Board 108: Low-Cost Hardware-in-the-Loop Real-Time Simulation Platform

Aaron Fan, New Jersey Institute of Technology
Milad Shojaee, New Jersey Institute of Technology

MILAD SHOJAEE (Graduate Student Member, IEEE) received the B.S. degree in electrical engineering from the Hamedan University of Technology, Hamedan, Iran, in 2012, and the M.S. degree in electrical engineering from the University of Tehran, Tehran, Iran, in 2016. He is currently pursuing the Ph.D. degree with the Department of Electrical and Computer Engineering, New Jersey Institute of Technology, Newark, NJ, USA. His research interests include modeling, robust control, decentralized control, fault diagnosis, renewable energies, and machine learning.

Dr. Mohsen Azizi, New Jersey Institute of Technology

Mohsen Azizi is an assistant professor in the School of Applied Engineering and Technology at New Jersey Institute of Technology (NJIT). He received the M.Sc. and Ph.D. degrees in electrical and computer engineering from Concordia University, Montreal, Canada, in 2005 and 2010, respectively. From 2010 to 2013, he was an R&D engineer at Aviya Tech Inc. and Pratt & Whitney Canada Inc., Longueuil, Canada, where he designed and developed control and fault diagnosis systems for jet engines. His research has been focused on decentralized control and fault diagnosis techniques in microgrids, renewable energy systems, mechatronics, and aerospace.

Low-Cost Hardware-in-the-Loop Real-Time Simulation Platform

Abstract

In this paper, the design and development of a low-cost laboratory-scale hardware-in-the-loop (HIL) simulation platform for power systems is presented by employing a Raspberry Pi and three Arduino UNOs. HIL simulations are vital in system design and prototyping and offer a safe and efficient method to test hardware and software systems within a simulated operational context. The proposed platform leverages a Raspberry Pi to emulate the dynamic model of a three-area power generation system, with the three Arduino UNOs performing as three individual local controllers. This cost-effective approach minimizes the need for physical prototypes, leading to substantial cost savings and enhanced scalability. The platform functions as an educational tool for understanding closed-loop control systems, which eliminates the necessity for costly industrial hardware. The implemented three-area power generation system includes synchronous generators, in which the impacts of renewable energy sources and energy storage systems are considered as disturbances. Initially, the closed-loop power system is modeled and tested using MATLAB as a benchmark, and then simulated and implemented on the developed low-cost HIL platform. The HIL simulation results exhibited negligible deviations from the anticipated MATLAB outcomes, which suggest the platform's potential to be used in other industrial applications including but not limited to aerospace, automotive, and mechatronics systems in future investigations.

Introduction

HIL simulation is a method of testing and debugging hardware and embedded software systems by simulating the environment in which they are expected to operate. This type of simulation has been extremely popular as it allows engineers to test their products accurately and efficiently, without having to build all parts of the physical prototypes. In addition, HIL simulations can be used across a wide variety of fields including but not limited to power systems, aerospace, and automotive systems. There are many advantages to using HIL for the system design and prototype, including safety, cost savings, and scalability [1].

HIL simulations are typically performed by combining and connecting the mathematical model of a system with real hardware components. The models created can include any combination of software components such as the dynamic models of industrial systems, controllers, embedded processors, sensors, actuators, etc., which are connected through networks and integrated into the simulation environment. This environment is then used to accurately simulate and represent the characteristics of the overall physical system under various operating conditions, which provides engineers with useful data on the system performance and behavior [2].

To create accurate HIL closed-loop control system simulations, engineers first develop detailed mathematical models encompassing every aspect of the dynamic system that could affect its behavior. This includes technical considerations such as sensor noise and power consumption levels, all of which must be taken into account when constructing an accurate simulation environment. Additionally, engineers create proper control algorithms, which can help optimize the performance of the system under different feasible scenarios, while still maintaining safety standards.

The significance of HIL simulations is in their safety and cost effectiveness. Engineers rely on HIL simulations to rapidly prototype networks and test closed-loop control systems without expensing budgets on the real physical system production or exposing themselves to hazards.

Safety is the foremost concern when testing any industrial system. By using HIL simulations, engineers can create realistic scenarios, in which their products must operate accurately, while controlling potentially hazardous environments or situations. This allows them to test the product without being exposed to potential risks. Additionally, since there is no physical environment and system aside from computing devices involved in a HIL simulation, it eliminates the need for costly safety precautions such as fire suppression system or special protective clothing.

Cost saving is another major advantage of using HIL simulations. By eliminating expensive environment system prototypes and instead creating virtual models that interact with the product hardware, companies can save large amounts of money on development costs. Moreover, due to advances in computing power and commercialization of HIL simulation tool sets such as MATLAB/Simulink, businesses can successfully perform control system tests with very limited budgets. In addition, the cost savings can lead to increased scalability, which is an important factor when considering HIL simulations for product design and prototype purposes. With HIL simulations, it is possible to quickly scale up tests from small components to larger systems without having to invest in additional physical components each time, which would be prohibitively difficult with traditional prototyping methods due to budget constraints. Furthermore, they allow multiple users to simultaneously work together on different parts of the same project, enabling faster development cycles, while significantly reducing cost overheads associated with hiring additional staff for a single project.

Real-time operation is another important feature of HIL simulation as it enables accurate and timely interaction between the simulated system model and the physical hardware under test. In real-time HIL simulations, the simulation cycle time of the system and controller models takes place in step with the real-life sampling time, with minimal leading or lagging. The (compensated) simulation time cycle refers to the time interval in which (i) the system and controller dynamic model calculations are updated, (ii) the system outputs are measured, (iii) the communications are made among all connected entities (Raspberry Pi and Arduino in this paper) of the HIL simulation platform, and (iv) the idle (compensation) time is calculated and executed to go in step with the (real) sampling time. Therefore, the (compensated) simulation time cycle must last long enough to accommodate the execution of all calculations, measurements, and communications, while it must not exceed the (real) sampling time.

In this paper, the objective is to design and develop a low-cost laboratory-scale platform for HIL simulation of a three-area power generation system. This project provides students with a user-friendly platform to learn about HIL simulations and implement them for different industrial applications including but not limited to power systems, aerospace, and automotive systems.

