Wiley
Journal of Engineering
Volume 2024, Article ID 2822711, 10 pages
https://doi.org/10.1155/2024/2822711

WILEY

Research Article

Virtual Commissioning with TIA Step7 and Simulink without S-Functions

Dusan Horvath,[1] **Martin Klauco**,[2] **and Maximilian Stremy**[1]

[1]*Advanced Technologies Research Institute, Faculty of Materials Science and Technology in Trnava, Slovak University of Technology in Bratislava, Jána Bottu 2781/25, Trnava 917 24, Slovakia*
[2]*Institute of Information Engineering, Automation, and Mathematics, Faculty of Chemical and Food Technology STU in Bratislava, Slovak University of Technology in Bratislava, Radlinského 2101/9, Bratislava 812 37, Slovakia*

Correspondence should be addressed to Dusan Horvath; dusan_horvath@stuba.sk

Received 15 January 2024; Revised 24 June 2024; Accepted 11 July 2024

Academic Editor: Franca Giannini

This paper presents the development and validation of a custom MATLAB library, designed to facilitate seamless and efficient real-time communication between the Siemens S7-PLCSIM Advanced simulator and MATLAB/Simulink. The library uses Siemens.Simulation.Runtime.API to enable this integration and its architecture is structured into three classes—PLCSimAdv, PLCSR, and PLCSW. To demonstrate the library's capabilities, we have chosen the validation parameters and conducted functional experiments including real-time communication at the millisecond level, integration and control of a simulated industrial process, and simultaneous operation with multiple virtual controllers. One of the library's preliminary conditions was to avoid the requests for regenerating of any version of control program or Simulink model as it is presently by S-functions in TIA portal. The results suggest that the proposed library provides a robust tool for various applications in industrial automation, from the digital twin modelling or simulation of advanced technologies to the operation of traditional control systems. This work has the potential to significantly streamline the simulation, control, and validation of automated processes and opens new ways for research and development in automation and control systems.

1. Introduction

Computer simulation plays a critical role in scientific research and development [1–3]. It uses digital computing to save both time and money [4–6]. These days, we are seeing increased blending of the physical world with the virtual one. This offers a lot of advantages. For example, we can simulate how processes and control systems work, which helps us adjust process parameters, optimize control programs, and validate systems. Plus, virtual commissioning, or evaluating a system in a virtual environment before real-world deployment, is a key part of Industry 4.0, the latest industrial revolution that heavily involves digitization and automation [7, 8]. Numerous tools and methods are available for modelling and simulation, ranging from Simulink to software specifically designed for industrial systems such as Tecnomatix Plant Simulation and Factory IO. For the integration with Siemens-based Programmable Logic Controllers (PLCs), simulation of control programs, and virtual commissioning, we utilized MATLAB and Simulink's inherent libraries, as well as customized ones, to develop models and simulate processes across diverse industrial sectors. Siemens provides the capability to interconnect a virtual controller operating in PLCSim Advanced with Simulink through TIA Portal-generated s-functions or OPC UA. However, this feature is exclusive to the specific software controller S7-1500S, and any alterations to the PLC program necessitate the regeneration of the s-function.

Our work is centered around the creation of a library that facilitates direct connections to one or multiple virtual

controllers, eliminating the need for s-function regeneration. Therefore, no modifications into the Simulink model are required following alterations to the PLC control program, thereby overcoming the restriction of solely using the 1500S PLC type and ensuring uninterrupted real-time operation. Such a library will extend the present possibilities of virtual commissioning.

1.1. Motivation. Programmable Logic Controller (PLC) simulation is a critical tool in the design and verification of algorithms, particularly in the realm of industrial control systems. It works as a virtual space for engineers to create, check, and confirm control systems. They can assess how the system works using a digital model of the industrial plant. This approach reduces the risk to machine operators and the machine itself, as potential issues can be identified and resolved in the simulated environment before actual deployment.

In the field of industrial automation, the process of commissioning is crucial. Equally important is the deployment of PLC in the context of Industry 4.0 concept. In the case of modern approach with virtual commissioning applied, a lot of resources are saved [9, 10].

The process of PLC simulation plays a key role in the Model-Based Design (MBD) workflow. MBD is a mathematical and visual method of addressing problems associated with designing complex control, signal processing, and communication systems. It enables engineers to use simulation to design control algorithms (such as PID control), prototype control logic (such as finite state machines), and even generate automatic PLC code (IEC 61131-3, C, or C++ code) from the design and simulation environment. This approach provides flexibility and independence from the PLC hardware platform and contributes to greater reliability through testing and avoidance of programming errors by automatically generating code from models.

Aiming our attention on Simatic PLCs, we can utilize s-functions generated in the TIA portal, which effectively tests our s7 code. However, the utilization of s-functions is accompanied by several challenges. For example, debugging an s-function can be significantly more complex compared to debugging a standard Simulink model. Moreover, as a model evolves, the associated s-functions necessitate regular maintenance and updates, potentially requiring additional effort compared to the use of built-in blocks. The generation of s-functions in the TIA portal is also constrained to the software PLC S7-1500S.

During commissioning, code modifications are not uncommon, thereby necessitating the time-consuming regeneration of s-functions for the modified model. Our objective is to simplify the simulation process and facilitate the use of a wider range of simulated controllers. This approach not only enhances work efficiency but also aligns with the need for cost-effective deployments.

