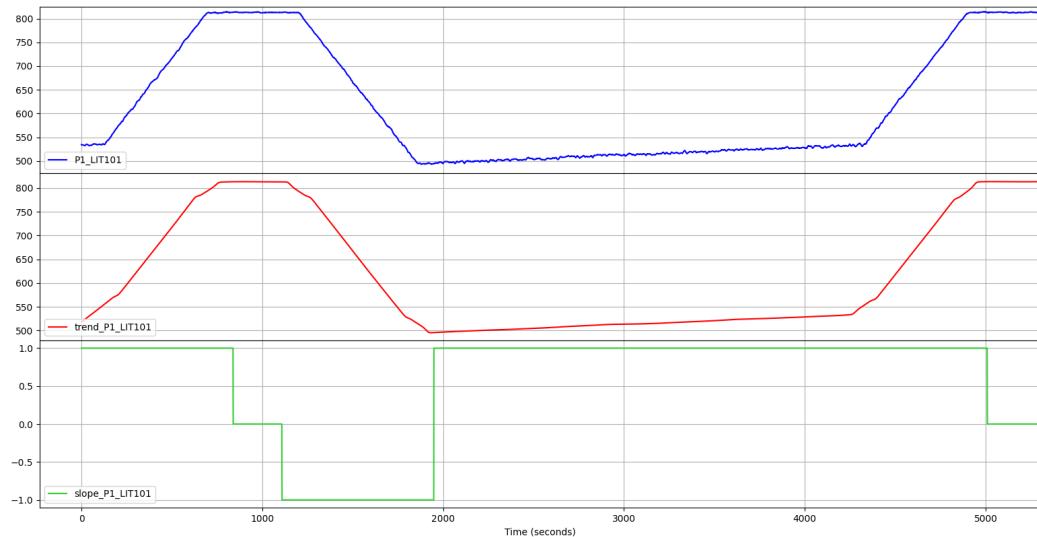
*Figure 4.5: Slope after the application of the STL decomposition*

1878 The periodicity, which defines the sampling time window for decomposi-  
 1879 tion and the level of noise smoothing, can be configured using the `trend_period`  
 1880 directive in the `config.ini` file.  
 1881 During the slope calculation, the analysis will be performed on the data  
 1882 from the additional measurement trend attributes specified in the `trend_cols_list`  
 1883 directive of the configuration file, rather than on the original unfiltered  
 1884 data.

1885 To ensure proper interpretation by Daikon, the decimal values repre-  
 1886 senting the calculated slopes are converted into **three numerical values**:  
 1887 -1, 0, and 1. These values correspond to *decreasing* (if the slope is less than  
 1888 zero), *stable* (if it is equal to zero), and *increasing* (if it is greater than zero)  
 1889 trends, respectively. Figure 4.6 displays the modified slopes along with  
 1890 the curve obtained from the STL decomposition:

#### 1891 4.2.3.3 Datasets Merging

1892 During this step, the datasets of the individual PLCs are merged, re-  
 1893 sulting in two separate datasets. The first dataset is enriched with addi-



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

1894 tional attributes but excludes the timestamp column. This dataset is in-  
 1895 tended for inference and invariant analysis. The second dataset does *not*  
 1896 contain any additional data and is specifically used in the process mining  
 1897 phase. This dataset contains the timestamp.

1898 By default, the enriched dataset will be saved in CSV format in the  
 1899 `$(project-dir)/daikon/Daikon_Invariants` directory. The other dataset,  
 1900 without additional data, will be saved in the `$(project-dir)/process-mining/data`  
 1901 directory. It's worth noting that both paths can be configured in the  
 1902 `config.ini` file. The dataset name can be specified in the `config.ini` file or  
 1903 through the `-o` command-line option. When generating the dataset for  
 1904 process mining, the script will automatically add a `_TS` suffix to the file-  
 1905 name to indicate that it includes the timestamp. This flexibility allows the  
 1906 user to provide a different filename for each output, preventing overwrit-  
 1907 ing of previous datasets. It enables the user to save the execution trace of  
 1908 the selected subsystem separately and utilize them in subsequent analysis  
 1909 phases.

1910 4.2.3.4 Preliminary Analysis of the Obtained Subsystem

1911 After merging the datasets, the user has the option to perform an **optional analysis** of the resulting dataset to extract preliminary data. This  
1912 analysis aims to gather basic information about the (sub)system and po-  
1913 tentially refine the enrichment process. If the user chooses to proceed with  
1914 the analysis, the `mergeDatasets.py` script invokes another Python script lo-  
1915 cated in the `$(project-dir)/pre-processing` directory called `system_info.py`.  
1916 Relying on an analysis based on a combination of Daikon and Pandas this  
1917 script performs a quick analysis of the dataset allowing to **estimate**, al-  
1918beit approximately, the **type of registers** (sensors, actuators, ...), also iden-  
1919tifying possible maximum and minimum values of measurements and  
1920 hardcoded setpoints. Furthermore, leveraging the use of the additional  
1921 attribute `prev_`, the `system_info.py` script is capable of deriving measure-  
1922ment values corresponding to state changes of individual actuators. This  
1923 allows for the identification of specific measurements associated with the  
1924activation or deactivation of certain actuators within the system.  
1925 As the last information we have duration of actuator states for each cy-  
1926cle of the system: this information can be useful for making assumptions  
1927and conjectures about the behavior of an actuator in a specific state or, by  
1928observing the duration values of each cycle, highlighting anomalies in the  
1929system.

1931

1932 Listing 4.4 shows an example of the output this brief analysis related to  
1933 our testbed (for brevity, only one measurement is reported in the analysis  
1934 of actuator state changes):

```
1935 Do you want to perform a brief analysis of the dataset? [y
1936 ↵ /n]: y
1937
1938 Actuators:
1939 MV101 [0.0, 1.0, 2.0]
1940 P101 [1.0, 2.0]
1941
1942 Sensors:
```

```
1943     FIT101 {'max_lvl': 2.7, 'min_lvl': 0.0}
1944     LIT101 {'max_lvl': 815.1, 'min_lvl': 489.6}
1945
1946     Hardcoded setpoints or spare actuators:
1947     P102 [1.0]
1948
1949     Actuator state changes:
1950         LIT101      MV101      prev_MV101
1951         800.7170    0          2
1952         499.0203    0          1
1953         800.5992    0          2
1954         498.9026    0          1
1955         800.7170    0          2
1956         499.1381    0          1
1957         801.3058    0          2
1958         498.4315    0          1
1959         801.4628    0          2
1960         498.1567    0          1
1961
1962         LIT101      MV101      prev_MV101
1963         805.0741    1          0
1964         805.7414    1          0
1965         805.7806    1          0
1966         805.1133    1          0
1967         804.4068    1          0
1968
1969         LIT101      MV101      prev_MV101
1970         495.4483    2          0
1971         497.9998    2          0
1972         495.9586    2          0
1973         495.8016    2          0
1974         494.5847    2          0
1975
1976         LIT101      P101      prev_P101
1977         536.0356    1          2
1978         533.3272    1          2
1979         542.1591    1          2
1980         534.8581    1          2
1981         540.5890    1          2
```

```

1982
1983      LIT101      P101      prev_P101
1984      813.0031      2          1
1985      813.0031      2          1
1986      811.8256      2          1
1987      812.7283      2          1
1988      813.3171      2          1
1989
1990      Actuator state durations:
1991      MV101 == 0.0
1992      9   9   10   9   9   10   9   9   10   9
1993
1994      MV101 == 1.0
1995      1174   1168   1182   1160   1172
1996
1997      MV101 == 2.0
1998      669   3019   3012   3000   2981
1999
2000      P101 == 1.0
2001      1073   1074   1061   1056   1056
2002
2003      P101 == 2.0
2004      118   3133   3143   3125   3108

```

*Listing 4.4: Example of preliminary system analysis*

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

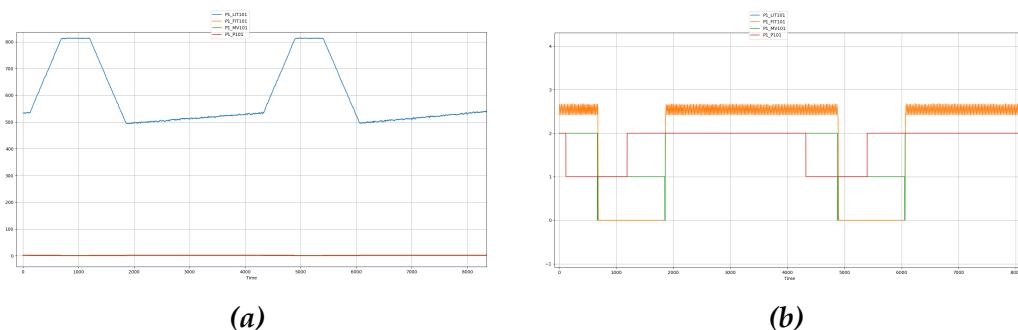
The *system\_info.py* file can also run in standalone mode if needed: it takes as command-line arguments the dataset to be analyzed, a list of actuators, and a list of sensors. For analysis related to state changes, the dataset must mandatorily be of the enriched type.

#### 2013 4.2.4 Phase 2: Graphs and Statistical Analysis

2014 The introduction of the new *graph analysis* is motivavted by from the re-  
 2015 quirement to provide users with a comprehensive overview of the (sub)system  
 2016 derived from the preceding pre-processing phase. The objective is to facil-  
 2017 itate the identification of register types, enhance the understanding of re-  
 2018 lationships, and effectively grasp the dynamics among registers controlled  
 2019 by one or more PLCs. This analysis component serves to validate initial  
 2020 conjectures made during the preliminary analysis described in the previ-  
 2021 ous section or generate new insights with the aid of visual chart represen-  
 2022 tations. It enables users to gain a deeper understanding of the system by  
 2023 leveraging the support of visual charts.

2024 In Ceccato et al.'s framework, as mentioned in Section 3.2.7, it was only  
 2025 possible to view the chart of one register at a time. While this allowed for  
 2026 the identification or hypothesis of the register type, it made it challenging  
 2027 to establish relationships with other components of the system and derive  
 2028 conjectures about their behavior. To address this limitation, there was a  
 2029 need for a new tool that could provide more information in a more acces-  
 2030 sible manner.

2031 Initially, we considered adopting an approach similar to Figure 3.5,  
 2032 where all the graphs are displayed within a single plot. However, we soon  
 2033 realized that this solution was not feasible and could not be adopted. Fig-  
 2034 ure 4.7 helps to understand why.

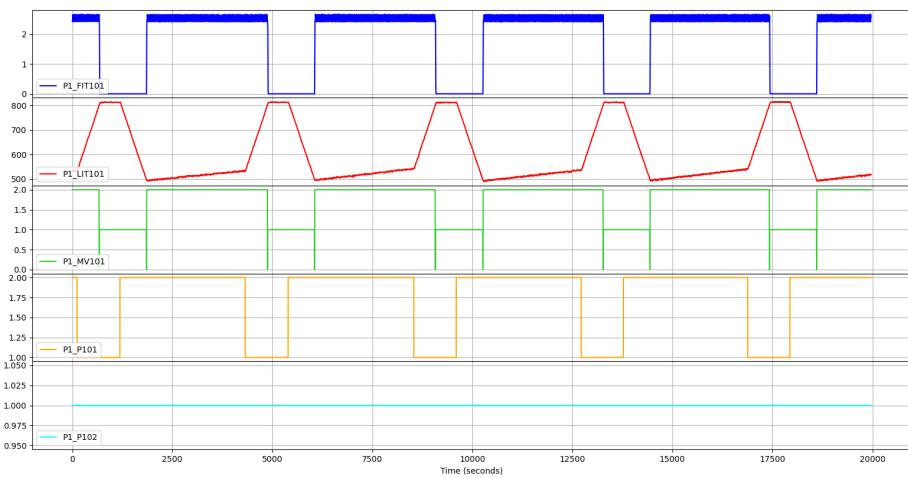


**Figure 4.7:** Plotting registers on the same y-axis

Figure 4.7a highlights the main issue with this approach, which is the use of the same y-axis for all the charts, representing the values of individual registers. When the range between register values is wide, it can result in some charts appearing as a single flat line or becoming indistinguishable, making them difficult to read. Additionally, as shown in Figure 4.7b, when registers have similar values, the graphs can become confusing and harder to interpret.

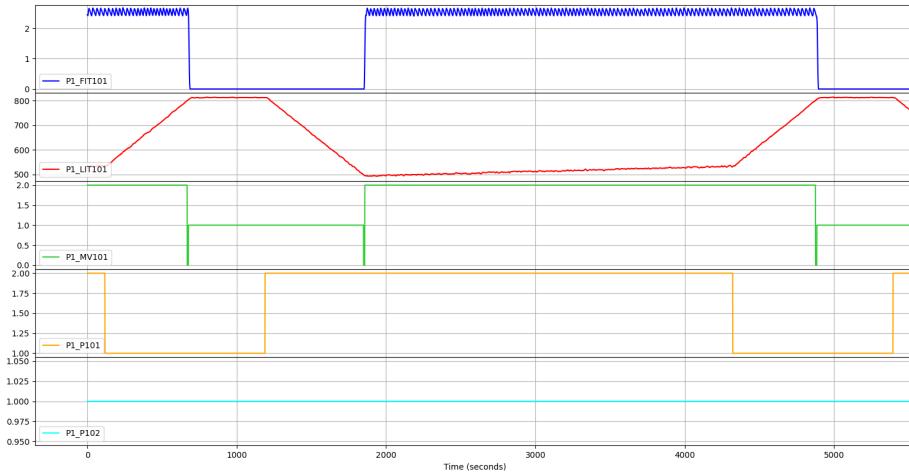
The solution to this issue is simple and effective: the use of **subplots**. Basically, each register corresponds to a subplot of the graph that shares the time axis (the x-axis) with the other subplots, but keeps the y-axis of the values of each register independent. This maintains the readability and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

Figure 4.8 illustrates more clearly what has just been explained.



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

**Figure 4.8:** Example of the new graph analysis



(b) Zooming on a particular zone of the charts

Figure 4.8: Example of the new graph analysis (cont.)

2052        As the reader has probably already noticed, the majority of the graphs  
 2053      presented in the previous sections and chapters were generated using the  
 2054      new graph analysis script, specifically the `runChartsSubPlots.py` script.  
 2055      This Python script is located in the `$(project-dir)/statistical-graphs`  
 2056      directory and utilizes the `matplotlib` libraries to generate the graph plots,  
 2057      similar to the Ceccato et al.'s tool.

2058      The script accepts the following command-line parameters:

- 2059      • **-f or --filename:** specifies the CVS format dataset to read data from.  
                 The dataset must be within the directory containing the enriched  
                 datasets for the invariant analysis phase. If subsequent parameters  
                 are not specified, the script will display all registers in the dataset,  
                 excluding any additional attributes;
- 2064      • **-r or --registers:** specifies one or more specific registers to be dis-  
                 played;
- 2066      • **-a or --addregisters:** adds one or more registers to the default visual-  
                 ization. This option is useful in case an additional attribute such as  
                 slope is to be analyzed;

2069 • **-e or --excluderegisters:** excludes one or more specific registers from  
2070 the default visualization. This option is useful to avoid displaying  
2071 hardcoded setpoints or spare registers.

2072 This script, like the previous ones, is designed to provide the maximum  
2073 flexibility and ease of use for the user, combined with greater power and  
2074 effectiveness in deriving useful information about the analyzed system.

2075 **Statistical Analysis** After careful consideration, we made the decision  
2076 not to include the statistical analysis aspect of the Ceccato et al.'s tool in  
2077 the framework. We found that there was no practical use for it. Instead, we  
2078 integrated the relevant statistical information into the preliminary analysis  
2079 conducted after the pre-processing phase. Additionally, we deemed the  
2080 histogram to have limited utility and considered it outdated in comparison  
2081 to the new graph analysis approach we implemented.

2082 Although we decided not to include the statistical analysis aspect in the  
2083 framework, the Python script `histPlots_Stats.py` from the original tool  
2084 remains in the directory. This script is essentially unchanged from the  
2085 version developed by Ceccato et al. and can be used in the future if the  
2086 need arises.

#### 2087 4.2.5 Phase 3: Invariant Inference and Analysis

2088 The phase of invariant inference and analysis has undergone a redesign  
2089 and improvement to offer the user a more comprehensive and easier ap-  
2090 proach to identify invariants. This has been achieved through the applica-  
2091 tion of new criteria to analyze and reorganize the Daikon analysis results.  
2092 The outcome of this is a more compact presentation of information that  
2093 highlights the possible relationships among invariants.

2094 The new design not only enables the identification of undiscovered aspects  
2095 of the system behavior but also confirms the hypotheses made during the  
2096 earlier stages of analysis. This step is now semi-automated, unlike before,  
2097 and allows for the analysis of invariants on individual actuator states and  
2098 their combinations.

2099    **4.2.5.1 Revised Daikon Output**

2100    To streamline the process of identifying invariants quickly and effi-  
2101    ciently, it is necessary to revise the output generated by standard Daikon  
2102    analysis. The goal is to create a more compact and readable format for the  
2103    output.

2104    The current Daikon results basically consist of three sections, referred  
2105    as *program point sections* [64]:

- 2106    1. the first section containing **general invariants**, i.e., valid regardless  
2107    of whether a condition is specified for the analysis;
- 2108    2. the second section containing **conditional invariants**, obtained by  
2109    specifying conditions for the analysis in the *.spininfo* file.
- 2110    3. a third section containing the invariants that are obtained from the  
2111    **negation of the condition** potentially specified in the *.spininfo* file.

