electronics

MDPI

Article

An Internet of Things—Supervisory Control and Data Acquisition (IoT-SCADA) Architecture for Photovoltaic System Monitoring, Control, and Inspection in Real Time

Wei He, Mirza Jabbar Aziz Baig and Mohammad Tariq Iqbal *

Department of Electrical and Computer Engineering, Memorial University of Newfoundland, 230 Elizabeth Ave, St. John's, NL A1C5S7, Canada; weih@mun.ca (W.H.); mjabaig@mun.ca (M.J.A.B.)
* Correspondence: tariq@mun.ca

Abstract: The Internet of Things (IoT) serves as a key component to enhance operational efficiency and decision-making in the context of supervisory control and data acquisition (SCADA) systems. Featuring the improved system robustness and real-time parameters, including images of the load, a new design of SCADA system monitoring for a photovoltaic (PV) system based on dual IoT platforms is proposed in this paper. Two voltage sensors collect the voltages of the PV module and the battery, while three current sensors accumulate the current data from the PV module, the battery, and the load. ESP32-E assembles the data and then transmits them to the Arduino Cloud via MQTT for real-time display and ESP32-S3 via HTTP. The relay and the load are controlled by ESP32-E to turn ON/OFF based on the battery voltage as well. In addition, ESP32-S3 forwards the received data to ThingSpeak for advanced analysis, data storage, and real-time display via HTTP. The load images are also displayed on a camera web server built by ESP32-S3. Successfully monitoring for over 20 days, the proposed system demonstrated its robustness and versatility even during the downtime of the Arduino Cloud, with a one-day voltage measurement ranging to a maximum of 13 V and current ranging from zero amperes to 4.42 amperes. To add to this system, it incorporates visual load monitoring features, which are unseen in traditional systems.

Keywords: SCADA; Internet of Things; Arduino Cloud; ThingSpeak; solar energy; renewable energy

Academic Editor: Rui Castro

Received: 20 September 2024
Revised: 22 December 2024
Accepted: 24 December 2024
Published: 26 December 2024

Citation: He, W.; Baig, M.J.A.; Iqbal, M.T. An Internet of Things—Supervisory Control and Data Acquisition (IoT-SCADA) Architecture for Photovoltaic System Monitoring, Control, and Inspection in Real Time. *Electronics* **2025**, *14*, 42. https://doi.org/10.3390/electronics14010042

1. Introduction

The Internet of Things (IoT) is a technology that integrates objects with the internet, enabling them to collect, exchange, and act upon data. This interconnected network of devices—ranging from household appliances [1] and wearable technology [2] to industrial machinery [3,4] and smart city infrastructure [5,6]—facilitates seamless communication and automation. By leveraging advanced sensors, actuators, and connectivity technologies, the IoT enhances efficiency, improves decision-making, and fosters innovation across various sectors. The transformative potential of the IoT is the ability to create intelligent systems that optimize operations, enhance user experiences, and drive significant economic growth.

Supervisory control and data acquisition (SCADA) refers to a control system architecture that utilizes computers, communication networks, and graphical user interfaces to supervise machines and processes at a high level. A few taxonomies of SCADA components should be briefly introduced. The remote terminal unit (RTU) collects data and information from sensors and then forwards them to the main terminal unit (MTU). The

human machine interface (HMI) is responsible for visually presenting the data to human operators. Communication networks can be wired or wireless, providing communication services across the other SCADA system components.

During the evolution of SCADA systems, there are typically four stages: monolithic, distributed, networked, and IoT-based [7,8]. Using the proprietary communication protocol (wide area network, WAN), MTU and RTU must be from the same vendor to communicate with each other. Similarly in distributed SCADA, the communication server uses WAN to communicate with RTUs. However, local area networks (LANs) bridge the gap between the communication server and the multiple operation stations. Furthermore, in networked SCADA, the internet protocol (IP) replaces the proprietary protocols, allowing the SCADA system components to be from different vendors and separated geographically. Finally, IoT-based SCADA utilizes cloud computing, trying to extract insightful information from the collected data. Due to the advantages of flexibility, easy development, cost efficiency, and scalability, IoT-based SCADA systems are becoming popular.

However, new vulnerabilities emerge when the IoT is integrated into the traditional SCADA systems. The management of large-scale systems and cross-layer collaborations [9], significant required storage space, and data security, along with high power consumption [10] and big data analytics [11], are challenges to be overcome. Furthermore, system metrics are of great importance, such as bandwidth overload and latency, and rely on the cloud service providers [7]. Production loss could occur if the cloud service is down.

In this study, we proposed an open-source and low-power-consumption IoT-SCADA system that monitors the PV system at the MUN ECE laboratory. Two voltage sensors collect the PV module voltage and the battery voltage data, while three current sensors are responsible for fetching the PV module current, the battery current, and the lamp load current. ESP32-E assembles the voltage and current data, forwarding them to ESP32-S3 via HTTP and the Arduino Cloud platform via MQTT. Furthermore, ESP32-E controls the ON/OFF state of the relay, turning ON/OFF the load based on the battery voltage. In addition, ESP32-S3 transmits the received data to the ThingSpeak platform via HTTP and sends load images to the web server it built. The effectiveness of the proposed IoT-SCADA system was verified through experiment results. The novel contribution of this article includes the use of two IoT platforms that enhance the system robustness. Additionally, the designed system incorporates the feature of visual load status verification over the webserver.

The rest of the paper is organized as follows. Related works are reviewed in Section 2. Section 3 describes the overview of the system. Sections 4–6 present the system components, implementation methodology, and experimental setup. In Section 7, the experimental results are demonstrated and analyzed. Discussions of the designed system are covered in Section 8. Section 9 concludes this paper.

2. Related Work

In [12], an IoT-based SCADA system for supervising two connected microgrids was designed and implemented. The power sources of the load were changed among the local microgrid, neighbor microgrid, national power line, and diesel generator, regarding the energy production situation of the microgrids. The ESP32 microcontroller collected voltage, current, and temperature data from the corresponding sensors via I2C and sent them to the Cayenne Web Server cloud via an MQTT protocol over Wi-Fi connectivity. A dashboard showing the monitored system values and the power source was created by the Widgets application. The authors claimed that an efficient and reliable power sharing can be achieved under power shortages.

In [13], the authors developed a new approach using an improved artificial bee colony (IABC) algorithm based on IoT-based SCADA to achieve MPPT for a solar power plant. The duty ratio of the DC-DC converter was determined by the inputs of the PV panel voltage and current using IABC. After the initialization of random duty ratio values, each employed bee is evaluated for the fitness and searched nearby. Onlook bees then searched for the neighborhood of the positions of employed bees to find a higher fitness value. The employed bees became scout bees to find new solutions if the fitness cannot increase. Smart sensors published the PV voltage, PV power, and the duty ratio to ThingSpeak platform organized by a microcontroller, via MQTT. Subscribers of the topics can view the values on mobile devices and laptops. The authors claimed that the steady-state restoration times are improved by 89%, 87%, and 88% for the system's power, voltage, and duty cycle, better than incremental conductance and the cuckoo search method.

An IoT open-source platform for monitoring a 3 MW PV plant was proposed in [14]. A personal computer with Eclipse Kura installed was the IoT Gateway, while an Eclipse Kapua instance received the current and voltage data from the industrial devices via MQTT. Several integrations can easily extract the received data, such as ElasticSearch, MQTT broker, Kapua API, Kura API, and camel routes. The authors claimed that the interoperability of the proposed solution was proven by the adaptability of industrial protocols (Modbus, OPC, S7) and communication systems (Ethernet, Wi-Fi, Bluetooth).

A SCADA system for a PV plant based based on open-source software was designed in [15]. IoT-ready inverters sent the electrical parameters via Modbus TCP/IP, while gauges that are not IoT-ready sent environmental parameters via Modbus RS-485, both to the communication server. The communication server then processed and sent the data, via an MQTT protocol, to the application and database server, which run Emoncms dashboards. The authors claimed that the developed SCADA system using IoT devices achieved the easy customization of the existing supervisory system and increased the polling frequency.

