

**UNIVERSITY OF VERONA**

---

Department of COMPUTER SCIENCE

Master's Degree in  
COMPUTER SCIENCE AND ENGINEERING

Master Thesis

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

*Candidate:*

**Marco OLIANI**  
VR457249

*Supervisor:*

**Prof. Massimo MERRO**

*Co-supervisor:*

**Prof. Ruggero LANOTTE**  
University of Insubria

---

Academic Year 2022/2023



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

(Richard Clarke)



## **Abstract**

Bla bla bla



# Contents

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

3.2	Ceccato et al.'s black-box dynamic analysis for water-tank systems . . . . .	26
3.2.1	Testbed . . . . .	27
3.2.2	Scanning of the System and Data Pre-processing . . . . .	31
3.2.3	Graphs and Statistical Analysis . . . . .	33
3.2.4	Invariants Inference and Analysis . . . . .	34
3.2.5	Business Process Mining and Analysis . . . . .	36
3.2.6	Application . . . . .	38
3.2.7	Limitations . . . . .	44
<b>4</b>	<b>A Substantial Improvement to Ceccato et al.'s Framework</b>	<b>55</b>
4.1	The Proposed New Framework . . . . .	56
4.1.1	Framework Structure . . . . .	58
4.1.2	Python Libraries and External Tools . . . . .	60
4.2	Analysis Phases . . . . .	60
4.2.1	Phase 1: Data Pre-processing . . . . .	60
4.2.1.1	Subsystem Selection . . . . .	62
4.2.1.2	Dataset Enrichment . . . . .	64
4.2.1.3	Datasets Merging . . . . .	71
4.2.1.4	Brief Analysis of the Obtained Subsystem . . . . .	72
4.2.2	Phase 2: Graphs and Statistical Analysis . . . . .	76
4.2.3	Phase 3: Invariant Inference and Analysis . . . . .	82
4.2.3.1	Revised Daikon Output . . . . .	82
4.2.3.2	Types of Analysis . . . . .	86
4.2.4	Phase 4: Business Process Analysis . . . . .	92
4.2.5	Phase 5: Network Analysis . . . . .	92
<b>5</b>	<b>Case study: the iTrust SWaT System</b>	<b>93</b>
5.1	Architecture . . . . .	93
5.1.1	Physical Process . . . . .	94
5.1.2	Control and Communication Network . . . . .	96
5.2	Datasets . . . . .	98
5.2.1	Our Case Study: the 2015 Dataset . . . . .	99

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



## Introduction

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

### 1.1 Contribution

Lorem Ipsum is simply dummy text of the printing and typesetting industry. Lorem Ipsum has been the industry's standard dummy text ever since the 1500s, when an unknown printer took a galley of type and scrambled it to make a type specimen book. It has survived not only five centuries, but also the leap into electronic typesetting, remaining essentially unchanged. It was popularised in the 1960s with the release of Letraset sheets containing Lorem Ipsum passages, and more recently with desktop publishing software like Aldus PageMaker including versions of Lorem Ipsum.

### 1.2 Outline

The thesis is structured as follows:

**Chapter 2:** provides background on the topics covered in this thesis: Industrial Control Systems (ICSs), Supervisory Control And Data Acquisition (SCADA), Programmable Logic Controllers (PLCs) and other devices, industrial communication protocols.

**Chapter 3:**

**Chapter 4:**

**Chapter 5:**

**Chapter 6:**

**Chapter 7:**

## Background on Industrial Control Systems

<sup>1</sup> INDUSTRIAL CONTROL SYSTEMS (ICSs) are information systems used to  
<sup>2</sup> control industrial processes such as manufacturing, product handling,  
<sup>3</sup> production, and distribution [1]. ICSs are often found in critical infrastruc-  
<sup>4</sup> ture facilities such as power plants, oil and gas refineries, and chemical  
<sup>5</sup> plant ICSs are different from traditional IT systems in several key ways.

<sup>6</sup> Firstly, ICSs are designed to control physical processes, whereas IT sys-  
<sup>7</sup> tems are designed to process and store data. This means that ICSs have dif-  
<sup>8</sup> ferent requirements for availability, reliability, and performance. Secondly,  
<sup>9</sup> ICSs are typically deployed in environments that are harsh and have lim-  
<sup>10</sup> ited resources, such as extreme temperatures and limited power. Thirdly,  
<sup>11</sup> the protocols hardware and software used in ICSs are often proprietary.

<sup>12</sup> ICSs are becoming increasingly connected to the internet and other net-  
<sup>13</sup> works, which has led to increased concerns about their security. Industrial  
<sup>14</sup> systems were not originally designed with security in mind, and many of  
<sup>15</sup> them have known vulnerabilities that could be exploited by attackers. Ad-  
<sup>16</sup> ditionally, the use of legacy systems and equipment can make it difficult  
<sup>17</sup> to implement security measures. As a result, ICSs are increasingly seen  
<sup>18</sup> as a potential target for cyber attacks, which could have serious conse-  
<sup>19</sup> quences for the safe and reliable operation of critical infrastructure: some  
<sup>20</sup> notorious examples of cyber attacks are (*i*) the **STUXnet** worm [2], which

<sup>21</sup> purpose was to sabotage the nuclear centrifuges of the enrichment plant  
<sup>22</sup> at the Natanz nuclear facility in Iran; (ii) **Industroyer** [3], also referred  
<sup>23</sup> as *Crashoverride*, responsible for the attack on the Ukrainian power grid  
<sup>24</sup> on December 17, 2016; (iii) the attack on February, 2021 to a water treat-  
<sup>25</sup> ment plant in Oldsmar, Florida [4], where the level of sodium hydroxide  
<sup>26</sup> was intentionally increased to a level approximately 100 times higher than  
<sup>27</sup> normal.

<sup>28</sup>

<sup>29</sup> The increasing connectivity of ICSs and the associated security risks have  
<sup>30</sup> led to a growing interest in the field of ICS security. Researchers and prac-  
<sup>31</sup> titioners are working to develop new security technologies, standards, and  
<sup>32</sup> best practices to protect ICSs from cyber attacks. This includes efforts to  
<sup>33</sup> improve the security of ICS networks and devices, as well as the develop-  
<sup>34</sup> ment of new monitoring and detection techniques to identify and respond  
<sup>35</sup> to cyber attacks.

<sup>36</sup>

<sup>37</sup> Table 2.1 summarizes the differences between traditional IT and ICSs [5]:

	<b>Traditional IT</b>	<b>ICSs</b>
<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>38</sup> 2.1 Industrial Control Systems Architecture

<sup>39</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>48</sup>

<sup>49</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>53</sup> • Level 0 (**Processes, or Field I/O Devices**): contains **field devices**.
- <sup>54</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>59</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>63</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.

## 2.1 Industrial Control Systems Architecture

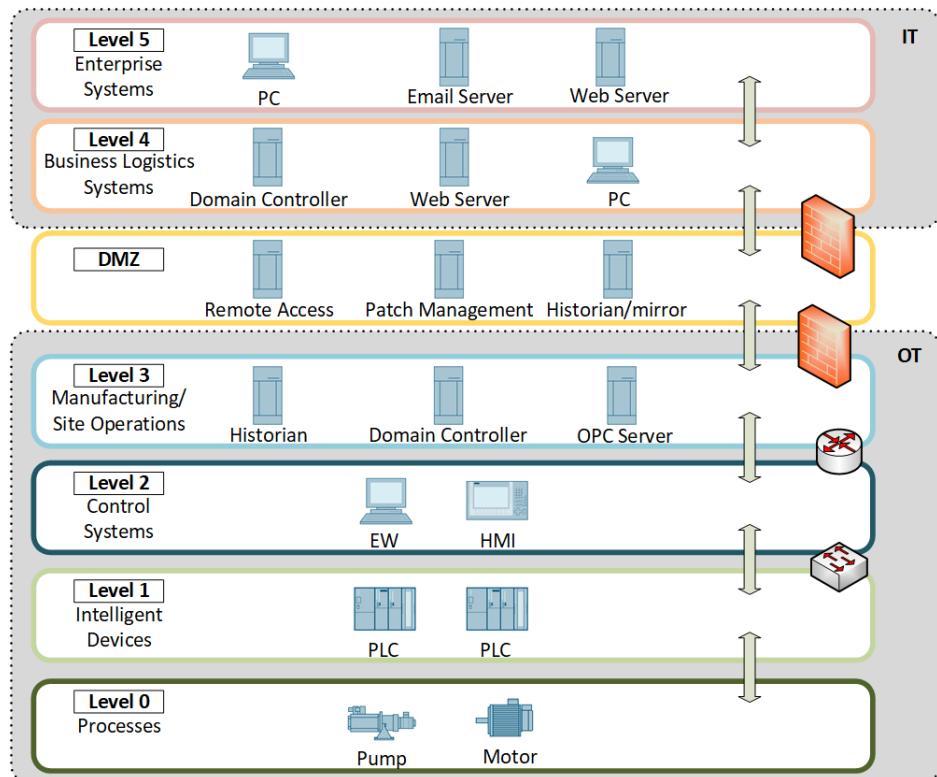


Figure 2.1: ICS architecture schema

- **Industrial Demilitarized Zone (DMZ):** intermediate level that connects the *Operational Technology* (OT) part (levels 0-3) with the *Information Technology* (IT) part of the system (levels 4 and 5). Communication takes place indirectly through services such as *proxy servers* and *remote access servers*, which act as intermediaries between the two environments.
- Level 4 (**Business Logistics Systems**, or **Business Planning/Logistics**): collect and aggregates data from the Manufacturing/Site Operations level overseeing the IT-related activities to generate **reporting** to the Enterprise System layer. At this layer we can find application and e-mail servers, and *Enterprise Resource Planning* (ERP) systems.
- Level 5 (**Enterprise Systems**): represents the enterprise network, used for the business-to-business activities and for business-to-client pur-

pose services. At Enterprise Systems level are typical IT services such as mail servers, web servers and all the systems used to manage the ongoing process.

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

## 2.2 Operational Technology Networks

*Operational Technology* primarily encompasses the **tangible aspects** of Industrial Control Systems and directly interfaces with the physical processes of the monitored systems. Its main purpose is to **manage and control the procedures** involved in creating and correcting physical value in various equipment.

This section will focus on the key aspects and components of Operational Technology network, with specific reference to the first four levels of the Purdue model previously seen.

### 2.2.1 Field I/O Devices Layer

This level concerns all aspects related to the physical environment and the physical elements that are part of it, which have the ability to actively influence the environment.

These physical elements are represented by **Field Devices**, i.e., **sensors** and **actuators** used to collect data from the process and control it: sensors are the elements responsible for reading specific values related to the physical environment (e.g., the level of a liquid), while actuators change its behavior and characteristics (e.g., opening or closing a valve to make the liquid flow).

Examples of field devices include temperature sensors, pressure sensors, valves and pumps.

### <sup>109</sup> 2.2.2 Controller Network Layer

<sup>110</sup>      *Controller Network* layer includes devices that handle data from and to  
<sup>111</sup> the *Field I/O Devices* layer. This kind of device is capable of gathering data  
<sup>112</sup> from sensors, updating its internal state, and activating actuators (for ex-  
<sup>113</sup> ample opening or close a pump that controls the level of a tank), making  
<sup>114</sup> decisions based on a customized program, known as its control logic.  
<sup>115</sup> Commonly found within this layer are *Programmable Logic Controllers* (PLCs)  
<sup>116</sup> and *Remote Terminal Units* (RTUs): in the upcoming sections, we will ex-  
<sup>117</sup> amine these elements in detail.

#### <sup>118</sup> 2.2.2.1 Programmable Logic Controllers

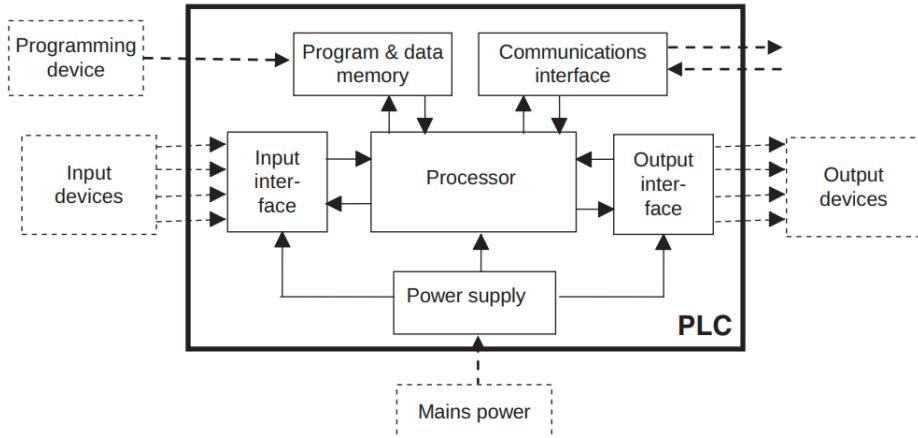
<sup>119</sup>      A *Programmable Logic Controller* (PLC) is a **small and specialized in-**  
<sup>120</sup> **dustrial computer** having the capability of controlling complex industrial  
<sup>121</sup> and manufacturing processes [7].

<sup>122</sup>      Compared to relay systems and personal computers, PLCs are opti-  
<sup>123</sup> mized for control tasks and industrial environments: they are rugged and  
<sup>124</sup> designed to withdraw harsh conditions such as dust, vibrations, humid-  
<sup>125</sup> ity and temperature: they have more reliability than personal computers,  
<sup>126</sup> which are more prone to crash, and they are more compact a require less  
<sup>127</sup> maintenance than a relay system.

<sup>128</sup> Furthermore, I/O interfaces are already on the controller, so PLCs are eas-  
<sup>129</sup> ier to expand with additional I/O modules (if in a rack format) to manage  
<sup>130</sup> more inputs and ouputs, without reconfiguring hardware as in relay sys-  
<sup>131</sup> tems when a reconfiguration occurs.

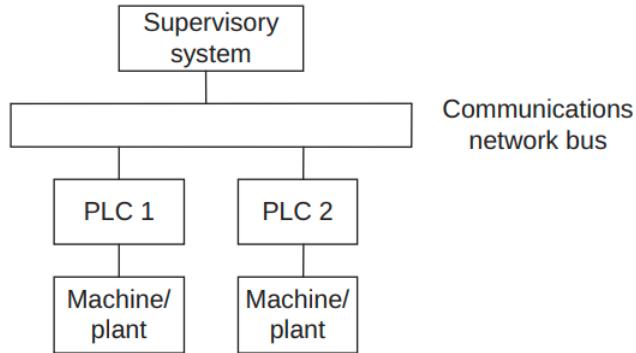
<sup>132</sup>      PLCs are more *user-friendly*: they are not intended (only) for computer  
<sup>133</sup> programmers, but designed for engineers with a limited knowledge in  
<sup>134</sup> programming languages: control program can be entered with a simple  
<sup>135</sup> and intuitive language based on logic and switching operations instead of  
<sup>136</sup> a general-purpose programming language (*i.e.* C, C++, ...).

<sup>137</sup> **PLC Architecture** The basic hardware architecture of a PLC consists of  
<sup>138</sup> these elements [8]:



*Figure 2.2: PLC architecture*

- <sup>139</sup> • **Processor unit (CPU):** contains the microprocessor. This unit interprets the input signals from I/O modules, executes the control program stored in the Memory Unit and sends the output signals to the I/O Modules. The processor unit also sends data to the Communication interface, for the communication with additional devices.
- <sup>140</sup> • **Power supply unit:** converts AC voltage to low DC voltage.
- <sup>141</sup> • **Programming device:** is used to store the required program into the memory unit.
- <sup>142</sup> • **Memory Unit:** consists in RAM memory and ROM memory. RAM memory is used for storing data from inputs, ROM memory for storing operating system, firmware and user program to be executed by the CPU.
- <sup>143</sup> • **I/O modules:** provide interface between sensors and final control elements (actuators).
- <sup>144</sup> • **Communications interface:** used to send and receive data on a network from/to other PLCs.



*Figure 2.3: PLC communication schema*

<sub>155</sub> **PLC Programming** Two different programs are executed in a PLC: the <sub>156</sub> **operating system** and the **user program**.

<sub>157</sub> The operating system tasks include executing the user program, managing memory areas and the *process image table* (memory registers where <sub>159</sub> inputs from sensors and outputs for actuators are stored).

<sub>160</sub> The user program needs to be uploaded on the PLC via the programming device and runs on the process image table in *scan cycles*: each scan <sub>162</sub> is made up of three phases [9]:

- <sub>163</sub> 1. reading inputs from the process images table
- <sub>164</sub> 2. execution of the control code and computing the physical process evolution
- <sub>166</sub> 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 <sub>168</sub> refreshed by the CPU

<sub>169</sub> Standard PLCs **programming languages** are basically of two types: <sub>170</sub> **textuals** and **graphicals**. Textual languages include languages such as <sub>171</sub> *Instruction List* (IL) and *Structured Text* (ST), while *Ladder Diagrams* (LD), <sub>172</sub> *Function Block Diagram* (FBD) and *Sequential Function Chart* (SFC) belong <sub>173</sub> to the graphical languages.

