S & M 4278

Design and Implementation of Web-based Remote Monitor and Control System Using Raspberry Pi, Programmable Logic Controller, and Django Framework

Po-Hsun Chen,[1] Chung-Hong Lee,[1*] Chun-Chieh Chang,[2] and Cheng-Yi Chen[2**]

[1]Department of Electrical Engineering, National Kaohsiung University of Science and Technology, Kaohsiung 807618, Taiwan
[2]Department of Electrical Engineering, Cheng Shiu University, Kaohsiung City 83347, Taiwan

(Received November 20, 2025; accepted December 19, 2025)

Keywords: Internet of Things, Django, programmable logic control, web-based control

With the rapid development of Industry 4.0, upgrading traditional control systems to Internet of Things (IoT)-based solutions has become increasingly important. We propose a low-cost remote monitoring and control system that integrates sensing, data acquisition, and control within a unified IoT framework. The system is implemented on a Raspberry Pi platform using the web framework Django and communicates with a programmable logic controller (PLC) via the Modbus transmission control protocol (Modbus TCP) to acquire electrical and environmental sensing data. A web-based human–machine interface developed with HTML5 and JavaScript enables cross-platform remote monitoring, real-time sensing visualization, and control through standard web browsers. The novelty of this work lies in the integration of lightweight web technologies with industrial PLC-based sensing and control, thereby providing a scalable, extensible, and web-native IoT solution that relies on no proprietary software.

1. Introduction

In today's rapidly evolving digital era, the integration of communication and automation technologies has become a powerful driver of transformation in both daily life and industrial operations. The rapid advancement of fifth-generation (5G) mobile communication and Internet of Things (IoT) presents unprecedented opportunities and challenges. Within the industrial sector, communication technologies—particularly the Modbus protocol—have gained increasing attention. As a key application of 5G, IoT has enabled a new era of connectivity for personal and business activities. By linking diverse devices, sensors, and systems, IoT allows real-time monitoring, data analysis, and automated control. This capability holds significant potential to enhance urban infrastructure, increase productivity, and improve energy efficiency. Moreover, IoT expands the scope of automation technologies, such as Modbus-based programmable logic controller (PLC) systems, by enabling more efficient industrial automation. The emergence of single-board computers, such as the Raspberry Pi, further offers a cost-effective and scalable

*Corresponding author: e-mail: leechung@mail.ee.nkust.edu.tw
**Corresponding author: e-mail: k0464@gcloud.csu.edu.tw
https://doi.org/10.18494/SAM6072

platform for IoT. These compact yet powerful devices can act as IoT nodes for data collection, processing, and transmission, and their robust computing power and versatile interfaces make them well suited to developing customized IoT applications.

Smart home systems leverage IoT to interconnect household appliances, enabling automation and remote control. Such systems often rely on the Message Queuing Telemetry Transport (MQTT) protocol and use cost-effective hardware platforms, such as the ESP32 and Raspberry Pi, in combination with Node-RED and serverless technologies to achieve efficient home automation.(1,2) Guo *et al.*(3) also applied MQTT-based IoT networks and robotic supervisory control and data acquisition (SCADA) systems to smart homes. By employing Raspberry Pi as the primary control unit, the system demonstrated effectiveness in automation and real-time responsiveness. Studies show that this approach can coordinate the operation of multiple robots and home devices via MQTT, thereby enhancing performance through real-time data analysis. Furthermore, a new MQTT protocol flag has been proposed to improve data transmission security.(4) Experimental results demonstrated that this method effectively mitigates common network attacks and strengthens the overall system security.

Additionally, an innovative agricultural system integrating PLC, cloud computing, LoRa, and LoRaWAN technologies has been developed to enable the remote monitoring and management of farm environments and equipment.(5) Experimental results indicate that this system significantly improves crop yields and resource efficiency while reducing adverse environmental impacts. In Ref. 6, various innovative agricultural applications were reviewed, including smart weather stations and hydrometeorological data collection systems. The review provided detailed experimental evidence demonstrating how IoT technologies can enhance automation in farm production and support data-driven decision-making. Faid et al.(7) examined a farming system combining AI and IoT, employing low-cost weather stations to collect meteorological data. Results showed that this approach can effectively predict weather conditions, improve production efficiency, and reduce the risk of crop failure. Furthermore, a low-cost measurement system for field-based hydrological and meteorological data collection has been reported as a cost-effective solution for continuous environmental monitoring and real-time feedback through cloud-based analysis.(8,9)