Given our institutional emphasis on ion beam technologies, the proposed solution and the PLCSimAdv library aim to be employed for modelling and simulating processes associated with the ion beam accelerator (IBA) as we mentioned and examined in our previous article [11]. This includes the simulation of the automatic configuration of accelerator optics for multienergy ion implantation [12–14]. Such an implementation demands a specific accelerator configuration, which may differ for each energy level. Each reconfiguration entails shutting down the accelerator, resulting in considerable system downtime. The library we have developed could assist us in devising an optimal algorithm for continuous parameter reconfiguration during the implantation process. There are more interesting areas where our library could be used, e.g., in the experimental measuring system based on PLC [15]. Additionally, this tool will be helpful for our future research in process automation and control systems. It will open new paths for us to explore in this field.

2. Materials and Methods

In the design and development of the library, an object-oriented approach was employed. Our solution is comprised of three classes: the primary class, PLCSimAdv, serves as the interface between Siemens.Simulink.Runtime.API and the MATLAB application. Figure 1 shows a basic class model. The classes PLCSR and PLCSW are derived from the matlab.System class and are thus utilized as library blocks within the Simulink environment. Figure 2 illustrates the potential physical connectivity, based on the standard PLCSim Advanced connection configurations and capabilities.

2.1. PLCSimAdv. The PLCSimAdv class is designed as a static-like class to facilitate between MATLAB and the Siemens S7-PLCSIM Advanced simulator, fully developed by the authors. This class enables seamless communication and data sharing with the simulator, crucial for simulating and testing PLC programming in a virtual environment.

Key properties and setup:

(i) *Library Path and Files*. The class maintains properties that specify the paths to essential API files necessary for interfacing with the simulator. These properties include default paths for both 32 bit and 64 bit versions of the library, ensuring compatibility across different system architectures.

(ii) *Automatic Initialization*. Initialization of this interface is streamlined through the execution of a startup.m file, which prepares the MATLAB environment for communication with PLCSim Advanced by setting up necessary paths and configurations.

Core methods and functionalities:

(i) *Library Management*. It includes methods for loading library files, checking the existence of these libraries, and loading the required .NET assemblies to ensure all dependencies are properly managed and ready for use.

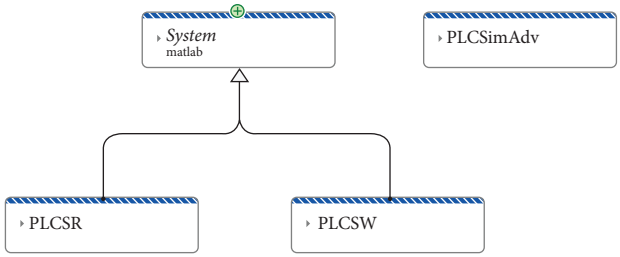

Figure 1: Simplified class model of library.

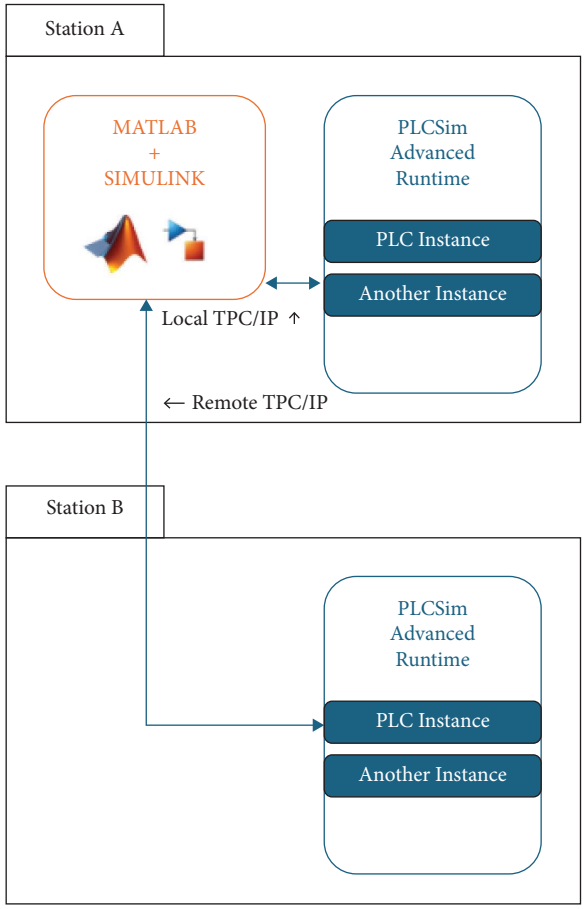

Figure 2: PLCSim advanced communication.

(ii) *Instance Management*. It utilizes a persistent container, Instances, to register and manage simulator instances. This design pattern ensures efficient management of multiple connections to the simulator, reducing overhead and enhancing performance.

(iii) *Dynamic Instance Handling*. The getInstance() static method is a critical component of the class, designed to either retrieve an existing instance from the container or create a new one if it is not already available. This method supports dynamic connection handling, allowing users to work with multiple simulated PLC instances concurrently.

Interaction with simulator:

(i) *Tag Management*. It includes functionality to read and write PLC tags, which are essential for real-time simulation and testing. This ability is pivotal for developing and debugging PLC programs within MATLAB.

(ii) *Error Handling and Data Type Verification*. It provides error handling mechanisms to deal with common issues such as missing tags or incorrect configurations. Additionally, it can retrieve and display data types of PLC tags, enhancing debugging capabilities.

Enhanced communication:

(i) *Secure and Efficient Communication*. Methods like write() and read() are optimized for secure and efficient communication between MATLAB and the simulator, ensuring that data integrity is maintained during transactions.

Overall, the PLCSimAdv class provides an intuitive and powerful way to interact with the Siemens S7-PLCSIM Advanced simulator from MATLAB. It abstracts the complexity of direct API calls into a user-friendly interface, allowing for enhanced productivity in developing, testing, and simulating PLC programs.

