

الجمهورية الجزائرية الديمقراطية الشعبية

PEOPLE'S DEMOCRATIC REPUBLIC OF ALGERIA

وزارة التعليم العالي والبحث العلمي

Ministry of Higher Education and Scientific Research

جامعة أبي بكر بلقايد - تلمسان

**About Bekr Belkaid University – Tlemcen -
Faculty of Technology**



THESIS

Presented for the award of the **Master's Degree**

Field: Automation

Specialty: Automation and Industrial Computing

By: Mr. BARKI Mohamed
and
Mr. ABBES Nazih Ayoub

Subject

Programming and Supervision of PLCs from the internet network

Defended publicly on: 18 / 06 / 2026, before the jury composed of:

Mr. MELIANI Sidi Mohammed	Profesor	Tlemcen Univ.	President
Mr. HADJ ABDELKADER Mohammed amine	Profesor	Tlemcen Univ.	Supervisor
Ms. SAIDI FARAH	MCB	Tlemcen Univ.	Examiner

Academic Year: 2025 / 2026

Dedication

All praise is due to Allah, the Creator of the heavens and the earth. O Allah, to You belongs all praise for having guided us to Islam. May Your peace, blessings, and grace be upon our master Muhammad and upon the family of our master Muhammad, as You bestowed peace and blessings upon Abraham and the family of Abraham. Indeed, You are Most Praiseworthy, Most Glorious.

We would like to express our sincere gratitude to our parents and our families for their continuous support and encouragement throughout our academic journey.

We also extend our appreciation to the entire university community for their dedication and commitment to providing a supportive and enriching academic environment despite the challenges encountered.

Acknowledgment

We would like to express our special thanks to the administration of the Faculty of Technology for providing its services with efficiency and flexibility.

We sincerely thank all our professors who delivered high-quality lectures and demonstrated patience and dedication in conveying knowledge. From them, we learned that patience is essential and that success can only be achieved through perseverance.

May Allah reward them abundantly for their efforts.

We would also like to extend our special appreciation to Prof. Amine HADJ ABDELKADER and Dr. Brixi-Nigassa Mohammed El Amine. Through them, we were introduced to the world of Programmable Logic Controllers both theoretically and practically. We always looked forward with enthusiasm to their lectures.

We also wish to thank the members of the examination jury Prof. MELIANI Sidi Mohammed , our supervisor Prof. Amine HADJ ABDELKADER, and Dr. Saidi Farah for dedicating their time to reading and evaluating this work. Their constructive remarks and valuable suggestions will be carefully taken into consideration in order to further refine and improve it.

Furthermore, we would like to thank all our professors, including Dr. BENYAHIA BOUMEDIENE, Dr. BENSALAH Choukri, Dr. MOULAI KHATIR AHMED NASSIM, and Dr. HANDOUZI Wahida. Unfortunately, this small page cannot include all these names, nor can it summarize five years in just a few lines. May God bless them all and reward them abundantly.

Contents

List of Figures	8
List of Tables	9
List of Abbreviations	10
Introduction	12
I State of the Art	14
Introduction	15
I.1 Architecture of PLCs	15
I.1.1 Definition	15
I.1.2 Advantages	16
I.1.3 Parts of PLCs	18
I.1.4 Memory and Data Organization in Siemens PLC Architecture	21
I.2 Industrial Remote Access and IIoT Connectivity Platforms	27
I.2.1 EWON® by HMS NETWORKS	27
I.3 University Research on Remote Automation Laboratories and Industry 4.0 Systems	31
I.3.1 IOT-OPEN.EU Reloaded Project (EUROPE)	31
I.3.2 Remote Control and Monitoring Protocols for PLC and HMI Systems: A Case Study of MODBUS, WMI and VNC (Türkiye)	31
I.3.3 TwinCAT PLC Web Based Remote Control Laboratories for Mobile and Browser Access (Spain)	32

1.3.4 Networked PLC-OPC Remote Laboratories for Fuzzy Control and Data Acquisition (Spain)	32
Conclusion	33
2 Hardware, Software and Network Tools	34
Introduction	35
2.1 Hardware Architecture	35
2.1.1 The Raspberry Pi 4 Model B	35
2.1.2 Raspberry Pi OS Lite	35
2.1.3 iptables	36
2.2 TIA Portal	36
2.3 Networking	37
2.3.1 Industrial communication and protocols	37
2.3.2 Industrial Ethernet Standards	38
2.3.3 OSI Model (ISO/OSI Reference Model)	39
2.3.4 TCP/IP	40
2.3.5 S7 Communication (ISO-TCP) and Snap7 library	41
2.4 The Web Application Stack	42
2.4.1 Flask Framework	42
2.4.2 Gunicorn	42
2.4.3 Nginx	42
2.4.4 JSON	43
2.5 Remote Access and Tunneling	43
2.5.1 Virtual Private Networks (VPNs)	43
2.5.2 Layer 2 vs Layer 3 VPNs	44
2.5.3 SoftEther VPN	44
2.5.4 Ethernet over HTTP	45
2.5.5 VPN session	45
2.5.6 Virtual Hub and Local Bridge	47
2.5.7 SoftEther VPN client and Virtual Network adapter	47

Conclusion	48
3 Application	49
Introduction	50
3.1 Install and compile	50
3.2 Initial Server Configuration	51
3.3 Disabling Third-Party Services	53
3.3.1 VPN Azure	53
3.3.2 Dynamic DNS	53
3.3.3 NAT Traversal	54
3.4 Connect engineer PC	54
3.5 TIA Portal adapter problem	56
3.6 Bridge Configuration	58
3.7 Routing Missing or Conflict	58
3.7.1 Solution: Manual Ethernet Configuration and Route Prioritization .	59
3.7.2 Explanation of the Configuration Parameters	60
3.7.3 Result	60
3.8 Unit Testing	60
3.8.1 Description of the Controlled System	62
3.9 Experimental Results	67
3.10 PLC Monitor Application	76
3.10.1 Features	76
3.10.2 System Requirements	77
3.10.3 Access to PLC Monitor Application	77
Conclusion	85
Conclusion	86
1 Appendix	87
1.1 PLC MONITOR source code	87
1.2 Motor Speed Test Full script	87

List of Figures

1.1	Example of a PLC from Siemens [1].	16
1.2	Relay Logic Vs Ladder Logic [3].	17
1.3	High-Speed Liquid Vial Sorting [4].	18
1.4	Compact PLC Diagram [5].	19
1.5	Modular PLC Diagram [5].	19
1.6	A simple ladder logic program in TIA-PORTAL [6].	21
2.1	The Raspberry Pi 4B [36].	36
2.2	TIA PORTAL boot interface [35].	37
2.3	Industrial network example [21].	38
2.4	Typical System Configuration for Ethernet [20].	39
2.5	The 7 Layers of the OSI Model [30].	39
2.6	Mapping of OSI layers to the layers of the TCP/IP Model [30].	40
2.7	S7 comm in the OSI Reference Model [17].	41
2.8	Global architecture for The Web Application Stack [32].	43
2.9	VPN connection to Virtual Hub by SoftEther VPN Client [27].	44
2.10	Ethernet over HTTP [29].	45
2.11	Passage through NAT proxy server or CGNAT by SoftEther VPN [27].	46
2.12	Virtual Hub and Local Bridge network [28].	47
2.13	VPN connection to Virtual Hub by SoftEther VPN Client [28].	48
2.14	System Global Architecture [32].	48
3.1	Install and compile commands [34].	50

3.2 Set permissions and start server commands [34].	51
3.3 Enter <code>vpncmd</code> and set server password commands [34].	52
3.4 Create Hub and User commands [34].	53
3.5 Disable Dynamic DNS in <code>vpnserv.conf</code> [34].	54
3.6 Disable NAT Traversal in <code>vpnserv.conf</code> [34].	54
3.7 Create new VPN connection interface [38].	55
3.8 TIA portal adapter problem [35].	57
3.9 Registry changes [33].	57
3.10 Network adapter appearance [35].	58
3.11 <code>BridgeCreate</code> and <code>BridgeList</code> commands [34].	58
3.12 Motor Speed Control System [39].	62
3.13 HMI used to control motor speed [14].	64
3.14 HMI interface during Test 1 [39].	68
3.15 HMI interface during Test 2 [39].	69
3.16 HMI interface during Test 3 300 rpm [39].	70
3.17 HMI interface during Test 3 700 rpm [39].	71
3.18 HMI interface during Test 3 1000 rpm [39].	71
3.19 HMI interface during Test 4 [39].	72
3.20 HMI interface during Test 5 [39].	74
3.21 PLC Monitor Logo [40].	76
3.22 The process is referred to as "Speed Control" [41].	78
3.23 Linking S7-1200 PLC to PLC MONITOR application [41].	78
3.24 A flashing green indicator denotes successful connectivity [41].	79
3.25 Variable Name Field [41].	79
3.26 Memory Area List [41].	79
3.27 Data Block Number Field [41].	80
3.28 Data Type Field [41].	80
3.29 Byte Number Field [41].	80
3.30 Bit Number Field [41].	81

3.31 Motor Speed Control System Tag List [41].	82
3.32 Process Control Panel [41].	84

List of Tables

1.1 PLC Memory Areas Overview [7].	23
1.2 PLC Data Types and Address Examples [7].	25
1.3 Integer data types (U = unsigned, S = short, D = double) [7].	26
1.4 Floating-point real data types (L=Long) [7].	26
1.5 PLC Time, Date, and DTL Data Types [7].	27
3.1 Motor Start Procedure [37].	64
3.2 Motor Stop Procedure [37].	65
3.3 Motor Speed Increase Procedure [37].	65
3.4 Motor Speed Decrease Procedure [37].	65
3.5 Motor Speed Reset Procedure [37].	65
3.6 Drive Fault Acknowledgment Procedure [37].	66
3.7 Complete Address Reference for Motor Speed Control [37].	66

List of Abbreviations

AES	Advanced Encryption Standard
API	Application Programming Interface
CGNAT	Carrier-Grade Network Address Translation
CPU	Central Processing Unit
CSMA/CD	Carrier Sense Multiple Access with Collision Detection
DCP	Discovery and Configuration Protocol
EJS / EJsS	Easy Java Simulations / Easy JavaScript Simulations
GPIO	General Purpose Input/Output
HMI	Human Machine Interface
HTTP/S	Hypertext Transfer Protocol / Secure
IIoT	Industrial Internet of Things
IP	Internet Protocol
IPC	Industrial Personal Computer
IRT	Isochronous Real-Time
JIL	Java-Internet-LabVIEW
KPI	Key Performance Indicator
LAN	Local Area Network
LCD	Liquid Crystal Display
LLDP	Link Layer Discovery Protocol
LReal	Long Real
MAC	Media Access Control
MPI	Multi-Point Interface
MQTT	Message Queuing Telemetry Transport
NAT	Network Address Translation

OPC / OPC UA

Open Platform Communications / OPC Unified Architecture

OS Operating System**OSI** Open Systems Interconnection**P2P** Peer-to-Peer**PID** Proportional-Integral-Derivative**PLC** Programmable Logic Controller**PROFINET** Process Field Net**PS / PSU** Power Supply / Power Supply Unit**RDP** Remote Desktop Protocol**RLL** Relay Ladder Logic**RS-232 / RS-485**

Recommended Standard 232 / 485 (Serial Communication Standards)

RT Real-Time**S7** SIMATIC S7 (Siemens Proprietary Protocol)**SCADA** Supervisory Control and Data Acquisition**SMS** Short Message Service**SQL** Structured Query Language**SSL** Secure Sockets Layer**TCP/IP** Transmission Control Protocol / Internet Protocol**TIA Portal** Totally Integrated Automation Portal**VSD** Variable Speed Drive**VNC** Virtual Network Computing**VPN** Virtual Private Network**WMI** Windows Management Instrumentation**WSGI** Web Server Gateway Interface

Introduction

Since the dawn of industrial automation, the installation and maintenance of industrial systems have remained among the most demanding engineering disciplines. Behind every continuously operating production line lies the expertise of technical teams whose mission is to ensure reliability, efficiency, and uninterrupted operation. As industries evolve, so too must the technologies and methods that support them.

At the heart of every modern automated factory stands the PLC, which is the digital brain responsible for monitoring sensors, processing information, and coordinating every stage of the production process. Through a direct connection to the factory network, engineers can access valuable operational data, monitor system variables in real time, diagnose faults, and reprogram industrial equipment whenever required.

Yet, despite the remarkable advances in automation, one major challenge remains unchanged: physical accessibility. An engineer cannot be present in multiple factories at the same time, and industrial facilities are often located hundreds of kilometers apart, in remote regions or difficult terrain. As a result, the simple act of connecting a maintenance computer to a factory network may require hours or even days of travel. During this time, production losses continue to accumulate while critical equipment remains unavailable. These operational constraints have made secure remote access not merely a convenience, but an essential requirement for modern industry.

Today, the unprecedented expansion of global Internet connectivity, reinforced by the rapid deployment of satellite communication networks, has transformed the way industrial systems can be accessed. Distance is no longer the obstacle it once was. Secure communication technologies now offer the possibility of extending the factory network beyond its physical boundaries, allowing engineers to interact with industrial equipment from virtually anywhere in the world.

Motivated by this technological evolution and by the growing demand for secure remote industrial maintenance, this Master's thesis presents the design and implementation of a complete Internet-based PLC remote access solution. The proposed architecture integrates a Raspberry Pi deployed within the factory network as a VPN server, creating an encrypted communication tunnel that enables secure remote connectivity. Furthermore,

a lightweight web application has been developed and hosted directly on the Raspberry Pi, allowing technicians to monitor and control PLC variables through any standard web browser without requiring dedicated engineering software.

From the theoretical foundations of industrial automation to the implementation and validation of the proposed remote access solution, this thesis is organized as follows: The first chapter presents the state of the art by introducing the fundamentals of PLCs, followed by a review of current industrial remote access and IIoT platforms, as well as relevant academic research on remote automation and Industry 4.0 solutions.

The second chapter establishes the theoretical foundation of the proposed solution. It introduces the hardware platform, networking concepts, and industrial communication protocols. It also presents the web application architecture and remote access technologies.

The third chapter presents the implementation and validation of the proposed solution. It concludes with the deployment of a secure Layer Two VPN communication tunnel to the PLC network and the presentation of PLC Monitor, a lightweight web based application developed for PLC monitoring and supervision.

Chapter 1

State of the Art

Introduction

This chapter presents the theoretical background required for the development of the proposed remote PLC control solution. It begins with an overview of PLCs, including their architecture, operating principles, memory organization, and data management. Particular attention is given to Siemens PLCs, which are used throughout this project.

The chapter then reviews existing industrial remote access and IIoT connectivity solutions that enable remote programming, monitoring, and maintenance of automation systems. Finally, several academic research projects and remote laboratory initiatives are examined to provide an overview of current developments and to position the proposed solution within the current state of the art.

1.1 Architecture of PLCs

1.1.1 Definition

Before the appearance of PLCs, engineers used relay logic and hard wiring to control their systems, where for every new function a new design had to be made. However, with these industrial-grade computers, the task became easier and cheaper. These devices are capable of being programmed to perform control functions. Since industrial engineering adopted PLC technology, it improved its control speed and enabled data exchange via several communication protocols, ensuring network compatibility [2].

The industrial environment is usually harsh, so PLCs are designed to withstand high temperatures, electrical noise, and vibrations. The operation program and manufacturing process are stored in non-volatile memory (which keeps data even after power loss) or in battery-backed memory (a memory supported by a small battery) to save process status. On the other hand, the PLC is designed for complex industrial systems, equipped with digital and analog input/output interfaces. It is an example of a real-time system, where the output depends on the PLC input. Figure  represents a CPU, the brain of the PLC.



Figure 1.1: Example of a PLC from Siemens [1].

1.1.2 Advantages

Enhanced flexibility : the relationship between input and output is determined by the user-provided program. It is very easy to update the process logic by downloading a new version to the PLC. Manufacturers also send updates to end users to update firmware and security. These devices allow protection of the design through hardware protection such as key-locks or software protection [2].

Improved system reliability : the PLC company provides engineering software to write, compile, simulate, and finally test on a real PLC. Once these steps are completed, the program can easily be downloaded to other PLCs. There are no wiring errors since the process logic is contained in the program, and it replaces most external wiring. PLC components are solid-state electronics with no mechanical motion, which improves reliability [2].

Reduced cost : PLCs are designed to replace relay logic for cost purposes. If the system has more than 6 relays and contains tens to hundreds of signals, it is recommended to use a PLC [2].

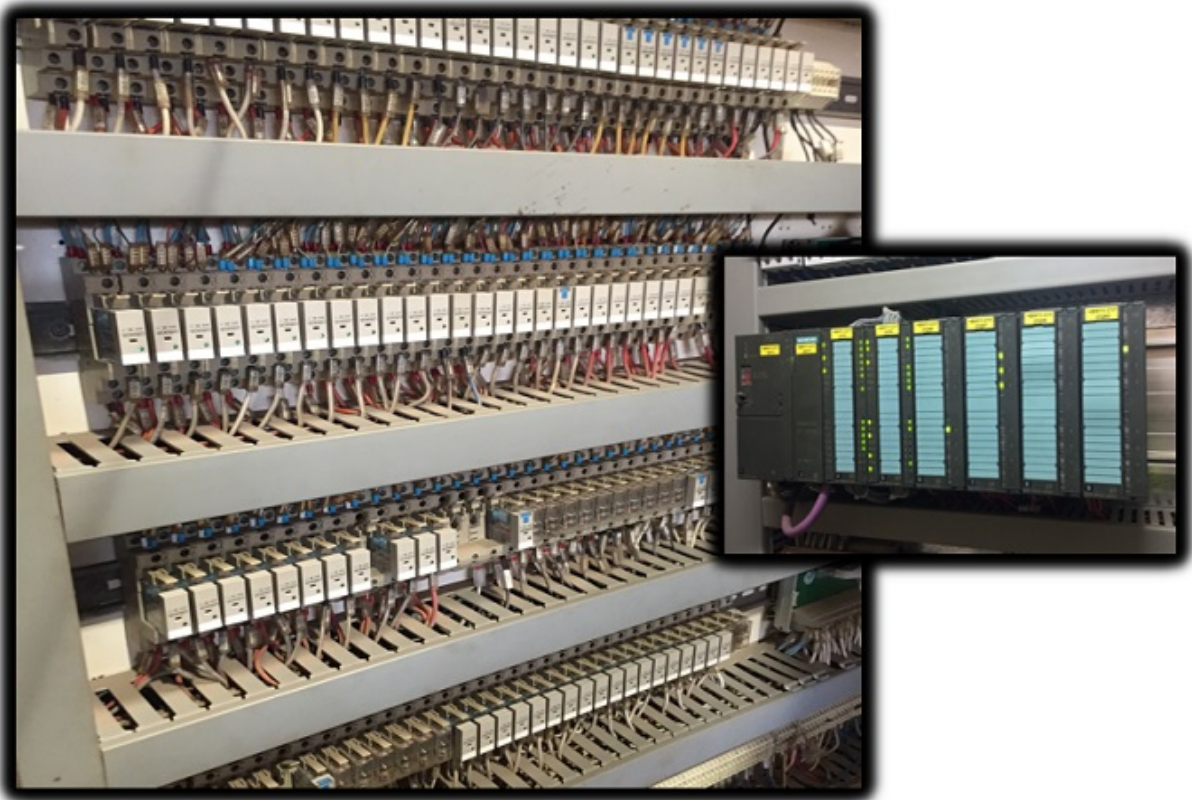


Figure 1.2: Relay Logic Vs Ladder Logic [3].

Advanced communication and networking functions : the PLC can communicate with HMI, SCADA systems, other PLCs, and laptops via several communication ports (depending on device class and version), such as Ethernet (RJ45), serial ports (RS232/RS485), USB ports, and some proprietary ports like MPI/Profibus for Siemens [2].

Fault detection and diagnostics : PLCs can perform process diagnostics and generate alarms for specific events. In large manufacturing areas, it takes much time to test each device to find an error, but if the devices are connected to the PLC, it becomes easier to test device signals [2].

Real-time response : PLCs have quick response capabilities designed for high-speed and real-time applications, which means that if a process deals with thousands of items per second, sensors can detect any manufacturing error, and the PLC takes action in a fraction of a second [2].

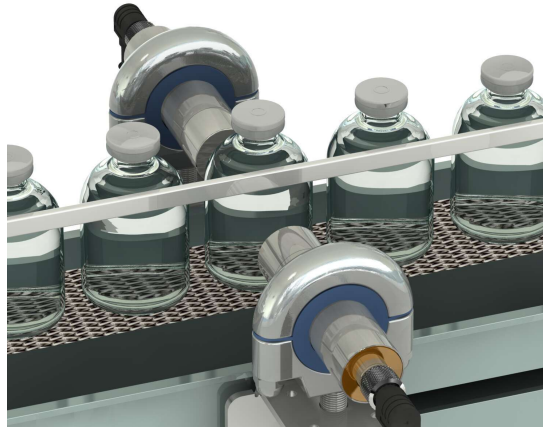


Figure 1.3: High-Speed Liquid Vial Sorting [2].

1.1.3 Parts of PLCs

The PLC hardware architecture can be divided into the central processing unit (CPU), the input/output interface, the power supply, and the programming device. The term architecture may refer to the hardware design, the software design, or both. An open architecture is designed to allow PLCs to connect to devices and programs from other manufacturers, while a closed architecture system is proprietary and designed by a single manufacturer. Therefore, before adding any generic component, it must be ensured that it is compatible with the specific PLC.

The programming languages and methods are generally the same across manufacturers; however, there may be differences in memory addressing and data handling. Consequently, programs cannot always be exchanged between different manufacturers without modifications.

Modular versus Compact PLCs (Hardware) : There are two architectures used to combine PLC components together. The first is the fixed I/O type, where the CPU and the input/output interface are integrated into a single unit, usually in a small box without separate removable modules. The I/O terminals have a fixed number of built-in connections for inputs and outputs. The main advantage of this type is its lower cost. The available I/O varies depending on the device version, and it can be expanded only by purchasing additional fixed I/O units. However, this system has a disadvantage in its lack of flexibility, as the user is limited to what is provided in the package [2].

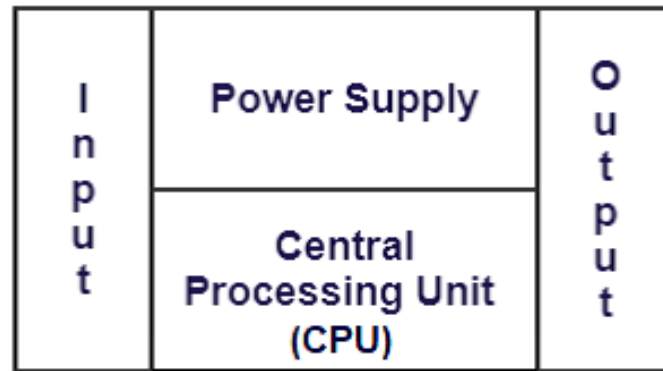


Figure 1.4: Compact PLC Diagram [5].

In the modular system, the modules are divided according to their functions. Additional modules provided by the manufacturer can be easily added using a simple plug-in mechanism. A basic modular controller consists of a rack, a power supply, a CPU, input/output modules, and an operator interface for programming and monitoring. By plugging the modules into the rack, they are automatically connected through the backplane, which is a series of electrical contacts [2].

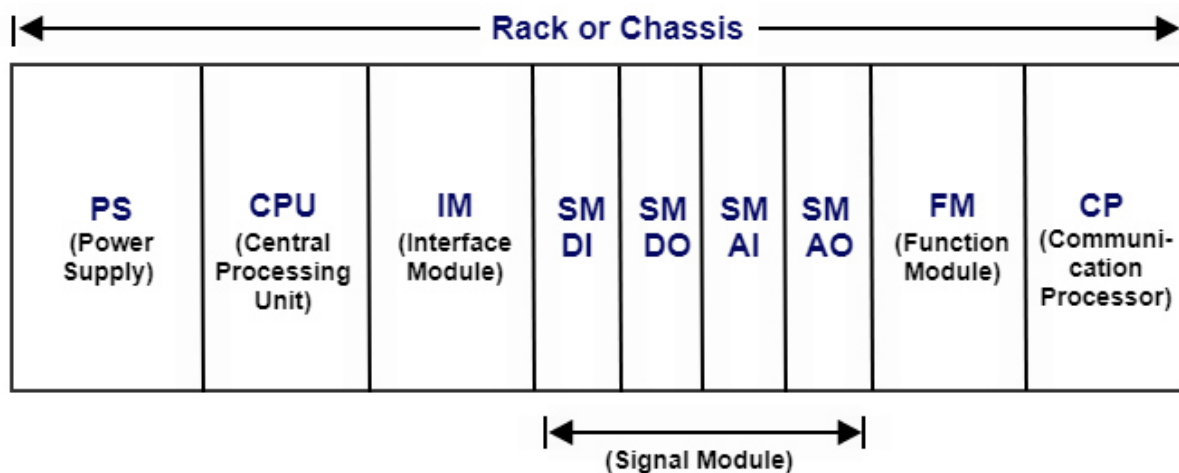


Figure 1.5: Modular PLC Diagram [5].