A low-cost real-time PV monitoring system was developed using IoT architecture in [16]. Two data loggers both based on an ESP32 microcontroller were implemented: the first one collected meteorological data and the second collected PV generation data. The received transducer readings were first sent to the LoRa gateway (ESP32) via LoRa and then sent via MQTT to the Google Cloud Platform (GCP) MQTT broker. Hosted on Heroku, two webpages were built as the subscribers to display the real-time monitoring meteorological data and the PV generation history data, respectively. The authors claimed that some of the advantages of the proposed monitoring system were complete meteorological measurements, wireless communication, and data stored both locally and in the cloud.

The authors of [17] used a SCADA system to acquire operating parameters of a PV plant, to calculate its quality indicators for energy management. Meteorological parameters, including the solar irradiation, ambient temperature, PV module temperature, and wind direction, were collected by the weather station. Electrical parameters, including the power, current, and voltage of each array box, were collected from the inverters. The HMI was able to show the monitored parameters, along with the alarm status. Combined with the SUNTECH module characteristics, the authors calculated a series of performance indicators for the PV plant during 2019 and 2020. The performance ratio, the most important indicator for a PV plant, was 83.41% in 2019 and 81.24% in 2020, showing a 2.17% decrease and suggesting that the maintenance can be performed.

In [18], a solution for the monitoring and management of distributed PV systems using cloud infrastructure at IoT devices has been proposed. Analog sensors collected environmental parameters and the DC current/voltage of PV panels via the data acquisition level and then sent them to the process level. The implementation of the network infrastructure depended on the location of the PV installation. In remote areas, the cellular network and

satellite network were suitable for sending data to the data center. After inspections by the firewall, the data would be displayed in real time and stored for future use. Inverters were able to receive commands from the data center and run them in the under-excited condition to compensate for reactive power. The authors claimed that the remote controlling of PV systems' active power and reactive power was achieved.

In [19], the authors designed and implemented a wireless sensor network (WSN) and an IoT platform for a rooftop PV system. A current sensor, voltage sensor, temperature and humidity sensor, and optical dust sensor were deployed to collect key parameters of the PV system. The Arduino UNO microcontroller sent the data to the Node MCU through serial communication. Via a Wi-Fi network, the Node MCU transferred the data to Ubidots, a cloud server. Dashboards for displaying the collected data were created in Ubidots. Statistical analysis, including the average value in different intervals and maximum/minimum value, was also achieved by the platform.

Using the IoT architecture, the authors of [20] proposed an anomaly-detection methodology for the non-ideal operating conditions of PV plants. The irradiance, ambient temperature, PV module temperature, and output current/voltage/power from the inverter were collected by Raspberry Pi Zero and ADC1115. The parameters were then published to the MQTT broker and sent to dashboards, which served as the subscriber. The core of the anomaly-detection algorithm is whether the deviation from the mean value is larger than 2-sigma. If the short-window average value of power, voltage, and PV module temperature all deviated during the same period, non-ideal conditions were identified. The authors claimed that this algorithm only used real-time data, not requiring historical system data.

The inverter performance was monitored and analyzed in real time by SCADA Haiwell in [21]. Input parameters of the inverter, including the DC voltage/current/power, and output parameters, including the energy yield, AC power, and AC phase voltage/current, were collected by PLC via the Modbus TCP/IP protocol. The Haiwell Cloud also received the data via the Ethernet and created HMI to show all the collected data, using Haiwell HMI C7S. Verified by the measurements of the existing iSolarCloud system, the newly developed system was claimed to precisely monitor the inverter online via mobile phones or personal computers.

The authors of [22] proposed a smart embedded system to remote monitor a PV array and diagnose the faults via machine learning algorithms. The PV current, PV voltage, ambient temperature, PV cell temperature, and solar irradiance were collected by a NodeMCU ESP8266 module and then sent to an open-source IoT application Blynk for storage and remote display. Furthermore, these data were also sent to a Raspberry Pi 4 running fault detection, fault classification, and user notification. The authors claimed that the fault detection showed a 97.5% accuracy using multilayer perceptron neural networks, and the fault classification showed a 96.8% accuracy using a stacking ensemble algorithm.

A P2P-based method to protect the SCADA communication channels was proposed in [23]. Multiple copies of data are stored at different nodes in the network, and are is transmitted along multiple paths. Even if one node or one path fails, the data can still be transmitted. Achieved through data replication and path redundancy, the availability and the integrity of the system are improved when faults or attacks are present. Evaluated based on experiments, the discovery and recovery ratios of data were improved from above 60% to above 70%.

The design and implementation of a low-cost and open-source SCADA system using Thinger.IO were presented in [24]. The ESP32 Thing microcontroller collected the PV panel voltage, the PV panel current, and the battery voltage. The data were then sent to the Thinger.IO IoT platform running locally on a Raspberry Pi microcontroller. HMI dashboards were created on the platform to visualize the data and monitor remotely.

However, some drawbacks of the design exist. The overall power consumption of the SCADA system is as high as 5.0 W, excluding the wireless router power consumption, the system will cease to log and display data if the communication between the ESP32 Thing and the Raspberry Pi fails, and the Thinger.IO Raspberry Pi ISO Image is not free of charge.

In [25], the authors proposed an IoT-based SCADA system to monitor the PV system parameters. Node-RED running on the single board computer Banana Pi M4 Berry serves as the MTU to aggregate, display data, and control the load. However, the system design only allows local access to the server. Devices that are not on the same LAN fail to read the logged data. Furthermore, the designed system will stop logging and displaying the PV system parameters if the Node-RED ceases to function.

In [26–28], false data injection cyberattacks on smart power systems can be mitigated by using a remedial action scheme and multi-layer perception networks, long short-term memory, and a deep recurrent neural network, respectively. However, the risk of information and communication technology shutting down was not addressed.

Previously reviewed papers are part of the bibliography we composed to comprehensively study the latest research results. Table 1 lists the monitored parameter types and deployed IoT platforms in related papers. To the best of our knowledge, there is no IoT-based SCADA system where two IoT platforms are utilized for improved system robustness based on data redundancy, and graphic images are collected. To be specific, ESP32-S3 hosts a camera web server and forwards data to ThingSpeak via HTTP for analysis and real-time display, and ESP32-E collects data from the PV module, controls the load, and displays data in the Arduino Cloud via MQTT. While the images in this design are from the load, it can be easily transferred into other applications, such as the identification of dust on PV panels [29,30] and short-term weather forecasting based on sky imagery [31]. The main contributions of this newly proposed design are listed below.

Table 1. Types of monitored parameters and IoT platforms used in related papers.

Reference	Monitored Parameter Types	IoT Platforms
[12]	Voltage, current, temperature	Cayenne
[13]	Voltage, power, MQTT duty cycle	ThingSpeak
[14]	Voltage, current, power, solar irradiation	Eclipse Kapua
[15]	Voltage, current, power, energy, environmental measurements	Emoncms
[16]	Voltage, current, power, meteorological variables	Google Cloud Platform
[19]	Voltage, current, power, temperature and humidity, dust	Ubidots Cloud
[20]	Voltage, current, power, solar irradiance, Ambient and PV module temperature	Apache (web server)
[21]	Voltage, current, active/reactive power, grid frequency	Haiwell Cloud
[22]	Voltage, current, power, ambient and PV module temperature	Blynk IoT
[24]	Voltage, current, power	Thinger.IO
[25]	Voltage, current, power	Node-RED
This design	Voltage, current, power, graphic image	Arduino Cloud and ThingSpeak

- Real-time monitoring and control of a PV system are achieved using Internet of Things architecture. The integration of IoT platforms facilitates the system implementation by providing functions, such as data aggregation, communication security, and data-driven applications.
- The design using two IoT platforms increased the system robustness based on the data redundancy. When one platform shuts down its service by schedule or accident, the other platform can continue to function.
- Images of the load can be accessed on a web server, enabling the visual verification of the load status. Anomalies that might not be detectable through voltage or current sensors alone, such as a burned-out lamp, can be observed and detected. Visual surveillance also provides versatility regarding environmental changes around the lamp and intuitions about the status of multiple lamps.

