Czech Technical University in Prague
Faculty of Electrical Engineering

**Department of Measurement**
**Master's programme: Cybernetics and Robotics**

# Testing and Interfacing Environment for Point Machine Simulators

# Prostředí pro testování a připojení simulátorů výhybek

MASTER'S THESIS

Author: Bc. Alisa Pavlova
Supervisor: Prof. Ing. Jan Holub, Ph.D.

August 2022

# MASTER‘S THESIS ASSIGNMENT

## I. Personal and study details

Student's name: **Pavlova Alisa** Personal ID number: **453105**

Faculty / Institute: **Faculty of Electrical Engineering**

Department / Institute: **Department of Measurement**

Study program: **Cybernetics and Robotics**

Branch of study: **Cybernetics and Robotics**

## II. Master’s thesis details

Master’s thesis title in English:

**Testing and Interfacing Environment for Point Machine Simulators**

Master’s thesis title in Czech:

**Prostředí pro testování a připojení simulátoru výhybek**

Guidelines:

Develop and implement PC SW application (GUI, backend) for controlling of various point machine simulators and data logging from interfaces of various point machine simulators. Perform testing of this application as an example of EN50128 development (functional testing, static analysis, code coverage). The application shall be compatible with Windows OS, Linux OS compatibility is optional. Perform factory (assembly) testing of the 7-wire “point machine simulators” and document on this example the product development process according to EN 50129 (requirements definition, requirement tests).

Bibliography / sources:

[1] - CENELEC standard EN 50128: Railway applications – Software for railway control and protection systems, 2011-06
[2] - CENELEC standard EN 50129: Railway applications - Communications, signaling and processing systems - Safety related electronic systems for signalling, 2018-11
[3] – Python manual, https://docs.python.org/3/

Name and workplace of master’s thesis supervisor:

**prof. Ing. Jan Holub, Ph.D. Department of Measurement FEE**

Name and workplace of second master’s thesis supervisor or consultant:

Date of master’s thesis assignment: **31.01.2022** Deadline for master's thesis submission: **15.08.2022**

Assignment valid until:
**by the end of summer semester 2022/2023**

| prof. Ing. Jan Holub, Ph.D. | Head of department’s signature | prof. Mgr. Petr Páta, Ph.D. |
|---|---|---|
| Supervisor’s signature | | Dean’s signature |

## III. Assignment receipt

The student acknowledges that the master’s thesis is an individual work. The student must produce her thesis without the assistance of others, with the exception of provided consultations. Within the master’s thesis, the author must state the names of consultants and include a list of references.

| . | |
|---|---|
| Date of assignment receipt | Student’s signature |

**Declaration**

I declare that the presented work was developed independently and that I have listed all sources of information used within it in accordance with the methodical instructions for observing the ethical principles in the preparation of university theses.

Prague, August 2022

.....
Bc. Alisa Pavlova

**Acknowledgement**

I would like to thank my supervisor Prof. Ing. Jan Holub, Ph.D., for his guidance throughout this project. I also would like to thank my family and partner for moral support and my colleagues for their professional recommendations.

Bc. Alisa Pavlova

**Testing and Interfacing Environment for Point Machine Simulators**

*Abstract:* The master's thesis deals with the following processes: development and implementation of a software application for controlling various point machine simulators and data logging from their interfaces; performance of the application testing according to *EN 50128*; performance of the seven-wire point machine simulator testing according to *EN 50129*. The thesis describes the creation of the application and its documentation (including a user manual) based on information from the simulators' manuals. The implemented application is then checked for problems and errors via testing according to the chosen methods from *EN 50128*. These methods, their techniques, and created testing software tool are also presented in the thesis. *EN 50129* is used to test the seven-wire point machine simulator and to document detected issues. The testing part of the thesis also contains a description of the testing hardware tool assembly. The final test results are analyzed and discussed. In conclusion, the project run is summarized, and future steps are defined.

*Key words:* CENELEC, EN 50128, EN 50129, development process, product requirements, verification

**Prostředí pro testování a připojení simulátorů výhybek**

*Abstrakt:* Tato diplomová práce se zabývá následujícími procesy: vývojem a implementací softwarové aplikace pro řízení různých simulátorů výhybek a logování dat z jejich rozhraní; provedením testování této aplikace podle *EN 50128*; provedením testování simulátoru výhybky se sedmi vodiči podle *EN 50129*. Práce popisuje tvorbu aplikace a dokumentace (včetně uživatelské příručky) na základě informací z manuálů k simulátorům. Implementovaná aplikace je následně zkontrolována na problémy a chyby s pomocí testovaní podle zvolených metod z *EN 50128*. Tyto metody, jejich techniky a vytvořený testovací softwarový nástroj jsou také prezentovány v práci. *EN 50129* se používá k testování simulátoru výhybky se sedmi vodiči a k dokumentování nalezených problémů. Testovací část práce obsahuje také popis sestavy testovacího hardwarového nástroje. Konečné výsledky testů jsou analyzovány a diskutovány. Na závěr je shrnut průběh projektu a jsou definovány budoucí kroky.

*Klíčová slova:* CENELEC, EN 50128, EN 50129, vývojový proces, požadavky na produkt, verifikace

# Contents

# List of Acronyms

| | | |
|---|---|---|
| **PMSs** | *Point Machine Simulators* | 1 |
| **PMs** | point machines | 1 |
| **GUI** | graphical user interface | 1 |
| **SW** | software | 1 |
| **HW** | hardware | 29 |
| **EP** | end position | 6 |
| **1P** | one phase | 5 |
| **3Ps** | three phases | 5 |
| **4Ws** | four wires | 5 |
| **7Ws** | seven wires | 5 |
| **TCP/IP** | Transmission Control Protocol/Internet Protocol | 17 |
| **I/O** | input/output | 6 |
| **EN** | European norms | 27 |
| **BVA** | **Boundary value analysis** | 28 |
| **ECIPT** | **equivalence class and input partition testing** | 28 |
| **PS** | **process simulation** | 29 |
| **TCC** | Test coverage for code | 29 |
| **MPD** | *Manual Point Driver* | 41 |
| **FEPL** | Force end position left | 7 |
| **FEPR** | Force end position right | 7 |
| **CVCU** | Control Voltage Connector Unplugging | 7 |
| **ELT** | Earth Leakage Test | 7 |
| **EPR** | End position right | 7 |
| **EPL** | End position left | 7 |
| **COVP** | Control voltage presence | 7 |
| **MRPO** | Moving right phase order | 7 |
| **MLPO** | Moving left phase order | 7 |
| **POS** | point operating system | 1 |
| **SWTI** | switching time | 7 |

# List of Figures

# List of Tables

# Introduction

The rail industry consists of manufacturing, selling, and operating railway technologies. The purpose of railway systems manufacturing is to create a reliable and high-quality product that would ensure flawless operation in the railway transport area. In order to achieve this purpose, there must be such steps as

- studies and design,
- hardware and software development,
- verification, tests, integration, and validation.

The master's thesis deals with the testing part of railway system manufacturing. More precisely, it works with components used for testing systems for controlling railway points and derailleurs. These components are *Point Machine Simulators* (PMSs), which perform the functions of real point machines (PMs). More details about them are in Chapter 1.

The goal of the thesis is to design and implement a software (SW) application with a graphical user interface (GUI) for controlling various PMSs and data logging from their interfaces. The proper functioning of the developed application must be verified with the help of the particular norm described in Section 4.1.1. Also, factory testing of one PMS type must be provided according to the norm described in Section 4.1.2

The master's thesis is divided into the following chapters:

- **Point operating system architecture**
  This chapter briefly describes the basic principles of point operating system (POS) functionality. This theory is needed for a better understanding of the master's thesis motivation.
- **Point Machine Simulator**
  The chapter contains specifications of PMSs that were used during work on the thesis.
- **Application design and development**
  This part of the thesis documents the development process of the application for interaction with PMSs.
- **System testing**
  This chapter describes the testing of the developed application and the PMS prototype. It also includes a description of the tools used during the testing.

# Chapter 1

# Point operating system

POS is a complex of devices intended for point blades control. The point blades (points) are mobile rails directing the railway transport towards the straight or the diverging track [1]. This chapter briefly describes the parts of POS driving the points.

## 1.1 Architecture

The devices composing POS are shown in Figure 1.1. The system consists of interlocking, point driver, and switch motors (PMs). The interlocking communicates and interchanges information with PMs through the point driver. Their cooperation ensures safe railway processes [2].

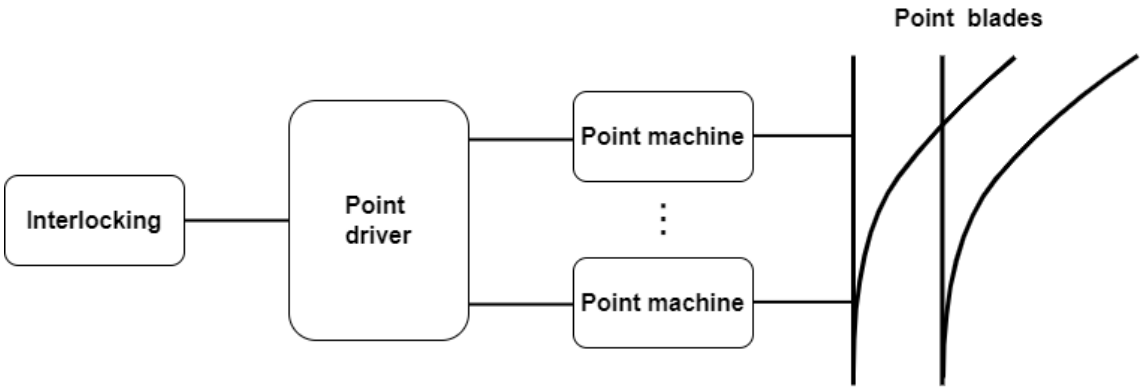



**Figure 1.1:** Point operating system

### 1.1.1 Interlocking

Railway interlocking serves to prevent conflicting or unsafe situations between trains by regulating such track devices as derailments, junctions, and crossings [3].

When the train receives a signal to move along the established route, all movable components on the train's path are set to the locked position. The components remain in this position until the train receives a signal to cancel the movement or until it leaves the area belonging to the route [3].

The aims of interlocking are the following [3]:

- to provide details about movement requests, velocities of trains, and workloads on railway sections;
- to provide and maintain trains routes safety;
- to make sure that the system will end in a safe state in case of failure.

### 1.1.2 Point driver

This part of POS serves as an intermediary between the interlocking and PMs. The driver collects and transmits information to control the points' behavior and ensure the system's safety [2].

### 1.1.3 Point machine

Effective throwing of points is needed to ensure operational safety and high train throughput. PMs are the way to achieve these goals. These devices provide the function of locking/unlocking and operating the point switches [4]. Figure 1.2 shows the *S 700 K* PM from Siemens Mobility s.r.o.

Current PMs contain an electric motor. Its torque is converted into the throwing force: the rotary motion is transformed into the linear motion necessary for the point switching [5].

**Figure 1.2:** S 700 K point machine [4]

# Chapter 2

# Point Machine Simulator

This chapter explains the principles of PMS functionality.

PMS is an internal Siemens product used for testing point controllers. It performs the functionality of PM: each PMS represents two independent PMs. Two PMS types are considered in this thesis:

- *2PMS-3P4W* with three phases and four wires (4Ws);
- *2PMS-1P3P7W* with one phase (1P)/three phases (3Ps) and seven wires (7Ws).

Their specifications are described in the following sections.

## 2.1 2PMS-3P4W

This section is based on the PMS documentation *2PMS-3P4W Design Specification and Manual* [6].

*2PMS-3P4W* is used for simulation of PMs with 3P and 4W [6]. Figure 2.1 represents what its front panel looks like.


**Figure 2.1:** *2PMS-3P4W* [6]

### 2.1.1 Specifications

This subsection contains specifications and characteristics of 3P 4W PMS.

### Functions & requirements

PMS has to provide the following functions [6]:

- simulation of throwing to the right/left end position (EP) depending on the phases' order;
- simulation of throwing to the reverse direction;
- simulation of the point trailed event;
- simulation of the situation, when the trailed point ends in the opposite direction;
- simulation of the situation, when the trailed point ends in the same direction after some delay;
- simulation of the point jammed event;
- simulation of the situation, when the jammed point starts to move after some delay;
- simulation of two independent points;
- remotely setting of throwing time;
- available communication via Modbus TCP;
- available 24V parallel input/output (I/O) interface.

### Position sensing

PM EP and the switching direction can be detected via special relays inside PMS. Table 2.1 shows the relation between relay states and indicated positions. 1 represents closed relay, and 0 represents opened relay [6].

**Table 2.1:** *2PMS-3P4W* indicated positions and corresponding relay states [6]

| **Position** | **L1-N** | **L2-N** | **L1-L3** | **L2-L3** |
|---|---|---|---|---|
| Unplugged | 0 | 0 | 0 | 0 |
| Middle | 0 | 0 | 1 | 1 |
| Left | 0 | 1 | 1 | 0 |
| Right | 1 | 0 | 0 | 1 |
| Invalid | 1 | 1 | 0 | 0 |

### Motor phase orders

- L1-L2-L3 - *Moving LEFT* [6].
- L2-L3-L2 - *Moving RIGHT* [6].

### Inputs and outputs

Each simulated PM operates with inputs and outputs described in this section. The tables below describe the inputs' functions: Table 2.3 contains definitions of 24 V pulse inputs reacting on rising edge; Table 2.2 has definitions of 24 V switch continuous inputs (their high level represents an active state) [6].

**Table 2.2:** *2PMS-3P4W* pulse inputs [6]

| Signal name | Description and restrictions |
|---|---|
| Force end position left (FEPL) | This input sets PMS EP to the left without delay. The control voltage must be disconnected. |
| Force end position right (FEPR) | This input sets PMS EP to the right without delay. The control voltage must be disconnected. |
| Trail | This input sets the PMS position to the middle without delay. The control voltage must be disconnected. |

**Table 2.3:** *2PMS-3P4W* switch inputs [6]

| Signal name | Description and restrictions |
|---|---|
| Earth Leakage Test (ELT) | This input simulates earth leakage. |
| Control Voltage Connector Unplugging (CVCU) | This input simulates unplugging the control voltage. The control voltage must be disconnected. Unplugging cannot be done while PM changes its position. |
| Jamm | This input prolongs the switching time. PMS remains in the middle position if the control voltage is not present. If it is, PMS is switched to the required EP. |

There is also the possibility of setting PMS time parameters [6]:

- switching time (SWTI);
- end position reaching delay (EPRED).

Their range is from 0 to 32767 ms [6].

The following list contains 24 V switch outputs of each PM [6]:

- End position right (EPR);
- End position left (EPL);
- Control voltage presence (COVP);
- Moving right phase order (MRPO);
- Moving left phase order (MLPO).

### Logic

This subsection describes the logic of 4W PMS, which is similar to the real PM behavior.

The diagram shown in Figure 2.2 demonstrates the basic switching logic. EP depends on the phase order [6]:

- L1-L2-L3 (MLPO) - *LEFT* EP;
- L1-L3-L2 (MRPO) - *RIGHT* EP.

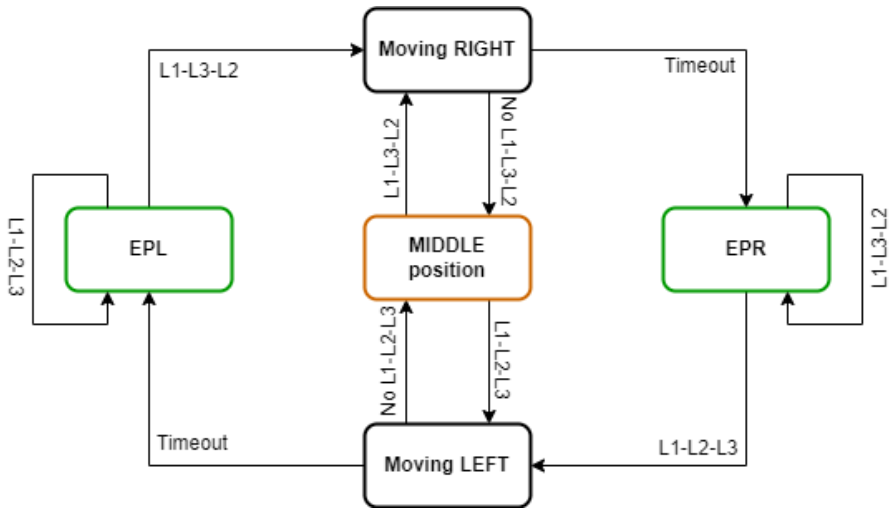



**Figure 2.2:** 3P switching process diagram (adapted from [7])

The switching diagram with simulated Jamming is in Figure 2.3. Figure 2.4 shows the PMS reaction to trailed pulse. Forcing EP process diagram is shown in Figure 2.5 [6]:

- EPL forces *LEFT* EP;
- EPR forces *RIGHT* EP.

