Simulation of Industrial Processes using I/O Factory and UniPi

Technical Report, FIT BUT

Petr Matoušek, Ján Pristaš, Mária Masárová

Technical Report no. FIT-TR-2020-09
Faculty of Information Technology, Brno University of Technology

Last modified: 03.03.2021

Contents

Abstract

Industrial networks form a special class of computer networks that employ specific devices, communication protocols and communication patterns. In order to study industrial networks, it is important to have an access to industrial devices and their communication. This is, however, not easy to implement in university environment. Real devices are expensive, require regular maintenance and are available to few operators. As alternative to the real industrial environment, it is possible to combine real devices with emulated environment.

This study shows how it is possible to create an industrial network with Modbus protocols and real devices like PLCs and RTUs together with emulator of physical processes using I/O Factory software. In this study we show how to build a virtual factory that includes a simple assembly line and the sorting conveyor controlled by PLCs.

Introduction

As a members of the TRACTOR[1] (TRaffic Analysis and seCuriTy OpeRations for ICS/SCADA) project at Faculty of Information Technology, Brno University of Technology, our task was to create a testing environment for Modbus TCP communication protocol, which would allow testing of various types of attacks on SCADA networks.

Nowadays, great emphasis is placed on the automation of various industrial systems. However, the more it is automated, the more number of components that need to be interconnected increases. With a large number of these devices, it is impossible to communicate on the physical layer, and therefore their mutual communication had to be transferred to the IP layer. However, moving to the IP layer gives attackers new ways to break into the system, which we would like to prevent, as these systems are often a part of the critical infrastructure and their disruption could cause major damage (power plant - interruption of electricity supply to thousands of households, factory - production shutdown, etc.) [2].

Our job was to create a testing environment where Modbus TCP communication can be created, captured and analysed. Our testing environment simulate real world production line, where single components communicate via Modbus TCP protocol. It also allows to create several types of attacks and analyse how the system would behave.

Section 1 describes creation of two types of production lines in Factory I/O simulation program. The hardware part, which includes all physical components and their interconnections, is described in Section 2. Software part is described in Section 3, where main focus is on scripts for automatic control of our lines. One line also can be controlled via HMI (Human Machine Interface), where user can control several part of line manually.

[1]https://www.fit.vut.cz/research/project/1321/.en

Chapter 1

Simulation of the factory

Our production lines are simulated by Factory I/O[1] simulation software. Factory I/O is a software for 3D factory simulation that allows to build a virtual factory using common industrial parts and control each component of this factory production line directly with PLCs connected to computer.

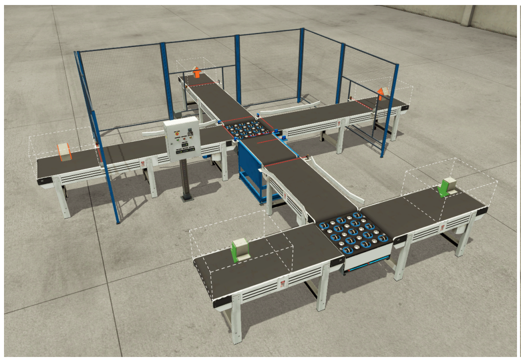
Factory I/O includes a list of example scenes which are inspired by typical industrial systems. In this project we choose sorting line, which sorts items by weight and assembly line, which creates one object from two parts (base and lid). First scene uses both analog and digital inputs/outputs, which is what we need for our future testing scenarios. This scene was slightly modified to use more pins from the PLCs, but the modifications couldn't be very large because we were limited by number of PLCs, that we could use. The appearance of this sorting line can be seen in Figure 1.1a. Second scene is much simpler and uses only digital inputs/outputs. This scene was created as an exercise for students and can be seen in Figure 1.1b.

Factory I/O software displays a 3D visualization of the production line, where after starting the line, the user can monitor the operation of the line in real time. However, without virtual or physical PLCs the software supports only manual control over the production line which is not always sufficient. For automatic operation there have to be added some PLCs. Factory I/O software supports several PLCs from various brands. In this testbed Advantech USB-4750 and Advantech USB-4704 are used. First one is used for digital inputs/outputs and second for digital and analog inputs/outputs.

A detailed description how to work with Advantech PLCs in the Factory I/O program can be seen on the factoryio website[2]. In case of problems, it is necessary to check whether the drivers were automatically downloaded when connecting the Advantech PLCs to the computer. If not, they can be

[1]https://factoryio.com/

[2]https://docs.factoryio.com/manual/drivers/advantech/


(a) Sorting line

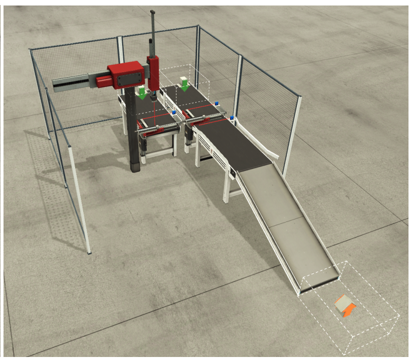

(b) Assembly line

Figure 1.1: Production lines in Factory I/O

downloaded additionally from the advantech website[3].

In Factory I/O software, there is a menu on top bar of application, where *File* menu item can be found. This item contains *Drivers* item, where Advantech PLCs can be attached to the software. Then, the individual sensors and actuators can be mapped to specific input or output pins of the Advantech PLCs. In Figure 1.2 you can see how the sensors and actuators of our sorting line are connected to the PLCs pins. In Figure 1.3 you can see mapping of sensors and actuators of our assembly line.