Literature Review

In [3], a low-cost real-time control system platform is implemented by using an Arduino board and a Raspberry Pi. The platform allows the students to design their control scheme in Simulink and then run in a real-time simulation. In [4], a simple real-time simulation framework is designed for a single-phase inverter model using a Raspberry Pi for the inverter model, and an Arduino for the control algorithm computation. In [5], a hardware test application on the min-max algorithm is used to regulate a two-axis turbofan engine's fuel. This technique employs an Arduino microcontroller that uses a nonlinear model based on the min-max control scheme to control the fuel consumption of a turbofan.

In [6], a low-cost autonomous vehicle is designed, which is based on the combination of a Raspberry Pi, an Arduino, and a Zumo track-driven robot chassis. The control law is calculated on the Arduino in a real-time manner, while the Raspberry Pi performs additional computations, web interfacing, and wireless data streaming for tuning and debugging purposes. A control system based on the adaptive neuro-fuzzy inference system is studied in [7], which is implemented by using a Raspberry Pi and applied to control an inverted pendulum system. In [8], the design and implementation of hardware and software frameworks are studied for a basic mobile robot moving with radio frequency identification system. The hardware is based on a Raspberry Pi and an Arduino, where the former is used for sending commands to actuators and receiving data from sensors, and the latter is used for the control algorithm calculation.

In this paper, the real-time HIL simulation platform is developed for the first time by utilizing Raspberry Pi and Arduinos. As compared with similar works in the literature, they have used custom prototyped boards instead of the low-cost Arduino microcontroller used in this paper. It should be noted that their codes were developed by using MATLAB/Simulink and C++ environments on each device [9], which are similarly explored in this paper.

Hardware Setup

The hardware used in this project includes one Raspberry Pi 4 and three Arduino UNOs. In general, the Raspberry Pi can be used to simulate the mathematical model of the entire dynamic system, and the three Arduino UNOs can be used to simulate three local controllers. In the case of the power system considered in this paper, the Raspberry Pi is used to simulate the interconnected three-area power generation system, and the three Arduinos act as the local proportional-integral (PI) controllers for the three generators. The Raspberry Pi performs all the calculations for the dynamic model of the power system, while each Arduino performs calculations and responds to the Raspberry Pi with the control command for its associated generator, as depicted in Figure 1.

The Raspberry Pi performs calculation for the time evolution of the power system dynamic equation, which is represented by $F(.)$ in Figure 1 (the details are presented later in equations (2) and (4)). Each Arduino performs calculation for one of the three local controller equations, which is represented by f_i $(i = 1, 2, 3)$ in Figure 1 (the details are presented later in equation (3)). The Raspberry Pi broadcasts a unique value of y_i and p_i to each Arduino. The Arduino then performs basic arithmetic and responds with the new value of u_i. The Raspberry Pi then performs the calculations necessary to update the y_i values. These steps continue for a certain number of iterations specified by the user.

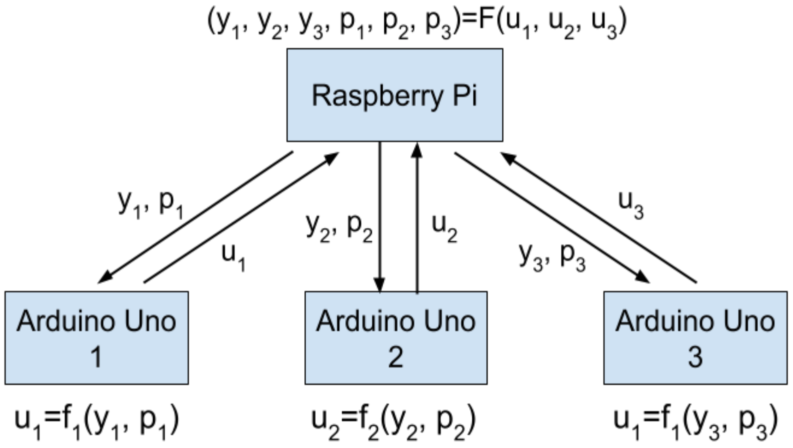

Figure 1. Diagram of the communication between Raspberry Pi and three Arduinos.

Communication Protocols and Setup

The main challenge in this project is to determine a communication protocol that is both fast and resource efficient. Due to the limited number of pins on the Raspberry Pi, the communication protocols may need external multiplexing modules to service multiple Arduinos at the same time. In the next section, three communication protocols will be reviewed and compared. They will be used for communication and data transmission between Raspberry Pi and Arduinos, and the most effective one will be chosen based on the communication speed and scalability performance.

Three wired communication protocols are presented for communication between the Raspberry Pi and three Arduinos: Serial Peripheral Interface (SPI), Universal Asynchronous Receiver/Transmitter (UART), and Inter-Integrated Circuit (I2C). All three protocols involve physically wiring the devices together, although each protocol requires a different wiring scheme.

SPI protocol requires four wires: Serial Clock (SCLK), Master Out Slave In (MOSI), Master In Slave Out (MISO), and Slave Select (SS). The Master device controls the clock line and selects which Slave device to communicate with by driving the appropriate SS line low. In this paper, multiple Arduinos communicate with a Raspberry Pi, so each Arduino would need its own SS line, which means that the Raspberry Pi would need as many SS lines as the number of Arduinos. This can quickly consume all of the general-purpose input/output (GPIO) pins on the Raspberry Pi. This is not feasible in our model using three Arduinos. Therefore, SPI is ruled out due to its hardware limitation.

UART is a form of serial communication that is simple and easy to implement. It requires only two wires (transmit and receive), which makes it ideal for point-to-point communication. It is asynchronous, so the sender and receiver do not need to synchronize their clocks before communication can occur. However, the two devices must agree on the same data rate (baud rate) beforehand. Multiplexing must be used in the case of communication between a Raspberry Pi and multiple Arduinos. However, achieving the multiplexing mechanism is simple. In addition, communication speed is quite fast since communication happens independently for each Arduino through an individual USB connection. Hardware wise, connecting the Arduinos to the Raspberry Pi is made simple by utilizing USB connections.

I2C is a serial communication protocol that uses two wires: Serial Data (SDA) and Serial Clock (SCL). Unlike UART, I2C supports multi-master and multi-slave communication, which makes it suitable for connecting multiple Arduinos to a Raspberry Pi. The key advantage of I2C is that many devices can be connected using just two wires. Each device on the I2C bus has its own address, and the Raspberry Pi can talk to individual Arduinos by addressing them directly. In the case of a single Raspberry Pi communicating with a single Arduino, the results show that I2C is the fastest protocol, but scaling up to multiple Arduinos, the results show a significant lag. Therefore, compared to UART and SPI, I2C is generally slower in the case of multiple Arduinos.