174 Graphical languages are more simple and immediate comparing to the  
 175 textual ones and are preferred by programmers because of their features  
 176 and simplicity, in particular the **Ladder Logic programming** (see Figure  
 177 2.4 for a comparison).

```

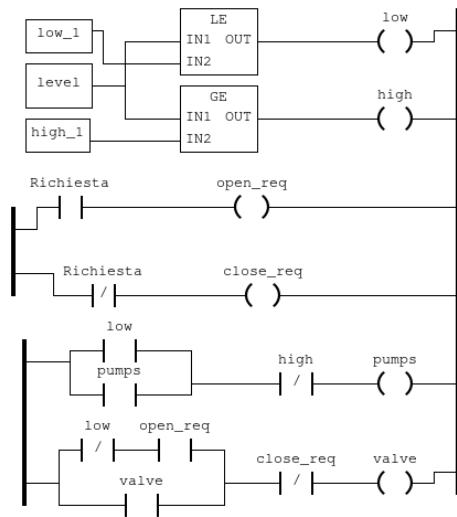
PROGRAM PLC1
VAR
  level AT %IW0 : INT;
  Richiesta AT %QX0..2 : BOOL;
  request AT %IW1 : INT;
  pumps AT %QX0..0 : BOOL;
  valve AT %QX0..1 : BOOL;
  low AT %MW0..0 : BOOL;
  high AT %MW0..1 : BOOL;
  open_req AT %MW0..3 : BOOL;
  close_req AT %MW0..4 : BOOL;
  low_1 AT %MW0 : INT := 40;
  high_1 AT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

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

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

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

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

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

#### 199     2.2.2.2 Remote Terminal Units

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

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

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

### <sup>218</sup> 2.2.3 Area Control Layer

<sup>219</sup>      The Area Control layer encompasses hardware and software systems  
<sup>220</sup> useful for supervising, monitoring and controlling the physical process,  
<sup>221</sup> driving the behavior of the entire infrastructure. The layer includes sys-  
<sup>222</sup> tems such as *Supervisory Control and Data Acquisition* (SCADA), *Distributed*  
<sup>223</sup> *Control Systems* (DCSs), that perform SCADA functions but are usually de-  
<sup>224</sup> ployed locally, and engineer workstations.

#### <sup>225</sup> 2.2.3.1 Supervisory Control And Data Acquisition

<sup>226</sup>      *Supervisory Control And Data Acquisition (SCADA)* is a system of soft-  
<sup>227</sup> ware and hardware elements that allows industrial organizations to [12]:

- <sup>228</sup>      • Control industrial processes locally or at remote locations;
- <sup>229</sup>      • Monitor, gather, and process real-time data;
- <sup>230</sup>      • Directly interact with devices such as sensors, valves, pumps, mo-  
<sup>231</sup> tors, and more through human-machine interface (HMI) software;
- <sup>232</sup>      • Record and aggregate events to send to historian server.

<sup>233</sup>      The SCADA software processes, distributes, and displays the data, help-  
<sup>234</sup> ing operators and other employees analyze the data and make important  
<sup>235</sup> decisions.

#### <sup>236</sup> 2.2.3.2 Human-Machine Interface

<sup>237</sup>      The *Human-Machine Interface* (HMI) is the hardware and software in-  
<sup>238</sup> terface that operators use to monitor the processes and interact with the  
<sup>239</sup> ICS. A HMI shows the operator and authorized users information about  
<sup>240</sup> system status and history; it also allows them to configure parameters on  
<sup>241</sup> the ICS such as set points and, send commands and make control deci-  
<sup>242</sup> sions [13].

<sup>243</sup>      The HMI can be in the form of a physical panel, with buttons and indicator  
<sup>244</sup> lights, or PC software as shown in Figure 2.5.

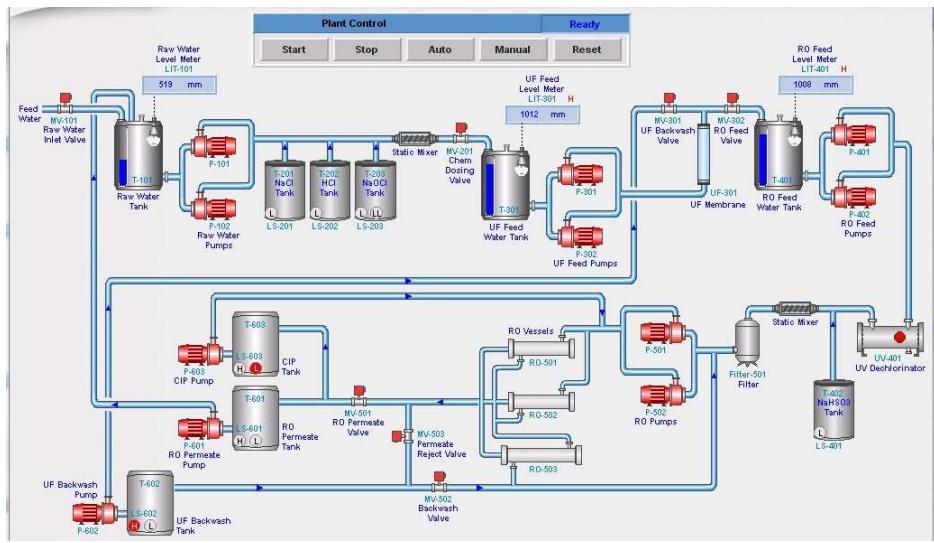


Figure 2.5: Example of HMI for a water treatment plant

#### 2.2.4 Operations/Control Layer

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

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

#### 2.2.5 Demilitarized Zone

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

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

### 265 2.2.6 Industrial Protocols

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

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

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

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

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

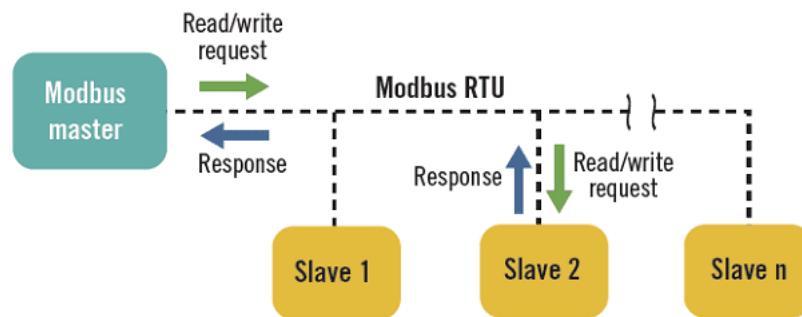
<sup>287</sup> **2.2.6.1 Modbus**

<sup>288</sup> *Modbus* is a serial communication protocol developed by Modicon (now  
<sup>289</sup> Schneider Electric) in 1979 for use with its PLCs [15] and designed ex-  
<sup>290</sup> pressly for industrial use: it facilitates interoperability of different devices  
<sup>291</sup> connected to the same network (sensors, PLCs, HMIs, ...) and it is also  
<sup>292</sup> often used to connect RTUs to SCADA acquisition systems.

<sup>293</sup> Modbus is the most widely used communication protocol among in-  
<sup>294</sup> dustrial systems because it has several advantages:

- <sup>295</sup> • simplicity of implementation and debugging
- <sup>296</sup> • it moves raw bits and words, letting the individual vendor to repre-  
<sup>297</sup> sent the data as it prefers
- <sup>298</sup> • it is, nowadays, an **open** and *royalty-free* protocol: there is no need  
<sup>299</sup> to sustain licensing costs for implementation and use by industrial  
<sup>300</sup> device vendors

<sup>301</sup> Modbus is a **request/response** (or *master/slave*) protocol: this makes it  
<sup>302</sup> independent of the transport layer used.



*Figure 2.6: Modbus Request/Response schema*

<sup>303</sup> In this kind of architecture, a single device (master) can send requests  
<sup>304</sup> to other devices (slaves), either individually or in broadcast: these slave  
<sup>305</sup> devices (usually peripherals such as actuators) will respond to the master

306 by providing data or performing the action requested by the master using  
 307 the Modbus protocol. Slave devices cannot generate requests to the master  
 308 [16].

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

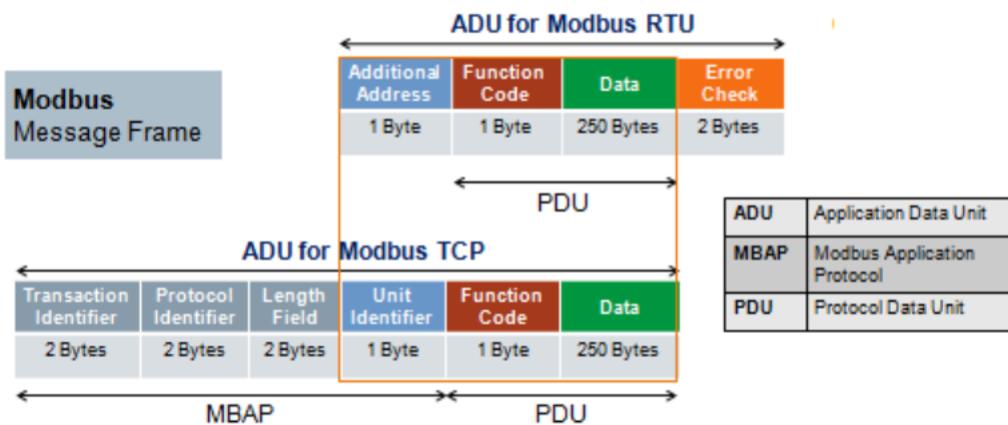


Figure 2.7: Modbus RTU frame and Modbus TCP frame

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

- 316 • *Coil*: binary type, read/write accessible by both masters and slaves
- 317 • *Discrete Input*: binary type, accessible in read-only mode by masters  
 318 and in read/write mode by slaves
- 319 • *Analog Input*: 16 bits in size (word), are accessible in read-only mode  
 320 by masters and in read/write mode by slaves
- 321 • *Holding Register*: 16 bits in size (word), accessible in read/write mode  
 322 by both masters and slaves. Holding Registers are the most com-  
 323 monly used registers for output and as general memory registers.

<sup>324</sup> **Modbus Function Codes** *Modbus Function Codes* are specific codes used  
<sup>325</sup> by the Modbus master within a request frame (see Figure 2.7) to tell the  
<sup>326</sup> Modbus slave device which register type to access and which action to  
<sup>327</sup> perform on it.

<sup>328</sup> Two types of Function Codes exists: for data access and for diagnostic  
<sup>329</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

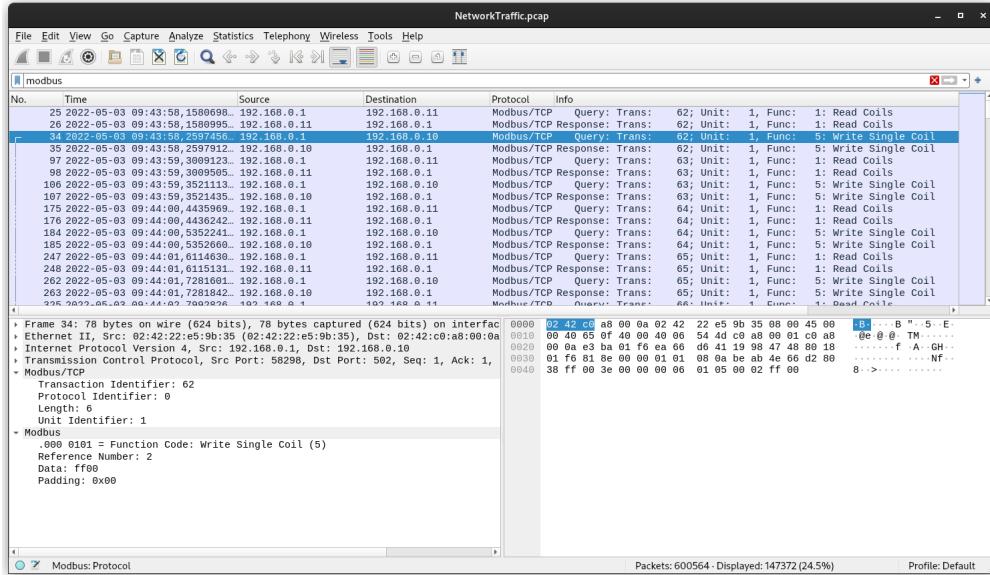
*Table 2.2: Modbus Function Codes list*

<sup>330</sup> **Modbus Security Issues** Despite its simplicity and widespread use, the  
<sup>331</sup> Modbus protocol does not have any security feature, which exposes it to  
<sup>332</sup> vulnerabilities and attacks.

<sup>333</sup> Data in Modbus are transmitted unencrypted (*lack of confidentiality*),  
<sup>334</sup> with no data integrity controls (*lack of integrity*) and authentication checks  
<sup>335</sup> (*lack of authentication*), in addition to the *lack of session*. Hence, the protocol  
<sup>336</sup> is vulnerable to a variety of attacks, such as Denial of Services (DoS), buffer  
<sup>337</sup> overflows and reconnaissance activities.

<sup>338</sup> The easiest attack to bring to the Modbus protocol, however, is **packet**  
<sup>339</sup> **sniffing**: since, as mentioned earlier, network traffic is unencrypted and  
<sup>340</sup> the data transmitted is in cleartext, it is sufficient to use a packet sniffer to  
<sup>341</sup> capture the network traffic, read the packets and thus gather informations

- 342 about the system such as ip addresses, function codes of requests and to  
 343 modify the operation of the devices.



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

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

### 350 2.2.6.2 EtherNet/IP

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

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

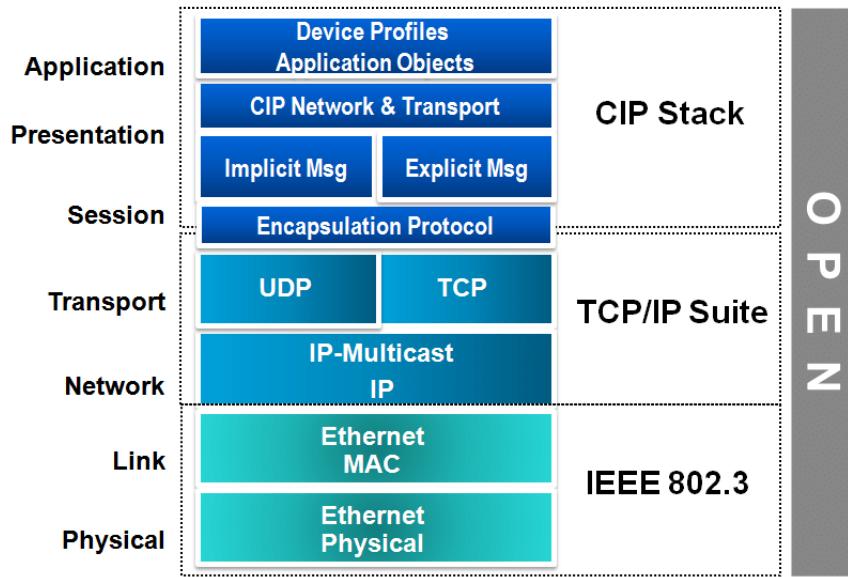


Figure 2.9: OSI model for EtherNet/IP stack

<sup>358</sup> and other industrial protocols for data access and exchange such as *Open*  
<sup>359</sup> *Platform Communication* (OPC).

<sup>360</sup> **Physical and Data Link layer** The use of the IEEE 802.3 standard allows  
<sup>361</sup> EtherNet/IP to flexibly adopt different network topologies (star, linear,  
<sup>362</sup> ring, etc.) over different connections (copper, fibre optic, wireless, etc.), as  
<sup>363</sup> well as the possibility to choose the speed of network devices.  
<sup>364</sup> IEEE 802.3 in addition defines at Data Link layer the *Carrier Sense Multiple*  
<sup>365</sup> *Access - Collision Detection* (CSMA/CD) protocol, which controls access to  
<sup>366</sup> the communication channel and prevents collisions.

<sup>367</sup> **Transport layer** At the transport level, EtherNet/IP encapsulates mes-  
<sup>368</sup> sages from the CIP stack into an Ethernet message, so that messages can  
<sup>369</sup> be transmitted from one node to another on the network using the TCP/IP  
<sup>370</sup> protocol. EtherNet/IP uses two forms of messaging, as defined by CIP  
<sup>371</sup> standard [18][20]:

- <sup>372</sup> • **unconnected messaging:** used during the connection establishment  
<sup>373</sup> phase and for infrequent, low priority, explicit messages. Uncon-

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

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

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

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

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

### 394        2.2.6.3 Common Industrial Protocol (CIP)

395        The *Common Industrial Protocol* (CIP) is an open industrial automation  
396        protocol supported by ODVA. It is a **media independent** (or *transport in-*  
397        *dependent*) protocol using a *producer-consumer* communication model and  
398        providing a **unified architecture** throughout the manufacturing enterprise  
399        [21][22].

400        CIP has been adapted in different types of network:

- 401     • **EtherNet/IP**, adaptation to *Transmission Control Protocol* (TCP) tech-  
402         nologies
- 403     • **ControlNet**, adaptation to *Concurrent Time Domain Multiple Access*  
404         (CTDMA) technologies
- 405     • **DeviceNet**, adaptation to *Controller Area Network* (CAN) technolo-  
406         gies
- 407     • **CompoNet**, adaptation to *Time Division Multiple Access* (TDMA) tech-  
408         nologies

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

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

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

## State of the Art

425 IN COVENTIONAL IT SYSTEMS, the objective of an attacker is to comprehend  
426 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).

432 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.

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

439 There is limited literature available concerning the gathering and analysis  
440 of information related to the comprehension and operation of an Industrial  
441 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  
442 will specifically focus on one of the presented papers.

### **3.1 Literature on Process Comprehension**

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

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

*Feng et al.* Feng et al. [29] developed a framework that can generate system *invariant rules* based on machine learning and data mining techniques from ICS operational data log. These invariants are then selected by systems engineers to derive IDS systems from them.

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

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

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

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

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

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

501 Reconnaissance phase is fundamental to process comprehension and  
502 thus to the execution of MitM attacks.

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

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

---

506           posed PLCs and the Modbus protocol network scans an approximate  
507           model of the physical process. This model is obtained by inferring  
508           statistical properties, business process and system invariants from  
509           data logs.

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

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

## **516 3.2 Ceccato et al.'s black-box dynamic analysis 517           for water-tank systems**

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

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

530           To accomplish this, the various types of memory registers are analyzed,  
531           and attempts are made to determine the nature of the data they might  
532           contain.

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

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

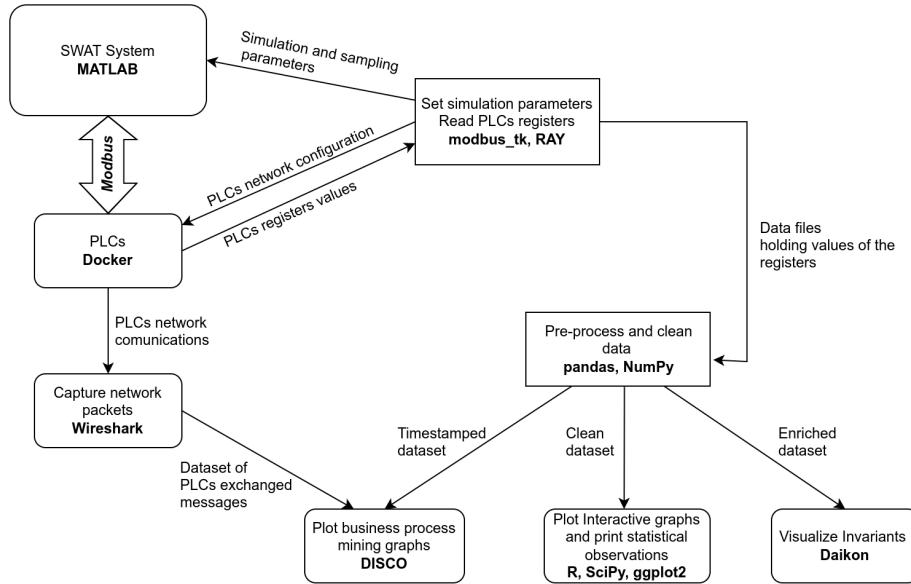
- 538 1. **scanning of the system and data pre-processing:** this phase involves  
539 gathering data to generate data logs for the registers of PLCs and for  
540 Modbus network communications.
- 541 2. **graphs and statistical analysis:** The collected data is utilized to pro-  
542 vide insights into the memory registers associated with the Modbus  
543 protocol by leveraging graphs and statistical data. This analysis ap-  
544 proach offers valuable information about the characteristics and pat-  
545 terns of the memory registers.
- 546 3. **invariants inference and analysis:** generates system invariants, which  
547 are used to identify specific patterns and regularities within the sys-  
548 tem. Additionally, this phase provides users with the capability to  
549 view invariants related to a particular sensor or actuator.
- 550 4. **business process mining and analysis:** Using event logs, this phase  
551 involves reconstructing the business process that depicts how a pro-  
552 cess is executed. This step enables a thorough understanding of the  
553 sequence of events that occur in the system and how they are in-  
554 terrelated, ultimately leading to a comprehensive overview of the  
555 business process.

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

### 561 3.2.1 Testbed

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

## 28 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

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

568 **Stage 1** During the initial stage, a **tank** referred to as T-201 with a capacity  
 569 of 80 gallons is filled with raw water using the P-101 pump. Connected to the T-201 tank, the MV-301 motorized valve flushes out the  
 570 accumulated water from the tank, directing it to the next stage. Initially,  
 571 the water flows from the T-201 tank to the *filtration unit* (which  
 572 is not specifically identified by any sensor), and subsequently to a  
 573 **second tank** denoted as T-202, with a capacity of 20 gallons.  
 574

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

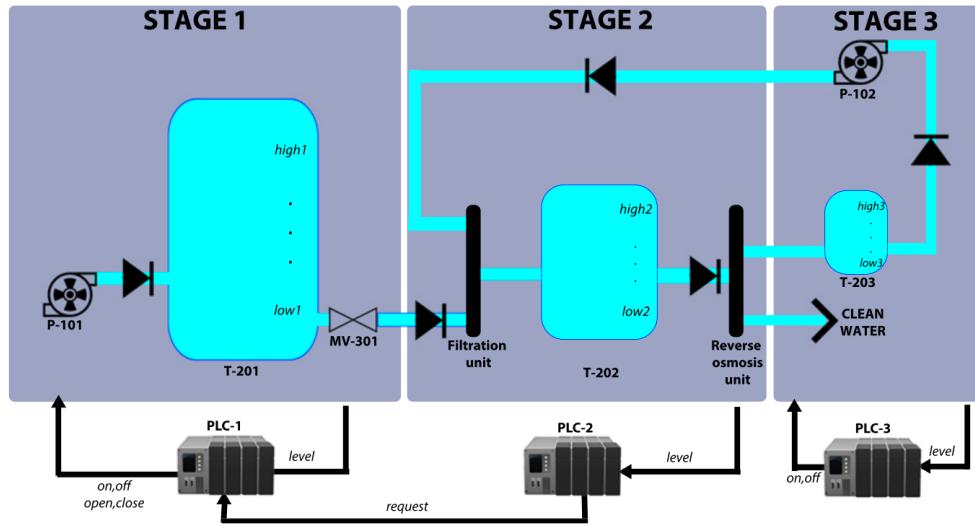


Figure 3.2: The simplified SWaT system used for running Ceccato et al. methodology

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

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

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

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

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

---

- 597           2. if the level of T-201 reaches the *high setpoint high1* (which is also  
598           hardcoded in a specific memory register), then the pump **P-101**  
599           **is closed**;
- 600           3. in cases where the level of T-201 is between the *low setpoint low1*  
601           and the *high setpoint high1*, PLC1 waits for a request from PLC2  
602           to open or close the valve MV-301. If a request to open the valve  
603           MV-301 is received, water will flow from T-201 to T-202. How-  
604           ever, if no request is received, the valve remains closed. In both  
605           situations, the pump P-101 remains closed.

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

- 608           1. when the water level in tank T-202 reaches *low setpoint low2* (also  
609           hardcoded in the memory registers), PLC2 sends a request to  
610           PLC1 through a Modbus channel to **open valve MV-301**. This  
611           request is made in order to allow the water to flow from tank T-  
612           201 to tank T-202. The transmission channel between the PLCs  
613           is established by copying a boolean value from a memory reg-  
614           ister of PLC2 to a corresponding register of PLC1.
- 615           2. when the water level in tank T-202 reaches the *high setpoint high2*  
616           value (also hardcoded in the memory registers), PLC2 sends a  
617           **close request to PLC1 for valve MV-301**. This request prompts  
618           PLC1 to close the valve, stopping the flow of water from tank  
619           T-201 to tank T-202.
- 620           3. In cases where the water level in tank T-202 is between the low  
621           and high setpoints, the valve MV-301 remains in its current state  
622           (open or closed) while the tank is either filling or emptying.

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

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

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

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

### 632 3.2.2 Scanning of the System and Data Pre-processing

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

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