As you can see in the pictures, sensors are mapped on the left side and represent the outputs of the Advantech PLCs, while actuators are mapped on the right side as the inputs of Advantech PLCs. Sensors and actuators are assigned with a name in Factory I/O software for better understanding and work with them. It is not necessary to map all sensors and actuators, but it is necessary to keep in mind that those that are not mapped will not be possible to control using PLCs. As you can see, our lines do not use all sensors and actuators. Firstly, it is because some are not necessary, but also because we could map them to Advantech PLCs, but we do not have a sufficient number of UniPi PLCs, which means that we would still not be able to control them. Difference between these PLCs is described in next section.

[3]https://www.advantech.com/support/details/driver?id=1-13L33UP

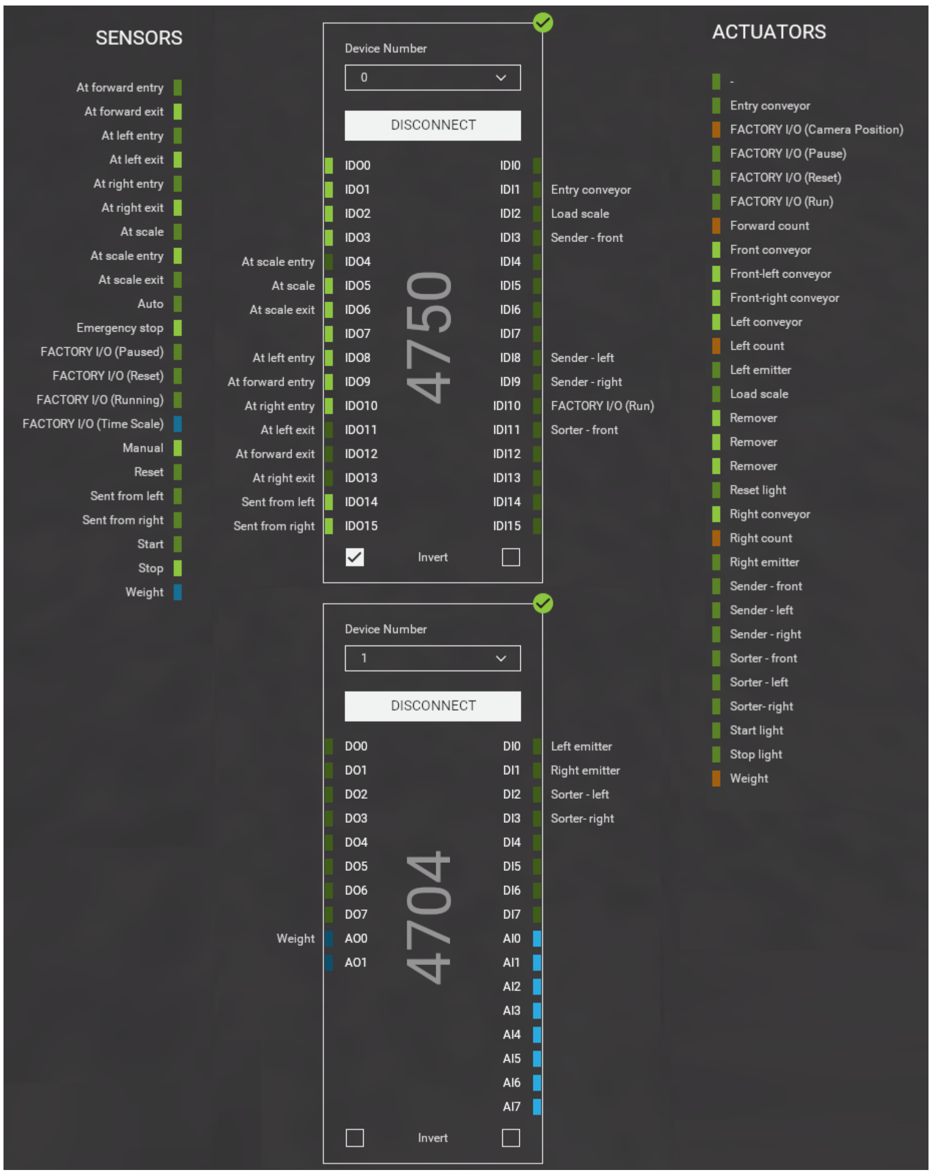
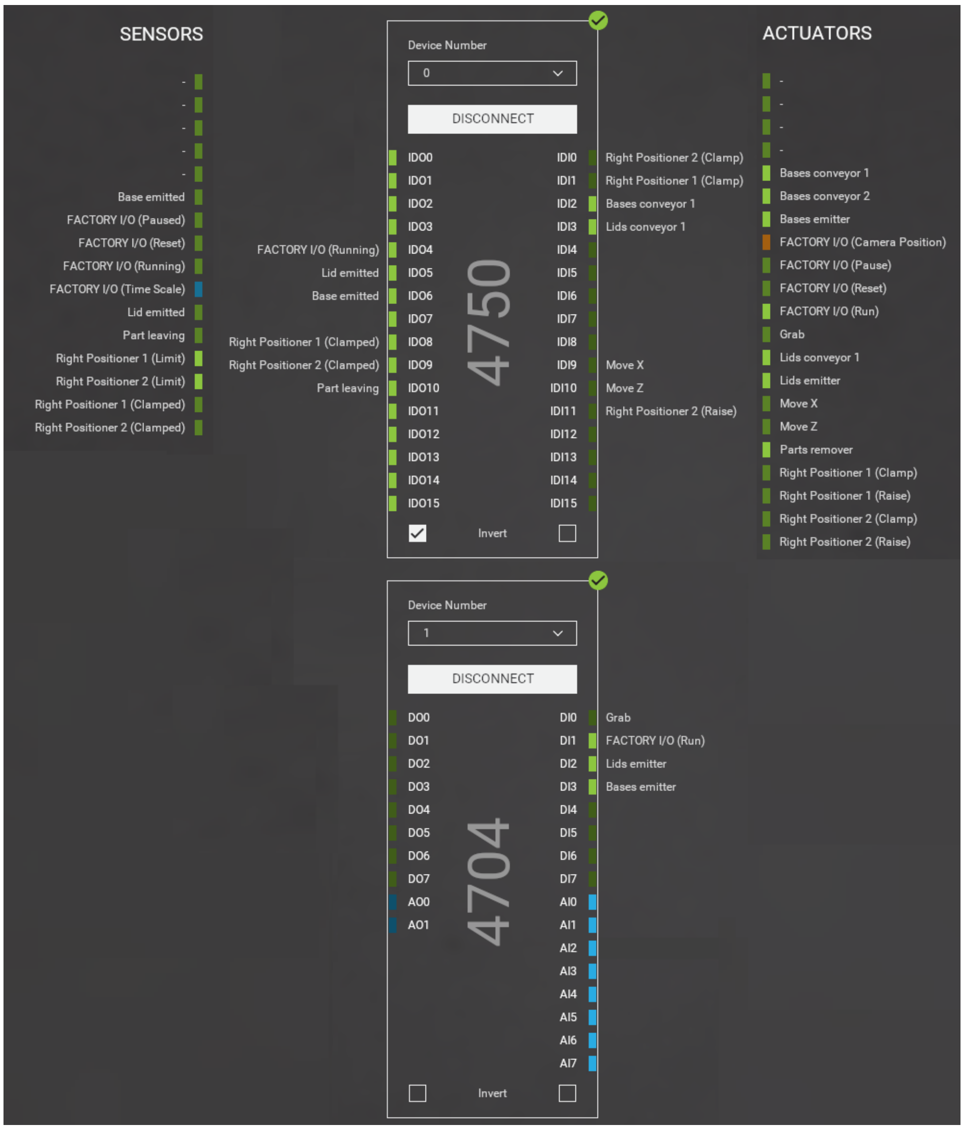