In UART protocol, the connections are made through the USB ports, while in I2C and SPI the connections are made through the GPIO pins and, hence, their voltage levels must be equalized. The Raspberry Pi pins are designed for 3.3(v), while the Arduino pins are designed for 5(v).

Connecting Arduino pins directly to Raspberry Pi pins could cause damage. Therefore, I2C and SPI require additional voltage level shifter modules to interface the two GPIO pins.

In this paper, the UART communication protocol is chosen as depicted in Figure 2. The most important factors in making this decision include communication speed, hardware limitation, and scalability as discussed above.

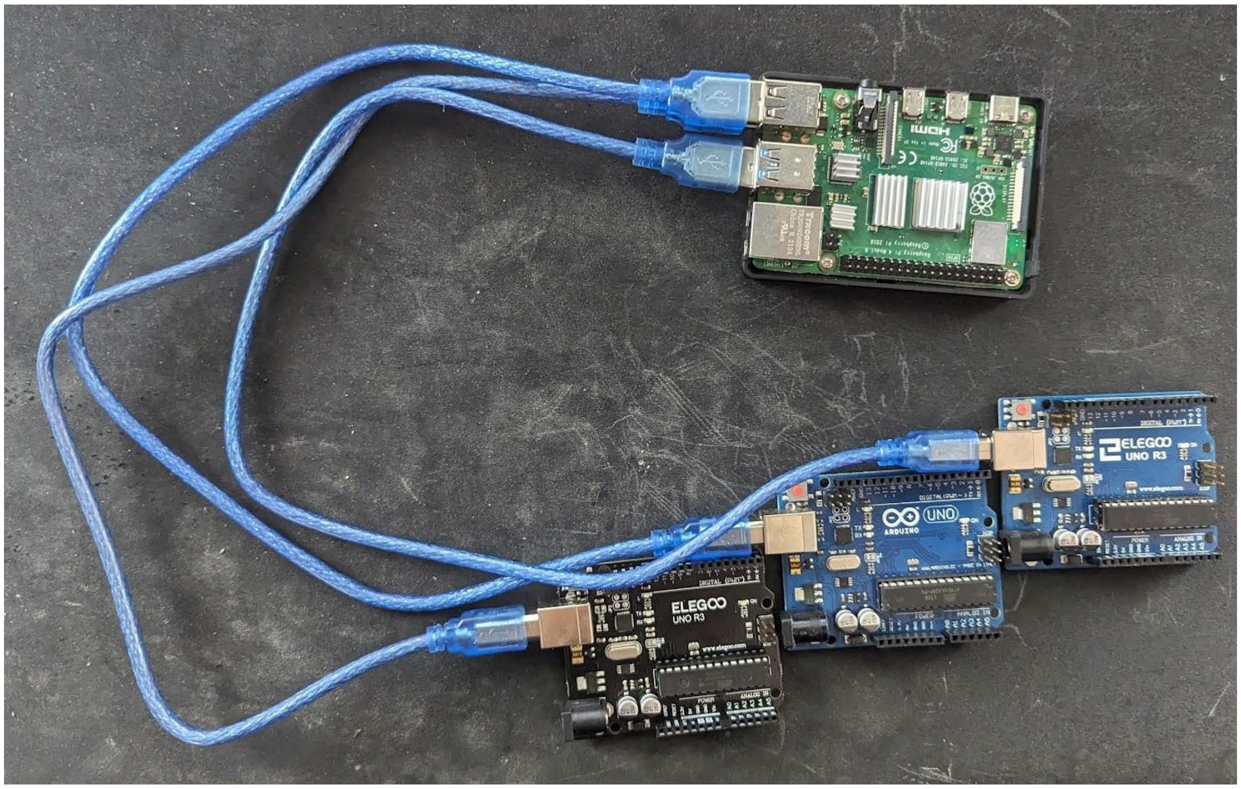
Figure 2. Hardware connection between Raspberry Pi and Arduinos using UART communication protocol and USB ports.

Dynamic Equations of the Power Generation System and the Three Controllers

The system studied in this paper is a stand-alone three-area power generation system that consists of synchronous generators [10]. The block diagram of this system is illustrated in Figure 3, where the effects of renewable energy resources, energy storage systems, and local loads are modelled by the external disturbance d_i for the area #*i* (i=*1,2,3*), which are all set to 0.1 (pu). In this figure, y_i, p_i, u_i, and e_i are the frequency deviation, the tie-line power deviation, the control signal of PI controller, and the area control error of area #*i*. The area control error e_i is defined by a linear combination of y_i (with coefficient B_i) and p_i (with coefficient 1) as follows:

$$e_i = B_i y_i + p_i \quad (i = 1,2,3) \tag{1}$$

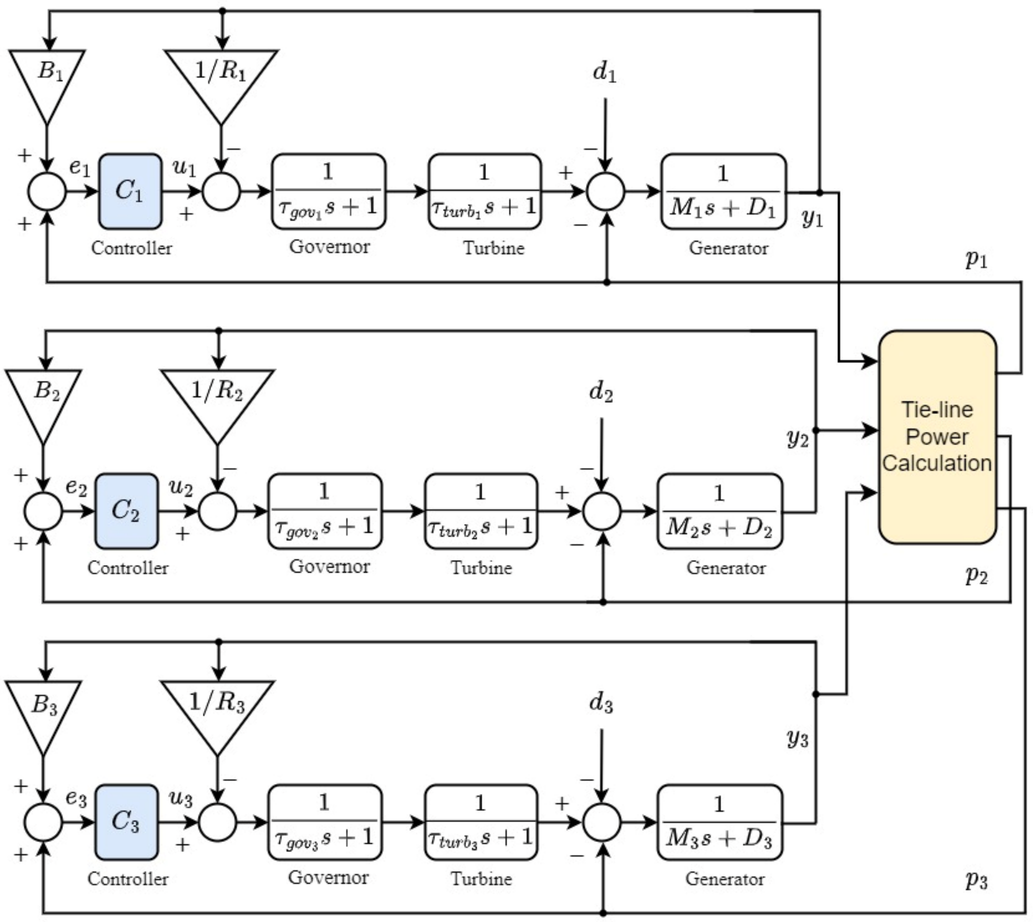

Figure 3. Block diagram of the three-area power generation system.

The details of the calculation of the tie-line power p_i in Figure 3 are demonstrated in Figure 4.

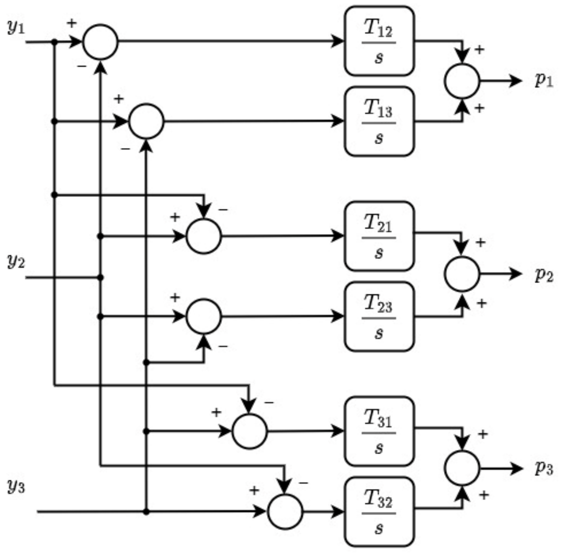

Figure 4. Tie-line power calculations.

The continuous system defined in Figures 3 and 4 are discretized by using the Tustin method with the sampling time of $T_s = 0.02\ (s)$. The discretized equations are represented in equations (2)-(4) for the three areas #*i* (*i=1, 2, 3*).

$$\begin{aligned} y_i(k) &= y_{i1}(k) + y_{i2}(k) \\ y_{i1}(k) &= \sum_{j=1}^{6} a_{i1}^{j} y_{i1}(k-j) + \sum_{j=0}^{6} b_i^j u_i(k-j) \\ y_{i2}(k) &= \sum_{j=1}^{4} a_{i2}^{j} y_{i2}(k-j) + \sum_{j=0}^{4} c_i^j [p_i(k-j) + d_i(k-j)] \end{aligned} \tag{2}$$

$$\begin{aligned} u_i(k+1) &= u_i(k) + f_i^1\, e_i(k+1) + f_i^0\, e_i(k) \\ e_i(k+1) &= B_i\, y_i(k) + p_i(k) \end{aligned} \tag{3}$$

$$\begin{aligned} p_i(k) &= z_{i1}(k) + z_{i2}(k) + z_{i3}(k) \\ z_{i1}(k) &= z_{i1}(k-1) + g_{i1}^1\, y_1(k-1) + g_{i1}^2\, y_1(k-2) \\ z_{i2}(k) &= z_{i2}(k-1) + g_{i2}^1\, y_2(k-1) + g_{i2}^2\, y_2(k-2) \\ z_{i3}(k) &= z_{i3}(k-1) + g_{i3}^1\, y_3(k-1) + g_{i3}^2\, y_3(k-2) \end{aligned} \tag{4}$$

The values of the coefficients in the discrete equations (2)-(4) are illustrated in Tables 1-9.

Table 1. The values of a_{i1}^{j} coefficients in equation (2).

i \ *j*	**1**	**2**	**3**	**4**	**5**	**6**
1	5.73	-13.67	17.39	-12.44	4.7	-0.75
2	5.8	-14.01	18.06	-13.09	5.06	-0.81
3	5.77	-13.87	17.78	-12.82	4.92	-0.78

Table 2. The values of a_{i2}^{j} coefficients in equation (2).

i \ *j*	**1**	**2**	**3**	**4**
1	3.86	-5.59	3.59	-0.87
2	3.89	-5.69	3.69	-0.9
3	3.88	-5.65	3.65	-0.88

Table 3. The values of b_i^j coefficients in equation (2).

i \ *j*	**0**	**1**	**2**	**3**	**4**	**5**	**6**
1	9.3e-7	1.3e-7	-2.7e-6	-2.5e-7	2.5e-6	1.19e-7	-8.1e-7
2	6.6e-7	6.6e-8	-1.9e-6	-1.3e-7	1.8e-6	6.3e-8	-5.9e-7
3	8.6e-7	9.8e-8	-2.5e-6	-1.9e-7	2.4e-6	9.3e-8	-7.6e-7

Table 4. The values of c_i^j coefficients in equation (2).

i \ *j*	**0**	**1**	**2**	**3**	**4**
1	-9.99e-4	0.0019	1.31e-4	-0.0019	8.67e-4
2	-0.0012	0.0024	1.21e-4	-0.0024	0.0011
3	-0.0012	0.0024	1.39e-4	-0.0024	0.0011

Table 5. The values of f_i^j coefficients in equation (3).

i \ *j*	***0***	***1***
1	0.099	-0.101
2	0.099	-0.101
3	0.099	-0.101

Table 6. The values of B_i in equation (3).

i	***1***	***2***	***3***
$\boldsymbol{B_i}$	20.6	16.9	18.93

Table 7. The values of g_{i1}^j coefficients in equation (4).