3. System Descriptions

The conceptual representation of the developed IoT-based SCADA architecture in five layers, along with the schematic diagram of the PV system and the SCADA system, are presented in this section. In Figure 1, the perception layer, the network layer, the security layer, the middleware layer, and the application layer consist of the overall IoT architecture. Within the perception layer, an ESP32-E (FireBeetle 2 ESP32-E, DFRobot®, Shanghai, China) collects voltage readings and current readings from the respective sensors. Moreover, the ESP32-E sends the control signal to the actuator (a relay) for turning on or off the load. Another ESP32 series board ESP32-S3 (ESP32-S3-WROOM CAM Board, Freenove®, Shenzhen, China) builds a webserver to display the load images from a 2-Megapixel OV2640 camera mounted on the development board. The network layer, as the second layer, ensures that data packets are transmitted from the source IoT devices to other devices, central servers, or the cloud. ESP32-E forwards the collected sensor data to the Arduino Cloud using the MQTT protocol and to the ESP32-S3 webserver through HTTP. ESP32-S3 transmits the parsed sensor data from the webserver to ThingSpeak through HTTP, afterwards. The security layer, middleware layer, and the application layer are integrated and implemented by the Arduino Cloud and ThingSpeak platform. In the Arduino Cloud, the device authentication is achieved by matching the device login name and the device key generated by the platform. Similarly, Read/Write API keys are provided in ThingSpeak to grant read/write access to the ThingSpeak channel data. The two IoT platforms can also function as the middleware layer. The Arduino Cloud aggregates all the sensor data from ESP32-E, stores the historical sensor data, and serves as a bridge between the hardware devices and the application layer. In addition to these functions, ThingSpeak also features easy-to-implement data processing, such as the average, median, sum, and rounding. The final application layer consists of several applications, including dashboards, email alert, and integration with other software (MATLAB R2023a for ThingSpeak).

Figure 2 illustrates the developed system from the view of the SCADA system. Voltage sensors are placed in parallel with the PV panel and the battery. Current sensors are in series arrangements with the PV panel, the battery, and the load. A relay is also connected in series with the load, controlling its ON/OFF state. The sensors and the relay comprise the FIDs. The RTU (ESP32-E) collects the sensor data and controls the relay according to the battery voltage, where low voltage leads to the load being turned off. Additionally, the RTU sends the data to the MTU (ESP32-S3) and to the middleware (Arduino Cloud). The MTU forwards the received data to the middleware (ThingSpeak) and sends the images of the load to the webserver. Two middlewares both provide HMIs (dashboards) and data historians (online data storage).

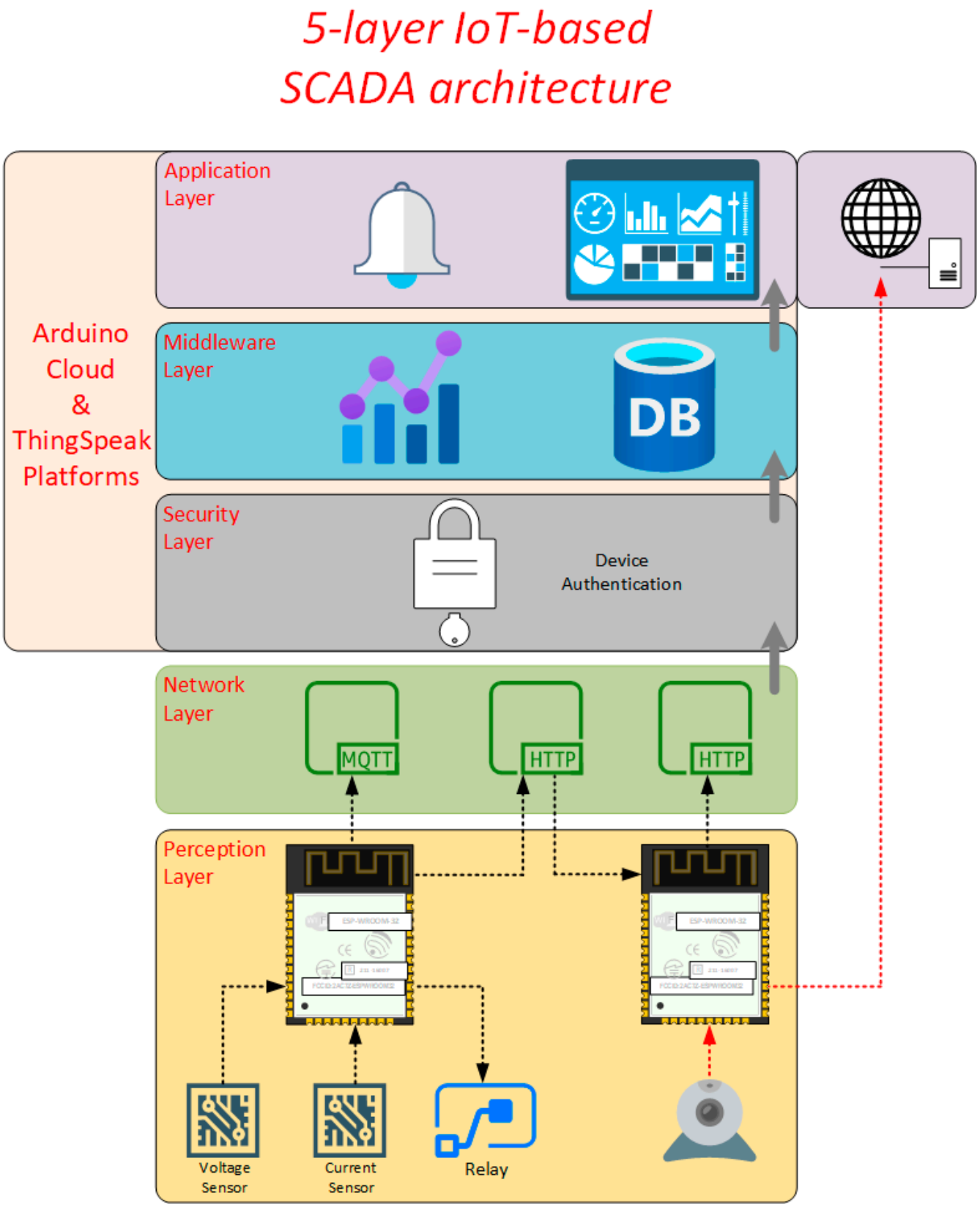

Figure 1. Proposed design of 5-layer IoT-based SCADA architecture.

