Variable Speed Simulation for Accelerated Industrial Control System Cyber Training

Luke Bradford, Barry Mullins, Stephen Dunlap, Timothy Lacey

Chapter 15

VARIABLE SPEED SIMULATION FOR ACCELERATED INDUSTRIAL CONTROL SYSTEM CYBER TRAINING

Luke Bradford, Barry Mullins, Stephen Dunlap and Timothy Lacey

Abstract Industrial control systems employ a variety of hardware, software and network protocols to control physical processes that are critical to societal functions. It is vital that industrial control system operators receive quality training to defend against cyber attacks. Hands-on training exercises with real-world control systems enable operators to learn defensive techniques and understand the real-world impacts of their control decisions. However, cyber attacks and operator actions have unforeseen effects that can take a significant amount of time to manifest and potentially cause physical harm to systems, making high-fidelity training exercises costly and time-consuming.

This chapter presents a methodology for accelerating training exercises by simulating and predicting the effects of cyber events in partially-simulated control systems. A hardware-in-the-loop simulation comprising a software-modeled water tank and a commercially-available programmable logic controller are used to demonstrate the feasibility of the methodology. The experimental results demonstrate that the effects of cyber events can be accurately simulated at speeds faster than real time, significantly enhancing operator training regimens.

Keywords: Industrial control systems, training, hardware-in-the-loop simulation

1. Introduction

Most critical infrastructure operators lack the training to prevent and properly respond to sophisticated cyber attacks against industrial control systems [1]. Additionally, information security personnel lack an understanding of industrial control systems and cannot predict the impacts of changes to control systems and networks. Adversaries know that cyber attacks launched against industrial control systems have the potential to cause significant physical harm to critical infrastructure assets. The ability to cause physical damage and the lack

of well-trained personnel render industrial control systems attractive targets for adversaries. Exacerbating the lack of cyber-capable control system operators is the fact that most training programs provide instruction at the basic or intermediate knowledge levels [7].

The absence of thorough, advanced training regimens for industrial control system security is primarily due to the lack of robust facilities that provide experience with real-world control system components, physical systems and processes. Consequently, there is an urgent need for training environments that blend real control system components with physical processes to enable trainees to understand the effects of cyber attacks and defensive actions. Since cyber effects often take significant amounts of time to manifest themselves, replicating their impacts in a learning environment becomes infeasible. Additionally, unforeseen consequences of cyber events have the potential to cause catastrophic damage to the equipment used for training. The potential damage to equipment makes high-fidelity training exercises prohibitively expensive. Thus, a critical component of any solution is the ability to quickly model the effects of cyber attacks so that more time can be devoted to analysis and evaluation, which correspond to higher levels of learning [4].

An ideal training environment is a full-scale, real-world facility with several interconnected processes [7]. However, using such an environment for training is expensive and time-consuming. To address these challenges, this chapter proposes a methodology for augmenting industrial control system security training environments by enabling exercise coordinators to rapidly model and predict the effects of cyber events. The methodology engages a hardware-in-the-loop (HiL) simulation and commercially-available programmable logic controllers (PLCs) to increase the speed of a physical process while enabling the programmable logic controllers to operate as intended. The speed-up feature enables trainees to gain valuable expertise and to understand the consequences of their actions while limiting the possibility of physical damage to equipment.

2. Background

This section provides background information about industrial control systems, cyber training environments and hardware-in-the-loop simulation, and discusses related work.

2.1 Industrial Control Systems

The United States describes the critical infrastructure as systems and assets, both physical and virtual, so crucial to the country that their destruction or incapacity would threaten U.S. national security, the economy, power supply, public health, public safety and other areas [5]. Presidential Policy Directive 21 [6] lists sixteen U.S. critical infrastructure sectors, including energy, transportation, water and communications. Assets in all the critical infrastructure sectors engage industrial control systems in order to achieve increased automation, efficiency and maintainability.

Industrial control systems monitor and control industrial processes with the help of sensors, actuators, control units and networks [9]. In an industrial control system, control units receive data from sensors. Based on the data received from sensors, the control units direct actuators to control the processes being monitored by the sensors in order to produce the desired outputs [9]. An industrial control system can be implemented locally, such as within the confines of a factory, or across numerous devices and pieces of equipment over a large geographical area, such as a power grid. Industrial control systems are typically systems of systems that control multiple interconnected, mutually dependent processes that function together to achieve industrial objectives.

2.2 Cyber Training Environments

Plumley et al. [7] have specified various levels of cognitive complexity for control system training environments. Their goal was to create an industrial control system educational framework that could offer training regimens for a range of organizational budgets and needs. The primary determiner of the level of a training environment is the realism it provides in the context of real industrial control systems. The complexity of the training scenarios that can be provided by a training environment depends on the amount of realism provided. The level of cognitive complexity increases as the training environment realism increases [7]. With this in mind, Plumley and colleagues created and mapped four levels of industrial control system training environments to Bloom's Taxonomy [4] in order to create a comprehensive industrial control system training framework. Training environments capable of administering exercises at higher levels of thinking map to higher levels of the taxonomy while providing access to all the lower levels of the taxonomy.

Bloom's Taxonomy, which was created by Benjamin Bloom (1913-1999), classifies educational objectives based on cognitive complexity [4]. The taxonomy is widely used by educators to structure courses that enable students to learn, apply knowledge, think critically and create new ideas. Bloom's Taxonomy was revised in 2001 to comprise six categories of educational goals [7]. The taxonomy progresses from the lowest cognitive level of basic understanding to the highest cognitive level, which is the creation of original ideas. Bloom's Taxonomy offers a means for aligning educational tools to specific levels of cognitive complexity. Bloom emphasized the acquisition of concrete knowledge before increasing the complexity of a training regimen. In other words, it is imperative that trainees master their current levels in the taxonomy before proceeding to higher levels. This is why, in many high-risk or critical fields, training involves several levels of simulation, where the complexity of each level progressively increases until trainees are proficient enough to attempt real tasks using real equipment.

A Level 1 training environment is entirely software defined. It uses software to simulate an industrial controller or control system. Level 1 environments encompass the lowest two levels of Bloom's Taxonomy – remembering and understanding [7].

A Level 2 training environment includes an automated process that creates real physical effects. However, instead of employing the same hardware and software used in industry, it incorporates simple embedded systems (e.g., Arduino and Raspberry Pi devices) that are programmed to monitor and control physical processes using common programming languages (e.g., C and Python). Level 2 environments cover the applying and analyzing levels of Bloom's Taxonomy [7].

A Level 3 environment uses real hardware and software. The hardware and software control a partial industrial control system [7]. For example, a Level 3 environment for training prison guards would enable them to control the locks on a block of prison cell doors. These environments are not full-scale industrial control systems, but they enable trainees to familiarize themselves with real-world equipment, industrial networks and process systems. Since they are not full-scale systems, they cannot provide an understanding of real-world systems, where a malfunction in one process can affect other processes due to system interdependencies. Level 3 environments reach the evaluating level of Bloom's Taxonomy. Trainees can compare observations against standard operational criteria and data. Realistic data enables them to make realistic evaluations that transfer to real-world systems. To maximize realism and minimize cost and space, Level 3 environments employ hardware-in-the-loop simulations of industrial processes that are controlled by real hardware. Hardware-in-the-loop simulations eliminate the need to incorporate large and expensive physical equipment in training environments by replacing them with accurate simulations that interface with real-world industrial control hardware.