Innovative building systems integrate Modbus and MQTT communication into Raspberry Pi–based embedded platforms to enable remote monitoring.(10,11) In such studies, a PLC control system was successfully upgraded into an IoT-enabled solution, resulting in a web-based, cross-platform environmental monitoring and control system. In the broader context of smart offices and smart cities, IoT and big data technologies play critical roles.(12–14) For instance, IoT technologies have been applied to innovative, secure office systems to enable intelligent environmental management.(13) Moreover, an open-source MQTT protocol implementation based on NS-3 has been developed for smart building IoT applications.(15) Simulation results demonstrated efficient resource management and low-latency data transmission, showing that the system can operate reliably under various network loads, making it suitable for large-scale smart building deployments.

In industrial automation, the integration of IoT technologies and Industry 4.0 principles has gradually transformed traditional production models. Flexible automated optical inspection architectures enable real-time data collection and analysis, thereby improving product quality and manufacturing efficiency. Such systems demonstrate strong potential for industrial production, as they effectively detect defects and enable immediate process adjustments.(16) Moreover, IoT- and machine-learning-based models for abnormal product detection and classification provide intelligent solutions for industrial production, with experimental results showing significant improvements in both product quality and efficiency.(17) Industrial automation also benefits from the real-time simulation capabilities of digital twin technology, which serves as an effective tool for optimizing mechanical equipment and manufacturing processes. Experimental results confirm that this approach increases efficiency and quality while reducing downtime and maintenance costs.(18) In addition, to further support the advancement of Industry 4.0, establishing a center of excellence based on computer engineering principles is crucial. Such a center can drive technological innovation, enhance productivity, and provide enterprises with advanced technical support and platforms.(19)

Django is an open-source Python web framework that enables developers to build scalable and maintainable web applications rapidly. It provides a comprehensive development environment with built-in tools, libraries, and configurations to manage all aspects of web development. Django follows the model–template–view (MTV) design pattern, which differs from the conventional model–view–controller approach commonly adopted in web development.(20) It has been successfully applied across diverse domains, including blogging platforms, e-commerce, social media, content management systems, and news portals. Django's continuous evolution and extensive functionality make it a strong choice for modern web application development. In parallel, Asynchronous JavaScript and XML (AJAX) is a front-end technology that enables dynamic web interactions without requiring full-page reloads.(21)

In today's highly connected world, IoT has demonstrated significant achievements and profoundly transformed both daily life and business operations. IoT enables a seamless interconnection of devices and sensors, supporting real-time monitoring, automated control, and intelligent data analysis. Nevertheless, the successful deployment of IoT applications relies on a robust software foundation and a stable development platform. Hence, the choice of an efficient and reliable back-end framework is critical. In this study, Django was selected as the development framework for the control system. As a mature and widely adopted Python-based web framework, Django provides the efficiency, scalability, and security required for modern web applications.

2. Monitoring and Control System Design: Django Approach

As reported in Refs. 10 and 11, the hardware architecture is locally managed by PLCs through a distributed control system using serial communication. In contrast, these distributed controllers are centrally supervised by a host computer via Modbus transmission control protocol (Modbus TCP). To address the high cost of hardware and software, Raspberry Pi 4B, combined with the Node-RED programming tool (developed by IBM), was used to integrate MQTT and Modbus TCP protocols for remote monitoring and control.(11) In this study, the free and open-source web framework Django is adopted to develop a Python-based web control system, replacing Node-

RED and other drivers. Figure 1 presents the overall operation of the proposed Django-based architecture, which comprises six major components: views, models, databases, the uniform resource locator (URL) dispatcher, templates, and PLCs. Views are Python functions that handle HTTP requests, execute logic, and return responses. Models define the structure of user-generated tables, enabling data manipulation directly in Python without requiring SQL syntax. URLs map HTTP requests to the corresponding view functions, while templates—implemented with HTML—serve as the interface between back-end logic and front-end display, supporting dynamic content presentation. The PLC executes user-issued commands, which are sent from HTML pages via Ajax-based HTTP requests, routed through URLs, processed by view functions, and ultimately transmitted to the PLC through the Modbus protocol. This layered architecture promotes a clear separation of concerns, facilitating system development, maintenance, and scalability. Leveraging Django's capabilities, we present a comprehensive, cross-platform solution for web-based monitoring and control, as detailed in the following sections.