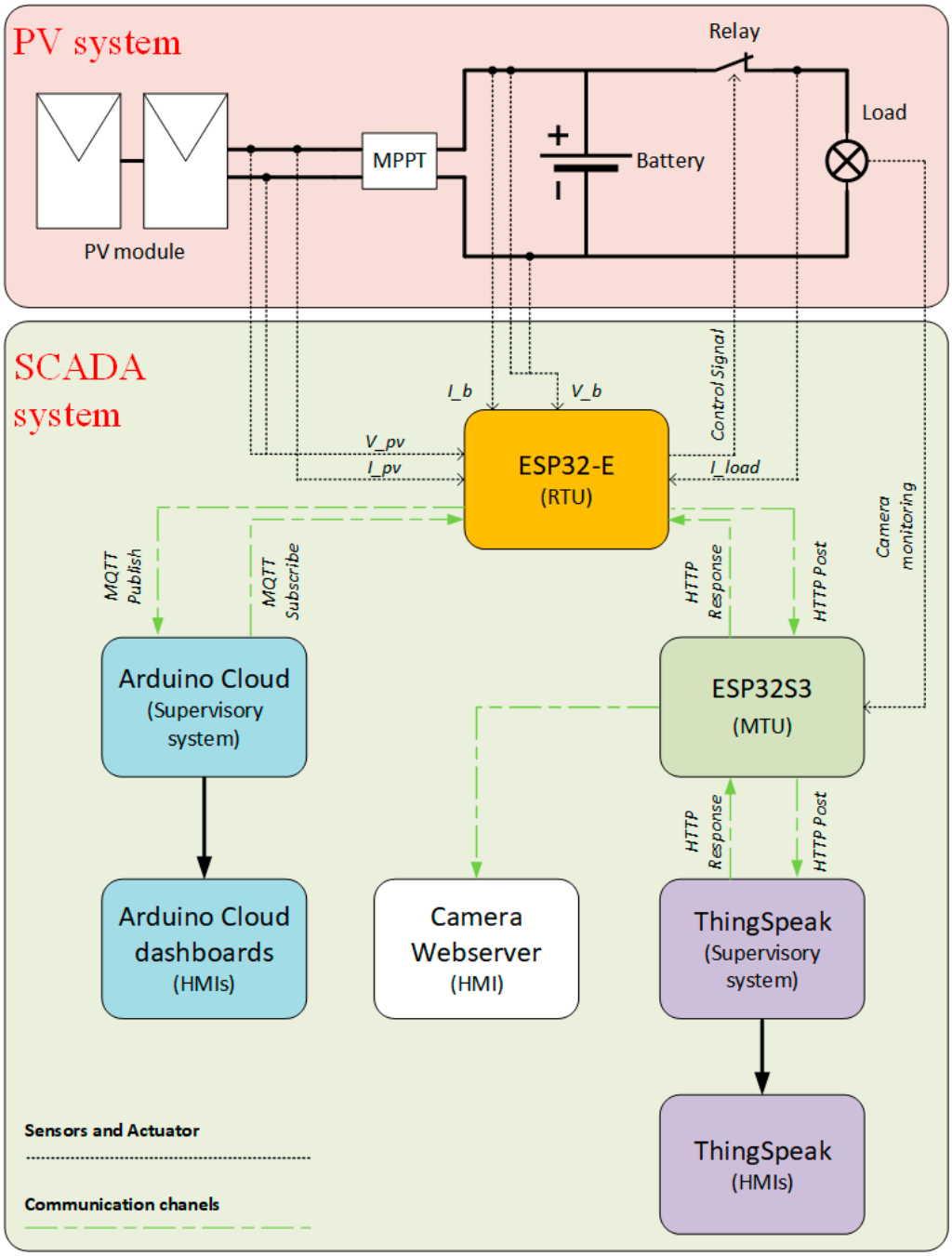

Figure 2. SCADA components of the proposed design.

4. SCADA System Components

In this section, we introduce the hardware and IoT platforms crucial to the system design. The ESP32-E as the MTU and part of the perception layer, controls the relay and collects the sensor data to forward them to the ESP32-S3 and the Arduino Cloud. The MTU, ESP32S3, captures the images of the monitored load and sends the sensor data to ThingSpeak. The current sensors, voltage sensors, the relay, and the camera comprise the perception layer, which serve as the FIDs.

4.1. ESP32-S3 and ESP32-E

The ESP32-S3-WROOM-1-N8R8 is a powerful microcontroller module that comprises the compact ESP32-S3-DEV-KIT-N8R8 development board. For hardware, equipped with an Xtensa® 32-bit LX7 dual-core processor(Cadence®, San Jose, CA, USA), the module has

a main frequency that can be up to 240 MHz [32]. PSRAM provides extra memory space to increase the usable ESP32 system memory, which is helpful for accelerating access to memory and expanding the buffer. The 8 MB PSRAM on the ESP32-S3 module allows for the seamless integration of the camera. It also has 512 kBSRAM, 384 kB ROM, 16 kB SRAM in RTC, and 8 MB of flash memory. Its working voltage is from 3.0 V to 3.6 V. Resourceful peripherals include up to 45 GPIOs, two 12-bit ADC, two I^2C/I^2S, one camera interface, four SPIs, one USB OTG, three UART interfaces, and so on; 2.4 GHz Wi-Fi (802.11 B/G/N) and Bluetooth® LE are adopted to enable wireless communication with superior RF performance. On the development board, USB and UART development can be achieved simultaneously by using onboard CH343 and CH334 chips. Excelling in its performance and versatility, ESP32-S3 is an excellent choice for developing IoT applications. The superior support for vector instructions also shows great potential for computer vision tasks, such as facial recognition, object detection, and image classification.

Another ESP32 module employed in this design is ESP32-WROOM-32E-N4, powering the FireBeetle 2 ESP32-E development board. Having an Xtensa® 32-bit LX6 dual-core processor, the module has a clock frequency up to 240 MHz [33]. It also has 448 kBROM, 520 kBSRAM, and 16 kB SRAM in RTC. The operating voltage is from 3.0 V to 3.6 V. As for wireless communication, it supports 2.4 GHz Wi-Fi (802.11 B/G/N) and Bluetooth V4.2 and BLE. Peripherals are supported, such as an SD card, UART, SPI, SDIO, I^2C, GPIOs, and so on. ESP32-WROOM-32E is more of a generic and powerful MCU module that can target various applications, including low-power sensor networks, when compared to ESP32-WROOM-S3.

4.2. Arduino Cloud

Developed by the Arduino® company(Somerville, MA, USA), the Arduino Cloud is an online open-source platform facilitating the creation, deployment, and monitoring of IoT projects. Key features of the platform are briefly introduced.

- Device Management: It allows users to manage multiple Arduino boards or third-party devices on one platform with a customized user interface.
- Security: To set up a new device in the platform, a device ID and a secret key are provided to realize the authentication process.
- Cloud Programming: Users can write, compile, and upload codes to the devices from the web browser, instead of installing a local programming IDE environment.
- Remote Control: Widgets that link to pre-defined variables can control the states of these variables.
- Over-the-Air Updates: Users can update the firmware of the devices over the air, without removing the field devices from their deployment place.
- Cloud Services Integration: The Arduino Cloud supports integration with other cloud services, such as Google Cloud, AWS, and IFTTT, allowing for complex and automated workflows.

Things handle the communication between IoT devices and the Arduino Cloud. The cloud variables, the associated device, network SSID and password, sketch, and metadata comprise Things. The Arduino Cloud free plan supports 2 Things, with 5 cloud variables for each Thing. Cloud data retention is 1 day on a rolling basis.

4.3. ThingSpeak

ThingSpeak is an IoT analytics platform that supports the aggregating, visualization, and analysis of real-time data streams in the cloud. Users can transmit data from their devices to ThingSpeak, generate immediate visualizations, and send alerts via web services. By incorporating MATLAB® analytics, ThingSpeak allows users to write and execute

MATLAB codes for data preprocessing, visualization, and analysis. The free plan supports 3 million messages per year, a 15 s updating interval, and 4 channels.

The user can read/write data from/to the channel, by using HTTP calls from the REST API or MQTT subscribe/publish method. After that, the MATLAB Analysis app helps to prepare, filter, and analyze the channel data. Calculating average values, eliminating data outliers, and replacing missing values in data are some common functions that the MATLAB Analysis app can realize. Users can then visualize the data and trigger an action, such as sending an email. Finally, MATLAB Add-On Toolboxes can be utilized to perform a specialized analysis, including prediction with data. Figure 3 is the ThingSpeak channel created on 5 June 2024, with more than 33,000 entries.