i \ *j*	***1***	***2***
1	0.04	0.04
2	-0.02	-0.02
3	-0.02	-0.02

Table 8. The values of g_{i2}^j coefficients in equation (4).

i \ *j*	***1***	***2***
1	-0.02	-0.02
2	0.04	0.04
3	-0.02	-0.02

Table 9. The values of g_{i3}^j coefficients in equation (4).

i \ *j*	***1***	***2***
1	-0.02	-0.02
2	-0.02	-0.02
3	0.04	0.04

Software Architecture

The Raspberry Pi first establishes connections with all three Arduinos. It performs the handshakes by waiting for the controllers to send the string "READY". Once all three controllers are connected, the simulation starts to execute. As indicated in Figure 5, first the Raspberry Pi calculates all the y_i ($i = 1, 2, 3$) values as per equation (2) and sends the calculated y_i and (previous) p_i values to the respective Arduinos #*i*. The Arduinos use these values to calculate e_i and u_i as per equation (3) and send the new u_i values back to the Raspberry Pi (as indicated in Figure 6). Once all these u_i values are received, the Raspberry Pi calculates new p_i values as per equation (4). Then, in order to meet the 0.02 (s) sampling time interval, the Raspberry Pi intentionally executes an idle (compensation) time.

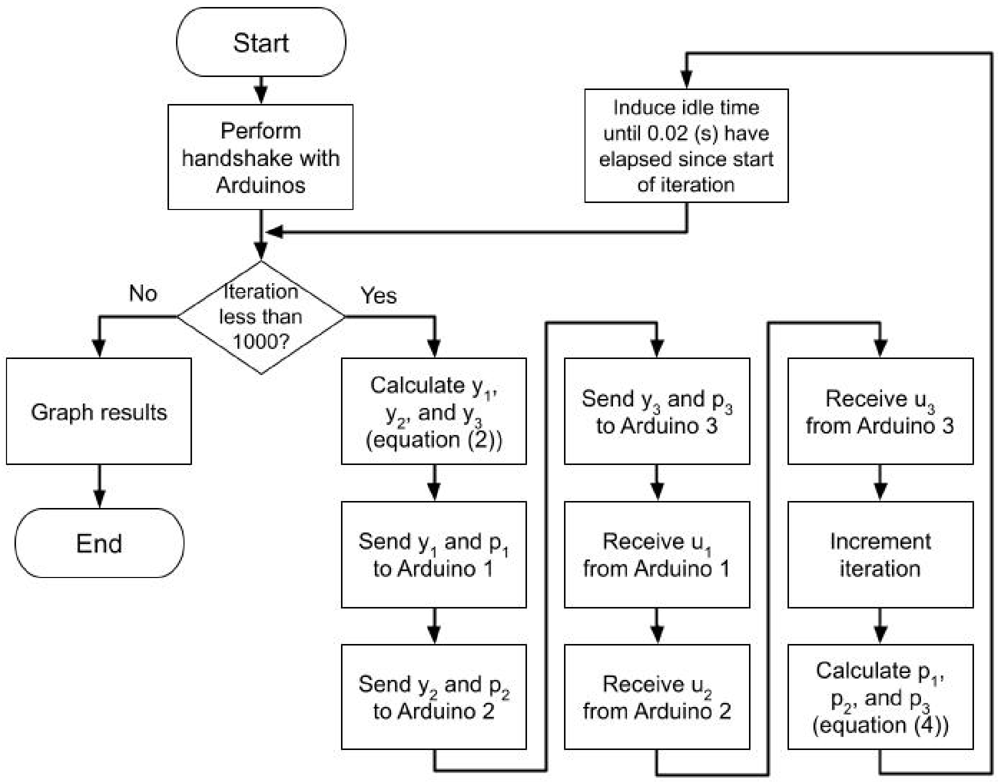
Start
Perform handshake with Arduinos
Induce idle time until 0.02 (s) have elapsed since start of iteration
No
Iteration less than 1000?
Yes
Graph results
End
Calculate y1, y2, and y3 (equation (2))
Send y1 and p1 to Arduino 1
Send y2 and p2 to Arduino 2
Send y3 and p3 to Arduino 3
Receive u1 from Arduino 1
Receive u2 from Arduino 2
Receive u3 from Arduino 3
Increment iteration
Calculate p1, p2, and p3 (equation (4))

Figure 5. Flow chart of the program on Raspberry Pi

Figure 6 illustrates the flowchart for the Arduino controllers #i ($i = 1, 2, 3$). On each Arduino, once the y_i and p_i values are received, the controller calculates the e_i and u_i values as per equation 3, and sends the new u_i value back to the Raspberry Pi.

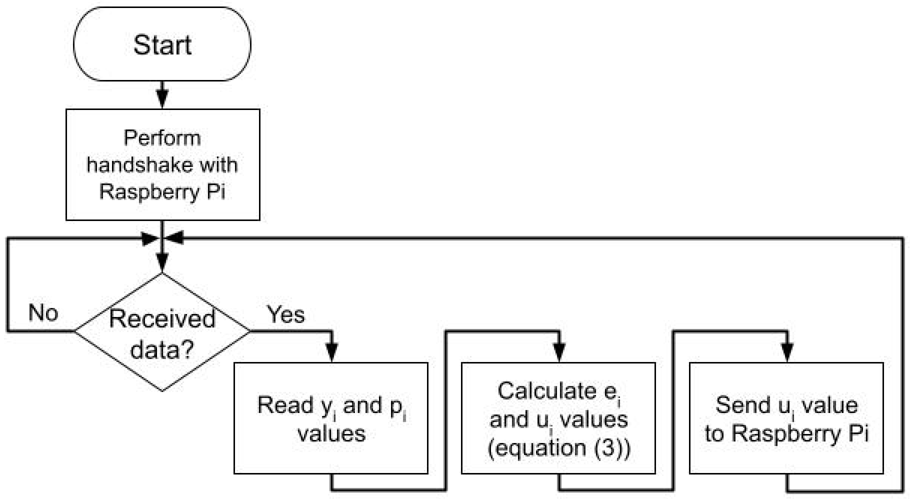
Start
Perform handshake with Raspberry Pi
No
Received data?
Yes
Read yi and pi values
Calculate ei and ui values (equation (3))
Send ui value to Raspberry Pi

Figure 6. Flow chart of the program on each Arduino.