The power supply PS : This module provides power to all the devices in the rack. In large systems, the field devices (such as sensors, push buttons, solenoid contactors, motors, etc.) are powered from an external source. In this case, the PLC power supply is used only for control signals. However, in low-power and small-scale applications, it can also supply some low-energy devices.

The processor CPU : is the brain of the PLC, often a microprocessor used to handle logic operations and communication control. It requires memory to store the

user program, process data, and the status of inputs and outputs. It is responsible for controlling all PLC activity.

The CPU controls all PLC functions and is designed so that the user can enter the required program using relay ladder logic. The program is executed as part of a repetitive process called a scan. A typical PLC scan begins with the CPU reading the status of the inputs, after which the application program is executed. Once the program execution is completed, the status of all outputs is updated. Then, the CPU performs internal diagnostics and communication tasks. This cycle is continuously repeated as long as the PLC is in run mode.

The I/O Interface : forms the system through which field devices are connected to the controller. The purpose of this interface is to condition the different signals received from or sent to external field devices. Input devices such as pushbuttons, limit switches, and sensors are hardwired to the input terminals. Output devices such as small motors, motor starters, solenoid valves, and indicator lights are hardwired to the output terminals. To provide electrical isolation between the internal components and the input/output terminals, PLCs commonly use an optical isolator, which uses light to couple the circuits together. External devices are also referred to as field or real-world inputs and outputs. These terms are used to distinguish the actual external devices that are physically wired from the internal user program, which replicates the function of relays, timers, and counters.

Programming device : The processor memory contains the required program. Most commonly, this program is written in Relay Ladder Logic and is entered by connecting ladder elements together. The PLC ladder language uses graphical symbols to make it easier for engineers who previously worked with relay logic systems, since it resembles a relay control schematic.

To load the program, the engineer uses either a laptop with the manufacturer's software or a compact hand-held device. The hand-held unit, which has a small LCD screen, can be used to edit the program or monitor the process, but it is suitable only for small programs. Both methods are frequently used on the factory floor.

The best choice for an engineer is the laptop with the manufacturer's software because it provides full access to PLC programming and monitoring functions. The computer monitor can display more logic on the screen than hand-held devices.

The computer can be connected through a serial link or Ethernet. Once the program is downloaded, the engineer disconnects the link, and the system can start operating.

User Program : a series of instructions that control the PLC actions. The programming language is responsible for combining these instructions to produce the desired functions.

Relay Ladder Logic (RLL) is the standard language used in PLCs. It represents contacts, coils, and special function blocks. This language has been developed and maintained because it is highly demanded in industry.

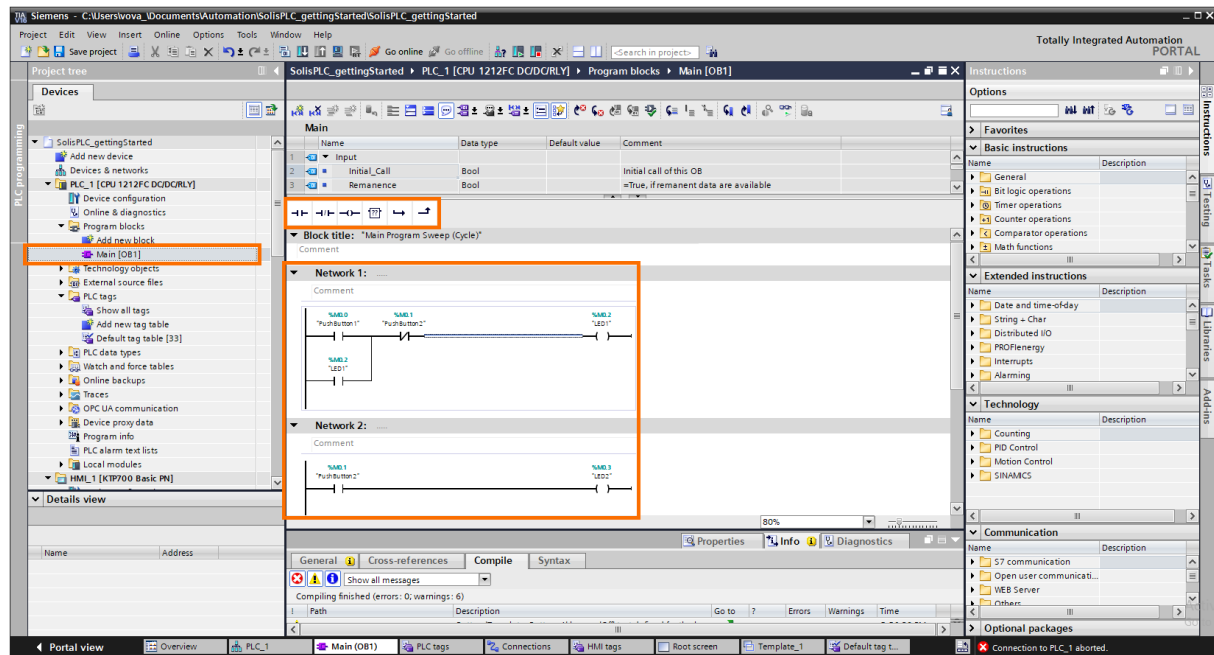


Figure 1.6: A simple ladder logic program in TIA-PORTAL [6].

1.1.4 Memory and Data Organization in Siemens PLC Architecture

This part is about the SIMATIC S7-1200, as it is the PLC used in our application. For a deeper understanding, the reader is referred to the CPU memory section.

Accessing the data of the S7-1200

STEP 7 (STEP 7 is the PLC programming component embedded in the TIA Portal engineering platform) enables symbolic programming through the use of symbolic identifiers, known as tags, which are assigned to data addresses. These tags can represent PLC variables linked to memory locations and I/O points, as well as local variables defined within specific program blocks. Within the user program, these tags are used directly by referencing their names as instruction parameters.

To gain a clearer understanding of how the CPU structures and accesses memory, the following sections introduce the concept of absolute addressing, which underlies the symbolic tags used in PLC programming.

During program execution, the CPU provides different mechanisms for data storage:

Global memory : the CPU contains several special memory areas such as inputs

(I), outputs (Q), and bit memory (M). This type of memory is called global because it can be accessed by any program block in the PLC without restriction. It is shared across the entire user program, which means all code blocks can read from and write to it whenever needed.

PLC tag table : in STEP 7, the PLC tag table is used to define symbolic names (tags) that are associated with specific memory addresses within the PLC. Each tag serves as an abstraction of a physical or memory location, enabling the programmer to reference data in a more meaningful and structured way.

These tags are global within the STEP 7 project, meaning they can be accessed from any program block. This approach enhances program readability, maintainability, and reduces dependency on direct absolute addressing, allowing a more structured and application-oriented programming style.

Data block (DB) : used in a PLC program to store data required by the user program and its code blocks. The information stored in a DB remains available even after the execution of the associated code block has finished.

A distinction is made between global data blocks and instance data blocks. A global DB contains data that can be accessed by any code block within the program, whereas an instance DB is assigned to a specific function block (FB) and stores the data related to its parameters and internal variables. This structure allows each FB to maintain its own dedicated memory space during program execution.

Temporary memory (Local memory) : Whenever a code block is called, the CPU operating system allocates a portion of temporary or local memory (L) for use during the execution of that block. This memory is used to store intermediate values that are only required while the block is active.

Once the execution of the code block is completed, the allocated local memory is released and can be reassigned by the CPU for the execution of other code blocks. This ensures efficient use of memory resources during program operation.

When referencing the input (I) or output (Q) memory areas such as I0.3 or Q1.7, the program interacts with the process image. To directly access the corresponding physical input or output, the reference must be followed by the suffix :P (for example, I0.3:P, Q1.7:P, or Stop:P).

Memory area	Description	Force	Retentive
I	Copied from physical inputs at the beginning of the scan cycle	No	No
I_P (Physical input)	Immediate read of the physical input points on the CPU, SB, and SM	Yes	No
Q	Copied to physical outputs at the beginning of the scan cycle	No	No
Q_P (Physical output)	Immediate write to the physical output points on the CPU, SB, and SM	Yes	No
M Bit memory	Control and data memory	No	Yes (optional)
L Temp memory	Temporary data for a block local to that block	No	No
DB Data block	Data memory and also parameter memory for FBs	No	Yes (optional)

Table 1.1: PLC Memory Areas Overview [7].

To directly access whether for reading or writing the physical inputs or outputs, the suffix :P must be appended to the corresponding address or tag (for example, I0.3:P, Q1.7:P, or Stop:P).

Each memory location within the system is assigned a unique address. The user program relies on these addresses to access and manipulate the data stored in the respective memory locations. The absolute address is composed of the following elements:

- The memory area identifier (such as I, Q, or M).
- The data size to be accessed (e.g., B for Byte, W for Word, or D for Double Word).
- The starting address of the data (for example, byte 3 or word 3).
- When accessing a bit representing a Boolean value, no size identifier is required. Only the memory area, byte address, and bit position are specified (for example, I0.0, Q0.1, or M3.4).

Data type

Data types define both the size of a data element and the way in which that data is interpreted. Each instruction parameter supports at least one data type, while some parameters support multiple data types. The supported data types for a given parameter can be viewed by hovering the cursor over the parameter field of the instruction.

A formal parameter refers to the identifier within an instruction that specifies the location of the data to be used by that instruction (for example, the IN1 input of an ADD instruction). An actual parameter, in contrast, refers to the memory location (preceded by the % symbol) or constant that contains the data supplied to the instruction (for example, %MD400 "Number_of_Widgets"). The data type of the actual parameter must correspond to one of the data types supported by the formal parameter of the instruction.

When defining an actual parameter, either a symbolic tag or an absolute memory address must be specified. Tags associate a symbolic name with a data type, memory area, memory offset, and an optional comment. They can be created either in the PLC tag editor or in the interface editor of a program block (OB, FC, FB, or DB). If an absolute address is entered without an associated tag, an appropriately sized default tag is automatically created.

All data types, with the exception of String, are available in both the PLC tag editor and block interface editors. The String data type is only available within block interface editors. In addition, constant values may be assigned to many input parameters where applicable.

Data type	Bit size	Number type	Number range	Constant examples	Address examples
Bool	1	Boolean	FALSE or TRUE	TRUE, 1,	I1.0 Q0.1 M50.7 DB1.DBX2.3
		Binary	0 or 1	0, 2#0	
		Octal	8#0 or 8#1	8#1	
		Hexadecimal	16#0 or 16#1	16#1	
Byte	8	Binary	2#0 to 2#11111111	2#00001111	IB2 MB10 DB1.DBB4 Tag_name
		Unsigned integer	0 to 255	15	
		Octal	8#0 to 8#377	8#17	
		Hexadecimal	B#16#0 to B#16#FF	B#16#F, 16#F	
Word	16	Binary	2#0 to 2#1111111111111111	2#11110000 11110000	MW10 DB1.DBW2 Tag_name
		Unsigned integer	0 to 65535	61680	
		Octal	8#0 to 8#177777	8#170360	
		Hexadecimal	W#16#0 to W#16#FFFF, 16#0 to 16#FFFF	W#16#F0F0, 16#F0F0	
DWord	32	Binary	2#0 to 2#11111111111111111111111111111111	2#11110000 11111111001111	MD10 DB1.DBD8 Tag_name
		Unsigned integer	0 to 4294967295	15793935	
		Octal	8#0 to 8#377777777777	8#74177417	
		Hexadecimal	DW#16#0000_0000 to DW#16#FFFF_FFFF, 16#0000_0000 to 16#FFFF_FFFF	DW#16#F0FF0F, 16#F0FF0F	

Table 1.2: PLC Data Types and Address Examples [7].

Data type	Bit size	Number range	Constant examples	Address examples
USInt	8	0 to 255	78, 2#01001110	MB0, DB1.DBB4, Tag_name
SInt	8	-128 to 127	+50, 16#50	
UInt	16	0 to 65,535	65295, 0	MW2, DB1.DBW2, Tag_name
Int	16	-32,768 to 32,767	30000, +30000	
UDInt	32	0 to 4,294,967,295	4042322160	MD6, DB1.DBD8, Tag_name
DInt	32	-2,147,483,648 to 2,147,483,647	-2131754992	

Table 1.3: Integer data types (U = unsigned, S = short, D = double) [7].

Data type	Bit size	Number range	Constant Examples	Address examples
Real	32	-3.402823e+38 to -1.175 495e-38, ± 0 , +1.175 495e-38 to +3.402823e+38	123.456, -3.4, 1.0e-5	MD100, DB1.DBD8, Tag_name
LReal	64	-1.7976931348623158e+380 to -2.2250738585072014e-308, ± 0 , +2.2250738585072014e-308 to +1.7976931348623158e+380	12345.123456789e40, 1.2E+40	DB_name.var_name Rules: • No direct addressing support • Can be assigned in an OB, FB, or FC block interface table

Table 1.4: Floating-point real data types (L=Long) [7].

Data type	Size	Range	Constant Entry Examples
Time	32 bits	T#- 24d_20h_31m_23s_648ms to T#24d_20h_31m_23s_647ms Stored as: -2,147,483,648 ms to +2,147,483,647 ms	T#5m_30s T#1d_2h_15m_30s_45ms TIME#10d20h30m20s630ms 500h10000ms 10d20h30m20s630ms
Date	16 bits	D#1990-1-1 to D#2168-12-31	D#2009-12-31 DATE#2009-12-31 2009-12-31
Time_of_Day	32 bits	TOD#0:0:0.0 to TOD#23:59:59.999	TOD#10:20:30.400 TIME_OF_DAY#10:20:30.400 23:10:1
DTL (Date and Time Long)	12 bytes	Min.: DTL#1970-01-01-00:00:00.0 Max.: DTL#2554-12-31- 23:59:59.999 999 999	DTL#2008-12-16-20:30:20.250

Table 1.5: PLC Time, Date, and DTL Data Types [7].

1.2 Industrial Remote Access and IIoT Connectivity Platforms

1.2.1 EWON® by HMS NETWORKS

Ewon is an industrial remote access solution based on a Virtual Private Network (VPN). It relies on the installation of an Industrial Remote Access Gateway, such as the Ewon Cosy 131 or Ewon Flexy, within the plant's industrial network.

The Ewon device establishes a secure outbound connection to the Talk2M cloud platform using TCP port 443 (HTTPS). On the other side, the engineer connects to Talk2M via the eCatcher client software. After authentication, the engineer can select the target machine and access the industrial network as if physically present on the factory's local network.

This architecture eliminates the need for a direct inbound connection to the industrial site, thereby enhancing overall security.

Security IT Perspective (Site IT Staff) :

The Ewon solution has been designed to meet the stringent security requirements of industrial environments.

Hardware-Level Security : Ewon devices are equipped with digital inputs that enable:

- activation or deactivation of the VPN connection.
- triggering the transmission of predefined emails or SMS notifications.

Firewall Configuration and Management : On-site IT teams can implement the following security policies:

- Blocking all inbound connections.
- Allowing exclusively TCP port 443 for outbound connections.
- Filtering based on the Ewon device's MAC address and IP address.
- Setting up an approved list of destination IP addresses for Talk2M servers.

Encryption and Authentication : The Talk2M platform relies on recognized industrial security standards:

- User authentication via SSL.
- Data encryption using the AES algorithm, ensuring data confidentiality during transmission.

Compatibility with Proxy Servers : The solution is compatible with proxy servers, making it suitable for industrial sites with high security requirements.

Presentation of the Ewon Cosy :

The Ewon Cosy is a simple, cost-effective, and rapidly deployable remote access solution.

Main Advantages

- Reduced cost, making it suitable for small machines and simple systems.
- Easy configuration in only a few steps.
- Non-technical deployment facilitated through the use of a USB key.

Inputs and Outputs : The integrated input/output interfaces of the Cosy are dedicated to predefined functions:

- Input 1: Activation/deactivation of Talk2M or transmission of predefined email/SMS notifications.
- Input 2: Transmission of predefined email/SMS notifications.
- Output 3: Indication of the VPN connection status.

Models and Connectivity : Three models are available depending on the communication medium:

- Ethernet.
- Wi-Fi + Ethernet.
- Cellular + Ethernet.

Access to the Talk2M Account

eCatcher Desktop : The eCatcher Desktop application is a Windows-based client that enables:

- Remote access to PLC and HMI programming tools.
- Management of the Talk2M account.

M2Web : M2Web provides browser-based access to industrial equipment such as:

- Web servers.
- VNC servers.
- RDP access.
- HMIs, IPCs, and industrial cameras.

Authentication is identical to that used with eCatcher.

eCatcher Mobile : eCatcher Mobile is a mobile application that enables remote access to proprietary HMI applications from a smartphone.

Data Services with Ewon Flexy

PLC Data Acquisition : PLC data can be acquired by configuring tags within the Ewon device [9]:

- Siemens S7-communication based PLCs: use of the Snap7 library.
- Other PLCs: use of the OPC UA protocol.

KPI Development and Visualization : Key Performance Indicators (KPIs) can be developed and visualized:

- Via a web browser,
- Via eCatcher.

Geographic Representation of Machine Locations : Machine locations can be displayed on a geographic map to facilitate monitoring and asset management.

Alarms and Notifications : Alarms can be configured to generate notifications:

- By email using the internal Ewon mail server or an external server (e.g., Gmail),
- By SMS,
- Through advanced cloud-based solutions using MQTT.

Email and SMS services are based on open standards.

Compatibility with Existing Equipment : The Ewon solution is compatible with existing industrial equipment, including brownfield installations.

Industrial Use Cases

- **Cofresco :** Implementation of remote maintenance to improve machine commissioning and standardize interfaces.
- **UK Power Networks Services :** Deployment of a cellular Ewon Cosy solution for remote monitoring of power quality in a water treatment facility.
- **Fournier Industries Group :** Implementation of the Ewon Cosy+ to optimize industrial processes and enhance operational efficiency.

1.3 University Research on Remote Automation Laboratories and Industry 4.0 Systems

This section presents existing university and laboratory projects.

1.3.1 IOT-OPEN.EU Reloaded Project (EUROPE)

IOT-OPEN.EU Reloaded is an initiative funded by the European Union under the Erasmus+ program aimed at improving higher education in the field of the Internet of Things (IoT). It focuses on strengthening collaboration between universities and industry partners by developing and deploying remote laboratory infrastructures and hands-on learning platforms. The project supports the improvement of students' practical skills and helps align academic education with the needs of the modern labor market, particularly in the context of Industry 4.0 and smart manufacturing systems [10].

1.3.2 Remote Control and Monitoring Protocols for PLC and HMI Systems: A Case Study of MODBUS, WMI and VNC (Türkiye)

This study focuses on the use of PLCs in a wide range of industrial automation applications. The PLC serves as the central control unit of machines and systems, acquiring data from sensors and managing output signals. In addition, PLCs can facilitate communication between various devices and sensors within an industrial environment.

In this work, different PLC remote control and communication protocols are investigated, and a remote monitoring and control system is implemented using Windows Management Instrumentation (WMI), RS-485, RS-232, and VNC protocols. A dedicated machine was designed for separating sprues from injected plastic components. The system is equipped with a GMT 396R PLC, a 7-inch touchscreen HMI panel, and a Variable Frequency Drive.

The PLC is remotely accessed via a WMI connection through an Ethernet interface, while the HMI is monitored and controlled using the VNC protocol. The VFD is controlled by the PLC via RS-485 communication, and the HMI is connected to the PLC using the RS-232 MODBUS protocol. This integrated communication architecture provides enhanced flexibility, improves system efficiency, and reduces overall maintenance costs [11].

1.3.3 TwinCAT PLC Web Based Remote Control Laboratories for Mobile and Browser Access (Spain)

This study presents a new approach for developing remote control labs accessible via modern browsers and student devices (PCs, tablets, smartphones). The system combines TwinCAT PLCs for the back-end (closing the feedback loop with industrial real-time control), Easy JavaScript Simulations (EJS) for interactive front-ends (generating HTML5/JavaScript pages), and a Node.js web server for lightweight access management and real-time communication. This solution ensures cross-platform compatibility and robust connectivity. The paper also demonstrates how this strategy overcomes accessibility limitations of existing PID and statefeedback controller labs that previously relied on TwinCAT PLCs with Easy Java Simulations (EJS) applets. The main benefits include enhanced student access from any device without plugins, reduced laboratory maintenance costs, and scalable remote experimentation. This impact extends to bridging the gap between theoretical control education and industrial automation practices, enabling institutions to offer inclusive, hands-on learning experiences even in distance or resourceconstrained settings [12].

1.3.4 Networked PLC-OPC Remote Laboratories for Fuzzy Control and Data Acquisition (Spain)

This paper presents the design and integration of a network combining PLCs, the OPC protocol, and the open-source Easy Java Simulations (EJS) package. A LabVIEW interface and the Java-Internet-LabVIEW (JIL) server complete the data exchange scheme, enabling remote user interaction with the PLC. This integration is a novel contribution to remote control and sensor data acquisition for industrial plants. An experimental application for a DC servomotor fuzzy controller is developed, where users tune parameters online and observe real-time effects via an EJS interface while the PLC executes the control logic. Experimental results confirm the system's feasibility and benefits for engineering remote users.

The main benefits include low-cost open-source tools combined with industrial-grade OPC and PLCs, making advanced remote labs accessible to institutions with limited budgets. The impact extends to democratizing hands-on automation training, bridging theory and practice, and preparing students for Industry 4.0 roles regardless of their physical location [13].

Conclusion

In this chapter, we started from the very basics: what a PLC actually is and how its hardware is built. We paid special attention to Siemens PLCs specifically how they organize memory and data, Understanding this is important because it affects how the PLC can be accessed remotely.

Next, we looked at modern industrial IoT platforms that allow remote access to PLCs, such as the EWON gateway from HMS Networks. These are powerful commercial solutions, but they can be expensive and sometimes lack flexibility. We also reviewed academic research from different universities on remote automation laboratories. In particular, we examined monitoring protocols like MODBUS for industrial communication and VNC for remote desktop access to HMIs.

While several capable tools exist, many solutions are either costly or not fully open. This creates a need for an approach that is affordable, flexible, and works well with standard industrial equipment. This is precisely what our project aims to achieve. In the next chapters, we will explain how we built a lightweight remote programming, supervision, and control solution.

Chapter 2

Hardware, Software and Network Tools

Introduction

This project uses several hardware and software tools to create a secure and efficient communication link between an engineer's laptop and a PLC. The purpose of this chapter is to explain the theoretical concepts and the main components used in this system.

First, we examine the Raspberry Pi 4 Model B, which acts as a server to bridge factory machinery with the engineer's PC. Second, we introduce industrial Ethernet protocols such as PROFINET and S7 communication, which allow devices to communicate with each other, along with the SNAP7 library used for S7 protocol interaction.

Next, the chapter describes the Web Application Stack, including Flask, Nginx, and JSON, which are used to build a local dashboard for data. Finally, we will explain how a VPN creates a secure "tunnel" across the internet to protect the system from cyberattacks while allowing remote access.

2.1 Hardware Architecture

2.1.1 The Raspberry Pi 4 Model B

Raspberry Pi 4 Model B is a compact and powerful single-board computer designed for industrial automation, networking, and IoT applications. It features a quad-core 1.5 GHz processor, Gigabit Ethernet, Wi-Fi, Bluetooth 5.0, USB 3.0, and GPIO pins for easy integration with sensors and industrial devices [15].

Low power consumption and strong connectivity make it perfect for remote industrial automation. Powered by a Debian-based Linux distribution, it provides lightweight, flexible, and efficient system management.

2.1.2 Raspberry Pi OS Lite

Raspberry Pi OS Lite is a lightweight version of Raspberry Pi OS without a graphical interface. It is designed for low-resource devices and is mainly used for servers, networking projects, and embedded systems. It provides high performance, fast boot time, and full command-line control for advanced configurations with iptables preinstalled for network and firewall management [16].

2.1.3 iptables

iptables is a Linux firewall tool used to control network traffic. It filters packets based on rules defined by the administrator, such as IP address, port, and protocol. It is widely used for securing systems, managing VPN traffic, and controlling incoming and outgoing connections efficiently [18].