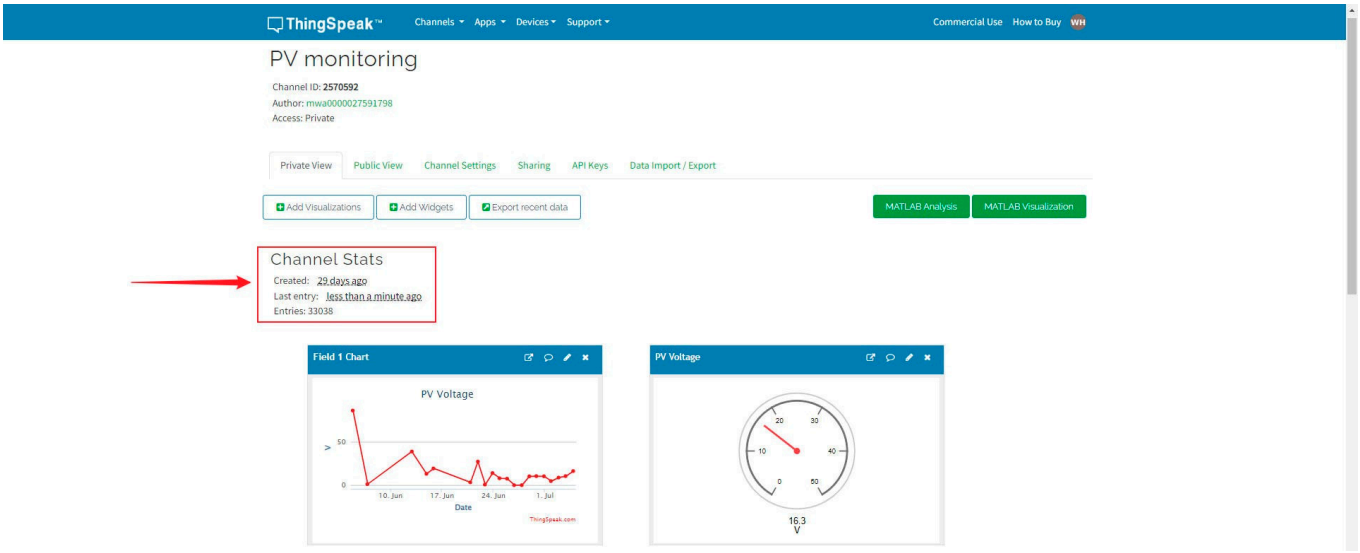

Figure 3. ThingSpeak channel private view.

4.4. Voltage Sensor and Current Sensors

The voltage sensor used in this design is suitable for 0–25 V DC voltage measuring. It is essentially a voltage divider that reduces the input voltage by a factor of 5 and generates a corresponding analog output voltage signal. Manufactured by HiLetgo®, Shenzhen, China, this is an open-source electronic device featuring a low price, down to about 1.7 USD per item [34].

A calibration process is needed to implement voltage measuring. One fifth of the source voltage divided by the ESP32-E reference voltage is equal to the voltage analog signal divided by 4096 (12-bit ADC for ESP32). The corresponding relationship between the measured voltage and ESP32-E analog voltage readings is given in Equation (1), where V_s is the source voltage to be measured, V_{ref} is the reference voltage for ESP32-E (3.26 V), and V_{sig} is the analog value read by the ESP32-E.

$$\frac{V_s/5}{V_{ref}} = \frac{V_{sig}}{4096}$$

$$V_s = \frac{5 \times V_{ref}}{4096} \times V_{sig} \quad (1)$$

The current sensor deployed in our implementation is the ACS712 current sensor [35] manufactured by Allegro Microsystems®, Manchester, NH, USA. This sensor is comprised of a linear current sensor IC based on Hall Effect, 2.1 kVRMS voltage isolation, and a low-resistance current conductor.

The ACS712 current sensor also requires calibration to make the results significant. When VCC and GND are connected to 5 V and ground, the OUT pin has an AcsOffset analog reading on ESP32-E if no current is flowing through the sensor. In addition, the sensitivity of ACS712-30A is 66 mV/A, meaning that every change of 1 A leads to a change of 66 mV on the OUT pin. Therefore, we have Equation (2) giving the correspondence between the measured current and the analog readings on ESP32-E, where sensitivity is 66 mV/A, AcsOffset is the analog reading on ESP32-E when the measured current is zero, and A_{sig} is the analog reading of the measured current, where the AcsOffset for a 5 V sensor is 2.5 V.

$$\frac{I_s \times \text{Sensitivity}}{V_{ref}} = \frac{A_{sig} - \text{AcsOffset}}{4096}$$

$$I_s = \frac{V_{ref}}{4096 \times \text{Sensitivity}}(A_{sig} - \text{AcsOffset}) \quad (2)$$

4.5. OV2640 Camera

The OV2640 is a small (1/4 inch) CMOS UXGA image sensor that is functioning as the combination of a single-chip UXGA (1600 × 1200) camera and an image processor [36]. The single-chip camera supports image sizes, including UXGA, SXGA (1280 × 1024), SVGA (800 × 600), and any size down to 40 × 30. The maximum image transfer rate is 15 frames per second at UXGA resolution. The S/N ratio and dynamic range are 40 dB and 50 dB, respectively. In addition, the image processor can be programmed to select the settings of resolution, frame rate, exposure control, white balance, gain control, noise reduction, and so on. In terms of the power supply and power consumption, the maximum voltage and power needed are 3.3 V and 140 mW. Having the powerful on-chip ISP with JPEG encoding and low power consumption, the OV2640 image sensor is popular for IoT projects. Figure 4 is the visual image of the Arducam® OV2640 camera module, with a DVP-friendly interface, which makes it easy to connect to microcontrollers.

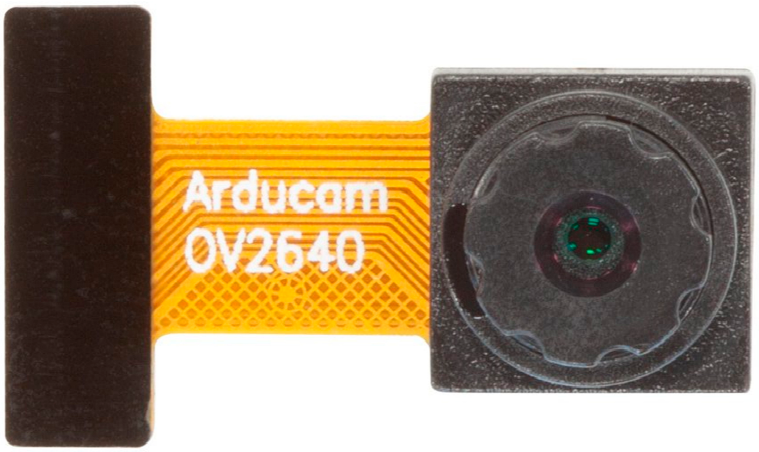

Figure 4. OV2640 camera module [37].

5. Implementation Methodology

In the developed IoT-based SCADA system, one microcontroller (ESP32-E) sends parameters of the monitored PV system, which are collected by sensors, to another microcontroller (ESP32-S3) and two IoT platforms (Arduino Cloud and ThingSpeak). Furthermore, ESP32-E also controls the relay to turn ON/OFF the load, while ESP32-S3 sends load images to the web server it built.

Figures 5 and 6 describe the sketch flowchart for ESP32-E and ESP32-S3, respectively. Two processes are executed in parallel in ESP32-E: one is translating sensor readings to

usable sensor values, controlling the relay, and sending sensor values to ESP32-S3, the other is to update the sensor values to the Arduino Cloud regularly. This asynchronous programming ensures that multiple tasks will run concurrently without blocking each other, leading to an improved system efficiency. Similarly, in ESP32-S3, there are also Process A and Process B that are executed simultaneously. Process A consists of two steps: Web server A listens to port 81 for incoming HTTP POST requests sent by ESP32-E, posting sensor data to the ThingSpeak platform. Meanwhile, Process B is that web server B monitors default port 80 for any HTTP GET requests for the latest image from a web browser. The use of different port numbers can separate services to avoid conflicts. Algorithm 1 is the pseudocode flashed into ESP32-E by the Arduino Cloud. Furthermore, Algorithm 2 is the pseudocode running on ESP32-S3, which is uploaded by Arduino IDE (version 2.3.2).

Algorithm 1: Data acquisition, automatic control, and data communication by ESP32-E.