Real-Time Closed-Loop Simulation of the Three-Area Power Generation System

In this HIL simulation, the sampling time $T_s = 0.02\ (s)$, which is presented earlier in this paper to discretize the dynamic models as per equations (2)-(4), is chosen to be bigger than the maximum communication time Com_n for all iterations $\#n$ ($n = 1, 2, 3, \dots, 1000$), which is defined as the time interval for the Raspberry Pi and Arduinos to send, process, and receive data per each iteration $\#n$ of the simulation. It is crucial that Com_n does not exceed (or fall behind) T_s at each sampling time window; otherwise, the simulation will be delayed and no longer real-time. As demonstrated in Figure 7 and Table 10, T_s is normally chosen to be much bigger than Com_n to provide enough margin. The communication protocol chosen, UART, ensures that transmission and receiving of data to and from the controllers will not serve as a bottleneck.

On the other hand, the real-time simulation should not fall ahead of the real elapsed time (cumulative sampling time) nT_s with n representing the number of iterations. Therefore, in order to compensate for this elapsed time error, an intentional idle time $Idle_n$ is added to Com_n for all simulation cycles $\#n$ ($n = 1, 2, \dots, 1000$) as demonstrated in Figure 7. As a result, the compensated simulation time cycle $Cycle_n$, which is defined as the summation of Com_n, $Idle_n$, and Del_n, goes in step with the nT_s to realize the real-time simulation. The time chart of real-time simulation is demonstrated in Figure 7 and Table 10.

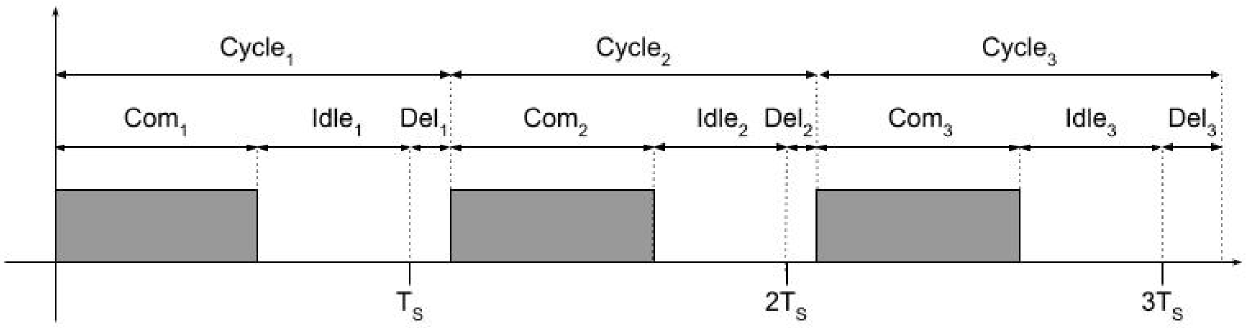

Figure 7. Time chart of the real-time simulation (notations defined in Table 10).

Table 10. Definitions of the notations used in Figure 7.

T_s	Sampling time with which the real-time simulation should be in step
nT_s	Real elapsed time (cumulative sampling time)
Com_n	Communication time (cumulative sending, processing, and receiving time)
$Idle_n$	Idle (compensation) time intentionally executed to follow real elapsed time
Del_n	Undesired delay as a result of timing inaccuracies in the calculation and execution of $Idle_n$
$Cycle_n$	Compensated simulation time cycle ($Cycle_n = Com_n + Idle_n + Del_n$)

The software program follows the steps in the algorithm in Table 11 to simulate the system in a rea-time manner.

Table 11. Software algorithm for real-time simulation.

Step	Function	Part of
1	Start the first iteration $n = 1$.	Com_n
2	Raspberry Pi calculates new y_i values by using equation (2).	Com_n
3	Raspberry Pi sends y_i and p_i values to three Arduinos.	Com_n
4	Arduinos use y_i and p_i values to calculate u_i values by using equation (3) and send them back to Raspberry Pi.	Com_n
5	Raspberry Pi receives u_i values back from three Arduinos.	Com_n
6	Raspberry Pi calculates p_i values by using equation (4).	Com_n
7	**Elapsed Time Error Compensation:** Raspberry Pi executes an idle time $Idle_n$ to compensate for the elapsed time error, which is the difference between the real elapsed time nT_s and the simulated elapsed time $Com_n + \sum_{k=1}^{n-1} Cycle_k$. Therefore, $Idle_n = nT_s - Com_n - \sum_{k=1}^{n-1} Cycle_k$.	$Idle_n$ and Del_n
8	Increment iteration $n = n + 1$ and go to Steps 2 if $n \leq n_{max}$.	-

Elapsed Time Error Compensation by Idle Time

As per Step 7 of the algorithm in Table 11, the "elapsed time error compensation" not only takes into account the current cycle's communication time Com_n but also the compensated simulation time cycle of all the previous cycles, that is $\sum_{k=1}^{n-1} Cycle_k$. The Raspberry Pi keeps idle for a time window of $Idle_n$ so that the simulated elapsed time $Com_n + \sum_{k=1}^{n-1} Cycle_k$ catches up with the real elapsed time nT_s. However, the timing inaccuracies in the calculation and execution of $Idle_n$ are represented and captured by the undesired delay Del_n at each iteration #n.

Figure 8 illustrates the communication time Com_n only, which excludes the idle (compensation) time $Idle_n$ and undesired delay Del_n. This figure shows that Com_n values are on average around 0.008 (s) and have an upper limit of 0.012 (s), which is used to determine that the sampling time $T_s = 0.02\ (s)$ is a proper choice ($T_s = 0.02\ (s) > 0.012\ (s)$).

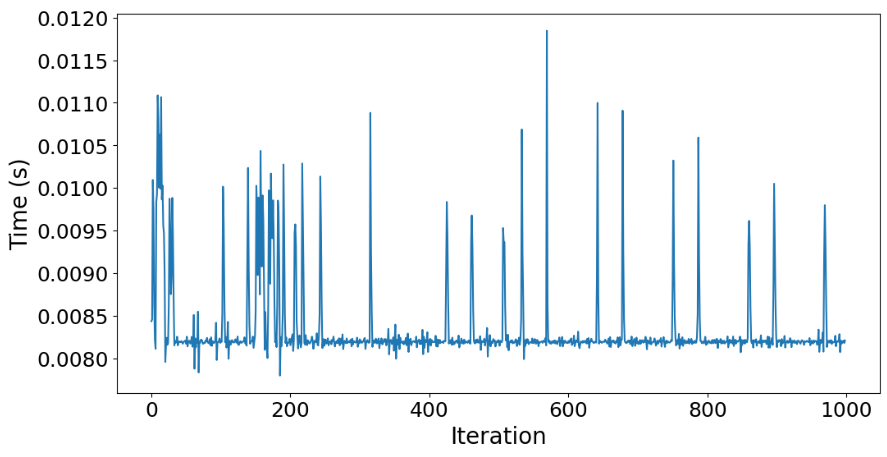

Figure 8. Communication time Com_n.