2112    In each section only a single invariant per row is shown, without relat-  
2113    ing it in any way to the others: this makes it difficult to identify significant  
2114    invariants and any invariant chain that might provide much more infor-  
2115    mation about the behavior of the system than the single invariant.

2116    A brief example of the structure and format of this output related to  
2117    PLC1 of the iTrust SWaT system is shown in Listing 4.5, where a condition  
2118    was specified on the measurement LIT101 and on actuator MV101:

```
2119    aprogram.point:::POINT
2120     P102 == prev_P102
2121     FIT101 >= 0.0
2122     MV101 one of { 0.0, 1.0, 2.0 }
2123     P101 one of { 1.0, 2.0 }
2124     P102 == 1.0
2125     max_LIT101 == 816.0
2126     min_LIT101 == 489.0
2127     slope_LIT101 one of { -1.0, 0.0, 1.0 }
2128     [...]
2129     LIT101 > MV101
```

```

2130      LIT101 > P101
2131      LIT101 > P102
2132      LIT101 < max_LIT101
2133      LIT101 > min_LIT101
2134      [...]
2135      MV101 < min_LIT101
2136      MV101 < trend_LIT101
2137      P101 >= P102
2138      P101 < max_LIT101
2139      [...]
2140      =====
2141     aprogram.point:::POINT;condition="MV101 == 2.0 && LIT101 <
2142      ↪ max_LIT101 - 16 && LIT101 > min_LIT101 + 15"
2143      MV101 == prev_MV101
2144      P102 == slope_LIT101
2145      MV101 == 2.0
2146      FIT101 > MV101
2147      FIT101 > P101
2148      FIT101 > P102
2149      FIT101 > prev_P101
2150      MV101 >= P101
2151      MV101 >= prev_P101
2152      P101 <= prev_P101
2153      =====
2154      aprogram.point:::POINT;condition="not(MV101 == 2.0 &&
2155      ↪ LIT101 < max_LIT101 - 16 && LIT101 > min_LIT101 + 15)
2156      ↪ "
2157      P101 >= prev_P101
2158      Exiting Daikon.

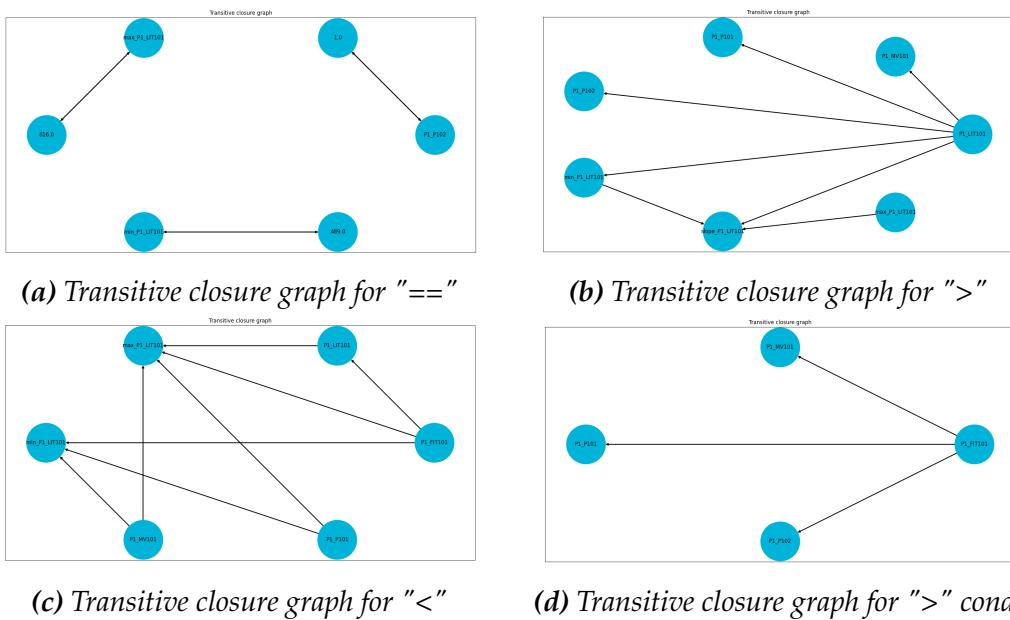
```

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

2159 In the presented framework, the output is simplified to **two sections**:  
2160 the *general section* and the section related to the *user-specified condition*. The  
2161 section related to the negated condition is eliminated as it is not relevant in  
2162 this context and could lead to potential misinterpretation. Moreover, the  
2163 relationships between invariants will be emphasized by utilizing **transitive closures**:  
2164 transitive closure of a relation  $R$  is another relation, typically  
2165 denoted  $R^+$  that adds to  $R$  all those elements that, while not necessarily

<sup>2166</sup> related directly to each other, can be reached by a *chain* of elements related  
<sup>2167</sup> to each other. In other words, the transitive closure of  $R$  is the smallest (in  
<sup>2168</sup> set theory sense) transitive relation  $R$  such that  $R \subset R^+$  [65].

To implement the transitive closure of the invariants generated by Daikon, we initially categorized the invariants in each section by type based on their mathematical relation ( $==$ ,  $>$ ,  $<$ ,  $>=$ ,  $<=$ ,  $!=$ ), excluding any invariant related to additional attributes except for the slope. For each type of invariant, we constructed a **graph** using the NetworkX library, where registers were represented as nodes, and arcs were created to connect registers that shared a common endpoint in the invariant, applying the transitive property. To reconstruct the individual invariant chains, we employed a straightforward approach known as *Depth-first Search* (DFS) on each of the graphs. This method allowed us to traverse the graphs and obtain the desired outcome, identifying the invariant chains. Figure 4.9 shows some examples of these graphs:



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

2181 At the end of this process, still applied to PLC1 of the iTrust SWaT sys-

2182 tem and with the same analysis condition, we get the following complete  
 2183 output:

```

2184 1 =====
2185 2 General
2186 3 =====
2187 4 MV101 one of { 0.0, 1.0, 2.0 }
2188 5 P101 one of { 1.0, 2.0 }
2189 6 slope_LIT101 one of { -1.0, 0.0, 1.0 }
2190 7 FIT101 != P101, P102
2191 8 P102 == 1.0
2192 9 max_LIT101 == 816.0
2193 10 min_LIT101 == 489.0
2194 11 LIT101 > MV101
2195 12 LIT101 > P101
2196 13 LIT101 > P102
2197 14 LIT101 > min_LIT101 > slope_LIT101
2198 15 FIT101 >= 0.0
2199 16 P101 >= P102 >= slope_LIT101
2200 17
2201 18 =====
2202 19 MV101 == 2.0 && LIT101 < max_LIT101 - 16 && LIT101 >
2203 20 ↢ min_LIT101 + 15
2204 21 =====
2205 21 slope_LIT101 == P102
2206 22 MV101 == 2.0
2207 23 FIT101 > MV101
2208 24 FIT101 > P101
2209 25 FIT101 > P102
2210 26 MV101 >= P101

```

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

2211 Transitive closures can be appreciated in lines 7, 14 and 16 of Listing 4.6.  
 2212 In general, the output has been reduced in the number of effective rows  
 2213 (19 versus the 61 in the output of Listing 4.5, making it certainly better to  
 2214 read and identify significant invariants) and the invariant chains make it  
 2215 more immediate to grasp the relationships between registers.

2216    **4.2.5.2 Types of Analysis**

2217    In contrast to Ceccato et al.'s solution, which involves manual individual analyses, our proposal is to introduce **two types of semi-automated analysis**.

2220    The first type focuses on analyzing **all states for each individual actuator**.  
2221    This automated analysis aims to provide comprehensive insights into the behavior of each actuator in relation to a specific measurement selected by the user.

2224    The second type of analysis considers the current system configuration  
2225    and examines the **actual states of the actuators**. This analysis takes into account the interplay and combined effects of multiple actuators on the selected measurement. By incorporating the real-time states of the actuators, this semi-automated analysis offers a more accurate assessment of the system behavior.

2230    These two types of analysis will be handled by the Python script  
2231    `daikonAnalysis.py`, contained in the default directory `$(project_dir)/daikon`.  
2232    The script accepts the following command-line arguments:

- 2233    • **-f or --filename**: specifies the enriched dataset, in CSV format, from  
2234    which to read the data. The dataset must be located within the directory containing the enriched datasets specified in the `config.ini` file  
2235    (by default `$(project_dir)/daikon/Daikon_Invariants`);
- 2237    • **-s or --simpleanalysis**: performs the analysis on the states of individual actuators;
- 2239    • **-c or --customanalysis**: performs the analysis on combinations of actual states of the actuators;
- 2241    • **-u or --uppermargin**: defines a percentage margin on the maximum value of the measurement;
- 2243    • **-l or --lowermargin**: defines a percentage margin on the minimum value of the measurement;

2245 The selection of one or both types of analysis is possible. Additionally,  
2246 the last two parameters can be used to set a condition on the value of the  
2247 measurement. This condition is designed to bypass the transient periods  
2248 that occur during actuator state changes and the actual trend changes at  
2249 the maximum and minimum values of the measurement.

2250 This approach proves particularly beneficial for the first type of analysis,  
2251 as it enables more accurate data on the trends of the measurement. By  
2252 excluding the transient periods, the analysis can focus on the stable and  
2253 meaningful trends, providing improved insights into the behavior of the  
2254 system.

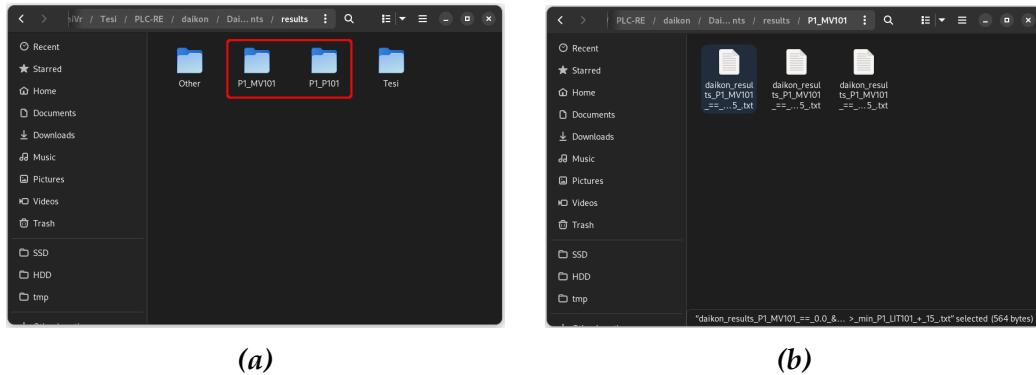
2255 **Analysis on single actuator states** Analysis on the states of individual  
2256 actuators is the simplest: after the user is prompted to input the measure-  
2257 ment, chosen from a list of likely available measurements, the script rec-  
2258ognizes the likely actuators and the relative states of each, using the same  
2259 mixed Daikon/Pandas technique adopted in the preliminary analysis dur-  
2260 ing the pre-processing phase.

2261 For each actuator and each state it assumes, a single Daikon analysis is  
2262 performed, eventually placing the condition on the maximum and mini-  
2263 mum level of the measurement.

2264 The result of these analyses are saved in the form of text files in a directory  
2265 having the name corresponding to the analyzed actuator and contained in  
2266 the default parent directory `$(project_dir)/daikon/Daikon_Invariants/results`:  
2267 each file generated by the analysis is identified by the name of the actuator,  
2268 the state and the condition, if any, on the measurement (see Figure 4.10).

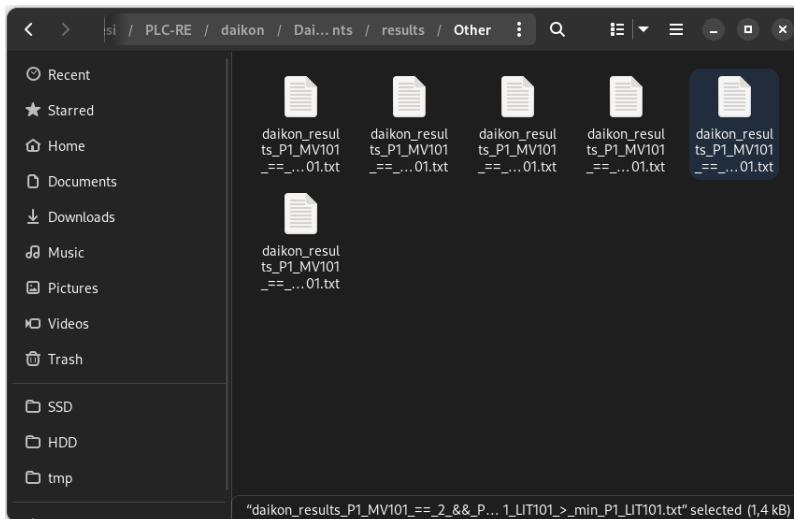
2269 Listing 4.6 provides an illustrative example of the analysis results. In  
2270 this case, we focus on the actuator MV101 in state 2.

2271 The condition-generated invariants provide the most interesting insights.  
2272 For example, at line 21, we observe that when MV101 is set to 2, the slope of  
2273 LIT101 is 1, indicating an increasing trend. This leads us to speculate that  
2274 MV101 represents the actuator's ON state and is responsible for filling the  
2275 tank (LIT101).



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

2276 **Analysis of the Current System Configuration** The analysis of the cur-  
 2277 rent system configuration based on the actual states of the actuators is  
 2278 more complex, but at the same time offers more interesting outcomes as  
 2279 it provides better evidence of actuator behavior in relation to the selected  
 2280 measurement.



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

2281 For this analysis, the script automatically identifies the configurations of  
 2282 the actuators that represent different system states (e.g., MV101 == 2, P101  
 2283 == 1). Daikon analysis is then performed for each of these configurations.

2284 The user is prompted to select the measurement attribute and, if desired,  
 2285 specific actuators for studying their configurations. If no actuators are se-  
 2286 lected, all previously detected actuators will be considered.

2287 The analysis results are saved in text format within a designated directory,  
 2288 located alongside the previous analysis outputs. The filenames of these  
 2289 result files follow a specific naming convention based on the analysis rule  
 2290 applied (see Figure 4.11).

2291

2292 An example of the obtained outcomes can be seen in Listing 4.7:

```

2293 1 =====
2294 2 General
2295 3 =====
2296 4 MV101 <= P101 ==> FIT101 >= 0.0
2297 5 MV101 <= P101 ==> MV101 one of { 0.0, 1.0, 2.0 }
2298 6 MV101 <= P101 ==> P101 one of { 1.0, 2.0 }
2299 7 MV101 <= P101 ==> slope_LIT101 one of { -1.0, 0.0, 1.0 }
2300 8 MV101 > P101 ==> FIT101 > MV101
2301 9 MV101 > P101 ==> FIT101 > P101
2302 10 MV101 > P101 ==> FIT101 > P102
2303 11 MV101 > P101 ==> FIT101 > slope_LIT101
2304 12 MV101 > P101 ==> MV101 == 2.0
2305 13 MV101 > P101 ==> MV101 > P102
2306 14 MV101 > P101 ==> MV101 > slope_LIT101
2307 15 MV101 > P101 ==> P101 == 1.0
2308 16 MV101 > P101 ==> P101 == P102
2309 17 MV101 > P101 ==> P101 == slope_LIT101
2310 18 MV101 > P101 ==> slope_LIT101 == 1.0
2311 19 [...]
2312 20
2313 21 =====
2314 22 MV101 == 2 && P101 == 1 && LIT101 < max_LIT101 && LIT101 >
2315 23 ↢ min_LIT101
2316 24 =====
2317 25 slope_LIT101 == P102 == P101 == 1.0
2318 26 MV101 == 2.0
2319 27 FIT101 > MV101

```

2320 27 FIT101 > P101

*Listing 4.7: Daikon outcomes for the system configuration MV101 == 2, P101 == 1 on LIT101*

2321 In contrast to Listing 4.6, the current analysis reveals the presence of impli-  
2322 cations in the general invariants section, which were previously absent (re-  
2323 maining generic invariants omitted for brevity). These implications offer  
2324 valuable insights, as demonstrated in this case. For instance, the invariant  
2325 stated in line 18 informs us that if the value of MV101 is greater than P101,  
2326 then the slope's value is 1, indicating an increasing tank level. This finding  
2327 further corroborates our previous understanding that the state MV101 ==  
2328 2 represents the ON state for the actuator responsible for filling the tank,  
2329 namely LIT101.

2330 **Refining the Analysis** In certain situations, the outcomes provided by  
2331 the semi-automated analyses may not meet the user's expectations. For  
2332 instance, the clarity of the slope value may be insufficient, or the user may  
2333 wish to delve deeper into a specific aspect of the system to uncover ad-  
2334 ditional invariants that were not previously identified. In such cases, the  
2335 user has the option to conduct a more targeted and specific invariant anal-  
2336 ysis using the Python script `runDaikon.py`, which enables precise investi-  
2337 gations of the system.

2338 The script, located in the default directory `$(project_dir)/daikon`, can be  
2339 executed with three command-line parameters:

- 2340 • **-f or --filename:** specifies the CSV format *enriched* dataset to read  
2341 data from. Even in this case, the dataset must be located within the  
2342 directory containing the enriched datasets;
- 2343 • **-c or --condition:** specifies the condition for the analysis, which will  
2344 be automatically overwritten in the `.spinfo` file. It is possible to spec-  
2345 ify more than one condition, but it is strongly recommended to use  
2346 the logical operator `&&` to avoid undesired behaviors in Daikon out-  
2347 comes;

- 2348 • **-r or --register:** specifies the directory where to save the text file with  
2349 the outcomes.