A Level 4 environment is a real-world, full-scale industrial control system training facility. The training facility is essentially identical to its real-world counterpart [7]. Level 4 environments reach the highest level of thinking in Bloom's Taxonomy – creating. In the context of industrial control system training, this type of cognitive complexity cannot be achieved without the real system. Trainees have the ability to view and manipulate every component in the actual industrial environment. New methods and solutions can be devised, tested and applied to the real system, and their effects can be observed.

Cyber attacks are often slow moving and are, therefore, difficult to detect. Stuxnet, the most famous malware to target industrial control systems, took months to conduct its attacks [11]. Since cyber attacks often employ the low-and-slow attack paradigm, it can take a long time to witness their effects. Thus, a key component of a realistic industrial control system training environment is a means for speeding up the progress of slow moving attacks and their impacts during training activities.

2.3 Hardware-in-the-Loop Simulation

The industrial control system training environment discussed in this chapter is a Level 3 system that can speed up simulations of physical processes, enabling trainees to quickly detect and observe the progression of cyber events, and predict their effects. It employs a hardware-in-the-loop simulation of an in-

dustrial process that is controlled by an actual programmable logic controller. Hardware-in-the-loop simulation is a common technique used in industry to develop and evaluate complex, real-time, embedded systems [8]. It yields a simulated process under control that creates a realistic test environment for embedded systems. During testing, the embedded system interacts with the process simulation. A key component of hardware-in-the-loop simulation is the electrical emulation of sensors and actuators, which serves as the interface between the simulation and embedded system under test. The process simulation determines the values of the electrically-emulated sensors. These values are read by the embedded system under test as feedback. The control algorithm running on the embedded system under test outputs actuator control signals based on the feedback received. The output control signals produce changes to the variable values in the simulation, including the values measured by the sensors. Thus, hardware-in-the-loop simulation incorporates a complete control loop.

Hardware-in-the-loop simulation is commonly used to test embedded systems because, in many cases, it is more efficient (and safer) than connecting the embedded system directly to a real process. For example, the simulation can enhance the quality of testing by increasing the scope of test scenarios and overcoming the testing limitations imposed by a real process. Using a real process also prevents the embedded system from being tested under failure conditions. Furthermore, the simulation assists in developing embedded systems under tight schedules that do not allow testing to be delayed until a process prototype becomes available. Finally, it is more economical to conduct testing using a high-fidelity, real-time hardware-in-the-loop simulation instead of a real process.

2.4 Related Work

Saco et al. [8] noted that the ideal learning environment for industrial control system operators is a real-world plant, but they acknowledged that such environments would be large, expensive and potentially dangerous to be used by novices. Recognizing the need for high-fidelity simulation tools that could provide effective training regimens, Saco and colleagues proposed a hardware-in-the-loop, real-time simulation system. MATLAB Simulink, a software tool for modeling dynamic systems, was used to model the control algorithm and plant (water tank with an inflow pump and pneumatic drainage valve). The control algorithm was converted to a C program using the Simulink Real Time Workshop software. The C code was downloaded to real-time prototyping hardware (dSpace 1102 floating-point controller board) and executed independently of MATLAB. After the C code was downloaded to the board and executed, the controller hardware was able to control the water level in the tank simulation. The functionality provided by Simulink enabled trainees to implement and refine their own (similar) hardware-in-the-loop systems to gain better understanding of the physical system and control principles.

Thornton and Morris [10] state that the ideal environment for studying cyber attacks against industrial control systems is a real system with real hardware, software and communications technologies. Since such environments are prohibitively expensive, Thornton and Morris developed a virtual laboratory testbed that was mobile, sharable and expandable. The testbed incorporated a Simulink gas pipeline simulation with sensors and actuators, a virtual programmable logic controller simulated with Python, and a human-machine interface (HMI). The Simulink gas pipeline and the virtual logic controller communicated via JavaScript Object Notation (JSON) attribute-value pairs contained in user datagram protocol (UDP) packets. The virtual logic controller communicated with other devices such as the human-machine interface and physical programmable logic controllers via Modbus/TCP, a standard industrial control system protocol. Communications between the virtual logic controller and physical logic controllers enabled the virtual laboratory testbed to produce accurate control system network traffic for analysis.

Unfortunately, the two environments described above do not incorporate real-world control system hardware. Incorporating industrial control system hardware brings simulated environments much closer to real industrial systems, especially when attempting to understand normal operations and attack scenarios. Moreover, the two environments cannot speed up operations to quickly model the effects of attacks and provide more time for operator training and analysis. For example, without a speedup capability, low and slow attacks such as Stuxnet cannot be replicated quickly enough to observe their effects and learn from them in a reasonable timeframe.

3. Methodology

This section describes the proposed methodology, including the test systems and experimental design.

3.1 Test Systems

The test environment incorporated two systems that implemented a water tank control loop. A Lab-Volt 3531 training system was used as the baseline, real-world control system [3]. A simulated water tank system replicated the operation of the Lab-Volt training system. The Lab-Volt system and the simulated water tank incorporated the control loop shown in Figure 1. The programmable logic controller in each control loop maintained the water level in the tank at a user-defined set point. It polled the water level sensor to obtain the current water level in the tank. Based on the current water level, the controller sent a command to the drainage valve to increase or decrease the outflow rate.

Lab-Volt 3531 Training System. Figure 2 shows the Lab-Volt 3531 system components. An Allen-Bradley ControlLogix 1756-L55 programmable logic controller that was programmed using RSLogix 5000 from Rockwell Au-

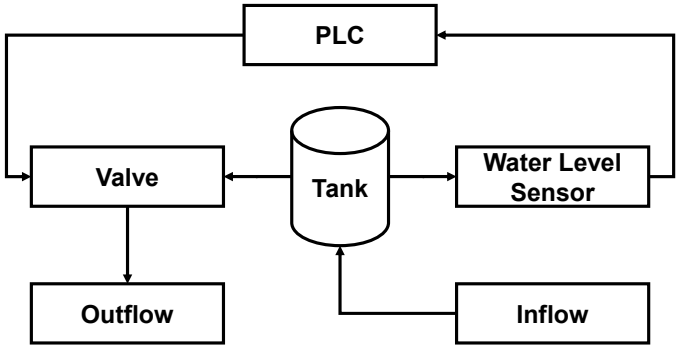

Figure 1. Test system control loop.