2.1 Reading data from the device: Example of electricity meter measurement

The real-time power data interface comprises two main components: the primary power data display page and the distribution board monitoring page. Figure 2 illustrates the design process of the human–machine interface (HMI) for the primary display page. The workflow can be summarized as follows. (1) Data acquisition and storage: a thread implemented in cp_ip101_views.py retrieves power meter data from the PLC via pyModbusTCP.client and stores the computed values in an array. (2) Front-end request: the web interface issues an HTTP request to the back-end views using the Ajax method. (3) Request handling: the HTTP request is routed through the URL dispatcher to the appropriate view function, which executes the corresponding command. (4) Data transfer and visualization: the view function returns data to the front-end using the HttpResponse method, where it is rendered on the webpage.

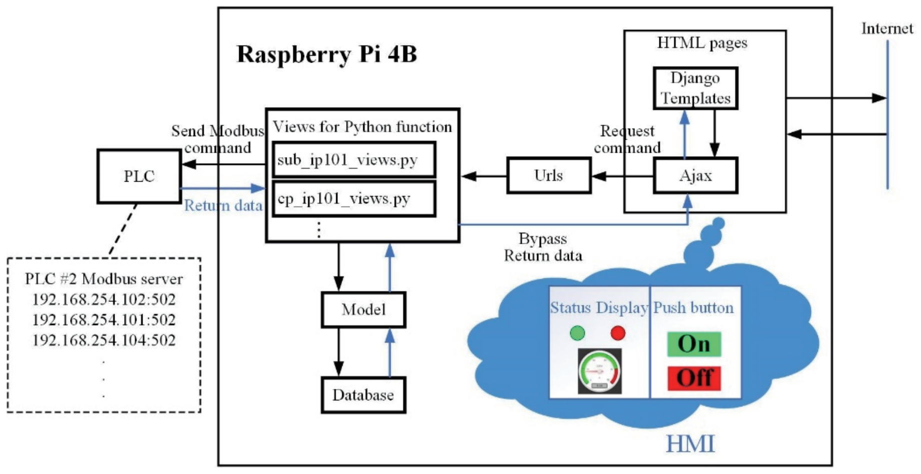


Fig. 1. (Color online) Design diagram of Django's approach.

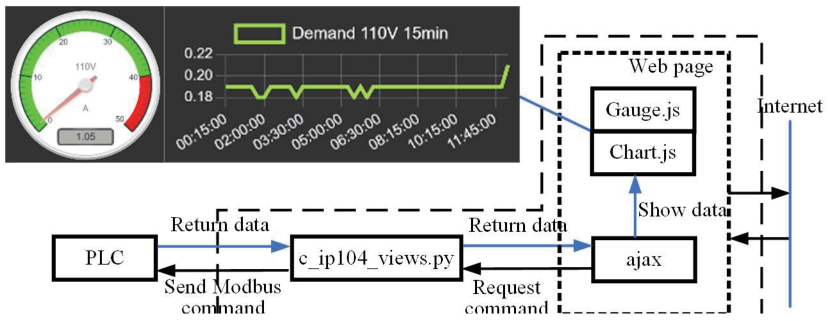


Fig. 2. (Color online) HMI design flowchart for electric real-time data display.

2.2 Write data to the device: Example of flat LED control

Figure 3 illustrates the control and status inspection process of the laboratory lighting system. The thread of cp_ip101_views.py periodically communicates with the PLC to obtain lighting status information and stores the received data in a variable array. The front-end interface retrieves this information via AJAX requests to the designated view program, ensuring real-time updates to the HMI. Control commands for the LED are issued via sub_ip101_views.py, which initiates Modbus TCP communication and writes the corresponding values into the input registers. Each lamp is assigned a unique Modbus address, and its brightness is adjusted by transmitting values in the range of 0 to 32,767. The slider value from the front end is passed to the back-end thread through Ajax, enabling real-time control. This architecture ensures seamless integration between the user interface and the hardware, as commands are transmitted instantly and executed by the Python program. Consequently, the system can dynamically adjust brightness or toggle lighting in accordance with the selected mode, thereby enhancing both responsiveness and reliability.