2.2. PLCSR. The PLCSR class is designed as a MATLAB class definition tailored for a Simulink block that facilitates reading values from Siemens PLCSim Advanced simulation software. Specifically, it retrieves data from a designated variable tag within the simulation and outputs these data as a double-precision scalar.

Key properties and configuration:

(i) *Runtime Configuration*. It includes properties such as RuntimeManagerIP, RuntimeManagerPort, InstanceName, and VariableTag, which are used to configure the connection to the Siemens PLCSim Advanced instance. These properties ensure the class can dynamically connect to and interact with the simulation based on user-defined settings.

(ii) *Initialization and Data Retrieval*. The setupImpl method is pivotal in initializing the connection to the PLCSim Advanced. It establishes the PLC object using the InstanceName, RuntimeManagerIP, and RuntimeManagerPort and updates the list of tags available in the simulation. Additionally, it determines the data type of the variable tag to ensure accurate data handling.

Core functionalities:

(i) *Data Reading and Conversion*. The stepImpl method is the core functional method that reads the current value of the specified variable tag during each simulation step. It employs conditional checks on the data type of the tag to convert the read value into

the appropriate MATLAB data type (e.g., bool, int16, and double), ensuring the output is consistent regardless of the underlying data type in the simulation.

(ii) *Output Configuration*. This class includes methods such as getOutputNamesImpl, which sets the output port name to the VariableTag name, and getOutputSizeImpl, ensuring the output is a scalar. It also defines the output data type through getOutputDataTypeImpl and specifies that the output is not complex via isOutputComplexImpl.

(iii) *Error Handling and Data Types*. The class is equipped to handle errors related to unsupported or unknown data types through the stepImpl method, which throws an informative error if the data type of the variable tag is not recognized.

By encapsulating the functionality to interact with the PLCSim Advanced within a Simulink block, the PLCSR class makes it easier for users to integrate PLC simulation data into broader Simulink models, supporting more comprehensive simulation and testing environments. Overall, the PLCSR class provides an efficient and user-friendly way to incorporate real-time data from Siemens PLCSim Advanced into MATLAB and Simulink workflows, enhancing the capability for detailed simulation and analysis in industrial automation projects.

2.3. PLCSW. The PLCSW class is a custom MATLAB System block designed for Simulink, facilitating the writing of values to a Siemens PLCSim Advanced instance. This class enables precise interaction with the simulation software by writing data directly to specified variable tags, which are critical components of the TIA project's variable table.

Key properties and configuration:

(i) *Connection Settings*. The class properties include RuntimeManagerIP, RuntimeManagerPort, InstanceName, and VariableTag. These settings configure the connection to the specified instance of PLCSim Advanced, ensuring targeted communication within the simulation environment.

(ii) *Initial Value Setup*. An additional property, InitValue, is used to set an initial value for the specified tag if it is not NaN (Not a Number), facilitating initial conditions or default settings for simulation runs.

Core functionalities:

(i) *Initialization and Setup*. The setupImpl method initializes the PLCSimAdv object, updates the list of available tags, and fetches the data type of the specified variable tag to ensure accurate data handling. This method also handles the initial writing of values if specified.

(ii) *Data Writing Mechanism*. The write method, which is crucial for the operation of this block, dynamically converts the input value to the appropriate data type of the tag before writing it to the simulation. This conversion ensures that data integrity and type consistency are maintained, preventing errors during simulation.

(iii) *Simulation Step Integration*. In the stepImpl method, the input value from Simulink is passed directly to the write method, ensuring that the data in the simulation are updated at every simulation step, reflecting real-time changes and interactions.

(iv) *Input Port Customization*. The getInputNamesImpl method sets the input port's name to match the VariableTag, enhancing the usability and clarity of the Simulink model, making it easier to trace and verify simulation flows.

(v) *Error Handling and Data Types*. It supports a wide range of data types from Boolean to 64 bit integers and floating-point numbers, accommodating various simulation needs. It provides error messages for unknown data types, ensuring that users are immediately aware of configuration issues.

Overall, the PLCSW class significantly streamlines the process of integrating and manipulating data within Siemens PLCSim Advanced via Simulink, providing a robust tool for engineers and researchers to simulate and test PLC programs effectively. By automating the data writing process and ensuring data type compatibility, this class enhances the simulation's accuracy and reliability, making it an invaluable tool for development and testing in industrial automation projects.

3. Development

The development of the Simulink library began with a comprehensive study of the Simulink runtime API. Essential methods and functions were identified that would enable effective utilization of the API for our specific purposes. This foundational step ensured a deep understanding of the available tools and how they could be adapted to meet the needs of our project. Subsequent to familiarizing ourselves with the necessary API functionalities, we investigated the process of creating libraries and modules within Simulink. This investigation provided insights into best practices and methodologies for structuring custom libraries that are both efficient and scalable, serving as the backbone for the subsequent development phases. The first major development milestone was the creation of the PLCSimAdv class. This class was designed as a read/write library that facilitated direct interaction with PLC simulations but initially did not interact with Simulink itself. To extend our library's capabilities, the PLCSR class was developed to test reading from PLC instances, followed by the PLCSW class, which enabled writing to PLC instances. These classes formed the core of our Simulink library, allowing for robust data manipulation within the PLC simulations.

The development process was inherently iterative, overlapping various stages of design, implementation, and testing. This approach allowed for continuous refinement of

the library through trial and error, ensuring that each functionality was thoroughly vetted and optimized before finalization. While iterative and occasionally involving setbacks, this method proved crucial in achieving a highly functional and reliable final product.

Once the core classes were established and functioning as intended, extensive testing and experimental trials were conducted. These tests were crucial for evaluating the library's performance, particularly the response times and reliability under different operational conditions. The insights gained from these experiments were instrumental in further refining the library, ensuring optimal performance and user experience.