```
652     "127.0.0.1/8502/2022-05-03 12_10_00.591": {
653         "DiscreteInputRegisters": {"%IX0.0": "0"},
654         "InputRegisters": {"%IW0": "53"},
655         "HoldingOutputRegisters": {"%QW0": "0"},
656         "MemoryRegisters": {"%MW0": "40", "%MW1": "80"},
```

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

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

*Listing 3.1: Example of registers capture*

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

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

```
666     Time,Source,Destination,Protocol,Length,Function Code,  
667     ↳ Destination Port,Source Port,Data,Frame length on the  
668     ↳ wire,Bit Value,Request Frame,Reference Number,Info  
669 2022-05-03 11:43:58.158,IP_PLC1,IP_PLC2,Modbus/TCP,76,Read  
670     ↳ Coils,46106,502,,76,TRUE,25,,,"Response: Trans: 62;  
671     ↳ Unit: 1, Func: 1: Read Coils"
```

*Listing 3.2: Example of raw network capture*

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

683

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

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

688 **3.2.3 Graphs and Statistical Analysis**

689 The paper mentions the presence of a *mild graph analysis*, performed  
 690 using the framework **R** [44] for statistical analysis at the time of data gath-  
 691 ering to find any uncovered patterns, trends and identify measurements  
 692 and/or actuator commands through the analysis of registers holding mu-  
 693 table values.

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

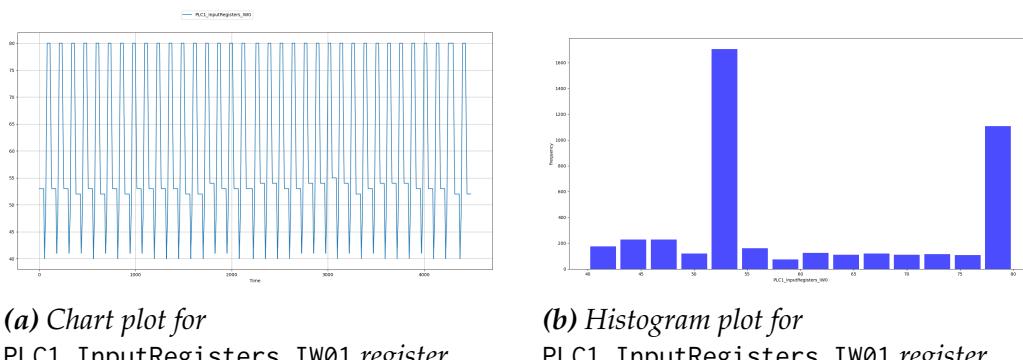


Figure 3.3: Output graphs from graph analysis

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

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

---

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

- 706 • the mean, median, standard deviation, maximum value and mini-  
707 umum value
- 708 • two tests for the statistical distribution: *Chi-squared* test for unifor-  
709 mity and *Shapiro-Wilk* test for normality, as shown in Listing 3.3:

```
710     Chi-squared test for uniformity
711     Distance      pvalue      Uniform?
712     12488.340    0.00000000    NO
713
714     Shapiro-Wilk test for normality
715     Test statistic   pvalue      Normal?
716     0.844        0.00000000    NO
717
718     Stats of PLC1_InputRegisters_IW0
719     Sample mean = 60.8881; Stddev = 13.0164; max = 80; min =
720     ↪ 40 for 4488 values
```

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

### 721 3.2.4 Invariants Inference and Analysis

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

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

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

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

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

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

```
747 =====
748 aprogram.point:::POINT
749 PLC2_MemoryRegisters_MW1 == PLC3_MemoryRegisters_MW1
750 PLC1_MemoryRegisters_MW0 == 40.0
751 PLC1_MemoryRegisters_MW1 == 80.0
752 PLC1_Coils_QX00 one of { 0.0, 1.0 }
753 [...]
754 =====
755 aprogram.point:::POINT; condition="PLC1_InputRegisters_IW0
756   ↪ > 60"
757 PLC1_InputRegisters_IW0 > PLC1_MemoryRegisters_MW0
758 PLC1_InputRegisters_IW0 > PLC1_Min_safety
759 PLC1_MemoryRegisters_MW0 < prev_PLC1_InputRegisters_IW0
760 [...]
761 =====
762 aprogram.point:::POINT; condition="not(
763   ↪ PLC1_InputRegisters_IW0 > 60)"
764 PLC1_InputRegisters_IW0 < PLC1_MemoryRegisters_MW1
765 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

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

---

```
766     PLC1_MemoryRegisters_MW1 > prev_PLC1_InputRegisters_IW0  
767     [...]
```

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

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

770  
771 Some examples of invariants derived from the enriched dataset may be:

- 772     • measurements bounded by some setpoint;
- 773     • actuators state changes occurred in the proximity of setpoints or,  
774       vice versa, proximity of setpoints upon the occurrence of an actuator  
775       state change;
- 776     • state invariants of some actuators correspond to a specific trend in  
777       the evolution of the measurements (ascending, descending, or sta-  
778       ble) or, vice versa, the measurements trend corresponds to a specific  
779       state invariant of some actuators.

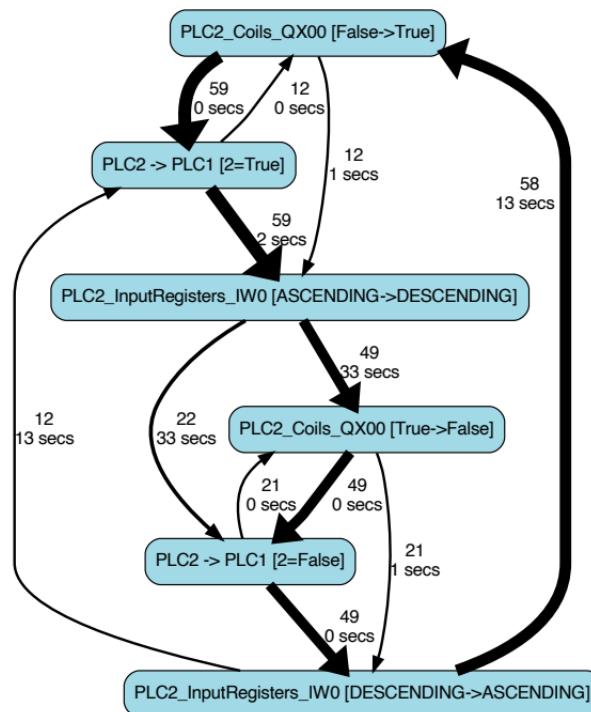
#### 780 3.2.5 Business Process Mining and Analysis

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

785  
786 In the considered system, process mining begins by analyzing the event  
787 logs derived from scanning the memory registers of the PLCs and moni-  
788 toring the network communications associated with the Modbus protocol,  
789 as detailed in Subsection 3.2.2. These event logs serve as the *execution trace*  
790 of the system. A Java program is utilized to extract and consolidate infor-  
791 mation from these event logs, resulting in a CSV format file that captures  
792 the relevant data.

793 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.

808 **3.2.6 Application**

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

817 **Data Collection and Pre-processing** According to what is described in  
818 the paper, the data gathering process lasted six hours, with a granular-  
819 ity of one data point per second (a full system cycle takes approximately  
820 30 minutes). Each datapoint consists of 168 attributes (55 registers plus  
821 a special register concerning the tank slope of each PLC) after the en-  
822 richment. In addition, IP addresses are automatically replaced by an ab-  
823 stract name identified by the prefix PLC followed by a progressive integer  
824 (PLC1, PLC2, PLC3), in order to make reading easier.

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

827 **Property 1:** PLC1\_MemoryRegisters\_MW0, PLC1\_MemoryRegisters\_MW1,  
828 PLC2\_MemoryRegisters\_MW0, PLC2\_MemoryRegisters\_MW1,  
829 PLC3\_MemoryRegisters\_MW0 and PLC3\_MemoryRegisters\_MW1  
830 registers contain constant integer values (40, 80, 10, 20, 0, 10 respec-  
831 tively)<sup>3</sup>. The authors speculate that they may be (relative) hardcoded  
832 **setpoints**.

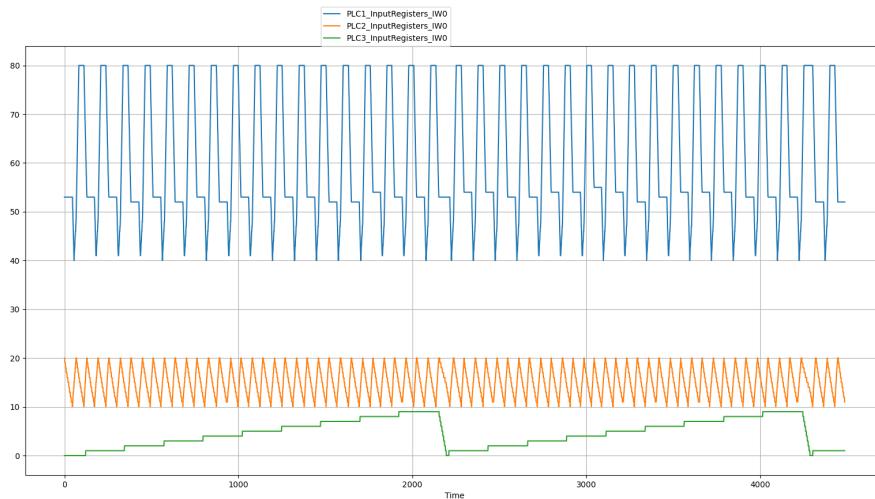
---

<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.

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

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

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



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

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

849

850     **Conjecture 2:**

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

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

---

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

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

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

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

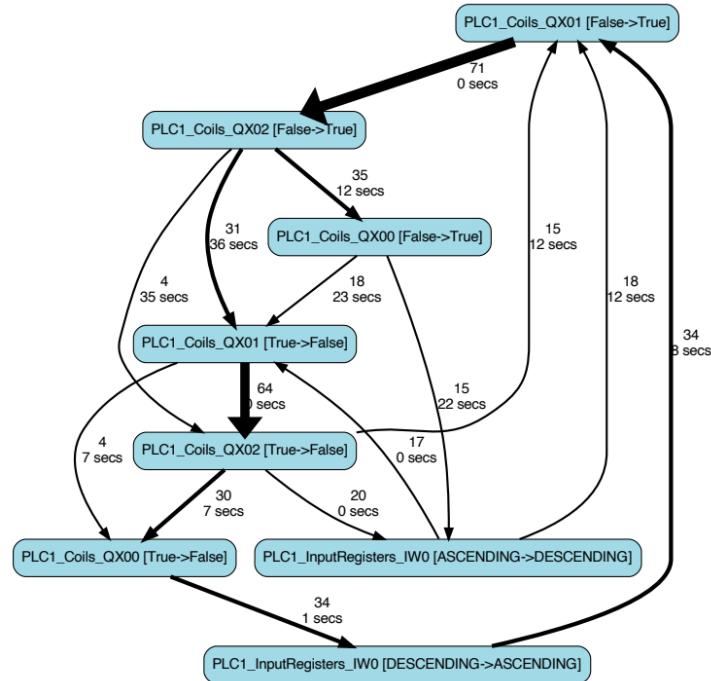
863 **Property 4:** PLC2 sends messages to PLC1 (see Figure 3.6b) which are  
864 recorded to PLC1\_Coils\_QX02.

865 **Conjecture 3:** PLC2\_Coils\_QX00 determines the trend in tank T-202 (Figure  
866 3.6b).

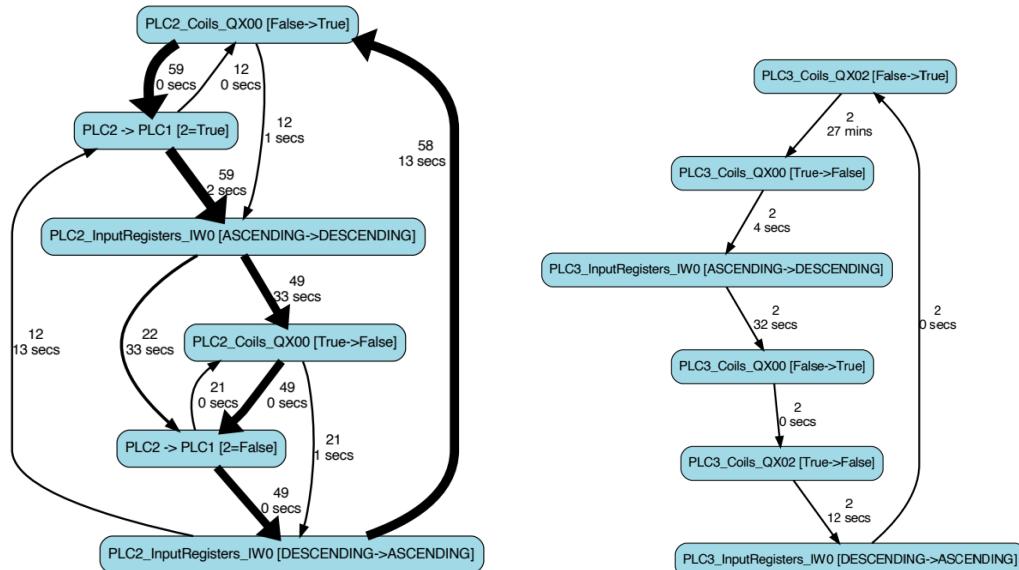
867 When this register is set to *True*, the input register PLC2\_InputRegisters\_IW0  
868 related to the tank controlled by PLC2 starts an **ascending trend**; vice  
869 versa, when the coil register is set to *False*, the input register starts a  
870 **descending trend**.

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

874 **Conjecture 5:** PLC3\_Coils\_QX00 starts a **decreasing trend** in tank T-203, re-  
875 lated to PLC3\_InputRegisters\_IW0 and controlled by PLC3, whereas  
876 PLC3\_Coils\_QX02 starts an **increasing trend** on the tank (see Figure  
877 3.6c)



**(a) States in PLC1**



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

**(c) States in PLC3**

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

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

---

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

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

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

894  
895 PLC1\_InputRegisters\_IW0 >= PLC1\_MemoryRegisters\_MW0 == 40.0  
896 PLC1\_InputRegisters\_IW0 <= PLC1\_MemoryRegisters\_MW1 == 80.0  
897 PLC2\_InputRegisters\_IW0 >= PLC2\_MemoryRegisters\_MW0 == 10.0  
898 PLC2\_InputRegisters\_IW0 <= PLC2\_MemoryRegisters\_MW1 == 20.0  
899 PLC3\_InputRegisters\_IW0 >= PLC3\_MemoryRegisters\_MW0 == 0.0  
900 PLC3\_InputRegisters\_IW0 <= PLC3\_MemoryRegisters\_MW1 == 9.0  
901  
902 i.e., that the MemoryRegisters\_MW0 and MemoryRegisters\_MW1 registers of  
903 each PLC contain the **absolute minimum and maximum setpoints**, re-  
904 spectively (*Property 5*).

905 There is also a confirmation regarding *Property 4*: from the computed  
906 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

907  
908 PLC1\_Coils\_QX01 == PLC1\_Coils\_QX02 == PLC2\_Coils\_QX00  
909  
910 and from this derive that there is a **communication channel between PLC2**  
911 **and PLC1**, where the value of PLC2\_Coils\_QX00 is copied to PLC1\_Coils\_QX01  
912 and PLC1\_Coils\_QX02 (*Property 6*).

913 Regarding the **relationships between actuator state changes and mea-**  
914 **surement trends**, invariant analysis yields the results summarized in the  
915 following rules:

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

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

919 PLC2\_InputRegisters\_IW0 == PLC2\_MemoryRegisters\_MW0 == 20.0 && PLC2\_slope > 0  
920 is verified. On the opposite hand, if the coil is *False*, the condition  
921 PLC2\_InputRegisters\_IW0 == PLC2\_MemoryRegisters\_MW0 == 20.0 && PLC2\_slope <= 0 is verified. The  
922 *slope* is increasing if > 0, decreasing if < 0, stable otherwise.

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

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

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

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

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

---

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

### 938 3.2.7 Limitations

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

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

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

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

957 Moreover, there is the presence of many *hardcoded* variables and condi-  
958 tions within the scripts: this makes the system unconfigurable and unable  
959 to properly perform the various stages of the analysis as errors can occur  
960 due to incorrect data and mismatches with the system under analysis.  
961 Having considered, furthermore, only the Modbus protocol for network  
962 communications between the PLCs is another major limiting factor and

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

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

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

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

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

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

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

---

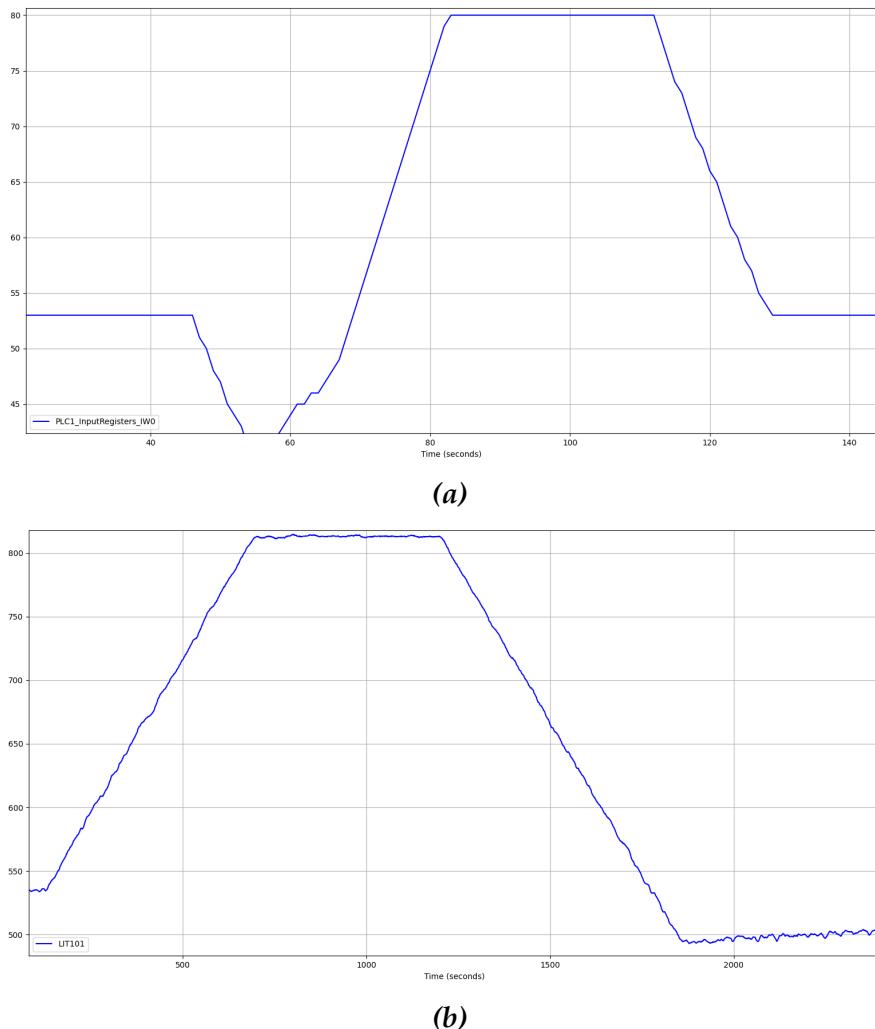
993 tanks. Consider, for example, measuring and controlling the chemi-  
994 cals in the water, the pressure of the liquid in the filter unit, or more  
995 simply the amount of water flow at a given point or time.

996 All these must be considered and represent a number of additional  
997 variables that makes analysis and consequently reverse engineering  
998 of the system more difficult.

999 ***Second limitation*** The second critical aspect concerns the **simulation of**  
1000 **the physics of the liquid** inside the tanks: Simulink does not con-  
1001 sider the fact that inside a tank that is filling (emptying) the liquid  
1002 in it undergoes **fluctuations** which cause the level sensor not to see  
1003 the water level constantly increasing (decreasing) or at most being  
1004 stable at each point of detection. Figure 3.7 exemplifies more clearly  
1005 with an example the concept just expressed: these oscillations cause  
1006 a **perturbation** in the data.