2350 During the execution of this analysis, a single output file is generated, con-  
2351 taining the discovered invariants. The user can specify the directory where  
2352 this file should be saved using the **-r** command-line option. By default, the  
2353 file is stored in the directory `$(project_dir)/daikon/Daikon_Invariants/results`.  
2354 The output file serves as a record of the identified invariants and can be  
2355 examined at a later time.

2356 In conclusion, the integration of these two analysis types, along with  
2357 the ability to conduct more refined analysis in the future and the enhanced  
2358 output format of Daikon, significantly enhances the completeness, clarity,  
2359 and effectiveness of this stage compared to the Ceccato et al.'s framework.

#### 2360 4.2.6 Phase 4: Business Process Analysis

2361 We have made significant revisions to the Business Process Analysis  
2362 we are presenting. Instead of relying on the previous Java solution and  
2363 proprietary process mining software Disco, we have adopted a **new inte-**  
2364 **grated solution** created in Python from scratch. The new solution utilizes  
2365 the Graphviz libraries to generate the corresponding activity diagram.

2366  
2367 In this updated Business Process, greater emphasis is placed on process  
2368 mining related to the physical system. Our goal is to extract as much  
2369 information as possible from the dataset, enabling us to promptly visu-  
2370 alize the system's behavior and its various states. This approach allows  
2371 us to validate the conjectures and hypotheses formulated in the previ-  
2372 ous phases, and potentially uncover hidden patterns that were previously  
2373 undisclosed.

2374 On the other hand, the aspect related to network communications was  
2375 reconsidered to enable operation with multiple protocols, expanding be-  
2376 yond the limitations of Modbus.

2377

2378 Now, let's examine the key aspects of this new phase in greater detail.

2379 **4.2.6.1 Process Mining of the Physical Process**

2380 The mining of the physical process is performed by the Python script  
2381 called `processMining.py`, located in the default directory `$(project_dir)/process-mining`.  
2382 This script accepts the following parameters from the command line:

- 2383 • **-f or --filename:** specifies the CSV format *timestamped* dataset to be  
2384 mined from. The dataset is obtained from the pre-processing stage  
2385 and is located in the default directory `$(project_dir)/process-mining/data`;
- 2386 • **-a or --actuators:** specifies one ore more actuators whose combina-  
2387 tions of states are to be analyzed. If this parameter is not provided,  
2388 all actuators in the subsystem will be considered;
- 2389 • **-s or --sensors:** specifies one or more measurements for which the  
2390 trend will be calculated based on actuator state changes. If this pa-  
2391 rameter is omitted, all available measurements will be considered;
- 2392 • **-t or --tolerance:** specifies the tolerance to be taken into account dur-  
2393 ing the trend calculation;
- 2394 • **-g or --graph:** shows the resulting activity diagram.

2395 The script processes the dataset in a sequential manner, analyzing each  
2396 actuator state change. It calculates the duration in seconds of the system  
2397 state, as well as the trend and slope of the specified measurement(s). Addi-  
2398 tionally, the script stores the next system state and the measurement values  
2399 corresponding to the state change points, allowing for the identification of  
2400 relative setpoints. These results are gradually collected and stored in a dic-  
2401 tionary, where the keys represent the system states based on the actuator  
2402 configurations, and the values represent the measured values mentioned  
2403 above. The dictionary is then saved as a JSON file, which can be accessed  
2404 by the user in the `$(project_dir)/process-mining/data` directory. The

2405 name of the file can be specified in the configuration file *config.ini*.

2406

2407 An example of the JSON file obtained in this step is shown in Listing 4.8:  
2408 the JSON file showcases the structure of the data obtained during the pro-  
2409 cess.

```
2410 {
2411     "MV101 == 2, P101 == 2": {
2412         "start_value_LIT101": [
2413             534.9366,
2414             495.4483,
2415             497.9998,
2416             495.9586,
2417             495.8016
2418         ],
2419         "end_value_LIT101": [
2420             536.0356,
2421             532.7384,
2422             541.7273,
2423             534.4656,
2424             540.8245
2425         ],
2426         "slope_LIT101": [
2427             0,
2428             0.015,
2429             0.018,
2430             0.016,
2431             0.018
2432         ],
2433         "trend_LIT101": [
2434             "STBL",
2435             "ASC",
2436             "ASC",
2437             "ASC",
2438             "ASC"
2439         ],
2440         "time": [
2441             119,
2442             2464,
```

```

2443     2477 ,
2444     2452 ,
2445     2440
2446     ],
2447     "next_state": [
2448     "MV101 == 2, P101 == 1",
2449     "MV101 == 2, P101 == 1",
2450     "MV101 == 2, P101 == 1",
2451     "MV101 == 2, P101 == 1",
2452     "MV101 == 2, P101 == 1"
2453   ]
2454 },
2455   "MV101 == 2, P101 == 1": {
2456   ...
2457 }
2458   "MV101 == 0, P101 == 1": {
2459   ...
2460 }
2461 ...
2462 ...
2463 }
```

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

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

2478     gests that the state is likely to occur only once within each cycle. Con-  
 2479     versely, a high standard deviation indicates that the state may occur mul-  
 2480     tiple times within the cycle.

2481     An example of the activity diagram generated by the processMining.py  
 2482     script is presented in Figure 4.12. This diagram illustrates the system's  
 2483     behavior by depicting transitions between different states. Nodes in the  
 2484     diagram represent specific system states, while arrows or edges indicate  
 2485     the flow between these states.

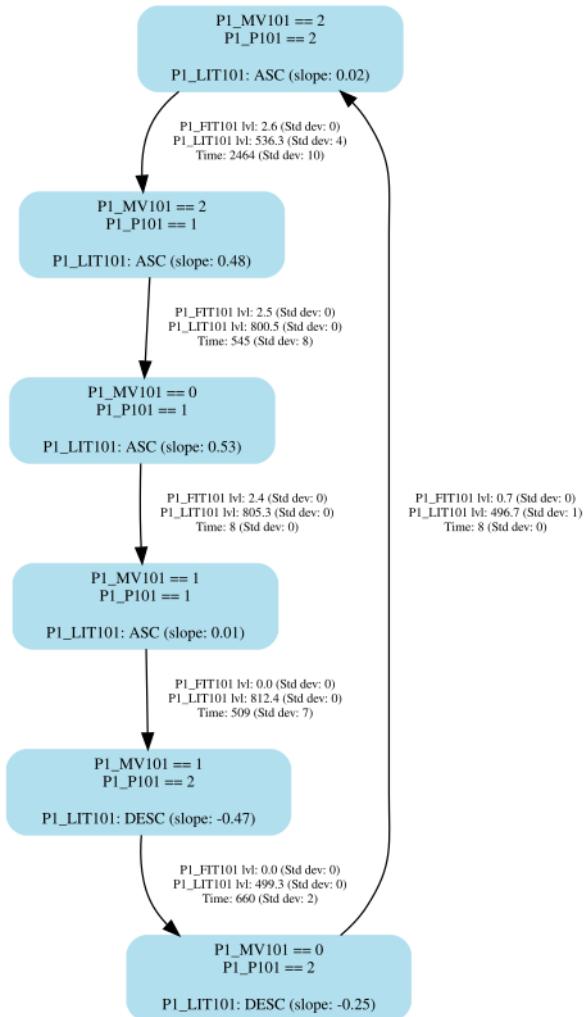


Figure 4.12: Activity diagram for PLC1 of the iTrust SWaT system

2486 The diagram reveals important aspects of the system, such as its **periodicity**  
2487 and the emphasis on the **temporal sequence of actuator states**. These  
2488 insights might not have been clearly evident in earlier stages of the anal-  
2489 ysis. The diagram provides a visual representation that allows us to ap-  
2490 preciate the recurring patterns and understand how the system evolves  
2491 over time. By observing the transitions and relationships between differ-  
2492 ent states, we gain a better understanding of the dynamics and behavior  
2493 of the system.

2494 It is important to acknowledge that despite considering the tolerance  
2495 set for trend and slope calculation, the accuracy of the related data may  
2496 vary. For instance, there could be instances where multiple trends exist  
2497 within the same node, or a trend may be stable instead of increasing or de-  
2498 creasing. These discrepancies arise due to **data perturbations**, which were  
2499 discussed in Section 4.2.3.2. Additionally, the noise reduction techniques  
2500 seen in the same Section were not employed in this analysis. Therefore, it  
2501 becomes the responsibility of the user to correctly interpret the outcomes  
2502 of this step, taking into account the information presented in the process  
2503 mining diagram and the findings from previous analyses. By consider-  
2504 ing these factors together, the user can make informed interpretations and  
2505 draw accurate conclusions from the results.

#### 2506 4.2.6.2 Network Communications

2507 The incorporation of network communications into Business Process  
2508 Analysis has been reconsidered to shift away from a *single-protocol* solu-  
2509 tion based on Modbus. Instead, the focus is on adopting a solution that  
2510 can handle **multiple protocols**, even at the same time.  
2511 The main concept was to develop a new Python script that would extract  
2512 and process data from PCAP files obtained through network traffic sniff-  
2513 ing. The intention was to export this data to a CSV file, which would then  
2514 be used as input for the process mining script, `processMining.py`, and inte-  
2515 grated with the physical system data to derive commands to the actuators.

2516     The analysis of the network data revealed that different protocols ex-  
2517     hibit distinct behaviors when commanding changes in actuator states. Con-  
2518     sequently, the previous approach proposed by Ceccato et al. becomes **im-**  
2519     **practical** when attempting to detect system state changes through network  
2520     commands sent to the actuators.

2521     However, considering that system state changes have already been iden-  
2522     tified through the process mining of the physical process, and with the  
2523     corresponding event timestamps available, we propose an alternative ap-  
2524     proach to Ceccato et al.'s method. Instead of seeking the correspondence  
2525     between network events and physical process events at the same instant,  
2526     we suggest reversing the perspective. By focusing on the correspondence  
2527     between a given event occurring in the physical process at a specific mo-  
2528     ment and the corresponding event in the network data at the same mo-  
2529     ment, it should be possible to achieve a similar, if not superior, outcome  
2530     compared to the previous solution.

2531

## Case study: the iTrust SWaT System

2532 HAVING introduced the innovative framework and highlighted its po-  
2533 tential in the preceding chapter, we now turn our attention to the  
2534 case study where we will apply this framework. As previously mentioned  
2535 in Chapter 4 and demonstrated through various examples in the same  
2536 chapter, our focus will be on the **iTrust SWaT system** [36], developed by  
2537 the iTrust – Center for Research in Cyber Security of University of Singa-  
2538 pore for Technology and Design [30]. The acronym SWaT represents *Secure*  
2539 *Water Treatment*.

2540 The iTrust SWaT system is a testbed that replicates on a small scale  
2541 a real water treatment plant arises to support research in the area of cy-  
2542 ber security of industrial control systems and has been operational since  
2543 March 2015: it is still being used by students at the University of Singapore  
2544 for educational and training purposes and is available to organizations to  
2545 train their operators on cyber physical incidents.

### 2546 5.1 Architecture

2547 In contrast to the virtualized testbed discussed in Section 3.2.1 by Cec-  
2548 cato et al., the iTrust SWaT system is composed entirely of physical hard-  
2549 ware components. It encompasses various elements, starting from field

2550 devices and extending to PLCs, HMI, SCADA workstations, and the SCADA  
2551 server (also referred to as the *historian*). The historian is responsible for  
2552 recording data from field devices for further analysis. In the upcoming  
2553 sections, we will delve deeper into the architecture of the physical process  
2554 and the communication network.

2555 **5.1.1 Physical Process**

2556 The physical process of the SWaT consists of six stages, denoted P1  
2557 through P6. These stages are [66][67]:

- 2558 P1. **taking in raw water:** feeds unfiltered water into the system
- 2559 P2. **chemical dosing:** adds chemicals to water useful for initial pretreat-  
2560 ment;
- 2561 P3. **Ultra Filtration (UF) system:** the water is filtered through a semi-  
2562 permeable membrane (ultrafiltration membrane) using the liquid pres-  
2563 sure, effectively capturing impurities and suspended solids, as well  
2564 as removing bacteria, viruses, and other pathogens present in the  
2565 water;
- 2566 P4. **dechlorination:** removes residual chlorine from disinfected water  
2567 using ultraviolet lamps;
- 2568 P5. **Reverse Osmosis (RO):** performs further filtration of the water;
- 2569 P6. **backwash process:** cleans the membranes in UF using the water pro-  
2570 duced by RO.

2571 Figure 5.1 shows a graphical representation of the architecture and the six  
2572 stages of the SWaT system.

2573 The SWaT system incorporates an array of sensors that play a crucial  
2574 role in monitoring the system's operations and ensuring their safe. These  
2575 sensors are responsible for continuously collecting data and providing  
2576 valuable insights into the functioning of the system. These sensors are:

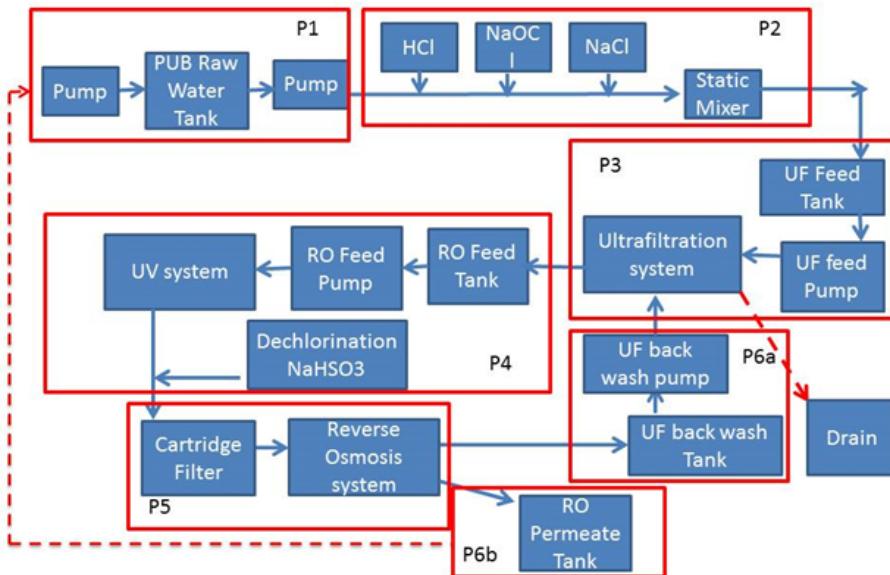


Figure 5.1: SWaT architecture

- Level Indication Transmitter (measured in mm)
- Flow Indication Transmitter (m<sup>3</sup>/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	DPII-301 0.95 kPa LL		AUT-503 264.23 µS/cm H	
	LS-203 Normal			AUT-504 14.27 µS/cm H	
	AIT-201 142.18 µS/cm L			FIT-501 0.00 m³/h L	
	AIT-202 7.20 H			FIT-502 0.00 m³/h HH	
	AIT-203 293.59 mV L			FIT-503 0.00 m³/h HH	
				FIT-504 0.00 m³/h LL	
				PIT-501 2.64 kPa LL	
				PIT-502 0.00 kPa H	
				PIT-503 0.00 kPa	

Figure 5.2: Sensors and actuators associated with each PLC

### 5.1.2 Control and Communication Network

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

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

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

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

2608 without any adverse impact on data or control functionality.  
 2609 PLCs between the different process stages at Layer 1 communicate with  
 2610 each other through a star network topology using the CIP protocol on Eth-  
 2611 erNet/IP, previously discussed in Section 2.2.6.3.

2612 Regarding zoning, the SWaT system is divided into three zones, each  
 2613 containing one or more layers. These zones are, in descending order of  
 2614 security level:

- 2615 • **Plant Control Network, or Control System:** includes layers from 0  
 2616 to 2;
- 2617 • **DMZ:** includes Layer 3.5;
- 2618 • **Plant Network:** includes Layer 3;

2619 Figure 5.3 provides a clearer visualization of the zoning and layer division  
 2620 within the network architecture of the SWaT system. This diagram high-  
 2621 lights the distinct zones and their corresponding layers, offering a com-  
 2622 prehensive overview of the system's network structure.