4. Experiments

To confirm the correctness and to validate the library functions, six validation parameters were chosen, and we conducted three experiments. In the first experiment, we tested the library's ability to communicate in real time, meaning that data exchange between the virtual PLC and Simulink will occur at the millisecond level. In the second experiment, we focus on modelling the process of filling a water tank, while examining the library's ability to facilitate communication between the virtual controller and the simulated model. The last experiment aims to confirm the library's ability to communicate with multiple virtual controllers simultaneously. The results of the time-based experiments performed on different stations may slightly vary, which can be caused by the different computing power of various computers. To provide a more comprehensive evaluation, we included objective performance metrics and comparative analysis with existing solutions. The experiments were designed, constructed, and selected based on the validation parameters outlined in the subsequent section.

4.1. Selection of Validation Parameters. The main discussion is about picking appropriate parameters to test our library in the next experiments. These parameters will help us assess key features of our library, such as real-time communication, process control, working with multiple PLCs, and managing different numeric data types. The following parameters have been selected for validation during the experimental phase:

(a) *Real-Time Communication*. Given the time-sensitive nature of industrial processes, the library exhibits the capability to transfer data within milliseconds. Hence, the response time of the library would be a key parameter that we intend to validate.

(b) *Industrial Process Control*. The library is proficient at handling complex control operations, such as PID regulation. The performance and accuracy of PID regulation under different operating conditions would form an essential part of our validation process.

(c) *Multi-PLC Cooperation*. One of the key validation parameters involves assessing the library's ability to facilitate seamless communication and coordination among multiple PLCs. We plan to evaluate this by running simultaneous processes involving multiple PLCs and gauging the effectiveness of the coordination.

(d) *Numeric Data Types*. We intend to validate the library's ability to handle all valid numeric data types, including Boolean. This will be carried out by testing the library's compatibility with a range of devices and equipment that use various data types.

(e) *Independence of Specific PLCSim Instance*. The library's flexibility and adaptability would be tested by deploying it across different PLCSim instances. The expectation is that the library functions seamlessly, regardless of the specific PLCSim instance in use.

(f) *Immediate and Accurate PLC State Reflection*. The PLCSimAdv class promptly and accurately mirrors changes in the PLC state. We plan to validate this by closely monitoring the correlation between actual PLC state changes and the corresponding reflection by the PLCSimAdv class.

To ensure a thorough evaluation, we also included metrics such as latency, error margins, and performance benchmarks against existing solutions. These parameters have been thoughtfully chosen to ensure that the library's capabilities align with the requirements of industrial automation tasks. The expectation is that the library will successfully meet the criteria set by these validation parameters in the forthcoming experiments, thereby demonstrating its suitability and effectiveness for use in real-world scenarios.

4.2. Preliminaries. S7-PLCSim Advanced is a product from Siemens AG that works as a virtual PLC (Programmable Logic Controller). It helps in testing and running S7 programs, which means a physical PLC is not needed. The software can operate in real time and work well with programming done through the TIA Portal. In our project, we used PLCSim Advanced V5.0 and MATLAB R2023a to develop the library. We programmed the PLC instructions with TIA Portal V18, and we picked a virtual controller CPU 1515-2 PN for our setup. We made sure our experimental setup meets the Nyquist criterion ($f_S \geq 2B$) for system control. According to the Nyquist theorem, the PLC's sampling frequency (f_S) must be double or more than the highest frequency (B) of the original signal from the Simulink model. So, we set the PLC sampling frequency to 5 kHz, which is more than double the model's 100 Hz frequency, satisfying the criterion. We set the PLC's minimum cycle time to "disabled" state, which is typically less than 0.2 ms. This lets the PLC sample at around 5 kHz. In contrast, the Simulink model uses a fixed-step solver that works with an automatic solver type and an open-ended step size of 10 ms. This setup means the Simulink model can sample at 100 Hz. We chose these conditions carefully to make sure the control system remains reliable and stable during the experiment.

4.3. The First Experiment: Real-Time Communication. The primary objective of this experiment was to validate the capability of the library in facilitating real-time communication between Simulink and PLCSim Advanced. The Simulink system model comprised a single PLCSR and one PLCSW block, which were interconnected (see Figure 3).

The OUTPUT tag's value was seamlessly transferred to the INPUT tag. A simple program was implemented in the PLC. This program compared the values of the OUTPUT and INPUT tags, and if they were the same, the value of the OUTPUT tag was changed. The simulation came to a halt after 5000 such changes. At the onset of the experiment, the time was recorded and subsequently subtracted from the end time of the simulation to determine the total duration of the simulation. Table 1 presents the details of each individual measurement along with their parameters.

The first experiment's results, summarized in Table 1, show that we can make the virtual PLC and Simulink communicate in real time, although there are limitations. We managed to make this real-time communication work, and the average delay was around 60 ms, which is acceptable in many cases. The data in Table 1 also show that transferring a block of data is quicker than transferring individual pieces. For example, it is quicker to transfer one INT than to transfer four BOOLs. This pattern is the same for various kinds of data and various levels of complexity, which tells us that how we structure the data matters for communication speed. Also, the choice of Simulink solver step makes a difference to performance. In general, using the "Variable discrete" solver step tends to be slower, so we need to think about this if we are trying to make things faster. However, there are exceptions to this, such as with the 4x REAL data transfer, where "Variable discrete" was faster. This tells us that the best choice of solver step might depend on what kinds of data we are transferring, and how much. This could be caused by various reasons:

(i) Size and structure of different data types may vary which could affect how the solver processes them.

(ii) The efficiency of a solver can vary based on the nature of the data being transferred and the computational load it imposes.