1007 This issue leads to the difficulty, on a real physical system, of **cor-**  
1008 **rectly calculating the trend of a measurement** by using the slope  
1009 attribute: if this was obtained with a too low granularity, the trend  
1010 will be oscillating between increasing and decreasing even when in  
1011 reality this would be in general increasing (decreasing) or stable; on  
1012 the other hand, if the slope was obtained with a too high granularity  
1013 there is a loss of information and the trend may be "flattened" with  
1014 respect to reality.

1015 In the present case, the slope in the Simulink model was calculated  
1016 statically with a (very) low granularity, 5 and 6 seconds according  
1017 to the Properties 7 and 9 described in the original paper: an aver-  
1018 agey careful reader will have already guessed that this granularity  
1019 is inapplicable to the real system in Figure 3.7b. As we will later see,  
1020 we need to **operate on the data perturbations** to be able to obtain a  
1021 suitable granularity and a correct calculation of the slope and conse-  
1022 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.

1023 **Pre-processing** In the pre-processing phase, the authors make use of a  
 1024 Python script to merge all the datasets of the individual PLCs into a single  
 1025 dataset, remove the (supposedly) unused registers, and finally enrich the  
 1026 obtained dataset with additional attributes. These attributes, as seen in  
 1027 3.2.2, are:

- 1028 • the **previous value** of all registers;

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

---

- 1029 • some **additional relative setpoints** named PLC $x$ \_Max\_safety and  
1030 PLC $x$ \_Min\_safety (where  $x$  is the PLC number), which represent a  
1031 kind of alert on reaching the maximum and minimum water levels  
1032 of the tanks;
- 1033 • the **measurement slope**.

1034 ***First limitation*** Merging the datasets of all individual PLCs into a single  
1035 dataset representing the entire system can be a sound practice if the  
1036 system to be analyzed is (very) small as is the testbed analyzed here,  
1037 consisting of a few PLCs and especially a few registers. If, however,  
1038 the complexity of the system increases, this type of merging can be-  
1039 come counterproductive and make it difficult to analyze and under-  
1040 stand the data obtained in subsequent steps.

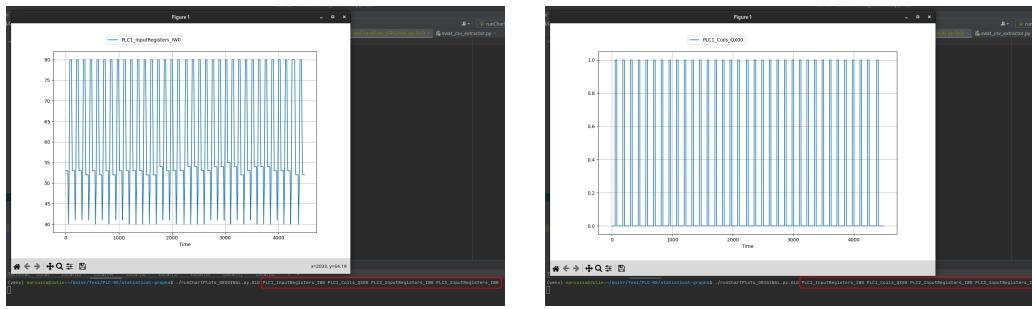
1041 In short, there is no possibility to analyze only a subsystem and thus  
1042 make the analysis faster and more understandable. Moreover, a data  
1043 gathering can take up to days, and the analyst/attacker may need to  
1044 make an analysis of the system isolating precise time ranges, ignor-  
1045 ing everything that happens before and/or after: all of this, with the  
1046 tool we have seen, cannot be done.

1047 ***Second limitation*** Regarding the additional attributes, looking at the code  
1048 of the script that performs the enrichment, we observed that **some at-**  
1049 **tributes were manually inserted** after the merging phase: we are re-  
1050 ferring in particular to the attributes PLC $x$ \_Max\_safety and PLC $x$ \_Min\_safety,  
1051 whose references were moreover hardcoded into the script, and the  
1052 *slope* whose calculation method we mentioned in the previous para-  
1053 graph about the testbed limitations.

1054 In the end, only the attribute *prev* related to the value at the previous  
1055 point of the detection is inserted automatically for all registers, more-  
1056 over without the possibility to choose whether this attribute should  
1057 be extended to all registers or only to a part.

1058 **Graphs and Statistical Analysis** Describing the behavior of graphical  
1059 analysis in Section 3.2.3 we had already mentioned that only one register

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



(a) Chart for PLC1\_InputRegisters\_IW0

(b) Chart for PLC1\_Coils\_QX00

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

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

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

**Second limitation** On the other hand, regarding the statistical analysis, two observations need to be made: the first is that for the given system, I personally was unable to appreciate the usefulness of the generated histogram in Figure 3.3b, as it does not provide any particular new information that has not already been obtained from the graphical analysis (except maybe something marginal); the second obser-

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

1082 vation pertains to the presentation of statistical information obtained  
1083 from the histogram plot. In certain cases, the histogram plot itself  
1084 can overshadow the displayed statistical information. These statis-  
1085 tics are actually shown on the terminal from which the script is exe-  
1086 cuted. However, to an inattentive or unfamiliar user, these statistics  
1087 may be mistaken for debugging output or warnings, as they coin-  
1088 cide with the display of the histogram plot window, which takes the  
1089 focus (see Figure 3.9).

1090 In general, however, little statistical information is provided.

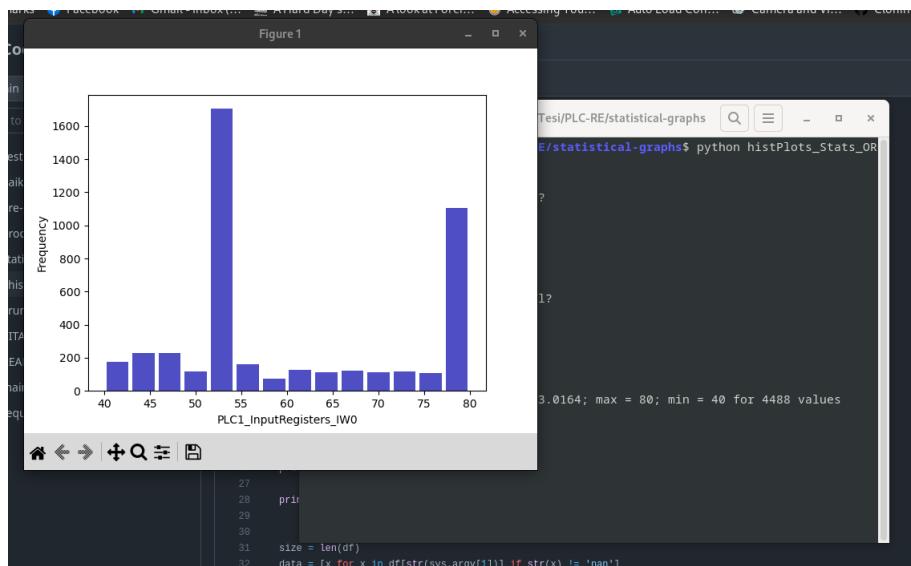


Figure 3.9: Histogram plot overshadowing statistical information shown on the terminal window in the background

1091 In general, however, little statistical information is provided.

1092 **Business Process Mining and Analysis** Concerning the data mining,  
1093 this is a purely ad *hoc* solution, designed to work under special conditions:  
1094 first, the timestamped dataset of the physical process and the one obtained  
1095 after the packet sniffing operation of Modbus traffic on the network need  
1096 to be synchronized and have the same granularity, in this case one event  
1097 per second.

1098 It is relatively easy, therefore, to find correspondences between Modbus

1099 commands sent over the network and events occurring on the physical  
1100 system, such as state changes in actuators, due in part to the fact that the  
1101 number of communications over the network is really small (see Section  
1102 3.2.1).

1103 ***First limitation*** In a real system, network communications are much more  
1104 numerous and involve many more devices even in the same second:  
1105 finding the exact correspondence with what is happening in the cy-  
1106 ber physical system becomes much more difficult.

1107 Since this is, as mentioned, an *ad hoc* solution, only the Modbus pro-  
1108 tocol is being considered: as widely used as this industrial protocol  
1109 is, other protocols that are widely used [53] such as EtherNet/IP (see  
1110 Section 2.2.6.2) or Profinet should be considered in order to extend  
1111 the analysis to other industrial systems that use a different commu-  
1112 nication network.

1113 ***Second limitation*** The other limiting aspect of the business process min-  
1114 ing phase is the **process mining software** used to generate the ac-  
1115 tivity diagram. As mentioned in Section 3.2.5, the process mining  
1116 software used by Ceccato et al. is **Disco**: this is commercial soft-  
1117 ware, with an academic license lasting only 30 days (although free of  
1118 charge), released for Windows and MacOS operating systems only,  
1119 which makes its use under Linux systems impossible except by us-  
1120 ing emulation environments such as Wine.

1121 For what is my personal vision and training as a computer scientist,  
1122 it would have been preferable to use a *cross-platform, freely licensed*  
1123 *open source* software alternative to Disco: one such software could  
1124 have been **ProM Tools** [54], a framework for process mining very  
1125 similar to Disco in functionality, but fitting the criteria just described,  
1126 or use Python libraries such as **PM4PY** [55], which offer ready-to-use  
1127 algorithms suitable for various process mining needs.

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

1128 **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 safetv == 77.0
```

Figure 3.10: Example of Daikon's output

1134 **First limitation** The biggest problem with Daikon applied to the computation of invariants of an industrial system is the difficult reading 1135 of the resulting output: the software in fact returns a very long list 1136 of invariants, one invariant per line, many of no use and without 1137 correlating invariants that may have common features or deriving 1138

1139 additional information from them. The process of screening and rec-  
 1140ognizing the significant invariants, as well as the correlation between  
 1141them, must be done by a human: certainly not an easy task given the  
 1142volume of invariants one could theoretically be faced with (hundreds  
 1143and hundreds of invariants). An example of Daikon's output can be  
 1144seen in Figure 3.10.

1145 **Second limitation** The bash script used in this phase of the analysis does  
 1146not help at all in deriving significant invariants more easily: it merely  
 1147launches Daikon and saves its output to a text file by simply redirect-  
 1148ing the stdout to file. No data reprocessing is done during this step.  
 1149In addition, if a condition is to be specified to Daikon before perform-  
 1150ing the analysis, it is necessary each time to edit the .spinfo file by  
 1151manually entering the desired rule, an inconvenient operation when  
 1152multiple analyses are to be performed with different conditions each  
 1153time.

1154 Table 3.1 provides a summary of the limitations discussed regarding the  
 1155 Ceccato et al. framework.

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

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

---

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

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

## A Substantial Improvement to Ceccato et al.'s Framework

1156 IN CHAPTER 3, we presented the state of the art of *process comprehension*  
1157 of an Industrial Control System (ICS) with a focus on the methodology  
1158 proposed by Ceccato et al. [9][Section 3.2], explaining what it consists of,  
1159 its practical application on a testbed, and most importantly highlighting  
1160 its limitations and critical issues (see Section 3.2.7).

1161 In this chapter we will present a **proposal to improve the method-**  
1162 **ology** presented in the previous chapter, addressing most of the critical  
1163 issues (or at least trying to do so) mentioned above by almost completely  
1164 rewriting the original framework, enhancing its functionalities and insert-  
1165 ing new ones where possible, while preserving its general structure and  
1166 approach. The system analysis will in fact consist of the same four steps  
1167 as in the original methodology (Data Pre-processing, Graph and Statistical  
1168 Analysis, Business Process Mining and Invariants Inference), but each of  
1169 them will be deeply revised in order to provide a richer, clearer and more  
1170 complete process comprehension of the industrial system to be analyzed  
1171 and its behavior.

1172 As it may have already been noted, my proposals do not involve im-  
1173 proving the data gathering phase: this is due simply to the fact that the  
1174 novel framework will not be tested on the same case study used by Cec-

1175 cato et, al. (Section 3.2.1), but on a different case study, the iTrust SWaT  
1176 system [36], of which (some) datasets containing the execution trace of  
1177 the physical system and the network traffic scan are already provided by  
1178 iTrust itself. For more details about this case study, the reader is referred  
1179 to Chapter 5.

## 1180 4.1 The Proposed New Framework

1181 In our version of the framework we decided to follow a few design  
1182 choices:

- 1183 1. it must be implemented in a **single programming language**;
- 1184 2. it must be **independent of the system** to be analyzed;
- 1185 3. It must provide greater **flexibility and ease of use** for the user at  
1186 every stage.

1187 In the following, we discuss these three features in more detail.

1188 **Single Programming Language** The original tool was implemented us-  
1189 ing various programming languages in each of the different phases:  
1190 from Python up to Java, passing through Bash scripting.  
1191 In our opinion, this heterogeneity makes it more difficult and less  
1192 intuitive for the user to operate on the tool: moreover, the use of  
1193 multiple technologies makes it more difficult to maintain the code  
1194 and add new features, particularly if only a single person is manag-  
1195 ing the code (he/she might be proficient in one language, but little  
1196 of the others).

1197 For these reasons, we decided to use a single programming language,  
1198 to ensure homogeneity to the framework and ease of use and main-  
1199 tenance of the code for anyone who wants to manage it in the future:  
1200 we chose to use Python, because of its simplicity and easy readability  
1201 combined with its versatility and powerfulness: moreover, Python

1202 can count on a massive number of available libraries and packages  
1203 that meet all kinds of needs.

1204 **System Independence** One of the biggest limitations of Ceccato et al.'s  
1205 tool that we highlighted in Section 3.2.7 is the fact that it is **highly**  
1206 **dependent on the testbed used**: that is, it is *not* possible to configure  
1207 any of the tool's parameters to analyze different industrial systems.  
1208 To overcome this issue and make my framework independent of the  
1209 system to be analyzed, also eliminating all references to hardcoded  
1210 variables and values present in the previous tool, we decided to use a  
1211 **general configuration file**, named *config.ini*, in which the user can, at  
1212 will, customize all the parameters necessary to perform the analysis  
1213 of the targeted system.

1214 **Flexibility and Ease on Use** The lack of flexibility and ease of use in a tool  
1215 can be a significant disadvantage, limiting its effectiveness and mak-  
1216 ing it challenging for the user to get the desired outcomes. The orig-  
1217 inal tool suffered from these limitations, with users having to run  
1218 scripts from the command line, with little to no options or param-  
1219 eters available to customize the analysis. As a result, the tool was  
1220 not user-friendly and lacked the flexibility to adapt to specific user  
1221 needs.

1222 To settle these issues, I enhanced the command-line interface in the  
1223 novel framework by adding new options and parameters. These  
1224 new features provide the user with greater flexibility, enabling to  
1225 specify parameters and options that allow for more in-depth anal-  
1226 ysis and focused results analyzing data more effectively and effi-  
1227 ciently. With these enhancements, the framework has become more  
1228 user-friendly, reducing the learning curve and making it more acces-  
1229 sible to a wider range of users.

1230 This, in turn, makes the framework more valuable and useful, in-  
1231 creasing its adoption and effectiveness across a range of industrial  
1232 control systems and applications.

1233 Moreover, with new options and parameters users no longer have to  
 1234 rely solely on the command line interface, which can be challenging  
 1235 and intimidating for those with limited technical expertise. Instead,  
 1236 users can now access a range of customizable options and parame-  
 1237 ters, making the tool more intuitive and user-friendly.

1238 Overall, the enhancements made to the framework represent a significant  
 1239 step forward in making it more effective, efficient, and user-friendly.

### 1240 4.1.1 Framework Structure

1241 The proposed framework follows a similar structure to the original  
 1242 tool, with a division into five main directories representing different phases  
 1243 of the analysis: **data pre-processing, graphs and statistical analysis, pro-**  
**1244 cess mining, and invariant analysis**. A new phase is added compared to  
 1245 the original, concerning the **analysis of the network traffic**. These directo-  
 1246 ries contain the corresponding Python scripts responsible for performing  
 1247 the analysis, along with any necessary subdirectories and input/output  
 1248 files to ensure the proper functioning of the framework.

```
1249 .
1250   |-- config.ini
1251   |-- daikon
1252     |-- Daikon_Invariants
1253     |-- daikonAnalysis.py
1254     |-- runDaikon.py
1255   |-- network-analysis
1256     |-- data
1257     |-- networkAnalysis.py
1258     |-- export_pcap_data.py
1259     |-- swat_csv_extractor.py
1260   |-- pre-processing
1261     |-- mergeDatasets.py
1262     |-- system_info.py
1263   |-- process-mining
1264     |-- data
1265     |-- process_mining.py
```

```
1266    |-- statistical-graphs  
1267        |-- histPlots_Stats.py  
1268        |-- runChartSubPlots.py
```

*Listing 4.1: Novel Framework structure and Python scripts*

1269     Ahead of these directories there is the most important part, that allows  
1270     the framework to be independent of the industrial control system being  
1271     analyzed: the *config.ini* file. Here the user can configure general parame-  
1272     ters and options, such as paths to read from or write files to, or related to  
1273     individual analysis phases.

1274     The file is divided into sections, each covering a different aspect of the con-  
1275     figuration: each section contains user-customizable parameters that will  
1276     then be called within the Python scripts that constitute the framework.

1277     Sections of *config.ini* are:

- 1278         • **[PATHS]**: defines general paths such as the project root directory;
- 1279         • **[PREPROC]**: includes parameters needed for the **pre-processing phase**,  
1280             like the directory containing the raw datasets of the individual PLCs  
1281             and *granularity* for the slope calculation. Granularity is the time in-  
1282             terval over which the slope is calculated;
- 1283         • **[DATASET]**: defines settings and parameters used during the **dataset**  
1284             **enrichment** stage, for example the additional attributes;
- 1285         • **[DAIKON]**: defines parameters needed for **invariant analysis** with  
1286             Daikon, e.g. directories and files containing the outcomes of the anal-  
1287             ysis;
- 1288         • **[MINING]**: contains parameters used during the **process mining**  
1289             phase, such as data directory;
- 1290         • **[NETWORK]**: includes specific settings for extracting the data ob-  
1291             tained from the packet sniffing phase on the ICS network and con-  
1292             verting it to CSV format. It also defines the **network protocols** that  
1293             are to be analyzed.

### 1294 4.1.2 Python Libraries and External Tools

1295 As the framework has been developed entirely in Python, the objective  
1296 was to minimize reliance on external tools and instead integrate various  
1297 functionalities within the framework itself. The aim was to make the  
1298 framework independent from external software. The only remaining ex-  
1299 ternal tool from the Ceccato et al. tool is Daikon. This choice was made  
1300 because there is currently no better alternative or Python package avail-  
1301 able that offers the same functionalities as Daikon.

1302 Instead, the framework extensively utilizes Python libraries for han-  
1303 dling various functionalities and input data. The core libraries on which  
1304 the framework relies are:

- 1305 • **Pandas**, also used in the previous tool for dataset management, but  
1306 whose use here has been deepened and extended
- 1307 • **NumPy**, often used together with Pandas to perform some opera-  
1308 tions to support it;
- 1309 • **MatPlotLib**, for managing and plotting graphical analysis;
- 1310 • other scientific libraries such as **SciPy**, **StatsModel** [56] and **Net-**  
1311 **workX** [57], for mathematical, statistical and analysis operations on  
1312 the data;
- 1313 • **GraphViz**, for the creation of activity diagrams in the process mining  
1314 phase.

1315 Having now seen the structure of the framework, in the next sections we  
1316 will go into more detail describing our proposal.

## 1317 4.2 Analysis Phases

### 1318 4.2.1 Phase 1: Data Pre-processing

1319 *Data Pre-processing phase* is probably the most delicate and significant  
1320 one: depending on how large the industrial system to be analyzed is, the

1321 data collected, and how it is enriched using the additional attributes, the  
1322 subsequent system analysis will provide more or less accurate outcomes.

1323 The previous tool has several limitations, particularly at this stage. It  
1324 does not allow for the isolation of a subsystem, either in terms of time or  
1325 the number of PLCs to be analyzed. The system is considered as a whole  
1326 without the ability to focus on specific subsystems. Additionally, many of  
1327 the additional attributes had to be manually added, and for the ones en-  
1328 tered automatically, there is no way to specify the register type to associate  
1329 them with.

1330 The combination of these limitations, along with the presence of hard-  
1331 coded references to attributes and registers in the tool's code, makes the  
1332 analysis of the system more challenging. Furthermore, it compromises the  
1333 accuracy and reliability of the obtained results in terms of both quantity  
1334 and quality.

1335 In the proposed framework, these issues have been addressed by in-  
1336 corporating new features. Firstly, the framework allows for the selection  
1337 of a subsystem from the command line based on both temporal criteria  
1338 and the specific PLCs to be included. This enables more focused and tar-  
1339 geted analysis. Additionally, we have revamped the process of enriching  
1340 the resulting dataset by eliminating manual entry of additional attributes.  
1341 Instead, users now have the flexibility to determine the type of additional  
1342 attribute to associate with a specific register.

1343 Furthermore, after the pre-processing stage, a preliminary analysis can be  
1344 conducted on the resulting dataset. This analysis aims to identify the reg-  
1345 isters that are associated with actuators, measurements, and hardcoded  
1346 setpoints or constants. It provides insights into the dataset and helps in  
1347 refining the enrichment step. The parameters for this analysis can be con-  
1348 figured in the *config.ini* file, allowing for customization and fine-tuning of  
1349 the process.

1350  
1351 In the upcoming sections, we will delve into a more comprehensive ex-  
1352 amination of the achievements made in this framework.

1353 **4.2.1.1 Subsystem Selection**

1354 In the previous tool, the datasets for each individual PLC in CSV for-  
 1355 mat were required to be placed in a specific directory that was hardcoded  
 1356 in the script. The script would then merge and enrich these datasets to  
 1357 generate a single output dataset representing the complete process trace  
 1358 of the industrial system. However, the script did not provide options to  
 1359 select specific PLCs for analysis or define a temporal range for analysis.  
 1360 This lack of flexibility made the analysis more complex, especially when  
 1361 dealing with *transient states* (i.e., general states in which the industrial sys-  
 1362 tem is still initializing before actually reaching full operation) or when fo-  
 1363 cusing on specific parts of the industrial system during certain periods of  
 1364 interest. The fixed dataset structure also may increase the number of vari-  
 1365 ables that could be analyzed.  
 1366 Furthermore, the previous tool did not allow for specifying an output CSV  
 1367 file to save the resulting dataset. Each dataset creation and enrichment op-  
 1368 eration would overwrite the previous file, making it inconvenient for com-  
 1369 parisons between different execution traces unless the files were manually  
 1370 renamed.

1371 The proposed framework addressed these issues by introducing im-  
 1372 provements. First of all, in the general *config.ini* file there are some general  
 1373 default settings about paths, and among them the one concerning the di-  
 1374 rectory where to place the datasets of the individual PLCs to be processed.  
 1375 In addition to this option, there are other ones that define further aspects  
 1376 related to the operations performed in this phase. Listing 4.2 shows the  
 1377 settings in question:

```
1378 [PATHS]
1379 root_dir = /home/marcuzzo/UniVr/Tesi
1380 project_dir = %(root_dir)s/PLC-RE
1381 net_csv_path = %(root_dir)s/datasets_SWaT/2015/Network_CSV
1382
1383 [PREPROC]
1384 raw_dataset_directory = datasets_SWaT/2015 # Directory
1385   ↪ containing datasets
```

```
1386     dataset_file = PLC_SWaT_Dataset.csv # Default output  
1387     ↪ dataset  
1388     granularity = 10 # slope granularity  
1389     number_of_rows = 20000 # Seconds to consider  
1390     skip_rows = 100000 # Skip seconds from beginning
```

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

1391 At the same time, the user has the option to specify these settings via the  
1392 command line using the new Python script called `mergeDatasets.py`, lo-  
1393 cated in the pre-processing directory of the project. Any options provided  
1394 through the command line will override the default settings specified in  
1395 the `config.ini` file. These options are:

- 1396 • **-s or --skiprows:** initial transient period (expressed in seconds) to  
1397 be skipped. This option is useful in case the system has an initial  
1398 transient or the analyzer wishes to start the analysis from a specific  
1399 point in the dataset;
- 1400 • **-n or --nrows:** time interval under analysis, expressed in terms of the  
1401 number of rows in the dataset.  
1402 This option makes a **selection** on the data of the dataset;
- 1403 • **-p or --plcs:** PLCs to be merged and enriched. The user can specify  
1404 the desired PLCs by indicating the CSV file names of the associated  
1405 datasets with no limitations on number.  
1406 This option makes a **projection** on the data of the dataset.
- 1407 • **-d or --directory:** performs the merge and enrichment of all CSV files  
1408 contained in the directory specified by user, overriding the default  
1409 setting in `config.ini`. It is in fact the old functionality of the previous  
1410 tool, maintained here to give the user more flexibility and conve-  
1411 nience in case he wants to perform the analysis on the whole system.  
1412 This is also the default behavior in case the `-p` option is not specified.
- 1413 • **-o or --output:** specifies the name of the file in which the obtained  
1414 dataset will be saved. It must necessarily be a file in CSV format.

- 1415 • **-g** or **--granularity**: specifies a granularity (expressed in seconds)  
 1416 that will be used to calculate the measurement slope during the dataset  
 1417 enrichment phase. We will discuss this later in Section 4.2.1.2.

1418 **4.2.1.2 Dataset Enrichment**

1419 After a step in which a function is applied to each PLC-related dataset  
 1420 to eliminate its unused registers within the system<sup>1</sup>, the **dataset enrichment**  
 1421 **operation** is performed.

1422 This operation differs from the previous version not only in the fact that it  
 1423 is performed on each individual dataset and not on the resulting dataset,  
 1424 but also in the additional attributes: not only are they greater in number,  
 1425 but they are automatically calculated and inserted by the `mergeDatasets.py`  
 1426 script into the dataset and, most importantly, it is possible to decide through  
 1427 the parameters in the `config.ini` configuration file under the [DATASET] sec-  
 1428 tion to which registers these attributes should be assigned.

1429 In Listing 4.3 we can see the list of additional attributes and how they  
 1430 should be associated with the registers of the dataset:

```
1431 [DATASET]
1432 timestamp_col = Timestamp
1433 max_prefix = max_
1434 min_prefix = min_
1435 max_min_cols_list = lit|ait|dpit
1436 prev_cols_prefix = prev_
1437 prev_cols_list = mv[0-9]{3}|p[0-9]{3}
1438 trend_cols_prefix = trend_
1439 trend_cols_list = lit
1440 trend_period = 150
1441 slope_cols_prefix = slope_
1442 slope_cols_list = lit
```

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

1443 Following is a brief explanation of the parameters just seen:

---

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

1444 **timestamp\_col** indicates the name of the column that contains the data  
1445 timestamps. This parameter is used not only in this phase, but is  
1446 also referred to in the Process Mining phase. In the previous work,  
1447 this parameter was hardcoded and not configurable (and thus caus-  
1448 ing errors if the system being analyzed changed)

1449 **max\_prefix, min\_prefix, max\_min\_cols\_list** refer to any relative maximum  
1450 or minimum values (*relative setpoints*) of one or more measures and  
1451 that can be found and inserted as new columns within the dataset.  
1452 The first two parameters indicate the prefix to be used in the column  
1453 names affected by this additional attribute, while the third specifies  
1454 of which type of registers we want to know the maximum and/or  
1455 minimum value reached (several options can be specified using the  
1456 logical operator | - or).

1457 If, for example, we want to know the maximum value of the regis-  
1458 ters associated with the tanks, indicated in the iTrust SWaT system  
1459 by the prefix LIT, we only need to specify the necessary parameter in  
1460 the *config.ini* file, so `max_min_cols_list = lit`.

1461 The result will be to have in the dataset thus enriched a new column  
1462 named `max_P1_LIT101`.

1463 **prev\_cols\_prefix, prev\_cols\_list** refer to the values at the previous time  
1464 instant of the registers specified in `prev_cols_list`. It is possible to  
1465 specify registers using *regex*, as in the example shown. It may be use-  
1466 ful in some cases to have this value available to check, for example,  
1467 when a change of state of a single given actuator occurs. The behav-  
1468 ior of these parameters is the same as described in the point above.

1469 **slope\_cols\_prefix, slope\_cols\_list** are related to the calculation of the  
1470 slope of a specific register that contains numeric values (usually a  
1471 measure), that is, its trend. Slope calculation makes little sense on  
1472 booleans. The slope can be **ascending** (if its value is greater than  
1473 zero), **descending** (if less than zero) or **stable** (if approximately equal  
1474 to zero). We will delve into the details of slope calculation in the fol-

1475        lowing paragraph, as it pertains to the attributes `trend_cols_prefix`,  
 1476        `trend_cols_list`, and `trend_period`.

1477        Initially, the parameters for registers to be associated with each additional  
 1478        attribute may be left blank, as we may not have prior knowledge  
 1479        about the system and are unsure about which registers correspond to actuators,  
 1480        measurements, or other attributes. This information can be obtained  
 1481        from the brief analysis that follows the merging of datasets. The analysis,  
 1482        performed based on user's choice, provides indications on potential sensors,  
 1483        actuators, and other relevant information. These indications help the  
 1484        user set the desired values in the `config.ini` file and refine the enrichment  
 1485        process by re-launching the `mergeDatasets.py` script.

1486        **Slope Calculation** The *slope* is an attribute that represents the **trend** of  
 1487        the measurement being considered. It is particularly useful, in our context,  
 1488        during the inference and invariant analysis phase to gather information  
 1489        about the trend under specific conditions. The slope can generally be  
 1490        classified as **increasing** ( $slope > 0$ ), **decreasing** ( $slope < 0$ ), or **stable** ( $slope = 0$ ).

1492        Normally, the slope is calculated through a simple mathematical formula:  
 1493        given an interval  $a, b$  relative to the measurement  $l$ , the slope is given by  
 1494        the difference of these two values divided by the amount of time  $t$  that the  
 1495        measurement takes to reach  $b$  from  $a$ :

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

1496        In the proposed framework, similar to the previous tool, this time interval  
 1497        (the granularity) can be adjusted to be either long or short. The choice  
 1498        of granularity depends on the desired accuracy of the slope calculation. A  
 1499        lower granularity will provide a slope that closely reflects the actual measurement  
 1500        trend, while a higher granularity will result in flatter slope data.  
 1501        Each time interval within which the measurement is divided corresponds  
 1502        to a slope value. These slopes are calculated and added as additional at-

1503 tributes in the dataset. Later on, these slope values are used to determine  
1504 the trend of the measurement in specific situations or conditions.

1505 Calculating the slope directly from the raw measurement data can be  
1506 a suitable approach for systems where the measurements are not heavily  
1507 influenced by **perturbations**. Perturbations, such as liquid oscillations in a  
1508 tank during filling and emptying phases, can lead to fluctuating readings  
1509 of the level. In such cases, maintaining a low granularity can provide a  
1510 more accurate calculation of the overall trend that closely aligns with the  
1511 actual measurement trend. The tanks of Ceccato et al.'s testbed are an ex-  
1512 ample where this holds true.

1513 However, if perturbations significantly affect the measurement readings,  
1514 calculating the slope on individual time intervals may result in an inaccur-  
1515 ate trend definition, irrespective of the chosen granularity. In such cases,  
1516 the fluctuating nature of the measurements due to perturbations can intro-  
1517 duce errors in the slope calculation, making it less reliable as an indicator  
1518 of the actual trend.

1519 Figure 4.1 demonstrates this assertion: the measurement, in blue, refers  
1520 to the P1\_LIT101 tank of the iTrust SWaT system; in red, the slope calcu-  
1521 lation related to the measurement with three different granularities: 30  
1522 (Figure 4.1a), 60 (Figure 4.1b) and 120 seconds (Figure 4.1c). It is notice-  
1523 able that as the granularity increases, the slope values flatten. Moreover,  
1524 in the time interval between seconds 1800 and 4200, the level of P1\_LIT101  
1525 exhibits a predominantly increasing trend, yet the calculated slope values  
1526 fluctuate between positive and negative. Consequently, during the invari-  
1527 ant analysis, the overall increasing trend may not be detected, resulting in  
1528 a loss of information.

1529  
1530 The previous tool did not take into account the possibility of having strongly  
1531 perturbed data, which presented a challenge that we needed to address in  
1532 the development of the proposed framework.

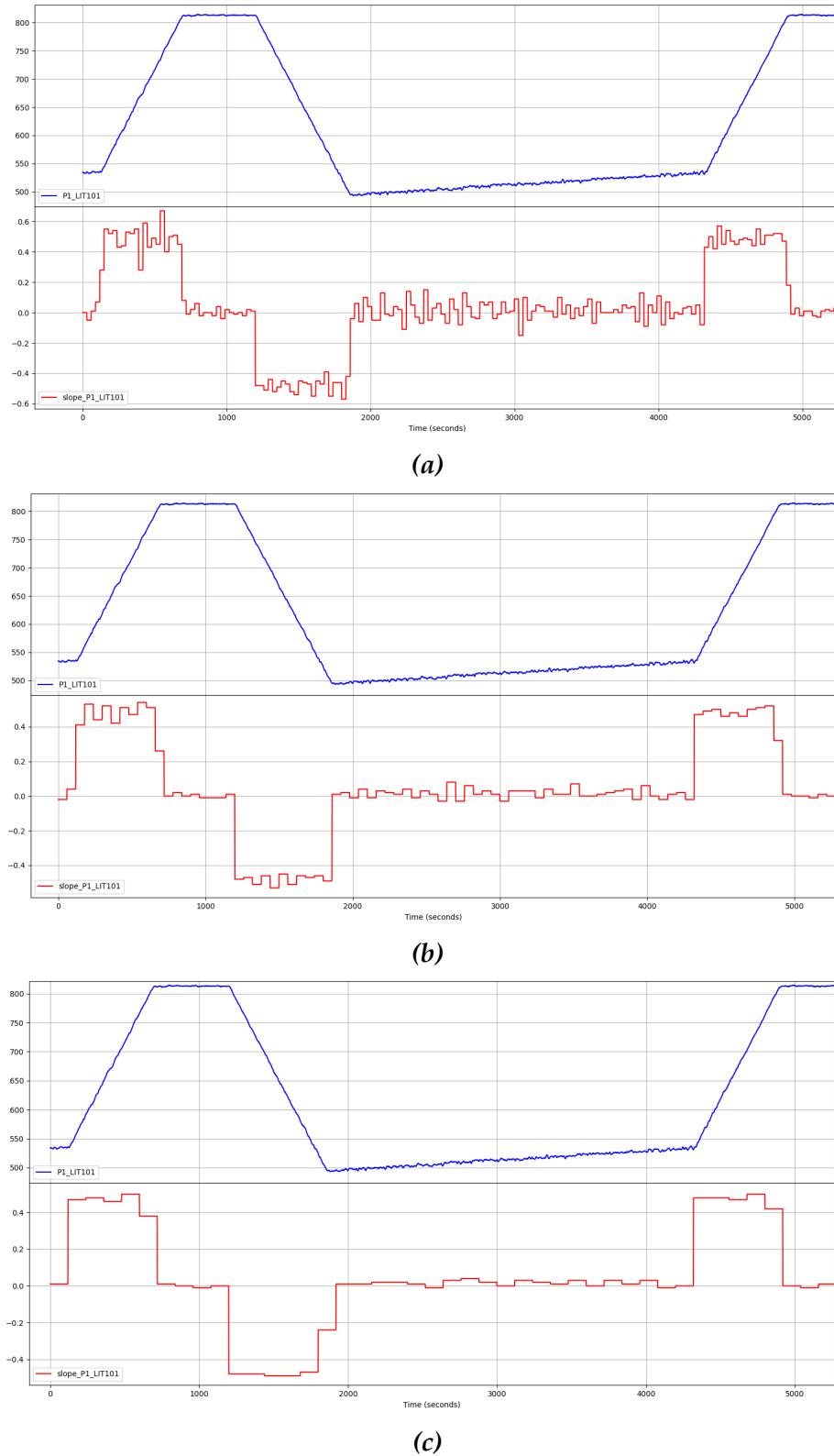


Figure 4.1: Slope comparison with granularity 30 (a), 60 (b) and 120 seconds (c)

1533     The solution to this problem involves applying techniques to reduce  
1534     the "noise" in the data, aiming to achieve a more linear trend in the mea-  
1535     surement curve. By minimizing the effects of perturbations, we can calcu-  
1536     late slopes more accurately.

1537     There are various methods available for smoothing out noise in the data.  
1538     In our framework, we focused on two commonly used approaches found  
1539     in the literature: **polynomial regression** and **seasonal decomposition**. In  
1540     addition to these two methods, we also explored the use of a **line simpli-**  
1541     **fication algorithm.**

1542

1543     *Polynomial regression* [58] is a technique that allows us to create a filter to  
1544     reduce the impact of noise on the data. By fitting a polynomial function  
1545     to the measurements, we can obtain a smoother curve that captures the  
1546     underlying trend while minimizing the effects of perturbations.

1547     *Seasonal decomposition* [59], specifically the part related to trending, is an-  
1548     other method we explored. It involves decomposing the time series into  
1549     different components, such as trend, seasonality, and residual. By isolat-  
1550     ing the trend component, we can obtain a cleaner representation of the  
1551     underlying pattern in the data.

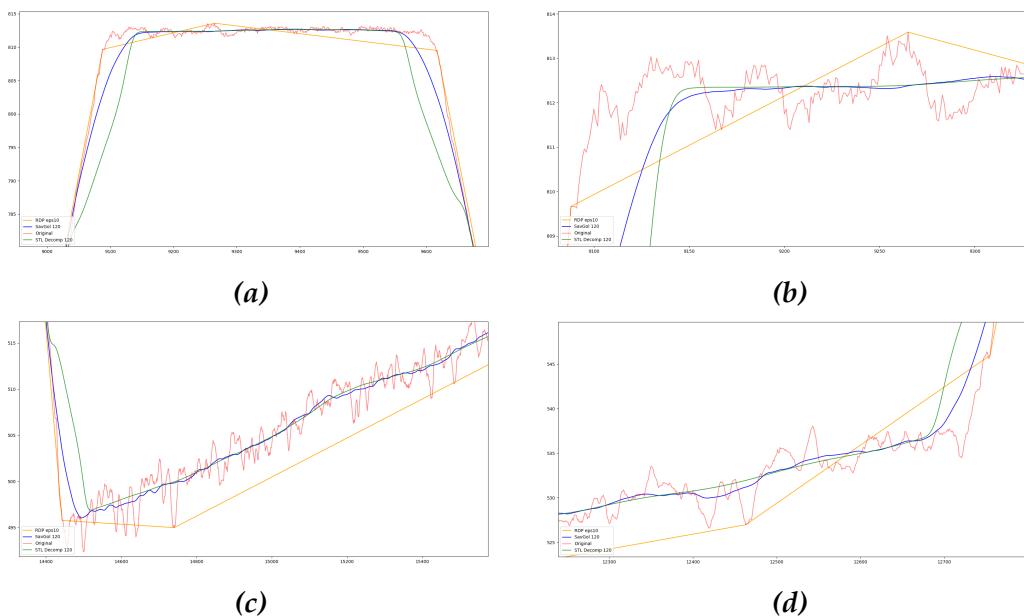
1552     *Line simplification algorithms* [60] aim to reduce the complexity of a polyline  
1553     or curve by approximating it with a simplified version composed of fewer  
1554     points. By selectively removing redundant or less significant points, line  
1555     simplification algorithms help reduce storage space and computational re-  
1556     quirements while preserving the overall shape and characteristics of the  
1557     original line.

1558     Regarding polynomial regression, we evaluated the use of the **Savitzky-**  
1559     **Golay filter** [61] as a smoothing technique. For seasonal decomposition,  
1560     we explored the **Seasonal-Trend decomposition using LOESS** (STL) method  
1561     [62]. For the line simplification algorithm, we specifically considered the  
1562     **Ramer-Douglas-Peucker** (RDP) algorithm [63].

1563

1564     Figure 4.2 shows a quick graphical comparison of these techniques com-  
1565     pared with the original data. The solution adopted is the *STL decomposition*

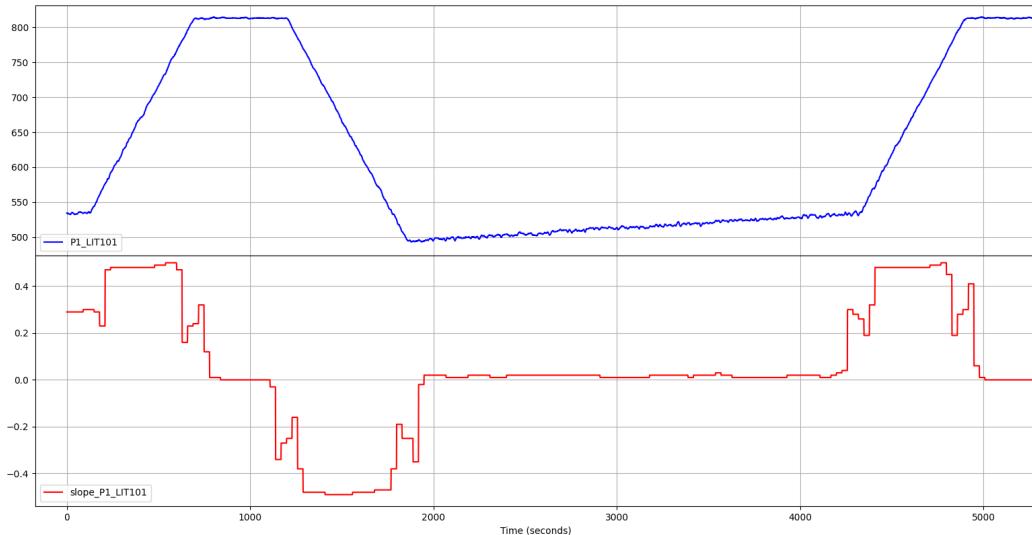
method, which effectively reduces noise compared to the Savitzky-Golay filter. However, it should be noted that this method may introduce some delay in certain parts of the data, as is typically observed in similar algorithms. Despite its apparent effectiveness, the RDP algorithm fails to accurately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



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

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

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



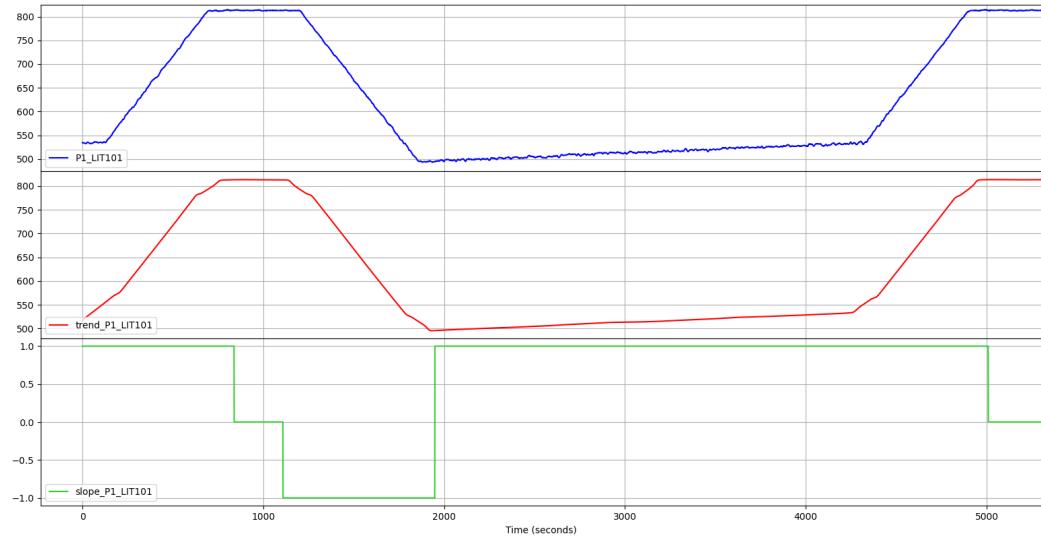
*Figure 4.3: Slope after the application of the STL decomposition*

1582 During the slope calculation, the analysis will be performed on the data  
 1583 from the additional measurement trend attributes specified in the trend\_cols\_list  
 1584 directive of the configuration file, rather than on the original unfiltered  
 1585 data.

1586 To ensure proper interpretation by Daikon, the decimal values repre-  
 1587 senting the calculated slopes are converted into **three numerical values**:  
 1588 -1, 0, and 1. These values correspond to *decreasing* (if the slope is less than  
 1589 zero), *stable* (if it is equal to zero), and *increasing* (if it is greater than zero)  
 1590 trends, respectively. Figure 4.4 displays the modified slopes along with  
 1591 the curve obtained from the STL decomposition:

#### 1592 4.2.1.3 Datasets Merging

1593 During this step, the datasets of the individual PLCs are merged, re-  
 1594 sulting in two separate datasets. The first dataset is enriched with addi-  
 1595 tional attributes but excludes the timestamp column. This dataset is in-  
 1596 tended for inference and invariant analysis. The second dataset does not  
 1597 contain any additional data and is specifically used in the process mining  
 1598 phase.



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

1599 By default, the enriched dataset will be saved in CSV format in the  
 1600 `$(project-dir)/daikon/Daikon_Invariants` directory. The other dataset,  
 1601 without additional data, will be saved in the `$(project-dir)/process-mining/data`  
 1602 directory. It's worth noting that both paths can be configured in the  
 1603 `config.ini` file. The dataset name can be specified in the `config.ini` file or  
 1604 through the `-o` command-line option. When generating the dataset for  
 1605 process mining, the script will automatically add a `_TS` suffix to the file-  
 1606 name to indicate that it includes the timestamp. This flexibility allows the  
 1607 user to provide a different filename for each output, preventing overwrit-  
 1608 ing of previous datasets. It enables the user to save the execution trace of  
 1609 the selected subsystem separately and utilize them in subsequent analysis  
 1610 phases.

#### 1611 4.2.1.4 Brief Analysis of the Obtained Subsystem

1612 After merging the datasets, the user has the option to perform an **op-**  
 1613 **tional analysis** of the resulting dataset to extract preliminary data. This  
 1614 analysis aims to gather basic information about the (sub)system and po-  
 1615 tentially refine the enrichment process. If the user chooses to proceed with

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

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

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