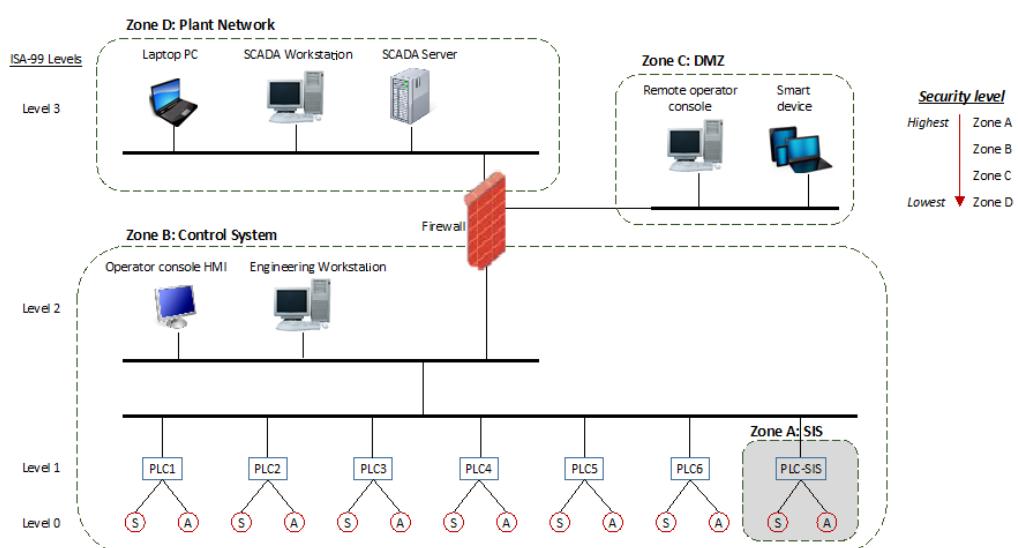


Figure 5.3: SWaT network architecture

2623 A specific IP address is associated with each device: in Table 5.1 we  
2624 report the addresses for the PLCs, historian, and Touch Panel in the *Plant*  
2625 *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*

## 2626 5.2 Datasets

2627 To facilitate the study and testing of technologies related to cyber security  
2628 in Industrial Control Systems and critical infrastructure, iTrust offers  
2629 researchers worldwide the opportunity to access a range of datasets [68].  
2630 These datasets consist of a collection of data obtained from the SWaT sys-  
2631 tem, encompassing information on both the physical processes and net-  
2632 work communications. The data is organized into different years, and  
2633 researchers can request access to these datasets for their analysis and ex-  
2634 perimentation purposes.

2635 **Physical Process Datasets** The datasets containing information about  
2636 the physical processes are provided in CSV format files. These files en-  
2637 compass data collected during different time intervals, which can vary  
2638 from a few hours to entire days. The granularity of the data is typically

2639 at a one-second interval, although there may be some exceptions. The col-  
2640 lected data primarily consists of timestamped sensor measurements and  
2641 actuator status values for each PLC, describing the physical properties of  
2642 the testbed in operational mode.

2643 **Network Communications Datasets** Network communications are typi-  
2644 cally available in the form of *Packet Capture* format (PCAP) files. These files  
2645 contain captures of communication network traffic, allowing researchers  
2646 to analyze and examine the network interactions. In some instances, CSV  
2647 files are provided instead of PCAP files, featuring different characteristics  
2648 for the collected data.

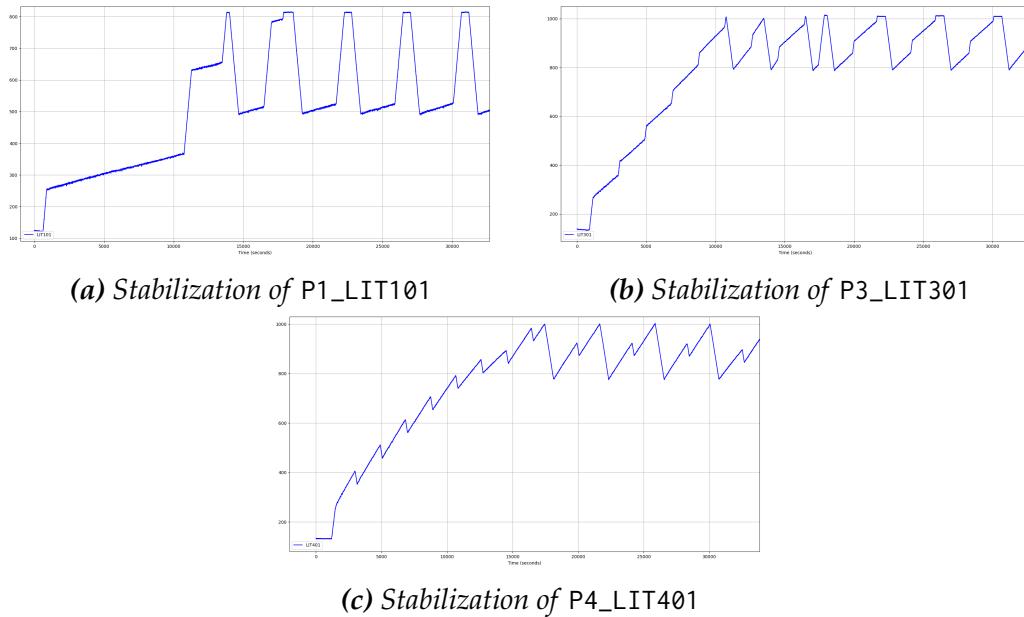
### 2649 5.2.1 Our Case Study: the 2015 Dataset

2650 The dataset selected as a case study to apply the framework discussed  
2651 in the previous chapter is specifically the dataset from the year 2015 [69].  
2652 The main reason for this choice is the unique characteristics found in the  
2653 physical process dataset that are not present in datasets from subsequent  
2654 years.

2655 **Physical Process Data** The data collection process lasted 11 consecutive  
2656 days, 24 hours per day. During the first 7 days, the system operated nor-  
2657 mally without any recorded attacks. However, attacks were observed dur-  
2658 ing the remaining 4 days. The collected data reflects the impact of these  
2659 attacks, leading to the creation of two separate CSV files: one containing  
2660 the recorded data of SWaT during the system's regular operations, and  
2661 the other containing data recorded during the days of the attacks. To en-  
2662 sure accurate information about the system, the dataset pertaining to the  
2663 normal operations, which spans seven days, was chosen for analysis.

2664 Data collection occurs at a frequency of one data point per second,  
2665 with the assumption that significant attacks cannot occur within a shorter  
2666 time frame. Additionally, the firmware of the PLCs remains unchanged  
2667 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.

2674 **Network Traffic** The network traffic was collected using an appliance  
2675 from a well-known network hardware manufacturer and was made avail-  
2676 able only in CSV format and not PCAP format. Table 5.2 shows some of  
2677 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*

2678     Unfortunately, the data provided are partial as it only includes read-  
 2679     ings from sensors and does not include information on actuator status.

2680

2681     Do not mislead by the word *Modbus* in the captured data fields: this is  
 2682     most likely a feature of the sniffing appliance, which may have encapsu-  
 2683     lated EtherNet/IP-CIP protocol packets in Modbus frames, as described  
 2684     by T. A. Snide in [70].

2685     To provide evidence that the protocol being used is CIP over EtherNet/IP,  
 2686     we can examine the "Source / Destination port" field in the captured data.

2687     The presence of TCP port 44818 as the destination port aligns with the  
 2688     TCP port used for transporting **explicit messages** in the EtherNet/IP pro-  
 2689     tocol, as explained in Section 2.2.6.2. Additionally, the "Application name",  
 2690     "Modbus Function Code" and "Modbus Function Description" fields fur-  
 2691     ther support this conclusion. The "Application name" explicitly states  
 2692     "CIP\_read\_tag\_service," while the "Modbus Function Code" field contains  
 2693     a decimal value of 76, which does not correspond to any known Modbus  
 2694     Function Calls. When converted to hexadecimal, this value is 4C, match-  
 2695     ing the CIP protocol read tag request code as indicated in the "Modbus  
 2696     Function Description" field.

2697     In summary, the datasets for subsequent years were not selected due

2698 to certain limitations. In the case of the year 2017, the physical system  
2699 data had a wide granularity, resulting in significant information loss. Ad-  
2700 ditionally, in some cases, the datasets were considered "spurious" as there  
2701 was no clear differentiation between data obtained during normal SWaT  
2702 operations and data associated with attacks.

2703 Regarding network traffic data, there were instances where the data was  
2704 either missing or fragmented into large PCAP files weighing several gi-  
2705 gabytes (more than 6 GB each!), despite containing only a few minutes'  
2706 worth of data. This made it impractical to manage such files given the  
2707 available resources.

2708 Despite their large quantity, the CSV files associated with network traffic  
2709 for the year 2015 are comparatively smaller in size, just slightly over 100  
2710 MB each. These files cover a more extensive time period, which facilitates  
2711 easier management and analysis. Prioritizing the physical process dataset  
2712 as crucial, I have selected the year 2015 as a case study, even though the  
2713 network data may be incomplete.

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

2714 IN THIS chapter, our main objective is to apply the framework and method-  
2715 ology introduced in Chapter 4 to the case study of the iTrust SWaT sys-  
2716 tem, as illustrated in Chapter 5. The purpose of this analysis is to assess  
2717 the effectiveness and potential of the proposed framework within the con-  
2718 text of a system that closely replicates a real-world water treatment plant,  
2719 albeit on a smaller scale.

2720 Due to the complexity of the system and the limited space available  
2721 in this thesis, we will not conduct a comprehensive analysis and reverse  
2722 engineering of the entire system. Instead, we will focus on specific parts  
2723 for analysis. We leave it to the reader or those interested in utilizing the  
2724 proposed methodology and framework to complete the analysis, should  
2725 they choose to do so.

2726 By focusing on selective components and leaving room for further ex-  
2727 ploration, we strike a balance between providing valuable insights and  
2728 acknowledging the potential for additional research. This approach em-  
2729 powers the reader and interested individuals to explore the iTrust SWaT  
2730 system further and leverage the proposed methodology and framework  
2731 for a more comprehensive analysis.

## 2732 6.1 Preliminary Operations

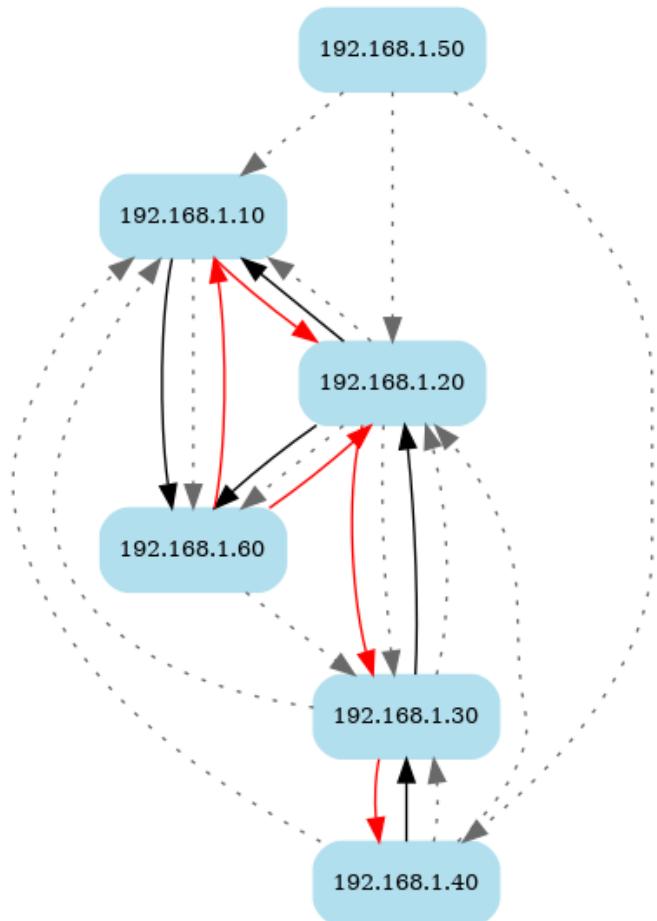
2733 Prior to beginning the actual analysis, several preliminary manual op-  
2734 erations need to be conducted on the physical process dataset utilized as  
2735 a case study, specifically the SWaT system dataset for the year 2015 as out-  
2736 lined in Section 5.2.1. To simulate the data-capture process performed by  
2737 Ceccato et al. using their scanning tool, the original dataset in XLSX format  
2738 (proprietary to Microsoft Excel) was divided into multiple datasets in CSV  
2739 format. Each of these datasets corresponds to the individual stages of the  
2740 SWaT system and contains the respective registers. These resulting files  
2741 were then saved in the directory specified by the `raw_dataset_directory`  
2742 directive in the framework configuration file, `config.ini`, ready to be used  
2743 in the pre-processing phase. Furthermore, the headers were manually re-  
2744 named by adding a prefix from P1\_ to P6\_ to each register's name. This  
2745 prefix indicates the stages, ensuring that each register is easily identifiable  
2746 and linked to its corresponding stage.

## 2747 6.2 Planning the Analysis Strategy

2748 The complexity of the system being analyzed necessitates the adoption  
2749 of a deliberate strategy for the analysis. It is not feasible to rely on trial and  
2750 error or attempt every possible combination between stages. The former  
2751 approach may overlook crucial relationships between PLCs or between  
2752 registers, while the latter may result in excessive and unproductive efforts  
2753 if the specific portion of the system being analyzed lacks significant infor-  
2754 mation or relationships. A sound analysis strategy helps us focus on the  
2755 important parts of the system, improving the quality of the analysis and  
2756 leading to better process comprehension. By prioritizing our attention, we  
2757 can gain a deeper understanding of the crucial components, resulting in  
2758 more informed decision-making and a comprehensive understanding of  
2759 the overall processes.

2760 To define this strategy, a potential starting point could involve analyz-

ing network traffic to determine the communication patterns and participants within the system. This can be accomplished by utilizing the techniques discussed in Section 4.2.2 on Network Analysis. By applying the Python script described in that section to the data extracted from the network traffic dataset debated in Section 5.2.1, we can generate a (simplified) network graph, as illustrated in Figure 6.1.



*Figure 6.1: Simplified graph of the iTrust SWaT system network*

The graph clearly illustrates the structure of communications between the PLCs. Referring back to Table 5.1, which displays the IP address - PLC associations, we can observe that PLCs 1 through 4 communicate directly and sequentially with each other in a Request/Response communication pattern (represented by red and black arrows, respectively). Additionally,

2772 PLC6 communicates with both PLC1 and PLC2. On the other hand, the  
2773 gray dotted arrows indicate communications for which we have knowl-  
2774 edge of a response, but the corresponding request is unknown. For the  
2775 purposes of our analysis strategy, we will not consider these communica-  
2776 tions within this context.

2777 Based on our observations, the analysis strategy we will adopt involves  
2778 considering sequential pairs of PLCs to effectively capture the relation-  
2779 ships and implications between registers. Therefore, the PLC pairs we  
2780 will focus on are PLC1-2, PLC2-3, and PLC3-4.

2781 **6.3 Reverse Engineering of the iTrust SWaT Sys-  
2782 tem**

2783 Before we delve into the analysis, it is important to provide some pre-  
2784 liminary remarks.

2785 **Analysis structure** Firstly, the analysis will be structured as a schematic  
2786 analysis due to space constraints, which prevent us from presenting the  
2787 extensive inferences and reasoning regarding the system in full detail.  
2788 However, the general procedure of the methodology and how to reason  
2789 about the data obtained from the framework have already been demon-  
2790 strated through examples on the PLC1 of the SWaT system in Chapter 4.  
2791 We encourage readers to refer to those examples for a more comprehen-  
2792 sive understanding. In this analysis, our focus will be on illustrating the  
2793 conjectures and properties that arise from the analysis, utilizing tables and  
2794 the outputs generated during the analysis.

2795 **Defining subsystem time duration** The second premise addresses the  
2796 process of defining the subsystems to be analyzed, which were obtained  
2797 during the pre-processing phase, during the merge phase of the individual  
2798 datasets. Apart from the projection determined by the considered PLCs,

2799 a time-based selection of the analysis period has also been performed (see  
2800 Section 4.2.3.1). This selection spans a duration of 20,000 seconds, which  
2801 is equivalent to approximately five and a half hours or roughly five sys-  
2802 tem cycles. The analysis begins at 100,000 seconds, which corresponds to  
2803 approximately 27 hours from the start of the available data. This deliber-  
2804 ate selection aims to exclude the initial transient period during which the  
2805 SWaT system is initialized. We believe that this time range is more than  
2806 sufficient for accurately defining the characteristics of the SWaT system  
2807 components.

2808 **Conventions** The third premise introduces some conventions regarding  
2809 the PLC registers that will be utilized during the analysis. These conven-  
2810 tions aid in identifying the nature and function of the registers. Specifi-  
2811 cally:

- 2812 1. Registers containing *discrete values*, such as binary or ternary values,  
2813 are likely to represent **actuators**.
- 2814 2. Registers with *continuous values* are likely to represent **measurements**.
- 2815 3. Actuators with three states are recognized as **valves**, while those  
2816 with two states are considered **pumps**. The definitions of *valve* and  
2817 *pump* in this context are arbitrary and primarily serve the purpose of  
2818 distinguishing between the two types of registers.
- 2819 4. Measurements that encompass a broader spectrum of values, fluctu-  
2820 ating between maximum and minimum thresholds, are commonly  
2821 designated as **level sensors** utilized for tank monitoring. Conversely,  
2822 Measurements characterized by a narrower range may serve other  
2823 measurement purposes, such as **flow or pressure sensing**.
- 2824 5. Registers with a single constant value are recognized as either **rela-**  
2825 **tive setpoints** or **backup actuators**. The distinction is made based  
2826 on the value contained in the register. Binary values or values corre-  
2827 sponding to the states of the main actuators indicate backup actua-

2828       tors, while different values (e.g., 20, 40, 80, etc.) suggest the presence  
2829       of relative setpoints.

2830       6. Registers with similar names, such as P1\_LIT101 and P3\_LIT301, or  
2831       P2\_MV201 and P3\_MV302, indicate the same type of register, such as a  
2832       valve, pump, level sensor, or other category.

2833       By following these conventions, it becomes easier to interpret and ana-  
2834       lyze the functionality of the PLC registers within the system.