Timeout on all the diagrams represents throwing time equal to the sum of SWTI and double EPRED [6].

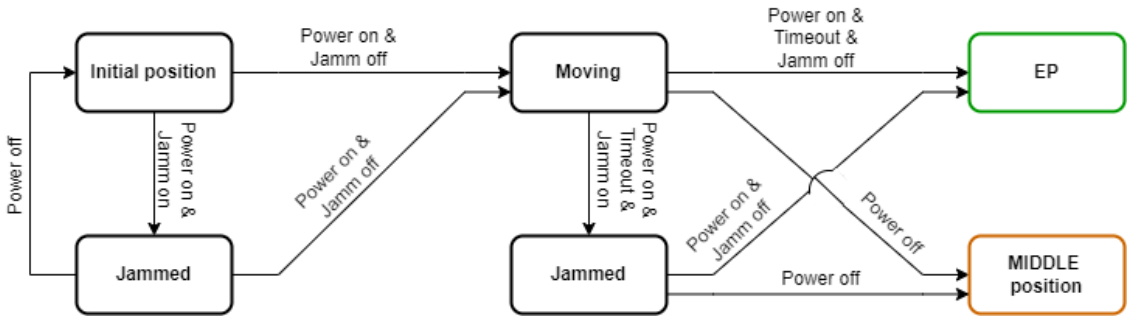



**Figure 2.3:** Jamming process diagram (adapted from [7])

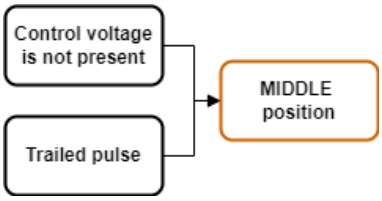



**Figure 2.4:** Trailing process diagram (adapted from [6])

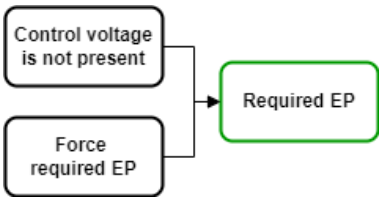



**Figure 2.5:** Forcing EP process diagram (adapted from [6])

### Interfaces

3P 4W PMS has three primary interfaces. They are described in the following list.

- **User interface**
  Figure 2.1 shows the front panel of 3P 4W PMS. Via the round buttons, the user can send the following inputs: FEPL, FEPR, Trail, Jamm. The square buttons allow setting such parameter as SWTI and EPRED. The display shares information about PMs' states [6].

- **Test automation interface**
  This interface contains a power connector for PMS consumption (230V/16A), two five-pin plugs for control voltage connection (400V/16A, Fig. 2.6), an Ethernet connector, and 24 V input described below [6].

  - **24 V interface**
    The interface contains one I/O connector for each PM. The following signals can be transmitted through it: EPR, EPL, COVP, MRPO, MLPO, Trail, Jamm, FEPR, FEPL, CVCU, ELT [6].

- **Modbus TCP interface**
  Modbus TCP inside the device allows setting inputs and holding registers (SWTI and EPRED) and reading out outputs [6]. The Modbus mapping table can be found in Appendix A.1.

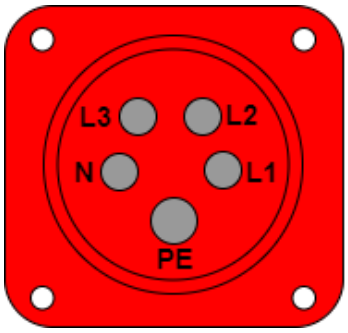



**Figure 2.6:** Control voltage connector

## 2.2 *2PMS-1P3P7W*

This section cites the 7W PMS documentation *Manual for Point Machine Simulator* [7].

*2PMS-1P3P7W* is a 7W expanded version of *2PMS-3P4W* and keeps most of its functionality [7].

### 2.2.1 Specifications

This subsection contains specifications and characteristics of 1P/3P 7W PMS.

#### Functions & requirements

The list of the required functions was expanded by the following points [7]:

- updated physical interface and logic for the 1P 7W and 3P 7W PM simulation;
- possibility to configure the PMS power interface (1P/3P) by the user;
- backward compatibility of the Modbus and configuration interfaces with *2PMS-3P4W*;
- reduced 24 V parallel I/O interface: only power interface selection (1P/3P) available.

#### Position sensing

The position detection circuitry of 7W PMS is separated from power lines. The detection signals are the following [7]:

- Positive voltage output,
- Negative voltage output,
- Detector 1,
- Detector 2.

The voltage outputs are shunted to the detectors according to the point position. Table 2.4 shows their functional logic. S1 and S2 are detector relays; the sign +/- represents positive/negative voltage; 1 represents closed relay, and 0 represents opened relay [7].

**Table 2.4:** *2PMS-1P3P7W* indicated positions and corresponding relay states [7]

| **Position** | **S2+** | **S2-** | **S1-** | **S1+** |
|---|---|---|---|---|
| Unplugged | 0 | 0 | 0 | 0 |
| Middle | 0 | 0 | 0 | 0 |
| Left | 0 | 1 | 0 | 1 |
| Right | 1 | 0 | 1 | 0 |
| Invalid | 0 | 1 | 1 | 0 |

### Motor phase orders and switching direction sensing

Motor phase orders for the 3P 7W power interface are the same as for 4W PMS [7]:

- L1-L2-L3 - *Moving LEFT*,
- L1-L3-L2 - *Moving RIGHT*.

The logic of sensing the switching direction for the 1P 7W power interface is the following [7]:

- *Moving LEFT* is simulated in case of detected voltage on L1 if the current EP is not left;
- *Moving RIGHT* is simulated in case of detected voltage on L2 if the current EP is not right.

### Inputs and outputs

Most of the 7W PMS inputs and outputs are the same as for 3P 4W PMS. The following tables demonstrate the differences: Table 2.5 describes pulse input changes, Table 2.6 shows changes in switch inputs.

**Table 2.5:** *2PMS-1P3P7W* pulse inputs [7]

| **Signal name** | **Description and restrictions** |
|---|---|
| Invalid end position | This input is a combination of the pressed Jamm and Trail buttons. |

**Table 2.6:** *2PMS-1P3P7W* switch inputs [7]

| **Signal name** | **Description and restrictions** |
|---|---|
| Position detectors override (PDOVRD) | High level of this input enables to override the position detectors. |
| Connect S2 to positive (OVS2P) | This input connects the relay S2 to the positive voltage. |
| Connect S2 to negative (OVS2M) | This input connects the relay S2 to the negative voltage. |
| Connect S1 to negative (OVS1M) | This input connects the relay S1 to the negative voltage. |
| Connect S1 to positive (OVS1P) | This input connects the relay S1 to the positive voltage. |
| Shunt L2-L3 if 1P 7W (ShuntL2L3) | The input connects the third resistor if the PM type is 1P 7W. |
| ~~ELT~~ | The input is no more available. |

The possibility of setting PMS time parameters is retained.

The 7W version of PMS also has some extra outputs [7]:

- 7W simulator indication (is7W);
- 1P 7W is configured (is1P7W);
- L1 and L2 are shunted (ShuntedL1L2);
- L2 and L3 are shunted (ShuntedL2L3);
- L1 is connected to the load (LoadL1);
- L2 is connected to the load (LoadL2);
- L3 is connected to the load (LoadL3);
- N is connected to the load (LoadN);
- S2 is connected to the positive detection voltage (S2P);
- S2 is connected to the negative detection voltage (S2N);
- S1 is connected to the negative detection voltage (S1N);
- S1 is connected to the positive detection voltage (S1P).

### Logic

The trailing, jamming and forcing EP diagrams for 7W PMS are the same as for 4W PMS (Fig. 2.4, 2.3, 2.5). The switching diagram shown in Figure 2.2 is also valid for the 3P 7W PMS switching. The diagram for the 1P 7W PMS switching is similar to Figure 2.7. The only difference is that EP depends on the phase voltage presence relays, not on the phase orders [7]:

- L1 voltage is present - *LEFT* EP;
- L2 voltage is present - *RIGHT* EP.

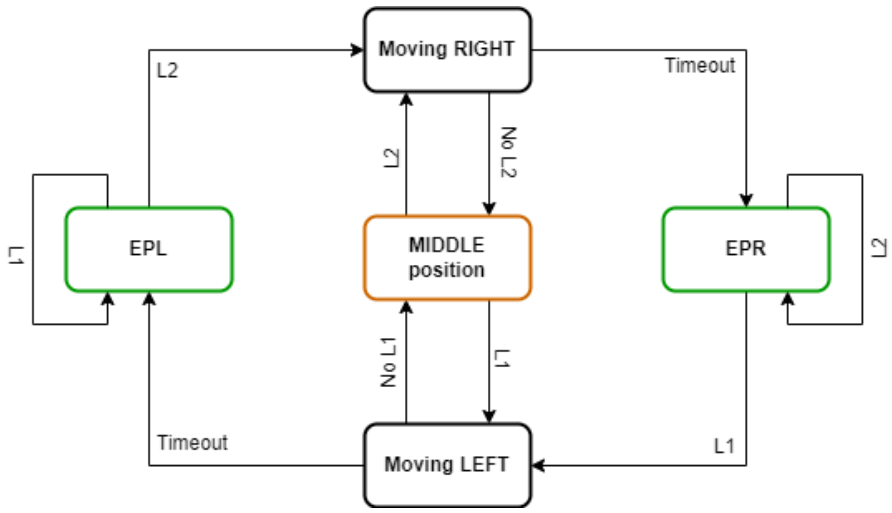



**Figure 2.7:** 1P switching process diagram (adapted from [7])

Figure 2.8 shows setting the *INVALID* position (both detectors are connected to the negative voltage) by pressing the *Jamm* and *Trail* buttons simultaneously [7].

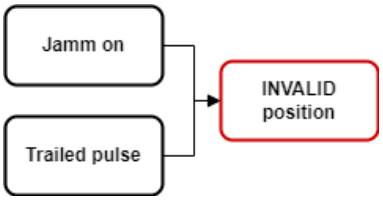



**Figure 2.8:** Invalid position setting process diagram

The 7W PMS also has the possibility to override the position detector interface and set the PM position manually [7]:

- the PDOVRD coil must be set to True;
- the detector outputs are set by the corresponding OVS input signals;
- the final position is then detected according to Table 2.4.

### Interfaces

- **Power interface**
  The power interface can be configured for simulation 1P 7W or 3P 7W PM [7].
- **User interface**
  The functionality of the user interface is adapted to the 7W PMS inputs and outputs [7].
- **Test automation interface**
  The test automation interface was expanded by two detector connectors with the following pins [7]:
  - positive detection voltage;
  - negative detection voltage;
  - S1;
  - S2.
- **24 V interface**
  The 24 V interface is reduced to perform only the function of selecting PM type [7]:
  - if the voltage is present, the 1P mode is selected;
  - if the voltage is not present, the 3P mode is selected.
- **Modbus TCP interface**
  The Modbus interface mapping also was expanded. It can be found in Appendix (A.2).

# Chapter 3

# Application design and development

One of the master's thesis aims is to create an application for communication and interaction with available PMS types. The application will be used as a testing and interfacing environment within Siemens Mobility s.r.o. This chapter is about application development and implementation.

## 3.1 Requirements

Requirements for the application are described in the current section.

### 3.1.1 Backend

The requirements for the application backend are the following:

- multiple instances (up to six) connection possible;
- communication with PMSs via Modbus TCP;
- support for the Modbus-driven simulators *2PMS-3P4W* and *2PMS-1P3P7W*;
- logging of the PMSs behavior;
- possibility to export logs to files;
- possibility to combine logs of each PMS to one file;
- possibility to setup parameters for the given type of PM;
- the application shall be compatible with Windows;
  compatibility with Linux is optional.

### 3.1.2 Frontend

The application should have a user-friendly GUI meeting the requirements described below.

### Observer tab

The first tab should contain a visualization of PMs movements. Figure 3.1 shows the example of what it can look like.

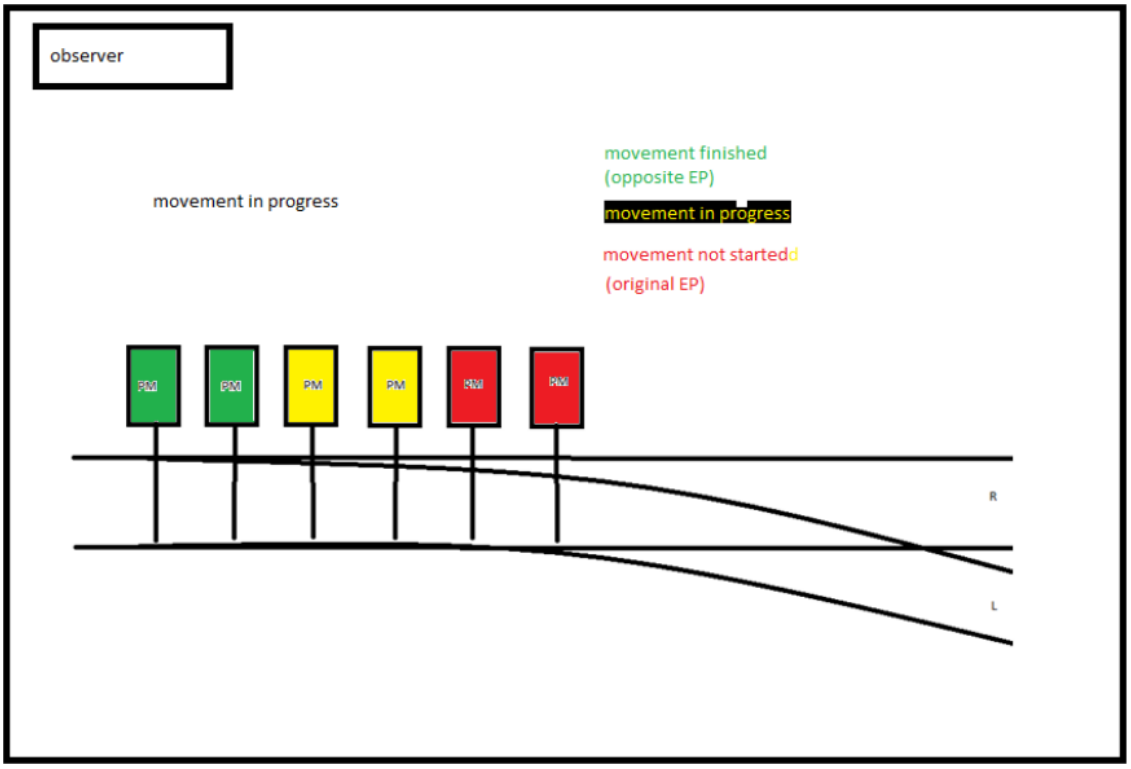



**Figure 3.1:** *Observer* tab of the application

### Setup tab

The second tab is intended for PMSs' parameters setting (Fig. 3.3).

### Logger tab

The *Logger* tab should create and display logs in a bash style for each PM.

## 3.1.3 Communication protocol

Modbus TCP should be used for the communication between the application and PMSs.