(iii) Real-time communication necessitates precise synchronization between Simulink and the PLC. The solver step size can influence this synchronization, as it determines the frequency at which data are processed and exchanged.

4.4. The Second Experiment: PI Control of Tank-Filling Simulink Model. For the second experiment, we utilized a model simulating a tank filling with liquid. We tested the model's (water tank) system response to a unit step (see Figure 4).

The water-tank Simulink model inspired this model. The filling process was regulated by a PI (Proportional-Integral) controller situated in a PLC. To facilitate this, we incorporated a PI regulator into a PLC control subsystem (see Figure 5). This subsystem featured PLCSR and PLCSW blocks for communication with the PLC. In the Step7 TIA program, we used OB30 (50 ms), which initiated (called based on constant time) the "PID_Compact V2" block.

The mathematical model of PI regulator in Simulink, where P is the proportional gain and ITS is the integral time constant, is shown in the following equation:

$$y = P + \frac{IT_S}{s}. \tag{1}$$

The time sampling for PI in Simulink model was set to 50 ms (called OB30 with PI in PLC). The mathematical model of PI regulator "PID_Compact V2" used in TIA portal is shown in the following equation:

$$y = P + f_S \frac{1}{T_i s}. \tag{2}$$

Firstly, we set the Simulink model to run a stop time set to Inf and with pacing set to 1. We then fine-tuned the PID regulator, which was started in the TIA environment. This helped us adjust the P and I parameters of the regulator. Once the fine-tuning was done, we limited the Simulink model to run for only 5 seconds. When we started the simulation in Simulink, we saw the results shown in Figure 6.

It is worth noting that there is a time delay between PLCSim and Simulink caused by communication reasons. However, this kind of delay is absent when the PI regulator in Simulink controls the system. This time lag results in a minor shift in the system's behavior, leading to observable differences in how the systems operate. We used the standard MATLAB function xcorr to calculate the lag. This function computes the cross-correlation of the two signals, allowing us to measure the similarity between the MATLAB and VirtualPLC signals. We applied cross-correlation to compare the responses of both systems. Our results (see Figure 7) indicated an almost identical correlation with a maximal value of lag close to 1 (0.9991), suggesting the two signals were nearly identical. The calculation details are straightforward and can be easily verified. The values and the calculation script (Lag_calculation.m) are available in the "results" directory on our GitLab repository [16].

This high degree of similarity implies that the two systems respond in an exceedingly similar manner to the input, with minor deviations occurring with minimal delay. Therefore, we can infer from this analysis that our control systems are operating efficiently and performing as expected. Other values were obtained from the "stepinfo" function. The main values are shown in Table 2.

As mentioned earlier, the goal of this experiment was to evaluate the library's ability to manage a PID control system in a simulated environment. Specifically, we modelled the process of filling a water tank and examined the library's capability to facilitate real-time communication between the virtual PLC and the simulated model. Figure 8 shows the progression of control deviation over time during Experiment 2. The control deviation represents the difference between the desired setpoint and the actual system output. This figure illustrates how the deviation changes as the PID controller, implemented through our library, adjusts the

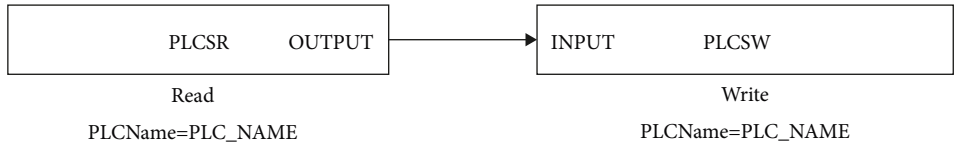

Figure 3: Experiment 1: output signals were written into PLC immediately. PLCSR represents a model of PLCSR class; PLCSW represents a model of PLCSW class.

Table 1: Results of experiment 1.

Exp. Nr.	Transferred data	Simulink solver step	AVG duration total (s)	AVG duration per cycle (ms)
1-1	1B	Fixed 2 ms	7.563	1.512
		VA	7.626	1.525
		VD	7.662	1.532
1-2	4B	Fixed 2 ms	17.148	3.430
		VA	17.931	3.586
		VD	31.307	6.261
1–3	1I	Fixed 2 ms	11.451	2.290
		VA	10.931	2.186
		VD	11.598	2.320
1–4	4I	Fixed 2 ms	47.136	9.427
		VA	49.336	9.867
		VD	46.768	9.354
1–5	1R	Fixed 2 ms	23.372	4.674
		VA	23.869	4.774
		VD	24.565	4.913
1–6	4R	Fixed 2 ms	97.893	19.579
		VA	97.741	19.548
		VD	72.746	14.549
1–7	2R + 6I + 10B	Fixed 2 ms	271.072	54.214
		VA	312.973	62.595
		VD	330.588	66.118

Simulink solver step (VA = variable auto; VD = variable discrete) and transferred data (R = real; I = integer; B = Boolean).