2835       **About the Business Process Analysis** In the end, the Business Process  
2836       Analysis will focus solely on the physical process part. This is because the  
2837       datasets of network traffic captures provided by iTrust for the year 2015  
2838       (as discussed in Section 5.2.1) are **incomplete**. While communications re-  
2839       lated to measurements are present, those associated with actuators are en-  
2840       tirely missing, as well as additional communications related to other sys-  
2841       tem characteristics that we observed in the datasets of subsequent years.  
2842       As a result, we were unable to incorporate the network event recognition  
2843       component into our Business Process Analysis. To implement this com-  
2844       ponent, we would require complete and overlapping network data, along  
2845       with a clean physical process dataset not affected by system attacks. Un-  
2846       fortunately, none of the available iTrust datasets fulfill these criteria.

### 2847       6.3.1 Reverse Engineering of PLC1 and PLC2

2848       The initial focus of analysis will be on the pair comprising PLC1 and  
2849       PLC2. Let's delve into the main features of this subsystem by examining  
2850       the outcomes obtained from applying the framework to it.

#### 2851       6.3.1.1 Pre-processing - Preliminary Analysis

2852       **Measurements and Actuators Recognition** Listing 6.1 shows the out-  
2853       comes obtained from automatic recognition of likely measurements and  
2854       actuators:

```

2855   1 Actuators:
2856   2 P1_MV101 [0.0, 1.0, 2.0]
2857   3 P1_P101 [1.0, 2.0]
2858   4 P2_MV201 [0.0, 1.0, 2.0]
2859   5 P2_P203 [1.0, 2.0]
2860   6 P2_P205 [1.0, 2.0]
2861   7
2862   8 Sensors:
2863   9 P1_FIT101 {'max_lvl': 2.7, 'min_lvl': 0.0}
2864  10 P1_LIT101 {'max_lvl': 815.1, 'min_lvl': 489.6}
2865  11 P2_AIT201 {'max_lvl': 256.5, 'min_lvl': 252.9}
2866  12 P2_AIT202 {'max_lvl': 8.4, 'min_lvl': 8.3}
2867  13 P2_AIT203 {'max_lvl': 342.8, 'min_lvl': 320.0}
2868  14 P2_FIT201 {'max_lvl': 2.5, 'min_lvl': 0.0}
2869  15
2870  16 Hardcoded setpoints or spare actuators:
2871  17 P1_P102 [1.0]
2872  18 P2_P201 [1.0]
2873  19 P2_P202 [1.0]
2874  20 P2_P204 [1.0]
2875  21 P2_P206 [1.0]

```

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

Based on the results shown in Listing 6.1, the framework has recognized P1\_MV101, P1\_P101, P2\_MV201, P2\_P203, and P2\_P205 as **likely actuators**. The actuators indicated by the *Pxxx* string are binary actuators, meaning they have two states represented by the values 1 and 2. On the other hand, the actuators identified by the *MVxxx* string are ternary actuators with three distinct states: 0, 1, and 2. According to the third premise of Section 6.3, for point 3, actuators denoted by the *Pxxx* string will be classified as **pumps**, while those labeled with *MVxxx* will be recognized as **valves**.

P1\_FIT101, P1\_LIT101, P2\_AIT201, P2\_AIT202, P2\_AIT203, and P2\_FIT201 have been identified as **likely measurements**. Each measurement provides a specific range of values, spanning from a maximum to a minimum value. Based on the range of values reported by these measurements and

according to the third premise of Section 6.3, for point 4, it is reasonable to infer that P1\_LIT101 could potentially be a **level sensor for a tank**.

In addition, there are some registers that have been recognized as **hard-coded setpoints or spare actuators** due to their constant values. These registers share a resemblance to the previously identified pump registers. According to points 5 and 6 of the third premise in Section 6.3, it is reasonable to assume that these registers correspond to **spare actuators**. Furthermore, the fact that they have a constant value of 1 indicates that they might represent the **OFF state** of the pumps.

**Actuator State Durations** To gain a deeper understanding of the different states (0, 1, and 2) associated with valves P1\_MV101 and P2\_MV201, we can analyze the duration of each state. Listing 6.2 provides information regarding the duration (in seconds) of states for these specific actuators:

```

2902 1 Actuator state durations:
2903 2 P1_MV101 == 0.0
2904 3 9 9 10 9 9 10 9 9 10 9
2905 4
2906 5 P1_MV101 == 1.0
2907 6 1174 1168 1182 1160 1172
2908 7
2909 8 P1_MV101 == 2.0
2910 9 669 3019 3012 3000 2981
2911 10
2912 11 P2_MV201 == 0.0
2913 12 8 8 8 9 9 8 9 9 9 9
2914 13
2915 14 P2_MV201 == 1.0
2916 15 1057 1057 1045 1038 1039
2917 16
2918 17 P2_MV201 == 2.0
2919 18 120 3135 3144 3127 3109

```

*Listing 6.2: Time duration of the states of actuators P1\_MV101 and P1\_MV201 of PLC1-2*

It is evident that the duration of **state 0** is relatively short, averaging

2921 around 8-10 seconds, while the other states have much longer durations.  
 2922 This observation suggests that state 0 of a valve is a **transient state**, in-  
 2923 dicating a transitional phase within the valve cycle. However, without  
 2924 further information, it is currently not possible to determine the specific  
 2925 position of state 0 within the overall valve cycle.

2926 **Actuator State Changes** Now that we have identified P1\_LIT101 as the  
 2927 supposed tank, we can examine the trend of the tank level as the actuators  
 2928 change state. Listing 6.3 provides information on the levels of the tank in  
 2929 correlation with the state changes of the P1\_P101 pump:

2930	1	P1_LIT101	P1_P101	prev_P1_P101
2931	2	536.0356	1	2
2932	3	533.3272	1	2
2933	4	542.1591	1	2
2934	5	534.8581	1	2
2935	6	540.5890	1	2
2936	7			
2937	8	P1_LIT101	P1_P101	prev_P1_P101
2938	9	813.0031	2	1
2939	10	813.0031	2	1
2940	11	811.8256	2	1
2941	12	812.7283	2	1
2942	13	813.3171	2	1

*Listing 6.3: P1\_P101 state changes in relation to P1\_LIT101*

2943 Based on the speculation that state 1 represents the OFF state of the  
 2944 pump and state 2 represents the ON state, we can analyze the data in List-  
 2945 ing 6.3. When pump P1\_P101 transitions from the ON state to the OFF  
 2946 state, the average level of P1\_LIT101 is 535. On the other hand, when  
 2947 P1\_P101 goes from the OFF state to the ON state, the average level of  
 2948 P1\_LIT101 is 813. These values correspond to the **minimum and maxi-**  
 2949 **mum relative setpoints** of P1\_P101, respectively.

2950 Based on this information, we can infer that pump P1\_P101 is respon-  
 2951 sible for **emptying the tank**. Moreover, it can be extended to assume that  
 2952 a pump, in general, is responsible for **water outflow**.

2953 Applying the same analysis to the data for valve P1\_MV101, which is not  
2954 reported for conciseness, we can speculate that P1\_MV101 is responsible for  
2955 **filling the tank**. In this case, states 1 and 2 would represent the **OFF and**  
2956 **ON states of the valve**, respectively. The relative setpoints of P1\_MV101  
2957 are approximately 500 (minimum) and 800 (maximum). By extending this  
2958 analysis, we can speculate that a valve, such as P1\_MV101, is responsible  
2959 for controlling the **water inflow**.

2960 Regarding the elements controlled by PLC2 and the sensor P1\_FIT101,  
2961 the analysis does not reveal the presence of another tank. Therefore, we  
2962 cannot determine the exact role of sensors P2\_AIT20x, P1\_FIT101 and P2\_FIT201  
2963 at this point. However, there is a similarity observed between the relative  
2964 setpoints of P1\_P101 and those of P2\_MV201, P2\_P203, and P2\_P205. These  
2965 registers exhibit very similar values during state changes, suggesting a  
2966 potential relationship or similar control behavior between them.

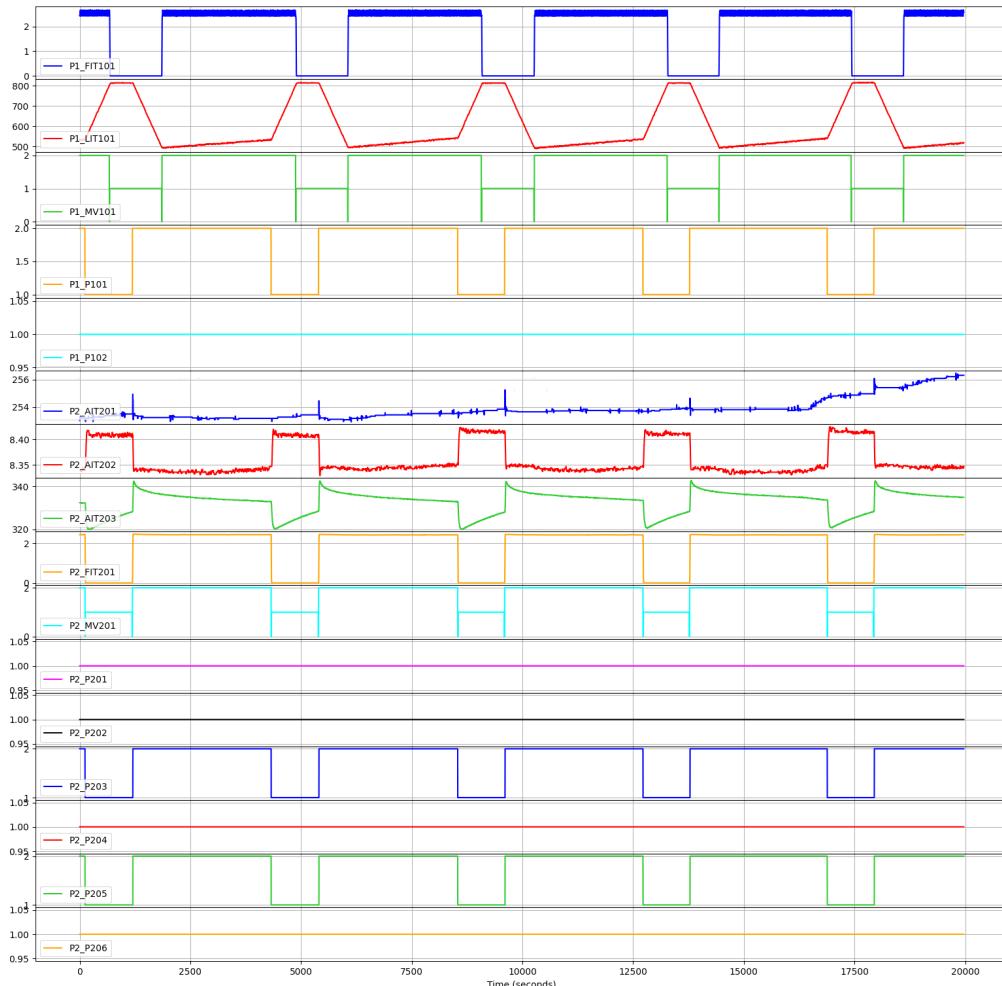
#### 2967 6.3.1.2 Graphs and Statistical Analysis

2968 Figure 6.2 illustrates the graphical representation of the registers in  
2969 PLC1-2 and their respective trends.

2970 The image provides additional support to the conjectures made during  
2971 the preliminary analysis regarding the spare actuators. Furthermore, it is  
2972 evident from the graphs that these spare actuators **do not appear to influ-**  
2973 **ence the trend** of any of the measurements. Therefore, based on this obser-  
2974 vation, we can confidently exclude these registers from further graphical  
2975 analysis.

2976 Figure 6.3 shows a clearer representation of the subsystem after remov-  
2977 ing the spare actuators.

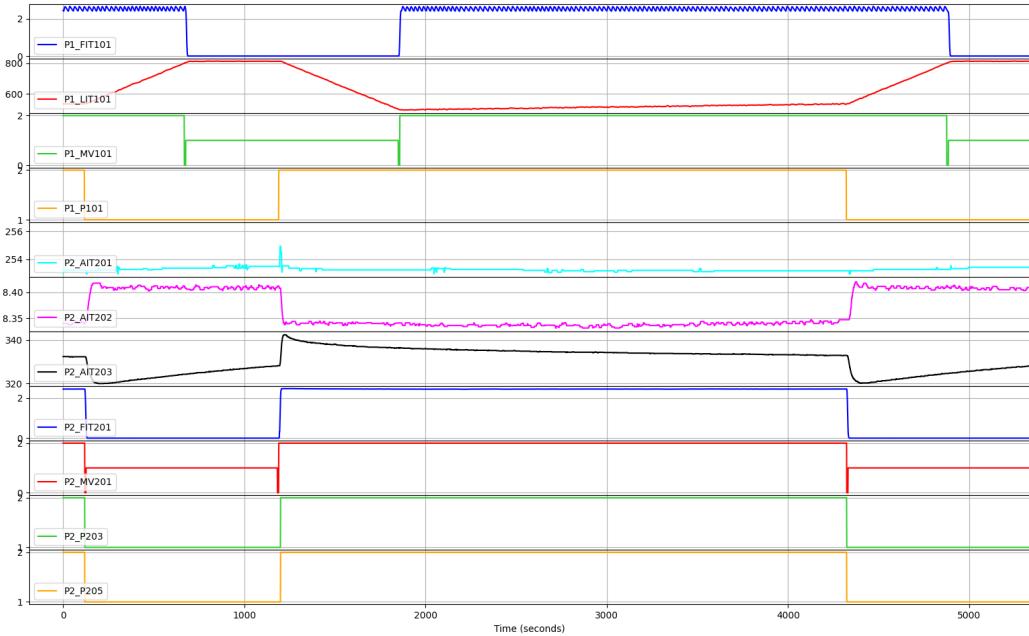
2978 Figure provides furthermore additional insights that allow us to spec-  
2979 ulate on aspects that remained unexplained during the preliminary analy-  
2980 sis. The most prominent aspect that stands out is the relationship between  
2981 the level behavior of P1\_LIT101 and the states of P1\_P101 and P1\_MV101. It  
2982 becomes evident that these two actuators **are not mutually exclusive**, and



*Figure 6.2: Chart of PLC1-2 registers*

when they are both in the ON state, the growth of the tank level is slow. However, as soon as P1\_P101 switches to the OFF state, the tank level starts to increase at a faster rate. Conversely, when P1\_MV101 switches to the OFF state, the tank level decreases. During periods when both actuators are in the OFF state, there is a *plateau* where the tank level remains relatively stable.

Furthermore, we observe a relationship between P1\_FIT101 and the trend of P1\_MV101. When the valve is in the off state, P1\_FIT101 registers a value of 0, whereas it registers a value greater than 2 when the valve is



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

2992 open. This suggests that P1\_FIT101 could be a **sensor associated with the**  
 2993 **flow** of water entering the tank, which is represented by P1\_LIT101. By  
 2994 drawing an analogy with its name, it is plausible to consider P2\_FIT201 as  
 2995 another flow sensor.

2996 Another intriguing aspect that arises from examining the graphs in Figure  
 2997 6.2 and Figure 6.3 is the **non-cyclic pattern** observed in the P2\_AIT201  
 2998 measurement. Instead of exhibiting a cyclical trend, it follows a linear  
 2999 pattern. Furthermore, considering the narrow range of values associated  
 3000 with this measurement, it is reasonable to speculate that P2\_AIT201 may be  
 3001 associated with a **sensor that measures a specific property of the water**.

3002 The limited range of values observed in P2\_AIT202 also raises the pos-  
 3003 sibility that it functions as a sensor for a particular water characteristic,  
 3004 despite exhibiting a cyclic pattern. As for P2\_AIT203, its role remains un-  
 3005 defined, although it also displays a cyclic trend. This trend, along with that  
 3006 of P2\_AIT202, does not appear to be related to the behavior of the valves  
 3007 (which rules out the possibility of it being related to a tank), but rather

3008 to that of the pumps. Consequently, it is imperative to conduct further  
3009 investigations into these aspects.

3010 By examining the trend of valve P2\_MV201, an additional speculation  
3011 can be made. It appears to be independent of the trends observed in any  
3012 of the measurements within this subsystem. Based on previous conjectures  
3013 regarding the role of valves and considering the duration of its ON  
3014 and OFF states, it is possible that P2\_MV201 is responsible for **filling a tank**  
3015 **that is not part of this particular subsystem**. Once again, a thorough investigation  
3016 is necessary to confirm this hypothesis.

3017 **6.3.1.3 Invariant Inference and Analysis**

3018 Through the process of *invariant analysis*, we aim to discover new information  
3019 about the system and determine whether the conjectures made  
3020 in the previous steps are supported by the data obtained from the Daikon  
3021 analysis.

3022 **General Invariants** We will begin this phase by analyzing the general  
3023 invariants (see Section 4.2.5.1.). Listing 6.4 presents a selection of these  
3024 invariants:

```
3025 1 P2_P206 == P2_P204 == P2_P202 == P2_P201 == P1_P102 == 1.0
  2 P2_P205 == P2_P203
  3 max_P1_LIT101 == 816.0
  4 min_P1_LIT101 == 489.0
  5 max_P2_AIT201 == 257.0
  6 min_P2_AIT201 == 252.0
  7 max_P2_AIT202 == 9.0
  8 min_P2_AIT202 == 8.0
  9 max_P2_AIT203 == 343.0
 10 min_P2_AIT203 == 320.0
```

*Listing 6.4: General Invariants for PLC1-2*

3035 The invariant mentioned on line 2 is particularly significant: it states  
3036 that P2\_P203 and P2\_P205 always have the same values. While this information  
3037 was somewhat apparent in the previous steps, it becomes more

3038 apparent and evident in this analysis. This observation leads us to speculate  
 3039 that the two pumps, P2\_P203 and P2\_P205, **are related to each other** in  
 3040 some way. The other invariants provided in this section further reinforce  
 3041 the hypotheses about the spare actuators.