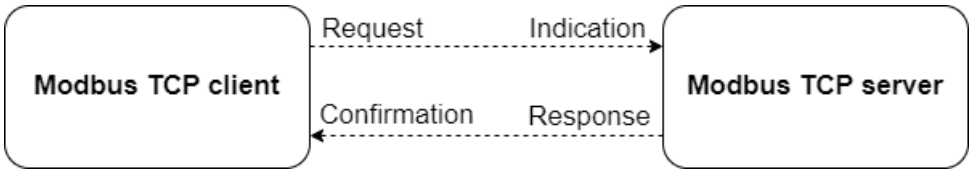



**Figure 3.2:** Modbus TCP Client/Server model (adapted from [8])

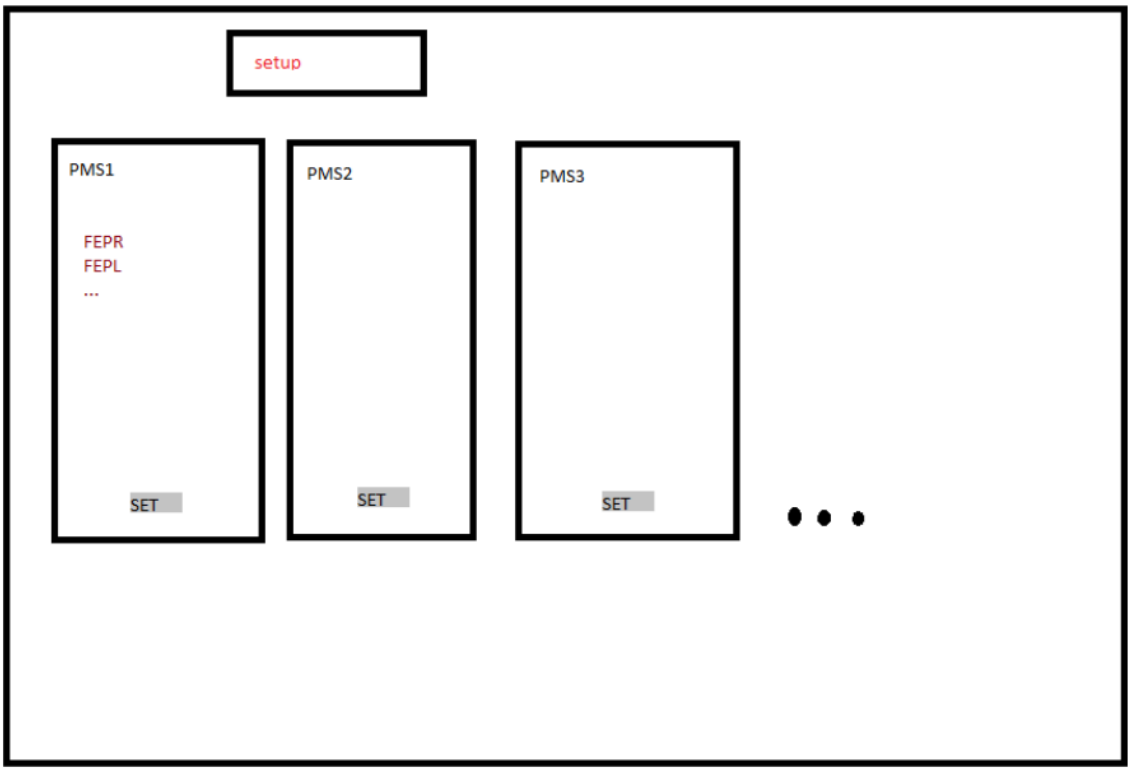



**Figure 3.3:** *Setup* tab of the application

Modbus TCP is a protocol from the application layer of the open systems interconnection model, which is used by devices connected to an Ethernet Transmission Control Protocol/Internet Protocol (TCP/IP) network for client-server communication. It can operate with four types of messages (Fig. 3.4) [8]:

- *Request*;
- *Indication*;
- *Response*;
- *Confirmation*.

The principle of the model functionality is the following [8]:

1. The client sends the *Request* message to the server for initiation of a transaction;
2. *Indication* is the *Request* message received by the server;
3. The server sends the *Response* message to the client;
4. *Confirmation* is the *Response* message received by the client.

Table 3.1 contains four primary data model tables on which the Modbus protocol is based [8].

PMS's logical inputs and outputs can be set and read out via Modbus TCP [7]. I will use the standard RJ-45 connector for the Modbus TCP interface accessing.

**Table 3.1:** Primary tables of Modbus data model [8]

| Primary tables | Object type | Type of |
|---|---|---|
| Discrete inputs | Single bit | Read-Only |
| Coils | Single bit | Read-Write |
| Input Registers | 16-bit word | Read-Only |
| Holding registers | 16-bit word | Read-Write |

## 3.2 Implementation

This section contains an explanation of the application functionality, a description of the tools used during the application implementation, and the application user manual. The name of the application is *PMS Control Application*.

### 3.2.1 Software tools and libraries

A programming language chosen for the application implementation is Python. It is convenient for creating applications with GUI, where it is necessary to combine backend and frontend implementation.

This subsection describes some tools and Python libraries that were used for the application development.

#### Pymodbus

*Pymodbus* is a Python implementation of the Modbus protocol. It enables [9]

- Modbus server-client connection,
- reading from coils and registers,
- writing to coils and registers.

In our case, the application is considered as a client, and PMSs are servers. The protocol uses port 502 as a default local port of the Modbus server [8].

#### Tkinter & Pygubu

*Tkinter* is a Python GUI cross-platform (supported by both Windows and Linux) framework. It is a good tool for creating a convenient and user-friendly interface. This framework was used for the implementation of application's GUI [10].

Some parts of GUI were designed with the help of *Pygubu*. *Pygubu* is a rapid application development tool for the *Tkinter* module used for fast and easy GUI creation [11].

#### Doxygen

Doxygen is a tool used to generate documentation from a set of source files, including files with source code. This tool supports Python and allows the generation

of different documentation kinds [12]. Online documentation in hypertext markup language format was created with the help of Doxygen.

### Pyinstaller

*Pyinstaller* is a Python package used for creating an executable file from a Python script [13].

## 3.2.2 Architecture

As it is defined in the requirements for the backend, communication between the application and PMSs should be provided via Modbus TCP (Fig. 3.4). The application acts as a client: it send requests to PMS (server), receive responses from the server, and analyze them. PMS's Modbus coils and registers, and their numbers, can be found in Appendix A.

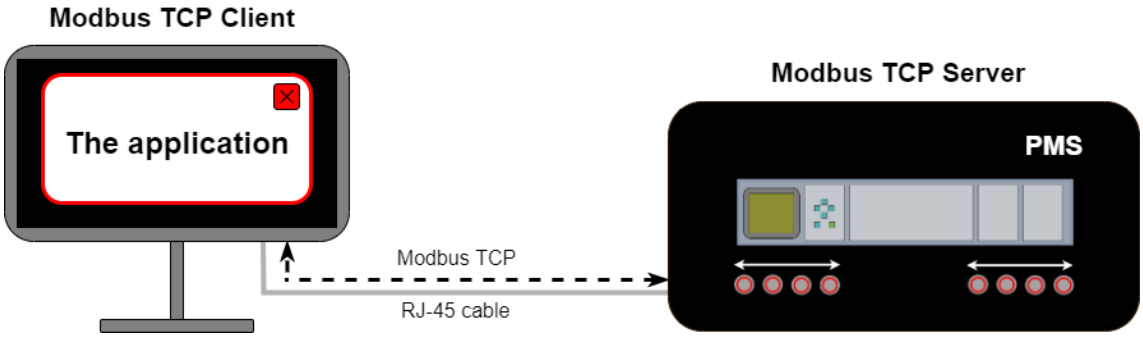



**Figure 3.4:** Modbus TCP Client/Server communication

### Functionality

The application code structure is shown in Figure 3.5. Two Python modules were created:

- *tools.py* is a module containing classes needed for the proper functionality of the created Modbus client and some tools used by *pms_control_app.py*. It consists of two classes: *WelcomeWindow* and *MainWindow*.
- *pms_control_app.py* is a module ensuring that the application functions according to the requirements by the appropriate connection of the backend and frontend. It contains the following classes: *PMStypes*, *Timer*, *Client*, *Registers* and *Logs*.

*WelcomeWindow* class is responsible for starting the application. It creates the window shown in Figure 3.7 with the help of *Tkinter* module, processes entered information and starts the *MainWindow* class object.

*MainWindow* class creates the main window of the application (Fig. 3.8) with all the tabs described in the requirements. With the help of specific modules, it also ensures the tabs' functionality (including sending particular Modbus messages) and definite reactions to received Modbus messages. Furthermore, the class provides possible

error prevention and controls all the frontend features such as animation, popup windows, and logging.

*Logs* is a class used by *MainWindow* for the *Logger* tab (Fig. 3.11) and probable warnings (Fig. 3.13) and error messages (Fig. 3.12). It contains all the possible announcements that could appear during the application run.

*Timer* is also used for the proper functionality of the *Logger* tab. It aims to count time and convert it to the correct format for logging.

*PMStypes* is a small enumeration class used by *MainWindow* and *Client* to distinguish different PMS types.

*Registers* class consists of Modbus register definitions for possible PMS types. It also has some functions facilitating working with the registers for *Client*.

*Client* is a class actively working with the *Pymodbus* library. It enables connection to the Modbus servers and communication with them. It also contains an additional function used for basic testing PMSs, which is partly described in Section 4.4.

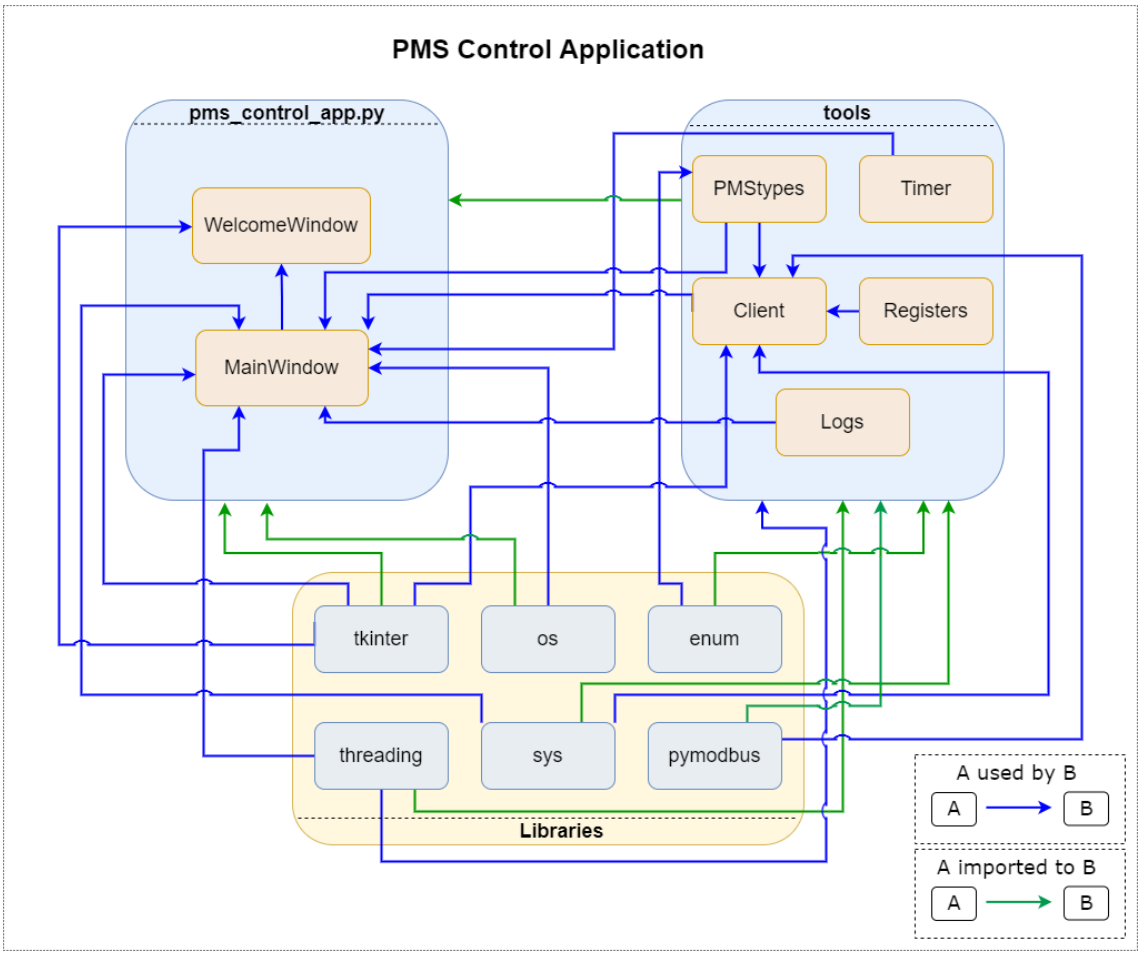



**Figure 3.5:** *PMS Control Application* code structure

#### Multithreading

The application has GUI consisting of several tabs and also has the possibility to connect multiply Modbus servers, which should be processed in parallel. That is why thread-based parallelism is needed for the correct functionality of the application.

The Python library named *Threading* was used to realize it. This module allows running different parts of the program concurrently [14].

All the threads start in the *MainWindow* class. The count of running threads depends on the number of connected PMSs. Three of them are started every time:

- one for logging,
- one for changing the rails' color (animation),
- a *Tkinter* main loop.

Each of the rest threads is started as many times as there is the value of calculated PMs. Their main functions are the following:

- updating positions of the PM rectangles in the *Observer* tab (animation);
- updating states of the input buttons in the *Setup* tab.

### 3.2.3 GUI and manual

The user has the possibility to create a shortcut shown in Figure 3.6 via the attached Python script *makeShotcut.py*.

After starting the application by double clicking on the shortcut or the executable

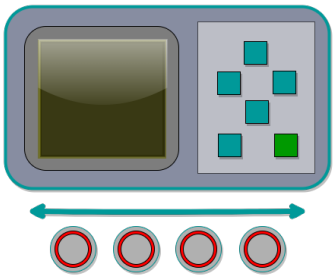


**Figure 3.6:** *PMS Control Application* shortcut

file, the user will see a window shown in Figure 3.7. He will enter the IP addresses of all the PMSs that should be connected and click *Ok* then. The first PM of the first PMS will be considered as a master. The main window with four tabs will be opened.

What each tab looks like depends on the number and the type of the connected PMs. It also can be automatically changed with a physically changed PM type (1P/3P for 7W PMS).

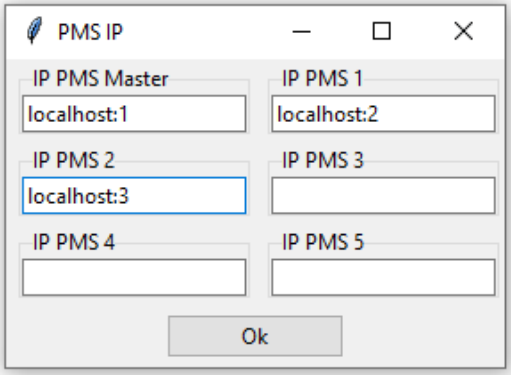



**Figure 3.7:** *PMS Control Application Welcome Window*

The *Observer* tab (Fig. 3.8) shows PMs' and the rails' states graphically and some information about PMSs (IP address of each PMS and throwing time of each PM). Every change in PMSs' states is shown in this tab as an animation in real-time.

The *Setup* tab (Fig. 3.9) can be used for sending commands to PMSs via Modbus TCP. Buttons and entries also react to PMSs' state changes, for example, if commands were sent to some PMS out of the application.