Figure 1.2: Drivers of Sorting line



SENSORS

-
-
-
-
-
Base emitted
FACTORY I/O (Paused)
FACTORY I/O (Reset)
FACTORY I/O (Running)
FACTORY I/O (Time Scale)
Lid emitted
Part leaving
Right Positioner 1 (Limit)
Right Positioner 2 (Limit)
Right Positioner 1 (Clamped)
Right Positioner 2 (Clamped)

ACTUATORS

-
-
-
-
Bases conveyor 1
Bases conveyor 2
Bases emitter
FACTORY I/O (Camera Position)
FACTORY I/O (Pause)
FACTORY I/O (Reset)
FACTORY I/O (Run)
Grab
Lids conveyor 1
Lids emitter
Move X
Move Z
Parts remover
Right Positioner 1 (Clamp)
Right Positioner 1 (Raise)
Right Positioner 2 (Clamp)
Right Positioner 2 (Raise)

Device Number 0

DISCONNECT

4750

	IDO0	IDI0	Right Positioner 2 (Clamp)
	IDO1	IDI1	Right Positioner 1 (Clamp)
	IDO2	IDI2	Bases conveyor 1
	IDO3	IDI3	Lids conveyor 1
FACTORY I/O (Running)	IDO4	IDI4	
Lid emitted	IDO5	IDI5	
Base emitted	IDO6	IDI6	
	IDO7	IDI7	
Right Positioner 1 (Clamped)	IDO8	IDI8	
Right Positioner 2 (Clamped)	IDO9	IDI9	Move X
Part leaving	IDO10	IDI10	Move Z
	IDO11	IDI11	Right Positioner 2 (Raise)
	IDO12	IDI12	
	IDO13	IDI13	
	IDO14	IDI14	
	IDO15	IDI15	

✓ Invert ☐

Device Number 1

DISCONNECT

4704

DO0	DI0	Grab
DO1	DI1	FACTORY I/O (Run)
DO2	DI2	Lids emitter
DO3	DI3	Bases emitter
DO4	DI4	
DO5	DI5	
DO6	DI6	
DO7	DI7	
AO0	AI0	
AO1	AI1	
	AI2	
	AI3	
	AI4	
	AI5	
	AI6	
	AI7	

☐ Invert ☐

Figure 1.3: Drivers of Assembly line

Chapter 2

Hardware part

Our testing environment consists of two types of PLCs:

- *Advantech USB-4750*[1]*/ Advantech USB-4704*[2]
- *UniPi Neuron S103*[3]

As already mentioned, the Factory I/O software runs a simulation of the production line where sensors and actuators from this production line are mapped to the pins of Advantech PLCs. But since we wanted to create testing environment for Modbus TCP, we had to use another type of PLC, because Advantech PLCs doesn't support Modbus TCP communication.

Advantech PLCs are connected directly to computer via USB cables and Unipi PLCs are connected directly to Advantech I/O ports via copper wires. We need to use two types of PLCs, because Factory I/O doesn't support Unipi PLCs and Advantech PLCs cannot be configured as Modbus TCP servers. Unipi PLCs are build on *Raspberry Pi 3* platform and were configured as Modbus TCP servers. Unipi also provides *Opensource OS*[4] for Raspberry Pi which is booted to system through SD card. Configuration procedure for Unipi PLCs is described below.