3042 **Analysis on Single Actuator States** We proceed with the examination  
 3043 of the invariants derived from the *first of the two semi-automatic analysis*  
 3044 discussed in Section 4.2.5.2. Specifically, we will focus on the analysis con-  
 3045 cerning the states of individual actuators in relation to a specific measure-  
 3046 ment, which in our case pertains to the tank represented by P1\_LIT101. For  
 3047 illustrative purposes, let's consider states 1 and 2 (OFF e ON respectively)  
 3048 of valve P1\_MV101 as an example (we will disregard state 0 as it is consid-  
 3049 ered transient). The conditional invariants pertinent to this scenario can  
 3050 be found in Listing 6.5.

```

3051   1 =====
3052   2 P1_MV101 == 1.0 && P1_LIT101 < max_P1_LIT101 - 16 &&
3053     ↪ P1_LIT101 > min_P1_LIT101 + 15
3054   3 =====
3055   4 ...
3056   5 P2_P205 == P2_P203 == P2_MV201 == P1_P101 == 2.0
3057   6 P1_FIT101 == 0.0
3058   7 slope_P1_LIT101 == -1.0
3059   8 P2_FIT201 > P1_FIT101
3060   9 P2_FIT201 > P1_MV101
3061  10 P2_FIT201 > P1_P101
3062  11 ...
3063  12
3064  13 =====
3065  14 P1_MV101 == 2.0 && P1_LIT101 < max_P1_LIT101 - 16 &&
3066     ↪ P1_LIT101 > min_P1_LIT101 + 15
3067  15 =====
3068  16 slope_P1_LIT101 == P1_P102
3069  17 P1_FIT101 > P1_MV101
3070  18 ...
3071  19 P1_MV101 >= P1_P101
3072  20 P1_MV101 >= P2_MV201

```

```
3073 21 P1_MV101 >= P2_P203  
3074 22 P2_P203 >= P1_P101  
3075 23 . . .
```

*Listing 6.5: Conditional Invariants for states 1 and 2 of P1\_MV101*

3076 To prevent transient periods caused by water flow stabilization when  
3077 the actuators change state, a condition is imposed on the level of P1\_LIT101.  
3078 However, this condition may result in an incomplete understanding of the  
3079 system's behavior. To address this, a manual refinement of the analysis is  
3080 required, utilizing the `runDaikon.py` script as outlined in Section 4.2.5.2.

3081 Based on the analysis, the following observations can be made when  
3082 the valve is in the OFF state:

- 3083 • The slope of P1\_LIT101, denoted as `slope_P1_LIT101`, is negative  
3084 (line 7). This indicates a **downward trend** in the tank level, as we  
3085 have seen in Section 6.3.1.1.
- 3086 • P1\_P101 is in state 2, or ON (line 5).
- 3087 • P1\_FIT101 is zero (line 6).
- 3088 • P2\_FIT201 has a value greater than 2 (line 10).

3089 On the other hand, when the valve is in the ON state:

- 3090 • `slope_P1_LIT101` is positive (line 16). This indicates an **upward trend**  
3091 in the tank level, as we have seen in Section 6.3.1.1.
- 3092 • P1\_FIT101 assumes a value greater than 2. The combination of this  
3093 finding, along with the previous one regarding the same register,  
3094 strengthens the hypothesis that this is indeed a flow sensor.
- 3095 • P1\_P101 can be in either the ON or OFF state, as we have seen in  
3096 Section 6.3.2.2.

3097 When conducting a manual analysis using the `runDaikon.py` script on  
 3098 tank levels that fall outside the range defined by the previous condition, it  
 3099 does not yield useful slope information. This situation can occur because,  
 3100 despite the noise attenuation applied to the tank level sensor data, if there  
 3101 is even a single cycle in the system where the calculated slope deviates  
 3102 from the expected outcome, it can adversely affect the entire Daikon anal-  
 3103 ysis. Figure 6.4 shows this behavior.

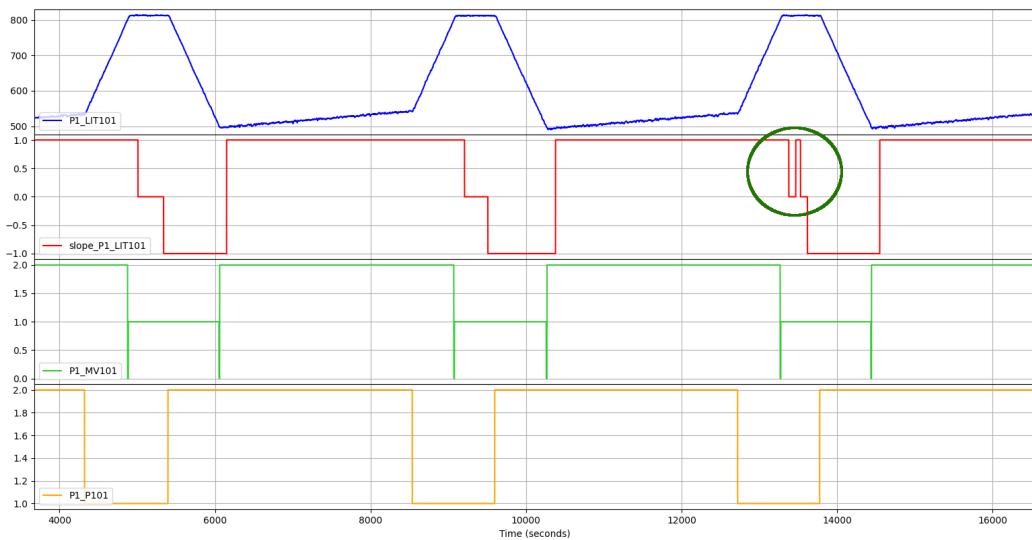


Figure 6.4: Slope calculation anomaly (in the circle)

3104 **Analysis of the Current System Configuration** We conclude our analy-  
 3105 sis of invariants by considering the *second semi-automatic analysis* outlined  
 3106 in Section 4.2.5.2, which focuses on the effective states of the system. Due  
 3107 to the comprehensive nature of this analysis, we will not provide a de-  
 3108 tailed report of the outputs to maintain brevity. However, this analy-  
 3109 sis confirms the findings observed in the analysis of individual actuator  
 3110 states. Additionally, one crucial piece of information becomes apparent  
 3111 for future steps: the changing state of the actuators controlled by PLC2 **do**  
 3112 **not impact the behavior of the tank** controlled by PLC1. Indeed, it is suf-  
 3113 ficient to examine the invariant pertaining to the slope of the tank to verify  
 3114 this observation.

**3115 6.3.1.4 Business Process Mining and Analysis**

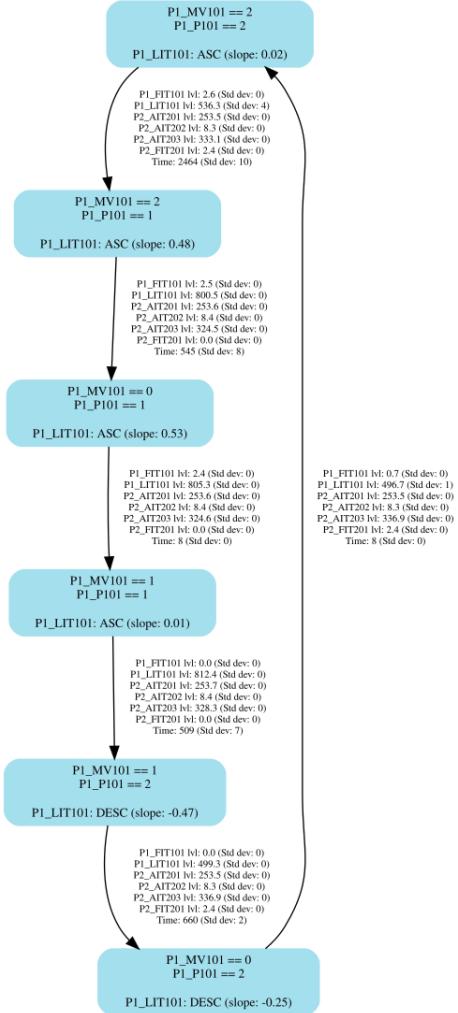
3116 As explained in Section 4.2.6.1, the *process mining phase* applied to the  
3117 physical system provides us with an immediate understanding of the sys-  
3118 tem cycle and the chronological sequence of states. It enables us to de-  
3119 termine the duration of time the system remains in a particular state and  
3120 at what relative setpoint the state transition occurs concerning the refer-  
3121 ence measure. Furthermore, we can analyze the trend of this measure  
3122 within each state. Additionally, we can examine the relative setpoints of  
3123 other measurements to identify any connections between changes in sys-  
3124 tem state and these values.

3125 Given that we have already identified the likely measurement repre-  
3126 senting the tank and the corresponding actuators that control its behavior,  
3127 an activity diagram can be generated as depicted in Figure 6.5.

3128 The activity diagram allows for easy interpretation of the tank level  
3129 trend and slope within different states. The states where the valve has a  
3130 value of 0 can be disregarded due to their short duration. Similar to the  
3131 graphical analysis, there is an observed change in slope during the increas-  
3132 ing trend of the tank level between the states  $[P1\_MV101 == 2, P1\_P101 == 2]$  and  $[P1\_MV101 == 2, P1\_P101 == 1]$ , where the slope changes from  
3133 0.02 to 0.48.  
3134

3135 Additionally, the timing analysis on the edges reveals that the tank  
3136 takes longer to fill than to empty, and the system remains in the  $[P1\_MV101 == 1, P1\_P101 == 1]$  state for approximately 8 minutes (509 seconds).

3138 Regarding the state  $[P1\_MV101 == 1, P1\_P101 == 1]$ , there appears  
3139 to be a discrepancy between the trend of the tank level as reported in the  
3140 activity diagram and the conjectures made in the previous phases of the  
3141 analysis. The activity diagram correctly depicts an increasing trend in the  
3142 tank level between the end of the state  $[P1\_MV101 == 2, P1\_P101 == 1]$  and the end of the state  $[P1\_MV101 == 1, P1\_P101 == 1]$ . However, the  
3143 discrepancy arises due to the fact that the interruption of flow during the  
3144 valve state change from state ON to state OFF is not immediate, mainly  
3145



*Figure 6.5: Activity diagram for PLC1-2*

due to the presence of the transient state 0. Additionally, water continues to flow within the piping towards the tank for a short duration even after the valve is closed. After this period, which usually lasts a few seconds, the tank level stabilizes, and there is no further inflow or outflow of water.

By adjusting the tolerance parameter  $-t$  in the `processMining.py` script, it is possible to obtain accurate data regarding the behavior of the state corresponding to the *plateau* observed in the graphical analysis.

The data presented on the arcs in the activity diagram represents the

3154 measurement values relative to the time of system state changes, specifically  
3155 the relative setpoints. These values are calculated based on the average data collected in each cycle. By analyzing this data, we can observe  
3156 the tank level values at which the system undergoes configuration changes  
3157 and how the trend of the tank level changes accordingly. Specifically, we  
3158 can see that the trend transitions from ascending to stable at a tank level  
3159 value of 800, from stable to descending at approximately 812, and from  
3160 descending back to ascending at around 499. The change in the speed of  
3161 tank filling occurs at approximately 535.

3163 Furthermore, the data provides additional support for the hypothesis  
3164 that the measurements associated with registers P2\_AIT20x are not influ-  
3165 enced by the tank's trends.

3166 **6.3.1.5 Properties**

3167 From the conjectures derived from the four phases of the analysis we  
3168 will derive the properties of the subsystem we are studying, which will be  
3169 placed within a **summary table**. This table contains an integer identifying  
3170 the property, the statement of the property itself, and from which of the  
3171 four phases it was derived.

#	Statement	Derived from
P1	The registers P1_LIT101, P1_FIT101, P2_AIT201, P2_AIT202, P2_AIT203, and P2_FIT201 are considered measurements.	Preliminary Analysis Graphical Analysis
P2	The registers P1_MV101, P1_P101, P2_MV201, P2_P203, and P2_P205 are considered actuators.	Preliminary Analysis Graphical Analysis
P3	The actuators that contain the substring "MVxxx" are considered to be three-state actuators. For simplicity, we refer to them as valves.	Preliminary Analysis
P4	The state 0 of these valves represents a transient state that occurs during the transition between state 1 (OFF) and state 2 (ON).	Preliminary Analysis Graphical Analysis

P5	The actuators that contain the substring "Pxxx" are considered to be binary actuators. For simplicity, we refer to them as pumps.	Preliminary Analysis
P6	The registers P1_P102, P2_P201, P2_P202, P2_P204, and P2_P206 are considered spare actuators. They do <i>not</i> influence the trend of any measurements and they are considered in the OFF state.	Preliminary Analysis Graphical Analysis Invariant Analysis
P7	P1_LIT101 represents the level sensor of the tank controlled by PLC1.	Preliminary Analysis Graphical Analysis
P8	P1_MV101 and P1_P101 are the actuators responsible for the level behavior of the water contained in the tank.	Graphical Analysis Invariant Analysis Business Process
P9	P1_MV101 is responsible for filling the tank. P1_P101 is responsible for emptying the tank.	Graphical Analysis Invariant Analysis
P10	Valve are responsible for water inflow. Pumps responsible for water outflow.	Preliminary Analysis Graphical Analysis Invariant Analysis
P11	The rate of tank level growth is slow when both P1_MV101 and P1_P101 are in the ON state. The growth speed increases when P1_P101 transitions to the OFF state.	Graphical Analysis Business Process
P12	The tank level decreases when P1_MV101 is in the OFF state and P1_P101 is in the ON state.	Graphical Analysis Invariant Analysis
P13	The tank level remains (relatively) stable when both P1_MV101 and P1_P101 are in the OFF state.	Graphical Analysis Business Process
P14	The trend of the tank level transitions from ascending to stable when the level reaches approximately 800. It shifts from stable to descending when the level averages around 812. It changes from descending back to ascending when the level reaches about 500. The speed of tank filling increases noticeably at around 535.	Business Process
P15	Absolute setpoints are 800, 812, 500 and 535.	Business Process
P16	P1_FIT101 serves as a flow sensor to measure the inflow of water into the tank.	Graphical Analysis Invariant Analysis Business Process

P17	None of the actuators connected to PLC2 have an impact on the level of the tank controlled by PLC1.	Graphical Analysis Invariant Analysis Business Process
P18	None of the measurements connected to PLC2 represent a tank.	Preliminary Analysis Graphical Analysis
P19	P2_AIT201 does not exhibit a cyclic trend.	Graphical Analysis
P20	Both P2_AIT201 and P2_AIT202 serve as sensors for measuring certain properties of the water.	Preliminary Analysis Graphical Analysis
P21	The behavior and trend of P2_AIT202 and P2_AIT203 are directly associated with the operation of the pump.	Graphical Analysis Invariant Analysis

*Table 6.1: Properties of the PLC1-2 subsystem*

### 3172 6.3.2 Reverse Engineering of PLC2 and PLC3

3173 Continuing our analysis of the iTrust SWaT system, our current focus  
 3174 will be on the registers of PLC3 and any potential relationships they may  
 3175 have with the registers of PLC2.

#### 3176 6.3.2.1 Pre-processing - Preliminary Analysis

3177 **Measurements and Actuators Recognition** Listing 6.6 shows the out-  
 3178 comes obtained from automatic recognition of likely measurements and  
 3179 actuators. After previously identifying the measurements and actuators  
 3180 of PLC2 in Section 6.3.1.1, the listing exclusively showcases the registers  
 3181 associated with PLC3.

```
3182 1 Actuators:  

  3183 2 ...  

  3184 3 P3_MV301 [0.0, 1.0, 2.0]  

  3185 4 P3_MV302 [0.0, 1.0, 2.0]  

  3186 5 P3_MV303 [0.0, 1.0, 2.0]  

  3187 6 P3_MV304 [0.0, 1.0, 2.0]  

  3188 7 P3_P302 [1.0, 2.0]  

  3189 8  

  3190 9 Sensors:  

  3191 10 ...
```

```

3192 11 P3_DPIT301 {'max_lvl': 20.4, 'min_lvl': 0.0}
3193 12 P3_FIT301 {'max_lvl': 2.4, 'min_lvl': 0.0}
3194 13 P3_LIT301 {'max_lvl': 1014.5, 'min_lvl': 786.5}
3195 14
3196 15 Hardcoded setpoints or spare actuators:
3197 16 ...
3198 17 P3_P301 [1.0]

```

*Listing 6.6: Preliminary analysis outcomes for sensors and actuators of PLC2-3*

From the provided listing, it is evident that the **likely measurements** related to PLC3 are P3\_DPIT301, P3\_FIT301, and P3\_LIT301. Drawing an analogy with the derived properties from Table 6.1, P3\_LIT301 can be associated with a **tank level sensor**, while P3\_FIT301 may be linked to a flow or pressure sensor. However, the specific role of P3\_DPIT301 cannot be speculated upon at this time.