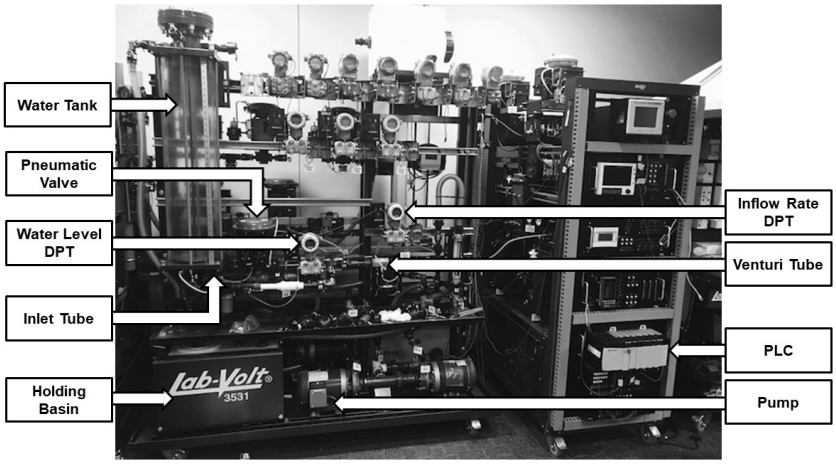

Figure 2. Lab-Volt 3531 training system.

tomation served as the primary control unit. The inflow rate of water into the cylindrical tank (8" diameter and 36" height) was controlled by an alternating current pump driven by an Allen-Bradley PowerFlex 40 variable frequency drive (VFD). A differential pressure transmitter (DPT) continuously monitored the inflow rate using a Venturi tube, which created a differential pressure proportional to the rate of flow through the tube. Taps at the high pressure and low pressure portions of the tube were connected to the differential pressure transmitter, which measured the pressure difference and computed the flow rate. The differential pressure transmitter sent the flow rate value to the programmable logic controller as a 4-20 mA analog signal. The programmable logic controller employed a proportional-integral-derivative (PID) control strat-

egy to determine the appropriate pump speed in order to ensure that the water inflow rate remained constant at the set point. The computed pump speed was transmitted to the variable frequency drive as an analog value. An Allen-Bradley PanelView Plus 600 human-machine interface was used to monitor and configure the system.

A globe type valve located at the bottom of the tank enabled water to exit the tank and drain into the holding basin. The valve closed partially or completely based on the analog signal it received from the programmable logic controller. The valve was pneumatically operated and equipped with a spring-and-diaphragm actuator. A plug in the valve restricted the flow of water into the tank outlet. The plug was designed to have a fixed linear relationship between the distance traveled by the valve stem and the amount of flow allowed through the valve. When the valve received an analog signal from the logic controller, the current-to-pressure converter of the valve linearly transformed the analog signal to a pneumatic pressure, which was applied to the surface of the valve diaphragm, producing a force that overcame the spring force and moved the plug up or down. The plug restricted the flow of water through the valve from 0% to 100%. The percentage of flow allowed through the valve is referred to as the valve position.

A second differential pressure transmitter measured the pressure in the bottom tank to compute its water level. The water level value was transmitted to the programmable logic controller as a 4-20 mA analog signal. The programmable logic controller used a PID control strategy to determine the valve position based on the water level value supplied by the differential pressure transmitter. The controller transmitted the desired valve position to the valve as a 4-20 mA analog signal. The PID control strategy maintained the water level in the tank with minimal overshoot, undershoot and set point deviation.

Simulated Water Tank. The process simulator experiment employed a hardware-in-the-loop simulation of the physical process and a ControlLogix 1756-L55 programmable logic controller. Figure 3 shows the simulated and real components of the test system. The hardware-in-the-loop simulation enabled the system to use real industrial control system hardware without having to incorporate physical equipment such as pumps and tanks. The simulated physical process managed by the programmable logic controller mirrored the Lab-Volt water tank system. A pump filled the tank with water at a constant inflow rate and a drainage valve at the bottom of the tank allowed water to exit. The simulation accurately replicated the behavior of the Lab-Volt system.

The simulated water tank was implemented as a MATLAB Simulink model. Simulink is a graphical programming environment that can model a variety of systems by selecting blocks from various block libraries and connecting them via input/output (I/O) arrows. Each block includes customizable features; some blocks enable users to write custom code. Simulink simplified the modeling process because it eliminated the need to write code for complex mathematical functions. Its graphical environment assisted in visualizing the system.

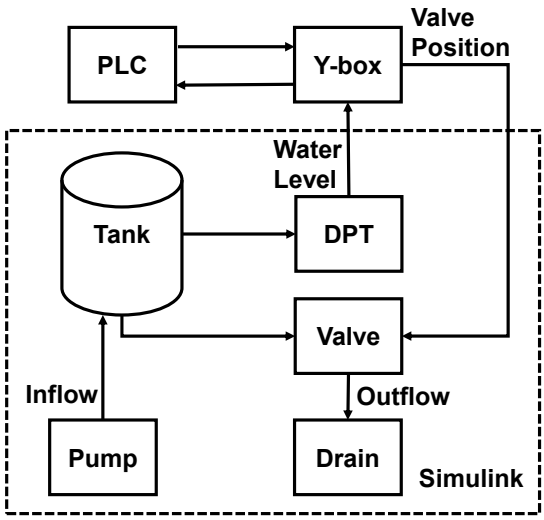

Figure 3. HiL simulation of the Lab-Volt 3531 system.

A MATLAB function block was developed to capture the water tank dynamics. The function block implemented Equations (1) through (5) below that define the dynamics and physical characteristics of the water tank.

The cross-sectional area of the water tank A (in^2) is computed as:

$$A = \pi r^2 \tag{1}$$

where r is the radius of the tank in inches.

The height (level) of water in the tank H in inches is computed as:

$$H = \frac{Vol}{A} \tag{2}$$

where Vol is the volume of water in the tank (in^3).

The inflow rate Q_{in} (in^3/min) is computed as:

$$Q_{in} = 588.9 \tag{3}$$

Note that Q_{in} is a user-provided value that could be changed at any point in time during the simulation; it was set to $588.9\,\text{in}^3/\text{min}$.

The outflow rate Q_{out} (in^3/min) is computed as:

$$Q_{out} = V_P \cdot V_C \cdot \sqrt{H\ (\text{psi})} \tag{4}$$

where V_P is the position of the drainage valve that ranges from zero to one ($V_P = 0.5$ corresponds to the valve being half closed); and V_C is the dimensionless valve constant/flow coefficient ($V_C = 6$). Note that Q_{out} increases as the

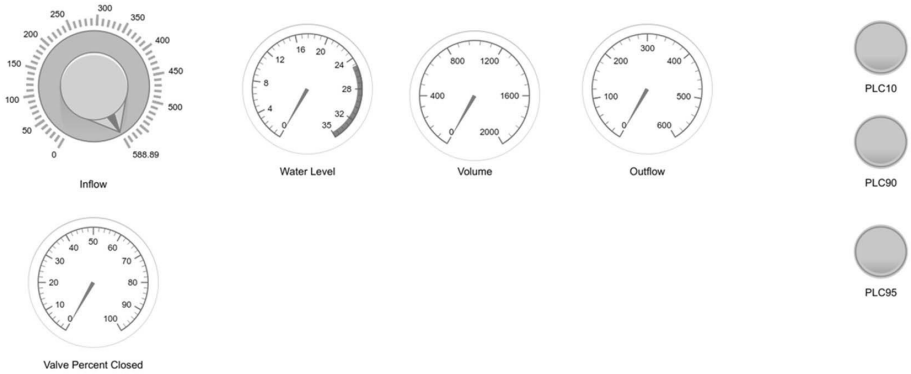

Figure 4. Simulink water tank graphical user interface.

height (level) of water in the tank increases. This reflects the effect of water pressure on the outflow rate.