2.3 Language switching and permission control

As illustrated in Fig. 4, a language selector is defined in the front-end HTML to determine the user's preferred language. Each object in the HTML structure is assigned a unique ID, with predefined content available in both English and Chinese. The selector invokes a JavaScript function to switch the display language. When English is selected, the corresponding elements are displayed in English; otherwise, their Chinese equivalents are shown. The implementation process is as follows: (1) a selector named languageSelect is added to the navigation bar with two options, namely, Chinese (zh) and English (en); (2) the function changeLanguage retrieves relevant elements using document.getElementById and updates their text content in accordance with the selected language.

In the Django Admin interface, two user groups are defined: External_Users and Regular_Users. Permission control is implemented through template tags associated with these groups and applied to the front-end HTML. As illustrated in Fig. 5, the front-end code uses the {% load

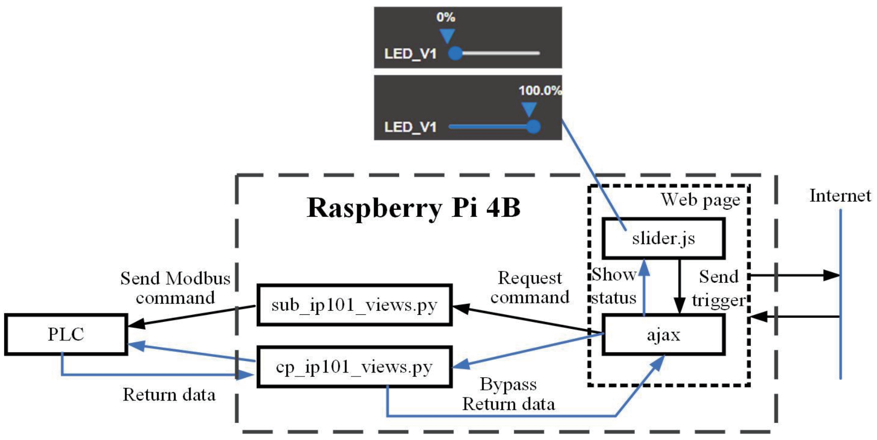


Fig. 3. (Color online) Individual flat LED light control flow chart.

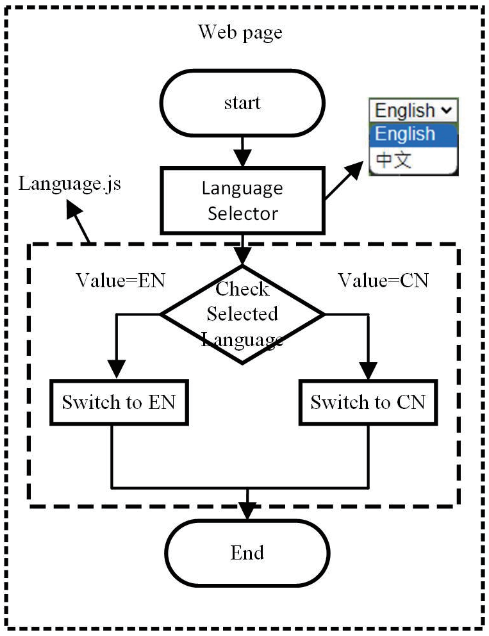
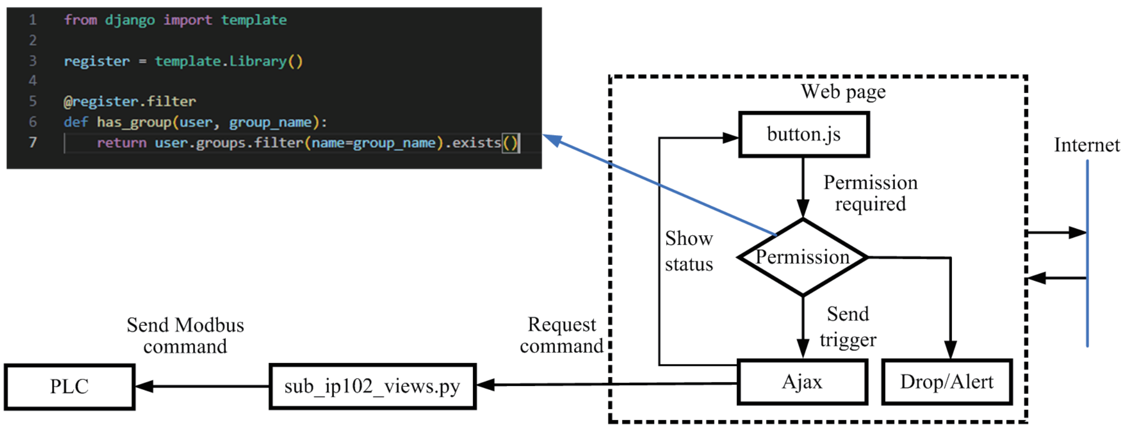