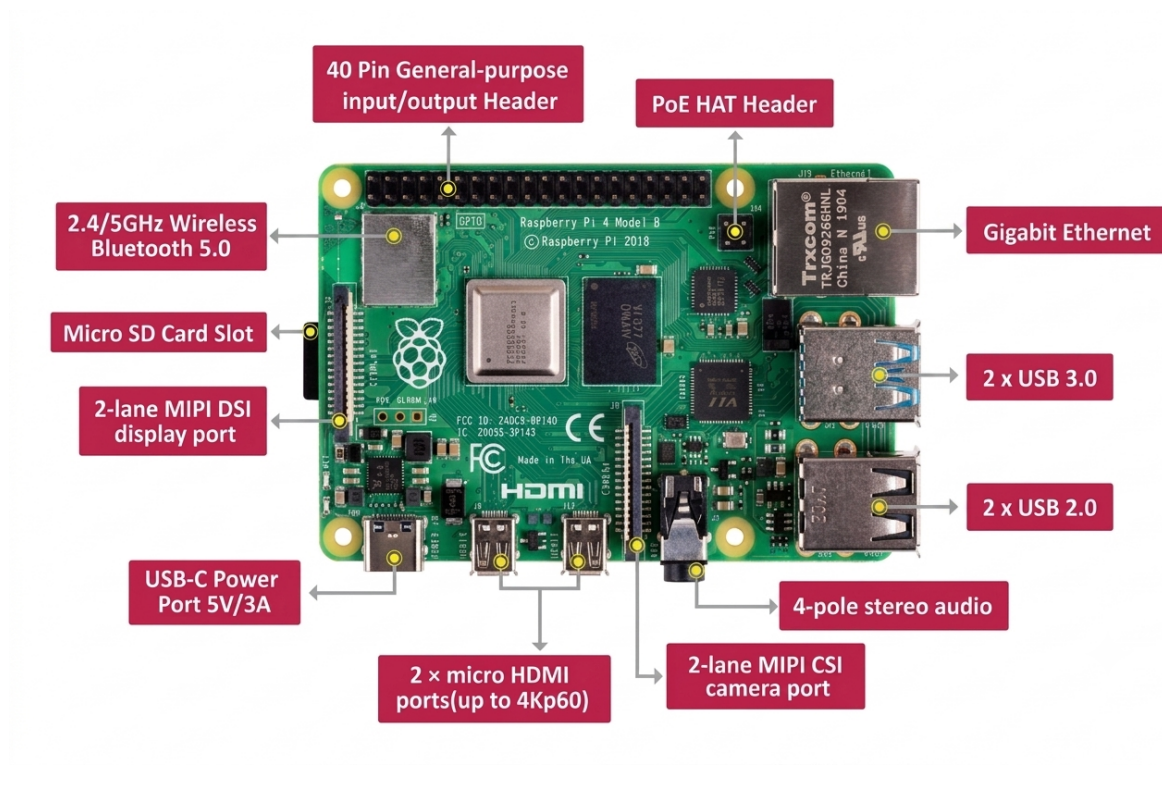


Figure 2.1: The Raspberry Pi 4B [36].

2.2 TIA Portal

TIA Portal, is an integrated engineering platform for programming, configuring, and managing industrial automation systems. It provides a user-friendly interface with advanced tools that simplify automation projects and improve programming efficiency. The platform includes STEP 7 for PLC programming using languages such as Ladder Logic, Function Block Diagram, and Structured Text, WinCC for HMI and process monitoring, and Startdrive for drive configuration. All tools are integrated into a single interface for seamless workflow and efficient project management.

From a networking and communications perspective, TIA Portal simplifies the configuration of standard industrial Ethernet protocols, particularly PROFINET, Modbus TCP, and OPC UA. The software includes a graphical network topology view that allows engineers to drag, drop, and visually connect PLCs, remote I/O modules, and security gateways.

This feature makes it significantly easier to plan data flows, manage IP address assignments, and set up secure industrial communication networks [19].

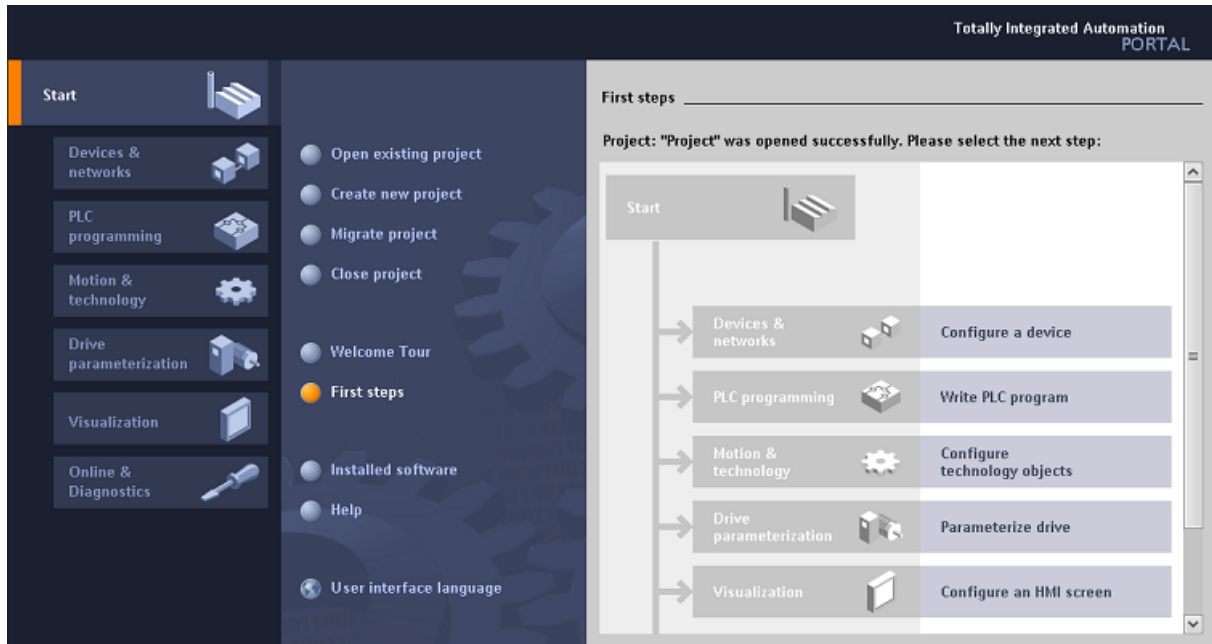


Figure 2.2: TIA PORTAL boot interface [35].

2.3 Networking

2.3.1 Industrial communication and protocols

Industrial communication refers to the specialized systems used to exchange data between devices in an automated environment, such as PLCs, HMIs, sensors, and actuators. The primary goal of industrial communication is to provide a robust, secure, and organized framework for devices to "talk" to each other and ensure the efficient and safe operation of the production process. Historically, industrial communication relied on serial-based "Fieldbus" systems like PROFIBUS, which used a master-slave principle where one central device (the master) controlled the communication of all other devices (slaves). Modern systems have largely shifted toward Ethernet-based networks, which offer higher speeds and better integration with higher-level enterprise systems [20].

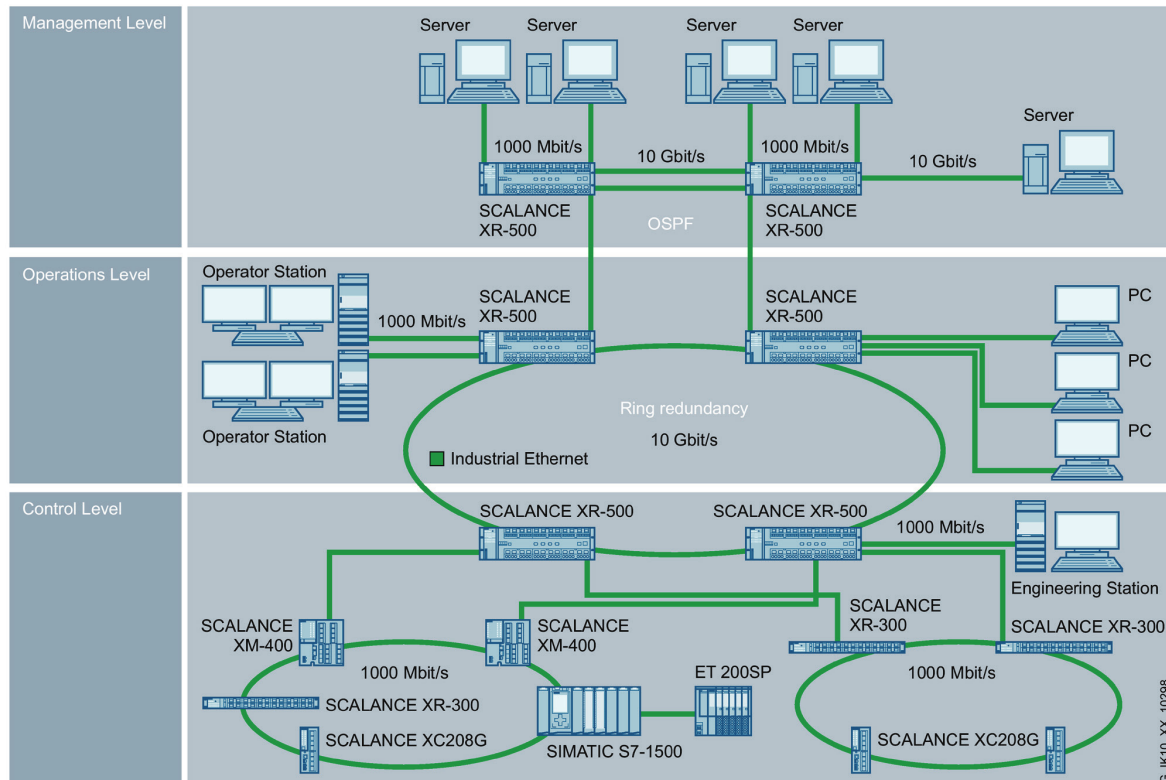


Figure 2.3: Industrial network example [21].

2.3.2 Industrial Ethernet Standards

Industrial Ethernet is the use of Ethernet technology in a manufacturing environment. Although it is based on the same core standards as office and home Ethernet, Industrial Ethernet is reinforced with robust hardware and specialized protocols to meet industrial requirements. Because standard Ethernet was not originally designed for real-time communication, it relies on a best-effort delivery method that can allow data collisions. To address this, Industrial Ethernet uses switched Ethernet, where switches create direct point-to-point connections between devices, reducing collisions and supporting full-duplex communication. As a result, the network becomes faster and more predictable. Siemens implements Industrial Ethernet through systems such as SIMATIC NET and the PROFINET standard, offering high bandwidth, long-distance communication through fiber optics, and flexible topologies such as star, line, or ring. This makes it possible to connect factory-floor equipment with management systems and support Industry 4.0 applications, according to the OSI and TCP/IP models [20].

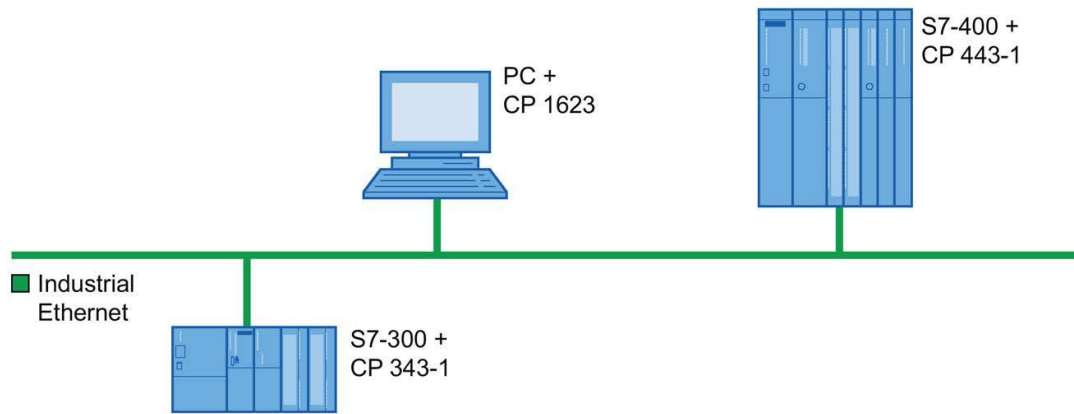


Figure 2.4: Typical System Configuration for Ethernet [20].

2.3.3 OSI Model (ISO/OSI Reference Model)

The OSI model is a theoretical framework used to understand and design network communications by dividing the process into seven distinct layers. Each layer has a specific job and interacts with the layers directly above and below it. In industrial automation, this model helps engineers troubleshoot problems and ensure that different products can work together. In many industrial protocols, some layers are combined or skipped to improve speed. For example, the TCP/IP model collapses the seven OSI layers into four, omitting some layers that are not required for general-purpose networking [20].

7	Application Layer	Human-computer interaction layer, where applications can access the network services
6	Presentation Layer	Ensures that data is in usable format and is where data encryption occurs
5	Session Layer	Maintains connections and is responsible for controlling ports and sessions
4	Transport Layer	Transmits data using transmission protocols including TCP and UDP
3	Network Layer	Decides which physical path the data will take
2	Data Link Layer	Defines the format of data on the network
1	Physical Layer	Transmits raw bit stream over the physical medium

Figure 2.5: The 7 Layers of the OSI Model [30].

2.3.4 TCP/IP

TCP/IP is a set of transport protocols used mainly on Ethernet networks. In an industrial context, it allows different types of controllers and computers to exchange data regardless of their manufacturer [20].

1. IP : This protocol functions at the Network Layer (Layer 3). Its main job is addressing and routing. Every device on a TCP/IP network is assigned a unique IP address , which allows data packets to find their way to the correct destination across multiple network segments.
2. TCP : This protocol functions at the Transport Layer (Layer 4). It is "connection-oriented," meaning it establishes a formal link between two devices before sending data. TCP is highly reliable because it checks if data arrived correctly; if a packet is lost or corrupted, TCP asks for it to be sent again.

In industrial communication, ISO-on-TCP is commonly used in industrial communication. Pure TCP/IP sends data as an unstructured stream, which is not ideal for large messages. ISO-on-TCP adds the ability to split large amounts of data into smaller packets and reassemble them, making it better for industrial applications. The S7 protocol can use ISO-on-TCP for communication over Ethernet [20].

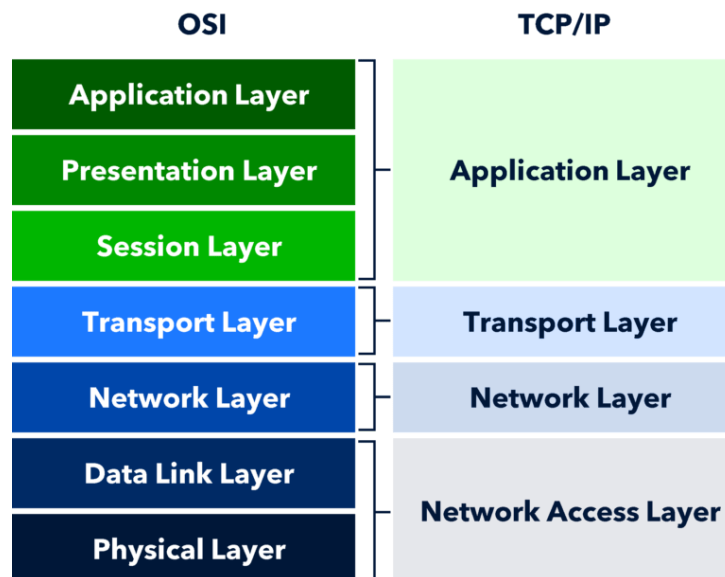


Figure 2.6: Mapping of OSI layers to the layers of the TCP/IP Model [30].

2.3.5 S7 Communication (ISO-TCP) and Snap7 library

S7 Communication (often called S7 comm) is a proprietary Siemens protocol used specifically for data exchange between SIMATIC S7 controllers and other Siemens devices like HMIs or PCs. It is a high-level protocol that allows a user to read or write variables, start or stop the PLC, and perform diagnostic checks. Modern S7 communication usually runs over Ethernet using a method called ISO-on-TCP. This approach combines the advantages of the older ISO transport protocol with the flexibility of modern TCP/IP. Essentially, it "wraps" the industrial S7 data inside a standard TCP packet, allowing it to travel across standard network hardware and routers. One of the biggest advantages of S7 communication is its integration with Siemens software like STEP 7 or TIA Portal, but because it is a closed protocol, it is not easily integrated with external software. For this reason, the Snap7 library is used [21].

Snap7 is an open-source communication library written in C/C++ extended support for Python that implements the S7 protocol for accessing Siemens PLC data. It allows a PC to access data blocks, I/O areas, flags, timers, and counters of S7 PLCs. Snap7 operates over S7 ISO-on-TCP on port 102 and includes three native components: Client, Server, and Partner [22].

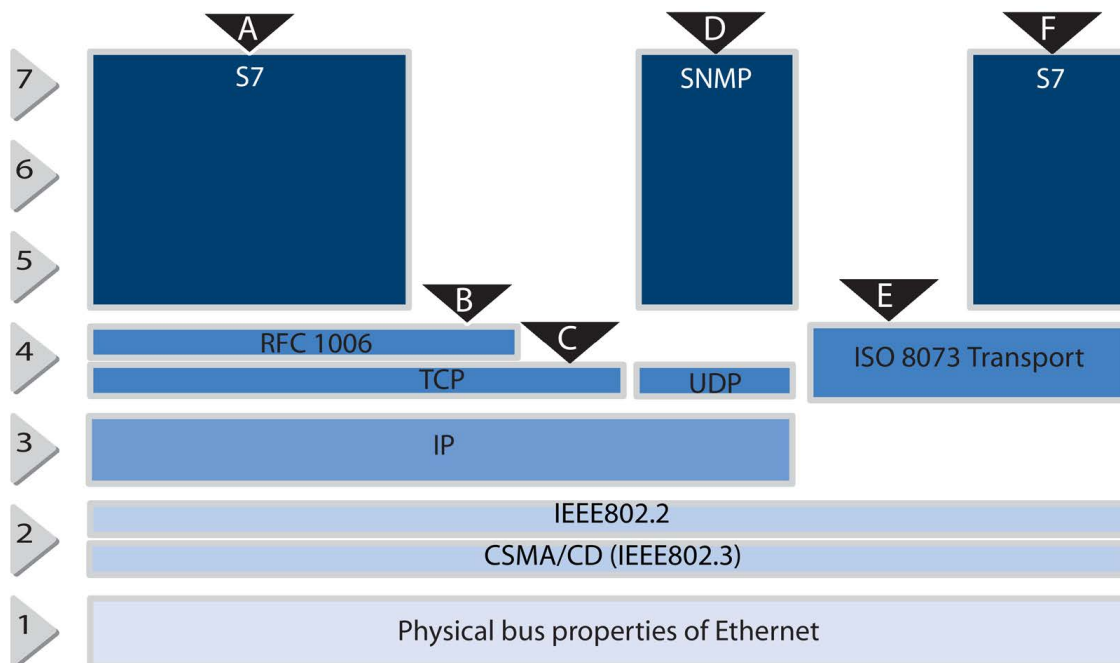


Figure 2.7: S7 comm in the OSI Reference Model [17].

2.4 The Web Application Stack

Once the PLC data is extracted using SNAP7 , we need a backend system to organize this data, create API endpoints, and serve the visual dashboard to remote users. Writing a web server from scratch in Python to handle HTTP requests would be incredibly inefficient and complex , so for this we need Flask Framework.

2.4.1 Flask Framework

Flask is a micro web framework written in Python. It provides the essential tools and libraries needed to build a web application. Because it is lightweight, it is perfect for IoT and industrial projects. While keeping application code fast, maintainable, and decoupled from heavy third-party software dependencies. Flask's built-in development server is designed only for local testing. It can only handle one request at a time. If multiple technicians try to view the remote PLC dashboard simultaneously, the built-in server will freeze, crash, or experience severe delays [23].

2.4.2 Gunicorn

Green Unicorn is a WSGI server designed for Unix-based operating systems. While development frameworks like Flask provide built-in web servers, those servers cannot handle heavy production traffic safely. Gunicorn resolves this limitation by creating multiple worker processes to handle hundreds of concurrent user requests smoothly. This process management guarantees that if one user request crashes, the web interface remains online and fully operational for other industrial control clients. While Gunicorn manages Python processes, it is not built to face the public internet directly. It lacks advanced security features, cannot handle SSL encryption (HTTPS) efficiently, and struggles with serving static files like images or CSS stylesheets quickly.

2.4.3 Nginx

Nginx is a high-performance web server and reverse proxy designed to handle massive internet traffic with a minimal memory footprint. In a standard web architecture, Nginx sits at the edge of the network, acting as a secure gateway that receives all incoming user connections. It handles SSL encryption, serves static assets, and forwards dynamic requests downstream to application servers like Gunicorn. Its event-driven architecture ensures exceptional security, low latency, and optimal resource usage on industrial host systems.

2.4.4 JSON

JSON (JavaScript Object Notation) is a lightweight, file-based data storage format that requires no database engine, no installation, and no configuration. Data is simply stored as plain ".json" files on disk and read or updated directly within the Flask application using Python's built-in 'json' module. This simplicity makes it a natural fit for edge computing devices and lightweight industrial dashboards [26].



Figure 2.8: Global architecture for The Web Application Stack [32].

2.5 Remote Access and Tunneling

After PLC data have been transferred to a web dashboard, the next step is to establish a remote connection that enables both real-time monitoring of the dashboard and visibility of the PLC on the engineer's network. However, direct access over the public internet exposes critical infrastructure to potential cyberattacks. For this reason, a VPN must be employed to secure the communication link.

2.5.1 Virtual Private Networks (VPNs)

A VPN is a foundational technology in modern industrial automation and telecommunications. It establishes a secure, logical communication channel, often called a tunnel, across an insecure public infrastructure like the Internet. By using cryptographic protocols, a VPN encapsulates and encrypts data payloads. This mechanism ensures data confidentiality, user authentication, and packet integrity, preventing unauthorized actors from eavesdropping on sensitive industrial processes. As networks expand and different industrial protocols appear, it becomes necessary to choose between different layers of the OSI model, which affects how field devices communicate across a tunnel. This creates a need to choose between bridging network segments completely (Layer 2) or routing specific data packets (Layer 3).

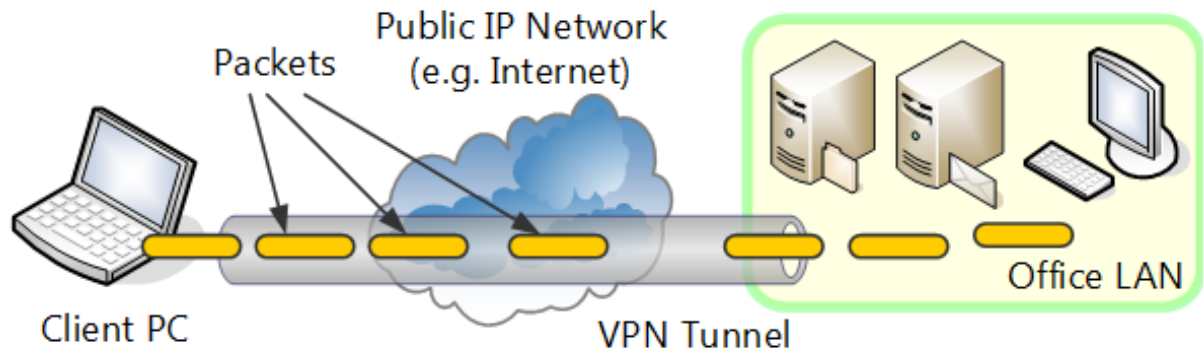


Figure 2.9: VPN connection to Virtual Hub by SoftEther VPN Client [27].

2.5.2 Layer 2 vs Layer 3 VPNs

To overcome the limitations of traditional VPNs, we must understand two main types: Layer 2 and Layer 3 VPNs.