The rate of change of water volume ΔVol (in^3/min) in the tank with respect to time is computed as:

$$\Delta Vol = Q_{in} - Q_{out} \tag{5}$$

The signal generated by the ΔVol output was fed to an integrator block, which calculated the integral of the derivative with respect to time in order to compute the volume Vol; initially $Vol = 0$, meaning that the tank was empty. This was reflected as an initial condition in the configuration options of the integrator block.

The Simulink Dashboard library supports the rapid development of graphical user interfaces (GUIs) for monitoring and controlling simulated processes. Figure 4 shows the graphical user interface for the simulated water tank. The interface includes a knob block for setting the inflow rate and gauge blocks for monitoring the water level, volume, current drainage valve configuration and outflow rate. The interface also has three light blocks that indicate if the current water level in the tank is at a critical level. By default, the lights shine green. When the water level reaches a critical height, the corresponding light shines red. The three lights correspond to whether the tank is less than 10% full, greater than 90% full and greater than 95% full, respectively.

Figure 5 shows a speedup block from the Real-Time Pacer library, which enabled the simulation speed to be increased or decreased as desired. The speedup block was configured by entering a number that represented the speedup factor for the simulation. For example, a speedup factor of two doubled the speed of the simulation.

A MATLAB function block reported the current water level in the tank to the programmable logic controller every 0.1 seconds. In order to maintain this reporting rate, the sample time t_S of the MATLAB function block was adjusted according to the simulation speedup factor f. When the simulation

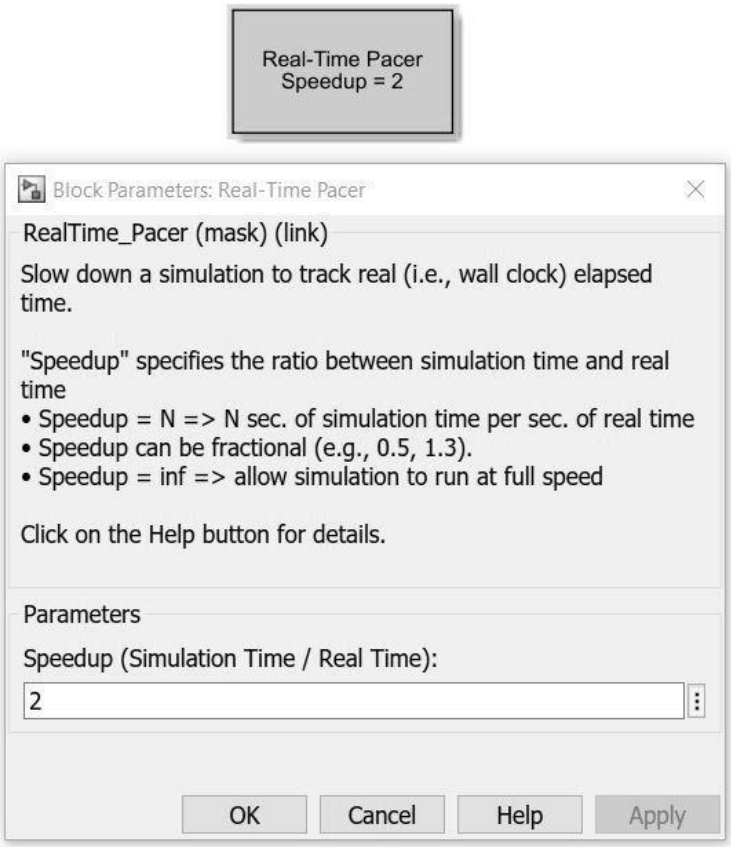

Figure 5. Simulink speedup block.

Figure 6. Setting the sample time.

was executed in real time, the sample time was set to 0.1 seconds as shown in Figure 6. A speedup factor of two required the sample time to be set to 0.2 seconds and a speedup factor of ten required the sample time to be set to 1 second. In general, the sample time t_S is computed as:

$$t_S = \frac{f}{10} \tag{6}$$

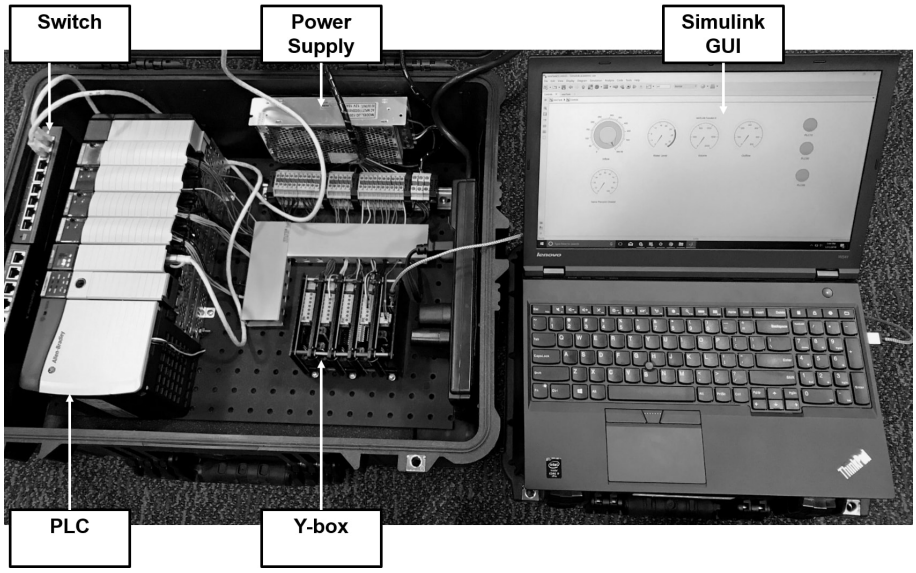

Figure 7. Simulated water tank.

A Y-box served as the interface between the simulated water tank and programmable logic controller by providing an electrical emulation of the sensors and actuators. The Y-box, which was created to support the development of hardware-in-the-loop simulations, received currents and voltages as inputs and generated currents and voltages as outputs based on commands received over a universal serial bus (USB) port [2]. The inputs and outputs enabled the Y-box to interface with a programmable logic controller in the same manner as regular sensors and actuators. The simulation and the Y-box communicated via a serial link. The test system utilized the same model programmable logic controller and ladder logic used to control the Lab-Volt system. Note that the ladder logic used for the simulation did not include the PID controller that controlled the inflow rate.