Fig. 4. (Color online) Language switching architecture diagram.



```
from django import template

register = template.Library()

@register.filter
def has_group(user, group_name):
    return user.groups.filter(name=group_name).exists()
```

Fig. 5. (Color online) Control authority architecture diagram.

user_tags %} syntax to reference the groups defined in the Django back-end. These groups can then be utilized in JavaScript to verify whether the current user possesses the required permissions. In the present system, Regular_Users are authorized to control curtains and lighting, whereas External_Users are restricted to viewing the webpages.

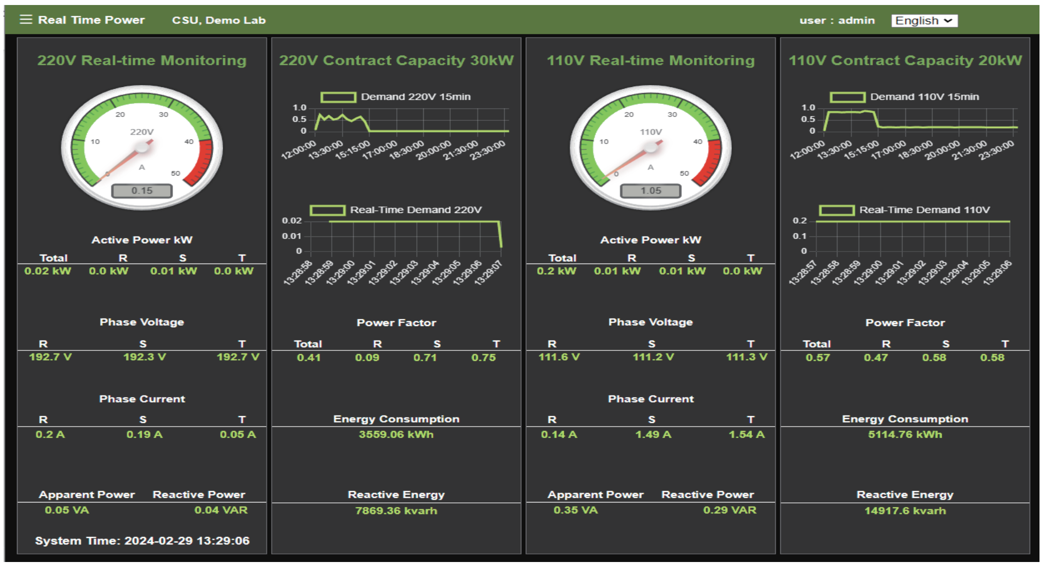
3. Experimental Results and Discussion

The integration of experimental software and hardware was achieved by transferring Modbus register data through Django URLs and employing front-end Ajax functions to invoke back-end programs (cp_ip101_views.py, cp_ip102_views.py, cp_ip104_views.py, Demand_views.py, and Environment_views.py. Additional control programs (sub_ip101_views.py, sub_ip102_views.py, and sub_ip104_views.py) together with Curtain.js, LED_Control.js, and Envoronment.js, transmitted commands via the Modbus register. The stability testing of the Django back-end (Python) and front-end HMI data transmission, as well as the integrated hardware display, was carried out on Raspberry Pi 4B. System verification was performed by logging into the Django website, accessing the HMI interface, and validating the displayed data in real time.

To access the system, users log in to the Django-based web platform at https://120.118.143.000:8000 using a valid username and password, and then open the HMI for real-time power monitoring. As shown in Fig. 6, the interface displays the power data of the 192 and 110 V circuits, including real-time demand, 15-min average demand, three-phase voltage and current (R, S, T), apparent power, reactive power, power factor, and cumulative energy consumption. The monitoring results confirm that the proposed system can stably acquire and visualize electrical parameters in real time through the Modbus communication architecture, without observable data loss or abnormal delay during continuous operation. This demonstrates the feasibility of integrating a PLC-based field layer with a web-based HMI for practical power monitoring applications.