- A Layer 2 VPN works at the Ethernet level. It sends complete Ethernet frames, including all types of protocols inside. This makes it much more flexible because it acts like a virtual network cable. For industrial automation, Layer 2 VPNs are powerful because they can connect machines that don't speak pure IP.
- A Layer 3 VPN works only with IP packets like those used by the Internet Protocol. It connects different networks by routing IP addresses. This is common, but it cannot carry non-IP traffic such as
 1. PROFINET LLDP
 2. PROFINET DCP
 3. PROFINET IRT
 4. PROFINET RT

2.5.3 SoftEther VPN

SoftEther VPN is a powerful, open-source, multi-protocol VPN software suite developed at the University of Tsukuba in Japan. It was designed as an alternative to classic VPN solutions and stands out because it can handle multiple protocols simultaneously, including OpenVPN, IPsec, L2TP, SSTP, and its own SoftEther SSL-VPN protocol. This flexibility allows administrators to maintain a single server while supporting diverse remote clients. Additionally, SoftEther features native optimizations for firewall traversal, dynamic DNS management, and comprehensive packet logging. SoftEther stands out

because it can natively support both Layer 2 and Layer 3 connections, but these connections are often blocked by network gateways, firewalls, or NAT devices. This leads to a need for a method to carry Ethernet traffic over standard web protocols [27].

2.5.4 Ethernet over HTTP

SoftEther VPN uses this method to wrap complete Ethernet frames inside normal HTTP or HTTPS packets. The effectiveness of this method stems from the fact that most firewalls and proxy servers allow HTTP (port 80) and HTTPS (port 443) traffic because they are used for web browsing. By disguising VPN traffic as regular web traffic, the VPN can pass through almost any gateway without needing special configuration. The system administrator does not need to configure additional inbound firewall rules. This makes deployment easy and secure. The VPN session looks like any other secure website connection. However, even with this powerful method, connecting two computers directly across the Internet (a peer-to-peer connection) still faces challenges, especially when both are hidden behind routers [29].

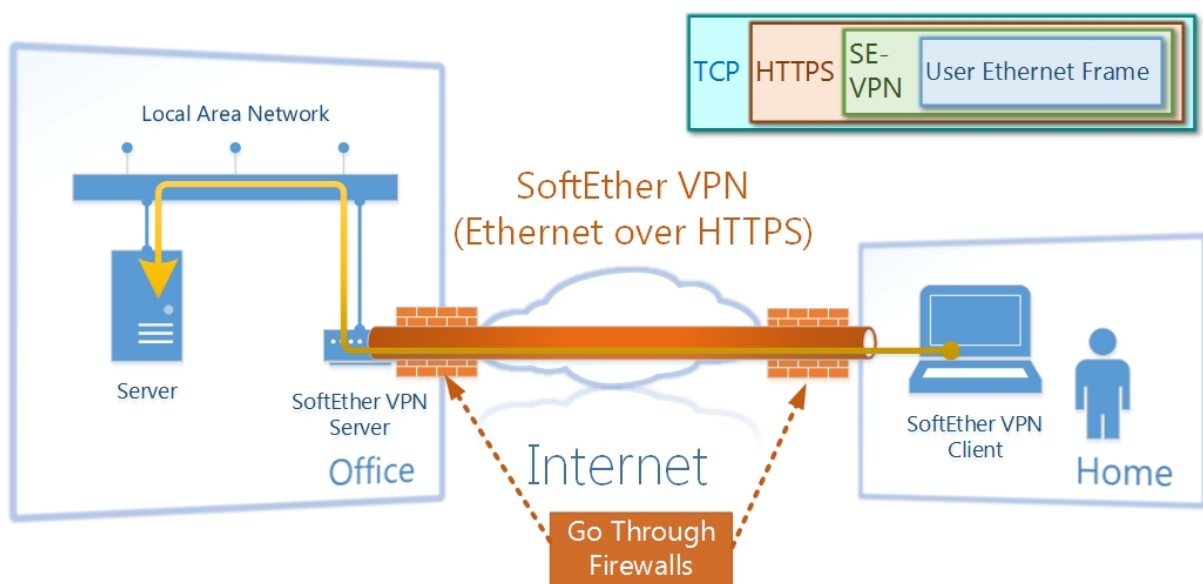


Figure 2.10: Ethernet over HTTP [29].

2.5.5 VPN session

For remote industrial monitoring and control, a VPN session is commonly established to create a secure communication channel between the operator and field equipment. To achieve low latency and efficient real-time data exchange, the VPN connection ideally operates in a Peer-to-Peer (P2P) manner, where both devices communicate directly without relying on a centralized relay server.

A (P2P) connection enables two devices on a network to communicate directly without routing traffic through a central server. In industrial VPN applications, P2P communication is important because it minimizes latency and avoids bandwidth bottlenecks, which is essential for real-time monitoring and control of field equipment.

However, establishing direct P2P communication is often difficult due to Network Address Translation (NAT). NAT is a networking mechanism that maps multiple private IP addresses to a single public IP address in order to conserve IPv4 address space. While effective for address management and security, NAT blocks unsolicited incoming traffic, preventing devices behind different networks from directly reaching each other.

To bypass this limitation, traditional networks commonly use port forwarding, where incoming traffic on a specific public IP address and port is redirected to a designated internal device. Nevertheless, this solution becomes ineffective in networks using Carrier-Grade NAT (CGNAT). In CGNAT environments, Internet Service Providers (ISPs) share a limited number of public IP addresses among many customers, meaning users no longer have administrative control over the public-facing router and cannot configure port forwarding rules.

This issue is especially common in mobile data networks (4G/5G), where most operators deploy CGNAT by default. As a result, devices connected through mobile networks cannot usually accept direct incoming connections, making traditional P2P VPN establishment impossible.

To address this issue, a bridge server equipped with a static public IP address can be deployed as an intermediary node. Both peers initiate outbound connections toward the bridge server, allowing the VPN tunnel to be established and maintained even in CGNAT-restricted environments.

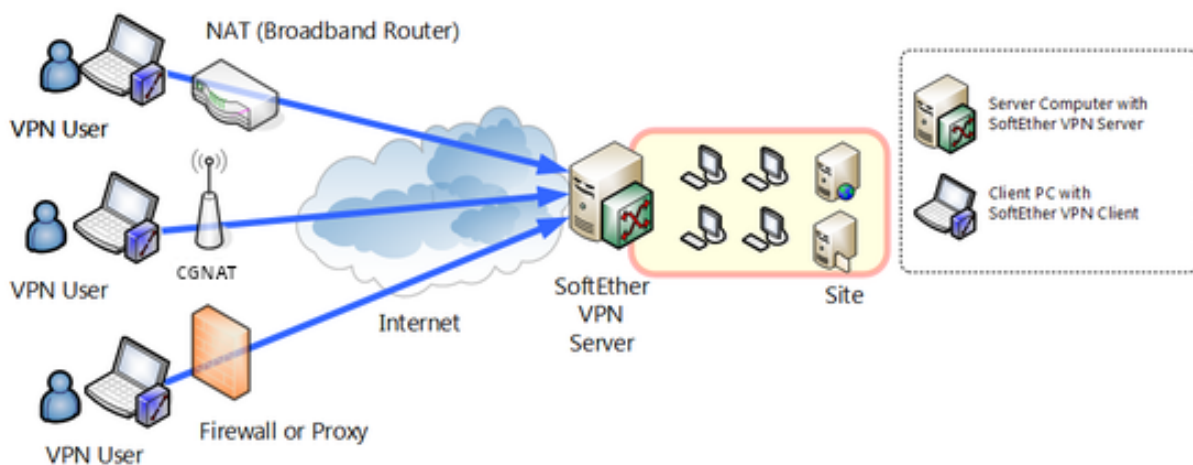


Figure 2.11: Passage through NAT proxy server or CGNAT by SoftEther VPN [27].

2.5.6 Virtual Hub and Local Bridge

A Virtual Hub is like a virtual Ethernet switch running inside the SoftEther VPN Server. It can receive connections from many VPN clients and manage all Ethernet frames between them. The Local Bridge function connects this Virtual Hub directly to a physical network adapter on the server computer. In simple terms, the Local Bridge acts as a connection between the virtual VPN network and the real physical LAN. This allows remote VPN clients to behave as if they were physically plugged into the PLC network. In the proposed architecture, the SoftEther VPN Client software must be installed on the engineer's workstation. [28].

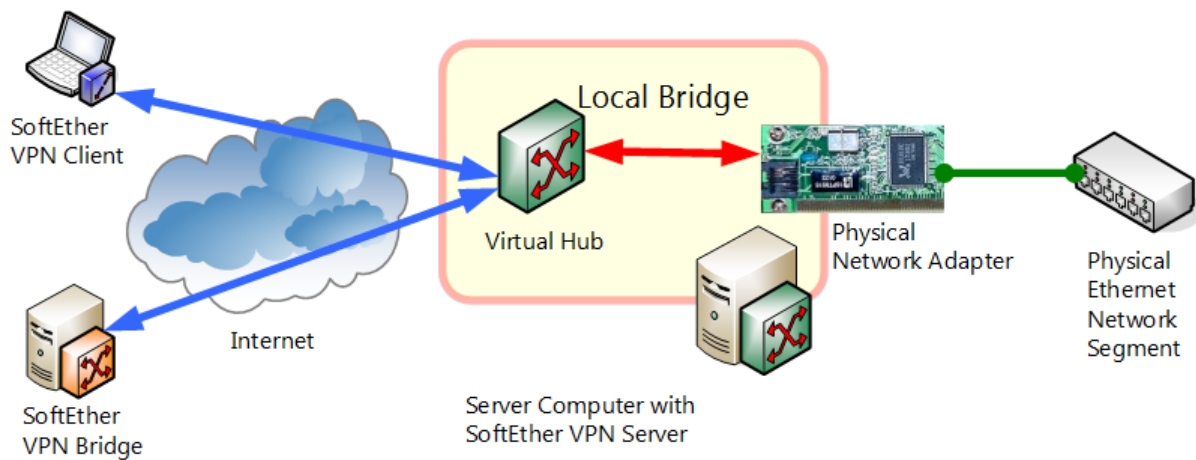


Figure 2.12: Virtual Hub and Local Bridge network [28].

2.5.7 SoftEther VPN client and Virtual Network adapter

The SoftEther VPN Client is the dedicated end-user application that allows remote endpoints to connect securely to a distant Virtual Hub over the Internet. Upon installation, the client software injects a specialized virtual network adapter into the host operating system, widely known as a TAP interface. The operating system and industrial software applications recognize this TAP interface as a standard, physical Ethernet network card. When an automation application sends a packet, the TAP interface captures the raw Layer 2 Ethernet frame, passes it to the SoftEther client for encryption and HTTP encapsulation, and transmits it across the internet tunnel. The remote Virtual Hub receives, decrypts, and bridges this frame onto the physical factory floor. This complete, software-defined ecosystem allows an automation engineer to perform remote diagnostic programming, execute updates, and control distant industrial machinery with the same ease as being physically connected via a local patch cable [28].

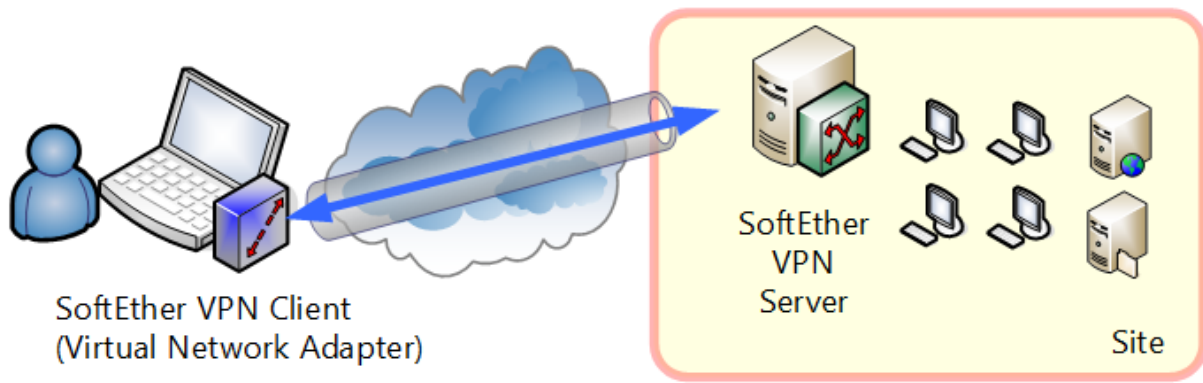


Figure 2.13: VPN connection to Virtual Hub by SoftEther VPN Client [28].

Conclusion

This chapter presented the main hardware, software, and networking technologies used to develop the proposed remote industrial access solution. The Raspberry Pi 4 was introduced as the edge computing platform, while TIA Portal and the Snap7 library were presented as the key tools for PLC programming and data exchange. The chapter also reviewed the industrial communication concepts required for the system, including Ethernet-based networking, the OSI model, TCP/IP, and S7 communication.

In addition, the web application stack composed of Flask, Gunicorn, Nginx, and JSON was described as the foundation for data processing and visualization. Finally, the principles of VPN technology were examined, with particular attention to Layer 2 communication and the SoftEther VPN architecture, which enables transparent and secure remote access to industrial devices while preserving compatibility with native Siemens protocols. These theoretical foundations provide the basis for the implementation and validation of the proposed solution presented in the following chapter.

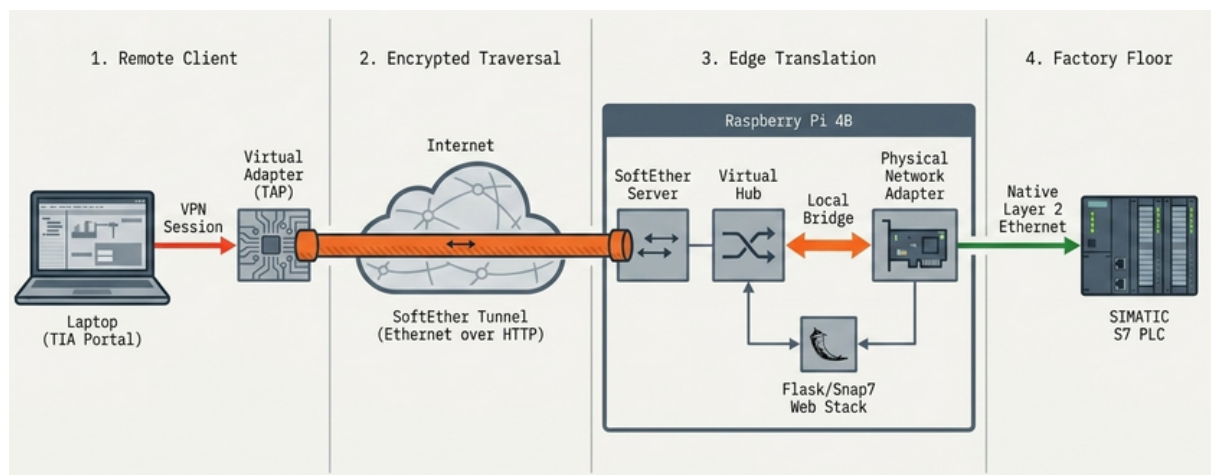


Figure 2.14: System Global Architecture [32].

Chapter 3

Application

Introduction

This chapter presents the practical implementation of the proposed remote industrial access solution. It describes the deployment of the SoftEther VPN infrastructure on a Raspberry Pi and the configuration steps required to establish secure Layer 2 connectivity between a remote engineer and a Siemens PLC.

The chapter then focuses on the development of a web-based supervision platform using Flask and Snap7. After validating PLC communication through a series of functional tests, the PLC Monitor application is introduced as a solution for real-time monitoring and control of industrial processes through a standard web browser. The chapter concludes with the experimental results obtained from the implemented system.

3.1 Install and compile

Begin by refreshing the package index and installing the necessary build tools. Next, download the **SoftEther VPN Server** package designed for Raspberry Pi (ARM architecture). When the download finishes, unpack the archive. This will create a directory called **vpnsrvr**. Switch your terminals working directory to this folder. Now compile the servers source code. If the compilation completes without errors, it will produce two executable files: **vpncmd** and **vpnsrvr**.

```
pfe@raspberrypi:~/Downloads $ sudo apt install build-essential gcc make wget tar -y
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
build-essential is already the newest version (12.9).
gcc is already the newest version (4:10.2.1-1).
make is already the newest version (4.3-4.1).
tar is already the newest version (1.34+dfsg-1+deb11u1).
wget is already the newest version (1.21-1+deb11u2).
The following packages were automatically installed and are no longer required:
  libmanette-0.2-0 libopengl0 libre2-9 libwpe-1.0-1 libwpebackend-fdo-1.0-1 linux-headers-5.10.0-42-amd64:sse3-support
Use 'sudo apt autoremove' to remove them.
0 upgraded, 0 newly installed, 0 to remove and 17 not upgraded.
pfe@raspberrypi:~/Downloads $ ls
softether-vpnsrvr-v4.44-9807-rtm-2025.04.16-linux-arm64-64bit.tar.gz
pfe@raspberrypi:~/Downloads $ tar xzf softether-vpnsrvr-v4.44-9807-rtm-2025.04.16-linux-arm64-64bit.tar.gz
pfe@raspberrypi:~/Downloads $ ls
softether-vpnsrvr-v4.44-9807-rtm-2025.04.16-linux-arm64-64bit.tar.gz  vpnsrvr
pfe@raspberrypi:~/Downloads $ cd vpnsrvr/
pfe@raspberrypi:~/Downloads/vpnsrvr $ make
-----
SoftEther VPN Server (Ver 4.44, Build 9807, ARM 64bit) for Linux Build Utility
Copyright (c) SoftEther Project at University of Tsukuba, Japan. All Rights Reserved.
-----
Copyright (c) all contributors on SoftEther VPN project in GitHub.
Copyright (c) Daiyuu Nobori, SoftEther Project at University of Tsukuba, and SoftEther Corporation.
```

Figure 3.1: Install and compile commands [34].

Next, go back to the previous folder and move the vpnsrvr folder to **/usr/local/**. This is the standard Linux location for programs you install manually. After moving it,

enter the installation folder and set the needed file permissions.

The command **chmod 600 *** gives only the owner permission to read and change the files. This protects configuration files, certificates, keys, logs, and other sensitive information from people who are not allowed to access them. The command **chmod 700 vpnserver** makes sure that only the root user or the file owner can run these programs. Since **vpnserver** is the admin console, normal users should not be able to access it. After you set the permissions, start the VPN server with **sudo ./vpnserver start**.

```

This HTML5 page is obviously under construction, and your HTML5 development contribution is very appreciated.
-----
make[1]: Leaving directory '/home/pfe/Documents/vpnserver'
pfe@raspberrypi:~/Documents/vpnserver $ ls
Authors.txt  hamcore.se2  Makefile      ReadMeFirst_Important_Notices_ja.txt  vpnserver
chain_certs  lang.config  ReadMeFirst_Important_Notices_cn.txt  ReadMeFirst_License.txt
code         lib          ReadMeFirst_Important_Notices_en.txt  vpnserver
pfe@raspberrypi:~/Documents/vpnserver $ cd ..
pfe@raspberrypi:~/Documents $ sudo mv vpnserver /usr/local/
pfe@raspberrypi:~/Documents $ cd /usr/local/vpnserver/
pfe@raspberrypi:~/usr/local/vpnserver $ sudo chmod 600 *
pfe@raspberrypi:~/usr/local/vpnserver $ sudo chmod 700 vpnserver
pfe@raspberrypi:~/usr/local/vpnserver $ sudo ./vpnserver start
The SoftEther VPN Server service has been started.

Let's get started by accessing to the following URL from your PC:

https://192.168.1.78:5555/
or
https://192.168.1.78/

Note: IP address may vary. Specify your server's IP address.
A TLS certificate warning will appear because the server uses self signed certificate by default. That is natural. Continue with ignoring the TLS warning.

pfe@raspberrypi:~/usr/local/vpnserver $

```

Figure 3.2: Set permissions and start server commands [34].

3.2 Initial Server Configuration

- **Select option 1** for “Management of VPN Server or VPN Bridge”.
This chooses the correct mode for managing the VPN server.
- **Leave the hostname or IP address empty** when prompted.
Pressing Enter connects the console to the local server; no external address is required.
- **Leave the Virtual Hub Name empty** when prompted.
A virtual hub will be created later, so no name is entered at this stage.
- **Set a password** for the VPN server’s administration settings.
This password protects the server’s management functions.
- **Create a virtual hub named HUB1** and set a password for the hub when prompted.
The hub serves as a logical container for users and connections; the password secures access to the hub.

- **Select HUB1** to manage it.
This allows further configuration specific to that hub.
- **Create a user named USER1.**
This user will be permitted to connect to the VPN.
- **Leave the Assigned Group Name, User Full Name, and User Description fields blank** when prompted.
These fields are optional and not required for basic user creation.
- **Set a password** for the user.
This password is used by USER1 to authenticate when connecting to the VPN.

```
pfe@raspberrypi:/usr/local/vpnserver $ sudo ./vpncmd
vpncmd command - SoftEther VPN Command Line Management Utility
SoftEther VPN Command Line Management Utility (vpncmd command)
Version 4.44 Build 9807 (English)
Compiled 2025/04/16 04:50:46 by buildsan at crosswin with OpenSSL 3.0.9
Copyright (c) 2012-2025 SoftEther VPN Project. All Rights Reserved.

By using vpncmd program, the following can be achieved.

1. Management of VPN Server or VPN Bridge
2. Management of VPN Client
3. Use of VPN Tools (certificate creation and Network Traffic Speed Test Tool)

Select 1, 2 or 3: 1

Specify the host name or IP address of the computer that the destination VPN Server or VPN Bridge is operating on.
By specifying according to the format 'host name:port number', you can also specify the port number.
(When the port number is unspecified, 443 is used.)
If nothing is input and the Enter key is pressed, the connection will be made to the port number 8888 of localhost (this computer).
Hostname of IP Address of Destination:

If connecting to the server by Virtual Hub Admin Mode, please input the Virtual Hub name.
If connecting by server admin mode, please press Enter without inputting anything.
Specify Virtual Hub Name:
Connection has been established with VPN Server "localhost" (port 443).

You have administrator privileges for the entire VPN Server.

VPN Server>ServerPasswordSet
ServerPasswordSet command - Set VPN Server Administrator Password
Please enter the password. To cancel press the Ctrl+D key.

Password: ****
Confirm input: ****

The command completed successfully.
```

Figure 3.3: Enter vpncmd and set server password commands [34].

```

VPN Server>HubCreate HUB1
HubCreate command - Create New Virtual Hub
Please enter the password. To cancel press the Ctrl+D key.

Password: ****
Confirm input: ****

The command completed successfully.

VPN Server>Hub HUB1
Hub command - Select Virtual Hub to Manage
The Virtual Hub "HUB1" has been selected.
The command completed successfully.

VPN Server/HUB1>UserCreate USER1
UserCreate command - Create User
Assigned Group Name:

User Full Name:

User Description:

The command completed successfully.

VPN Server/HUB1>UserPasswordSet USER1
UserPasswordSet command - Set Password Authentication for User Auth Type and Set Password
Please enter the password. To cancel press the Ctrl+D key.

Password: ****
Confirm input: ****

The command completed successfully.

VPN Server/HUB1>

```

Figure 3.4: Create Hub and User commands [34].

3.3 Disabling Third-Party Services

To keep the VPN server fully standalone and independent from external services, some optional SoftEther features should be disabled.

3.3.1 VPN Azure

VPN Azure is a Microsoft relay service used by SoftEther as a bridge server. Although it is disabled by default, its status can be confirmed with the `VpnAzureGetStatus` command. To disable it manually, run `VpnAzureSetEnable` and enter "no" when prompted.

3.3.2 Dynamic DNS

Dynamic DNS is a SoftEther feature that automatically tracks and publishes your public IP address to the client whenever it changes. To disable it, edit the `vpnserver.config` file and locate the `DdnsClient` section and change the `disabled` property from `false` to `true`.

```

GNU nano 5.4                                vpn_server.config *
# Software Configuration File
# -----
# You may edit this file when the VPN Server / Client / Bridge program is not running.
#
# In prior to edit this file manually by your text editor,
# shutdown the VPN Server / Client / Bridge background service.
# Otherwise, all changes will be lost.
#
declare root
{
    uint ConfigRevision 11
    bool IPsecMessageDisplayed false
    string Region $
    bool VgsMessageDisplayed false

    declare DDnsClient
    {
        bool Disabled true
    }
    declare IPsec
    {
        bool EtherIP_IPsec false
        string IPsec_Secret vpn
        string L2TP_DefaultHub DEFAULT
        bool L2TP_IPsec false
        bool L2TP_Raw false

        declare EtherIP_IDSettingsList
        {
        }
    }
}

```

Figure 3.5: Disable Dynamic DNS in `vpnserver.config` [34].