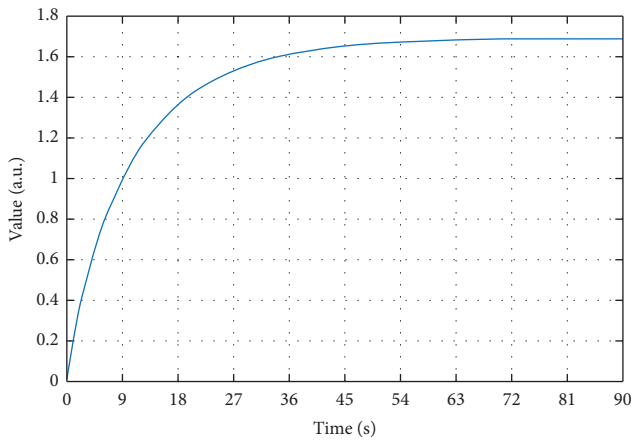

Figure 4: Experiment 2: unit step response (water tank model).

system to achieve the desired setpoint. The data show a similar progression of systems with maximal control deviation value 1.2184, as could be seen in Figure 8. In the second experiment, we made some important discoveries. The PLC, working with the PI controller, was great at managing the control model. This showed that we could successfully combine the PLC with the Simulink model. The communication between the PLC and Simulink was trustworthy, making sure that all input and output data were sent and received correctly. The experiment also highlighted the PLC's ability to deal with complex tasks, such as adjusting a PI controller for the tank filling process. In simple terms, the second experiment confirmed that using a PLC with Simulink to simulate and control processes like this one is a practical and reliable approach. The differences between the process controlled by the Simulink PI block and the PLC PI might not only come from the parameters used in each controller. Another factor that might affect the differences is the communication time between the Simulink and the PLC simulator. In the experiment, there is a certain amount of time needed for Simulink and the PLC simulator to exchange information. This time can add to the delay in the response of the system. If this communication delay is significant, it can influence the control behavior, resulting in different outcomes between the Simulink PI block and PLC PI controllers. It is crucial to understand that these differences do not indicate a problem. Instead, they highlight how different factors like parameter setup and communication delay can influence the behavior of the control system. Both controllers performed their tasks effectively, proving the

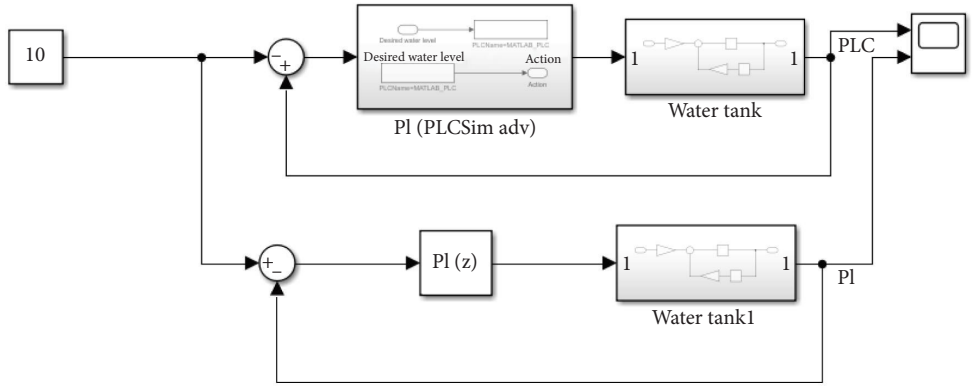

Figure 5: Experiment 2: the setup for a two-tank water system. The upper tank filling process is regulated by a PLC with a PI controller, while the lower tank is managed by a PI regulator within Simulink.

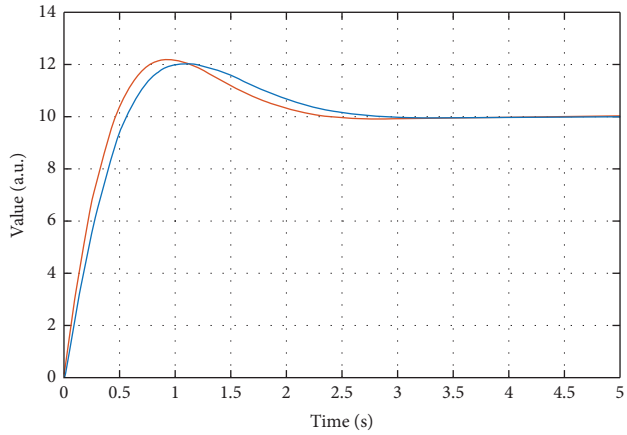

Figure 6: Experiment 2: results (red line: Simulink regulation; blue line: PLC regulation).

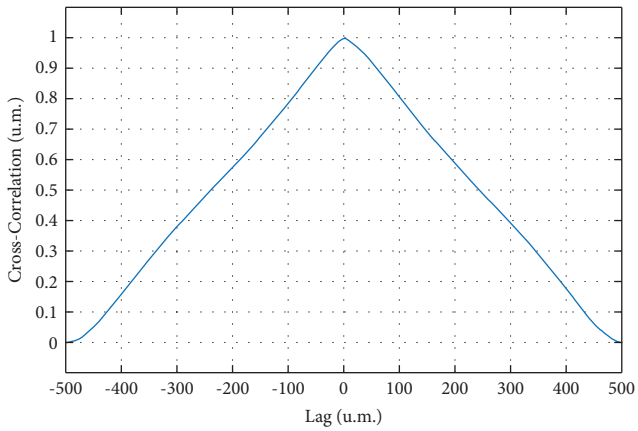

Figure 7: Experiment 2: cross-correlation of PLC and Simulink PI regulation results.

Table 2: Results of experiment 2.