[1]https://www.advantech.com/products/1-2mlkno/usb-4750/mod_43dfaaf0-a44c-4437-a8c8-0f7460c30b26

[2]https://www.advantech.com/products/1-2mlkno/usb-4704/mod_4d0800cc-f6fd-402a-9782-24cd0ffdaf42

[3]https://www.unipi.technology/unipi-neuron-s103-p93

[4]https://kb.unipi.technology/cs:files:software:os-images:00-start?tns

2.1 Unipi configuration

Each Unipi PLC was configured as Modbus TCP Server through SSH with commands:

1.
   ```
   echo "DAEMON_OPTS=--listen=0.0.0.0" |
   sudo tee -a /etc/default/unipi-modbus-tools
   > /dev/null
   ```

2.
   ```
   sudo systemctl restart unipitcp
   ```

SSH credentials:

- Login: unipi
- Password: unipi.technology

Physical wiring can be seen in Figure 2.1. There is also possible to see what the UniPi Neuron S103 PLCs looks like (blue) and what the Advantech USB PLCs looks like (black). The physical connection is made using copper wires. Wiring diagram, where is better illustration of connections, can be seen in Appendix A. As you can see, we created four circuit boards with pull-up resistors, because Advantech PLCs weren't able to provide enough output voltage to Unipi input ports[5]. Because of this connection, there is a need to invert output ports on Advantech PLCs in *Drivers* section of Factory I/O software.

As you can see on the pictures, our testbed consists of four UniPi Neuron PLCs and two Advantech PLCs. But we are using only three Unipi PLCs as we were limited by number of ethernet ports on our router. On Figure 2.2 you can see our Modbus TCP Client/Server topology, that we are using in the testbed.

Values of actuators in Factory I/O software can be controlled by changing their values on specific pins in those UniPi PLCs. Sensor values in Factory I/O software can be determined using UniPi by reading specific pins. Figure 1.2 and Figure 1.3 shows how the individual sensors and actuators are mapped to specific pins of Advantech PLCs in the Factory I/O software.

For example, in our Sorting line, the sensor named *At scale entry* is connected to Advantech USB-4750 PLC on pin IDO4. This pin is then connected via copper wire to PLC2, which is the name for one of the UniPi Neuron S103 PLCs, to pin DI1. This means that if the user wants to know the value of the *At scale entry* sensor, he needs to read the value of pin DI1 on PLC2. The same applies to actuators. For example, an actuator

[5]https://docs.factoryio.com/tutorials/wiring-diagrams/

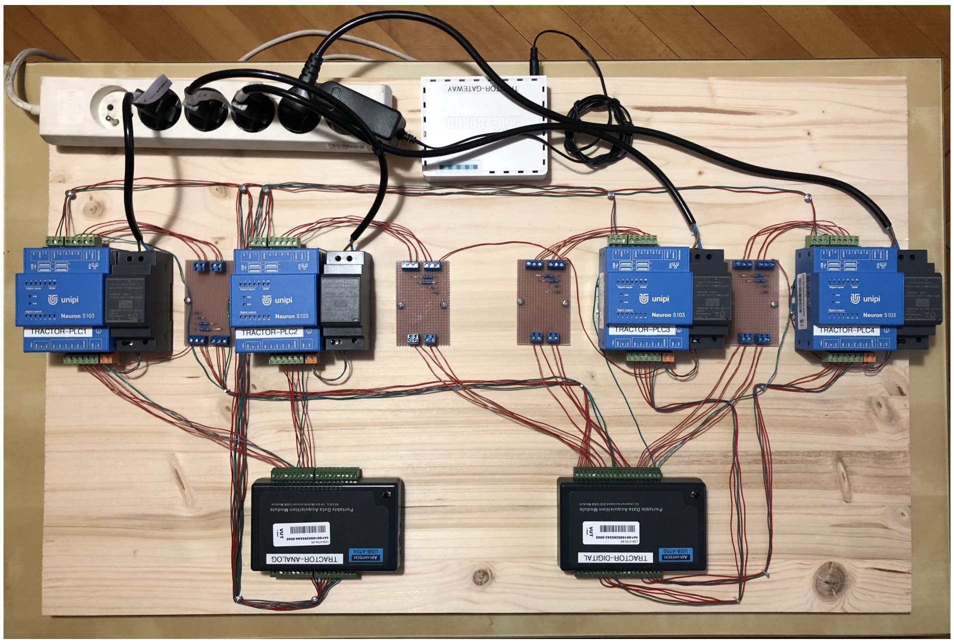


Figure 2.1: Physical wiring

called *Sorter - left* is connected to the Advantech USB-4704 to its pin DI2. Subsequently, this pin is connected to pin DO3 on PLC3. So if the user wants to change the value of the *Sorter - left* actuator, he has to change the value on pin DO3 on PLC3.

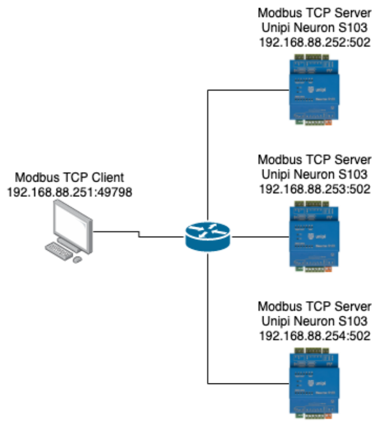


Figure 2.2: Modbus TCP Client/Server

Chapter 3

Software part

The software part of this project was to create scripts and HMI (Human machine interface) to control the lines simulated in Factory I/O software. This program (scripts and HMI) acts as a Modbus TCP client. The main task was the ability to capture communication between Modbus TCP client (program) and Modbus TCP servers (UniPi Neuron S103 PLCs). For our two lines, sorting line and assembly line, we created two separate control programs. First program, which control our sorting line, is more complex and also contains HMI. Second program, which control our assembly line, is simpler and serves mainly for study purposes.