The completed lighting control interface, as shown in Fig. 7, enables the centralized management of scenario modes, whiteboard lights, floodlights, and individual LED units. LED brightness is regulated using slider controls corresponding to Modbus register values (0–32,767), which are linearly mapped to control voltages ranging from 0 to 10 V. The current brightness percentage is displayed to provide immediate user feedback. The experimental control results, illustrated in Fig. 8, indicate that the lighting response follows the commanded brightness settings consistently and smoothly, validating the correctness of the Modbus value-to-voltage mapping and the stability of the control loop. These results demonstrate that the proposed web-based HMI can reliably perform real-time monitoring and lighting control tasks, making it suitable for smart building and energy management applications.

The field control system presented in this study is implemented using a PLC-based hardware architecture, as reported in Ref. 10. PLCs have been widely used in our laboratory for more than a decade and have consistently demonstrated stable control and data acquisition performance. In the past, SCADA systems typically relied on high-specification computer hosts. Although such solutions offered strong computational capabilities, they also incurred higher hardware costs and imposed limitations on software licensing and system integration, which were often tied to proprietary development environments.



Real Time Power CSU, Demo Lab user : admin English

220V Real-time Monitoring

Active Power kW

Total	R	S	T
0.02 kW	0.0 kW	0.01 kW	0.0 kW

Phase Voltage

R	S	T
192.7 V	192.3 V	192.7 V

Phase Current

R	S	T
0.2 A	0.19 A	0.05 A

Apparent Power	Reactive Power
0.05 VA	0.04 VAR

System Time: 2024-02-29 13:29:06

220V Contract Capacity 30kW

Power Factor

Total	R	S	T
0.41	0.09	0.71	0.75

Energy Consumption
3559.06 kWh

Reactive Energy
7869.36 kvarh

110V Real-time Monitoring

Active Power kW

Total	R	S	T
0.2 kW	0.01 kW	0.01 kW	0.0 kW

Phase Voltage

R	S	T
111.6 V	111.2 V	111.3 V

Phase Current

R	S	T
0.14 A	1.49 A	1.54 A

Apparent Power	Reactive Power
0.35 VA	0.29 VAR

110V Contract Capacity 20kW

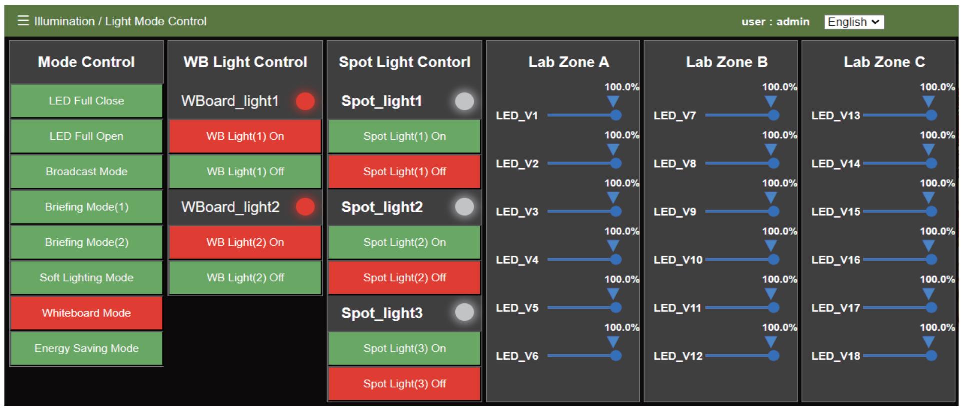
Power Factor

Total	R	S	T
0.57	0.47	0.58	0.58

Energy Consumption
5114.76 kWh

Reactive Energy
14917.6 kvarh

Fig. 6. (Color online) 192 and 110 V power real-time monitoring screen.