In the Simulink model, the inflow rate was set as a constant parameter. Minimal changes were made to the ladder logic to enable the programmable logic controller to operate in the simulated environment. The project path was updated with the IP address of the programmable logic controller used in the simulated system. Also, the module numbers were updated to match the hardware present in the programmable logic controller used to control the simulated system. The code section containing the PID controller that adjusted the inflow rate was not configured to activate because it was not used by the Simulink model. Figure 7 shows the setup of the simulated system.

Table 1. Run names.

Name	Meaning
LVTn	Lab-Volt 3531 Trial n
Simx1Tn	HiL System Real-Time Trial n
Simx2Tn	HiL System 2 × Real-Time Trial n
Simx10Tn	HiL System 10 × Real-Time Trial n

3.2 Experimental Design

Validation testing was performed to verify the accuracy and consistency of the simulation. The experiments compared the operation of the simulation against the operation of an actual water tank, specifically, against the Lab-Volt training system. The experiments sought to demonstrate that the simulation reflected the normal operation of a real water tank when executed at real time and at faster simulation rates. Previous pilot studies showed that the Lab-Volt system was consistent from run to run. Specifically, the standard deviation of the average differences between Lab-Volt runs was less than 0.01%. Thus, water level recordings from only three Lab-Volt runs were used in the experiments.

The water tank simulation was also executed three times at each speed specified in Table 1 for a total of nine runs; Table 1 shows the names of the runs executed in the experiments. The water level recordings from the three Lab-Volt runs were compared against the water level recordings from the nine simulated water tank runs. In order to ensure accurate comparisons, all the Lab-Volt and simulation runs followed the procedure shown in Figure 8, which depicts a typical Lab-Volt run in the experiments.

Each run in the experiments comprised the following steps:

- The pump was started, which caused the water level in the tank to rise.
- A Python script was executed to read and write tags in the programmable logic controller ladder logic. The script set the water level set point to 30% full and began to record the current water level and the time at half-second intervals starting at zero seconds. Water levels for the simulation and Lab-Volt runs were measured as percentages, where 0% corresponded to an empty tank and 100% corresponded to a full tank.
- As the water level in the tank approached 30%, the Python script monitored the rising water level to detect the water level steady state at 30%. In the experiments, steady state was considered to be reached when the water level reached the set point and remained at the set point with minimal deviations (less than ±0.5% from the set point). A well-tuned PID controller ensured that the process variable reached the set point with minimal overshoot (less than 3%). The PID controller then corrected the overshoot and the process variable remained close to the set point

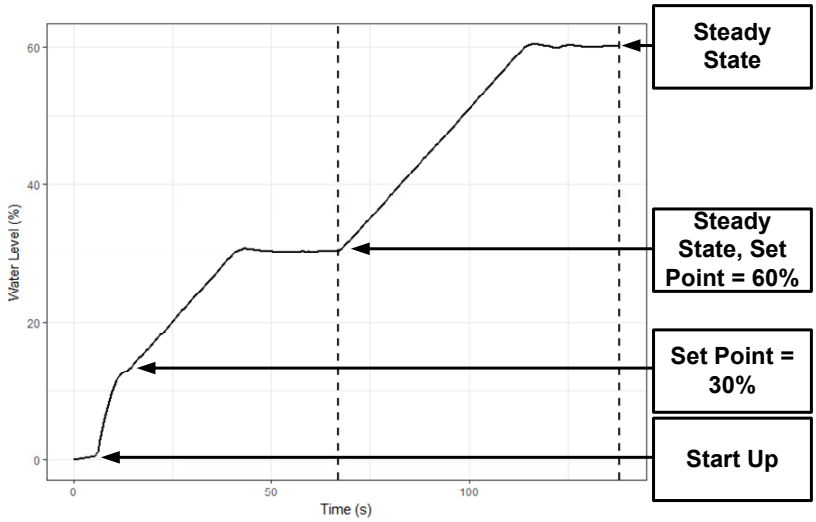

Figure 8. Experimental procedure.

with minimal deviations. The Python script detected the steady state by checking if the current water level was within ±0.1% of the set point. After meeting this threshold, the Python script waited for 25 seconds to enable the PID controller to correct any initial overshoot. After this time, the Python script sent a confirmation that the steady state was reached. Pilot studies demonstrated that a wait time of 25 seconds was adequate for the PID controllers in the simulation and Lab-Volt system to correct initial overshoots and achieve the water level steady state.

- After the script detected the steady state at 30%, it changed the water level set point to 60%. The water level in the tank began to rise and the script waited to detect when the water level was steady at 60%.

- After the script detected that the water level had reached steady state at 60%, it stopped collecting data and ended the run.

Each run in the experiments yielded a curve with water level on the y-axis and time on the x-axis. In order to ensure consistent analysis, the experiments considered an interval of time in which all three Lab-Volt runs achieved steady state at 30%, rose to 60% and achieved steady state at 60%.

After the three Lab-Volt runs were completed and the raw data was collected, another Python script adjusted the three curves so that they reached 45% at the same time (designated as zero seconds). In the first Lab-Volt curve, the Python script used linear interpolation to compute the time at which the curve reached 45% and subtracted this time from all other times in the curve. The script repeated this process for the other two Lab-Volt curves. These adjustments

shifted the curves so that that they centered at zero seconds with a height of 45%, the midpoint between the 30% and 60% set points.

After the three curves were adjusted, an 80-second time interval was selected. During this time interval, the three Lab-Volt runs completed the required behavior of achieving steady state at 30%, rising to 60% and achieving steady state at 60%. Before the curves from different runs were compared, all the curves were adjusted so that they reached 45% at zero seconds. Subsequent analysis compared only the portions of the curves within the 80-second time interval to ensure consistent comparisons. The consistency enabled accurate comparisons of each run against every other run in the experiments.

Table 2 shows the matrix used to compare all the runs. Note that the simulation run names are shortened in the matrix. Simx1Tn is denoted as Sx1Tn, Simx2Tn is denoted as Sx2Tn and Simx10Tn is denoted as Sx10Tn.

The experiments compared the curves from the trials by considering the average difference between them. The average difference corresponded to the average distance between the water level for the first curve and the water level for the second curve at each point in time. In order to compute the average difference between two curves, a Python script first adjusted the timestamps for each curve to account for speedup. For example, when the Sx1T1 run was being compared against the Sx10T1 run, all the timestamps in the Sx10T1 run were multiplied by ten to account for the speedup in the Sx10T1 run. The matrix cell represented by this comparison has row Sx10T1 and column Sx1T1.

Next, the Python script adjusted both curves so that they reached 45% at the same time, which was designated as zero seconds. In the case of the Sx1T1 curve, the Python script used linear interpolation to compute the time when the curve reached 45% and subtracted this time from all the other times in the curve. The script repeated this process for the Sx10T1 curve. These adjustments shifted both curves so that that they centered at zero seconds for a water level of 45%, the midpoint between the 30% and 60% set points.