```
1635      Do you want to perform a brief analysis of the dataset? [y
1636      ↵ /n]: y
1637
1638      Actuators:
1639      P1_MV101 [0.0, 1.0, 2.0]
1640      P1_P101 [1.0, 2.0]
1641
1642      Sensors:
1643      P1_FIT101 {'max_lvl': 2.7, 'min_lvl': 0.0}
1644      P1_LIT101 {'max_lvl': 815.1, 'min_lvl': 489.6}
1645
1646      Hardcoded setpoints or spare actuators:
1647      P1_P102 [1.0]
1648
1649      Actuator state changes:
1650          P1_LIT101  P1_MV101  prev_P1_MV101
```

1651	669	800.7170	0	2
1652	1850	499.0203	0	1
1653	4876	800.5992	0	2
1654	6052	498.9026	0	1
1655	9071	800.7170	0	2
1656	10260	499.1381	0	1
1657	13268	801.3058	0	2
1658	14435	498.4315	0	1
1659	17423	801.4628	0	2
1660	18603	498.1567	0	1
1661				
1662	P1_LIT101	P1_MV101	prev_P1_MV101	
1663	677	805.0741	1	0
1664	4885	805.7414	1	0
1665	9079	805.7806	1	0
1666	13276	805.1133	1	0
1667	17432	804.4068	1	0
1668				
1669	P1_LIT101	P1_MV101	prev_P1_MV101	
1670	1858	495.4483	2	0
1671	6060	497.9998	2	0
1672	10269	495.9586	2	0
1673	14443	495.8016	2	0
1674	18611	494.5847	2	0
1675				
1676	P1_LIT101	P1_P101	prev_P1_P101	
1677	118	536.0356	1	2
1678	4322	533.3272	1	2
1679	8537	542.1591	1	2
1680	12721	534.8581	1	2
1681	16883	540.5890	1	2
1682				
1683	P1_LIT101	P1_P101	prev_P1_P101	
1684	1190	813.0031	2	1
1685	5395	813.0031	2	1
1686	9597	811.8256	2	1
1687	13776	812.7283	2	1
1688	17938	813.3171	2	1
1689				

```

1690     Actuator state durations:
1691     P1_MV101 == 0.0
1692     9   9   10   9   9   10   9   9   10   9
1693
1694     P1_MV101 == 1.0
1695     1174   1168   1182   1160   1172
1696
1697     P1_MV101 == 2.0
1698     669   3019   3012   3000   2981
1699
1700     P1_P101 == 1.0
1701     1073   1074   1061   1056   1056
1702
1703     P1_P101 == 2.0
1704     118   3133   3143   3125   3108

```

*Listing 4.4: Example of preliminar system analysis*

From these results we can draw the following conjectures:

- the **probable actuators** are P1\_MV101 and P1\_P101. P1\_MV101 has three states identified by the values 0, 1, and 2, suggesting it is a multi-state actuator. P1\_P101 has two states identified by the values 1 and 2, indicating a binary actuator;
- there are **two probable measures**: P1\_FIT101 and P1\_LIT101. P1\_FIT101 has values ranging from 0 to 2.7, P1\_LIT101 has values ranging from 489.6 to 815.1. Based on this information, a further conjecture can be made that P1\_LIT101 represents a tank level measurement;
- apparently there is a probable **spare actuator**, P1\_P102, whose value is always 1. No related *hardcoded setpoints* were found. From this information, another speculation can be made that the value 1 represents the **OFF state** for binary actuators, while the value 2 represents the **ON state**;
- from the analysis of state changes we can derive some **relative set-points**. For example, we observe that P1\_P101 changes state from

1721            value 1 (OFF) to value 2 (ON) when the level of P1\_LIT101 is ap-  
 1722            proximately 813, and it changes from value 2 (ON) to 1 (OFF) when  
 1723            the level of P1\_LIT101 is around 535. We can deduce that P1\_P101 is  
 1724            responsible for emptying the tank;

- 1725            • Regarding the actuators states duration, the very short duration of  
 1726            P1\_MV101 in state 0 is observable: since we are unable, at this pre-  
 1727            liminary stage, to make assumptions about this, we will try to un-  
 1728            derstand the behavior of the system in this actuator state in the next  
 1729            stages of the analysis.

1730            The information obtained here can be used, as mentioned above, to  
 1731            refine the enrichment of the dataset by setting directives in the [DATASET]  
 1732            section of the *config.ini* file, should this be empty or only partially set, or to  
 1733            make the first conjectures about the system, as we have just seen.

1734            The *system\_info.py* file can also run in standalone mode if needed:  
 1735            it takes as command-line arguments the dataset to be analyzed, a list of  
 1736            actuators, and a list of sensors. For analysis related to state changes, the  
 1737            dataset must mandatorily be of the enriched type.

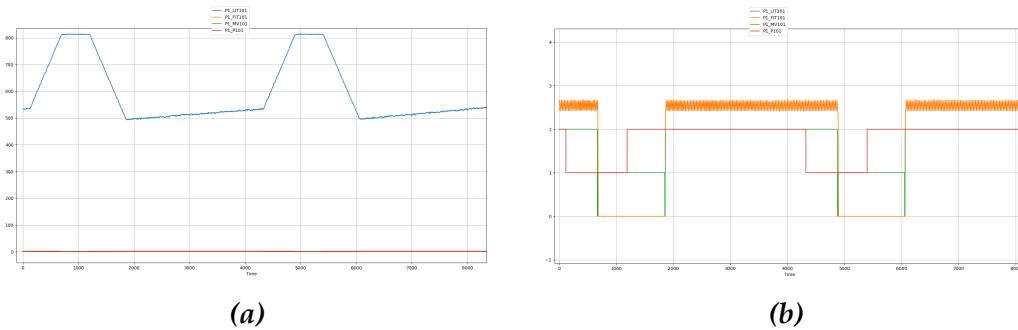
#### 1738          4.2.2 Phase 2: Graphs and Statistical Analysis

1739            The new *graph analysis* arises from the need to give the user an overview  
 1740            of the (sub)system obtained in the previous pre-processing phase, identi-  
 1741            fying more easily the typology of the registers and grasping more effec-  
 1742            tively the relationships and the dynamics that may exist between the reg-  
 1743            isters controlled by one or more PLCs, confirming the initial conjectures if  
 1744            the brief analysis described in the previous section has been performed, or  
 1745            making new ones thanks to the visual graph support.

1746            In Ceccato et al.'s framework, as mentioned in Section 3.2.7, it was only  
 1747            possible to view the chart of one register at a time. While this allowed for  
 1748            the identification or hypothesis of the register type, it made it challenging  
 1749            to establish relationships with other components of the system and derive

1750 conjectures about their behavior. To address this limitation, there was a  
 1751 need for a new tool that could provide more information in a more acces-  
 1752 sible manner.

1753 Initially, we considered adopting an approach similar to Figure 3.5,  
 1754 where all the graphs are displayed within a single plot. However, we soon  
 1755 realized that this solution was not feasible and could not be adopted. Fig-  
 1756 ure 4.5 helps to understand why.

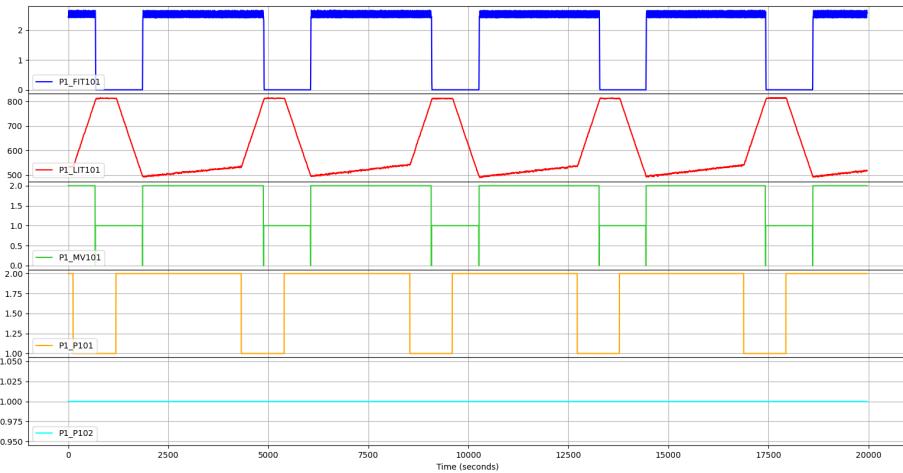


*Figure 4.5: Plotting registers on the same y-axis*

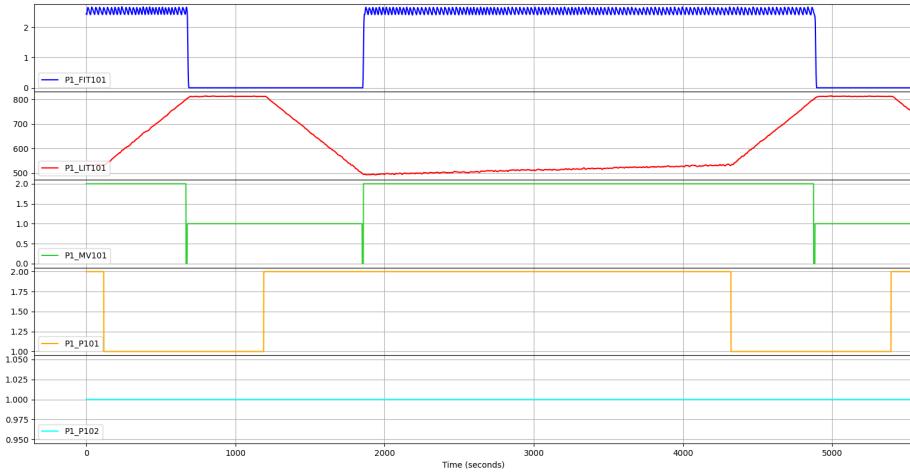
1757 Figure 4.5a highlights the main issue with this approach, which is the  
 1758 use of the same y-axis for all the charts, representing the values of individ-  
 1759 ual registers. When the range between register values is wide, it can result  
 1760 in some charts appearing as a single flat line or becoming indistinguish-  
 1761 able, making them difficult to read. Additionally, as shown in Figure 4.5b,  
 1762 when registers have similar values, the graphs can become confusing and  
 1763 harder to interpret.

1764 The solution to this issue is simple and effective: the use of **subplots**.  
 1765 Basically, each register corresponds to a subplot of the graph that shares  
 1766 the time axis (the x-axis) with the other subplots, but keeps the y-axis of  
 1767 the values of each register independent. This maintains the readability  
 1768 and comprehensibility of the charts, while simultaneously being able to  
 1769 immediately grasp the relationships between them. In addition, by shar-  
 1770 ing the time axis, it is possible to zoom in on a particular area of one of the  
 1771 charts and automatically the other ones will be zoomed in as well, thus

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



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



(b) Zooming on a particular zone of the charts

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

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

- 1779     • the data presented in the P1\_FIT101 chart provides support for as-  
1780         sociating this register with a **measurement**. Furthermore, the chart  
1781         indicates a close correlation between the trend of this register and  
1782         the periods in which P1\_LIT101 exhibits an upward trend and the  
1783         evolution of P1\_MV101. The values of P1\_FIT101, ranging from ap-  
1784         proximately 0 to 2.5, are too limited to conclude that this register  
1785         represents the level sensor of a tank. The overall trend also aligns  
1786         with this interpretation, suggesting that it is more likely a **pressure**  
1787         **or flow sensor**;
  
- 1788     • based on the information provided above, it appears that P1\_LIT101  
1789         indeed **represents the level sensor of a tank**, which was initially  
1790         assumed in the brief analysis. The hypothesis is further reinforced  
1791         by several factors. Firstly, the wide range of values observed in this  
1792         register supports the notion of it being a level sensor. Additionally,  
1793         the graphical representation of the level trend provides valuable in-  
1794         sights. It shows an initial sharp rise, followed by a period of stabi-  
1795         lization referred to as a *plateau*, and subsequently, a similarly steep  
1796         descent. Finally, there is a new phase of slower level growth com-  
1797         pared to the previous one. These patterns lend further support to  
1798         the interpretation of P1\_LIT101 as a sensor specifically measuring the  
1799         level of a tank;
  
- 1800     • at the start of the ascending trend of P1\_LIT101, the value of P1\_MV101  
1801         is 2. Conversely, during the *plateau* phase observed when the mea-  
1802         surement is approximately 800 and during the descending trend,  
1803         P1\_MV101 has a value of 1. These observations provide further confir-  
1804         mation of the initial hypothesis: the value 1 represents the **OFF state**  
1805         of the sensor, while 2 represents the **ON state**. Moreover, P1\_MV101  
1806         can be identified as the actuator responsible for causing the rising  
1807         level in P1\_LIT101;
  
- 1808     • during the brief analysis, we noticed the short duration of P1\_MV101  
1809         in state 0, but we were unable to speculate on the underlying rea-

1810           son. However, further analysis of the graph reveals that P1\_MV101  
1811           transitions and remains in the 0 state, acting as a type of "transient"  
1812           between states 1 and 2. It can be inferred that this period represents  
1813           the actual time required for the actuator to change its state.

- 1814           • we can observe the behavior of P1\_P101 in relation to P1\_LIT101 and  
1815           P1\_MV101 to understand its role. At the beginning of the descend-  
1816           ing trend of P1\_LIT101, after the plateau phase, P1\_P101 assumes a  
1817           value of 2. It then changes back to value 1 at the point of the (pre-  
1818           sumed) tank's level increase. Although this fact alone may not be  
1819           entirely clear, when comparing the behaviors of P1\_P101, P1\_LIT101,  
1820           and P1\_MV101, certain patterns emerge. When P1\_P101 and P1\_MV101  
1821           are both in state 1, the water level remains stable. However, when  
1822           P1\_P101 changes to state 2, the level of the measurement drops rapidly.  
1823           Subsequently, when P1\_MV101 also transitions to state 2, the level  
1824           slowly starts to rise again, with a sudden increase occurring when  
1825           P1\_P101 changes from 2 back to 1. From these observations, it can be  
1826           inferred that P1\_P101 serves as the **actuator responsible for empty-**  
1827           **ing the presumed tank**. State 1 represents the **OFF state**, while state  
1828           2 represents the **ON state** of this actuator.
- 1829           • it appears that P1\_P102 plays no active role in the system as its state  
1830           remains constant at 1 throughout. Considering this information, it  
1831           seems unlikely that P1\_P102 could serve as a relative setpoint. There-  
1832           fore, according to the brief analysis, it can be inferred that P1\_P102 is  
1833           most likely a **spare actuator** that is not actively utilized in the sys-  
1834           tem. A *spare actuator* is indeed a secondary actuator that is intended  
1835           to remain idle unless needed as a backup or replacement when the  
1836           primary actuator is unavailable or needs to be taken out of service.

1837           As the reader has probably already noticed, the majority of the graphs  
1838           presented in the previous sections and chapters were generated using the  
1839           new graph analysis script, specifically the `runChartsSubPlots.py` script.  
1840           This Python script is located in the `$(project-dir)/statistical-graphs`

1841 directory and utilizes the *matplotlib* libraries to generate the graph plots,  
1842 similar to the previous tool.

1843 The script accepts the following command-line parameters:

- 1844 • **-f or --filename:** specifies the CVS format dataset to read data from.  
1845 The dataset must be within the directory containing the enriched  
1846 datasets for the invariant analysis phase. If subsequent parameters  
1847 are not specified, the script will display all registers in the dataset,  
1848 excluding any additional attributes;
- 1849 • **-r or --registers:** specifies one or more specific registers to be dis-  
1850 played;
- 1851 • **-a or --addregisters:** adds one or more registers to the default visual-  
1852 ization. This option is useful in case an additional attribute such as  
1853 slope is to be analyzed;
- 1854 • **-e or --excluderegisters:** excludes one or more specific registers from  
1855 the default visualization. This option is useful to avoid displaying  
1856 hardcoded setpoints or spare registers.

1857 This script, like the previous ones, is designed to provide the maximum  
1858 flexibility and ease of use for the user, combined with greater power and  
1859 effectiveness in deriving useful information about the analyzed system.

1860 **Statistical Analysis** After careful consideration, we made the decision  
1861 not to include the statistical analysis aspect of the previous tool in the  
1862 framework. We found that there was no practical use for it. Instead, we  
1863 integrated the relevant statistical information into the brief analysis con-  
1864 ducted after the pre-processing phase. Additionally, we deemed the his-  
1865 togram to have limited utility and considered it outdated in comparison  
1866 to the new graph analysis approach we implemented.

1867 Although we decided not to include the statistical analysis aspect in the  
1868 framework, the Python script `histPlots_Stats.py` from the original tool

1869 remains in the directory. This script is essentially unchanged from the ver-  
1870 sion developed by Ceccato et al. and can be used in the future if the need  
1871 arises.

### 1872 4.2.3 Phase 3: Invariant Inference and Analysis

1873 The phase of invariant inference and analysis has undergone a redesign  
1874 and improvement to offer the user a more comprehensive and easier ap-  
1875 proach to identify invariants. This has been achieved through the applica-  
1876 tion of new criteria to analyze and reorganize the Daikon analysis results.  
1877 The outcome of this is a more compact presentation of information that  
1878 highlights the possible relationships among invariants.  
1879 The new design not only enables the identification of undiscovered aspects  
1880 of the system behavior but also confirms the hypotheses made during the  
1881 earlier stages of analysis. This step is now semi-automated, unlike before,  
1882 and allows for the analysis of invariants on individual actuator states and  
1883 their combinations.

#### 1884 4.2.3.1 Revised Daikon Output

1885 To streamline the process of identifying invariants quickly and effi-  
1886 ciently, it is necessary to revise the output generated by standard Daikon  
1887 analysis. The goal is to create a more compact and readable format for the  
1888 output.

1889

1890 The current Daikon results basically consist of three sections, referred as  
1891 *program point sections* [64]:

- 1892 1. the first section containing generic invariants, i.e., valid regardless of  
1893 whether a condition is specified for the analysis;
- 1894 2. the second section containing invariants obtained by specifying a  
1895 condition for the analysis in the *.spinfo* file, if any;

1896     3. a third section containing the invariants that are obtained from the  
1897       negation of the condition potentially specified in the *.spinfo* file.

1898     In each section only a single invariant per row is shown, without relat-  
1899       ing it in any way to the others: this makes it difficult to identify significant  
1900       invariants and any invariant chain that might provide much more infor-  
1901       mation about the behavior of the system than the single invariant.

1902     A brief example of the structure and format of this output related to PLC1  
1903       of the iTrust SWaT system is shown in Listing 4.5, where a condition was  
1904       specified on the measurement P1\_LIT101 and on actuator P1\_MV101:

```
1905    aprogram.point:::POINT
1906     P1_P102 == prev_P1_P102
1907     P1_FIT101 >= 0.0
1908     P1_MV101 one of { 0.0, 1.0, 2.0 }
1909     P1_P101 one of { 1.0, 2.0 }
1910     P1_P102 == 1.0
1911     max_P1_LIT101 == 816.0
1912     min_P1_LIT101 == 489.0
1913     slope_P1_LIT101 one of { -1.0, 0.0, 1.0 }
1914     [...]
1915     P1_LIT101 > P1_MV101
1916     P1_LIT101 > P1_P101
1917     P1_LIT101 > P1_P102
1918     P1_LIT101 < max_P1_LIT101
1919     P1_LIT101 > min_P1_LIT101
1920     [...]
1921     P1_MV101 < min_P1_LIT101
1922     P1_MV101 < trend_P1_LIT101
1923     P1_P101 >= P1_P102
1924     P1_P101 < max_P1_LIT101
1925     [...]
1926     =====
1927     aprogram.point:::POINT;condition="P1_MV101 == 2.0 &&
1928     ↪ P1_LIT101 < max_P1_LIT101 - 16 && P1_LIT101 >
1929     ↪ min_P1_LIT101 + 15"
1930     P1_MV101 == prev_P1_MV101
1931     P1_P102 == slope_P1_LIT101
1932     P1_MV101 == 2.0
```

```

1933      P1_FIT101 > P1_MV101
1934      P1_FIT101 > P1_P101
1935      P1_FIT101 > P1_P102
1936      P1_FIT101 > prev_P1_P101
1937      P1_MV101 >= P1_P101
1938      P1_MV101 >= prev_P1_P101
1939      P1_P101 <= prev_P1_P101
1940      =====
1941      aprogram.point::POINT; condition="not(P1_MV101 == 2.0 &&
1942          ↪ P1_LIT101 < max_P1_LIT101 - 16 && P1_LIT101 >
1943          ↪ min_P1_LIT101 + 15)"
1944      P1_P101 >= prev_P1_P101
1945      Exiting Daikon.

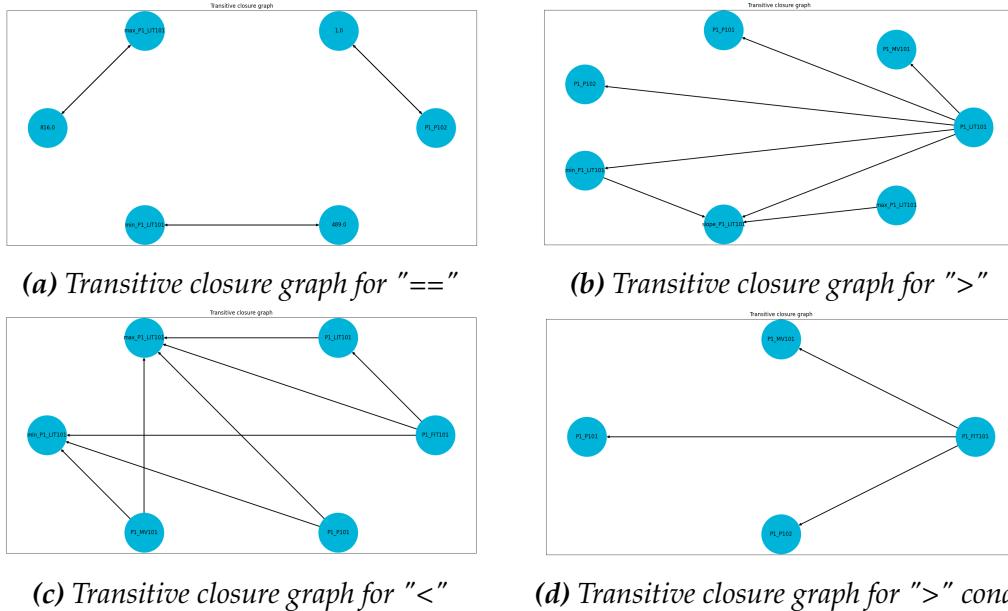
```

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

1946 In the presented framework, the output is simplified to **two sections**:  
1947 the *general section* and the section related to the *user-specified condition*. The  
1948 section related to the negated condition is eliminated as it is not relevant in  
1949 this context and could lead to potential misinterpretation. Moreover, the  
1950 relationships between invariants will be emphasized by utilizing **transi-**  
1951 **tive closures**: transitive closure of a relation  $R$  is another relation, typically  
1952 denoted  $R^+$  that adds to  $R$  all those elements that, while not necessarily  
1953 related directly to each other, can be reached by a *chain* of elements related  
1954 to each other. In other words, the transitive closure of  $R$  is the smallest (in  
1955 set theory sense) transitive relation  $R$  such that  $R \subset R^+$  [65].

1956 To implement the transitive closure of the invariants generated by Daikon,  
1957 we initially categorized the invariants in each section by type based on  
1958 their mathematical relation ( $==$ ,  $>$ ,  $<$ ,  $>=$ ,  $<=$ ,  $!=$ ), excluding any in-  
1959 variant related to additional attributes except for the slope. For each type  
1960 of invariant, we constructed a **graph** using the NetworkX library, where  
1961 registers were represented as nodes, and arcs were created to connect reg-  
1962 isters that shared a common endpoint in the invariant, applying the transi-  
1963 tive property. To reconstruct the individual invariant chains, we employed  
1964 a straightforward approach known as *Depth-first Search* (DFS) on each of  
1965 the graphs. This method allowed us to traverse the graphs and obtain the

1966 desired outcome, identifying the invariant chains. Figure 4.7 shows some  
 1967 examples of these graphs:



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

1968 At the end of this process, still applied to PLC1 of the iTrust SWaT sys-  
 1969 tem and with the same analysis condition, we get the following complete  
 1970 output:

```

1971 1 =====
1972 2 Generic
1973 3 =====
1974 4 P1_MV101 one of { 0.0, 1.0, 2.0 }
1975 5 P1_P101 one of { 1.0, 2.0 }
1976 6 slope_P1_LIT101 one of { -1.0, 0.0, 1.0 }
1977 7 P1_FIT101 != P1_P101, P1_P102
1978 8 P1_P102 == 1.0
1979 9 max_P1_LIT101 == 816.0
1980 10 min_P1_LIT101 == 489.0
1981 11 P1_LIT101 > P1_MV101
1982 12 P1_LIT101 > P1_P101
1983 13 P1_LIT101 > P1_P102
  
```

```

1984 14 P1_LIT101 > min_P1_LIT101 > slope_P1_LIT101
1985 15 P1_FIT101 >= 0.0
1986 16 P1_P101 >= P1_P102 >= slope_P1_LIT101
1987 17 =====
1988 18 =====
1989 19 P1_MV101 == 2.0 && P1_LIT101 < max_P1_LIT101 - 16 &&
1990 20   ↢ P1_LIT101 > min_P1_LIT101 + 15
1991 21 =====
1992 22 slope_P1_LIT101 == P1_P102
1993 23 P1_MV101 == 2.0
1994 24 P1_FIT101 > P1_MV101
1995 25 P1_FIT101 > P1_P101
1996 26 P1_FIT101 > P1_P102
1997 27 P1_MV101 >= P1_P101

```

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

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

#### 4.2.3.2 Types of Analysis

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

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

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

2015 ators, this semi-automated analysis offers a more accurate assessment of  
2016 the system behavior.

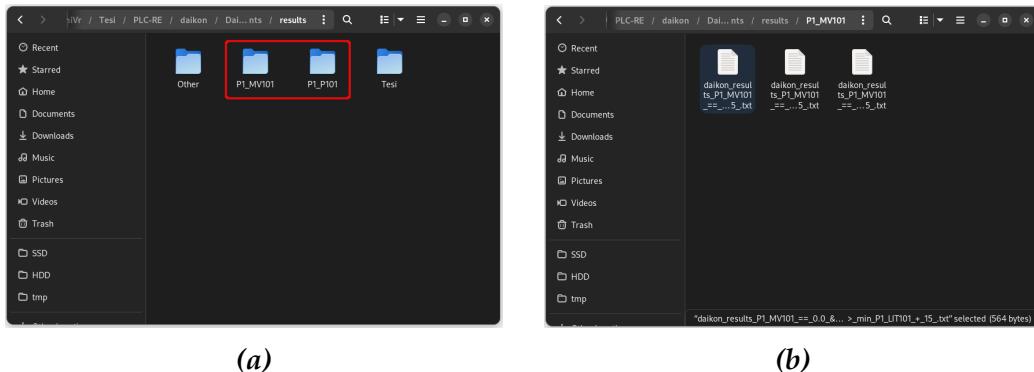
2017 These two types of analysis will be handled by the Python script  
2018 `daikonAnalysis.py`, contained in the default directory `$(project_dir)/daikon`.  
2019 The script accepts the following command-line arguments:

- 2020 • **-f or --filename:** specifies the enriched dataset, in CSV format, from  
2021 which to read the data. The dataset must be located within the di-  
2022 rectory containing the enriched datasets specified in the `config.ini` file  
2023 (by default `$(project_dir)/daikon/Daikon_Invariants`);
- 2024 • **-s or --simpleanalysis:** performs the analysis on the states of indi-  
2025 vidual actuators;
- 2026 • **-c or --customanalysis:** performs the analysis on combinations of ac-  
2027 tual states of the actuators;
- 2028 • **-u or --uppermargin:** defines a percentage margin on the maximum  
2029 value of the measurement;
- 2030 • **-l or --lowermargin:** defines a percentage margin on the minimum  
2031 value of the measurement;

2032 The selection of one or both types of analysis is possible. Additionally,  
2033 the last two parameters can be used to set a condition on the value of the  
2034 measurement. This condition is designed to bypass the transient periods  
2035 that occur during actuator state changes and the actual trend changes at  
2036 the maximum and minimum values of the measurement.

2037 This approach proves particularly beneficial for the first type of analysis,  
2038 as it enables more accurate data on the trends of the measurement. By  
2039 excluding the transient periods, the analysis can focus on the stable and  
2040 meaningful trends, providing improved insights into the behavior of the  
2041 system.

- 2042 **Analysis on single actuator states** Analysis on the states of individual  
 2043 actuators is the simplest: after the user is prompted to input the measure-  
 2044 ment, chosen from a list of likely available measurements, the script rec-  
 2045ognizes the likely actuators and the relative states of each, using the same  
 2046 mixed Daikon/Pandas technique adopted in the brief analysis during the  
 2047 pre-processing phase.
- 2048 For each actuator and each state it assumes, a single Daikon analysis is  
 2049 performed, eventually placing the condition on the maximum and mini-  
 2050 mum level of the measurement.
- 2051 The result of these analyses are saved in the form of text files in a directory  
 2052 having the name corresponding to the analyzed actuator and contained in  
 2053 the default parent directory \$(project\_dir)/daikon/Daikon\_Invariants/results:  
 2054 each file generated by the analysis is identified by the name of the actuator,  
 2055 the state and the condition, if any, on the measurement (see Figure 4.8).



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

2056 Listing 4.6 provides an illustrative example of the analysis results. In  
 2057 this case, we focus on the actuator P1\_MV101 in state 2. By examining the  
 2058 generic invariants, we can deduce that P1\_P101 is a probable actuator with  
 2059 binary values, as indicated in line 5. However, upon closer inspection  
 2060 in line 8, we discover that P1\_P102 consistently maintains a value of 1  
 2061 throughout, thus confirming its role as a spare actuator rather than a set-  
 2062 point. Furthermore, lines 9 and 10 present the maximum and minimum  
 2063 values achieved by P1\_LIT101, the register presumed to be connected to

2064 the tank.

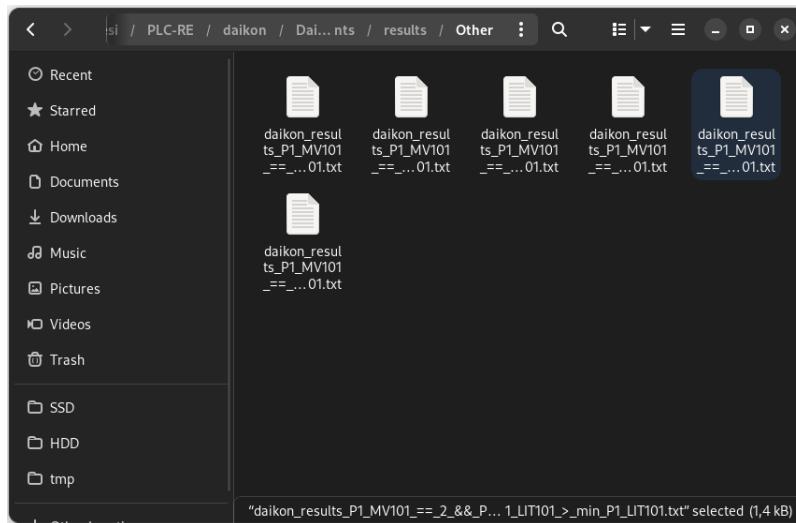
2065 The condition-generated invariants provide the most interesting insights.

2066 In particular, at line 21, we observe that when P1\_MV101 is set to 2, the  
2067 slope of P1\_LIT101 is 1, indicating an increasing trend. This confirms our  
2068 previous assumption that P1\_MV101 represents the actuator's ON state and  
2069 is responsible for filling the tank (P1\_LIT101).

2070 Furthermore, at line 23, we discover that P1\_FIT101 is greater than P1\_MV101.

2071 This implies that when the actuator assumes the value 2, the sensor P1\_FIT101  
2072 registers a measurement. In contrast, when P1\_MV101 is 1, P1\_FIT101 re-  
2073 mains at 0, signifying no recorded readings. This finding reinforces our  
2074 initial hypothesis that P1\_FIT101 may serve as a pressure or flow sensor.

2075 **Analysis of the Current System Configuration** The analysis of the cur-  
2076 rent system configuration based on the actual states of the actuators is  
2077 more complex, but at the same time offers more interesting outcomes as  
2078 it provides better evidence of actuator behavior in relation to the selected  
2079 measurement.



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

2080 For this analysis, the script automatically identifies the configurations  
2081 of the actuators that represent different system states (e.g., P1\_MV101 ==

2082 2, P1\_P101 == 1). Daikon analysis is then performed for each of these  
 2083 configurations. The user is prompted to select the measurement attribute  
 2084 and, if desired, specific actuators for studying their configurations. If no  
 2085 actuators are selected, all previously detected actuators will be considered.  
 2086 The analysis results are saved in text format within a designated directory,  
 2087 located alongside the previous analysis outputs. The filenames of these  
 2088 result files follow a specific naming convention based on the analysis rule  
 2089 applied (see Figure 4.9).

2090

2091 An example of the obtained outcomes can be seen in Listing 4.7:

```

2092 1 =====
2093 2 Generic
2094 3 =====
2095 4 P1_MV101 <= P1_P101 ==> P1_FIT101 >= 0.0
2096 5 P1_MV101 <= P1_P101 ==> P1_MV101 one of { 0.0, 1.0, 2.0
2097 6   ↪ }
2098 7 P1_MV101 <= P1_P101 ==> P1_P101 one of { 1.0, 2.0 }
2099 8 P1_MV101 <= P1_P101 ==> slope_P1_LIT101 one of { -1.0,
2100 9   ↪ 0.0, 1.0 }
2101 10 P1_MV101 > P1_P101 ==> P1_FIT101 > P1_MV101
2102 11 P1_MV101 > P1_P101 ==> P1_FIT101 > P1_P101
2103 12 P1_MV101 > P1_P101 ==> P1_FIT101 > P1_P102
2104 13 P1_MV101 > P1_P101 ==> P1_FIT101 > slope_P1_LIT101
2105 14 P1_MV101 > P1_P101 ==> P1_MV101 == 2.0
2106 15 P1_MV101 > P1_P101 ==> P1_MV101 > P1_P102
2107 16 P1_MV101 > P1_P101 ==> P1_MV101 > slope_P1_LIT101
2108 17 P1_MV101 > P1_P101 ==> P1_P101 == 1.0
2109 18 P1_MV101 > P1_P101 ==> P1_P101 == P1_P102
2110 19 P1_MV101 > P1_P101 ==> P1_P101 == slope_P1_LIT101
2111 20 P1_MV101 > P1_P101 ==> slope_P1_LIT101 == 1.0
2112 21 [...]
2113 22 =====
2114 23 P1_MV101 == 2 && P1_P101 == 1 && P1_LIT101 < max_P1_LIT101
2115 24   ↪ && P1_LIT101 > min_P1_LIT101
2116 25 =====
2117 26 slope_P1_LIT101 == P1_P102 == P1_P101 == 1.0

```