Figure 9 illustrates the idle time $Idle_n$, which is an intentionally executed wait time to compensate for the elapsed time error, which is the difference between the real elapsed time nT_s and the simulated elapsed time $Com_n + \sum_{k=1}^{n-1} Cycle_k$. Therefore, $Idle_n = nT_s - Com_n - \sum_{k=1}^{n-1} Cycle_k$. This figure shows that timing inaccuracies (in the calculation and execution of $Idle_n$) do not accumulate over time due to the "elapsed time error compensation" in Step 7 of the algorithm in Table 11.

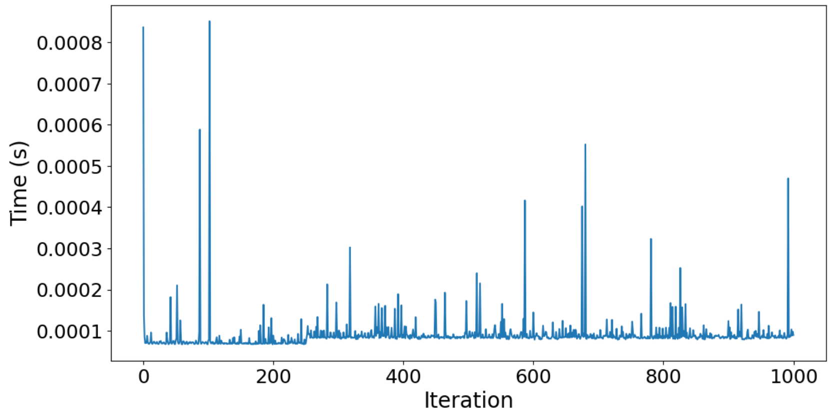

Figure 9. Elapsed time error $nT_s - Com_n - \sum_{k=1}^{n-1} Cycle_k$ compensated by idle time $Idle_n$.

Figure 10 demonstrates the graph of the compensated simulation time cycle $Cycle_n = Com_n + Idle_n + Del_n$, which includes the idle (compensation) time $Idle_n$. This figure shows that on average each cycle takes $0.02\ (s)$, which is equal to the sampling time $T_s = 0.02\ (s)$, with maximum deviations of $\pm 0.00075\ (s)$. This disparity is negligible since the spike in one iteration of the simulation is canceled out during the following iteration by another spike of the same magnitude in the opposite direction, so $Cycle_n$ will always maintain an average value of $0.02\ (s)$. This is due to the "elapsed time error compensation" in Step 7 of the algorithm in Table 11.

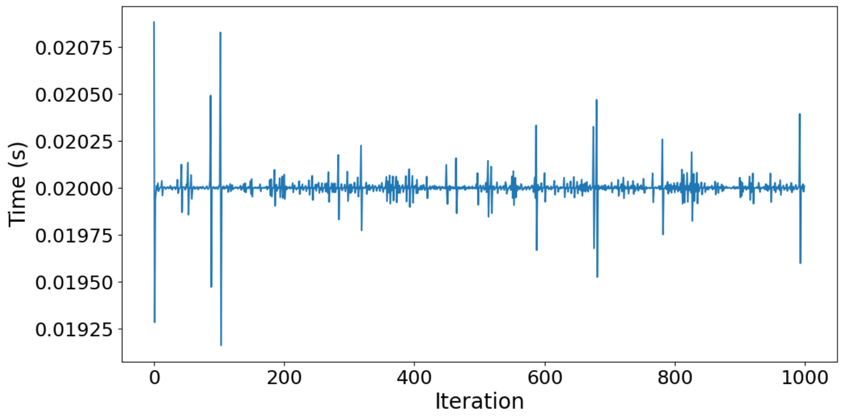

Figure 10. Compensated simulation time cycles $Cycle_n = Com_n + Idle_n + Del_n$.

Simulation Results of the Three-Area Power System

In this section, the simulation results of the three-area power system in Figures 3 and 4 and equations (1)-(4) are presented. These results include the MATLAB and HIL simulations, whose graphs are overlaid for the purpose of comparison. Figure 11 illustrates the frequency deviations of the three-area power system when a disturbance of $d_i = 0.1\ (pu)$ occurs at $t = 2\ (s)$. This figure shows that all the frequency deviations converge to zero. Figures 12 and 13 demonstrate the three control signals from the PI controllers and the tie-line power deviations of the three-area power system, which converge to zero. Moreover, these figures verify that the graphs resulting from the HIL simulation setup (Raspberry Pi and Arduinos) completely match the ones from MATLAB simulation.

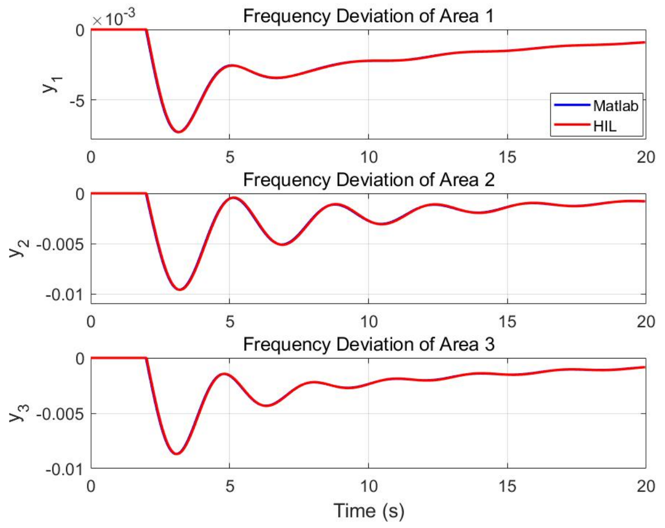

Figure 11. Frequency deviations of the three-area power system in Figures 3 and 4.