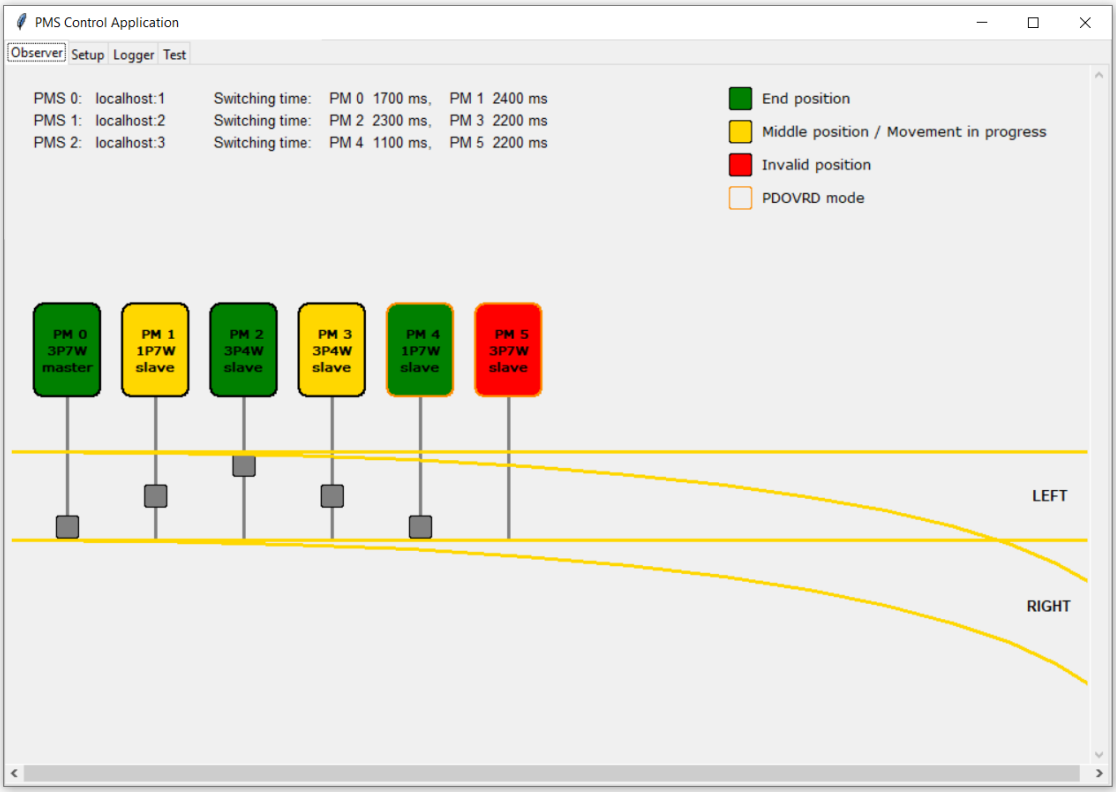



**Figure 3.8:** *PMS Control Application Observer* tab

The user can follow the PMSs behavior as logs with time stamps in the *Logger* tab (Fig. 3.11). There is always some basic information about each PM at the beginning of the log. Then *Logger* reacts to every change in PMSs' states. The log can be

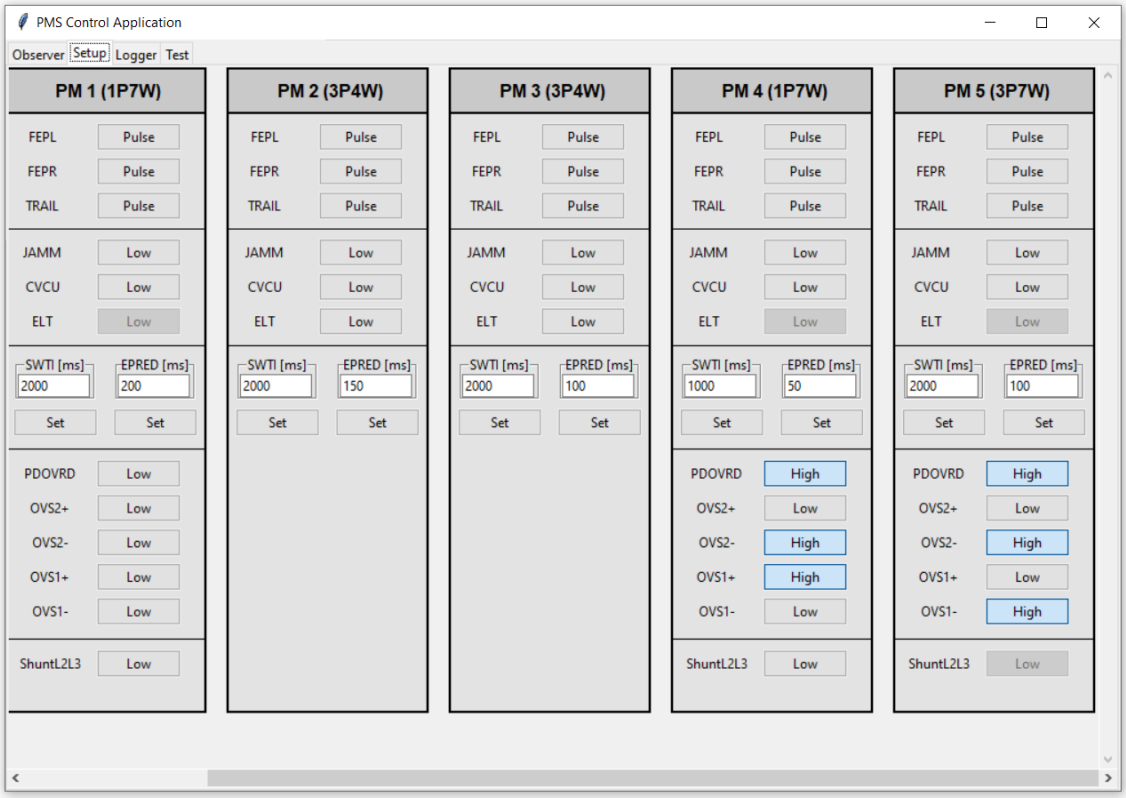



**Figure 3.9:** *PMS Control Application Setup* tab

exported to the user's device in the *.log* format. One overall log file and one file for each PM will be created (Fig. 3.10). Before exporting, logging must be stopped via the *Stop* button. After pressing *Export...* the user should choose the directory where logs will be stored. The folder logs with all the *.log* files will be created in this directory. In case of the existence of a folder with the same name, files inside will be rewritten.

The *Test* tab (Fig. 3.13) is intended for testing the basic functions of all the connected PMSs' Modbus interfaces. After pressing the *Test* button, the user will wait until all of them have been tested. The other tabs will be disabled during the process.

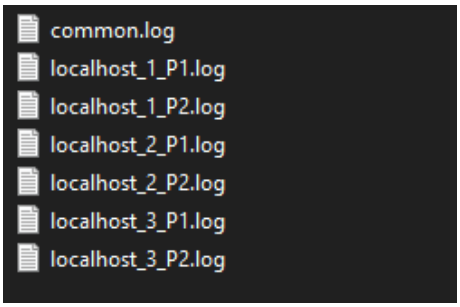



**Figure 3.10:** Exported *.log* files

The numbers of tests for one PM are the following:

- 3P 4W PM: 8;
- 3P 7W PM: 15;
- 1P 7W PM: 16.

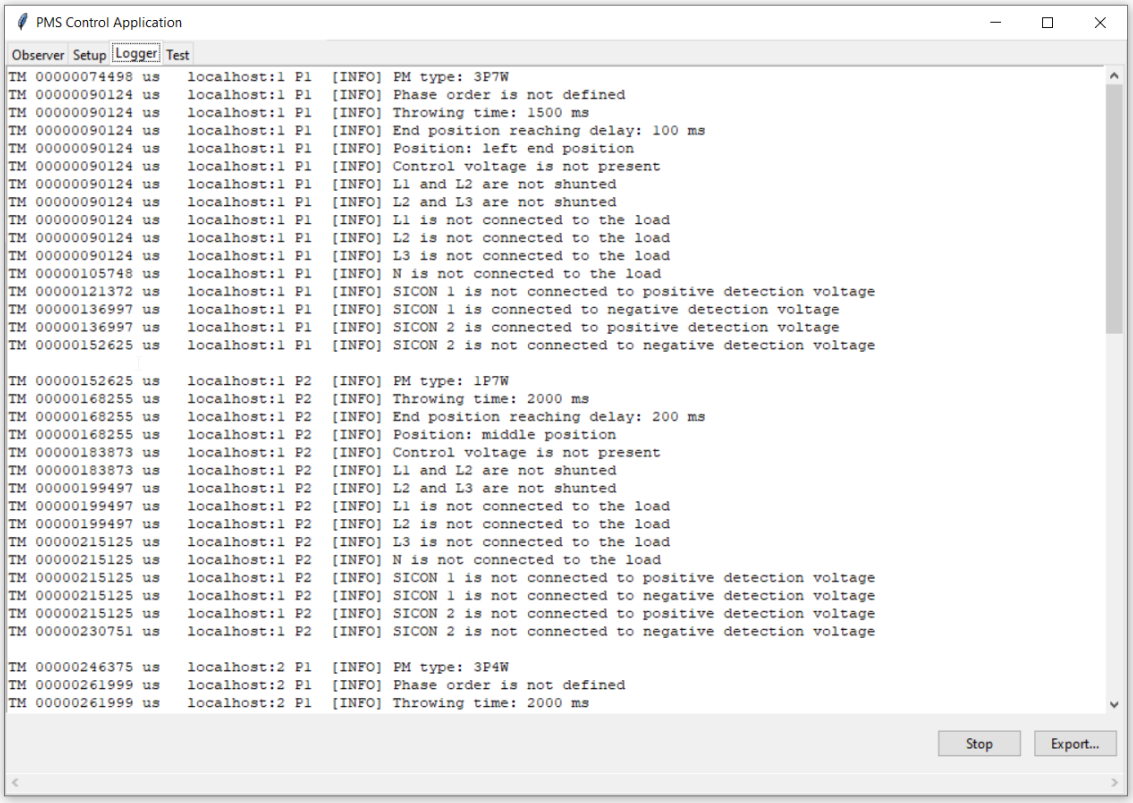



**Figure 3.11:** *PMS Control Application* Logger tab

The application also ensures certain communication with the user through warnings (Fig. 3.13) and error messages (Fig. 3.12).

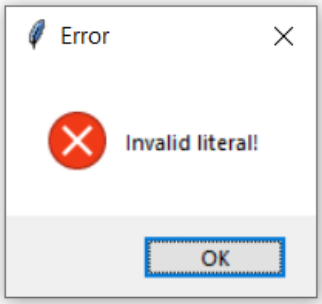



**Figure 3.12:** *PMS Control Application* error message

## 3.3 Summary

The created *PMS Control Application* is used for both-side communication with the different types of PMS. The communication is provided via Modbus TCP. Two PMS types are supported:

- *2PMS-3P4W*,
- *2PMS-1P3P4W*.

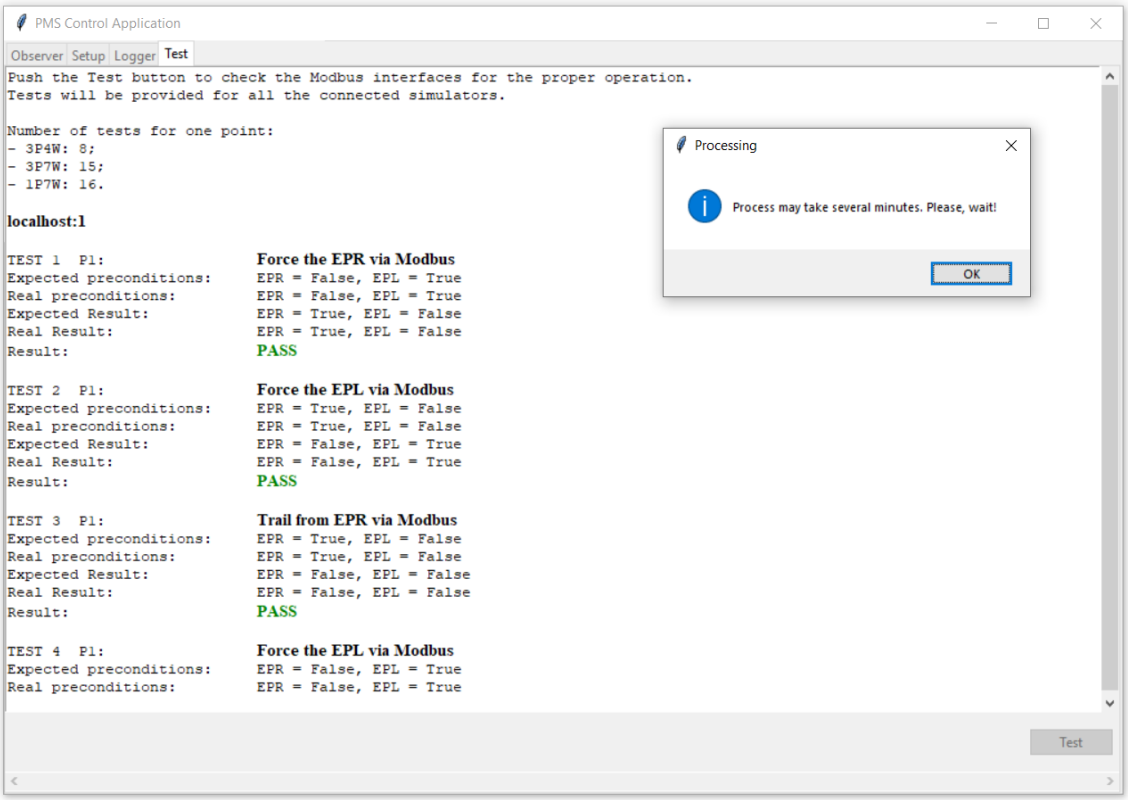



**Figure 3.13:** *PMS Control Application Test* tab

The application provides the following functions:

- Observation of the PMS set (up to six) behavior. The set can contain a combination of different PMS types.
- Sending commands to PMSs, receiving signals from PMSs, and updating their virtual states.
- Logging of the PMSs' behavior with the possibility of export (*.log* file format, one common file, one file for each connected PM).
- Testing the basic PMSs' Modbus interface functionality. This possibility is out of the requirements. It was implemented as part of the 7W PMS testing described in Section 4.4 and expanded for 4W PMS.

The whole application was created and tested in Windows with the help of tools also supported by Linux.

Software testing and verification are described in Section 4.2.

# Chapter 4

# System testing

This chapter describes the testing part of the project. It contains

- theoretical information about the standards according to which tests were provided (Section 4.1);
- description of creating necessary testing tools (Sections 4.2.3, 4.3);
- description of SW and HW testing processes themselves (Section 4.2, 4.4).

## 4.1 CENELEC

CENELEC means European Committee for Electrotechnical Standardization. It deals with the creation of European norms (EN) for standardization in all areas of the electrical engineering. These standards also guides SW and HW development

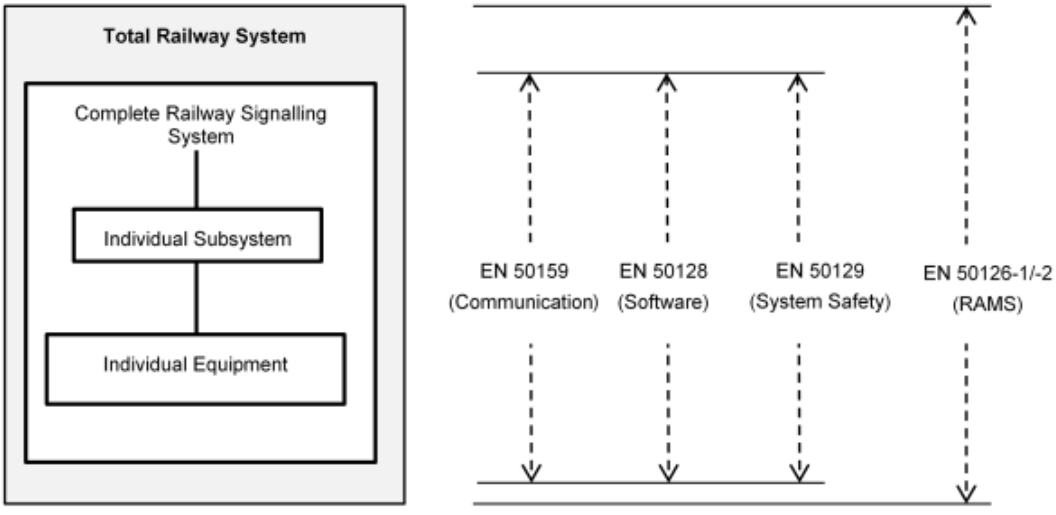



**Figure 4.1:** Scope of the main CENELEC railway applications standards [15]

for the railway industry. In Figure 4.1, there is a relationship between CENELEC standards for the railway applications [15]. As part of this work, I used two of them:

- *EN 50128* - to test the developed SW application for communication with PMS;
- *EN 50129* - to test 7W PMS.

### 4.1.1 EN 50128

Information in this section has been taken from the document *EN 50128* [15].

*EN 50128* is a European standard containing requirements for any safety-related railway control and protection of an application lifecycle. This standard is not suitable for SW, which behavior cannot affect the system's safety. It is intended solely for SW and its relation with the system of which it is part, including [15]

- operating systems,
- firmware,
- support tools,
- application programming containing high/low-level programming and special-purpose programming.

Standard also deals with the use of SW, which already exists [15].

ENs deal with five SW safety integrity levels (SILs): the lowest is SIL 0, the highest is SIL 4. Each of them has its own required measures and techniques. SIL of SW should be defined based on several factors [15]:

- SIL of the system,
- risks arising from the use of this software in this system,
- social factors,
- economic factors.

The software developed as a part of the master's thesis is an application with GUI for dealing with some railway devices. Testing of this application is performed according to CENELEC standard *EN 50128* as for SW with SIL 0. The following subsections describe some methods that can be used for application testing.

#### Functional testing

Functional (black box) testing is a method for analyzing the functionality of the SW application. It does not require information about the product's internal structure. The method's techniques are described below.

- **Boundary value analysis** (BVA) aims to detect and eliminate errors that might appear at parameter limits or boundaries. Every program has a set of possible inputs called an input domain. According to this technique, the input domain should be divided into several classes. Realized test cases should cover their extremes and boundaries, forcing the output values to their limits. Particular attention should be paid to zero values used as zero-divisors, null matrix, and non-printing control characters. They might be error-prone [15].
- The **equivalence class and input partition testing** (ECIPT) purpose is sufficient SW testing with a minimum of test data. As in the previous technique, the input domain should be partitioned into classes. There are two basic ways [15]:

  - Equivalence classes defined on the specifications:
    - input-oriented - the same operations with the selected values;
    - output-oriented - the same functional result for the selected values.
  - Equivalence classes defined on the internal program structure:
    - the class results are specified by static analysis of SW - the same way of execution for the selected values.