At this point, the script removed all the portions from the two curves that were not included in the 80-second time interval. Then, the script iterated through the remaining timestamps for the Sx1T1 curve and employed linear interpolation to determine the corresponding heights in the Sx10T1 curve. At this point, the Python script had three curves – the Sx1T1 curve, the Sx10T1 curve and the new Sx10T1 curve containing only the timestamps from the Sx1T1 curve and their associated water levels computed via linear interpolation. The matching timestamps enabled the Sx1T1 and Sx10T1 runs to be compared in a straightforward manner.

Finally, the Python script iterated through the Sx1T1 curve and the new Sx10T1 curve. For each timestamp, the script calculated the absolute value of the difference between the water level in the Sx1T1 curve and the water level in the new Sx10T1 curve. After iterating through both curves, the Python script added all the absolute values, divided the sum by the number of timestamps and returned the quotient as the average difference between the two runs.

Table 2. Test matrix.

	LVT1	LVT2	LVT3	Sx1T1	Sx1T2	Sx1T3	Sx2T1	Sx2T2	Sx2T3	Sx10T1	Sx10T2	Sx10T3
LVT1												
LVT2	X											
LVT3	X	X										
Sx1T1	X	X	X									
Sx1T2	X	X	X	X								
Sx1T3	X	X	X	X	X							
Sx2T1	X	X	X	X	X	X						
Sx2T2	X	X	X	X	X	X	X					
Sx2T3	X	X	X	X	X	X	X	X				
Sx10T1	X	X	X	X	X	X	X	X	X			
Sx10T2	X	X	X	X	X	X	X	X	X	X		
Sx10T3	X	X	X	X	X	X	X	X	X	X	X	

Two evaluation metrics were employed in the experiments. The first metric measured whether or not the simulation completed the required behavior of achieving the water level steady state at 30%, raising the water level to 60% and achieving the water level steady state at 60% within the 80-second time interval. If a simulation run completed the required behavior within the time interval, then the run passed the first metric; otherwise, it failed the metric.

The second metric considered the average difference between two runs. The experiments used the mean of the average differences between the three Lab-Volt runs as the threshold for the second evaluation metric. A high-fidelity simulation was expected to have a similar average difference when compared to the Lab-Volt system. If the average difference between two runs was less than or equal to the mean of the average differences between the Lab-Volt runs, then the runs passed the second metric; otherwise, they failed the metric.

4. Experimental Results

This section describes the experimental results.

4.1 Metric 1 (Required Behavior)

All the simulation runs completed the required behavior within the 80-second time interval. Regardless of the simulation speed, all the simulation runs passed the first evaluation metric. In order to verify that each run passed the first metric, all the runs were graphed. For each graph, the portion of the graph containing the 80-second time interval was visually inspected to ensure that the simulation run achieved the required behavior.

Figures 9, 10 and 11 demonstrate the accuracy of the simulation in real-time, two times the speed and ten times the speed, respectively. Note that the time scales for the Simx2 and Simx10 graphs were multiplied by their respective speedup factors to match real time. Each of the three graphs represents a typical comparison of the simulation against the Lab-Volt system. The slopes of the Lab-Volt run and the Simx1, Simx2 and Simx10 runs are almost identical. The slight differences in the slopes is most likely due to variations of the inflow rate in the Lab-Volt system. As mentioned above, the PID controller in the Lab-Volt system that maintained a constant inflow rate produced minor fluctuations in the inflow rate whenever it was forced to adjust the speed of the pump. Another possible cause for the slight differences in the slopes is the trial-and-error method used to tune the PID controller in the simulation. The main differences between the graphs occur as they approach 60% steady state. The differences are due to a control delay in the Lab-Volt system, corresponding to the amount of time between the programmable logic controller sending a command to the valve and the valve adjusting the plug to the correct position. The simulation did not model this control delay and was, therefore, not affected by the delay.

Figure 12 shows how well the simulation runs with speedup factors of two and ten match the simulation run executed at real time. Note that the time

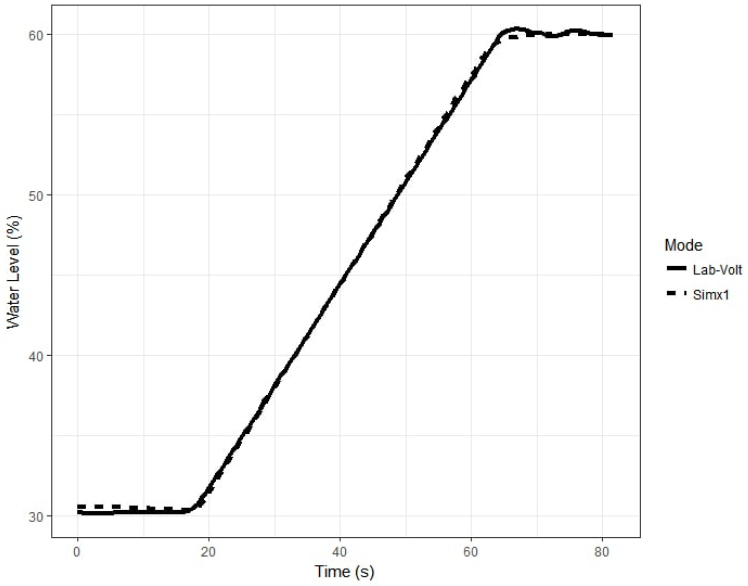

Figure 9. Lab-Volt vs. Simx1.

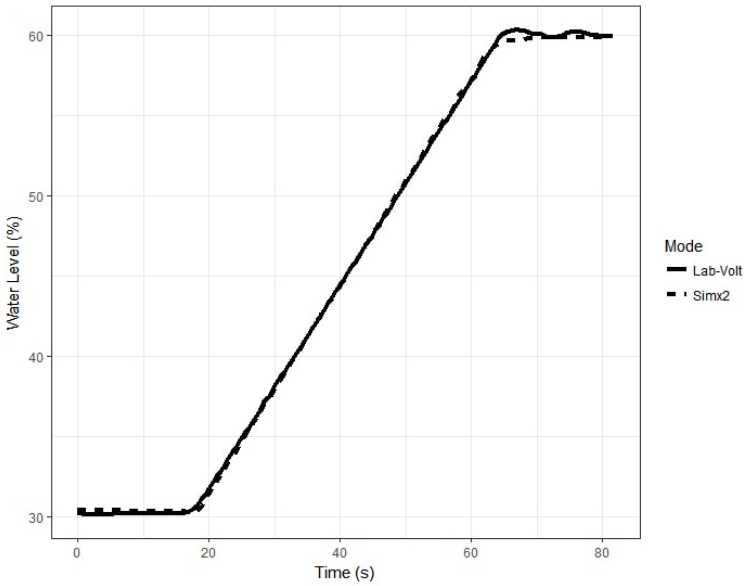

Figure 10. Lab-Volt vs. Simx2.