Mode Control	WB Light Control	Spot Light Contorl	Lab Zone A	Lab Zone B	Lab Zone C
LED Full Close	WBoard_light1 ●	Spot_light1 ●	LED_V1 100.0%	LED_V7 100.0%	LED_V13 100.0%
LED Full Open	WB Light(1) On	Spot Light(1) On	LED_V2 100.0%	LED_V8 100.0%	LED_V14 100.0%
Broadcast Mode	WB Light(1) Off	Spot Light(1) Off	LED_V3 100.0%	LED_V9 100.0%	LED_V15 100.0%
Briefing Mode(1)	WBoard_light2 ●	Spot_light2 ●	LED_V4 100.0%	LED_V10 100.0%	LED_V16 100.0%
Briefing Mode(2)	WB Light(2) On	Spot Light(2) On	LED_V5 100.0%	LED_V11 100.0%	LED_V17 100.0%
Soft Lighting Mode	WB Light(2) Off	Spot Light(2) Off	LED_V6 100.0%	LED_V12 100.0%	LED_V18 100.0%
Whiteboard Mode		Spot_light3 ●			
Energy Saving Mode		Spot Light(3) On			
		Spot Light(3) Off			

Fig. 7. (Color online) Flat LED control system.

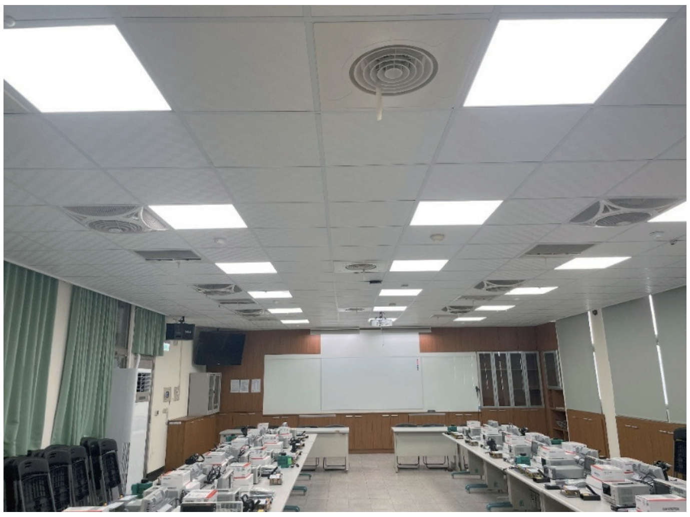


Fig. 8. (Color online) Actual control test of flat LED light.

With the widespread adoption of open-source software packages, acquiring and integrating these resources have become more accessible and flexible. As a result, implementing a single-board embedded system equipped with IoT and HMI functionalities is no longer a technically demanding task. Furthermore, Django, as a highly structured web framework, employs an MTV architecture that simplifies front-end and back-end integration. In addition, Django-based web applications support the HTML5 standard, enabling strong cross-platform compatibility and modern user interface design, thereby allowing internet-enabled devices to monitor and visualize system status in real time.

4. Conclusions

We presented a novel, cost-effective IoT integration framework based on the Django web framework, designed to enable bidirectional communication between front-end and back-end systems for the monitoring and control of laboratory-scale equipment. The front end employs AJAX to support real-time data updates and interactive visualization. In contrast, the back end uses Python-based Django applications integrated with Modbus TCP to communicate with multistation PLC systems. Front-end JavaScript handles message publishing and subscription, whereas back-end Python executes data acquisition and control commands via Modbus TCP, enabling all system functions to be accessed remotely via standard web browsers without requiring dedicated software installation. The novelty of this study lies in the integration of a lightweight, independent Django-based architecture with industrial PLCs to enable real-time data synchronization, power monitoring, and equipment control within a unified IoT platform. Compared with the approaches reported in Refs. 10 and 11, the proposed IoT design framework based on Django offers significant advantages in terms of low implementation cost, cross-platform compatibility, and independence from proprietary development platforms. Furthermore, unlike conventional PLC monitoring systems that rely on proprietary software or tightly coupled architectures, the proposed approach extends the operational lifespan of the existing Modbus-enabled PLC equipment and enhances overall system sustainability and maintainability. Future work will focus on incorporating multilingual language-switching capabilities and scaling the proposed architecture to larger IoT monitoring systems, with potential extensions to water resource monitoring, wind power generation, and energy storage systems, to further evaluate the security and stability.

Acknowledgments

The author would like to thank the National Science Council of Taiwan for financial support for this research under grant NSTC 114-2637-E-230-001.