Regarding the **likely actuators**, they include P3\_MV301, P3\_MV302, P3\_MV303, P3\_MV304, and P3\_P302. By drawing parallels with the previous analysis outcomes, it can be inferred that registers P3\_MV30x represent valves, while P3\_P302 corresponds to a pump.

Lastly, there is a **spare actuator** identified as P3\_P301. Similar to the previous analysis, this is an inactive pump, indicated by the constant value of 1 in this register, signifying that it is in the OFF state.

**Actuator State Durations** Let's proceed with the analysis of the actuator states' duration, as displayed in Listing 6.7. In this analysis, our focus will not be on examining the correspondence between values and the actual actuator states, as in Section 6.3.1.1. Instead, we will explore whether these actuators exhibit any distinct patterns or behaviors based on their duration.

```

3218 1 Actuator state durations:
3219 2 ...
3220 3 P3_MV301 == 1.0
3221 4 2527 4154 4154 4154 4094

```

```

3222   5
3223   6 P3_MV301 == 2.0
3224   7 36 35 36 35 34
3225   8
3226   9 P3_MV302 == 1.0
3227  10 662 138 654 138 656 139 658 137 656 137
3228  11
3229  12 P3_MV302 == 2.0
3230  13 62 1783 1596 1787 1591 1791 1576 1803 1540 1782
3231  14
3232  15 P3_MV303 == 1.0
3233  16 2526 4089 4088 4089 4028
3234  17
3235  18 P3_MV303 == 2.0
3236  19 97 96 97 96 96
3237  20
3238  21 P3_MV304 == 1.0
3239  22 689 1832 2206 1838 2203 1840 2191 1852 2152 1831
3240  23
3241  24 P3_MV304 == 2.0
3242  25 43 87 42 89 43 88 43 88 43 88
3243  26
3244  27 P3_P302 == 1.0
3245  28 637 115 632 115 632 114 634 114 632 115
3246  29
3247  30 P3_P302 == 2.0
3248  31 60 1821 1632 1825 1629 1829 1615 1841 1578 1820

```

*Listing 6.7: Time duration of the states of actuators of PLC3*

3249 A notable behavior is observed in the P3\_MV30x valves. P3\_MV301, P3\_MV303,  
 3250 and P3\_MV304 have a relatively **short duration in the ON state**, ranging  
 3251 from around 30 seconds to a minute and a half. In contrast, P3\_MV302 re-  
 3252 mains in the ON state for a longer duration but exhibits approximately  
 3253 twice as many cycles as the other actuators (10 cycles compared to 5 cy-  
 3254 cles). A similar characteristic is also observed in the OFF state of these  
 3255 actuators.

3256 This behavior displayed by the actuators warrants further investiga-  
 3257 tion in subsequent steps. However, based on the short duration of the ON

3258 state for P3\_MV301, P3\_MV303, and P3\_MV304, **it appears unlikely that they**  
 3259 **have a significant impact on the tank level.** On the other hand, the influ-  
 3260 ence of P3\_MV302 cannot be ruled out and requires additional examination.

3261 We can observed that P3\_P302, the pump in PLC3, exhibits a similar be-  
 3262 havior to P3\_MV302, with a cycle number of 10. Furthermore, the durations  
 3263 of the ON and OFF states for both actuators appear to be overlapping. This  
 3264 suggests a **potential relationship between the two actuators.** Addition-  
 3265 ally, considering that P3\_P302 is the only pump in PLC3, it is reasonable to  
 3266 speculate that it may have an influence on the tank level. Further inves-  
 3267 tigation is necessary to validate this speculation and explore the precise  
 3268 nature of the relationship between P3\_P302 and P3\_MV302.

3269 **Actuator State Changes** Based on the analysis of the probable measure-  
 3270 ments, we have identified the likely tank, represented by register P3\_LIT301.  
 3271 In the previous analysis of the PLC1-2 subsystem in Section 6.3.1, the role  
 3272 of valve P2\_MV201 remained unresolved. We speculated that this actuator  
 3273 might be responsible for the incoming water flow to an element outside  
 3274 the analyzed subsystem. To investigate this further, we can examine the  
 3275 relationship between P2\_MV201 and the tank within this subsystem. By  
 3276 extracting information from the corresponding setpoints, we can gather  
 3277 insights to verify if our speculation holds true.  
 3278 Listing 6.8 displays the setpoints associated with the state change of P2\_MV201  
 3279 in relation to the level of tank P3\_LIT301.

3280	1	Actuator state changes:		
3281	2	...		
3282	3	P2_MV201	prev_P2_MV201	P3_LIT301
3283	4	0	2	1000.2240
3284	5	0	1	799.1140
3285	6	0	2	1001.5060
3286	7	0	1	799.1942
3287	8	0	2	1001.5460
3288	9	0	1	799.1140
3289	10	...		

*Listing 6.8: P1\_P101 state changes in relation to P1\_LIT101*

3290     The setpoints provided in the listing support our conjecture. The maximum  
3291     relative setpoint of 1000 and the minimum relative setpoint very  
3292     close to 800 indicate a correlation that appears intentional. Based on this  
3293     information, we can speculate that P2\_MV201 is indeed the **valve responsible**  
3294     **for the tank** associated with P3\_LIT301.

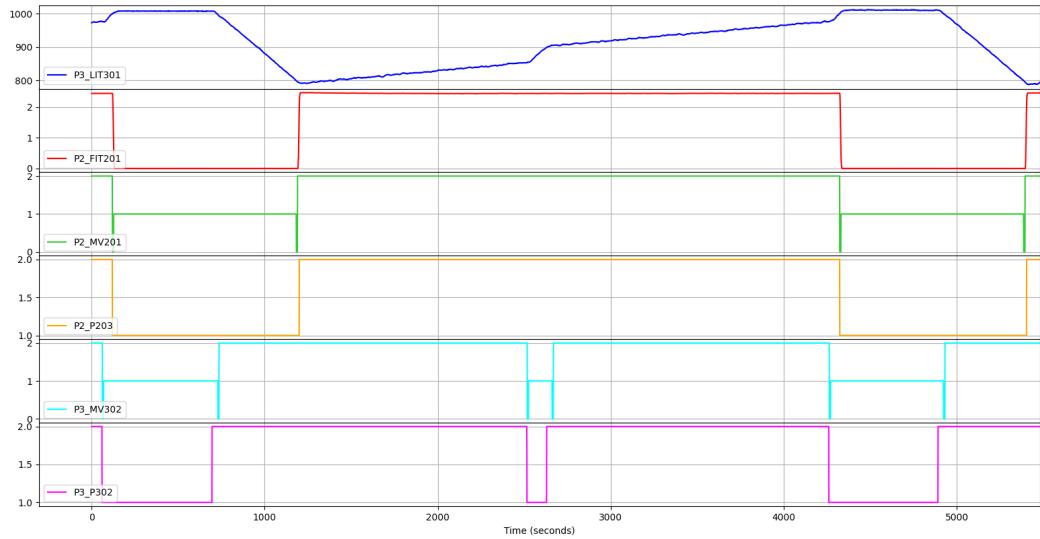
3295     Regarding further information obtained from this step, there is limited  
3296     insight available. While P3\_P302 appears to be the pump responsible for  
3297     emptying the tank associated with P3\_LIT301, the analysis of the actuator  
3298     lifetime indicates twice as many values compared to other actuators.  
3299     The pump exhibits setpoint values of approximately 850 and 970 for the  
3300     transition from ON to OFF, and 900 and 1000 for the transition from OFF  
3301     to ON. On the other hand, P3\_MV302 shows numerous state changes and  
3302     shares setpoints with values close to 850, 970, 1000, and 900 for the same  
3303     transitions. These values align perfectly with those of the P3\_P302 pump,  
3304     suggesting a potential relationship between the two actuators.

3305     Obtaining information about the remaining registers is challenging at  
3306     this stage. Further analysis steps are required to gather additional insights  
3307     and uncover more information about the system.

### 3308     6.3.2.2 Graphs and Statistical Analysis

3309     Graphical analysis can provide valuable insights and help validate the  
3310     conjectures made during the preliminary analysis, as well as uncover connec-  
3311     tions between registers that were not identified in the previous step.  
3312     To begin, we will test the hypothesis that valve P2\_MV201 is responsible for  
3313     filling the tank associated with P3\_LIT101. Figure 6.6 displays these regis-  
3314     ters, along with other registers whose behavior and relationships with the  
3315     tank level we will explore in an attempt to gain a deeper understanding.

3316     The figure shows the particular behavior of P3\_LIT103, with two slope  
3317     changes during the tank filling period. These slope changes correspond  
3318     approximately to the relative setpoints found for P3\_MV302 and P3\_P302.  
3319     However, we will analyze this aspect later. What we can see in relation



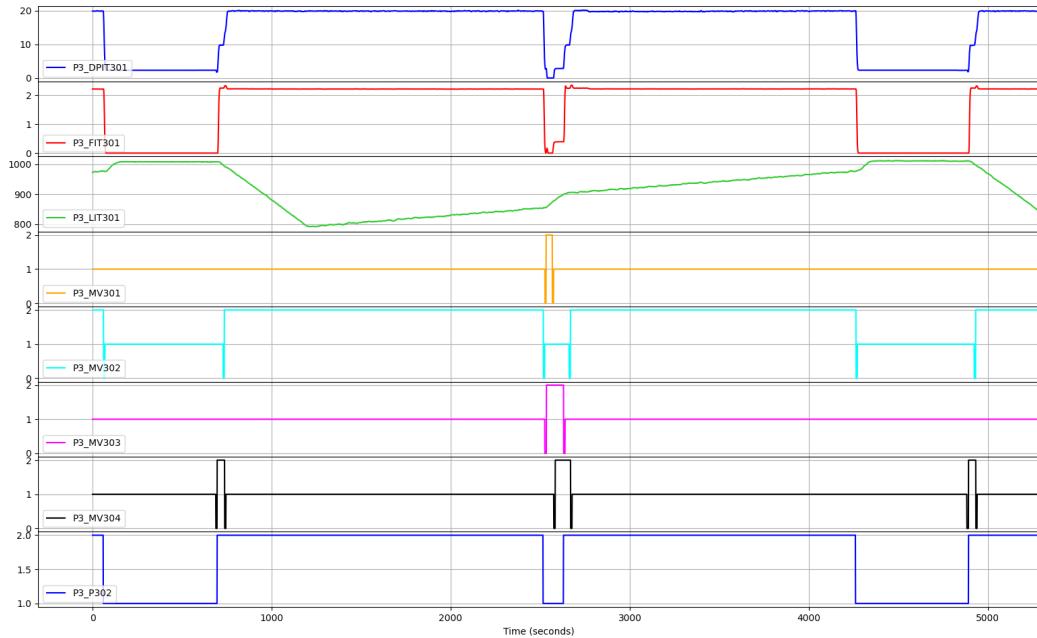
*Figure 6.6: Verifying the conjecture about valve P2\_MV201*

3320 to the initial conjecture about the role of P2\_MV201 is that the period dur-  
 3321 ing which the valve remains in the ON state corresponds exactly to the  
 3322 increasing trend of P3\_LIT301. The conjecture thus finds further support.  
 3323 We also note how P2\_FIT201 is related to the trend of P2\_MV201 and the  
 3324 increasing trend of P3\_LIT301.

3325 Now let's examine the relationships between the tank represented by  
 3326 P3\_LIT301 and the other actuators of PLC3 through Figure 6.7.

3327 From the charts, it is evident that the trends of P3\_DPIT301 and P3\_FIT301  
 3328 exhibit similarities. Furthermore, their overall pattern closely follows that  
 3329 of the valve P3\_MV302. Based on these observations, we can speculate that  
 3330 there is a relationship between these registers or that they serve similar  
 3331 functions, possibly as **pressure or flow sensors**.

3332 Regarding pump P3\_P302, we observe that its OFF state coincides with  
 3333 the increasing slope of P3\_LIT301 during its upward trend and the entire  
 3334 phase when the level remains relatively stable. Conversely, its ON state  
 3335 corresponds to the gradual increase and decrease of the water level in the  
 3336 tank. This observation provides further evidence to support the hypothe-  
 3337 sis that P3\_P302 is responsible for emptying the tank.



*Figure 6.7: PLC3 registers*

3338        Valve P3\_MV302 exhibits a similar pattern to pump P3\_P302. Building  
 3339        upon our previous findings, we can speculate that, similar to P2\_MV201 in  
 3340        the previous analyzed subsystem, P3\_MV302 is responsible for controlling  
 3341        the incoming flow to another element outside the analyzed subsystem.

3342        Let us now analyze the potential roles of valves P3\_MV301, P3\_MV303,  
 3343        and P3\_MV304 in relation to the indicated tank level. It seems unlikely  
 3344        that P3\_MV304 has any direct impact on the tank level. The valve is acti-  
 3345        vated twice within one system cycle, but there are no noticeable changes  
 3346        in P3\_LIT301 during its first opening. This suggests that the second open-  
 3347        ing is also insignificant in terms of tank level. However, it is worth noting  
 3348        the slight peaks in P3\_FIT301 that occur shortly after the valve's opening.

3349        Regarding the remaining valves, P3\_MV301 and P3\_MV303, their impact  
 3350        on the tank level is still unclear, particularly during the increased slope of  
 3351        P3\_LIT101 around the second 2500. Figure 6.8 provides us with a compre-  
 3352        hensive overview of the situation.

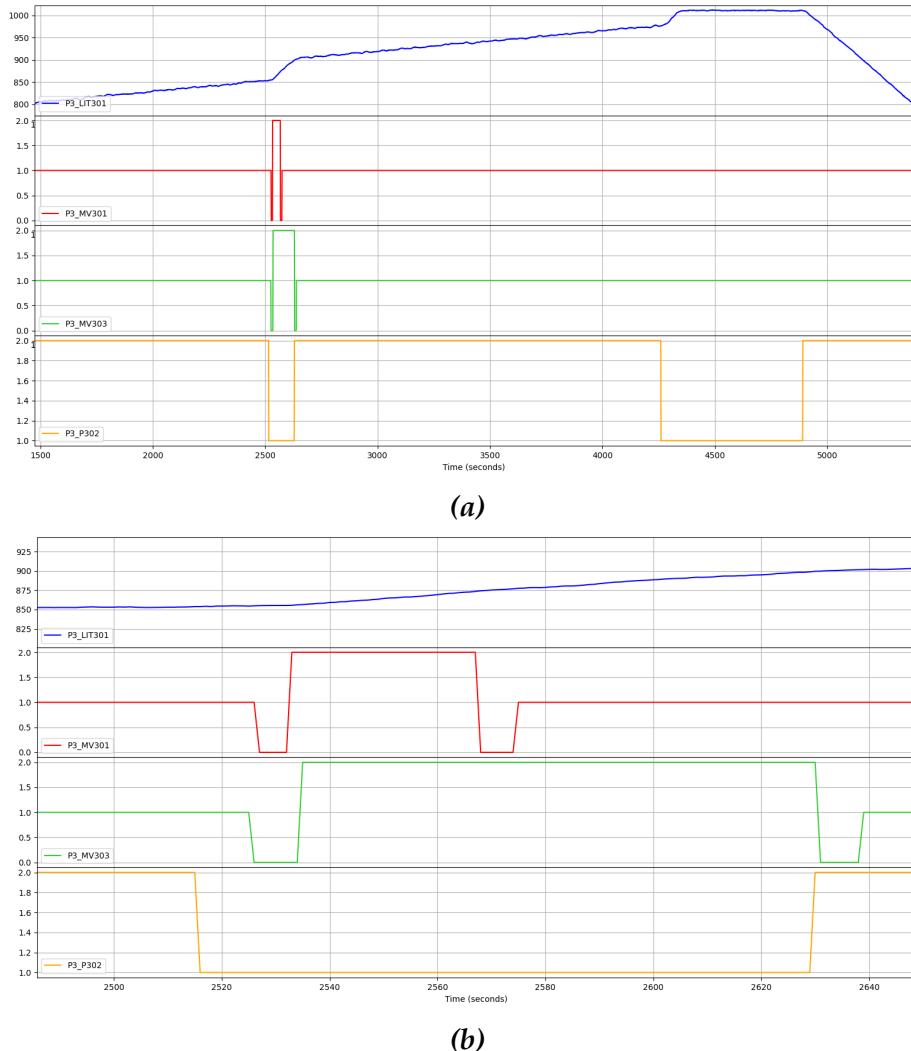


Figure 6.8: P3\_MV301 and P3\_MV303 analysis

3353     The analysis of the valves P3\_MV301 and P3\_MV303 indeed presents some  
 3354     challenges. However, upon closer examination of Figure 6.8b, we observe  
 3355     that when these valves are in the ON state, the slope of P3\_LIT301 remains  
 3356     relatively constant. This is unexpected, as one would anticipate a steeper  
 3357     slope due to the inflow of liquid. Additionally, in Figure 6.8a, we can ob-  
 3358     serve that when these valves are in the OFF state, the slope of P3\_LIT301  
 3359     from the second 4300 remains similar to the section we are currently ana-  
 3360     lyzing. Based on these observations, we speculate that these valves either

3361 **do not affect the water level of the tank or have a minimal, undetectable  
3362 impact.**

3363     Indeed, Figure 6.7 provides additional insights into the relationship  
3364     between the valves P3\_MV301, P3\_MV303, and P3\_MV304 and the sensors  
3365     P3\_DPIT301 and P3\_FIT301. The graph shows that the values of P3\_DPIT301  
3366     and P3\_FIT301 undergo noticeable changes during the activation of these  
3367     valves, suggesting a potential connection between them. This observation  
3368     supports the hypothesis that the valves and sensors are linked in some  
3369     way,

### 3370 6.3.2.3 Invariant Inference and Analysis

3371 **General Invariants** The analysis of general invariants does not yield any  
3372 significant insights. However, it confirms the maximum and minimum  
3373 values of the measurements seen in Section 6.3.2.1 and identifies the pres-  
3374 ence of the spare actuators P3\_P301.