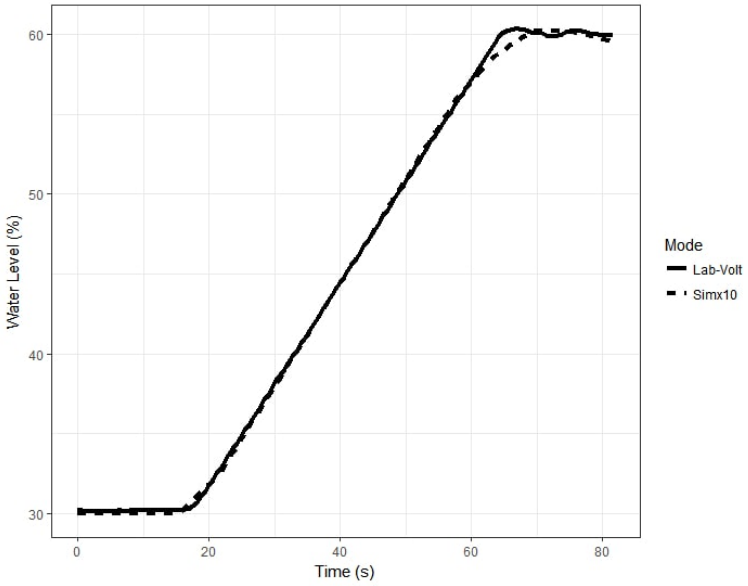

Figure 11. Lab-Volt vs. Simx10.

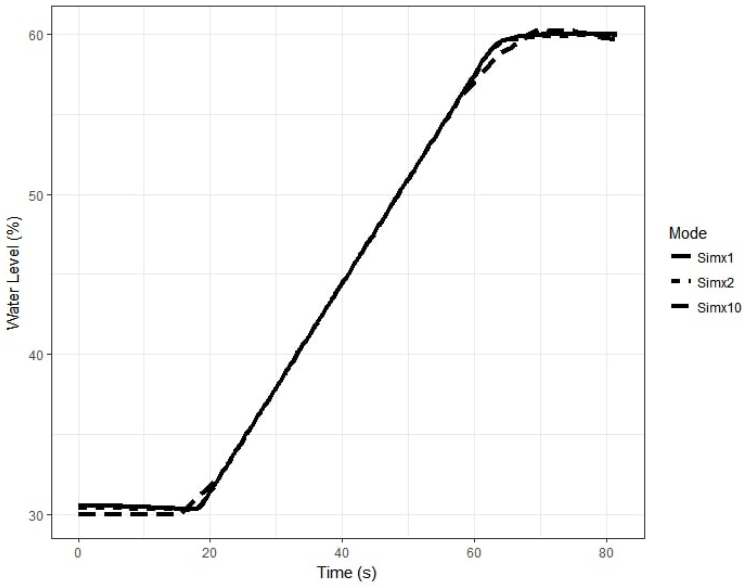

Figure 12. Simx1 vs. Simx2 vs. Simx10.

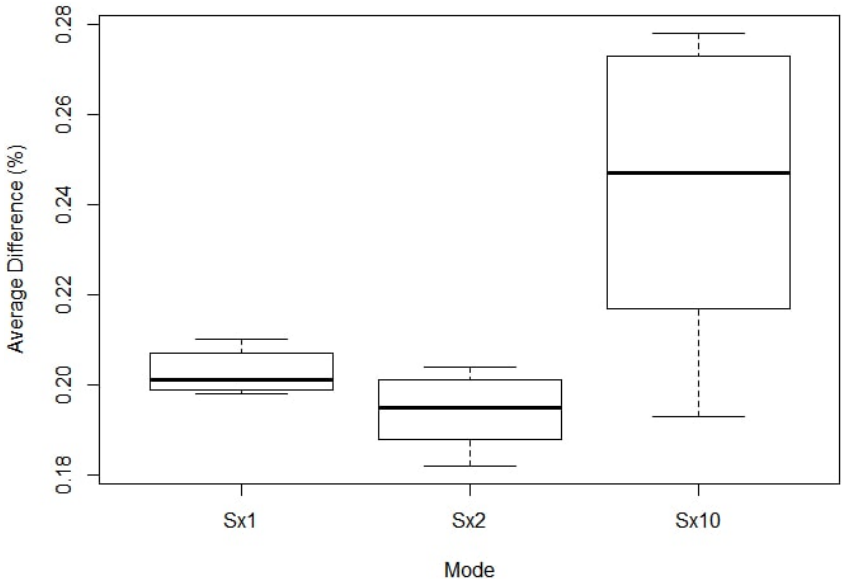

Figure 13. Lab-Volt vs. simulation.

scales for the Simx2 and Simx10 graphs were multiplied by their respective speedup factors to match real time. The primary differences between the Simx1, Simx2 and Simx10 graphs occur as they approach the 60% steady state. The differences most likely result from the trial-and-error method used to tune the PID controller in the simulation.

Table 3. Simulation accuracy – average differences (%).

Comparison	Mean	Min	Max	StDev
Lab-Volt vs. Lab-Volt	0.055	0.045	0.061	0.009
Lab-Volt vs. Simx1	0.203	0.198	0.210	0.005
Lab-Volt vs. Simx2	0.194	0.182	0.204	0.007
Lab-Volt vs. Simx10	0.245	0.193	0.278	0.031

4.2 Metric 2 (Average Difference)

Figure 13 and Table 3 summarize the average differences between the Lab-Volt system and the simulation.

The average difference when comparing the Lab-Volt runs is well below 0.1%. When executed at real time, the average difference between the Lab-Volt and simulation runs ranges from 0.198% to 0.21%. When executed at two times

Table 4. Run consistency – average differences (%).

	Mean	**Min**	**Max**	**StDev**
Lab-Volt	0.055	0.045	0.061	0.009
Simx1	0.021	0.014	0.026	0.006
Simx2	0.037	0.031	0.048	0.009
Simx10	0.102	0.045	0.131	0.049

faster than real time, the average difference between the Lab-Volt and simulation runs ranges from 0.182% to 0.204%. When executed at ten times faster than real time, the average differences between the Lab-Volt and simulation runs ranges from 0.193% to 0.278%. The mean of the average differences for all the comparisons between the simulation and the real water tank is 0.214%, which is well above 0.055%, the mean of the average differences between the Lab-Volt runs. These results demonstrate that the average difference between the simulation and Lab-Volt runs is significantly greater than the average difference between the Lab-Volt runs regardless of the simulation speed. Thus, the proposed simulation does not pass the second evaluation metric.

Although the simulation does not pass the second evaluation metric, all the simulation results are relatively consistent as shown in Figure 13. In fact, all the simulation runs produce relatively low average differences ranging from 0.182% to 0.278%. The mean of the average differences for the Simx2 runs is slightly lower than the mean for the Simx1 runs.

A permutation test comparing the average differences from the Simx1 and Simx2 runs produced a p-value of 0.01354, which is greater than the 0.01 threshold for the 99% confidence level. Thus, the permutation test showed that no significant difference exists between the mean of the average differences for the Simx2 runs and the mean of the average differences for the Simx1 runs at the 99% confidence level. The null hypothesis that the two means are equal cannot be rejected because the p-value is greater than 0.01. This implies that the simulation accuracy is the same when the simulation is run at real time and at two times real time. Overall, the experimental results demonstrate that the proposed simulation consistently models the Lab-Volt system with less than $\pm 0.28\%$ error on average at any point in time. The average differences between the simulation and the real water tank are consistently low with a standard deviation of 0.028%, even when the simulations were executed at speeds much faster than real time.