Modbus TCP client was implemented in programming language Python version 3.7 with usage of pyModbusTCP library[1]. It is important to install python and some other python packages/libraries to your computer before starting to use our line control scripts and HMI. For assembly line, only python version 3.7 and pyModbusTCP library has to be installed. For sorting line, there has to be python version 3.7, pyModbusTCP library and PyQt5.

We used the PyCharm[2], from JetBrains company, to create these scripts and HMI that control lines. PyCharm is the Python IDE which offers community version that is open-source and free. PyQt5 and pyModbusTCP packages were download through this IDE.

PyQt5 is a comprehensive set of Python bindings for Qt v5. Qt is a cross-platform application development framework, which allows user to create graphical user interface programs. PyQt also offers QtDesigner, which is a GUI builder, that is used to create a GUI using the drag and drop method. When creating our HMI, we mainly relied on this manual[3]. QtDe-

[1]https://pymodbustcp.readthedocs.io/en/latest/index.html
[2]https://www.jetbrains.com/pycharm/
[3]https://doc.qt.io/qt-5/qtdesigner-manual.html

signer offers a number of labels, buttons, fields and more. However, it does not offer anything that would serve as a switch. Therefore, it was necessary to define the switch separately. The code defined on this page[4], which we have slightly modified for our purposes, served as a template. In order to use the switch in our project, we had to connect it to QtDesigner. We followed the instructions[5,6] that described how to connect a custom widget to QtDesigner. Once we had created the final design of our GUI in QtDesigner, it had to be converted to a python file that we could work with. This conversion of a .ui file to a .py file is done using the PyUIC package. For easier work with QtDesigner and PyUIC, it is possible to connect them to PyCharm, so that they can be launched with a simple click in the IDE. A description of the procedure is described in this manual[7]. The procedure for creating a GUI was then as follows:

1. Open PyCharm.
2. In *Tools → External Tools* in PyCharm, open QtDesigner.
3. Create a GUI according to your own design.
4. Save .ui file and turn off QtDesigner.
5. In *Tools → External Tools* in PyCharm, execute PyUIC, to convert .ui file to .py file.
6. Use this .py file as your main file and add anything that you need.

PyModbusTCP library gives access to Modbus TCP server through the *ModbusClient* object. To work with this object, it has to be initialized at first. Initialization can be performed in two ways. The first way is to create a ModbusClient object by entering the IP address and port of the server.

```
plc_2 = ModbusClient(host="192.168.88.252", port=502)
```

The second way is to create a ModbusClient object without specifying parameters and then supplement them using `host()` and `port()` functions. We used the second option in our implementation.

```
plc_2 = ModbusClient()
plc_2.host("192.168.88.252")
plc_2.port(502)
```

[4]https://stackoverflow.com/questions/56806987/switch-button-in-pyqt

[5]https://stackoverflow.com/questions/47259825/how-to-insert-video-in-ui-file-which-made-at-qt-designer/47273625#47273625

[6]https://www.riverbankcomputing.com/static/Docs/PyQt5/designer.html

[7]https://www.programmersought.com/article/44931884816/

Each Modbus TCP server needs its own ModbusClient object. Since we used three PLCs in our testbed, we had to create three ModbusClient objects, one for each PLC.

After initialization of the object, it is possible to access the pins of the server (Unipi PLC). For this purpose, the Modbus protocol provides a set of functions to read from and write to data sources (coils, registers, input registers, holding registers, etc.)[3]. The pyModbusTCP library allows valid Modbus commands to be sent by using these functions. Here are few examples of how to use these functions in code:

```
plc_2.write_single_coil(2, True)
plc_2.read_coils(4, 4)
plc_2.read_holding_registers(3, 1)
```

One UniPi Neuron S103 PLC has four digital inputs, four digital outputs, one analog input and one analog output. If we want to access values of these inputs/outputs using the functions in the pyModbusTCP library, it is necessary to know at which addresses the individual inputs and outputs are located so that they can be read or written to.

For accessing each of the registers and coils there are two possible methods. As each group features its own processor, all the registers of the given group are accessible through a unit (address) according to the Group number (i.e. 1 – 3) and at the same time through unit 0. So all registers/coils of the given product are accessible through unit 0. If the access through unit 0 is used, register numbers are shifted according to 100*(group_number – 1) formule. Thus, it is possible to use both methods.

For example, register 1 of the Group 1 is accessible through the unit 1 on the address 1 and through the unit 0 on the register 1 as well. Register 1 of the group 2 is accessible through the unit 2 on register 1 and through the unit 0 on the register 101[1].

Table 3.1 shows the individual register addresses, while Table 3.2 shows the coils. NR in the table is an abbreviation for number. These two tables are just a slice of the most important registers and coils, which we use in our testbed. Complete tables of registers and coils can be seen in Appendix B.

Based on the knowledge of these tables and an explanation of how to approach the individual inputs and outputs of UniPi Neuron S103 PLCs, we can explain the individual commands mentioned above. The first command would write the value to bit 2 (Digital output 1.3) of the PLC_2 and since it is connected via Advantech PLC to the actuator with name *Right emitter*, this function will turn on this emitter. If there was *False* instead of *True*,

Register number		Content	Bit NR
Unit 0	Unit x Reg. NR		
0	1 x 0	Digital input 1.1	0
		Digital input 1.2	1
		Digital input 1.3	2
		Digital input 1.4	3
1	1 x 1	Digital output 1.1	0
		Digital output 1.2	1
		Digital output 1.3	2
		Digital output 1.4	3
2	1 x 2	Analog output 1	
3	1 x 3	Analog input 1	
4	1 x 4	Analog input 2	
5	1 x 5	VrefInt	
1009	1 x 1009	Vref	

Table 3.1: Registers - group 1

the function would cause the emitter to shut down. Second command reads all digital inputs of PLC_2 and returns an array with their values, so we know values of sensors *At scale entry*, *At scale* and *At scale exit*. The last digital input (Digital input 1.4) is not connected to any sensor in Factory I/O software.