	PLC	Simulink
Rise time	42.0453	35.3445
Overshoot	20.2420	21.7524
Peak	12.0263	12.1797

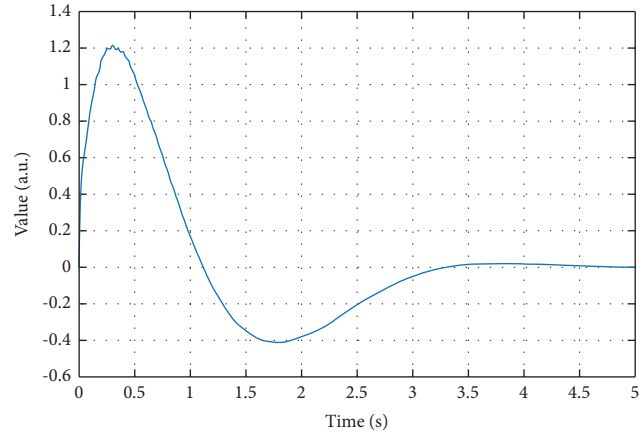

Figure 8: Experiment 2: control deviation progression.

usability of our library for complex tasks, even in the presence of such variables. It is important to remember that running the same simulation on different computers can give slightly different results. This can happen because each computer has a different number of resources and power to run the simulation. Things like how fast the computer's processor is, how much memory it has, and how efficient its operating system is can all affect how fast and accurately the simulation runs. So, you might see slight differences in results depending on the computer used for the simulation.

4.5. The Third Experiment: Multiple Instances. The aim of this experiment was to verify the feasibility of connecting several PLCs at the same time. We operated three PLCs (PLCa, PLCb, and PLCc) in the PLCSim Advanced environment. PLCa produced numbers representing angles in degrees, with an increase of 0.5 degrees every 50 ms. Simulink received this value and passed it to PLCb, which calculated the sine of the given angle. This calculated value was returned to Simulink and then forwarded to PLCc, which regenerated the original value. To maintain synchronization, we enabled simulation pacing and set it to one. Figure 9 visualizes the Simulink model used in this experiment. In relation to this experiment, the data from the complex interactions between three virtual controllers—PLCa,

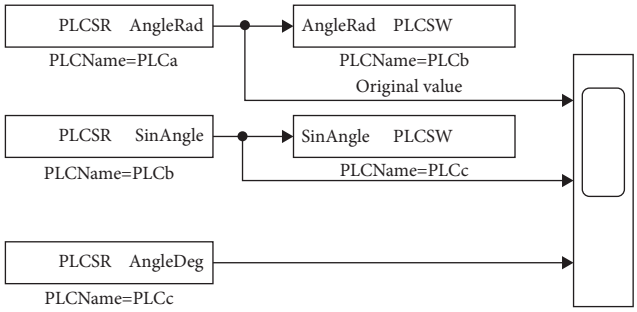

Figure 9: Experiment 3: simulation model in Simulink. The angle value from PLCa is read and sent to PLCb, where the sine of the angle is computed. This sine value is then transferred to PLCc, where it is decoded back into an angle.

Table 3: Results of experiments.

Parameter	Experiment 1	Experiment 2	Experiment 3
a	Valid	Valid	Valid
b	N/A	Valid	N/A
c	N/A	N/A	Valid
d	Valid	Valid	Valid
e	N/A	N/A	Valid
f	Valid	Valid	Valid

Parameters: as defined in chapter IV.A. Results: N/A = not available (experiment did not include test of this parameter). Valid = expectation confirmed.

PLCb, and PLCc—were obtained. These controllers were emulated and coordinated using our custom-built Simulink library. We used different types of PLC: S7-1518-4 PN/DP was chosen for PLCa, S7-1511-1 PN was chosen for PLCb, and S7-1513-1 PN was chosen for PLCc. The types of PLC selected here demonstrate the library's ability to work independently of the PLC type, as outlined in chapter III.A. PLCa generates a series of numbers, PLCb calculates the sine of these angles, and PLCc reproduces the original values in degrees. The changes in these numbers are recorded effectively and in real time, showing the successful communication enabled by the Simulink library.

From these observations, we can confidently affirm that the Simulink library meets our expectations and demonstrates robust capabilities in managing simultaneous operations of multiple virtual controllers. This not only underscores its potential in coordinating complex, multicontroller systems but also paves the way for further exploration into real-time data exchange and control in larger industrial setups. This experiment has confirmed the library's proficiency in handling simultaneous real-time communications among multiple virtual controllers.

5. Experiment Results

Findings from our research demonstrate the effectiveness and applicability of the library in various scenarios. One of the results is that small datasets appear to be more effectively processed. As shown in Table 3, a comprehensive summary of the results from each individual experiment is provided. In summary, the results prove the library's usability for each of the experiments.

6. Conclusions and Discussion

The comprehensive solution was built within the MATLAB environment, utilizing the Siemens Simulation Runtime API to bridge the gap between our model and the PLCSim Advanced virtual controllers. This integration facilitates the virtual commissioning of the newly developed industrial systems or simulations of multiple concurrent models, presenting a versatile tool for a diverse range of applications. Notably, these applications can span from the simulation of advanced material technologies, such as ion beam accelerator systems, to more traditional industrial systems. Moreover, the developed solution exhibits a compatibility range that includes even older control systems that remain operational in industrial factories globally. The custom-developed library in MATLAB, consisting of PLCSimAdv, PLCSR, and PLCSW classes, has shown significant advantages over the traditional s-function approach. By eliminating the need for regeneration of s-functions after control program modification, the proposed solution provides a seamless and efficient real-time operation. Additionally, three distinct experiments were conducted to validate the effectiveness and reliability of this solution. These tests verified real-time communication, the simultaneous use of multiple controllers, and the flexible configuration of different control strategies in a simulation environment. The library facilitates interconnection between the Simulink environment and PLCSim Advanced using the Simatic runtime API. This setup enables operation on a single machine when both applications are running on the same node. Additionally, it supports remote communication, allowing Simulink to run on one node and the runtime manager on another. This flexibility is made possible by the architecture of our library, which also permits communication among multiple runtime managers operating on various nodes simultaneously.