- The **process simulation** (PS) goal is to test the SW operation together with its interface. It is performed with the help of a testing method that simulates the behavior of the real system, driven by the system under test. This method may consist of SW only or SW and hardware (HW) combination. The following features should be simulated [15]:
  - all existing inputs for the system under test available after its installation;
  - responses to the system outputs corresponding with true reactions of the controlled system;
  - the possibility of some input perturbations, which the system under test has to be able to deal with.

#### Static analysis

Static analysis is a testing method for examining a program without executing it.

- **Walkthrough** is a technique used to detect errors in the system as soon as possible at the lowest cost. A small set of test cases representing the program's inputs and corresponding outputs is selected for Walkthrough. This test data is then manually traced through the program logic [15].
- **Design and code reviews** aim to check if the reviewed product fulfills the requirements and standards [15].

#### Test coverage for code

Test coverage for code (TCC) is used for the dynamic analysis associated with the program execution. Test coverage is a measure of the executed code lines amount. It is calculated during a run of the set of tests.

- **Structure based testing** aims to apply tests that exercise the specific program structure subsets. A set of input data is selected in such a way that a significant part of chosen program elements is exercised. These program elements can differ depending on the required severity level [15]:
  - the **statement** coverage is the least strict test: all statements of the source code can be executed without exercising two conditional statement branches;
  - the **branch** coverage should check both sides of every branch so that all possible outcomes are executed at least once for each condition;
  - the **compound conditions** coverage test executes each condition of a branch linked by operators $and/or$;

- a **linear code sequence and jump** is a part of the SW program containing a linear sequence of code statements terminated by a jump;
- the **data flow** coverage is based on the definition and usage of data - for each variable used in a statement, all possible execution paths where the variable was defined should be found;
- a **call graph** is a tree of program subroutine invocations - designed tests should cover all of them.
- an **entire path** means that all potential program paths should be executed. Total entire path testing is usually unattainable because of the large number of possible paths.

### 4.1.2 EN 50129

This section is based on the document *EN 50129* [16].

*EN 50129* describes requirements for the approval of safety-related electronic systems, subsystems, and tools for railway signaling applications. Besides generic systems, it can also be applied to specific applications. The standard cannot be used for any security aspects other than the functional safety of systems. It is intended for all safety-related electronic system life cycle phases (Fig. 4.2) [16].

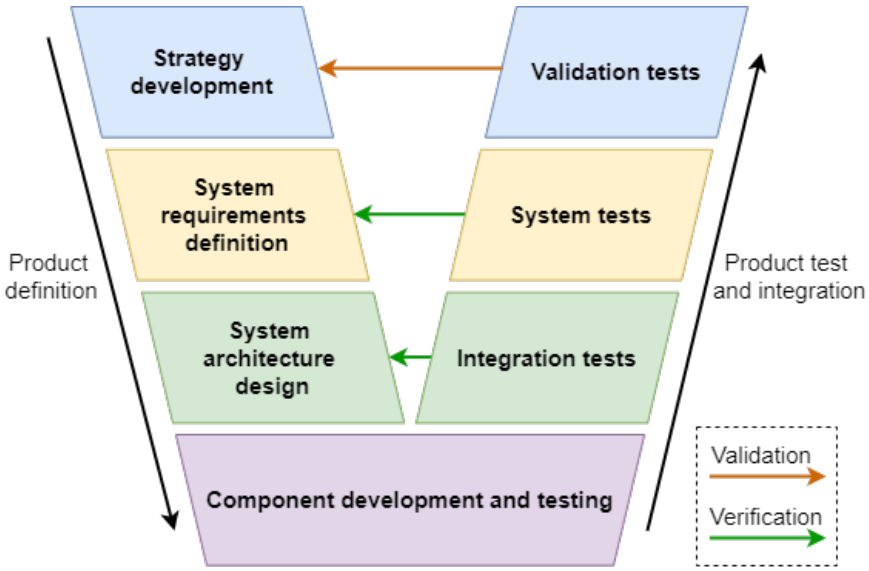



**Figure 4.2:** Product development life cycle (adapted from [16])

In this thesis, the norm is used for the factory testing of 7W PMS and documenting the product development process. The techniques needed for achieving the goals are listed below. They all belong to the testing part of the product development life cycle [16].

- **Design review** aims to check if the product complies with the input requirements for the development.

  - The purpose of **functional testing and formal verification of the system** is to control if the product conforms to the specifications and safety requirements. The results should answer the question: *"Has the product been developed correctly?"* [16].

- **Test facilities and simulation** method includes the creation of test equipment to simulate some functions of the system under test or associated with it wares [16].

## 4.2 Application testing

The created application was tested according to *EN 50128*. General principles of the tests are explained in Section 4.1. The current section describes the process of testing the developed SW and the test results.

### 4.2.1 Boundary value analysis

BVA implies the separation of an input domain of the program into classes with clear boundaries [15]. Classes' boundary values are used as test inputs. The program can get non-standard inputs from the user via GUI or from the server in the form of Modbus messages. A set of tests was created and provided manually from the server/user side. The details are in the following subsections.

**Welcome Window functionality: IP fields and Ok button**

- **Classes:**
  - Numbers of entered valid IP addresses from 1 to 6 (6 is the number of the *Welcome Window* fields);
    Empty sequence.
    * **Precondition**: six available Modbus servers.
    * **Inputs**: six entered IP addresses; an empty input.

- **Issues:**
  - The empty input raises a *ValueError*.
    * **Expected behavior**: the application should be closed after the Modbus error notifying.

**Setup tab functionality: input fields and corresponding Set buttons**

- **Classes:**
  - Integer numbers from 0 to 32767 (lower and upper limits for the SWTI and EPRED registers);
    Integer numbers from $-\infty$ to $-1$;
    Integer numbers from 32768 to $\infty$.
    * **Precondition:** at least one connected valid Modbus server.
    * **Test inputs:** $-1$; 0; 32767; 32768.

- **Issues:**
  - The inputs $-1$ and 32768 do not cause any error, but they are invalid. The application does not prevent sending invalid values to the server.
    - **Expected behavior**: the program should notify attempts to send an invalid value and prevent it.

### 4.2.2 Equivalence class and input partition testing

ECIPT also includes dividing the input domain into classes. In this case, clear boundaries are unnecessary because any value from each class can be used [15]. Tests were provided manually from the server/user side, as it was done for BVA. The test descriptions are in the lists below.

**Welcome Window functionality: IP fields and Ok button**

- **Classes:**
  - Available IP addresses in the correct format;
    Unavailable IP addresses in the correct format;
    Random symbols;
    Enormously long sequences.
    - **Precondition:** 10.1.255.101 : 502 is an available Modbus server.
    - **Test inputs:** 10.1.255.101 : 502; 10.1.255.102 : 502; $as.2.de4.\_12$ :,′; a million symbols long sequence generated in Python terminal.
- **Issues:**
  - Enormously long input leads to a failure: the application does not respond.
    - **Expected behavior**: the program should generate the error message for the user and continue to work or prevent entering such a long sequence.

**Setup tab functionality: input fields and corresponding Set buttons**

- **Classes:**
  - Integer numbers from 0 to 32767;
    Integer numbers from $-\infty$ to $-1$;
    Integer numbers from 32768 to $\infty$;
    Real numbers from 0 to 32767;
    Real numbers less than 0;
    Real numbers from greater than 32767;
    Random symbols;
    Enormously long sequences;
    No symbols.
    - **Precondition:** at least one connected valid Modbus server.
    - **Test inputs:** 1234; $-1234$; 54321; 1234.5; $-1234.5$; 54321.5; $a12hjdkha$; a million symbols long sequence; empty input.

- **Issues:**
  - The inputs less than 0 and greater than 32767 do not cause any error, but they are invalid. The application does not prevent sending invalid values to the server.
    - **Expected behavior**: the program should notify attempts to send an invalid value and prevent it.
  - The application stops answering because of enormously long input.
    - **Expected behavior**: the program should generate the error message for the user and continue to work or prevent entering such a long sequence.
  - The *Logger* tab reacts to setting random symbols.
    - **Expected behavior**: *Logger* should ignore attempts to set an invalid value.

**Observer, Setup and Logger tabs functionality: changing coils values**

- **Classes:**
  - Boolean values True and False;
    Numbers 0 and 1;
    Numbers from $-\infty$ to $-1$;
    Numbers from 2 to $\infty$;
    Random symbols;
    Empty sequence.
    - **Precondition:** at least one available Modbus server.
    - **Test inputs:** $False$, 1, $-123$, 123, "$t$", "".
- **Issues:**
  - There are no problems. The application behaves as expected.

BVA and ECIPT test methods have definite restrictions that significantly reduce the number of possible test cases. Despite this fact, provided tests were able to find several SW problems.

### 4.2.3 Process simulation

PS testing requires a tool able to simulate the behavior of a system driven by SW under test [15]. A Python module *pms.py* simulating the behavior of both PMS types was implemented to meet this condition.

#### Test software

*pms.py* was designed and implemented according to the PMSs' logic described in Chapter 2. So, it can be used as a virtual PMS to demonstrate the *PMS Control Application* functionality or for its manual testing. The module's code contains two classes:

- *Server*, ensuring the communication via Modbus and simulating PMSs behavior (*2PMS-3P4W*/*2PMS-1P3P4W*.);
- *MainApp*, connecting the server functionality with GUI.


**(a)** *pms.py* 4W PMS type **(b)** *pms.py* 7W PMS type

**Figure 4.3:** *pms.py* 7W PMS type window

The application provides the following functions:

- simulation of the PMSs' interfaces (Fig. 4.4a) including choice of a PMS type (Fig. 4.3);
- simulation of the PMSs' functionality;
- communication via Modbus TCP;
- demonstration of the PMS outputs (Fig. 4.4b).

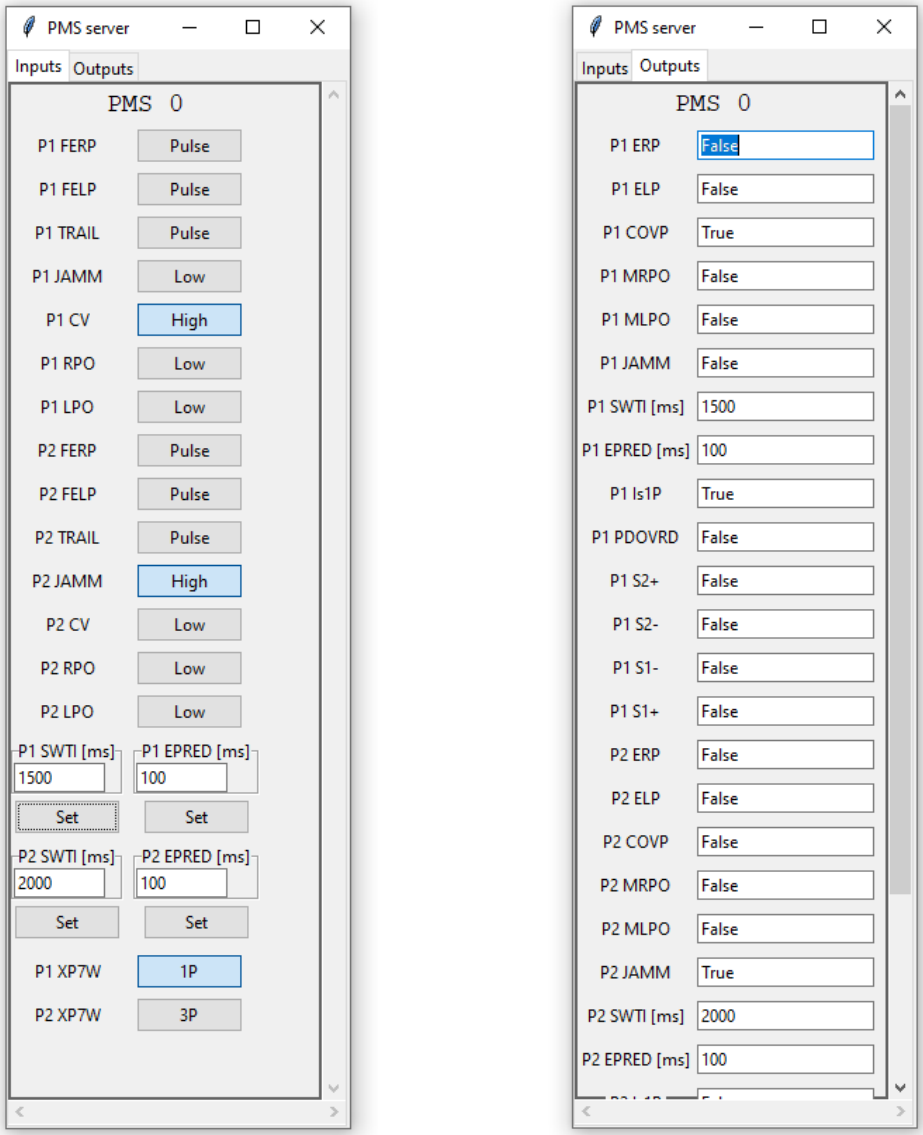
The *pms.py* functionality must not meet the requirements for a properly functioning PMSs because of intentionally added reaction perturbations. The purpose of the application is to simulate possible inputs to *PMS Control Application* and possible PMSs' reactions.

### Testing process

All tests were provided manually with the help of the developed PS SW. Its *Output* interface (Fig. 4.4b) was checked and compared with the interfaces of *PMS Control Application* during every running test.

The testing was divided into several parts described in the following list.

- *Observer tab*, *Setup tab*, and *Logger tab* functionalities were tested for the following simulations of PMS:
  - *2PMS-3P4W*:
    - two 3P PMs.
  - *2PMS-1P3P7W*:
    - two 3P PMs;




**(a)** *pms.py Inputs* tab **(b)** *pms.py Outputs* tab

**Figure 4.4:** *pms.py* GUI

  - two 1P PMs;
  - the first PM is 3P, the second PM is 1P;
  - the first PM is 1P, the second PM is 3P.

  Tests were provided according to the PMSs requirements described in the simulators' specifications (Chapter 2).

- After that, the export of logs and the *Test* tab were tested.
- The final testing part was closely related to the TCC described in Subsection 4.2.5. The simulations of different processes were being created until all lines of the code that could have been covered were covered. The PS SW functionality was changed when needed, for example, for error issues creation.

Problems and errors revealed during PS testing are listed below.

- If a new number is entered in some field of the *Setup* tab (Fig 3.9), *Logger* logs it no matter if the *Set* button was pushed or not. The log should be created only after pushing the button.

- If some connected PM is 3P, the *L2L3_Shunt* buttons are blocked for all PMs. The buttons should be blocked for 3P PMs only.

- There are no logs about disconnecting the position detectors of 7W PMS.

- The Modbus error appears when switching the power interface (1P/3P) for 7W PMS.

- The *ELT* buttons are blocked for all present PMs if at least one 7W PMS is also connected.

- The *Logger* tab does not react to external changes (provided on the server's side) in JAMM coil and SWTI and EPRED registers.

- The Modbus error appears if the connection between the server and the client is lost.

The PS testing supported by the TCC control has benefited the code. The tests have helped to find several substantial bugs influencing some basic functions of the program.

### 4.2.4 Code Review

A design review is recommended for SW with SIL 0 [15], but it is irrelevant if the SW developer does the review of their own code. For this reason, the testing was provided by one of my colleagues from Siemens Mobility s.r.o. His goal was to check that the coding standards are met and that the code has an adequate structure. The reviewer was not directly involved in the SW design. He is a Python expert, sufficiently qualified to review and comment on the design. This type of review is formal [17]. The repository with the source code and the documentation were given to the reviewer.

The summary of the received review and my comments on it are in the following lists. The reviewer has divided his comments into general and specific.

#### General comments

- *Pylint* and *Black* should be used.

  *Pylint* is a static Python code analyzer that goes through the code without running it. The analyzer inspects the code for the presence of errors and coding standards compliance. It also can recommend how to refactor the code better [18].

  *Black* is a tool for Python code formatting. The tool can reformat the code to make it more readable [19].

- A *gitignore* file should be created in the repository to filter such unnecessary files as *.pyd*, *.dll*.

  *gitignore* defines intentionally untracked files that should be ignored by Git [20].

- It is recommended to avoid wildcard imports due to the obscurity of which names are in the namespace.