The last function reads the register values at address 3 and returns an array with these values. At index 0, there is then a value of the analog input 1. Analog input usually serves for reading Voltage 0-10 V. In Factory I/O software, this value is named *Weight*. However, the value returned by the UniPi Neuron PLC does not correspond to the actual object weight displayed in the Factory I/O software. Weight in Factory I/O software is in range 0-10 V, actual weight of boxes and object is in range 0-20 kg, but weight values returned by UniPi Neuron PLC are in range 0-4096. This value is in volts, so if we want to know the actual weight of the box, we have to use a formula to convert. For correct measurement of analog input, it is necessary to do a correction of converted value with reference voltage stored in the processor and also a correction of other coefficients read directly from corresponding registers. For doing so there is a following formula of voltage calculation:

$$U_{AI1_{true}} = \left(3.3 * \frac{V_{ref}}{V_{refInt}}\right) * 3 * \frac{V_{AI}}{4096} * \left(1 + \frac{AI1_{vdev}}{10000}\right) + \frac{AI1_{voffset}}{10000} [V]$$

Coil number		Content
Unit 0	Unit x Coil	
0	1 x 0	Digital output 1.1
1	1 x 1	Digital output 1.2
2	1 x 2	Digital output 1.3
3	1 x 3	Digital output 1.4
4	1 x 4	Digital input 1.1
5	1 x 5	Digital input 1.2
6	1 x 6	Digital input 1.3
7	1 x 7	Digital input 1.4

Table 3.2: Coils - group 1

3.1 Sorting line control

Four files written in Python are used to control our sorting line. They are:

- *tractorHMI.py* - represents HMI
- *tractor.py* - controls line
- *init_script.py* - initializes values
- *qswitchbutton.py* - defines the appearance of the switch

As already mentioned, the sorting line in the Factory I/O software can be controlled either manually in the software or it can be controlled using connected PLCs. A Python script, *tractor.py*, has been implemented for automation of the sorting line.

The script runs in seven threads, where each thread controls a certain part of the line. Since we can't allow more than one thread to write to one pin of PLC at a time, we decided to set aside one special thread that does nothing but read and write values from/to PLCs. This thread reads all digital inputs from all PLCs and store values to variables which symbolize sensors. Then it reads analog input from PLC2 and subsequently it writes values to all digital outputs on PLC2, PLC3 and PLC4. These values are stored in variables that represent the state of the actuators. Their name starts with *state_* and continue with the name of actuator. For example variable *state_entry_conveyor*, which represents state of the *Entry conveyor*, is *True* when entry conveyor is on and *False* when entry conveyor is off. Same goes for sensors. For example variable *state_sent_from_left*, which represents state of the sensor *Sent from left*, is *True* when sensor is active and *False* when sensor is inactive. The other six threads work only with these state variables, from which they read (if they are sensors state variables) or write to them (if they are actuators state variables).

Before running the control script, it is important first to run the initialization script stored in the *init_script.py* file, which sets the individual components in the Factory I/O software to their required values.

As additional manipulation with actuators was required, we implemented the HMI, which is located in the *tractorHMI.py* file. HMI is described in detail in next section.

3.1.1 Human machine interface

For easier execution of the scripts and the extended manipulation, an application in Python language with usage of PyQt library was created which simplifies the user's work with a sorting production line. Implementation of this HMI is saved in *tractorHMI.py* file.

It allows you to set which emitter will be used, to run an initialization script as well as start or stop a script for automation of sorting line. The HMI also shows the number of objects that the sorting production line sent in which direction (left, forward, right) and shows weight of current box. It also allows you to turn on or turn off some actuators. Actuators whose values can be changed at runtime are both emitters, entry conveyor and load scale. These actuators are distinguished by a switch, the appearance and operation of which are defined in the *qswitchbutton.py* file.

3.1.2 Usage

To run our sorting line, it is necessary to proceed as follows:

1. Run the Factory I/O software with sorting line.
2. Connect sensors and actuators to the Advantech PLCs (in $File \rightarrow Drivers$).
3. Invert output sensors values (check the box on the left).
4. Open HMI using file *tractorHMI.py.* In Command line run `python tractorHMI.py`.
5. Press *Init.* This will run initialization script *init_script.py.*
6. Press *Start.* This will run control script *tractor.py*
7. Turn on some emittor.
8. Press *Stop* to end the control script.

3.2 Assembly line control

Assembly line serves for creating one complete object from two parts (base and lid). Line contains gripper which grab lid from one conveyor and put it on base on another conveyor. When the object is complete, it's moved to the remover.

Assembly line is simpler than sorting line. This line is controlled only by one script, written in Python programming language. The name of a file is *cv2-solution.py*. Undone version of this file is *cv2.py*, which is part of a laboratory for students. They have to finish *cv2.py*, so that this script can control the line.

3.2.1 Usage

To run our assembly line, it is necessary to proceed as follows:

1. Run the Factory I/O software with assembly line.
2. Connect sensors and actuators to the Advantech PLCs (in *File* → *Drivers*).
3. Invert output sensors values (check the box on the left).
4. Open Command line and run `python cv2.py`
5. Press *Stop* (square in main panel) in Factory I/O to stop the line.

Chapter 4

Conclusions

This paper dealt with the topics of creating a test environment and then capturing the Modbus TCP communication between the client and servers.