4.3 Consistency

Table 4 demonstrates the consistency of multiple runs of the Lab-Volt system and the simulation. The first row of the table represents the consistency of the Lab-Volt system from run to run. The mean of the average differences between Lab-Volt runs is 0.055%, the minimum average difference is 0.045% and the

Table 5. Effect of speedup – average differences (%).

Comparison	Mean	Min	Max	StDev
Simx1 vs. Simx1	0.021	0.014	0.026	0.006
Simx1 vs. Lab-Volt	0.203	0.198	0.210	0.005
Simx1 vs. Simx2	0.070	0.056	0.081	0.008
Simx1 vs. Simx10	0.256	0.220	0.280	0.025

maximum average difference is 0.061%. The standard deviation for the Lab-Volt runs is 0.009%. The other rows in the table demonstrate the consistency of the simulation from run to run at each speed. When executed at real time, the average differences between the simulation runs are more consistent than the average differences between the Lab-Volt runs. The consistency in the average differences decreases as the simulation speed increases. The table shows that there is much less consistency from run to run when the simulation was executed with a speedup factor of ten. This decrease in consistency may be due to the reduced precision of the linear interpolation technique used to compute the curves for the Simx10 runs. Since the simulations generated fewer points when they were executed at higher speeds, there were fewer reference points for the linear interpolation computations. Consequently, the Simx10 curves have higher variability.

4.4 Simulation Speedup

Table 5 summarizes the average differences between: (i) simulation runs executed at real time; (ii) simulation runs executed at real time versus the Lab-Volt system; (iii) simulation runs executed at real time versus simulation runs executed with a speedup factor of two; and (iv) simulation runs executed at real time versus simulation runs executed with a speedup factor of ten.

The first row in Table 5 shows that the simulation yields consistent results from run to run when executed at real time. The second and third rows demonstrate that, when executed at real time, the simulation is consistent with the Lab-Volt system and the simulation executed with a speedup factor of two, respectively. The fourth row shows that the average difference between the simulation executed at real-time is consistent with the simulation executed with a speedup factor of ten. The average differences between the simulation executed at real time and the simulation executed with a speedup factor of two are much lower than the average differences between the simulation executed at real time and the simulation executed with a speedup factor of ten. As mentioned above, this difference is due to the imprecise tuning of the PID controller in the simulation, which has a greater effect when the simulation is executed with a speedup factor of ten.

5. Conclusions

The proposed methodology has been designed to accelerate the pace of industrial control system training exercises. The methodology incorporates commercial programmable logic controllers to enable trainees to work with real control system components. A hardware-in-the-loop simulation is employed to speed up the simulated physical process, providing trainees with an understanding of the impacts of their control actions during normal operations, attacks and other cyber events. While consistency and accuracy are lost at high simulation speeds, the experimental results reveal that the physical process responds appropriately. In each test case, when a trainee changed the set point, the programmable logic controller responded to the change and adjusted the water level in the tank. Simulation runs were performed at real time, twice as fast as real time and ten times as fast as real time. As expected, the fidelity of the accelerated simulation runs depends on the accuracy and fidelity of the physical process model. Nevertheless, the environment incorporating a simulated system in conjunction with a full-scale industrial control system enables trainees to gain operational expertise efficiently, safely and at reduced cost.

While this research has demonstrated the feasibility of the simulation speedup methodology, additional work is required to decrease the negative impacts induced by increasing the simulation speed. The simulation should also be tested to determine how well it copes with diverse operating conditions and models the effects of attacks and other cyber events. Additionally, refinements should be made to improve the consistency and accuracy of the simulation. Other future improvements include: (i) devising a universal approach for tuning PID controller parameters; (ii) incorporating additional interconnected processes and components; (iii) testing the upper bound of the speedup factor to determine how fast the proposed methodology can accurately and consistently accelerate cyber events; and (iv) formulating an approach for seamlessly and concurrently transferring cyber events from full-scale, real-world testbeds such as Level 4 environments to the simulation system.

Note that the views expressed in this chapter are those of the authors and do not reflect the official policy or position of the U.S. Air Force, U.S. Department of Defense or U.S. Government.

Acknowledgement

This research was partially supported by the U.S. Department of Homeland Security Industrial Control Systems Cyber Emergency Response Team (ICS-CERT).

References

[1] J. Butts and M. Glover, How industrial control system security training is falling short, in *Critical Infrastructure Protection IX*, M. Rice and S. Shenoi (Eds.), Springer, Cham, Switzerland, pp. 135–149, 2015.

[2] A. Chaves, M. Rice, S. Dunlap and J. Pecarina, Improving the cyber resilience of industrial control systems, *International Journal of Critical Infrastructure Protection*, vol. 17, pp. 30–48, 2017.

[3] Festo Didactic, Familiarization with the Training System: Pressure, Flow and Level, User Guide 86004-E0, Quebec, Canada, 2016.

[4] M. Forehand, Bloom's Taxonomy, in *Emerging Perspectives on Learning, Teaching and Technology*, M. Orey (Ed.), Global Text Project, University of Georgia, Athens, Georgia (`textbookequity.org/Textbooks/Orey_Emergin_Perspectives_Learning.pdf`), pp. 41–47, 2010.

[5] J. Moteff and P. Parfomak, Critical Infrastructure and Key Assets: Definition and Identification, CRS Report for Congress, RL32631, Congressional Research Service, Washington, DC, 2004.

[6] B. Obama, Presidential Policy Directive 21: Critical Infrastructure Security and Resilience, The White House, Washington, DC, 2013.

[7] E. Plumley, M. Rice, S. Dunlap and J. Pecarina, Categorization of cyber training environments for industrial control systems, in *Critical Infrastructure Protection XI*, M. Rice and S. Shenoi (Eds.), Springer, Cham, Switzerland, pp. 243–271, 2017.

[8] R. Saco, E. Pires and C. Godfrid, Real-time controlled laboratory plant for control education, in *Proceedings of the Thirty-Second Annual Conference on Frontiers in Education*, pp. T2D-12–T2D-16, 2002.

[9] K. Stouffer, J. Falco and K. Scarfone, Guide to Industrial Control Systems (ICS) Security, NIST Special Publication 800-82, National Institute of Standards and Technology, Gaithersburg, Maryland, 2011.

[10] Z. Thornton and T. Morris, Enhancing a virtual SCADA laboratory using Simulink, in *Critical Infrastructure Protection IX*, M. Rice and S. Shenoi (Eds.), Springer, Cham, Switzerland, pp. 119–133, 2015.

[11] K. Zetter, An unprecedented look at Stuxnet, the world's first digital weapon, *Wired*, November 3, 2014.