3.3.3 NAT Traversal

NAT Traversal allows VPN connections to bypass routers and firewalls automatically. To disable it, locate `DisableNatTraversal false` in `vpnserver.config` and replace it with `bool DisableNatTraversal true`.

```

GNU nano 5.4                                vpn_server.config *
    declare Listener3
    {
        bool DisabledDos false
        bool Enabled true
        uint Port 5555
    }
}
declare LocalBridgeList
{
    bool DoNotDisableOffloading false
}
declare ServerConfiguration
{
    bool AcceptOnlyTls true
    uint64 AutoDeleteCheckDiskFreeSpaceMin 104857600
    uint AutoDeleteCheckIntervalSecs 300
    uint AutoSaveConfigSpan 300
    bool BackupConfigOnlyWhenModified true
    string CipherName AES128-SHA
    uint CurrentBuild 9807
    bool DisableCoreDumpOnUnix false
    bool DisableDeadLockCheck false
    bool DisableDosProction false
    bool DisableGetHostNameWhenAcceptTcp false
    bool DisableIntelAesAcceleration false
    bool DisableIPsecAggressiveMode false
    bool DisableIPv6Listener false
    bool DisableJsonRpcWebApi false
    bool DisableNatTraversal true
    bool DisableOpenVPNServer false
    bool DisableSessionReconnect false
}

```

Figure 3.6: Disable NAT Traversal in `vpnserver.config` [34].

3.4 Connect engineer PC

To connect from the engineer's PC, install the SoftEther VPN Client and create a virtual network adapter. Then, add a new VPN connection using the following settings:

- **Fill the settings name** with a chosen identifier.
This name is used to recognize the connection within the client.
- **Enter the server's public IP address** as the host name.
The client needs this address to locate the VPN server on the internet.
- **Keep the port number at 443.**
Port 443 is the standard port for HTTPS and is typically open on most firewalls.
- **Enter HUB1** as the HUB Name.
This specifies which virtual hub on the server the client should connect to.
- **Enter USER1** as the user name.
This identifies the user account created on the VPN server.
- **Enter the user's password** that was set previously.
The password authenticates the user and grants access to the VPN.

In the end, double-click on the settings name to start the connection.

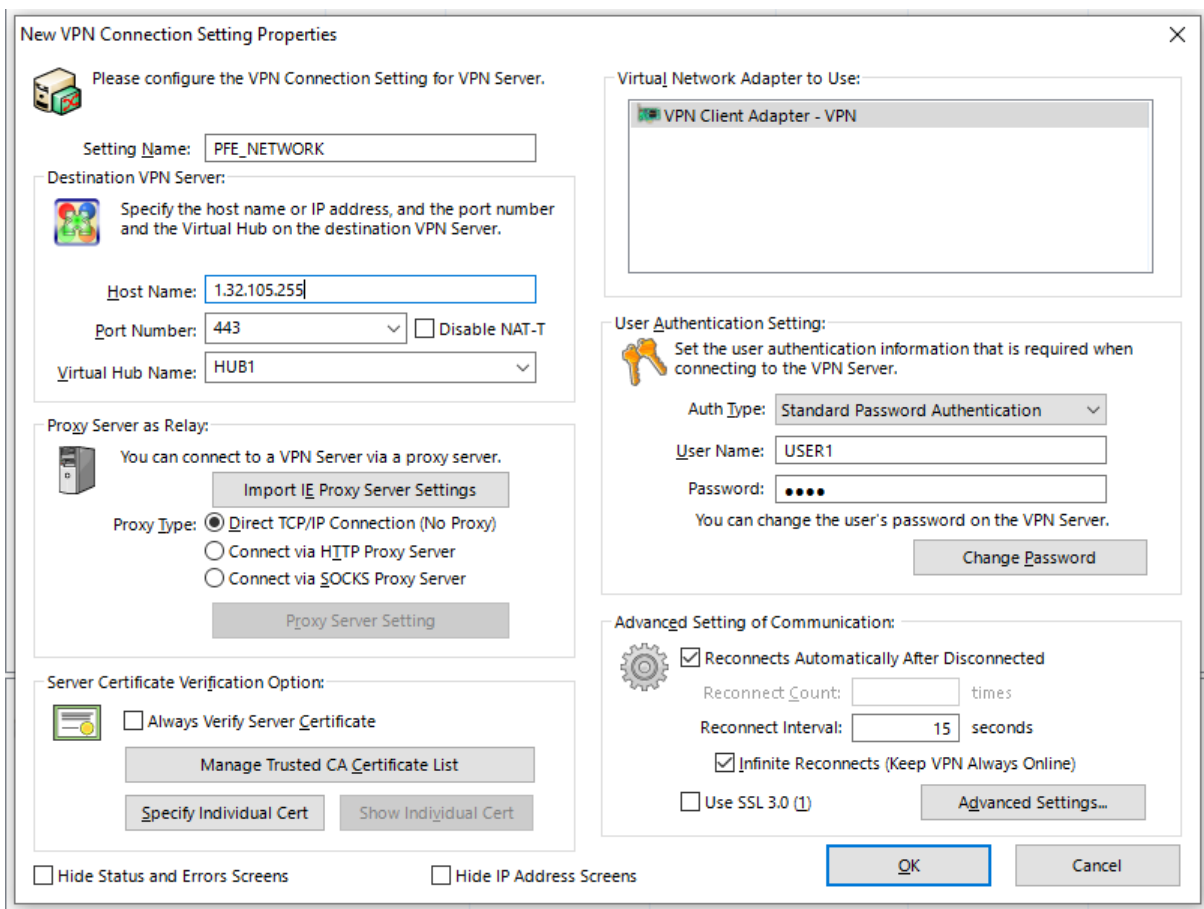


Figure 3.7: Create new VPN connection interface [38].

3.5 TIA Portal adapter problem

When the SoftEther VPN Client creates a virtual network adapter, issues arise in the TIA Portal software. The specific problems are described below.

Observed Behavior

- **The “Access Devices” window in TIA Portal does not list the virtual adapter.**

This window scans the network for available devices, such as PLCs, and to do that you need first to select which network interface you want to look through. In our case it does not list the virtual adapter created by SoftEther VPN Client.

- **The online access function within the TIA Portal project tree lists the virtual adapter but cannot use it.**

Unlike the “Access Devices” window, the online access feature does list the virtual adapter but it is unable to establish a connection and retrieve device information via the virtual VPN adapter and an error message is displayed:

Online : Function not implemented in loaded communication driver

This error indicates that the communication driver loaded by TIA Portal for the virtual adapter does not support the function being requested. The driver may lack the necessary capabilities to handle full two-way communication, or it may be incompatible with the virtual network interface created by SoftEther VPN Client.

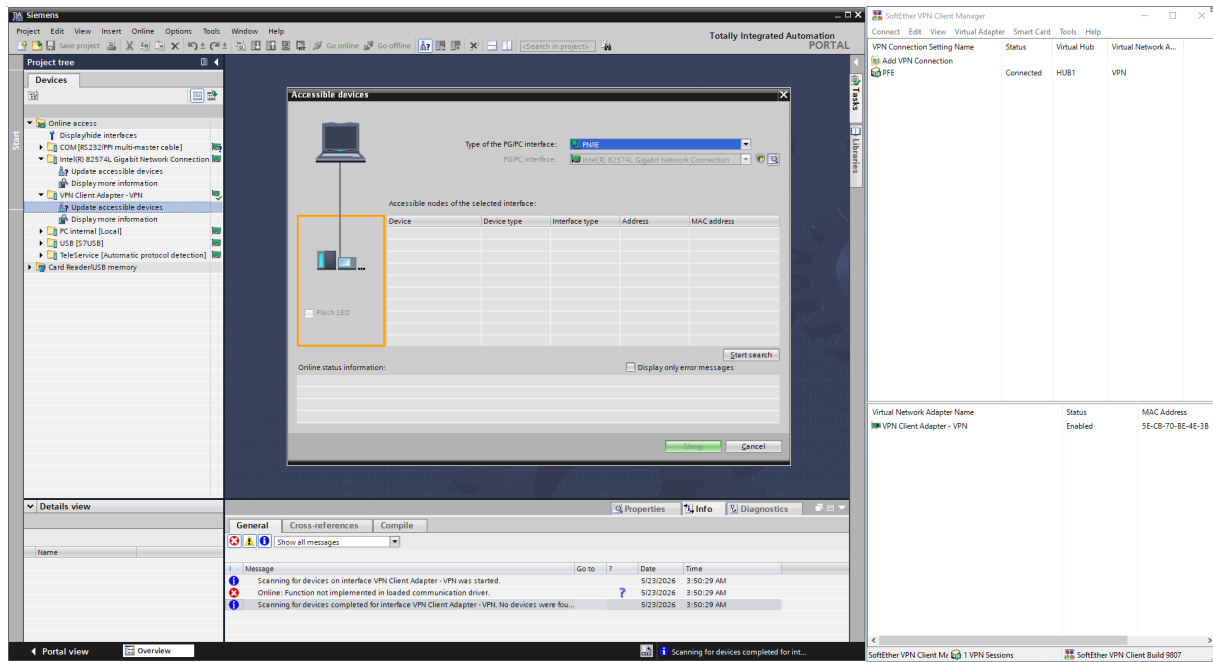


Figure 3.8: TIA portal adapter problem [35].

The solution is to change the driver description of the virtual adapter in the Windows Registry Computer\HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Control\Class\{4de36e972-e325-11ce-bfc1-08002be10318}\001 DriverDesc so that TIA Portal can load the correct driver.

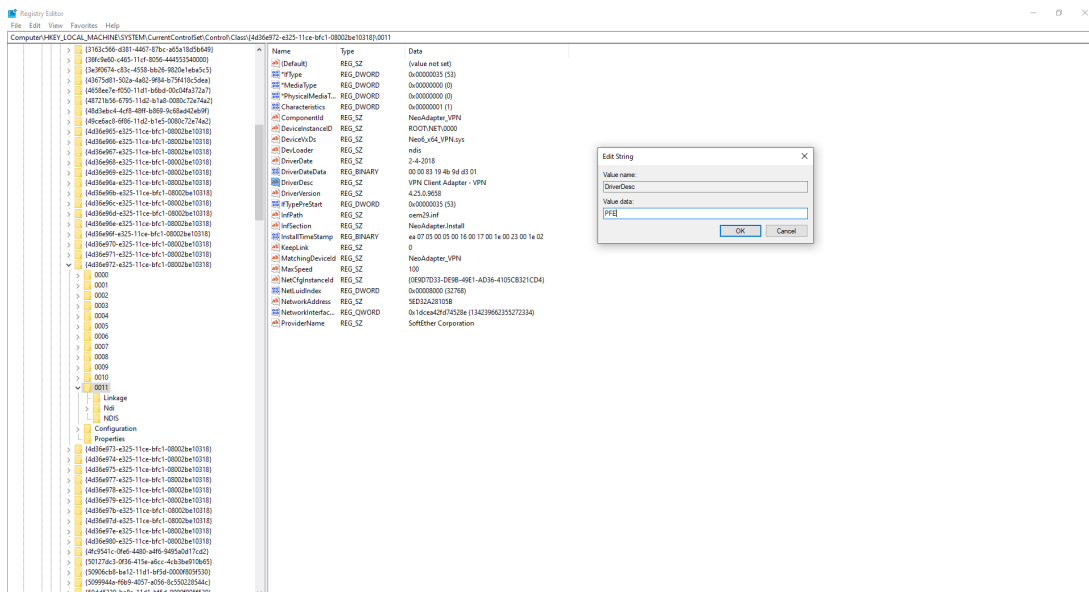


Figure 3.9: Registry changes [33].

However, this change prevents SoftEther from displaying the adapter, because SoftEther has a filter that only shows adapters whose names start with "VPN." Therefore, once the

adapter appears in TIA Portal, the driver description should be changed back to its original name. This way, the adapter remains visible in both SoftEther and TIA Portal.

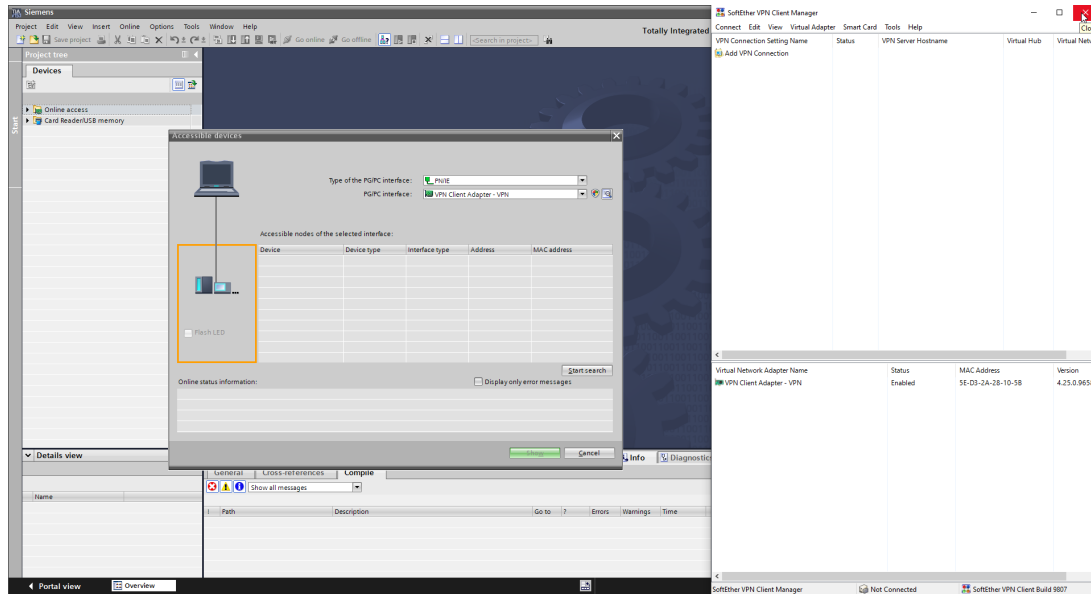


Figure 3.10: Network adapter appearance [35].

3.6 Bridge Configuration

Now, to make the PLC appear in the accessible devices list, the Ethernet port of the Raspberry Pi (eth0) must be bridged to the virtual hub (HUB1) using the **BridgeCreate** command in the VPN server. Once done, verify the bridge status using **BridgeList**.

```
VPN Server>Hub HUB1
Hub command - Select Virtual Hub to Manage
The Virtual Hub "HUB1" has been selected.
The command completed successfully.

VPN Server/HUB1>BridgeCreate
BridgeCreate command - Create Local Bridge Connection
Virtual Hub Name to Create Bridge: HUB1
Bridge Destination Device Name: eth0

VPN Server/HUB1>BridgeList
BridgeList command - Get List of Local Bridge Connection
Number|Virtual Hub Name|Network Adapter or Tap Device Name|Status
-----
1      |HUB1      |eth0      |Operating
The command completed successfully.
```

Figure 3.11: BridgeCreate and BridgeList commands [34].

3.7 Routing Missing or Conflict

After connecting the Raspberry Pi to the SCALANCE industrial switch, no DHCP server was available because the connection was direct. Normally, a DHCP server automatically assigns IPv4 addresses to devices on the network. Since no DHCP server was present in

this setup, the Raspberry Pi could not obtain an IP address automatically. Therefore, a static IPv4 address had to be assigned manually to eth0, ensuring that this address was on the same subnet as the PLC's IP address. To achieve this, it was necessary to determine which network management service was controlling the network interfaces on the Raspberry Pi. This was verified using the following commands:

```
systemctl is-active NetworkManager
# Output: active

systemctl is-active dhcpcd
# Output: inactive
```

The results indicate that " NetworkManager " was the active service responsible for managing network configurations, while "dhcpcd" was not running. Therefore, all network settings had to be configured using NetworkManager tools.

Another issue we had to consider was that the Wi-Fi and Ethernet interfaces must not end up on the same IP subnet. In our case, the Wi-Fi router's network was 192.168.0.0/24 and the eth0 also needed to be configured with 192.168.0.0/24 to match the PLC's address range.

This configuration caused a routing conflict because the operating system could not consistently determine which network interface should be used to reach devices within that subnet. Consequently, communication between the dashboard application running on the Raspberry Pi and the PLC became unstable.

3.7.1 Solution: Manual Ethernet Configuration and Route Prioritization

To resolve the problem, a static IP address was assigned to the Ethernet interface using NetworkManager. In addition, a routing metric was configured to control route priority and avoid conflicts with the Wi-Fi interface.

The following command was used:

```
sudo nmcli con mod \
"$(nmcli -t -f NAME,DEVICE con show | \
awk -F: ' $2=="eth0"{print $1}')" \
ipv4.method manual \
ipv4.addresses 192.168.1.200/24 \
ipv4.route-metric 500
```

3.7.2 Explanation of the Configuration Parameters

The command performs the following actions:

- `ipv4.method manual`: Disables automatic IP address assignment and enables manual configuration.
- `ipv4.addresses 192.168.1.200/24`: Assigns the static IPv4 address 192.168.1.200 with a subnet mask of 255.255.255.0.
- `ipv4.route-metric 500`: Assigns a routing metric value to the Ethernet interface. The routing metric determines the priority of network routes.

3.7.3 Result

After applying the static network configuration and adjusting the routing metric, the Ethernet interface was able to communicate reliably with the PLC. The routing conflicts between the Wi-Fi and Ethernet interfaces were eliminated.

3.8 Unit Testing

This Python script was developed to validate the reliability of the Snap7 library before its integration into a Flask-based industrial web application. It establishes Ethernet communication with a Siemens S7-1214C PLC and tests essential functions, including PLC connectivity, data block read/write operations, motor start/stop control, speed adjustment, and real-time monitoring of operational feedback. The script also provides diagnostic logging and exception handling to verify stable and accurate communication between the Raspberry Pi and the PLC, ensuring the safe use of Snap7 for remote industrial control and supervision.

Testing script :

In this section, the Snap7 library will be tested before being integrated into the Flask application. Only the most important functions are presented for the purpose of this chapter summary, while the complete test scripts are provided in [the appendix section](#).

Configuration Instructions

- `snap7.client.Client()` : Creates a Snap7 client instance used to establish communication with an S7 PLC.

- `connect(IP, rack, slot)` : Connects the client to the PLC using its IP address, rack number, and slot number.
- `get_cpu_info()` : Retrieves general PLC information such as model, name, and serial number.
- `get_cpu_state()` : Returns the current PLC operating state (RUN or STOP).
- `plc_hot_start()` : Switches the PLC from STOP mode to RUN mode.
- `plc_stop()` : Switches the PLC from RUN mode to STOP mode.

Memory Access Instructions

- `eb_read(start, size)` : Reads bytes from the PLC input memory area.
- `eb_write(start, size, data)` : Writes bytes into the PLC input memory area.
- `ab_read(start, size)` : Reads bytes from the PLC output memory area.
- `ab_write(start, size, data)` : Writes bytes into the PLC output memory area.
- `mb_read(start, size)` : Reads bytes from the PLC marker memory area.
- `mb_write(start, size, data)` : Writes bytes into the PLC marker memory area.
- `db_read(db_number, start, size)` : Reads bytes from a specified Data Block.
- `db_write(db_number, start, size, data)` : Writes bytes into a specified Data Block.

Utility Instructions

- `snap7.util.get_bool()` : Extracts a Boolean value from a byte array.
- `snap7.util.set_bool()` : Inserts a Boolean value into a byte array.
- `snap7.util.get_int()` : Extracts an integer value from a byte array.
- `snap7.util.set_int()` : Inserts an integer value into a byte array.
- `snap7.util.get_real()` : Extracts a REAL floating-point value from a byte array.
- `snap7.util.set_real()` : Inserts a REAL floating-point value into a byte array.

3.8.1 Description of the Controlled System

This section presents the system chosen for the validation of the Snap7 library and for the monitoring application described in the following section. The system is a speed control platform for an asynchronous electric motor, implemented using a PLC and a Variable Speed Drive (VSD). The platform was developed within the framework of a graduation project entitled :

"Identification and Speed Control of an Asynchronous Motor Using a Variable Speed Drive Connected to a Siemens S7-1200 PLC"[[14]]

Original title :

"Identification et commande de la vitesse d'un moteur asynchrone via un variateur de vitesse connecté avec un API S7 1200 de marque SIEMENS"

carried out by the students Mr. BOUREGUEBA Oussama and Mlle. HADJ BOUZIANE Chaima from the Electrical Control specialty and Supervised by Professor MELIANI Sidi Mohammed.

Motor Speed Control System Operator Reference Guide

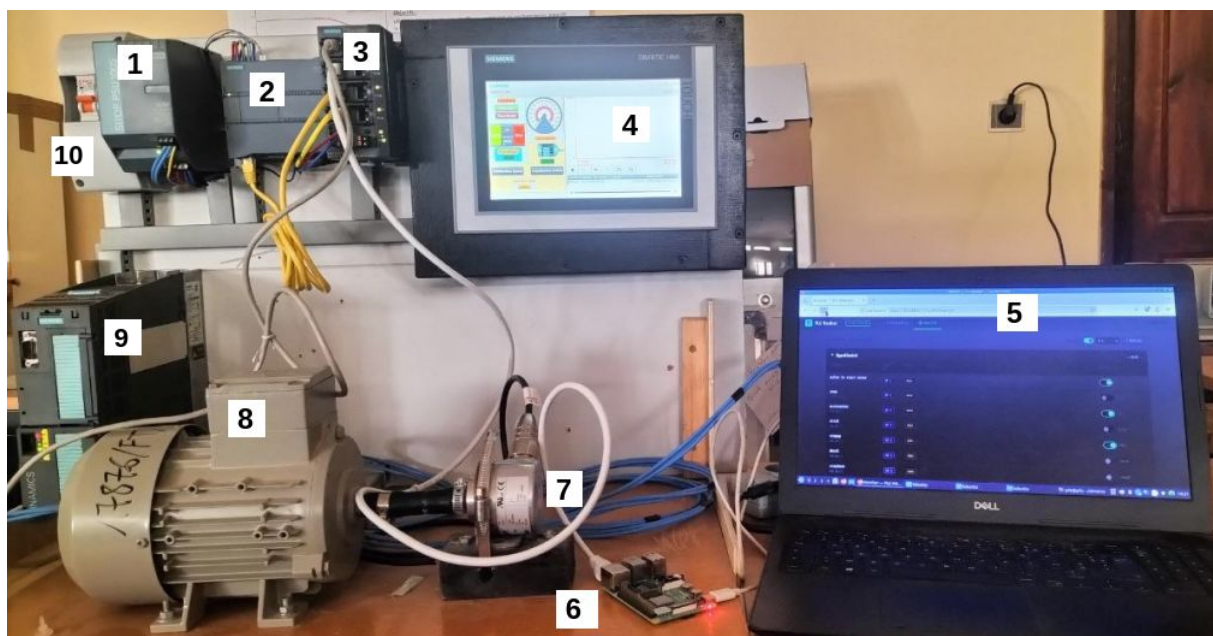


Figure 3.12: Motor Speed Control System [39].

The motor is controlled through a Siemens S7-1214C (PLC). From the operator's perspective, the PLC acts as a black box: to control the motor, one simply reads from or

writes to specific memory addresses inside the PLC. No knowledge of the internal program is required.

System Components

1. SITOP 24 VDC power supply
2. Siemens S7-1214C PLC
3. Siemens Scalance X208 switch "Ethernet Network"
4. Siemens HMI Comfort Panel
5. Laptop
6. Raspberry Pi 4
7. Encoder for speed feedback
8. Asynchronous motor
9. Variable Speed Drive (VSD)
10. Differential Circuit Breaker
11. 4G_LTE Router

The system allows the operator to:

- Start the motor
- Stop the motor
- Increase motor speed
- Decrease motor speed
- Monitor real motor speed
- Supervise motor status and feedback signals

HMI screen view :

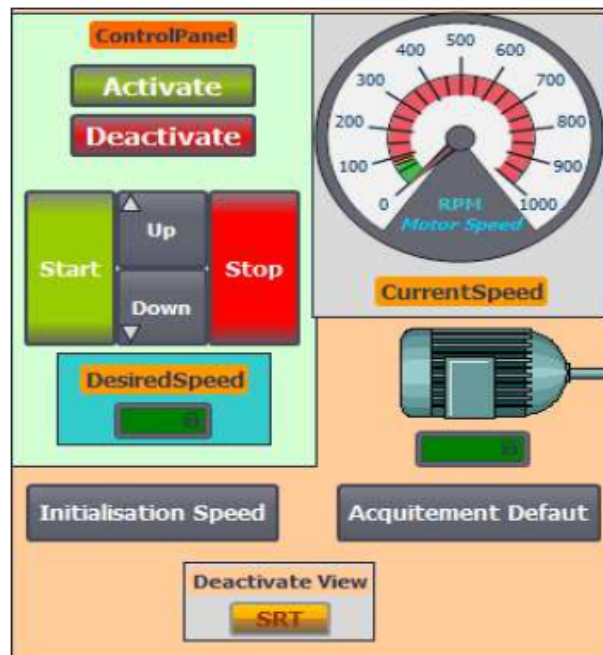


Figure 3.13: HMI used to control motor speed [14].