```
2119 25 P1_MV101 == 2.0  
2120 26 P1_FIT101 > P1_MV101  
2121 27 P1_FIT101 > P1_P101
```

***Listing 4.7:** Daikon outcomes for the system configuration  $P1\_MV101 == 2$ ,  $P1\_P101 == 1$  on  $P1\_LIT101$*

2122 In contrast to Listing 4.6, the current analysis reveals the presence of im-  
2123 plications in the general invariants section, which were previously absent  
2124 (remaining generic invariants omitted for brevity). These implications of-  
2125 fer valuable insights, as demonstrated in this case. For instance, the invari-  
2126 ant stated in line 18 informs us that if the value of  $P1\_MV101$  is greater than  
2127  $P1\_P101$ , then the slope's value is 1, indicating an increasing tank level.  
2128 This finding further corroborates our initial understanding that the state  
2129  $P1\_MV101 == 2$  represents the ON state for the actuator responsible for fill-  
2130 ing the tank, namely  $P1\_LIT101$ . Upon comparing the results of the other  
2131 analyses, we will uncover that  $P1\_P101$  serves the purpose of emptying the  
2132 tank, with its ON and OFF states denoted as 2 and 1, respectively.  
2133 Furthermore, the invariant presented in line 8 reveals that when  $P1\_MV101$   
2134 is in state 2 and  $P1\_P101$  is in state 1, the value of  $P1\_FIT101$  is greater  
2135 than 2. Consequently, it can be inferred that the associated sensor is mea-  
2136 suring something in relation to the tank represented by  $P1\_LIT101$ .  
2137 The aforementioned observations are further supported by the invariants  
2138 associated with the analysis condition. As indicated in line 24, the slope is  
2139 indeed equal to 1, confirming an upward trend. Additionally, at line 26, it  
2140 is evident that  $P1\_FIT101$  assumes values greater than 2 when  $P1\_MV101$  is  
2141 equal to 2.

2142 **Refining the Analysis** In certain situations, the outcomes provided by  
2143 the semi-automated analyses may not meet the user's expectations. For  
2144 instance, the clarity of the slope value may be insufficient, or the user may  
2145 wish to delve deeper into a specific aspect of the system to uncover ad-  
2146 dditional invariants that were not previously identified. In such cases, the  
2147 user has the option to conduct a more targeted and specific invariant anal-  
2148 ysis using the Python script `runDaikon.py`, which enables precise investi-

2149 gations of the system.

2150 The script, located in the default directory \$(project\_dir)/daikon, can be  
2151 executed with three command-line parameters:

2152 • **-f or --filename:** specifies the CSV format *enriched* dataset to read  
2153 data from. Even in this case, the dataset must be located within the  
2154 directory containing the enriched datasets;

2155 • **-c or --condition:** specifies the condition for the analysis, which will  
2156 be automatically overwritten in the *.sinfo* file. It is possible to spec-  
2157 ify more than one condition, but it is strongly recommended to use  
2158 the logical operator && to avoid undesired behaviors in Daikon out-  
2159 comes;

2160 • **-r or --register:** specifies the directory where to save the text file with  
2161 the outcomes.

2162 During the execution of this analysis, a single output file is generated, con-  
2163 taining the discovered invariants. The user can specify the directory where  
2164 this file should be saved using the -r command-line option. By default, the  
2165 file is stored in the directory \$(project\_dir)/daikon/Daikon\_Invariants/results.  
2166 The output file serves as a record of the identified invariants and can be  
2167 examined at a later time.

2168 In conclusion, the integration of these two analysis types, along with  
2169 the ability to conduct more refined analysis in the future and the enhanced  
2170 output format of Daikon, significantly enhances the completeness, clarity,  
2171 and effectiveness of this stage compared to the previous framework.

#### 2172 4.2.4 Phase 4: Business Process Analysis

#### 2173 4.2.5 Phase 5: Network Analysis

## Case study: the iTrust SWaT System

2174 HAVING introduced the innovative framework and highlighted its po-  
2175 tential in the preceding chapter, we now turn our attention to the  
2176 case study where we will apply this framework. As previously mentioned  
2177 in Chapter 4 and demonstrated through various examples in the same  
2178 chapter, our focus will be on the **iTrust SWaT system** [36], developed by  
2179 the iTrust – Center for Research in Cyber Security of University of Singa-  
2180 pore for Technology and Design [30]. The acronym SWaT represents *Secure*  
2181 *Water Treatment*.

2182 The iTrust SWaT system is a testbed that replicates on a small scale  
2183 a real water treatment plant arises to support research in the area of cy-  
2184 ber security of industrial control systems and has been operational since  
2185 March 2015: it is still being used by students at the University of Singapore  
2186 for educational and training purposes and is available to organizations to  
2187 train their operators on cyber physical incidents.

### 2188 5.1 Architecture

2189 In contrast to the virtualized testbed discussed in Section 3.2.1 by Cec-  
2190 cato et al., the iTrust SWaT system is composed entirely of physical hard-  
2191 ware components. It encompasses various elements, starting from field

2192 devices and extending to PLCs, HMI, SCADA workstations, and the SCADA  
2193 server (also referred to as the *historian*). The historian is responsible for  
2194 recording data from field devices for further analysis. In the upcoming  
2195 sections, we will delve deeper into the architecture of the physical process  
2196 and the communication network.

2197 **5.1.1 Physical Process**

2198 The physical process of the SWaT consists of six stages, denoted P1  
2199 through P6. These stages are [66][67]:

- 2200 1. **taking in raw water:** feeds unfiltered water into the system
- 2201 2. **chemical dosing:** adds chemicals to water useful for initial pretreat-  
2202 ment
- 2203 3. **Ultra Filtration (UF) system:** the water is filtered through a semi-  
2204 permeable membrane (ultrafiltration membrane) using the liquid pres-  
2205 sure, effectively capturing impurities and suspended solids, as well  
2206 as removing bacteria, viruses, and other pathogens present in the  
2207 water.
- 2208 4. **dechlorination:** removes residual chlorine from disinfected water  
2209 using ultraviolet lamps
- 2210 5. **Reverse Osmosis (RO):** performs further filtration of the water
- 2211 6. **backwash process:** cleans the membranes in UF using the water pro-  
2212 duced by RO

2213 Figure 5.1 shows a graphical representation of the architecture and the six  
2214 stages of the SWaT system.

2215 The SWaT system incorporates an array of sensors that play a crucial  
2216 role in monitoring the system's operations and ensuring their safe. These  
2217 sensors are responsible for continuously collecting data and providing  
2218 valuable insights into the functioning of the system. These sensors are:

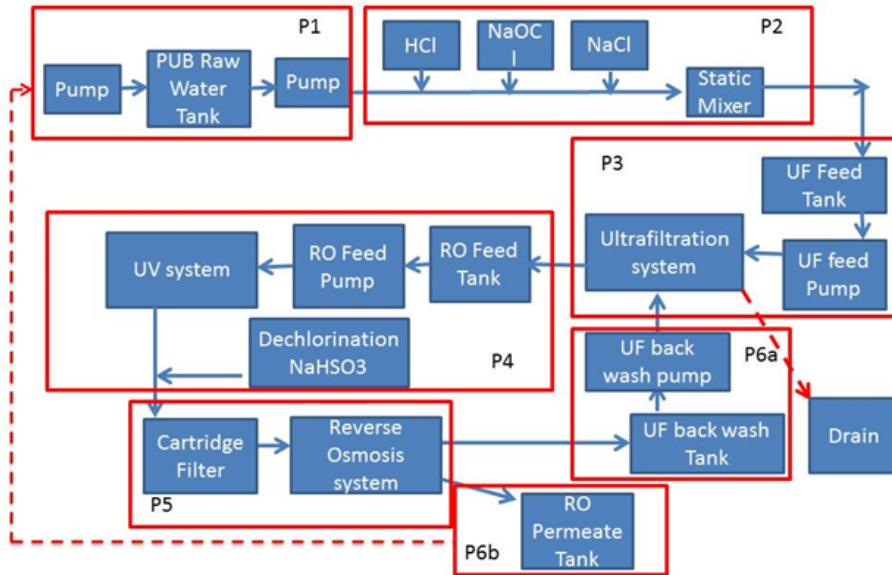


Figure 5.1: SWaT architecture

- 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)

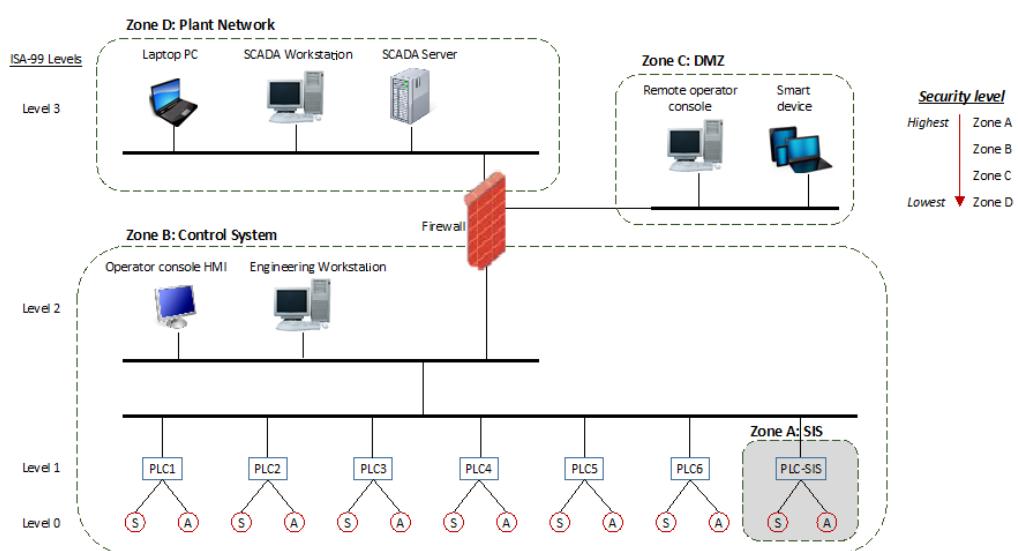
PLCs at Layer 1 communicate with their respective sensors and actuators at Layer 0 through a conventional ring network topology based on EtherNet/IP, to ensure that the system can tolerate the loss of a single link

2250 without any adverse impact on data or control functionality.  
 2251 PLCs between the different process stages at Layer 1 communicate with  
 2252 each other through a star network topology using the CIP protocol on Eth-  
 2253 erNet/IP, previously discussed in Section 2.2.6.3.

2254 Regarding zoning, the SWaT system is divided into three zones, each  
 2255 containing one or more layers. These zones are, in descending order of  
 2256 security level:

- 2257 • **Plant Control Network, or Control System:** includes layers from 0  
 2258 to 2
- 2259 • **DMZ:** includes Layer 3.5
- 2260 • **Plant Network:** includes Layer 3

2261 Figure 5.3 provides a clearer visualization of the zoning and layer division  
 2262 within the network architecture of the SWaT system. This diagram high-  
 2263 lights the distinct zones and their corresponding layers, offering a com-  
 2264 prehensive overview of the system's network structure.



*Figure 5.3: SWaT network architecture*

2265 A specific IP address is associated with each device: in Table 5.1 we  
2266 report the addresses for the PLCs, historian, and Touch Panel in the *Plant*  
2267 *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*

## 2268 5.2 Datasets

2269 To facilitate the study and testing of technologies related to cyber security  
2270 in Industrial Control Systems and critical infrastructure, iTrust offers  
2271 researchers worldwide the opportunity to access a range of datasets [68].  
2272 These datasets consist of a collection of data obtained from the SWaT sys-  
2273 tem, encompassing information on both the physical processes and net-  
2274 work communications. The data is organized into different years, and  
2275 researchers can request access to these datasets for their analysis and ex-  
2276 perimentation purposes.

2277 **Physical Process Datasets** The datasets containing information about  
2278 the physical processes are provided in CSV format files. These files en-  
2279 compass data collected during different time intervals, which can vary  
2280 from a few hours to entire days. The granularity of the data is typically

2281 at a one-second interval, although there may be some exceptions. The col-  
2282 lected data primarily consists of timestamped sensor measurements and  
2283 actuator status values for each PLC, describing the physical properties of  
2284 the testbed in operational mode.

2285 **Network Communications Datasets** Network communications are typi-  
2286 cally available in the form of *Packet Capture* format (PCAP) files. These files  
2287 contain captures of communication network traffic, allowing researchers  
2288 to analyze and examine the network interactions. In some instances, CSV  
2289 files are provided instead of PCAP files, featuring different characteristics  
2290 for the collected data.

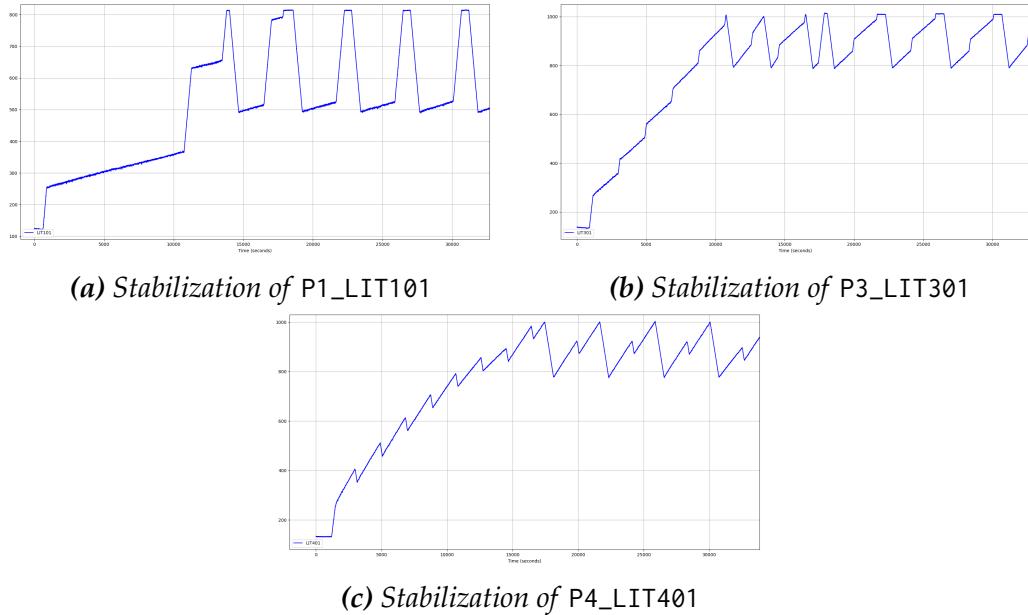
### 2291 5.2.1 Our Case Study: the 2015 Dataset

2292 The dataset selected as a case study to apply the framework discussed  
2293 in the previous chapter is specifically the dataset from the year 2015 [69].  
2294 The main reason for this choice is the unique characteristics found in the  
2295 physical process dataset that are not present in datasets from subsequent  
2296 years.

2297 **Physical Process Data** The data collection process lasted 11 consecutive  
2298 days, 24 hours per day. During the first 7 days, the system operated nor-  
2299 mally without any recorded attacks. However, attacks were observed dur-  
2300 ing the remaining 4 days. The collected data reflects the impact of these  
2301 attacks, leading to the creation of two separate CSV files: one containing  
2302 the recorded data of SWaT during the system's regular operations, and  
2303 the other containing data recorded during the days of the attacks. To en-  
2304 sure accurate information about the system, the dataset pertaining to the  
2305 normal operations, which spans seven days, was chosen for analysis.

2306 Data collection occurs at a frequency of one data point per second,  
2307 with the assumption that significant attacks cannot occur within a shorter  
2308 time frame. Additionally, the firmware of the PLCs remains unchanged  
2309 throughout the data collection period.

2310 At the beginning of data gathering the tanks are empty and the system  
 2311 must be initialized in order to then reach full operation: it typically takes  
 2312 around five hours for all tanks to be fully filled and for the system to sta-  
 2313 bilize and reach the appropriate operational state (see Figure 5.4).



*Figure 5.4: SWaT stabilization*

2314 In total, the dataset consists of thousand and thousand of samples re-  
 2315 lated to 51 attributes.

2316 **Network Traffic** The network traffic was collected using an appliance  
 2317 from a well-known network hardware manufacturer and was made avail-  
 2318 able only in CSV format and not PCAP format. Table 5.2 shows some of  
 2319 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*

2320        Unfortunately, the data provided are partial as it only includes read-  
 2321        ings from sensors and does not include information on actuator status.

2322  
 2323        Do not mislead by the word *Modbus* in the captured data fields: this is  
 2324        most likely a feature of the sniffing appliance, which may have encapsu-  
 2325        lated EtherNet/IP-CIP protocol packets in Modbus frames, as described  
 2326        by T. A. Snide in [70].

2327        To provide evidence that the protocol being used is CIP over EtherNet/IP,  
 2328        we can examine the "Source / Destination port" field in the captured data.  
 2329        The presence of TCP port 44818 as the destination port aligns with the  
 2330        TCP port used for transporting **explicit messages** in the EtherNet/IP pro-  
 2331        tocol, as explained in Section 2.2.6.2. Additionally, the "Application name",  
 2332        "Modbus Function Code" and "Modbus Function Description" fields fur-  
 2333        ther support this conclusion. The "Application name" explicitly states  
 2334        "CIP\_read\_tag\_service," while the "Modbus Function Code" field contains  
 2335        a decimal value of 76, which does not correspond to any known Modbus  
 2336        Function Calls. When converted to hexadecimal, this value is 4C, match-  
 2337        ing the CIP protocol read tag request code as indicated in the "Modbus  
 2338        Function Description" field.

2339        In summary, the datasets for subsequent years were not selected due

2340 to certain limitations. In the case of the year 2017, the physical system  
2341 data had a wide granularity, resulting in significant information loss. Ad-  
2342 ditionally, in some cases, the datasets were considered "spurious" as there  
2343 was no clear differentiation between data obtained during normal SWaT  
2344 operations and data associated with attacks.

2345 Regarding network traffic data, there were instances where the data was  
2346 either missing or fragmented into large PCAP files weighing several gi-  
2347 gabytes (more than 6 GB each!), despite containing only a few minutes'  
2348 worth of data. This made it impractical to manage such files given the  
2349 available resources.

2350 Despite their large quantity, the CSV files associated with network traffic  
2351 for the year 2015 are comparatively smaller in size, just slightly over 100  
2352 MB each. These files cover a more extensive time period, which facilitates  
2353 easier management and analysis. Prioritizing the physical process dataset  
2354 as crucial, I have selected the year 2015 as a case study, even though the  
2355 network data may be incomplete.

## Our framework at work: reverse engineering of the SWaT system

### 6.1 Pre-processing

### 6.2 Graph Analysis

#### 6.2.1 Conjectures About the System

### 6.3 Invariants Analysis

#### 6.3.1 Actuators Detection

#### 6.3.2 Daikon Analysis and Results Comparing

### 6.4 Extra information on the Physics

### 6.5 Business Process Analysis



Chapter

# 7

## Conclusions

### 7.1 Discussions

### 7.2 Guidelines

### 7.3 Future work



## List of Figures

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

3.8	Behavior of the Graph Analysis on the Ceccato et al.'s tool . . . . .	49
3.9	Histogram plot overshadowing statistical information shown on the terminal window in the background . . . . .	50
3.10	Example of Daikon's output . . . . .	52
4.1	Slope comparison with granularity 30 (a), 60 (b) and 120 seconds (c) . . . . .	68
4.2	Savitzky-Golay filter (blue line), STL decomposition (green) and RDP algorithm (orange) comparison . . . . .	70
4.3	Slope after the application of the STL decomposition . . . . .	71
4.4	The new slope representation (green line) and the smoothed measurement data obtaind with the STL decomposition (red) . . . . .	72
4.5	Plotting registers on the same y-axis . . . . .	77
4.6	Example of the new graph analysis . . . . .	78
4.7	Example of transitive closure graphs for invariants in PLC1 of iTrust SWaT system . . . . .	85
4.8	Directory (a) and outcome files (b) for the single actuator states analysis . . . . .	88
4.9	Daikon outcome files for system configuration analysis. Each file represents a single system state . . . . .	89
5.1	SWaT architecture . . . . .	95
5.2	Sensors and actuators associated with each PLC . . . . .	96
5.3	SWaT network architecture . . . . .	97
5.4	SWaT stabilization . . . . .	100

## List of Tables

2.1	differences between Information Technology (IT) and Industrial Control Systems (ICSs)	4
2.2	Modbus Function Codes list	18
3.1	Summary table of Ceccato et al. framework limitations	54
5.1	main IP addresses of the six PLCs and SCADA in SWaT	98
5.2	SWaT network traffic data	101



## Listings

3.1	Example of registers capture . . . . .	31
3.2	Example of raw network capture . . . . .	32
3.3	Statistical data for PLC1_InputRegisters_IW0 register . . . . .	34
3.4	Generic example of a .spinfo file for customizing rules in Daikon . . . . .	35
3.5	The three sections of Daikon analysis outcomes . . . . .	35
4.1	Novel Framework structure and Python scripts . . . . .	58
4.2	Paths and parameters for the Pre-processing phase in <i>config.ini</i> file . . . . .	62
4.3	<i>config.ini</i> parameters for dataset enriching . . . . .	64
4.4	Example of preliminar system analysis . . . . .	73
4.5	Standard Daikon output for PLC1 of the iTrust SWaT system	83
4.6	Revised Daikon output with transitive closures for PLC1 of the iTrust SWaT system . . . . .	85
4.7	Daikon outcomes for the system configuration P1_MV101 == 2, P1_P101 == 1 on P1_LIT101 . . . . .	90



## References

- [1] NIST. *ICS defintion*. URL: [https://csrc.nist.gov/glossary/term/industrial\\_control\\_system](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-%E2%80%93Douglas%E2%80%93Peucker\\_algorithm](https://en.wikipedia.org/wiki/Ramer-%E2%80%93Douglas%E2%80%93Peucker_algorithm) (visited on 2023-05-21).
- [64] *The Daikon Invariant Detector User Manual*. URL: <https://plse.cs.washington.edu/daikon/download/doc/daikon.html#Program-point-sections> (visited on 2023-05-25).
- [65] Wikipedia. *Transitive closures*. URL: [https://en.wikipedia.org/wiki/Transitive\\_closure](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%20Modbus%20Integration%20Hanover%20Fair\\_0408.pdf](https://www.modbus.org/docs/CIP%20Modbus%20Integration%20Hanover%20Fair_0408.pdf) (visited on 2023-05-17).