  **Example from the code:**

  ```
  from tools import *
  ```

  **Possible solution:**

  ```
  import tools
  ```

- Constants for some numbers used in the code are missing.

  Constants are more readable and convenient than numbers, and their names often explain their intentions.

  **Example from the code**:

  ```
  self.testWin.delete(10.0, tk.END)
  ```

  **Possible solution**:

  ```
  ROW_NUMBER = 10.0
  ...
  self.testWin.delete(ROW_NUMBER, tk.END)
  ```

- It would be better to separate the logic from the GUI.

  Currently, the *pms_control_app.py* combines the backend with the frontend. The solution is to divide the modules' classes into more specific functional object constructors and improve them so that the backend part of the code would be entirely independent of the frontend. In this case, the user would be able to use the application's abridged version via the command line without GUI.

- Lower letters should be used for variable naming.

  According to the Python programming language style guide, lowercase with words separated by underscores is used for variables for better readability. Functions follow the same convention [21].

  **Example from the code:**

  ```
  self.stopTabs = False
  ```

  **Possible solution**:

  ```
  self.stop_tabs = False
  ```

- It is recommended to reduce the code to smaller blocks.

  The solution is to create more classes and functions focused on more specific processes.

**Specific comments**

- The parameter *is_server* of *pms_control_app.py WelcomeWindow* class does not seem to be used. It is also not in the class description.

  The parameter should be described in the comments or removed from the code.

- The function *testModbusInterface()* from *tools.py* is too long and should be divided into several subroutines. Comments should be expanded.

  The function consists of the code block that repeatedly occurs with different arguments. So, this block could be transformed into a particular function, which would be used by *testModbusInterface()*. It improves the code readability.

- The function *logger()* from *pms_control_app.py* has the same problem as one described in the previous item. However, it has not a repeatedly occurred block of the code.

  The solution is to divide the function into several different, more specific subroutines.

The resulting review consists of some recommendations and comments for the code improvement, not indications of big mistakes. The whole review was analyzed and commented on. The results will be used for the improvement of the code.

### 4.2.5 Structure-based testing

*EN 50128* has no recommendation for or against structure-based testing for systems with SIL 0 [15], but it was provided for improving the code quality and the effectiveness of tests. The branch TCC method was chosen due to its adequate difficulty and efficiency.

For the measuring code coverage of the program, a Python tool *Coverage.py* was used. The tool monitored the program while providing each test. It noted parts of the code that was not executed, when they could be, and calculated the coverage of the code in percent [22].

*Coverage.py* monitored the code during the run of every test described in the previous subsections. The combination of the real PMSs performance and the PS testing has covered most of the code. However, it has been found out that some parts of the code cannot be covered at all. These parts should be removed from the program because they do not affect its functionality. Several errors also were revealed. The details about the arisen issues are in the tables 4.1 and 4.2. The rest of the

code was covered. Line numbers in the tables correspond to code lines in attached files *pms_control_app_py.html* and *tools_py.html* (example of the *Coverage.py* outputs), not in the Python files with the source code themselves. The Python files were edited after the testing, so they may not match the data in the tables.

**Table 4.1:** Code coverage issues for *pms_control_app.py*

| Line numbers | Description |
|---|---|
| 431-432<br>672-673 | **Problem**<br>The lines are *else* parts of the *if-else* blocks. Their functionality is to assign the name *XPXW* to some graphical objects in the case when the connected PMS type is neither 7W nor 4W. Getting into this *elses* is impossible because the class *Client* always assigns one of these two types to PMS.<br><br>**Solution**<br>The first possible solution is to remove these non-covered parts of the code. The second solution is to fix the *Client* class so it could assign more types to PMS. |
| 840-842<br>852-853<br>866-868 | **Problem**<br>The lines are *elif* parts of the *if-else* blocks. They are responsible for the switching buttons in the *Setup* tab. After starting the application, these lines change the buttons' states to *Not pressed* in case of the negative value of the corresponding Modbus coil. This functionality is useless because the original state of the buttons is automatically set to *Not pressed*.<br><br>**Solution**<br>The solution is to remove these lines. |
| 913-914<br>1038-1039<br>1107-1108<br>1132-1333<br>1299-1301<br>1352-1354 | **Problem**<br>The lines are *except* parts of the *try-except* blocks. They should ensure the safe stop of the program in cases of such issues as unexpected disconnection of the PMS Modbus interface. The problem is that the program sometimes does not catch exceptions and ends with an error.<br><br>**Solution**<br>The solution is to analyze the problem and reimplement the safe stop of the program in a different way. |

**Table 4.1: Code coverage issues for *pms_control_app.py***

| Line numbers | Description |
|---|---|
| 1209-1216 | **Problem**<br>This part of the code was planned to be removed during the implementation. However, it was revealed only while the TCC testing.<br><br>**Solution**<br>The lines should be entirely removed from the code. |

As a result, *pms_control_app.py* has eight hundred thirty-six statement runs. Twenty-seven of them were missed (not covered) during the testing. From nine hundred ninety-six statement runs of *tools.py*, eight were missed. The calculated coverage of the code is ninety-eight percent. This result can be improved if the recommended solutions from Table 4.1 and Table 4.2 are performed.

**Table 4.2:** Code coverage issues for *tools.py*

| Line numbers | Description |
|---|---|
| 146-147<br>161-162<br>177-178<br>192-193 | The listed lines have the same problem and solution as lines 913-914 of *pms_control_app.py* (Table 4.1). |
| 1518 | **Problem**<br>This line is an *else* statement of the *Timer* class's *if-else* block. Its functionality is to assign a zero value to the variable *zeroNumber*. This variable represents a number of zeros for the *Logger* timing. It is needed for the right time stamp format creation: time should be in microseconds, and notation should contain eleven or more symbols, for example, 00000031715. The *if-else* block calculates a number of zeros at the beginning of the notation. The considered line was not covered during any test because assigning a zero value to the variable *zeroNumber* would mean expiring a considerable amount of time from starting the application (almost three hours).<br><br>**Solution**<br>It is sufficiently clear that the line would be covered if the above-described condition was met. If complete testing is required, the solution is to give the program enough time to run. |

TCC testing has helped to find unnecessary parts of the code and some weak points and errors. Beyond the effectiveness of the code, this test method can improve the code structure and appearance. A significant amount of problems were found during the testing. All substantial errors that could influence the essential application functionality were fixed.

The next step is processing the rest of the issues. Some of them already have devised solutions, and some need to be analyzed more particularly.

### 4.2.6 Summary

A combination of the chosen test techniques found a set of SW issues and indicated the code disadvantages. Most of the problems were fixed right after testing. The rest of them will also be fixed according to the recommendations and solutions described in this section.

## 4.3 Manual Point Driver

This section describes the implementation of the *Manual Point Driver* (MPD), a HW tool needed for providing tests associated with starting the 7W PMS's engines. It is a forward-reverse motor starter created in two versions: the first one is for the 3P motors, and the second one is for the 1P motors. The device's purpose is to start a motor inside the connected PMS with the possibility of changing the direction of rotation. Basically, MPD provides one of the point driver's functions.

### 4.3.1 Requirements

The device must meet the following requirements:

- rotation to the right should be possible;
- rotation to the left should be possible;
- electrical circuit protection must be provided;
- the device must be compatible with the 7W PMS (Fig. 4.5);
- a user interface for manual control must be present.

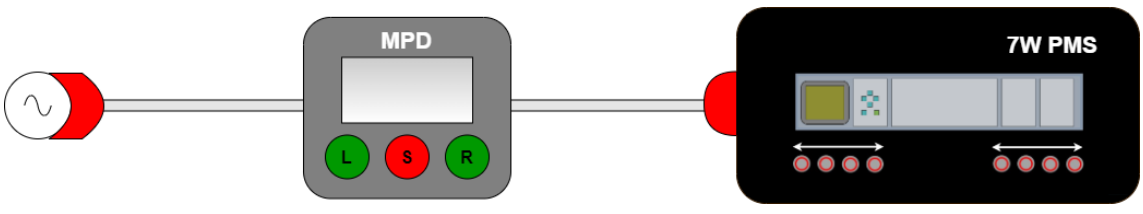



**Figure 4.5:** MPD and 7W PMS connection

### 4.3.2 Functionality

Figures 4.6 and 4.7 show MPD circuits for the different motor types. Each circuit can be logically divided into two parts:

- the protective part containing a residual-current circuit breaker (Q1), a circuit breaker (Q2), and fuses (F1, F2);
- the control part containing two contactors (Q3, Q4) and three push buttons (PB1, PB2, PB3).

#### 3P 7W/4W Manual Point Driver

The direction of the 3P motor rotation depends on its phase order. In the manual for *2PMS-1P3P7W*, the *Moving LEFT* phase order is defined as the primary order L1-L2-L3, and the *Moving RIGHT* phase order is defined as L1-L3-L2. The control voltage must be present in both cases [6].

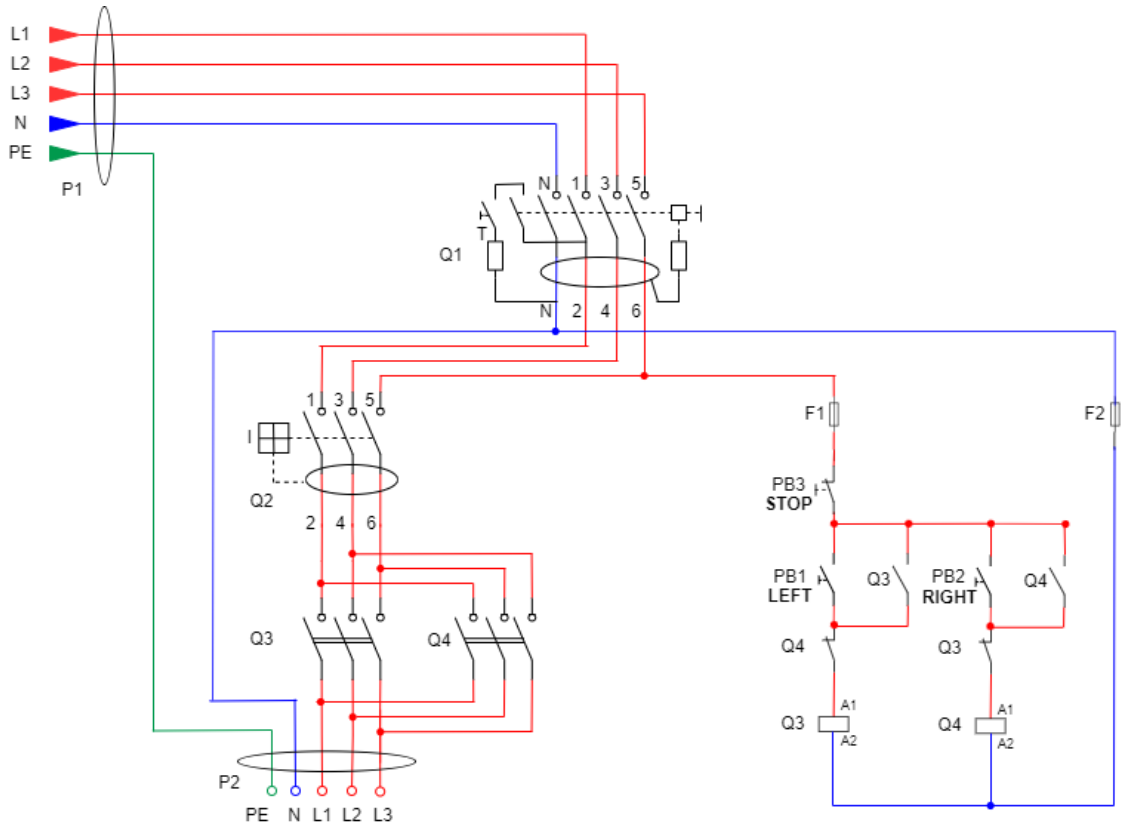



**Figure 4.6:** 3P 7W/4W MDP wiring diagram (adapted from [23])

The user can control the circuit and so the motor behavior via three push buttons [23]:

- Pressing the *LEFT* button activates the Q3 coil contactor, so the Q3 phase contacts are connected in the *Moving LEFT* order.
- Pressing the *RIGHT* button activates the Q4 coil contactor, so the Q4 phase contacts are connected in the *Moving RIGHT* order.

- Pressing the *STOP* button breaks the connection with the Q3 and Q4 coils, so all contacts are returned to their original positions. The rotation stops.

Prevention against the contactors' simultaneous connection is ensured by means of their normally closed auxiliary contacts. So, the reverse rotation may be started only after pressing the *STOP* button [23].

### 1P 7W Manual Point Driver

The 1P 7W MDP wiring diagram is similar to 3P 7W/4W MDP. The circuit functionality is the same. There are some differences in wire connections. The reasons are the following:

- The direction of the 1P motor rotation depends on which phase of the output plug (P2) is connected to the input phase P1:L1. According to the 1P/3P 7W PMS manual, voltage presence on the P2:L1 means *Moving LEFT* and voltage presence on the P2:L2 means *Moving RIGHT* [7].
- The manual also states that the input N should be connected to the output P2:L3 [7].
- The inputs L2 and L3 are not needed for the circuit functionality.

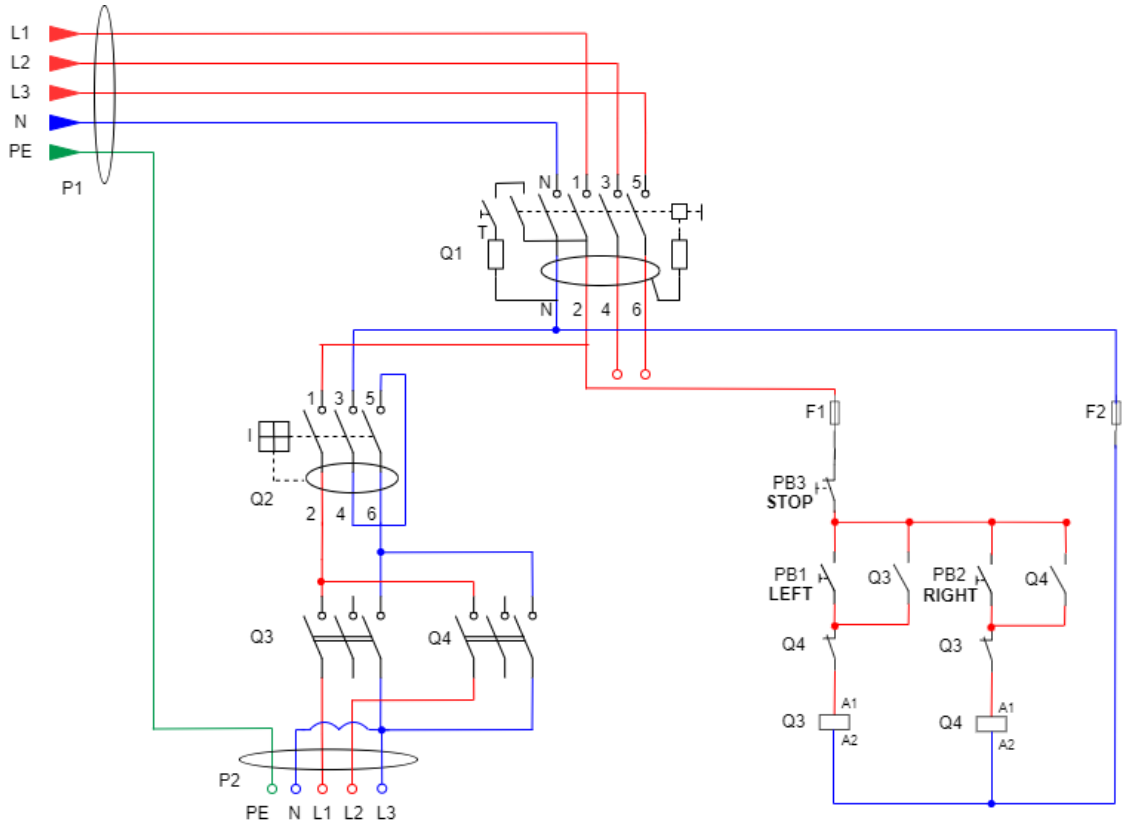



**Figure 4.7:** 1P 7W MPD wiring diagram (adapted from [24])

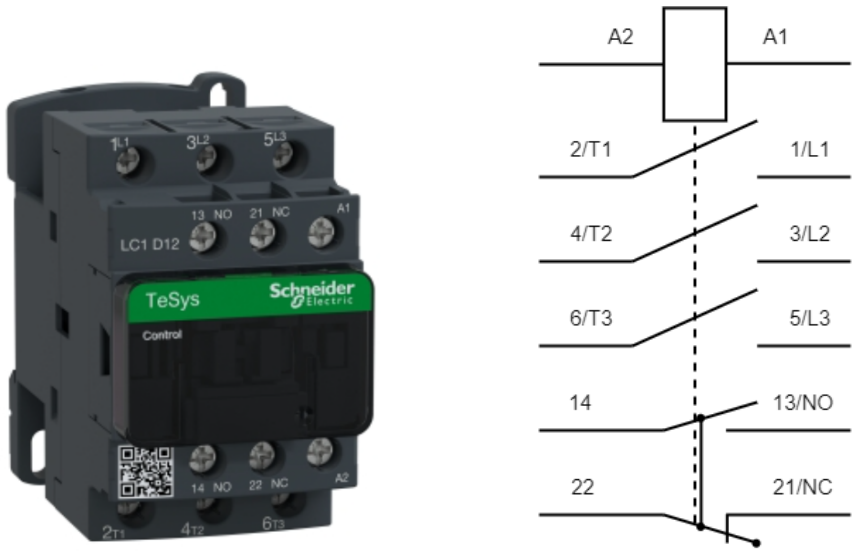
The functionality of each protective component used in the circuits is described in the subsection 4.3.3.