Addresses type:

There are two types of addresses used in this system:

- **Boolean** : hold either 1 (TRUE) or 0 (FALSE), used as on/off commands.
- **Integer / Real** : hold numeric values, used to read speed measurements.

Pulse rule: Every write command in this system works as a pulse write 1, then immediately write 0. The PLC registers the command on the rising edge and latches it internally. Holding the value at 1 has no additional effect.

Starting the Motor

Step	Action	Address
1	Pulse 1→0	DB1.DBX0.3 — arms the drive
2	Pulse 1→0	DB1.DBX0.0 — starts the motor
3	Read value	DB1.DBX0.4 — returns 1 if motor is running

Table 3.1: Motor Start Procedure [37].

Stopping the Motor

Step	Action	Address
1	Pulse 1→0	DB1.DBX0.1 — stops the motor
2	Pulse 1→0	DB1.DBX0.5 — disarms the drive
3	Read value	DB1.DBX0.4 — returns 0 when motor has stopped

Table 3.2: Motor Stop Procedure [37].

Increasing Motor Speed

Each pulse increases the speed by **100 rpm**. The motor must be running. Maximum speed is **1 000 rpm** (10 pulses from zero).

Step	Action	Address
1	Pulse 1→0	DB1.DBX2.1 — speed increases by 100 rpm
2	Repeat as needed	One pulse = +100 rpm
3	Read value	DB1.DBW10 — confirms current target speed in rpm

Table 3.3: Motor Speed Increase Procedure [37].

Decreasing Speed

Each pulse decreases the speed by 100 rpm. Minimum speed is 0 rpm.

Step	Action	Address
1	Pulse 1→0	DB1.DBX2.3 — speed decreases by 100 rpm
2	Repeat as needed	One pulse = −100 rpm
3	Read value	DB1.DBW10 — confirms current target speed in rpm

Table 3.4: Motor Speed Decrease Procedure [37].

Resetting Speed to Zero

Step	Action	Address
1	Pulse 1→0	DB1.DBX2.0 — resets speed setpoint to 0 rpm instantly

Table 3.5: Motor Speed Reset Procedure [37].

Acknowledging a Fault

Step	Action	Address
1	Pulse 1→0	DB1.DBX0.6 — clears the active drive fault

Table 3.6: Drive Fault Acknowledgment Procedure [37].

Speed Limits

- Minimum speed 0 rpm
- Maximum speed 1000 rpm
- Speed step per pulse 100 rpm
- Number of steps 10

Address	Name	Type	Access	Description
DB1.DBX0.0	Start	Bool	WRITE	Pulse 1→0 to start the motor
DB1.DBX0.1	Stop	Bool	WRITE	Pulse 1→0 to stop the motor
DB1.DBX0.3	Activate	Bool	WRITE	Pulse 1→0 to arm the drive before starting
DB1.DBX0.4	Activation	Bool	READ	1 = motor running, 0 = motor stopped
DB1.DBX0.5	Deactivate	Bool	WRITE	Pulse 1→0 to disarm the drive after stopping
DB1.DBX0.6	Acquiter	Bool	WRITE	Pulse 1→0 to acknowledge a drive fault
DB1.DBX2.0	Init_Speed	Bool	WRITE	Pulse 1→0 to reset speed setpoint to 0 rpm
DB1.DBX2.1	Accelation	Bool	WRITE	Each pulse adds +100 rpm to the speed setpoint
DB1.DBX2.3	Deceleration	Bool	WRITE	Each pulse subtracts −100 rpm from the speed setpoint
DB1.DBW10	Desired_Speed	Integer	READ	Current speed setpoint in rpm (range: 0–1 000 rpm)
DB1.DBW22	Scale_CU_Meas_Speed	Integer	READ	Actual speed feedback from the drive (scaled value)
DB1.DBD38	N(rpm)	Real	READ	Measured shaft speed from the incremental encoder in rpm (range: 0–1 100 rpm)

Table 3.7: Complete Address Reference for Motor Speed Control [37].

3.9 Experimental Results

This section presents the terminal output of each executed test, followed by a corresponding HMI screenshot and a discussion of the obtained results.

About the Scale_CU_Meas_Speed and N(rpm) : During the test, DB1.DBD38 (N_rpm) returned 0.00 at all speed setpoints due to the absence of encoder signal on %ID1000. However, DB1.DBW22 (Scale_CU_Meas_Speed) returned non-zero values at each setpoint, as it is derived from the drive's own internal feedback word %IW258, independently of the encoder. The PLC normalises %IW258 over the range 0–16 384 and scales it to 0–1 000 rpm, which explains why the HMI gauge displayed a speed value lower than the commanded setpoint. The discrepancy between the two measurements confirms that the encoder is non-functional at the hardware level, while the drive itself is operating and reporting its internal state correctly.

Test 1: PLC Connection Verification

Terminal output :

```
pfe@pfe:~/elmarsa $ python3 motor_test.py
=====

MOTOR SPEED TEST   Snap7 S7-1214C
PLC: 192.168.0.2   Rack: 0   Slot: 1   DB: 1
=====

STEP 1: Connect to PLC

PASS   Connect PLC       Connected to 192.168.0.2 R0/S1, DB1
accessible
```

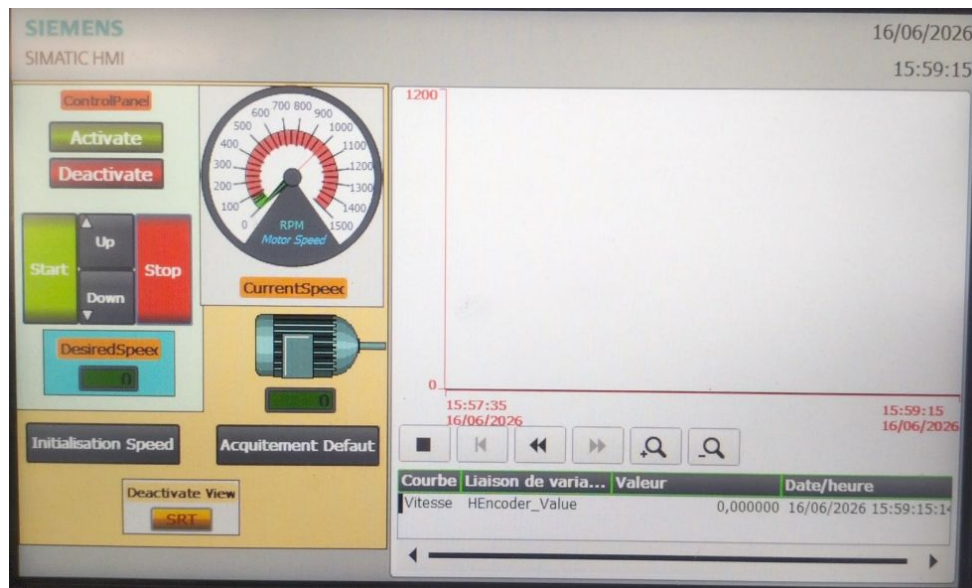


Figure 3.14: HMI interface during Test 1 [39].

The Snap7 library successfully established a TCP/IP connection to the S7-1214C at 192.168.0.2. DB1 was accessible, confirming that the library is correctly installed and PUT/GET access is enabled on the PLC.

Test 2: Reading PLC Variables

Terminal output

STEP 2: Start Motor (5s runtime)

Resetting PLC state (Stop + Deactivate)...

Pulsing Activate bit (DBX0.3)...

Pulsing Start bit (DBX0.0)...

Activation: True Desired_Speed: 0 rpm QW258: 0

Monitoring motor for 5s...

t= 0.0s	desired= 0 rpm	N(rpm)= 0.00	activa	t=
0.5s	desired= 0 rpm	N(rpm)= 0.00	activa	t= 1.1s
	desired= 0 rpm	N(rpm)= 0.00	activa	t= 1.6s
desired= 0 rpm	N(rpm)= 0.00	activa	t= 2.1s	
desired= 0 rpm	N(rpm)= 0.00	activa	t= 2.6s	
desired= 0 rpm	N(rpm)= 0.00	activa	t= 3.2s	
desired= 0 rpm	N(rpm)= 0.00	activa	t= 3.7s	
desired= 0 rpm	N(rpm)= 0.00	activa	t= 4.2s	

```

desired=    0 rpm | N(rpm)=    0.00 | activa    t=  4.7s |
desired=    0 rpm | N(rpm)=    0.00 | activation=True
PASS  Start Motor    Activation latched, avg N(rpm)=0.0, QW258
=0

```

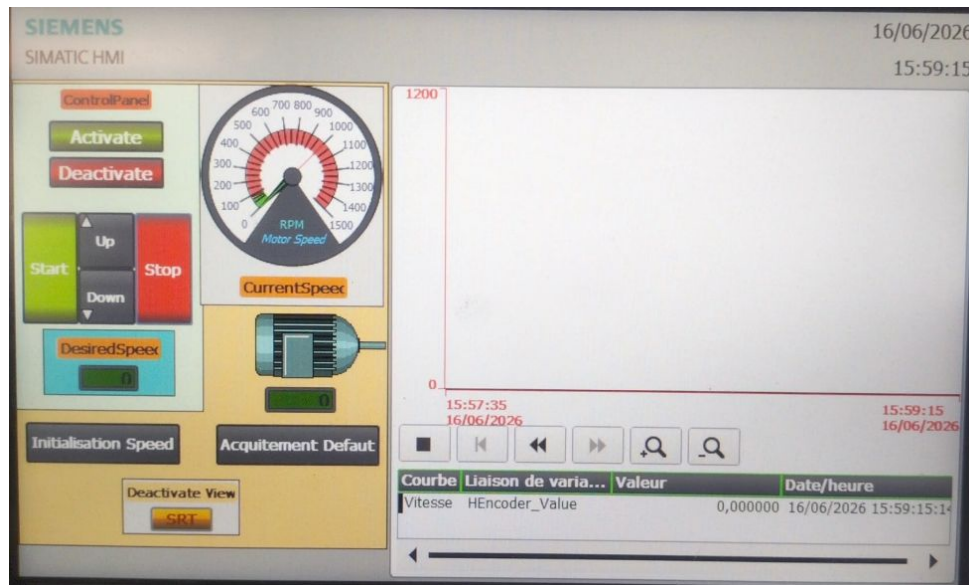


Figure 3.15: HMI interface during Test 2 [39].

After resetting both SR latches and pulsing the Activate then Start bits, the Activation latch latched to TRUE within the expected scan cycle. The Snap7 write and read operations on boolean addresses executed without errors.

Test 3: Writing PLC Variables

Terminal output :

STEP 3: Change Speed (Accel/Decel pulses)

Config: StepValue=100, Max=1000, Min=0

Target: 300 rpm Current: 0 Delta: 300 Pulses: 3 Accel ()

Pulse 3/3 Desired_Speed= 300 rpm

PASS Change Speed 300 rpm Desired_Speed=300 (clic=3)

Monitoring for 5s at 300 rpm...

t= 4.7s | desired= 300 | N(rpm)= 0.00

```

Target: 700 rpm   Current: 300   Delta: 400   Pulses: 4   Accel ()
Pulse 4/4   Desired_Speed= 700 rpm
PASS   Change Speed   700 rpm   Desired_Speed=700 (clic=7)
Monitoring for 5s at 700 rpm...
t= 4.7s | desired= 700 | N(rpm)= 0.00

Target: 1000 rpm   Current: 700   Delta: 300   Pulses: 3   Accel ()
Pulse 3/3   Desired_Speed=1000 rpm
PASS   Change Speed   1000 rpm   Desired_Speed=1000 (clic=10)
Monitoring for 5s at 1000 rpm...
t= 4.7s | desired=1000 | N(rpm)= 0.00

```

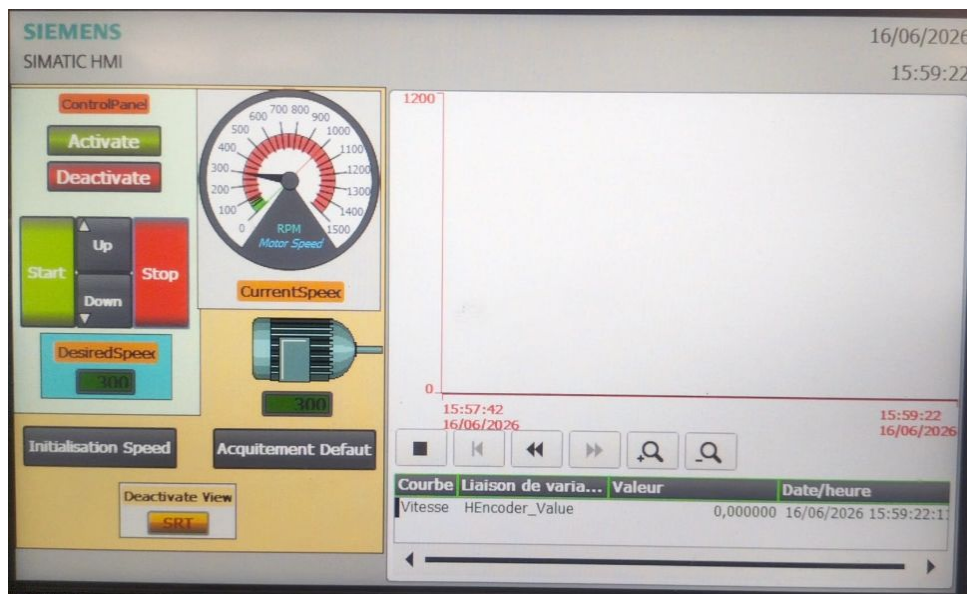


Figure 3.16: HMI interface during Test 3_300 rpm [39].

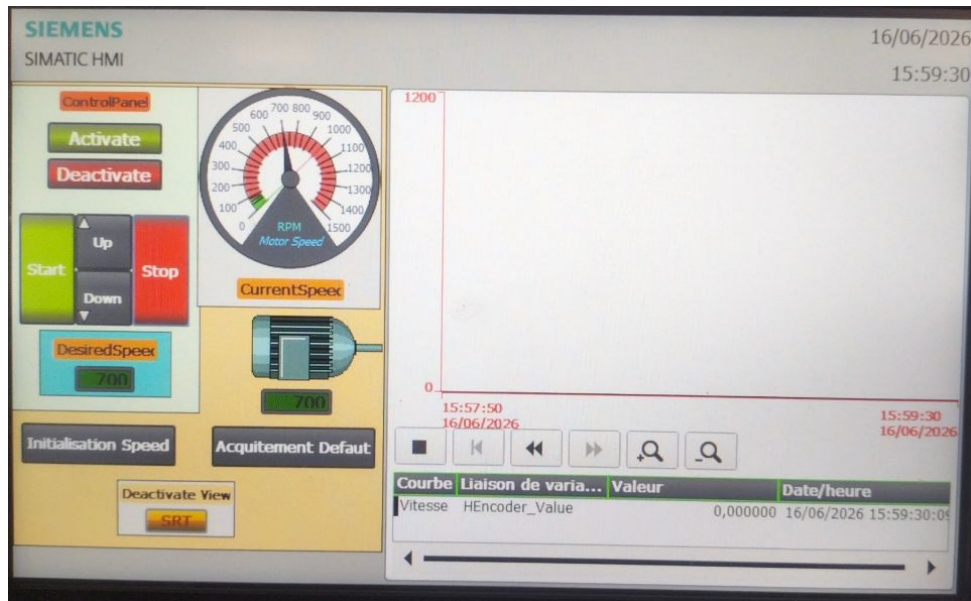


Figure 3.17: HMI interface during Test 3_700 rpm [39].

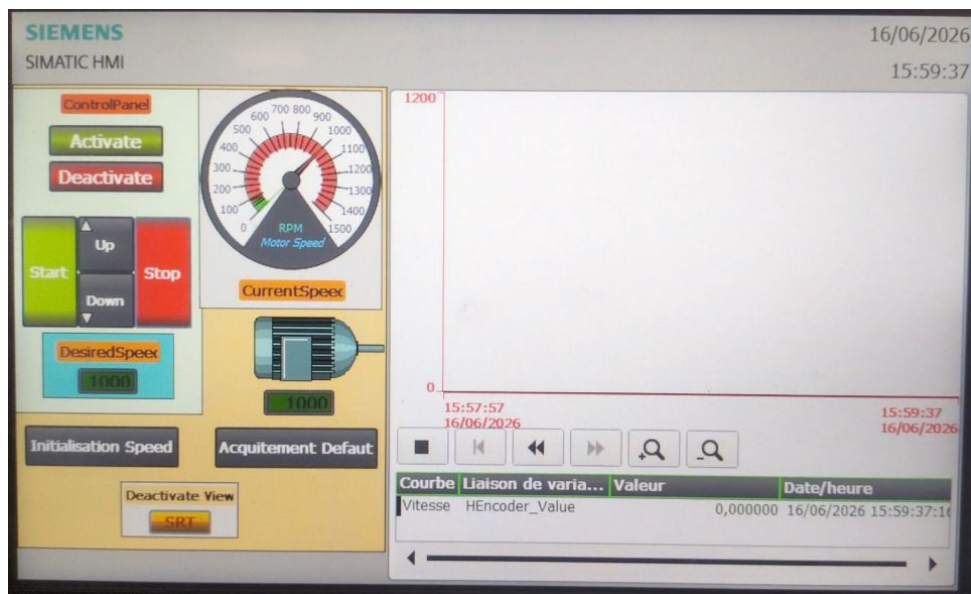


Figure 3.18: HMI interface during Test 3_1000 rpm [39].

Speed Change (300 → 700 → 1000 rpm), The counter-based speed control was validated across three consecutive setpoints. Each Accel pulse correctly incremented the ClicNumber by one unit, and the PLC recomputed Desired_Speed accurately as $\text{ClicNumber} \times 100 \text{ rpm}$, confirming reliable read/write operations on integer addresses.

Test 4: Speed Monitoring

Terminal output :

STEP 4: Display Speed (4s)

t(s)	Desired	N(rpm)	Scale	CU_Meas	Act
0.0	1000	0.00	1000	10923	True
0.6	1000	0.00	1000	10923	True
1.1	1000	0.00	1000	10923	True
1.7	1000	0.00	1000	10923	True
2.3	1000	0.00	1000	10923	True
2.8	1000	0.00	1000	10923	True
3.4	1000	0.00	1000	10923	True
3.9	1000	0.00	1000	10923	True
PASS	Display Speed	avg N(rpm)=0.00 over 8 samples			

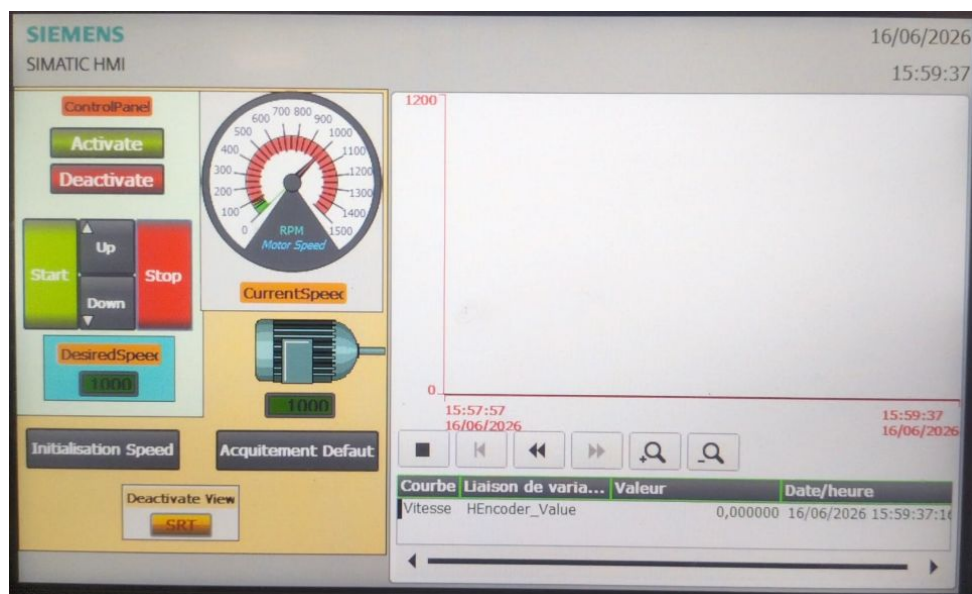


Figure 3.19: HMI interface during Test 4 [39].

During the 4-second display window, the following addresses were sampled every 0.5 seconds across 8 consecutive reads:

- DB1.DBW10 (Desired_Speed) via `util.get_int` — stable at 1 000 rpm

- DB1.DBD38 (N_rpm) via `util.get_real` — 0.00 across all samples
- DB1.DBW22 (Scale_CU_Meas_Speed) via `util.get_int` — constant at 667
- %IW258 (CU_Meas_Speed) via `read_area(Area.PE, 0, 258, 2)` — constant at 10923
- DB1.DBX0.4 (Activation) via `util.get_bool` — TRUE throughout

All 8 samples were consistent with zero variance, confirming reliable Snap7 polling stability across boolean, integer, and real data types.

Test 5: Stop Motor

Terminal output :

STEP 5: Stop Motor

Pulsing Stop bit (DBX0.1)...

Pulsing Deactivate bit (DBX0.5)...

PASS Stop Motor Activation cleared, QW258=2411, N(rpm)=0.0

PLC disconnected.

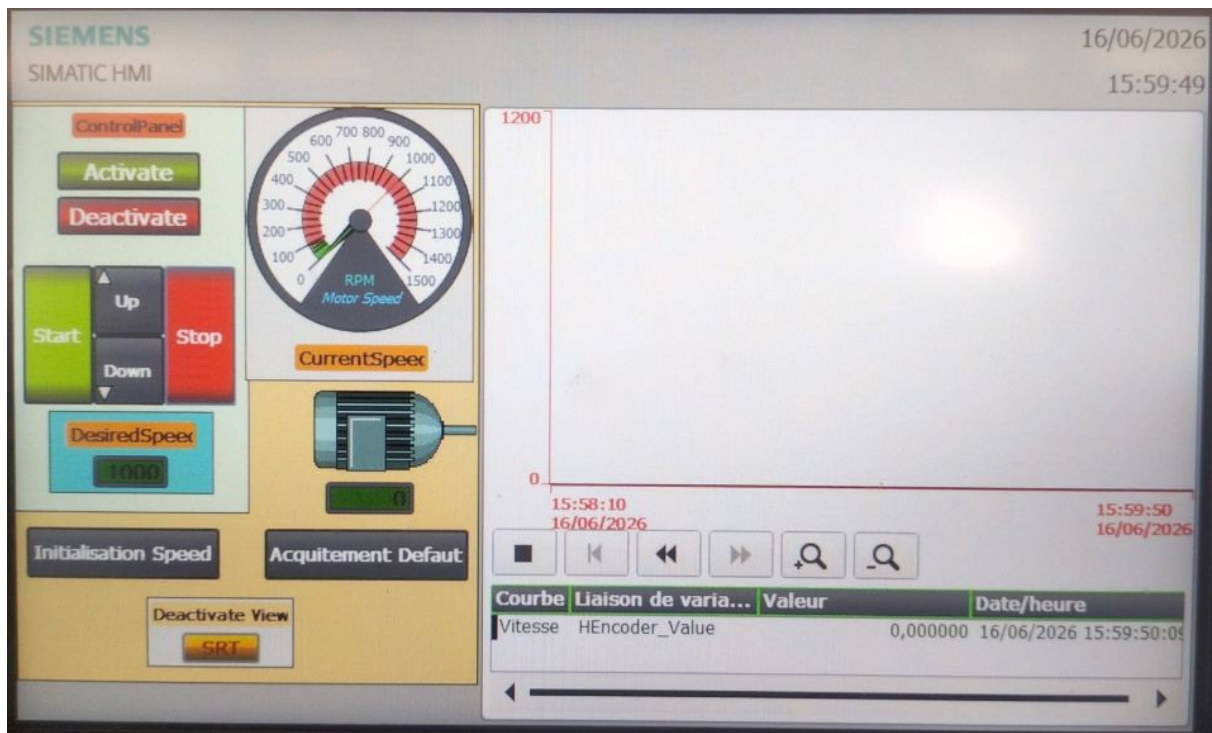


Figure 3.20: HMI interface during Test 5 [39].