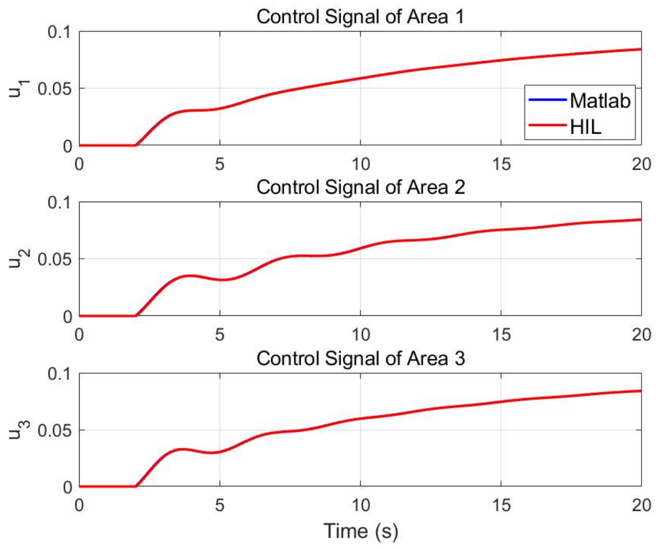

Figure 12. Control signals of the three-area power system in Figures 3 and 4.

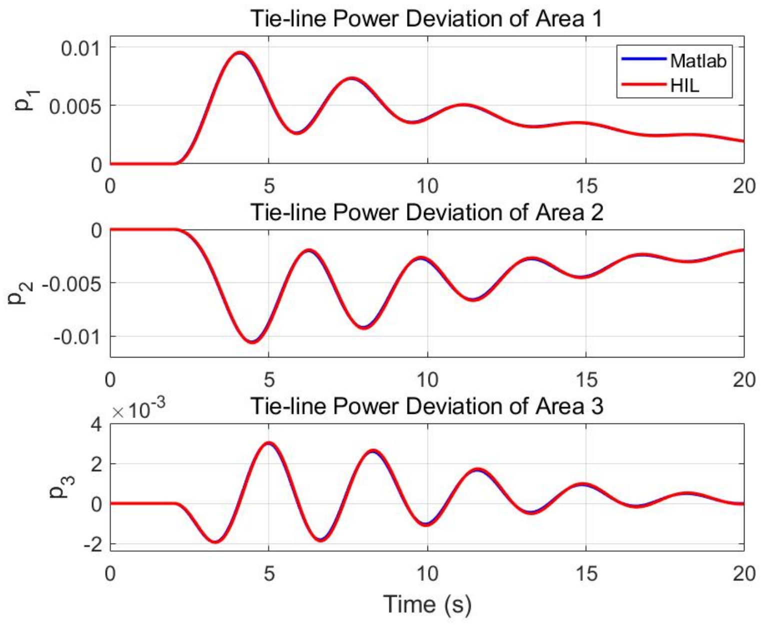

Figure 13. Tie-line power deviation of the three-area power system in Figures 3 and 4.

Educational Impact

This project was conducted at New Jersey Institute of Technology (NJIT). One undergraduate student from the Electrical and Computer Engineering Technology (ECET) program was awarded the Provost Undergraduate Research and Innovation (URI) Summer Research Fellowship to work on this project for ten weeks in Summer 2023. The undergraduate student collaborated with a PhD student and gained significant research experience. The project was showcased to graduate and undergraduate students at NJIT and gained significant attention from the students from different engineering disciplines. Moreover, the developed simulation platform in this project has been used as a case study in two courses, Embedded Systems I and II, and provided the students with a real engineering application of embedded systems and the HIL real-time simulation skill, which is in-demand in industry.

Conclusions

In this project, a hardware-in-the-loop simulation platform was designed and developed based on Raspberry Pi and Arduino UNO and used to demonstrate the real-time simulation of the closed-loop control system for a three-area power generation system. Three Arduinos simulated the three local controllers and communicated with the Raspberry Pi with negligible communication latency. In addition, the simulation platform produced outputs identical to those generated by MATLAB as a benchmark.

References

[1] N. Brayanov and A. Stoynova, "Review of Hardware-in-the-Loop – A Hundred Year Progress in the Pseudo-Real Testing", *Electrotechnica & Electronica (E+E),* vol. 54, pp. 3-4, 2019.

[2] F. Mihalič, M. Truntič, and A. Hren, "Hardware-in-the-Loop Simulations: A Historical Overview of Engineering Challenges", *Electronics*, vol. 11, no. 15, pp. 2462, 2022.

[3] J. Sobota, R. PiŜl, P. Balda, and M. Schlegel, "Raspberry Pi and Arduino Boards in Control Education", *IFAC Proceedings*, vol. 46, no. 17, pp. 7-12, 2013.

[4] S. A. Zulkifli and A. Hamzah, "Understanding Real-Time Simulation on Single-Phase Inverter Using Low-Cost Microcontroller for Undergraduate Level", *IEEE 13th International Colloquium on Signal Processing & its Applications (CSPA)*, pp. 144-148, 2017.

[5] M. Davoodi and H. Bevrani, "A New Application of the Hardware in The Loop Test of the Min-Max Controller for Turbofan Engine Fuel Control", *Advanced Control for Applications: Engineering and Industrial Systems*, vol. 5, pp. 138, 2023.

[6] R. Krauss, "Combining Raspberry Pi and Arduino to Form a Low-Cost, Real-Time Autonomous Vehicle Platform", *American Control Conference (ACC)*, pp. 6628-6633, 2016.

[7] H. Khati, H. Talem, R. Mellah, M. A. Touat, and M. A. Nehmar, "Processor-In-the-Loop Simulation of a Neuro-Fuzzy Controller on Raspberry Pi 3 board", *19th International Multi-Conference on Systems, Signals & Devices (SSD)*, Sétif, Algeria, pp. 146-151, 2022.

[8] I. Akli, H. Boukari Alidou, A. Chekir, and S. Ouazine, "Basic Mobile Robot Prototyping Using RFID", *3rd International Conference on Embedded & Distributed Systems (EDiS)*, Oran, Algeria, pp. 102-107, 2022.

[9] J. Walter, M. Fakih, and K. Grüttner, "Hardware-based Real-Time Simulation on the Raspberry Pi", *Proceedings of the 2nd Workshop on High Performance and Real-time Embedded Systems*, Vienna, Austria, vol. 20, 2014.

[10] M. Shojaee and S. M. Azizi, "Decentralized Robust Controller Design for Strongly Interconnected Generators", *IEEE Access*, vol. 11, pp. 16085-16095, 2023.