### 4.3.3 Components

The components described below were chosen for the device assembly.

#### Contactor

A contactor is used for switching an electrical power circuit. Since one MPD type should deal with the 3P motor, the contactor should contain minimally three poles. According to the schemes (Fig. 4.6, 4.7), it also should have one normally closed (NC) and one normally open (NO) contacts (Fig. 4.8b). The appropriate device is shown in Figure 4.8a. In the table 4.3 there are its parameters.




**(a)** LC1D12P7 look **(b)** LC1D12P7 wiring

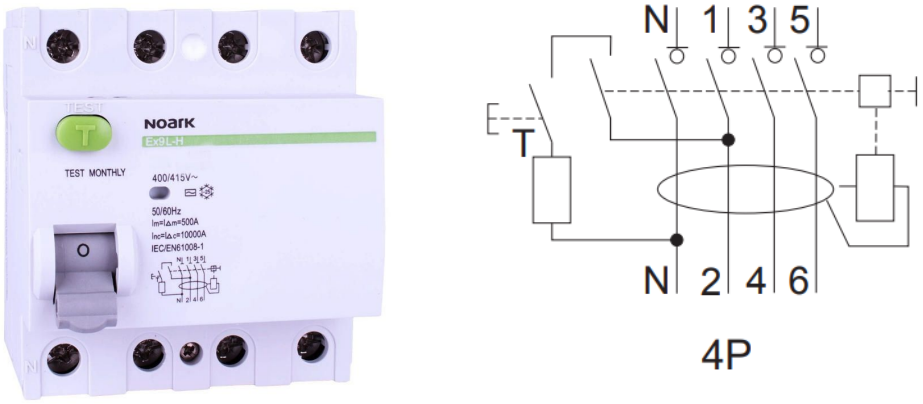
**Figure 4.8:** Contactor LC1D12P7 [25]

**Table 4.3:** Contactor LC1D12P7 parameters [25]

| Parameter | Value |
|---|---|
| Numbers of poles | 3 |
| Auxiliary contacts | NC, NO |
| Rated current | 12 A |
| Coil voltage | 230 V |

#### Residual-current circuit breaker

A residual-current circuit breaker serves to protect the electrical circuit. The circuit will be disconnected in case part of the incoming current leaks out of it. The chosen breaker is 3P. It is shown in Figure 4.12 and its parameters are in Table 4.4.




**(a)** EX9L-N look **(b)** EX9L-N wiring

**Figure 4.9:** Residual-current circuit breaker EX9L-N [26]

**Table 4.4:** Residual-current circuit breaker EX9L-N parameters [26]

| Parameter | Value |
|---|---|
| Numbers of poles | 4 |
| Rated voltage | 400 V |
| Rated current | 16 A |
| Frequency | 50 Hz |

### Circuit breaker

A circuit breaker is a device providing protection against an overcurrent. In case of an overcurrent, the breaker will disconnect the circuit and protect it from damage. The appropriate 3P circuit breaker is shown in Figure 4.10. Its parameters are in Table 4.5


**Figure 4.10:** Circuit breaker A9F06310 [27]

**Table 4.5:** Circuit breaker A9F06310 parameters [27]

| Parameter | Value |
|---|---|
| Numbers of poles | 3 |
| Rated current | 10 A |
| Tripping characteristic | B |
| short circuit resistance | 10 kA 240/415V |

### Fuses

Fuses were used as an additional overcurrent protection of the electrical circuit. They contain a fusible wire heated up when current flows through it. An overcurrent will remelt the wire, and the circuit will be interrupted. The chosen fuse is in Figure 4.11, and its characteristics are in Table 4.6.

**Figure 4.11:** Fuse OMEGA CF520310 [28]

**Table 4.6:** Fuse OMEGA CF520310 parameters [28]

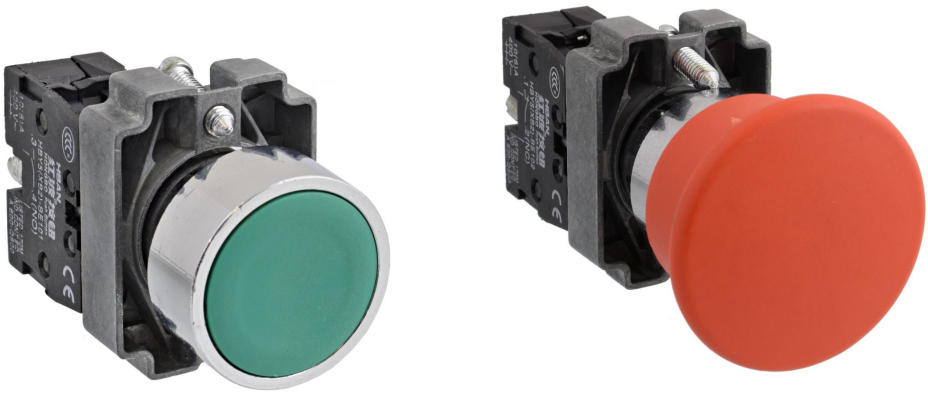
| Parameter | Value |
|---|---|
| Rated current | 10 A |
| Characteristics | F (Flink) |
| Fuse body | Glass |

### Buttons

Both circuits contain three push-button switches: two with an NO contact to start rotation to the left/right (Fig. 4.12a) and one with an NC contact to stop the rotation (Fig. 4.12b). Their electrical characteristics are in Table 4.7.

**Table 4.7:** Push-button switches parameters [29] [30]

| Parameter | Value |
|---|---|
| Operating voltage | 400 V |
| Switching current | 10 A |



**(a)** Green push button HBY5-10/G [29]

**(b)** Red push button HBY5-01M/R 40mm [30]

**Figure 4.12:** Push-button switches

### Cables

The device is powered by a three-phase 400 VAC socket, and its output goes to the input plug of the simulator. So, I have mounted a plug on the end of the MPD input cable and the socket on the end of the MPD output cable (Fig. 4.13). Parameters of the chosen plug and socket are in Table 4.8.

**Figure 4.13:** Input (right side) and output (left side) MPD cables

**Table 4.8:** Plug and socket parameters [31] [32]

| Parameter | Value |
|---|---|
| Number of poles | 5 |
| Rated voltage | 400 V |
| Rated current | 16 A |

### Housing

The plastic panel shown in Figure 4.14 was chosen as housing for the components and wiring. It contains a mounting rail and a terminal block. The panel can withstand heat up to 85°C [33]. Fuse holders for a panel and cable glands were also acquired.


**Figure 4.14:** Panel 3959 AcquaCOMBI IP65 [33]

## 4.3.4 Realization and testing

All the described components were placed in a box in such a way as shown in Figure 4.15. The circuit breakers are located on the mounting rail under the plastic window, so their switches are available for the users. The contactors are on the additional mounting rail, which I have installed in the panel. Both rails were grounded. For the buttons, fuses, and cables, I have drilled holes in the box and installed the components there (Fig. 4.16). Wiring was provided according to the schemes in Figures 4.6, 4.7.

After the complete assemblage of both MPDs (Fig. 4.17), revision and testing were provided for each device. The PMS reaction to pushing the *LEFT*/*RIGHT* button

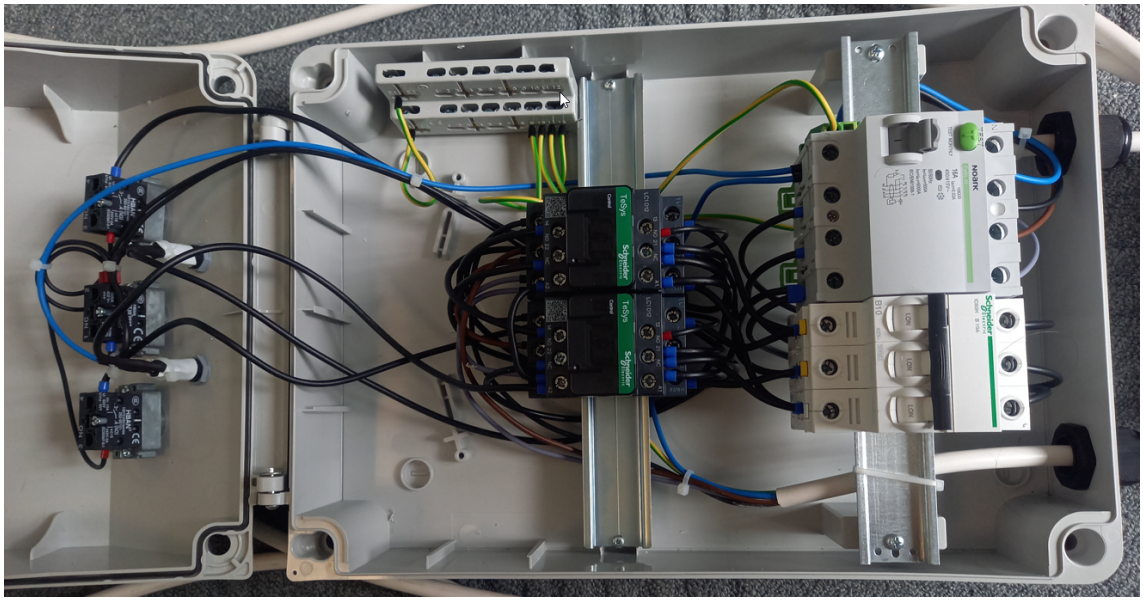


**Figure 4.15:** 3P 4W/7W MPD wiring

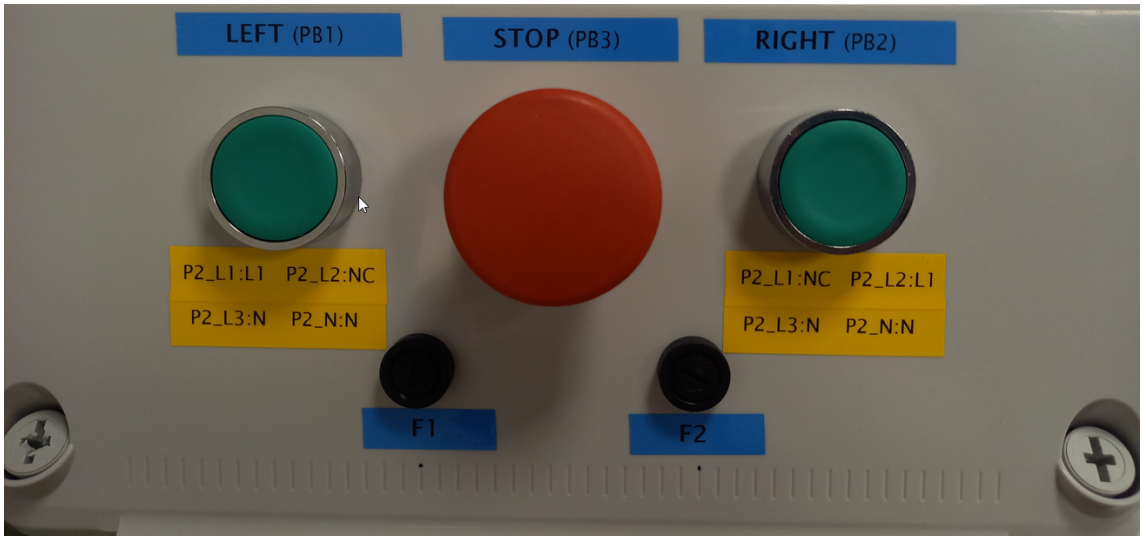



**Figure 4.16:** 1P 7W MPD control panel

corresponds to the expected behavior. Rotation stops after pushing the *STOP* button. Pushing a direction button during rotation does nothing. Reverse rotation is possible after pushing the *STOP* button only. Devices meet all the safety and functionality requirements described in Subsection 4.3.1. The MPD for the 3P motor is also compatible with *2PMS-3P4W*.


**Figure 4.17:** 1P 7W MPD (left side) and 3P 4W/7W MPD (right side)

## 4.4 2PMS-1P3P7W testing

This section documents and summarizes such phases of the product development process as testing and verification for *2PMS-1P3P7W*.

### 4.4.1 Functional testing and verification

MPD described in the previous subsection has been created for the 7W PMS testing. The device serves to simulate the function of the point controller connected to PMS. MPD was used for providing some functional tests.

The developed application *PMS Control Application* described in Chapter 3 was also used for the testing:

- the *Setup* tab - for sending Modbus messages to tested PMS;
- the *Observer* and *Logger* tabs - for control of the PMS Modbus outputs.

For the verification of requirements fulfillment, all functional tests were directly mapped to the tested product specifications described in Section 2.2. For safety, all tests results were always checked at two interfaces:

- manually at the user interface;
- semi-automatically at the Modbus interface via *PMS Control Application*.

Test cases that use only the Modbus TCP interface were automatized and added to *PMS Control Application* as an additional *Test* tab (Fig. 3.13). They automatically provide a check of the Modbus expected results.

The set of tests was created and applied to different configurations of 7W PMS:

- 1P mode:
  - the first PM;
  - the second PM.
- 3P mode:
  - the first PM;
  - the second PM.

Preconditions and expected results were defined for each test case.

This subsection contains the summarized set of test cases (Table 4.9) and describes issues that appeared during testing. The details are in the attached file *7w_pms_tests.xlsx*.

**Table 4.9:** *2PMS-1P3P7W* test cases

| Test case | Test interfaces | Test tools |
|---|---|---|
| Force EPL/EPR | Modbus TCP,<br>User interface | *PMS Control Application* |
| Trail from EPL/EPR | Modbus TCP,<br>User interface | *PMS Control Application* |
| Throw from EPL to EPR/from EPR to EPL | Test automation interface (control voltage connector) | MPD |
| Throw from EPL to EPR/from EPR to EPL with Jamm | Test automation interface: control voltage connector,<br>Modbus TCP,<br>User interface | MPD,<br>*PMS Control Application* |
| Set valid/invalid SWTI/EPRED | User interface,<br>Modbus TCP | *PMS Control Application* |
| Set CVCU to True/False | Modbus TCP | *PMS Control Application* |
| Switch PMS mode to 1P/3P | User interface,<br>Test automation interface: 24 V interface | Voltage source |
| Set PDOVRD to True/False | Modbus TCP | *PMS Control Application* |
| With PDOVRD on set S2+/S2-/S1-/S1+ to True/False | Modbus TCP | *PMS Control Application* |
| With PDOVRD on set S2- and S1+ to True, S2+ and S1- to False | Modbus TCP | *PMS Control Application* |

**Table 4.9: *2PMS-1P3P7W* test cases**

| Test case | Test interfaces | Test tools |
|---|---|---|
| With PDOVRD on set S2-, S1+, S2+ and S1- to False | Modbus TCP | *PMS Control Application* |
| With PDOVRD on set S2- and S1- to True, S2+ and S1+ to False | Modbus TCP | *PMS Control Application* |
| Set PDOVRD to True, force EPL/EPR, set PDOVRD to False | Modbus TCP | *PMS Control Application* |
| Set PDOVRD to True, Trail, set PDOVRD to False | Modbus TCP | *PMS Control Application* |
| Set Jamm and Trail to True | Modbus TCP | *PMS Control Application* |
| Check 7W indication | - | - |
| Check 1P indication | - | - |
| Check reduced 24 V interface | Test automation interface: 24 V interface | Voltage source |
| Backwards compatibility of the Modbus and configuration interfaces with *2PMS-3P4W* | - | - |

**Issues**

- **Set invalid SWTI/EPRED**
  The valid SWTI/EPRED values range is from 0 to 32767 ms. The test consisted of two test cases for each of these parameters: set a value less than 0; set the value greater than 32767. The 7W PMS manual does not describe an exact reaction to attempts to set invalid SWTI/EPRED values. So, I considered the absence of a reaction (keeping an old valid value) as a proper PMS behavior. However, the Modbus holding registers were rewritten to the entered invalid values. The user interface showed other invalid values less than 0. The test failed for all the PMS configurations.