Test Summary

Terminal output :

TEST SUMMARY 7/7 passed

Connect PLC

Connected to 192.168.0.2 R0/S1, DB1 accessible

Start Motor

Activation latched, avg N(rpm)=0.0, QW258=0

Change Speed 300 rpm

Desired_Speed=300 (clic=3)

Change Speed 700 rpm

Desired_Speed=700 (clic=7)

Change Speed 1000 rpm

Desired_Speed=1000 (clic=10)

Display Speed

avg N(rpm)=0.00 over 8 samples

Stop Motor

Activation cleared, QW258=2411, N(rpm)=0.0

ALL PASSED

Conclusion

All seven test cases passed successfully. The Snap7 functions `db_read`, `db_write`, `read_area`, and the utility functions `get_bool`, `set_bool`, `get_int`, and `get_real` all performed reliably across Boolean, integer, and real data types. Sequential read-modify-write operations on Boolean addresses correctly preserved adjacent bits, and multi-address polling over 8 consecutive samples produced consistent results with no communication errors.

3.10 PLC Monitor Application



Figure 3.21: **PLC Monitor** Logo [40]

Following the successful evaluation of the Snap7 library within the developed motor speed control system, a Flask-based industrial monitoring application was developed under the name **PLC Monitor**. The application operates over the industrial network and provides a modern web-based interface for real-time PLC monitoring and control directly from any web browser. In addition to local network accessibility, the application also supports secure remote access through **VPN connectivity**, ensuring flexible, reliable, and efficient industrial system management.

3.10.1 Features

- Web-based UI : No special software required, works in any browser
- Snap7 integration : Supports S7-300/400/1200/1500 PLCs
- Multi-PLC : Add as many PLCs as needed
- Live monitoring : Auto-polling with configurable intervals (1s10s)
- Read/Write : All standard data types (BOOL, BYTE, INT, DINT, WORD, DWORD, REAL)
- Thread-safe : Per-PLC locking for concurrent reads and writes
- Persistent config JSON storage, survives restarts
- Improved operational stability
- Suitable for continuous operation
- Multiple users can access simultaneously
- The application starts automatically when Raspberry Pi powers on (Auto-start on boot).
- Accessed directly through the Raspberry Pi IP

3.10.2 System Requirements

- OS: Linux (Raspberry Pi, Ubuntu), macOS, or Windows (WSL2)
- Python: 3.7 or later
- Network: The PLCs and the Raspberry Pi must be connected to the same local network to ensure proper communication and data exchange.

3.10.3 Access to PLC Monitor Application :

Once the Raspberry Pi is powered on, the PLC Monitor application becomes available to all devices connected to the same network. Access is achieved through a web browser by entering the Raspberry Pi IP address using the HTTP protocol.

(e.g., `http://<RaspberryPi_IP>`).

Page 1: Setup

- Upon the first launch of the application, the user is prompted to enter a name for the industrial process.
- Once confirmed, the application automatically creates and saves a dedicated database for the process and its associated variables.
- The configuration is stored in `data.json` file in the application folder (Raspberry).
- This page is shown only once during the initial setup and does not appear again unless the configuration is reset.

Process Name:

- Assigning a Name to the Process

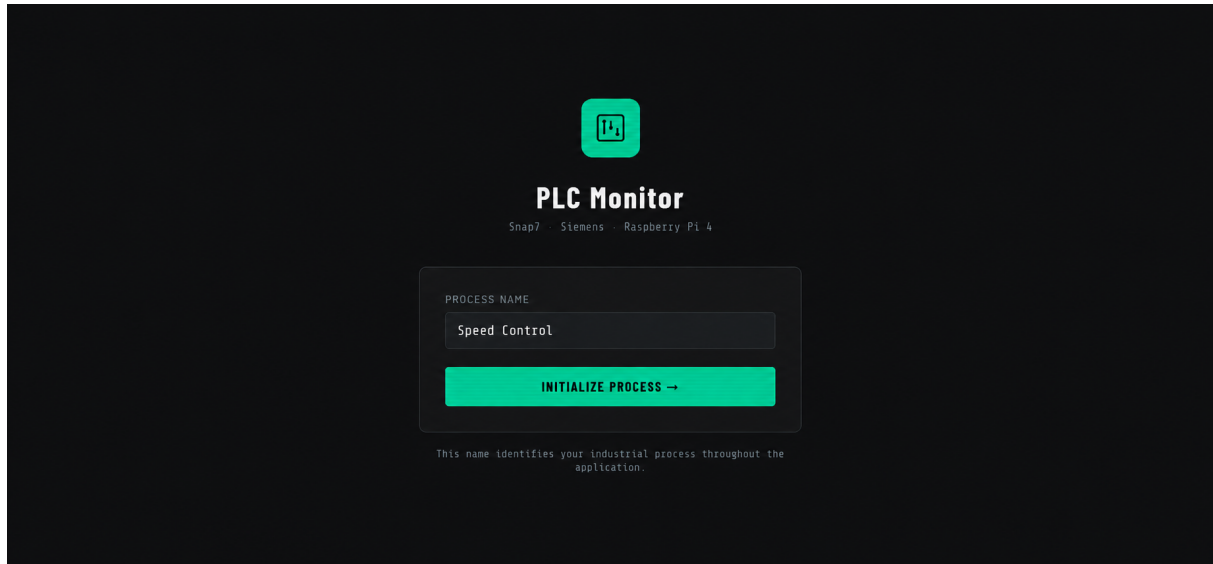


Figure 3.22: The process is referred to as "Speed Control" [41].

Page 2: Configuration

Add PLC

- Enter PLC : Name, IP, Rack, Slot

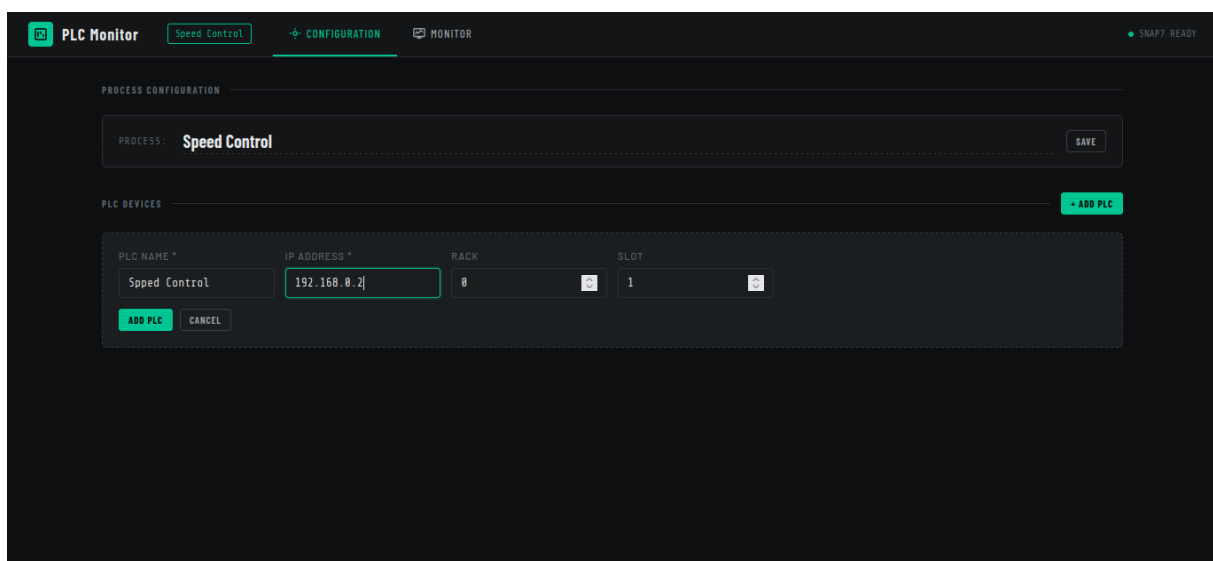


Figure 3.23: Linking S7-1200 PLC to PLC MONITOR application [41].

- Test Connection with (green/red indicator)

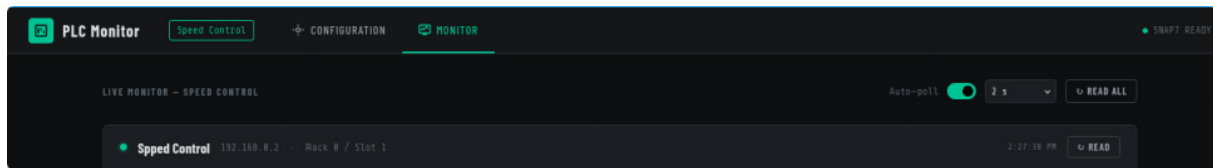


Figure 3.24: A flashing green indicator denotes successful connectivity [41].

The PLC Monitor application displays two connectivity indicators. The SNAP7 READY indicator in the navigation bar confirms that the Snap7 library is correctly installed and available on the host machine. The per-PLC status dot on the monitor page reflects the result of the most recent read operation – green means the last communication with the PLC succeeded, red means it failed.

Add Variables

- Clicking on the (+ ADD VARIABLE) button
- Fill in the Variable Name field.

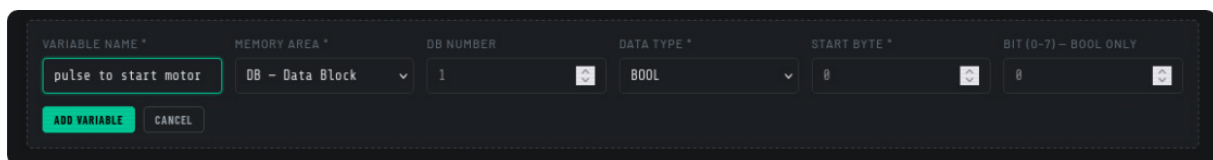


Figure 3.25: Variable Name Field [41].

- Select the Memory Area.

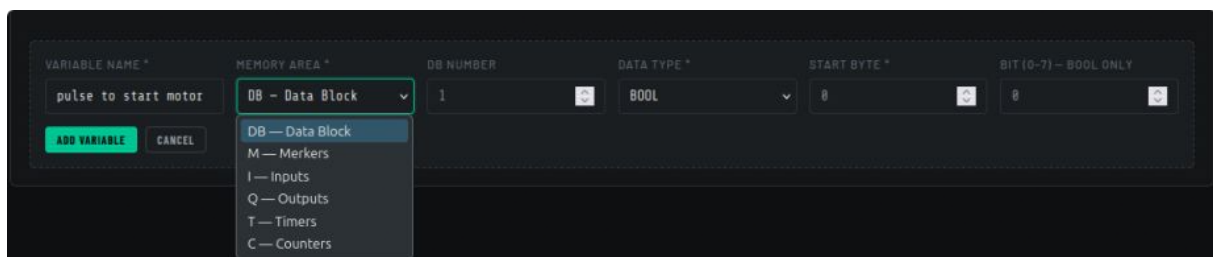
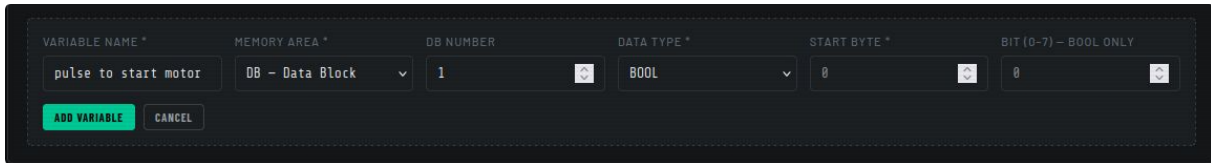


Figure 3.26: Memory Area List [41].

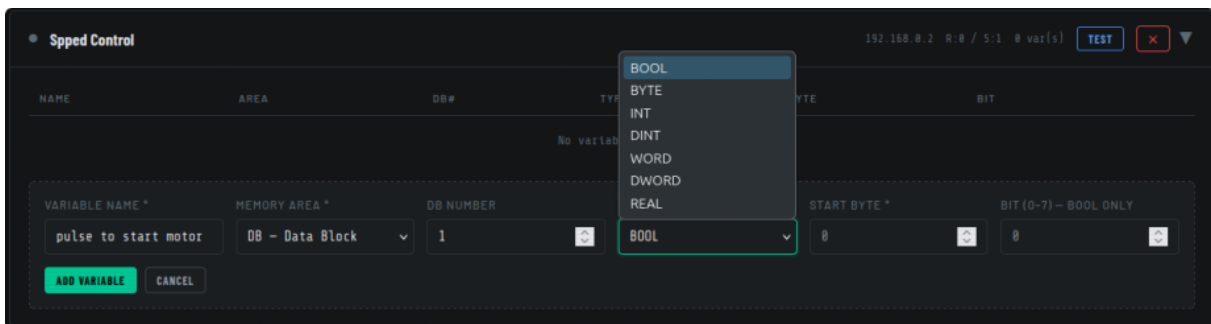
- Enter Data Block number. This field is enabled only when the selected memory area is set to Data Block (DB).



The screenshot shows a configuration form with the following fields: VARIABLE NAME * (pulse to start motor), MEMORY AREA * (DB - Data Block), DB NUMBER (1), DATA TYPE * (BOOL), START BYTE * (0), and BIT (0-7) - BOOL ONLY (0). The DB NUMBER field is highlighted with a green box. Below the fields are 'ADD VARIABLE' and 'CANCEL' buttons.

Figure 3.27: Data Block Number Field [41].

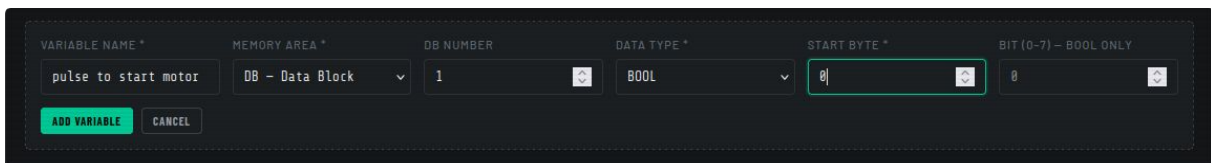
- Select the Data Type



The screenshot shows the same configuration form as Figure 3.27, but with the DATA TYPE * dropdown menu open. The menu lists the following options: BOOL, BYTE, INT, DINT, WORD, DWORD, and REAL. The 'BOOL' option is highlighted with a green box. The background shows a 'Speed Control' application window with a table of variables and a status bar at the top right displaying '192.168.0.2 R:0 / S:1 0 var(s) TEST'.

Figure 3.28: Data Type Field [41].

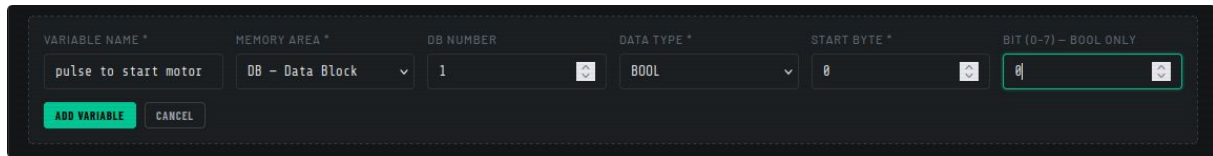
- Enter start byte



The screenshot shows the same configuration form as Figure 3.27, but with the START BYTE * field highlighted with a green box. The field contains the value '0'.

Figure 3.29: Byte Number Field [41].

- Enter bit number. This field is enabled only when the selected data type is set to **Bool**.



The image shows a configuration dialog box with a dark background and light text. It contains several input fields and buttons. The fields are: 'VARIABLE NAME *' with the text 'pulse to start motor'; 'MEMORY AREA *' with a dropdown menu showing 'DB - Data Block'; 'DB NUMBER' with a text box containing '1' and a small up/down arrow icon; 'DATA TYPE *' with a dropdown menu showing 'BOOL'; 'START BYTE *' with a text box containing '0' and a small up/down arrow icon; and 'BIT (0-7) - BOOL ONLY' with a text box containing '0' and a small up/down arrow icon. At the bottom left, there are two buttons: 'ADD VARIABLE' (highlighted in green) and 'CANCEL'.

Figure 3.30: Bit Number Field [41].

- Clicking the Add Variable button to instantly add the selected variable to the Tags list
- Easily browse all configured variables by clicking on the PLC name.
- The variable (Tags) list provides a complete overview of all configured variables, displaying their name, memory area, Data Block (DB) number (if applicable), data type, byte number, and bit number. From this interface, users edit the process by adding, or deleting variables.

PLC Monitor **Speed Control** [CONFIGURATION](#) [MONITOR](#) SNAP7 READY

PROCESS CONFIGURATION

PROCESS: **Speed Control** [SAVE](#)

PLC DEVICES [+ ADD PLC](#)

● **Speed Control** 192.168.0.2 R:0 / S:1 15 var(s) [TEST](#) [X](#) ▼

NAME	AREA	DB#	TYPE	BYTE	BIT	
pulse to start motor	DB	1	BOOL	0	0	X
stop	DB	1	BOOL	0	1	X
activation	DB	1	BOOL	0	4	X
accel	DB	1	BOOL	2	1	X
stopup	DB	1	BOOL	2	2	X
decel	DB	1	BOOL	2	3	X
stopdown	DB	1	BOOL	2	4	X
maxspeed	DB	1	WORD	4	-	X
minspeed	DB	1	WORD	6	-	X
desired speed	DB	1	WORD	10	-	X
stepvalue	DB	1	WORD	24	-	X
clicknumber	DB	1	WORD	26	-	X
encoder real	DB	1	WORD	34	-	X
N rpm	DB	1	WORD	38	-	X
scale desired speed	Q	-	WORD	258	-	X

[+ ADD VARIABLE](#)

Figure 3.31: Motor Speed Control System Tag List [41].

Page 3: Monitoring Page

The operator does not need to know anything about the internal PLC program or addresses; One screen is enough to start, stop, monitor and control the process.

- **Auto-poll:** Toggle ON to continuously read all PLCs.
- **Poll Interval:** Select the polling period from 1 s, 2 s, 5 s, or 10 s (default: 2 s).
- **Read All:** Manually triggers an immediate read of all configured variables.
- **Boolean Variables (BOOL):**
 - Click the toggle switch to write **TRUE** or **FALSE**.
 - Displays the current state as **ON** or **OFF**.
- **Numeric Variables (INT, DINT, REAL):**
 - The current value is displayed in real time.
 - Enter a new value in the input field.
 - Click **Write** to send the value to the PLC.
 - Successfully updated values are highlighted with a green flash.
- **Connection Indicators:**
 - **Green:** Connected and reading.
 - **Red:** Connection failed.
 - **Amber:** Reading in progress.

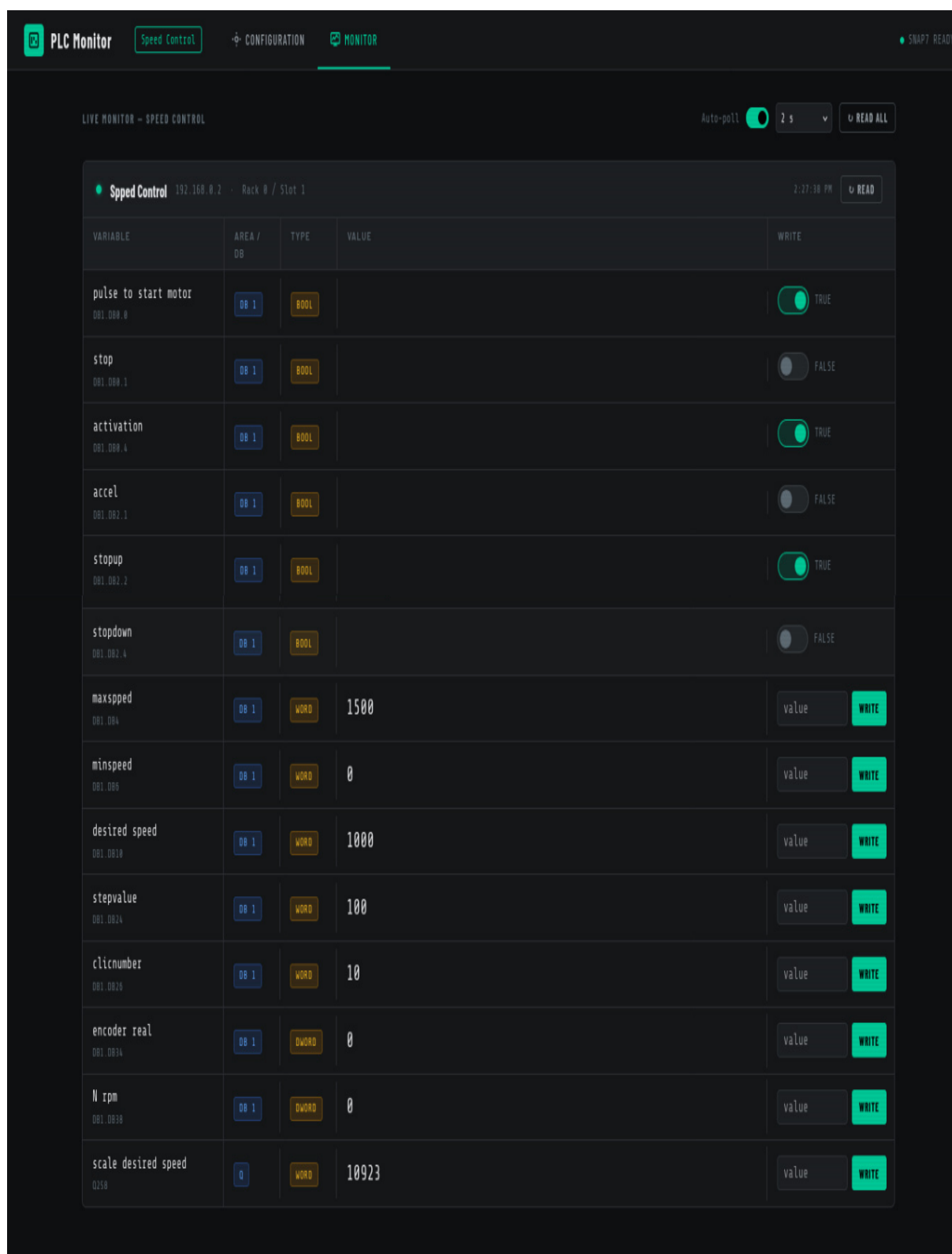


Figure 3.32: Process Control Panel [41].

Conclusion

This chapter presented the implementation and validation of the proposed remote industrial access solution. A secure Layer 2 VPN infrastructure was successfully deployed on a Raspberry Pi, enabling remote connectivity to the Siemens PLC. Communication reliability was verified through Snap7 functional tests, which confirmed stable read/write operations and real-time data exchange. Based on these results, the PLC Monitor application was developed to provide web-based monitoring and control of industrial processes. The experimental tests demonstrated that the system enables secure, reliable, and cost-effective remote supervision through a standard web browser.

General Conclusion

The motivation behind this research stemmed from two pressing industrial realities: the growing adoption of **remote work** and **Internet-based operations**, and the inherent difficulty of accessing industrial sites located in geographically challenging environments. By leveraging a standard Internet connection, this work demonstrates that engineers can achieve full **remote access** and control over industrial equipment regardless of their physical location.

The research followed a structured methodology, beginning with the establishment of a solid conceptual foundation spanning multiple disciplines. Industrial solutions from leading providers such as **EWON Network** were examined as benchmarks, alongside ongoing academic research, where it was noted that the European Union is making substantial investments in developing an **Industry 4.0**-ready workforce in response to growing global competition.

A strong emphasis was placed on computer networking, the VPN configuration process itself required extensive research and iterative experimentation. On the software side, prior experience in PLCs and microcontroller programming enabled effective use of Siemens' official documentation and informed design decisions, ensuring broad hardware compatibility.

This research culminated in the successful delivery of a complete remote control solution built around SoftEther VPN and the **PLC Monitor** application, enabling remote monitoring and control of Snap7-compatible PLCs over the Internet. The established Layer 2 **VPN tunnel** extends the local industrial network transparently, granting the remote engineer **full access** to all network devices as if physically present on site, including direct **TIA Portal** connectivity for advanced programming and diagnostics. By extension, the same infrastructure can support other engineering platforms such as Rockwell's Studio 5000 or Schneider's EcoStruxure, making the solution hardware-agnostic and broadly applicable. The outcome represents a practical, scalable, and **cost-effective** contribution to the field of industrial remote access.

.1 Appendix

.1.1 PLC MONITOR source code

The application source code will be available on GitHub under the organization name **PLC Monitor** [31]. The project will be distributed under the **Apache 2.0 License**, ensuring unrestricted use for commercial, educational, and personal purposes.

This work is intended to serve as a foundation for future Masters and doctoral research in the coming years. In addition, contributions from the community are strongly encouraged to support the continuous development and long-term sustainability of the project.