Within the project, a sorting line and an assembly line were created. These lines are simulated using the Factory I/O software and controlled automatically with scripts written in Python programming language. For sorting line, HMI was also created where user can control some parts of line manually through it.

Our testbed allows to create various types of attack on SCADA networks, which can be captured and analyzed. It also serve for educational purposes for students as it can be used in laboratories.

Our designed production lines are quite simple and shows only what are the options of Factory I/O software. For bigger, more realistic looking factory, more Unipi and Advantech PLCs would be needed, together with more routers to create bigger local network. Factory I/O software is really good tool for simulation of factory environment, and with proper hardware enables to create real-looking testing environment for SCADA networks.

Acknowledgements

This work is supported by Brno University of Technology project "Application of AI methods to cyber security and control systems"(2020–2022), no. FIT-S-20-6293.

Bibliography

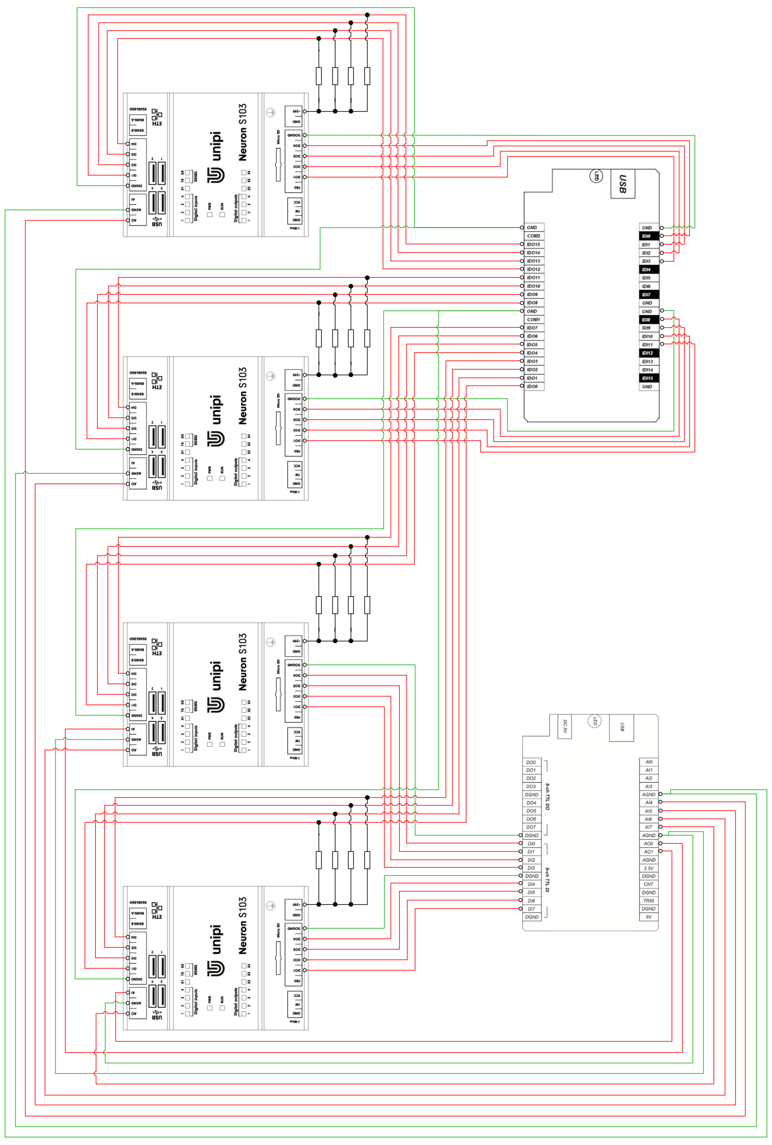
[1] *Product line of programmable controllers and extension modules, UniPi Neuron.* User manual and technical documentation. pages 11

[2] Ján Pristaš. Generování provozu IoT sítí a detekce bezpečnostních incidentů, 6 2018. pages 1

[3] Ondřej Ryšavý and Petr Matoušek. Monitoring Modbus/TCP traffic using IPFIX. Technical Report FIT-TR-2020-03, Faculty of Information Technology BUT, 2020. pages 11

Appendix A

Wiring scheme



Appendix B

Registers and coils

UniPi Neuron S10x

Registers – group 1

Register Number		R/W	DataType	Content	Bit Nr.
Unit 0	**Unit × Reg. NR**				
0	1 × 0	R	MixedBits	Digital inputs of group 1	
				Digital input 1.1	0
				Digital input 1.2	1
				Digital input 1.3	2
				Digital input 1.4	3
1	1 × 1	RW	MixedBits	Digital outputs of group 1	
				Digital output 1.1	0
				Digital output 1.2	1
				Digital output 1.3	2
				Digital output 1.4	3
2	1 × 2	RW	Word	Analog output 1	
3	1 × 3	R	Word	Analog input 1	
4	1 × 4	R	Word	Analog input 2	
5	1 × 5	R	Word	VrefInt	
6	1 × 6	RW	MixedBits	MasterWatchDog (MWD) status of group 1	
				MWD enable	0
				MWD reboot detected	1
7	1 × 7	R	Word	Length of TX queue	
8 – 9	1 × 8 – 9	RW	DWord	Counter of Digital input 1.1	
10 – 11	1 × 10 – 11	RW	DWord	Counter of Digital input 1.2	
12 – 13	1 × 12 – 13	RW	DWord	Counter of Digital input 1.3	
14 – 15	1 × 14 – 15	RW	DWord	Counter of Digital input 1.4	
16	1 × 16	RW	Word	PWM of DO1.1	
17	1 × 17	RW	Word	PWM of DO1.2	
18	1 × 18	RW	Word	PWM of DO1.3	
19	1 × 19	RW	Word	PWM of DO1.4	
20	1 × 20	RW	MixedBits	User programmable LED settings	
				User LED X1	0
				User LED X2	1
				User LED X3	2
				User LED X4	3
1000	1 × 1000	R		Firmware version of group 1	
1001	1 × 1001	R	MixedBits	Number of DI/Dos	
				Number of Dos	0 – 7
				Number of Dis	8 – 15
1002	1 × 1002	R	MixedBits	Number of AI/Ao/Serials of group 1	
				Number of seriál lines	0 – 3
				Number of AOs of	4 – 7
				Number of AIs of	8 – 15
1003	1 × 1003	R		HW Version of group 1	
1004	1 × 1004	R	Word	Board HW version of group 1	
1005 – 1006	1 × 1005 – 1006	R	DWord	Board serial number of group 1	
1007	1 × 1007	R	MixedBits	Interrupt mask of group 1	
				Serial line RX ququе not empty	0
				Sending on srial line finished	1
				Receiveing ModBus RTU frame finished	2