- **Switch PMS mode to 1P/3P**
  This test has passed for all the PMS configurations with the following comments:

  – When PM is booted in the 1P mode, the user interface parameter is1P is False. So, it is impossible to switch the PM mode to 3P without setting this parameter to True before the switching.

  – Modbus TCP interface detects the 1P mode immediately with no problems.

- **Backwards compatibility of the Modbus and configuration interfaces with 2PMS-3P4W**
  A comparison of the mapped Modbus signals stated in Table 11 and Table 12 has indicated that not all the signal values of 7W PMS correspond to the values of 4W PMS. The changes are shown in Table 4.10.

**Table 4.10:** Modbus TCP replacements

| 2PMS-3P4W value | 2PMS-1P3P7W value | Signal name |
|---|---|---|
| 8204 | 8265 | Point 1, EPR |
| 8205 | 8266 | Point 1, EPL |
| 8206 | 8268 | Point 2, EPR |
| 8207 | 8269 | Point 2, EPL |
| 8208 | 8264 | Point 1, COVP |
| 8209 | 8267 | Point 2, COVP |
| 1 | 2 | Point 2, SWTI |
| 2 | 4 | Point 1, EPRED |
| 3 | 6 | Point 2, EPRED |

### 4.4.2 Summary

The testing of 7W PMS has turned out in the following way:

- Several problems were found and documented.
- Some expected states were defined by the tester because they have no exact definitions in the documentation.
- Documentation was also updated by the developers during the testing after noticing some errors.

The next step is discussing test results with the development team and creating a plan to improve the product and its documentation.

# Conclusion

The main master's thesis goal was to demonstrate some phases of the product development lifecycle on two examples:

- the SW application (product definition, implementation, and tests);
- the device simulating PM (system tests).

The first step to achieving this goal was to collect and examine the necessary documentation:

- PMS manuals explaining the principles of the product's functionality;
- CENELEC standards for the safe development and testing of the products.

After studying the materials and extracting all the needed information, the practice part of the project started. The first purpose was to develop the application (*PMS Control Application*) with GUI for the interaction with PMSs of two types (*2PMS-3P4W* and *2PMS-1P3P7W*) communicating with them by Modbus TCP. The requirements were defined by my colleagues from Siemens Mobility s.r.o. The application was then implemented according to them and expanded by adding the further functionality of testing the PMSs' Modbus interfaces. The created documentation and user manual contain all information necessary for working with the application.

The next step was to test the application according to one of the studied CENELEC norms - *EN 50128*. Several techniques were chosen from the following test methods:

- functional testing;
- static analysis;
- test coverage for code.

One technique required a tool simulating the behavior of a system driven by SW under test. So, the application reproducing the PMS functions was implemented and used for the testing. The chosen combination of techniques detected several SW problems. Some of them were fixed, and the rest were analyzed.

The last part of the project was to provide requirement testing of 7W PMS according to *EN 50129*. For convenient testing, the driving PMS HW tool (MPD) was created. This tool was used for the tests associated with starting the PMS's motors. Besides MPD, PMS interfaces and *PMS Control Application* were used for the testing, including the check of test results. All the detected problems were documented.

All the test results obtained while working on this master's thesis will be used to improve the tested products:

- *PMS Control Application* problems will be fixed by me;
- 7W PMS documented issues will be discussed with the development team.

# Bibliography


1. Rail Switch Makes Trains Change Tracks Safely | Railway Turnout Frog. *Anyang General International Co., Ltd. [online].* 2022. Copyright © [cit. 29.06.2022]. Available also from: `http://www.railroad-fasteners.com/railroad-switch.html`.
2. Point operating systems. *Siemens Mobility. [online].* 2022. Copyright © [cit. 29.06.2022]. Available also from: `https://www.mobility.siemens.com/ch/en/portfolio/rail/automation/wayside-crossing-and-on-board-products/point-operating-systems.html`.
3. Railway Interlocking: how does it work? *railwaysignalling.eu | walk the rail talk. [online].* 2014. Copyright © [cit. 29.06.2022]. Available also from: `https://www.railwaysignalling.eu/railway-interlocking-principles`.
4. S 700 K point machine. *Siemens AG. [online].* 2008. Copyright © [cit. 29.06.2022]. Available also from: `https://assets.new.siemens.com/siemens/assets/api/uuid:a0351b96-65f9-43c8-b976-45eee5d9d1d5/s-700-k.pdf`.
5. Maintenance Instructions for Electric Point Machine. *Centre for Advanced Maintenance Technology. [online].* 2020. Copyright © [cit. 29.06.2022]. Available also from: `https://rdso.indianrailways.gov.in/works/uploads/File/Maintenance%5C%20Instructions%5C%20for%5C%20Electric%5C%20Point%5C%20Machine_March%5C%202020.pdf`.
6. 2PMS-3P4W Design Specification and Manual. *Siemens s.r.o.* 2020.
7. Manual for Point Machine Simulator. *Siemens Mobility GmbH.* 2022.
8. MODBUS Messaging on TCP/IP Implementation Guide V1.0b. *Modbus Organization. [online].* 2006, 2–8. Copyright © [cit. 20.02.2022]. Available also from: `https://www.modbus.org/docs/Modbus_Messaging_Implementation_Guide_V1_0b.pdf`.
9. PyModbus - A Python Modbus Stack. *Sanjay Revision. [online].* 2017. Available also from: `https://pymodbus.readthedocs.io/en/latest/readme.html`.
10. tkinter — Python interface to Tcl/Tk. *Python Software Foundation. [online].* 2022. Copyright © [cit. 01.03.2022]. Available also from: `https://docs.python.org/3/library/tkinter.html`.
11. AUTALAN, Alejandro. Welcome to Pygubu! *GitHub, Inc. [online].* 2016. Copyright © [cit. 01.03.2022]. Available also from: `https://github.com/alejandroautalan/pygubu`.
12. HEESCH, Dimitri van. Doxygen. *Doxygen. [online].* 2022. Copyright © [cit. 01.03.2022]. Available also from: `https://www.doxygen.nl/index.html`.

13. DAVID CORTESI based on structure by Giovanni Bajo William Caban, based on Gordon McMillan's manual. PyInstaller Manual. *PyInstaller. [online]*. 2022. Copyright © [cit. 01.03.2022]. Available also from: `https://www.doxygen.nl/index.html`.

14. threading — Thread-based parallelism. *Python Software Foundation. [online]*. 2022. Copyright © [cit. 01.03.2022]. Available also from: `https://docs.python.org/3/library/threading.html`.

15. EN 50128. Railway applications - Gommunication, signalling and processing systems - Software for railway control and protection systems. 2011, 130p.

16. EN 50129. Railway applications - Communication, signalling and processing systems - Safety related electronic systems for signalling. 2018, 154p.

17. Review objectives Formal design reviews. *Pearson Education Limited. [online]*. 2004. . Copyright © [cit. 15.05.2022]. Available also from: `https://slidetodoc.com/oht-8-1-review-objectives-formal-design-reviews/`.

18. pylint 2.14.5. *The Python Package Index. [online]*. 2022. Copyright © [cit. 16.07.2022]. Available also from: `https://pypi.org/project/pylint/`.

19. Black | Improve Your Code with Pylint and Black. *Adafruit Learning System. [online]*. 2022. Copyright © [cit. 16.07.2022]. Available also from: `https://learn.adafruit.com/improve-your-code-with-pylint/black`.

20. Git - gitignore Documentation. *Git. [online]*. 2021. Copyright © [cit. 16.07.2022]. Available also from: `https://git-scm.com/docs/gitignore`.

21. GUIDO VAN ROSSUM Barry Warsaw, Nick Coghlan. PEP 8 – Style Guide for Python Code. *Python Enhancement Proposals. [online]*. 2013. Copyright © [cit. 16.07.2022]. Available also from: `https://peps.python.org/pep-0008/`.

22. BATCHELDER, Ned. Coverage.py. *Coverage.py 6.4.2 documentation. [online]*. 2022. Copyright © [cit. 17.07.2022]. Available also from: `https://coverage.readthedocs.io/en/6.4.2/`.

23. RUS, Dorinel. Forward-Reverse Direct ON Line (DOL) starting. *Automation-Electric. [online]*. 2016. Copyright © [cit. 24.06.2022]. Available also from: `https://automation-electric.com/pornirea-directa-rotatie-in-ambele-sensuri/`.

24. RUS, Dorinel. Forward-Reverse Single phase motor starting. *Automation-Electric. [online]*. 2016. Copyright © [cit. 24.06.2022]. Available also from: `https://automation-electric.com/category/single-phase-motor-starting-rotation-in-both-directions/`.

25. Technicky list | stykac 3P(3Z) 12A AC-3 440V pomocne kontakty 1Z+1V-civka 230V 50Hz | LC1D12P7. *Schneider-Electric. [online]*. 2022. Copyright © [cit. 24.06.2022]. Available also from: `https://www.se.com/cz/cs/product/download-pdf/LC1D12P7`.

26. Proudove chraniče Ex9L-N, 6 kA. *GM electronic. [online]*. 2022. Copyright © [cit. 24.06.2022]. Available also from: `https://www.gme.cz/data/attachments/dsh.614-193.1.pdf`.

27. Technicky list | modularni jistic iC60H - 3P - 10A - charakteristika B | A9F06310. *Schneider-Electric. [online]*. 2022. Copyright © [cit. 24.06.2022]. Available also from: `https://www.se.com/cz/cs/product/download-pdf/A9F06310`.

28. Pojistka sklen. 5x20 10,0A F rychla, vyp.schopnost 35A/250V. *VOGEL electric s.r.o. [online]*. 2022. Copyright © [cit. 24.06.2022]. Available also from: https://www.elektrotechnika-shop.cz/pojistka-sklen-5x20-100a-f-rychla-vypschopnost-35a250v.

29. Tlacítkovy spínac, 1 pol, OFF-(ON), 10A/400V, zeleny HBY5-10/G. *GM electronic. [online]*. 2022. Copyright © [cit. 24.06.2022]. Available also from: https://www.gme.cz/tlacitkovy-spinac-hby5-10-g.

30. Tlacitkovy spinac, 1 pol, ON-(OFF), 10A/400V, cerveny HBY5-01M/R 40mm. *GM electronic. [online]*. 2022. Copyright © [cit. 24.06.2022]. Available also from: https://www.gme.cz/tlacitkovy-spinac-hby5-01m-r-40mm.

31. Vidlice 5P 16A 400V IP44 6h IVN 1653, sroubova. *VOGEL electric s.r.o. [online]*. 2022. Copyright © [cit. 24.06.2022]. Available also from: https://www.elektrotechnika-shop.cz/vidlice-5p-16a-400v-ip44-6h-ivn-1653-sroubova.

32. Zasuvka vestavna 23331 IP44/400V/16A/5P 6h, sikma 10° - SpeedPRO. *Famatel - CZ s.r.o. [online]*. 2019. Copyright © [cit. 24.06.2022]. Available also from: https://www.elektrotechnika-shop.cz/vidlice-5p-16a-400v-ip44-6h-ivn-1653-sroubova.

33. Skrin 3959 AcquaCOMBI IP65, 12 modulu, 398x266x153mm. *Famatel - CZ s.r.o. [online]*. 2019. Copyright © [cit. 24.06.2022]. Available also from: https://www.eshop.famatel.cz/skrin-3959-acqua-combi-ip65-12-modulu-390x265x150mm#tb1=2.

# Attachments

## A Modbus TCP mapping

The Modbus TCP mapping tables (Tab. 11, 12) contain PMS signals and their directions concerning the Modbus coil and register values.

### A.1 2PMS-3P4W

**Table 11:** *2PMS-3P4W* Modbus TCP mapping [6]

| **Modbus address** | **Read/Write** | **Signal** |
|---|---|---|
| 8308 | W | Point 1, Trail |
| 8309 | W | Point 1, Jamm |
| 8310 | W | Point 1, FEPL |
| 8311 | W | Point 1, FEPL |
| 8312 | W | Point 1, CVCU |
| 8313 | W | Point 1, ELT |
| 8314 | W | Point 2, Trail |
| 8315 | W | Point 2, Jamm |
| 8316 | W | Point 2, FEPL |
| 8317 | W | Point 2, FEPL |
| 8318 | W | Point 2, CVCU |
| 8319 | W | Point 2, ELT |
| 8204 | R | Point 1, EPR |
| 8205 | R | Point 1, EPL |
| 8206 | R | Point 2, EPR |
| 8207 | R | Point 2, EPL |
| 8208 | R | Point 1, COVP |
| 8209 | R | Point 2, COVP |
| 8256 | R | Point 1, MRPO |
| 8257 | R | Point 1, MLPO |
| 8258 | R | Point 2, MRPO |
| 8259 | R | Point 2, MLPO |
| 0 | R/W | Point 1, SWTI |
| 1 | R/W | Point 2, SWTI |
| 2 | R/W | Point 1, EPRED |
| 3 | R/W | Point 2, EPRED |

## A.2 2PMS-1P3P7W

**Table 12:** *2PMS-1P3P7W* Modbus TCP mapping [7]

| Modbus address | Read/Write | Signal |
|---|---|---|
| 8256 | R | Point 1, MRPO |
| 8257 | R | Point 1, MLPO |
| 8258 | R | Point 2, MRPO |
| 8259 | R | Point 2, MLPO |
| 8160 | R | PMS, is7W |
| 8261 | R | Point 1, is1P7W |
| 8262 | R | Point 2, is1P7W |
| 8264 | R | Point 1, COVP |
| 8265 | R | Point 1, EPR |
| 8266 | R | Point 1, EPL |
| 8267 | R | Point 2, COVP |
| 8268 | R | Point 2, EPR |
| 8269 | R | Point 2, EPL |
| 8298 | W | Point 1, PDOVRD |
| 8299 | W | Point 1, OVS2P |
| 8300 | W | Point 1, OVS2M |
| 8301 | W | Point 1, OVS1M |
| 8302 | W | Point 1, OVS1P |
| 8303 | W | Point 2, PDOVRD |
| 8304 | W | Point 2, OVS2P |
| 8305 | W | Point 2, OVS2M |
| 8306 | W | Point 2, OVS1M |
| 8307 | W | Point 2, OVS1P |
| 8308 | W | Point 1, Trail |
| 8309 | W | Point 1, Jamm |
| 8310 | W | Point 1, FEPL |
| 8311 | W | Point 1, FEPL |
| 8312 | W | Point 1, CVCU |
| 8313 | W | Point 1, ELT |
| 8314 | W | Point 2, Trail |
| 8315 | W | Point 2, Jamm |
| 8316 | W | Point 2, FEPL |
| 8317 | W | Point 2, FEPL |
| 8318 | W | Point 2, CVCU |
| 8319 | W | Point 2, ShuntL2L3 |
| 8192 | R | Point 1, ShuntedL1L2 |
| 8193 | R | Point 1, ShuntedL2L3 |
| 8194 | R | Point 2, ShuntedL1L2 |
| 8195 | R | Point 2, ShuntedL2L3 |
| 8196 | R | Point 1, LoadL1 |
| 8197 | R | Point 1, LoadL2 |
| 8198 | R | Point 1, LoadL3 |

**Table 12: *2PMS-1P3P7W* Modbus TCP mapping [7]**

| Modbus address | Read/Write | Signal |
|---|---|---|
| 8199 | R | Point 1, LoadN |
| 8200 | R | Point 1, S2P |
| 8201 | R | Point 1, S2N |
| 8202 | R | Point 1, S1N |
| 8203 | R | Point 1, S1P |
| 8196 | R | Point 2, LoadL1 |
| 8197 | R | Point 2, LoadL2 |
| 8198 | R | Point 2, LoadL3 |
| 8199 | R | Point 2, LoadN |
| 8200 | R | Point 2, S2P |
| 8201 | R | Point 2, S2N |
| 8202 | R | Point 2, S1N |
| 8203 | R | Point 2, S1P |
| 0 | R/W | Point 1, SWTI |
| 2 | R/W | Point 2, SWTI |
| 4 | R/W | Point 1, EPRED |
| 6 | R/W | Point 2, EPRED |

# B Directory tree

A structure of the attached files mentioned in the thesis and their descriptions are shown below.