References

1 M. Esposito, A. Belli, L. Palma, and P. Pierleoni: Sensors **23** (2023) 4459. https://doi.org/10.3390/s23094459
2 A. Zare and M. T. Iqbal: Proc. 2020 IEEE International IOT, Electronics and Mechatronics Conf. (IEEE, 2020) 1–5. https://doi.org/10.1109/IEMTRONICS51293.2020.9216412

3 J.-H. Guo, T.-Y. Lin, and K.-H. Hsia: J. Rob., Networking Artif. Life **9** (2022) 202. https://doi.org/10.57417/jrnal.9.2_202
4 A. Munshi: Sensors **22** (2022) 2174. https://doi.org/10.3390/s22062174
5 M. Saban, M. Bekkour, I. Amdaouch, J. El Gueri, B. Ait Ahmed, M.Z. Chaari, J. Ruiz-Alzola, A. Rosado-Muñoz, and O. Aghzout: Sensors **23** (2023) 2725. https://doi.org/10.3390/s23052725
6 S. Amertet, G. Gebresenbet, H. M. Alwan, and K. O. Vladmirovna: Appl. Sci. **13** (2023) 7315. https://doi.org/10.3390/app13127315
7 A. Faid, M. Sadik, and E. Sabir: Agric. **12** (2022) 35. https://doi.org/10.3390/agriculture12010035
8 K. Tatsumi, T. Yamazaki, and H. Ishikawa: Technol. **9** (2021) 78. https://doi.org/10.3390/technologies9040078
9 P. B. Leelavinodhan, M. Vecchio, F. Antonelli, A. Maestrini, and D. Brunelli: Sensors **21** (2021) 3831. https://doi.org/10.3390/s21113831
10 C.-Y. Chen, C.-Y. Liu, C.-C. Kuo, and C-F. Yang: Micromachines **8** (2017) 241. https://doi.org/10.3390/mi8080241https://doi.org/10.3390/mi8080241
11 C.-Y. Chen, S.-H. Wu, B.-W. Huang, C.-H. Huang, and C-F. Yang: Internet Things **27** (2024) 101305. https://doi.org/10.1016/j.iot.2024.101305
12 L.-B. Bilius, A.-T. Andrei, and R.-D. Vatavu: Proc. 2024 Int. Conf. Development and Application Systems (IEEE, 2024) 152–155. https://doi.org/10.1109/DAS61944.2024.10541208
13 M. S. A. Jion and M. Ahmad: Proc. Int. Conf. Adv. Comput. Commun. Electr. and Smart Systems (IEEE, 2024) 1–6. https://doi.org/10.1109/iCACCESS61735.2024.10499529
14 C. Inibhunu and A. C. McGregor: Proc. 2020 IEEE 22nd Int. Conf. High Performance Computing and Commun. (IEEE, 2020) 1096–1103. https://doi.org/10.1109/HPCC-SmartCity-DSS50907.2020.00197
15 M. A. Salimee, M. A. Pasha, and S. Masud: Proc. Int. Conf. Commun. Computing and Digital Systems (IEEE, 2023) 1–6. https://doi.org/10.1109/C-CODE58145.2023.10139859
16 F. Morselli, L. Bedogni, M. Fantoni, and U. Mirani: Proc. 2023 IEEE 9th World Forum Int. Things (IEEE, 2023) 1–6. https://doi.org/10.1109/WF-IoT58464.2023.10539523
17 S. Kumar, S. K. Chandra, R. N. Shukla, and L. Panigrahi: Proc. 2022 Int. Technol. Conf. Emerging Technol. for Sustainable Development (IEEE, 2023) 1–6. https://doi.org/10.1109/OTCON56053.2023.10114045
18 F. Salama, Z. Han, E. Korkan, S. KäEbisch, and S. Steinhorst: Proc. 2023 IEEE 9th World Forum Int. Things (IEEE, 2023) 1–7. https://doi.org/10.1109/WF-IoT58464.2023.10539381
19 R. Chaisricharoen, P. Temdee, C. Kamyod, S. Wicha, J.-M. Thiriet, and H. Yahoui: Proc. 2022 14th Int. Conf. Software, Knowledge, Information Management and Applications (IEEE, 2022) 266–269. https://doi.org/10.1109/SKIMA57145.2022.10029563
20 P. Vought: Django Made Easy: Build and deploy reliable Django Applications (Independently published, 2020) Chap. 4.
21 J. J. Garrett: https://designftw.mit.edu/lectures/apis/ajax_adaptive_path.pdf