Initialization;

1. Include libraries for Arduino Cloud, HTTP client, and Arduino Cloud connection handler;
2. Define constants, including device login name, device key, Wi-Fi network SSID and password, server URL, thresholds, intervals, and counter for averaging;
3. Define cloud variables including PV panel voltage/current, battery voltage/current, and the load current;
4. Declare sensor calibration functions;

Setup Function (executed only once);

5. Initialize cloud properties with read permissions and update intervals;
6. Connect to Arduino Cloud;

Loop Function (executed repeatedly);

7. Call a callback function to update cloud variables at predefined update intervals to Arduino Cloud;
8. Get current time;

If *interval_1 (30 s) has passed* **then**

9. Calculate average values (dividing the accumulated values by counter) of cloud variables;

If *battery voltage is below threshold_1* **then**

10. Cut the load by turning off the relay;

Else If *battery voltage is above threshold_2* **then**

11. Power the load;

Else:

12. Continue;

End

13. Begin HTTP client;
14. Send values of cloud variables to server (ESP32-S3) via HTTP POST request;
15. Get HTTP response code;
16. Reset counter;

Else If *interval_2 (one second) has passed* **then**

17. Read and accumulate values of the cloud variables;
18. Increment counter;

Else:

19. Continue;

End

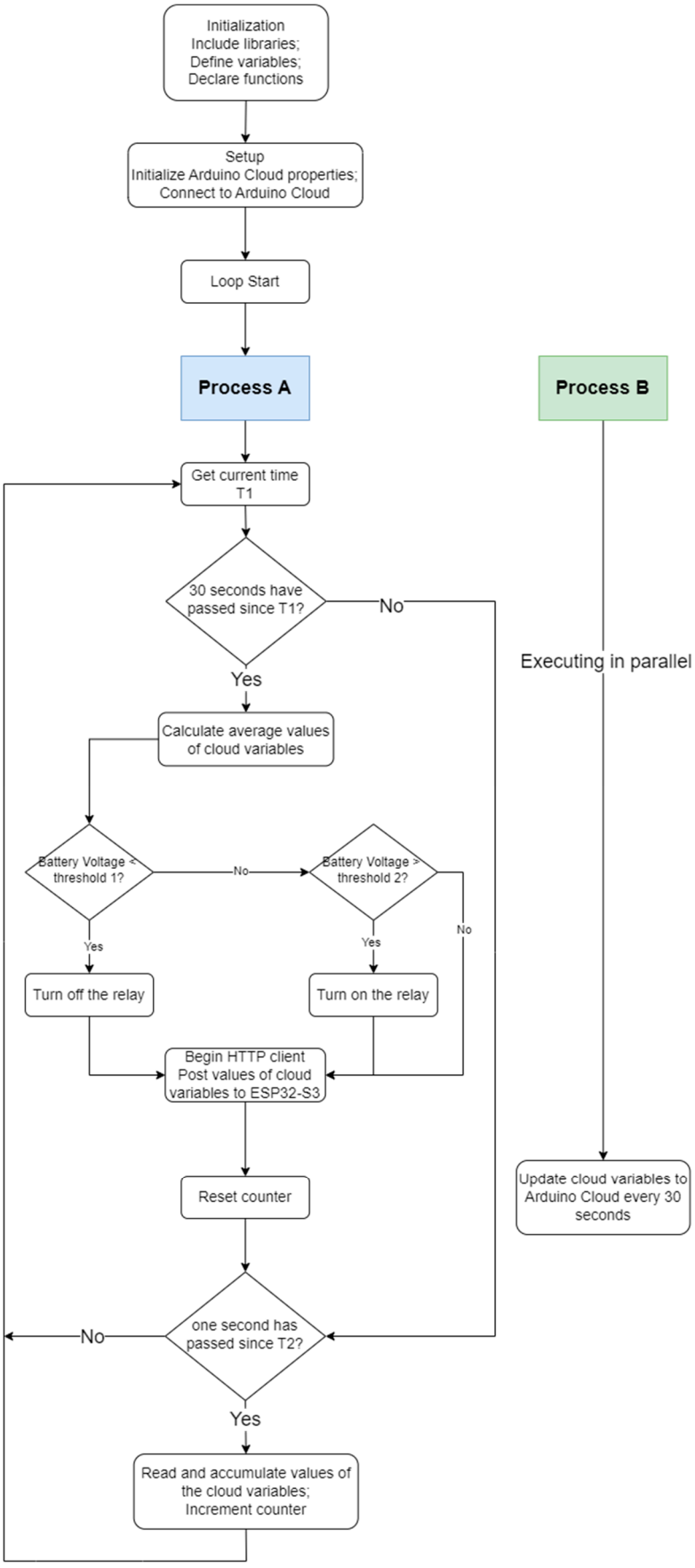

Figure 5. Flowchart of programs on ESP32-E.

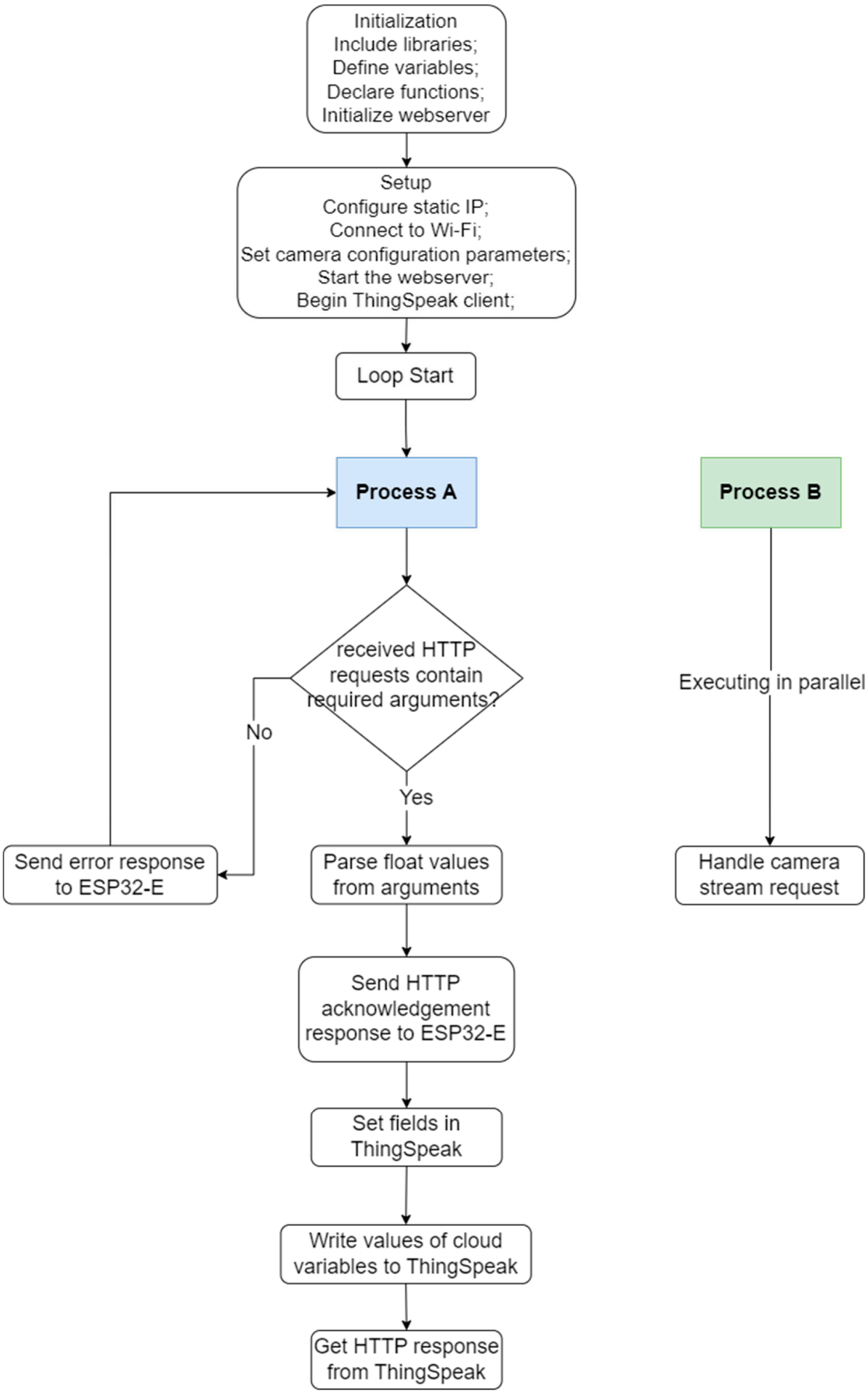

Figure 6. Flowchart of programs on ESP32-S3.