3375 **Analysis on Single Actuator States** The analysis of single actuator states  
3376 reveals additional information. The resulting invariants provide further  
3377 support for the conjecture regarding the roles of P2\_MV201 and P3\_P302 in  
3378 regulating the water level in the tank, with the former responsible for fill-  
3379 ing and the latter for emptying. Listing 6.9 presents the specific invariants  
3380 involved in this analysis.

```
3381 1 =====
3382 2 P2_MV201 == 1.0 && P3_LIT301 < max_P3_LIT301 - 20 &&
3383 3   ↪ P3_LIT301 > min_P3_LIT301 + 24
3384 4 =====
3385 5 P3_P301 == P3_MV303 == P3_MV301 == P2_P205 == P2_P203 ==
3386 6   ↪ P2_MV201 == 1.0
3387 7 P3_P302 == 2.0
3388 8 slope_P3_LIT301 == -1.0
3389 9 P3_FIT301 > P2_MV201
3390 10 P3_DPIT301 > P3_FIT301 > P3_P302
3391 11 ...
```

```

3392 10 =====
3393 11 P2_MV201 == 2.0 && P3_LIT301 < max_P3_LIT301 - 20 &&
3394 12   ↪ P3_LIT301 > min_P3_LIT301 + 24
3395 13 =====
3396 14 P2_P205 == P2_P203 == P2_MV201 == 2.0
3397 15 slope_P3_LIT301 == P2_P201
3398 16 P2_FIT201 > P2_MV201
3399 17 P2_FIT201 > P2_P201
3400 18 P2_FIT201 > P3_FIT301
3401 19 ...
3402 20 =====
3403 21 P3_P302 == 1.0 && P3_LIT301 < max_P3_LIT301 - 20 &&
3404 22   ↪ P3_LIT301 > min_P3_LIT301 + 24
3405 23 =====
3406 24 P2_P205 == P2_P203 == P2_MV201 == 2.0
3407 25 slope_P3_LIT301 == P3_P302 == P2_P201
3408 26 P2_FIT201 > P2_MV201
3409 27 P2_FIT201 > P3_FIT301
3410 28 ...
3411 29 =====
3412 30 P3_P302 == 2.0 && P3_LIT301 < max_P3_LIT301 - 20 &&
3413 31   ↪ P3_LIT301 > min_P3_LIT301 + 24
3414 32 =====
3415 33 P2_MV201 one of { 1.0, 2.0 }
3416 34 slope_P3_LIT301 one of { -1.0, 1.0 }
3417 35 P3_DPIT301 > P3_P302 > slope_P3_LIT301
3418 36 ...

```

***Listing 6.9:*** Conditional Invariants for P2\_MV201 and P3\_P302

Moreover, from the analysis of the invariants it becomes apparent that when P3\_P302 is in the ON state and P2\_MV201 is in the OFF state, both P3\_FIT301 and P3\_DPIT301 take values greater than 2 (as derived from lines 5, 7, 8 and 32 of the listing).

Regarding the valves P3\_MV301 and P3\_MV303, Daikon's analysis does not provide specific information about their behavior during changes in slope when the water level rises. Therefore, further analysis is required to understand their role in the system.

3427     However, one observation can be made regarding P3\_MV304. Daikon's  
3428    analysis reveals two different slopes (increasing and decreasing) when this  
3429    valve is in the ON state, which aligns with the observations made in the  
3430    Graphical Analysis in Section 6.3.2.2. This finding strengthens the conjecture  
3431    that P3\_MV304 does not play a significant role in the tank cycle represented  
3432    by P3\_LIT301.

3433   **Analysis of the Current System Configuration** To simplify the analysis  
3434    and facilitate the interpretation of outcomes, two separate groups of actuators  
3435    were analyzed. The first group consists of P2\_MV201 and P3\_P302,  
3436    which are conjectured to regulate the level of the tank. The second group  
3437    includes P3\_MV301, P3\_MV303, and P3\_MV304, for which it is speculated that  
3438    they do not play a role in regulating the tank level. This grouping allows  
3439    for a clearer examination of the states of the system and provides an opportunity  
3440    to gain a better understanding of the behavior of these actuators.  
3441    By analyzing these two groups separately, it becomes easier to draw conclusions  
3442    and make comparisons between the different sets of actuators.

3443     The analysis of the first group of actuators, specifically P2\_MV201 and  
3444    P3\_P302, further supports the conjectures made regarding their behavior  
3445    in relation to the trend of the tank level. It was necessary to refine the  
3446    analysis manually using the runDaikon.py script to obtain more detailed  
3447    insights, particularly for the state [P2\_MV201 == 2, P3\_P302 == 2].

3448     Regarding the second group of actuators, the analysis of their states  
3449    only reinforces the hypothesis that their activation does not have a significant  
3450    impact on the trend of tank level represented by P3\_LIT301.

3451     Unfortunately, no further useful information can be derived from this  
3452    phase of the analysis. To address the remaining questions and clarify any  
3453    outstanding issues, we will proceed to the next and final phase of the sub-  
3454    system analysis.

<sup>3455</sup> **6.3.2.4 Business Process Mining and Analysis**

<sup>3456</sup> bla

<sup>3457</sup> **6.3.2.5 Properties**

<sup>3458</sup> **6.3.3 Reverse Engineering of PLC3 and PLC4**

<sup>3459</sup> **6.3.3.1 Pre-processing - Preliminary Analysis**

<sup>3460</sup> **6.3.3.2 Graphs and Statistical Analysis**

<sup>3461</sup> **6.3.3.3 Invariant Inference and Analysis**

<sup>3462</sup> **6.3.3.4 Business Process Mining and Analysis**

<sup>3463</sup> **6.3.3.5 Properties**

## Conclusion and Future Work

3464 **Discussion** Upon completing our black-box analysis of the iTrust SWaT  
3465 system, although partial, we can confidently state that we have achieved a  
3466 significant level of process comprehension for the examined system. Fur-  
3467 thermore, we have successfully derived a relatively accurate model of its  
3468 behavior. Specifically:

- 3469 1. we successfully identified the registers associated with the physical  
3470 process measures and their respective operating ranges (setpoints);
- 3471 2. among these measurements, we were able to distinguish the sensors  
3472 responsible for monitoring the water level in the tanks;
- 3473 3. we identified the registers associated with the actuators, such as pumps  
3474 and valves, and determined their ON and OFF states;
- 3475 4. we were able to determine whether an actuator is responsible for  
3476 controlling incoming or outgoing flows within the system;
- 3477 5. we successfully identified the possible states that the system may  
3478 assume and determined the chronological sequence in which these  
3479 states occur;
- 3480 6. we recognized the specific events that trigger changes in the system  
3481 state through the actuators, such as reaching specific water level val-  
3482 ues in the tanks, and vice versa;

- 3483     7. we elucidated the relationships between registers belonging to dif-  
3484       ferent PLCs, highlighting their interconnections and dependencies;
- 3485     8. we partially reconstructed the communication network between the  
3486       PLCs, outlining the connections and interactions between them.

3487       For some registers it was not possible to derive their specific role (e.g.,  
3488       register P2\_AIT203 monitored by PLC2). In such cases, we were restricted  
3489       to making conjectures or relying on generic properties inferred from other  
3490       aspects. Additionally, based on the available data, we were unable to es-  
3491       tablish the presence of piping or other elements such as filter membranes  
3492       or tanks containing chemicals.

3493       Items 4, 5, and 8 in the aforementioned list highlight the expanded  
3494       scope of information that can be obtained from the proposed framework  
3495       compared to the work of Ceccato et al. Furthermore, the increased quan-  
3496       tity and depth of information acquired at each step of the methodology  
3497       reinforce our belief that the primary objective of this thesis has been fully  
3498       accomplished.

3499       **Future work** Undoubtedly, the proposed framework can be further im-  
3500       proved. Throughout the development of this thesis, ideas for potential  
3501       enhancements have emerged, representing a natural evolution of the cur-  
3502       rent proposal. In the following, we will illustrate these areas for future  
3503       work:

- 3504       • **Scanning of the system and data gathering:** In our methodology,  
3505       we excluded a particular step since we utilized the datasets pro-  
3506       vided by iTrust, encompassing both physical process and network  
3507       traffic data (for more details, please refer to Section 5.2). However,  
3508       in cases where ready-made datasets are unavailable, this step may  
3509       be necessary. It is important to note that the main limitation of this  
3510       step, compared to the Ceccato et al. tool, is its exclusive focus on the

3511       recognition of PLCs based on the Modbus protocol. As we have ob-  
3512       served, there exist numerous industrial protocols, and Modbus, al-  
3513       though widely used, is just one among many. Therefore, it becomes  
3514       imperative to ensure that this phase can effectively handle the recog-  
3515       nition of PLCs using multiple protocols. To address this challenge,  
3516       one can leverage the protocols directive within the [NETWORK] sec-  
3517       tion of the general configuration file, *config.ini*.

3518       Another potential solution we propose is the implementation of an  
3519       automatic recognition method utilizing Machine Learning and Arti-  
3520       ficial Intelligence techniques. This approach has been explored and  
3521       tested on the datasets related to the physical process of the SWaT  
3522       system provided by iTrust. Unfortunately, these datasets proved to  
3523       be inconsistent in many instances, which hindered the adequacy of  
3524       training. As a result, this avenue was abandoned. However, if an  
3525       attacker has the capability to conduct repeated series of scans on the  
3526       target system, they could utilize this data collection to train a model  
3527       for recognition purposes;

- 3528       • **Automatic recognition of measurements and actuators:** this aspect  
3529       was implemented within our framework using a mixed Daikon-Pandas  
3530       technique. While this technique has shown positive results, it can be  
3531       further improved by modifying it to utilize only the Pandas library.  
3532       The revised technique would involve checking the type and num-  
3533       ber of distinct values in each register to accurately identify measure-  
3534       ments and actuators. This modification would offer greater flexibil-  
3535       ity and precision in the recognition process, allowing for the defini-  
3536       tion of specific criteria, such as setting a maximum number of dis-  
3537       tinct values to determine whether a register should be classified as  
3538       an actuator;
- 3539       • **Completion of Business Process Analysis and Network Analysis:**  
3540       in our implementation, these aspects remained incomplete, as stated  
3541       in Section 6.3, due to the unavailability of comprehensive and over-

3542 lapping data in the datasets provided by iTrust for the given years.  
3543 To address this, improvements are required, such as enhancing the  
3544 multiprotocol support of the network data export script (currently  
3545 adapted to CIP only) and suggesting a modular solution. Addition-  
3546 ally, it is necessary to implement an event recognition mechanism to  
3547 identify system state changes within network communications;

- 3548 • **Recognition of actual system states:** in the steps related to Invariant  
3549 Analysis and Business Process, it was necessary to manually specify  
3550 the actuators that directly impact the subsystem under study, omit-  
3551 ting the remaining ones. This approach allowed us to obtain the ac-  
3552 tual states of the subsystem, enabling the derivation of information  
3553 that is more immediate and likely more accurate. Throughout the  
3554 thesis development, various methods were explored, including ap-  
3555 proximating tank level data (similar to the approach used for slope  
3556 calculation in Section 4.2.3.2) and identifying points of slope change.  
3557 However, none of these methods proved effective due to significant  
3558 data perturbations. Consequently, addressing this issue remains an  
3559 ongoing challenge that requires additional time to find a consistent  
3560 and reliable solution.

## List of Figures

2.1	ICS architecture schema . . . . .	8
2.2	PLC architecture . . . . .	11
2.3	PLC communication schema . . . . .	12
2.4	Comparison between ST language and Ladder Logic . . . . .	13
2.5	Example of HMI for a water treatment plant . . . . .	16
2.6	Modbus Request/Response schema . . . . .	18
2.7	Modbus RTU frame and Modbus TCP frame . . . . .	19
2.8	Example of packet sniffing on the Modbus protocol . . . . .	21
2.9	OSI model for EtherNet/IP stack . . . . .	22
3.1	Workflow of Ceccato et al.'s stages and operations with used tools . . . . .	30
3.2	The simplified SWaT system used for running Ceccato et al. methodology . . . . .	31
3.3	Output graphs from graph analysis . . . . .	35
3.4	An example of Disco generated activity diagram for PLC2 .	39
3.5	Execution traces of InputRegisters_IW0 on the three PLCs .	41
3.6	Business process with states and Modbus commands for the three PLCs . . . . .	43
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. . . . .	49

3.8	Behavior of the Graph Analysis on the Ceccato et al.'s tool . . . . .	51
3.9	Histogram plot overshadowing statistical information shown on the terminal window in the background . . . . .	52
3.10	Example of Daikon's output . . . . .	54
4.1	Network Communications between 192.168.1.40 (Source) and 192.168.1.30 (Destination) . . . . .	70
4.2	Network Communications between 192.168.1.40 (Source) and 192.168.1.30 (Destination) in textual mode . . . . .	71
4.3	Slope comparison with granularity 30 (a), 60 (b) and 120 seconds (c) . . . . .	79
4.4	Savitzky-Golay filter (blue line), STL decomposition (green) and RDP algorithm (orange) comparison . . . . .	81
4.5	Slope after the application of the STL decomposition . . . . .	82
4.6	The new slope representation (green line) and the smoothed measurement data obtaind with the STL decomposition (red)	83
4.7	Plotting registers on the same y-axis . . . . .	87
4.8	Example of the new graph analysis . . . . .	88
4.8	Example of the new graph analysis (cont.) . . . . .	89
4.9	Example of transitive closure graphs for invariants in PLC1 of iTrust SWaT system . . . . .	93
4.10	Directory (a) and outcome files (b) for the single actuator states analysis . . . . .	97
4.11	Daikon outcome files for system configuration analysis. Each file represents a single system state . . . . .	97
4.12	Activity diagram for PLC1 of the iTrust SWaT system . . . . .	104
5.1	SWaT architecture . . . . .	109
5.2	Sensors and actuators associated with each PLC . . . . .	110
5.3	SWaT network architecture . . . . .	111
5.4	SWaT stabilization . . . . .	114
6.1	Simplified graph of the iTrust SWaT system network . . . . .	119
6.2	Chart of PLC1-2 registers . . . . .	127

6.3	Chart of PLC1-2 registers without spare actuators (particular)	128
6.4	Slope calculation anomaly (in the circle) . . . . .	132
6.5	Activity diagram for PLC1-2 . . . . .	134
6.6	Verifying the conjecture about valve P2_MV201 . . . . .	142
6.7	PLC3 registers . . . . .	143
6.8	P3_MV301 and P3_MV303 analysis . . . . .	144



## List of Tables

2.1	differences between Information Technology (IT) and Industrial Control Systems (ICSs)	6
2.2	Modbus Function Codes list	20
3.1	Summary table of Ceccato et al. framework limitations	56
5.1	main IP addresses of the six PLCs and SCADA in SWaT	112
5.2	SWaT network traffic data	115
6.1	Properties of the PLC1-2 subsystem	137



## Listings

3.1	Example of registers capture . . . . .	33
3.2	Example of raw network capture . . . . .	34
3.3	Statistical data for PLC1_InputRegisters_IW0 register . . . . .	36
3.4	Generic example of a .spinfo file for customizing rules in Daikon . . . . .	37
3.5	The three sections of Daikon analysis outcomes . . . . .	37
4.1	Novel Framework structure and Python scripts . . . . .	61
4.2	Paths and parameters for the Pre-processing phase in <i>config.ini</i> file . . . . .	73
4.3	config.ini parameters for dataset enriching . . . . .	75
4.4	Example of preliminary system analysis . . . . .	84
4.5	Standard Daikon output for PLC1 of the iTrust SWaT system	91
4.6	Revised Daikon output with transitive closures for PLC1 of the iTrust SWaT system . . . . .	94
4.7	Daikon outcomes for the system configuration MV101 == 2, P101 == 1 on LIT101 . . . . .	98
4.8	Example of the data contained in the produced JSON file . . . . .	102
6.1	Preliminary analysis outcomes for sensors and actuators of PLC1-2 . . . . .	123
6.2	Time duration of the states of actuators P1_MV101 and P1_MV201 of PLC1-2 . . . . .	124
6.3	P1_P101 state changes in relation to P1_LIT101 . . . . .	125
6.4	General Invariants for PLC1-2 . . . . .	129

6.5	Conditional Invariants for states 1 and 2 of P1_MV101 . . . . .	130
6.6	Preliminary analysis outcomes for sensors and actuators of PLC2-3 . . . . .	137
6.7	Time duration of the states of actuators of PLC3 . . . . .	138
6.8	P1_P101 state changes in relation to P1_LIT101 . . . . .	140
6.9	Conditional Invariants for P2_MV201 and P3_P302 . . . . .	145

## References

- [1] NIST. *ICS defintion*. URL: [https://csrc.nist.gov/glossary/term/industrial\\_control\\_system](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](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](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](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](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](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](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](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](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](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](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](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](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](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](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](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](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/](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](https://www.modbus.org/docs/CIP-Modbus-Integration-Hanover-Fair_0408.pdf) (visited on 2023-05-17).