.1.2 Motor Speed Test Full script

```
#!/usr/bin/env python3
"""
Motor Speed Test  Snap7 S7-1214C
Target PLC: Siemens S7-1214C DC/DC/DC @ 192.168.0.2

KEY FEATURES:
    Correct pulse-based speed control (via Accel/Decel, not direct
        write)
    Proper Stop sequence (Stop + Deactivate)
    Thread-safe operation
    Colored output with full diagnostics

DB1 Address Map:
    DBX0.0 Start  pulse to start motor
    DBX0.1 Stop   pulse to stop motor
    DBX0.3 Activate pulse to arm drive
    DBX0.4 Activation  motor running feedback
    DBX0.5 Deactivate pulse to disarm drive
    DBX2.1 Accel  pulse to increase speed (+StepValue rpm)
    DBX2.2 StopUp pulse to hold speed after accel
    DBX2.3 Decel  pulse to decrease speed (-StepValue rpm)
    DBX2.4 StopDown pulse to hold speed after decel
    DBW4  MaxSpeed (1000 rpm)
    DBW6  MinSpeed (0 rpm)
    DBW10 Desired_Speed (R0 PLC-computed)
    DBW24 StepValue (100 rpm per pulse)
```

```

    DBW26 ClicNumber (R0 CTUD counter output)
    DBD34 Encoder_Real
    DBD38 N_rpm (measured speed)
    QW258 Scale_Desired_Speed (output to drive)
"""

import snap7, snap7.util as util, struct, time, sys, threading

# Configuration
PLC_IP      = "192.168.0.2"
RACK, SLOT  = 0, 1
DB_NUM      = 1
RUN_SECONDS = 5      # monitoring duration per speed step
DISPLAY_SECONDS = 4  # step 4 display duration
POLL_INTERVAL = 0.5
PULSE_HOLD   = 0.15
PULSE_GAP    = 0.1
SETTLE       = 0.3
SPEED_TARGETS = [300, 700, 1000]

# Colors (ANSI)
GREEN = "\033[92m"
RED   = "\033[91m"
YELLOW = "\033[93m"
CYAN  = "\033[96m"
RESET = "\033[0m"

results = []

# Low-level helpers

def rb(c, byte_, bit_):
    """Read boolean bit."""
    return util.get_bool(c.db_read(DB_NUM, byte_, 1), 0, bit_)

def wb(c, byte_, bit_, val):
    """Write boolean bit (read-modify-write)."""
    buf = c.db_read(DB_NUM, byte_, 1)
    util.set_bool(buf, 0, bit_, val)
    c.db_write(DB_NUM, byte_, buf)

```

```

def ri(c, byte_):
    """Read signed 16-bit integer."""
    return util.get_int(c.db_read(DB_NUM, byte_, 2), 0)

def rr(c, byte_):
    """Read 32-bit real (float)."""
    return util.get_real(c.db_read(DB_NUM, byte_, 4), 0)

def read_iw(c, addr):
    """Read input word (e.g. IW258)."""
    try:
        from snap7 import Area
        buf = c.read_area(Area.PE, 0, addr, 2)
    except Exception:
        buf = c.eb_read(addr, 2)
    return struct.unpack(">H", bytes(buf))[0]

def pulse(c, byte_, bit_, hold=PULSE_HOLD, gap=PULSE_GAP):
    """Assert a bit for <hold> seconds, then clear."""
    wb(c, byte_, bit_, True)
    time.sleep(hold)
    wb(c, byte_, bit_, False)
    time.sleep(gap)

# Test result tracking

def log(name, ok, detail=""):
    tag = f"{GREEN} PASS{RESET}" if ok else f"{RED} FAIL{RESET}"
    print(f" {tag} {name}" + (f" {detail}" if detail else ""))
    results.append({"name": name, "ok": ok, "detail": detail})

def summary():
    passed = sum(1 for r in results if r["ok"])
    total = len(results)
    print(f"\n{' '*70}")
    print(f" TEST SUMMARY {passed}/{total} passed")
    print(' '*70)
    for r in results:
        tag = f"{GREEN}{RESET}" if r["ok"] else f"{RED}{RESET}"
        print(f" {tag} {r['name']}")

```

```

        if r["detail"]:
            print(f"          {YELLOW}{r['detail']}{RESET}")
        color = GREEN if passed == total else RED
        status = "ALL PASSED " if passed == total else f"{total-
passed} FAILED"
        print(f"\n  {color}{status}{RESET}\n{' '*70}")

# Test steps

def step1_connect():
    print(f"\n{CYAN}{' '*70}{RESET}")
    print(f"{CYAN}STEP 1: Connect to PLC{RESET}")
    print(f"{CYAN}{' '*70}{RESET}")
    try:
        c = snap7.client.Client()
        c.connect(PLC_IP, RACK, SLOT)
        if not c.get_connected():
            log("Connect PLC", False, "get_connected() returned
False")
            return None
        c.db_read(DB_NUM, 0, 2)
        log("Connect PLC", True,
            f"Connected to {PLC_IP} R{RACK}/S{SLOT}, DB{DB_NUM}
accessible")
        return c
    except Exception as e:
        log("Connect PLC", False, str(e))
        return None

def step2_start(c):
    print(f"\n{CYAN}{' '*70}{RESET}")
    print(f"{CYAN}STEP 2: Start Motor ({RUN_SECONDS}s runtime){
RESET}")
    print(f"{CYAN}{' '*70}{RESET}")
    try:
        # Full reset clear both SR latches regardless of
previous state
        print("  Resetting PLC state (Stop + Deactivate)...")
        wb(c, 0, 1, True)      # Stop = 1
        time.sleep(0.2)

```

```

wb(c, 0, 1, False)    # Stop = 0
time.sleep(0.2)
wb(c, 0, 5, True)     # Deactivate = 1
time.sleep(0.2)
wb(c, 0, 5, False)    # Deactivate = 0
time.sleep(0.5)       # Allow PLC scan to fully clear both
latches

# Arm the drive first (Activate)
print("    Pulsing Activate bit (DBX0.3)...")
pulse(c, 0, 3, hold=0.3)
time.sleep(0.2)

# Start the motor
print("    Pulsing Start bit (DBX0.0)...")
pulse(c, 0, 0, hold=0.3)

# Wait for Activation latch
for _ in range(20):
    if rb(c, 0, 4):
        break
    time.sleep(0.1)

act      = rb(c, 0, 4)
qw       = read_iw(c, 258)
desired  = ri(c, 10)

if not act:
    log("Start Motor", False,
        "Activation bit did not latch after Start pulse")
    return False

print(f"    Activation: {act}    Desired_Speed: {desired} rpm
QW258: {qw}")
print(f"    Monitoring motor for {RUN_SECONDS}s...")

t0       = time.time()
samples  = []
while time.time() - t0 < RUN_SECONDS:
    elapsed = time.time() - t0
    rpm     = rr(c, 38)

```

```

        samples.append(rpm)
        print(f"      t={elapsed:5.1f}s | desired={ri(c,10):4d}
rpm | "
            f"N(rpm)={rpm:8.2f} | activation={act}", end="\
r")

        time.sleep(POLL_INTERVAL)
        print()

        avg_rpm = sum(samples) / len(samples) if samples else 0
        log("Start Motor", True,
            f"Activation latched, avg N(rpm)={avg_rpm:.1f}, QW258
={qw}")
        return True

    except Exception as e:
        log("Start Motor", False, str(e))
        return False

def step3_change_speed(c):
    print(f"\n{CYAN}{' '*70}{RESET}")
    print(f"{CYAN}STEP 3: Change Speed (Accel/Decel pulses){RESET
}")
    print(f"{CYAN}{' '*70}{RESET}")

    step_val = ri(c, 24)
    max_spd = ri(c, 4)
    min_spd = ri(c, 6)
    print(f"  Config: StepValue={step_val}, Max={max_spd}, Min={
min_spd}")

    if step_val <= 0:
        log("Change Speed", False, "StepValue=0  cannot compute
pulse count")
        return False

    all_ok = True
    for target in SPEED_TARGETS:
        target = max(min_spd, min(max_spd, target))
        current = ri(c, 10)
        delta = target - current

```

```

pulse_count = abs(delta) // step_val
direction    = "Accel ()" if delta > 0 else "Decel ()"
bit_byte, bit_bit    = (2, 1) if delta > 0 else (2, 3)
stop_byte, stop_bit  = (2, 2) if delta > 0 else (2, 4)

print(f"\n Target: {target} rpm Current: {current} "
      f"Delta: {delta} Pulses: {int(pulse_count)} {
direction}")

try:
    for i in range(int(pulse_count)):
        pulse(c, bit_byte, bit_bit)
        readback = ri(c, 10)
        print(f"      Pulse {i+1}/{int(pulse_count)} "
              f" Desired_Speed={readback:4d} rpm", end="\
r")

    print()

    pulse(c, stop_byte, stop_bit)
    time.sleep(SETTLE)

    readback = ri(c, 10)
    ok = readback == target
    log(f"Change Speed {target} rpm", ok,
        f"Desired_Speed={readback} (clic={ri(c,26)})")
    if not ok:
        all_ok = False

    print(f" Monitoring for {RUN_SECONDS}s at {target}
rpm...")
    t0 = time.time()
    while time.time() - t0 < RUN_SECONDS:
        elapsed = time.time() - t0
        print(f"      t={elapsed:5.1f}s | desired={ri(c,10)
:4d} | "
              f"N(rpm)={rr(c,38):8.2f}", end="\r")
        time.sleep(POLL_INTERVAL)
    print()

except Exception as e:
    log(f"Change Speed {target} rpm", False, str(e))

```

```

        all_ok = False

    return all_ok

def step4_display(c):
    print(f"\n{CYAN}{' '*70}{RESET}")
    print(f"{CYAN}STEP 4: Display Speed ({DISPLAY_SECONDS}s){RESET}")
    print(f"{CYAN}{' '*70}{RESET}")
    try:
        print(f"    {'t(s)':>6}    {'Desired':>8}    {'N(rpm)':>10}    "
              f"{'Scale':>10}    {'CU_Meas':>8}    {'Act':>5}")
        print("    " + "-" * 55)
        t0 = time.time()
        samples = []
        while time.time() - t0 < DISPLAY_SECONDS:
            elapsed = time.time() - t0
            rpm = rr(c, 38)
            samples.append(rpm)
            print(f"    {elapsed:5.1f}    {ri(c,10):>8}    {rpm:>10.2f}"
                  "
                  f"{ri(c,22):>10}    {read_iw(c,258):>8}    "
                  f"{str(rb(c,0,4)):>5}")
            time.sleep(POLL_INTERVAL)
        avg = sum(samples) / len(samples) if samples else 0
        log("Display Speed", True,
            f"avg N(rpm)={avg:.2f} over {len(samples)} samples")
        return True
    except Exception as e:
        log("Display Speed", False, str(e))
        return False

def step5_stop(c):
    print(f"\n{CYAN}{' '*70}{RESET}")
    print(f"{CYAN}STEP 5: Stop Motor{RESET}")
    print(f"{CYAN}{' '*70}{RESET}")
    try:
        print("    Pulsing Stop bit (DBX0.1)...")
        pulse(c, 0, 1, hold=0.3)

```

```

    time.sleep(0.2)

    print("    Pulsing Deactivate bit (DBX0.5)...")
    pulse(c, 0, 5, hold=0.3)

    # Wait for Activation to clear
    for _ in range(25):
        if not rb(c, 0, 4):
            break
        time.sleep(0.1)

    act = rb(c, 0, 4)
    qw = read_iw(c, 258)
    rpm = rr(c, 38)

    if not act:
        log("Stop Motor", True,
            f"Activation cleared, QW258={qw}, N(rpm)={rpm:.1f
}"))
        return True
    else:
        log("Stop Motor", False,
            "Activation still True after Stop + Deactivate
pulse")
        return False
    except Exception as e:
        log("Stop Motor", False, str(e))
        return False

```

Main

```

def main():
    print("=" * 70)
    print("  MOTOR SPEED TEST  Snap7 S7-1214C")
    print(f"  PLC: {PLC_IP}  Rack: {RACK}  Slot: {SLOT}  DB: {
DB_NUM}")
    print("=" * 70)

    c = step1_connect()
    if not c:

```

```
        print(f"\n{RED}Cannot continue without PLC connection.{
RESET}")
        summary()
        sys.exit(1)

    try:
        step2_start(c)
        step3_change_speed(c)
        step4_display(c)
        step5_stop(c)
    except KeyboardInterrupt:
        print(f"\n{YELLOW}Interrupted sending emergency stop...{
RESET}")
        try:
            wb(c, 0, 1, True)
            time.sleep(0.5)
            wb(c, 0, 1, False)
            time.sleep(0.2)
            wb(c, 0, 5, True)
            time.sleep(0.5)
            wb(c, 0, 5, False)
        except Exception:
            pass
    finally:
        try:
            c.disconnect()
            print(f"\n{CYAN}PLC disconnected.{RESET}")
        except Exception:
            pass

    summary()

if __name__ == "__main__":
    main()
```

Bibliography

- [1] Siemens AG, *SIPLUS S7-300 CPU 314 Catalog*. [Online]. Available: <https://sieportal.siemens.com/en-jp/products-services/detail/6AG13141AG147AB0?tree=CatalogTree>. [Accessed: May 2026].
- [2] F. D. Petruzella, *Programmable Logic Controllers*, 5th ed. New York, NY, USA: McGraw-Hill, 2017.
- [3] Ladder Logic World, “Relay Logic vs. Ladder Logic,” *ladderlogicworld.com*. [Online]. Available: <https://ladderlogicworld.com/relay-logic-vs-ladder-logic/>. [Accessed: May 2026].
- [4] Banner Engineering, “Smart Sensors Applications: Pharmaceutical Bottle Detection,” *bannerengineering.com*. [Online]. Available: <https://www.bannerengineering.com/us/en/solutions/smart-sensors/pharmaceutical-bottle-detection.html>. [Accessed: May 2026].
- [5] XiongBa, “Difference Between Compact PLC and Modular PLC,” *plc-module.com*. [Online]. Available: <https://www.plc-module.com/News/94.html>. [Accessed: May 2026].
- [6] SolisPlc, “Siemens PLC Programming: Ladder Logic, HMI Development and Code Simulation,” *solisplc.com*. [Online]. Available: <https://www.solisplc.com/tutorials/siemens-plc-hmi-development-simulation>. [Accessed: May 2026].
- [7] Siemens AG, *SIMATIC S7-1200 Programmable Controller System Manual*. [Online]. Available: https://cache.industry.siemens.com/dl/files/465/36932465/att106119/v1/s71200_system_manual_en-US_en-US.pdf. [Accessed: May 2026].
- [8] M.A.C. SolUK, “Ewon Talk2M Remote Access Overview,” *YouTube* [Video]. [Online]. Available: <https://www.youtube.com/watch?v=AGoJZ07-inU>. [Accessed: May 2026].
- [9] M.A.C. SolUK, “Ewon Talk2M Remote Data Overview,” *YouTube* [Video]. [Online]. Available: https://www.youtube.com/watch?v=TsXUkJ_p0Gc&t=7s. [Accessed: May 2026].

- [10] IoT-Open, “European Internet of Things (IoT) Education Project,” *iot-open.eu*. [Online]. Available: <https://iot-open.eu/>. [Accessed: May 2026].
- [11] “Remote Control and Monitoring Protocols for PLC and HMI Systems: A Case Study of MODBUS, WMI and VNC,” *ResearchGate*. [Online]. Available: [https://www.researchgate.net/publication/394479128 Remote Control and Monitoring Protocols for PLC and HMI Systems A Case Study of MODBUS WMI and VNC](https://www.researchgate.net/publication/394479128_Remote_Control_and_Monitoring_Protocols_for_PLC_and_HMI_Systems_A_Case_Study_of_MODBUS_WMI_and_VNC). [Accessed: May 2026].
- [12] “Developing Web and TwinCAT PLC-Based Remote Control Laboratories for Modern Web-Browsers or Mobile Devices,” in *Proc. IEEE Conf.*, 2016. [Online]. Available: <https://ieeexplore.ieee.org/abstract/document/7587918>.
- [13] “Novel Networked Remote Laboratory Architecture for Open Connectivity Based on PLC-OPC-LabVIEW-EJS Integration: Application in Remote Fuzzy Control and Sensors Data Acquisition,” *Sensors*, vol. 16, no. 11, p. 1822, 2016. [Online]. Available: <https://www.mdpi.com/1424-8220/16/11/1822>.
- [14] A. Oussama and H. Chaima, “Identification et commande de la vitesse d’un moteur asynchrone via un variateur de vitesse connecté avec un API S7 1200 de marque SIEMENS,” M.S. thesis, Univ. Tlemcen, Tlemcen, Algeria, 2021. [Online]. Available: <https://dspace.univ-tlemcen.dz/items/c2ac0e4b-f10b-4d96-8ad0-f2ccf81693bc>.
- [15] Raspberry Pi Foundation, “Raspberry Pi 4 Model B,” *raspberrypi.com*. [Online]. Available: <https://www.raspberrypi.com/products/raspberry-pi-4-model-b/>. [Accessed: May 2026].
- [16] Raspberry Pi Foundation, “Raspberry Pi OS Documentation,” *raspberrypi.com*. [Online]. Available: <https://www.raspberrypi.com/documentation/computer-s/os.html>. [Accessed: May 2026].
- [17] Siemens AG, *SIMATIC S7 Communication*, Doc. No. A0350. [Online]. Available: https://cache.industry.siemens.com/dl/files/889/90471889/att_114738/v1/a0350.pdf. [Accessed: May 2026].
- [18] Netfilter.org, “The Netfilter.org ‘iptables’ Project,” *netfilter.org*. [Online]. Available: <https://netfilter.org/projects/iptables/>. [Accessed: May 2026].
- [19] “OPC UA Data Exchange Between MATLAB and TIA Portal with Application to the Control of a Coupled Twin Tank System,” M.S. thesis, Univ. Jijel, Jijel, Algeria, 2024. [Online]. Available: <http://dspace.univ-jijel.dz:8080/xmlui/bitstream/handle/123456789/15812/M-AUT.INF.IN-2024-03.pdf>.

- [20] Siemens AG, *SIMATIC NET PC Software: Industrial Communication with PG/PC, Vol. 1 — Basics*, Doc. No. SYH_OPC_76. [Online]. Available: https://cache.industry.siemens.com/dl/files/110/77376110/att_904388/v1/SYH_OPC_76.pdf.
- [21] Siemens AG, *SIMATIC NET Industrial Ethernet / PROFINET Industrial Ethernet*. Nuremberg, Germany: Siemens AG.
- [22] Snap7 Project, “Snap7: Open-Source, Cross-Platform Ethernet Communication Suite for Siemens S7 PLCs,” *snap7.sourceforge.net*. [Online]. Available: <https://snap7.sourceforge.net/>. [Accessed: May 2026].
- [23] Pallets, “Flask: Lightweight WSGI Web Application Framework,” *flask.palletsprojects.com*. [Online]. Available: <https://flask.palletsprojects.com/en/stable/>. [Accessed: May 2026].
- [24] Pallets, “Deploying Flask with Gunicorn,” *flask.palletsprojects.com*. [Online]. Available: <https://flask.palletsprojects.com/en/stable/deploying/gunicorn/>. [Accessed: May 2026].
- [25] I. Sysoev, “NGINX,” NGINX, Inc., 2004. [Online]. Available: <https://nginx.org/en/>. [Accessed: May 2026].
- [26] D. Crockford, “JSON (JavaScript Object Notation),” 2001. [Online]. Available: <https://www.json.org>. [Accessed: May 2026].
- [27] SoftEther VPN Project, “SoftEther VPN Manual, Sec. 1.1,” *softether.org*. [Online]. Available: <https://www.softether.org/4-docs/1-manual/1/1.1>. [Accessed: May 2026].
- [28] SoftEther VPN Project, “SoftEther VPN Manual, Sec. 1.2,” *softether.org*. [Online]. Available: <https://www.softether.org/4-docs/1-manual/1/1.2>. [Accessed: May 2026].
- [29] SoftEther VPN Project, “Layer-2 Ethernet-Based VPN,” *softether.org*. [Online]. Available: <https://www.softether.org/1-features/2.Layer-2-Ethernet-based-VPN>. [Accessed: May 2026].
- [30] M. Powell, “What Is the OSI Model? The 7 Layers Explained,” *Corero Network Security*. [Online]. Available: <https://www.corero.com/what-is-the-osi-model/>. [Accessed: May 2026].
- [31] PLC Monitor, “PLC Monitor Application,” *GitHub*. [Online]. Available: <https://github.com/PLC-MONITOR>. [Accessed: May 2026].
- [32] Illustration created by the author.

- [33] Screenshot captured by the author using Windows Registry Editor.
- [34] Screenshot captured by the author using Windows Terminal.
- [35] Screenshot captured by the author using Siemens TIA Portal V16.
- [36] “Adaptive Autonomous Assistance Using Raspberry Pi,” *ResearchGate*. [Online]. Available: https://www.researchgate.net/publication/390163146_Adaptive_Autonomous_Assistance_Using_Raspberry_Pi. [Accessed: May 2026].
- [37] Tables and figures prepared by the author.
- [38] Screenshot captured by the author using SoftEther VPN software.
- [39] Photograph of the experimental platform taken by the author.
- [40] PLC Monitor logo designed by the author.
- [41] Screenshot of the PLC Monitor application captured by the author.

ملخص

يقترح هذا المشروع نظامًا منخفض التكلفة وآمنًا للبرمجة والمراقبة والتحكم عن بُعد في وحدات التحكم المنطقية القابلة للبرمجة (PLC) الصناعية عبر الإنترنت. تعمل بوابة **Raspberry Pi 4** على التواصل مع وحدات **Siemens SIMATIC PLC** باستخدام بروتوكول **S7** (من خلال مكتبة **Snap7**)، في حين يضمن نفق **SoftEther VPN** من الطبقة الثانية (Layer 2) وصولًا آمنًا عن بُعد، واتصالًا كاملاً ببرنامج **TIA Portal**، وامتدادًا شفافًا للشبكة المحلية (LAN)، مع إمكانية توسيع الحل ليشمل منصات **Rockwell** و **Schneider**. تم التحقق من فعالية الحل على نظام حقيقي للتحكم في سرعة محرك، وتُوج ذلك بتطوير تطبيق **PLC Monitor**، وهو لوحة تحكم ويب مخصصة مبنية باستخدام **Flask** و **Nginx**، تُمكن من مراقبة متغيرات PLC والتحكم فيها في الوقت الفعلي.

الكلمات المفتاحية: الإشراف على الأتمتة عبر الويب، الوصول الصناعي عن بُعد، الشبكة الافتراضية الخاصة من الطبقة الثانية، **TIA Portal**، **Snap7**، بوابة **Raspberry Pi**، إنترنت الأشياء الصناعي.

Abstract

This project proposes a low-cost, secure system for remote programming, monitoring, and control of industrial **PLCs** via the Internet. A **Raspberry Pi 4 gateway** communicates with **Siemens SIMATIC PLCs** using the **S7** protocol (**Snap7 library**), while a **SoftEther VPN** Layer 2 tunnel ensures secure **remote access**, full **TIA Portal** connectivity, and transparent LAN extension — extendable to **Rockwell** and **Schneider** platforms. The solution is validated on a real motor speed control system, culminating in the development of **PLC Monitor**, a custom **Flask/Nginx** web dashboard enabling real-time PLC variable monitoring and control.

Keywords: Web-Based PLC Supervision, Remote Industrial Access, Layer 2 VPN, TIA Portal Remote Connectivity, Snap7, Raspberry Pi Gateway, IIoT.

Résumé

Ce projet propose une solution économique et sécurisée pour la programmation, la supervision et le contrôle à distance des automates industriels via Internet. Une passerelle **Raspberry Pi 4** communique avec les automates **Siemens SIMATIC** via le protocole **S7** (bibliothèque **Snap7**), tandis qu'un tunnel **VPN SoftEther** de couche 2 garantit un accès distant sécurisé, une connectivité complète avec **TIA Portal** et une extension transparente du réseau local — extensible aux plateformes **Rockwell** et **Schneider**. La solution est validée sur un système réel de contrôle de vitesse moteur, aboutissant au développement de **PLC Monitor**, un tableau de bord web personnalisé sous **Flask/Nginx** permettant la supervision et le contrôle en temps réel des variables automate.

Mots-clés : Supervision API via Web, Accès Industriel à Distance, VPN de Couche 2, Connectivité à Distance TIA Portal, Snap7, Passerelle Raspberry Pi, IIoT.