Algorithm 2: Data communication, and camera web server by ESP32-S3.

Initialization;

1. Include libraries for ThingSpeak, Wi-Fi, Webserver, and ESP Camera;
2. Define constants, including static IP, ThingSpeak channel number and write API key, Wi-Fi network SSID and password, and ThingSpeak client;
3. Define cloud variables including PV panel voltage/current, battery voltage/current, and the load current;
4. Initialize webserver on port 81;
5. Declare the function for starting the camera server, and the callback function for handling float-type HTTP requests;

Setup Function (executed only once);

6. Configure static IP;
7. Connect to Wi-Fi network;
8. Set camera configuration parameters based on the model;
9. Start the webserver for camera on port 80, and for receiving sensor data from ESP32-E on port 81;
10. Begin ThingSpeak client;

Loop Function (executed repeatedly);

11. Handle incoming client (ESP32-E) POST requests to the server;

Handle function;

If *received requests contain required arguments (such as "pv_voltage")* **then**

12. Parse and convert the arguments to float values;
13. Send HTTP acknowledgement response to the client (ESP32-E);
14. Set fields in ThingSpeak;
15. Write values of cloud variables to ThingSpeak;
16. Get HTTP response from ThingSpeak;

Else

17. Send error response to the client (ESP32-E);

End

6. Experimental Setup

This section explains the details of the experiments carried out to prove the effectiveness of the IoT-SCADA system design. At Memorial University ECE laboratory, the IoT-SCADA system is integrated into the installed PV modules for monitoring and control. The PV module contains two panels, with each one producing 130 W at 17.1 V under ideal conditions.

Figure 7 demonstrates the overview of the designed system setup. Positive and negative power terminals of the PV modules on the rooftop are connected to the digital solar charge controllers, the batteries and the load that are placed indoor, through red and black power wires.

Figure 8 presents the hardware schematic diagram of the IoT-based SCADA system. The voltage signal output pin VS of the PV voltage sensor is connected to the SENSOR VN pin of ESP32-E; meanwhile, the VS pin of the battery voltage sensor is connected to the SENSOR VP pin. GPIO15, GPIO34, and GPIO35 of ESP32-E are connected to the voltage output pins of the battery current sensor, the PV module current sensor, and the load current sensor. Additionally, the DVP interface is used to connect the OV2640 camera module to ESP32-S3. The interface consists of several signal lines. To be specific, 8-bit data lines (D2-D9) carry pixel data from the camera to the microcontroller; vertical synchronization (VSYNC) and horizontal reference (HREF) indicate the start of a new frame and a new row

of pixels, respectively; pixel clock (PLCK) is responsible for providing a clock signal when data lines have valid data; external clock (XCLK) is an input clock signal provided to the camera by the microcontroller to drive its internal logic; serial clock (SIOC) synchronizes the data transfer over the I^2C bus, while serial data (SIOD) enables bi-directional data transfer (camera configuration commands and settings) between the master ESP32-S3 and the slave camera. Figure 9 is the implementation of the hardware schematic diagram.

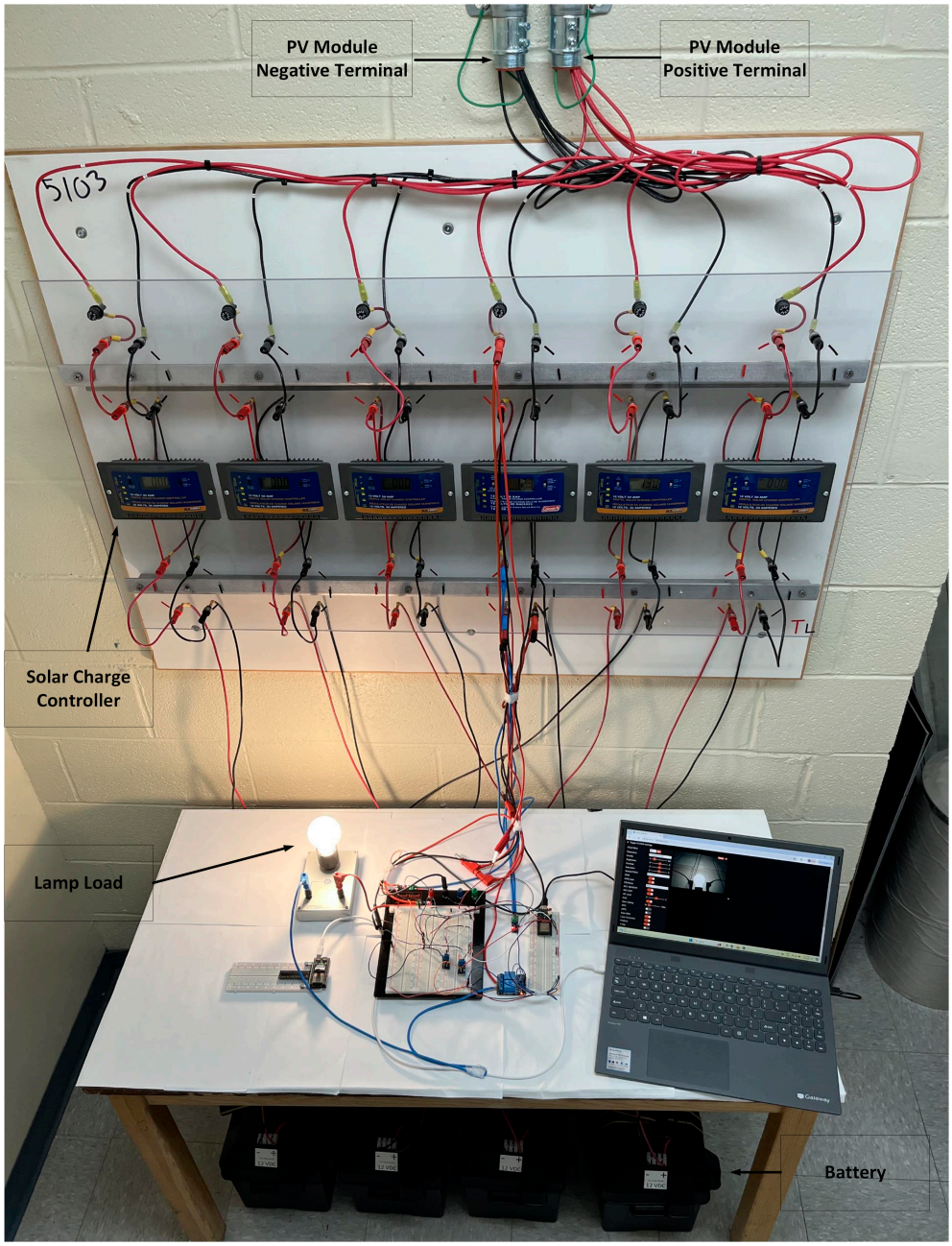

Figure 7. Overview of the experimental setup.

Despite the hardware experimental setup, the application layer, which is implemented in the Arduino Cloud, ThingSpeak, and the camera web server, is also introduced.

The Arduino Cloud provides different widgets in the dashboard application, including Switch, Push Button, Slider, Chart, etc. By simply linking the cloud variables to the widgets, the values of the cloud variables can be shown in the dashboard without much effort. In this setup, the Chart widget is chosen to display the changing values. Different time scales

of the data are available in Chart, which are 15/7/1 day(s), 1 h, and live. However, with the free plan, the maximum length of data displayed is one day. If 15 days are selected, it will only present the one-day data 15 days ago, instead of displaying all 15-day data.