				Digital input changed state	3
1008	1 × 1008	RW	word	MWD timeout of group 1	
1009	1 × 1009	R	word	Vref	
1010	1 × 1010	RW	word	Debounce time of DI1.1 [100μs]	
1011	1 × 1011	RW	word	Debounce time of DI1.2 [100μs]	
1012	1 × 1012	RW	word	Debounce time of DI1.3 [100μs]	
1013	1 × 1013	RW	word	Debounce time of DI1.4 [100μs]	
1014	1 × 1014	RW	MixedBits	Direct Switch function of group 1	
				Enable DS on DI1.1	0
				Enable DS on DI1.2	1
				Enable DS on DI1.3	2
				Enable DS on DI1.4	3
1015	1 × 1015	RW	MixedBits	Enable DS polarity function of group 1	
				Enable DS arity on DI1.1	0
				Enable DS polarity on DI1.2	1
				Enable DS polarity on DI1.3	2
				Enable DS polarity on DI1.4	3
1016	1 × 1016	RW	MixedBits	Enable DS toggle function of group 1	
				Enable DS toggle on DI1.1	0
				Enable DS toggle on DI1.2	1
				Enable DS toggle on DI1.3	2
				Enable DS toggle on DI1.4	3
1017	1 × 1017	RW	word	PWM prescale of group 1	
1018	1 × 1018	RW	word	PWM cycle of group 1	
1019	1 × 1019	RW	MixedBits	AO 1 settings of	
				Enable current output	0
1020	1 × 1020	R	Word	AO 1 Voltage deviation	
1021	1 × 1021	R	Word	AO 1 Voltage offset	
1022	1 × 1022	R	Word	AO 1 Curent deviation	
1023	1 × 1023	R	Word	AO 1 Current offset	
1024	1 × 1024	RW	MixedBits	AI 1 settings	
				Enable current input	0
1025	1 × 1025	R	Word	AI 1 Voltage deviation	
1026	1 × 1026	R	Word	AI 1 Voltage offset	
1027	1 × 1027	R	Word	AI 1 Curent deviation	
1028	1 × 1028	R	Word	AI 1 Current offset	
1029	1 × 1029	R	Word	AI 2 Voltage deviation (on AO1)	
1030	1 × 1030	R	Word	AI 2 Voltage offset (on AO1)	
1031	1 × 1031	RW	MixedBits	Configuration of RS485 serial line	
				Baud rate	0 – 12
				Parity enable	13
				Parity – 0=Even, 1=Odd	14
				ModBus RTU support enabled (interrupt)	15

Baud rate configuration

Value	Speed [bps]
11	2 400
12	4 800
13	9 600
14	19 200
15	38 400
4097	57 600
4098	115 200

Coils – group 1

Coil Number		R/W	Content
Unit 0	**Unit × Coil**		
0	1 × 0	RW	Digital Output 1.1
1	1 × 1	RW	Digital Output 1.2
2	1 × 2	RW	Digital Output 1.3
3	1 × 3	RW	Digital Output 1.4
4	1 × 4	RW	Digital Input 1.1
5	1 × 5	RW	Digital Input 1.2
6	1 × 6	RW	Digital Input 1.3
7	1 × 7	RW	Digital Input 1.4
8	1 × 8	RW	User programmable LED X1
9	1 × 9	RW	User programmable LED X2
10	1 × 10	RW	User programmable LED X3
11	1 × 11	RW	User programmable LED X4
1000	1 × 1000	RW	MWD reset indication/reset of group 1
1001	1 × 1001	RW	Disable 1-Wire bus
1002	1 × 1002	RW	Reset CPU of group 1
1003	1 × 1003	RW	Save current config as default to NV RAM of group 1
1016	1 × 1016	RW	Enable DS on DI 1.1
1017	1 × 1017	RW	Enable DS on DI 1.2
1018	1 × 1018	RW	Enable DS on DI 1.3
1019	1 × 1019	RW	Enable DS on DI 1.4
1020	1 × 1020	RW	Enable DS polarity on DI 1.1
1021	1 × 1021	RW	Enable DS polarity on DI 1.2
1022	1 × 1022	RW	Enable DS polarity on DI 1.3
1023	1 × 1023	RW	Enable DS polarity on DI 1.4
1024	1 × 1024	RW	Enable DS toggle on DI 1.1
1025	1 × 1025	RW	Enable DS toggle on DI 1.2
1026	1 × 1026	RW	Enable DS toggle on DI 1.3
1027	1 × 1027	RW	Enable DS toggle on DI 1.4