Our future goal is to expand the applicability of this solution and to conduct more thorough accuracy and performance tests. One area of focus could be the utilization of this tool in simulations related to ion beam accelerators (IBAs). This might facilitate the development of an efficient algorithm that can continually readjust parameters during the implantation process, minimizing system downtime. Another potential area, which has already been mentioned in this article, is the experimental measurement system based on PLC [15]. We also expect that our solution will play an important role in future research, especially in process automation and control systems. By offering capabilities for real-time and multicontroller simulations, this solution opens the door to a wide range of new research avenues. The whole source code of the library, simulation files, TIA Portal V18 project, and results of experiments are available online at [16].

Data Availability

The data that support the findings of this study are openly available in GitLab at https://gitlab.websupport.sk/dusanhorvath/plcsimadv-simulink-library, reference number [16].

Conflicts of Interest

The authors declare that they have no conflicts of interest.

Acknowledgments

We would like to extend our sincere gratitude to Assoc. Prof. Dr. Martin Juhás from the Institute of Applied Informatics, Automation, and Mechatronics at the Faculty of Materials Science and Technology in Trnava, Slovak University of Technology in Bratislava, for his invaluable assistance and expert advice. This research was funded by the European Regional Development Fund, Operational Program Research and Innovation, project "Scientific and Research Centre of Excellence SlovakION for Material and Interdisciplinary Research," no. ITMS2014+: 313011W085 and by the Ministry of Transport and Construction of the Slovak Republic, Operational program Integrated Infrastructure, project "Research and development into potential application of autonomous aircraft in the fight against the pandemic caused by COVID-19," no: 313010ATR9. M. Klaučo this research is funded by the European Union's Horizon Europe under grant no. 101079342 (Fostering Opportunities towards Slovak Excellence in Advanced Control for Smart Industries). Gratefully acknowledge the contribution of the Scientific Grant Agency of the Slovak Republic under the grants 1/0239/24, the Slovak Research and Development Agency under the projects APVV-21-0019. This article was written thanks to the generous support under the Operational Program Integrated Infrastructure for the project: 313021BXZ1 - Support of research activities of excellent STU laboratories in Bratislava, co-financed by the European Regional Development Fund.

References

[1] K. M. Kam, P. Saha, M. O. Tadé, and G. P. Rangaiah, "Models of an industrial evaporator system for education and research in process control," *Developments in Chemical Engineering and Mineral Processing*, vol. 10, no. 1-2, pp. 105–127, 2002.

[2] S. Pati, O. P. Verma, R. K. Arya, and A. K. Tiwari, "Transient modeling and simulation of a multiple-stage evaporator in the paper industry," *Chemical Engineering and Technology*, vol. 45, no. 3, pp. 456–466, 2022.

[3] S. Schütz, R. Schmidt, C. Henke, and A. Trächtler, "Virtual commissioning of the trajectory tracking control of a sensor-guided, kinematically redundant robotic welding system on a PLC," in *2022 IEEE International Systems Conference (SysCon)*, Montreal, Canada, April 2022.

[4] P. Ahrweiler and S. Wörmann, "Computer simulations in science and technology studies," in *Computer Simulations in Science and Technology Studies*, P. Ahrweiler and N. Gilbert, Eds., pp. 33–52, Springer, Berlin, Heidelberg, 1998.

[5] I. Fagarasan, S. Iliescu, and I. Stamatescu, "Process Simulator using PLC technology," *Scientific Bulletin-University Politehnica of Bucharest*, vol. 72, pp. 17–26, 2010.

[6] E. Winsberg, "Computer simulations in science," in *The Stanford Encyclopedia of Philosophy, Winter 2019*, E. N. Zalta, Ed., Metaphysics Research Lab, Stanford University, Stanford, CA, USA, 2019, https://plato.stanford.edu/archives/win2019/entries/simulations-science/.

[7] C. Bock, R. Barbau, I. Matei, and M. Dadfarnia, "An extension of the systems modeling language for physical interaction and signal flow simulation," *Systems Engineering*, vol. 20, no. 5, pp. 395–431, 2017.

[8] J. Cigánek and F. Žemla, "Design of digital twin for PLC system," in *Proceedings of the 2022 Cybernetics and Informatics (K&I)*, Visegrád, Hungary, September 2022.

[9] T. Lechler, E. Fischer, M. Metzner, A. Mayr, and J. Franke, "Virtual Commissioning—scientific review and exploratory use cases in advanced production systems," *Procedia CIRP*, vol. 81, pp. 1125–1130, 2019.

[10] M. Schamp, L. V. D. Ginste, S. Hoedt, A. Claeys, E.-H. Aghezzaf, and J. Cottyn, "Virtual commissioning of industrial control systems—a 3D digital model approach," *Procedia Manufacturing*, vol. 39, pp. 66–73, 2019.

[11] M. Stremy, D. Horvath, D. Vana et al., "RBS channeling MATLAB application for automated measurement control and evaluation for 6MV tandetron accelerator," *Applied Sciences*, vol. 11, no. 9, p. 3817, 2021.

[12] J. Orloff, *Handbook of Charged Particle Optics*, CRC Press, Boca Raton, FL, USA, 2017.

[13] B. Wolf, *Handbook of Ion Sources*, CRC Press, Boca Raton, FL, USA, 1995.

[14] H. Wollnik, *Gaussian Optics and Transfer Matrices*, CRC Press, Boca Raton, FL, USA, 2022.

[15] P. Fabo, S. Sedivy, M. Kuba, A. Buchholcerova, J. Dudak, and G. Gaspar, "PLC based weather station for experimental measurements," in *Proceedings of the 2020 19th International Conference on Mechatronics—Mechatronika (ME)*, Prague, Czech Republic, December 2020.

[16] D. Horvath, "PLCSimAdv simulink library GitLab," 2023, https://gitlab.websupport.sk/dusanhorvath/plcsimadv-simulink-library.