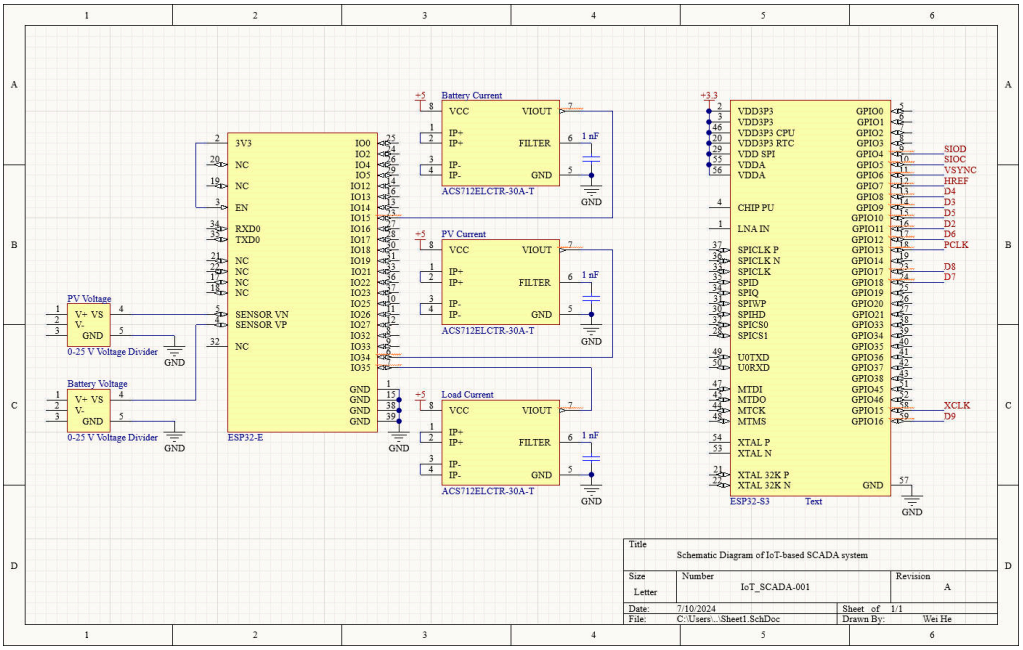

Figure 8. Hardware schematic diagram.

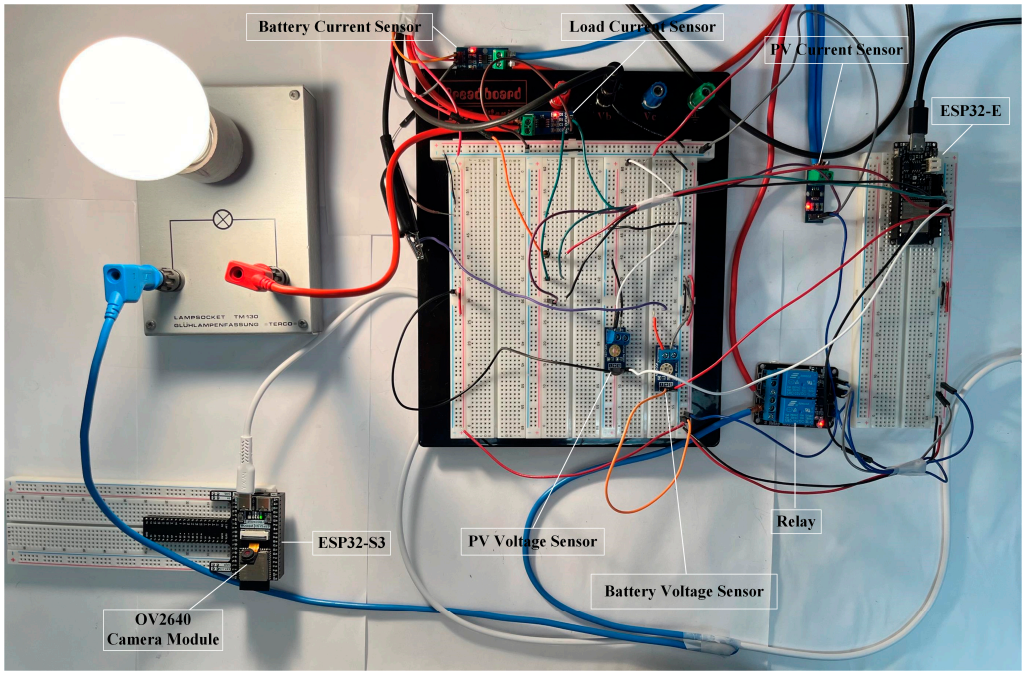

Figure 9. Hardware setup of IoT-SCADA system.

Another function is the email alert by ThingSpeak. Users can first read the ThingSpeak channel data, using the *thingSpeakRead* function. The Channel ID, read API key, numbers of data points, and field number in the channel should be configured in the function. Furthermore, an email alert can be configured with the HTTP POST method and triggered when conditions are met. To achieve this, Headers that include the user-specific ThingSpeak Alerts API key and Body parameters containing the monitored value are sent to the

URL https://api.thingspeak.com/alerts/send by using *webwrite* function. To enable the automatic alert function, the TimeControl application in ThingSpeak helps to trigger an action at a specific time. The recurrence can be down to the minute level. By combining the TimeControl application and the email alert function, the battery voltage LOW condition can be alerted to the email account in no more than five minutes. The Matlab program can be found in Figure A1 in the Appendix A.

Finally, the camera web server is built on ESP32-S3. By typing the configured ESP32-S3 IP address with a default port number 80 in a web browser, the camera web server processes the HTTP request from the browser and sends back a response to the browser. Figure 10 shows the camera web server interface. On the left side there is a settings column, providing options to alter the OV2640 settings. The clock frequency, image resolution/brightness/contrast, H-Mirror, V-Filp, etc. are subject to the users' preferences.

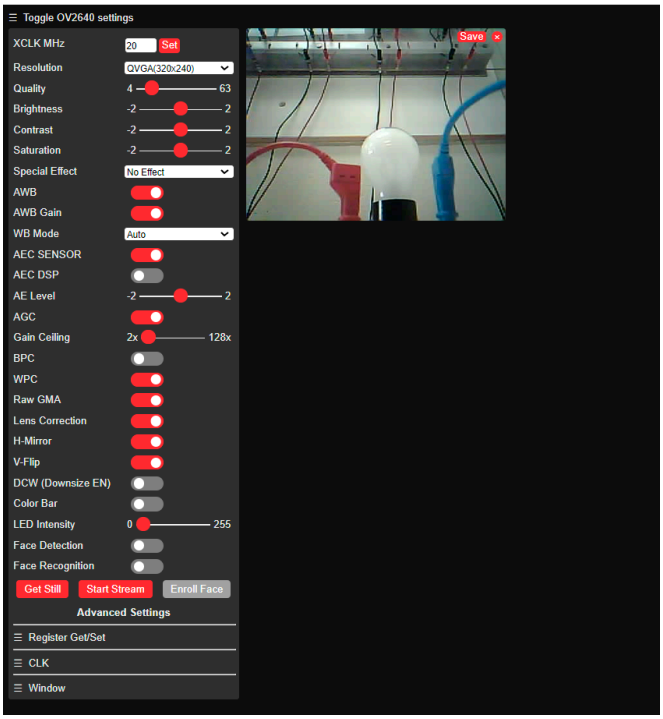

Figure 10. Camera webserver interface.

7. Results

From 23 June 2024, to 13 July 2024, 20-day results were recorded to validate the correct performance of the developed IoT-SCADA system. On 1 July, there was a scheduled maintenance for the Arduino Cloud from 08:17 UTC to 12:00 UTC, 2 July [38]. During that time, no data were recorded on the Arduino Cloud platform since the device connectivity was cut off. However, ThingSpeak received, stored, and displayed the data on 1 July as normal, without being affected. Based on the design of data redundancy, our developed system successfully maintained its function, even when the Arduino Cloud shut down the service.

Figure 11 shows the dashboards displaying the recorded PV system parameters on 11 July and 12 July on the Arduino Cloud. Five digital meters (five Velleman DVM890L) were utilized to measure the PV system parameters simultaneously, the readings of which matched the IoT-SCADA system results. A $\pm 0.5\%$ accuracy for the DC voltage, ranging from 200 mV to 200 V, and $\pm 0.8\%$ accuracy for the DC current ranging, from 2 mA to 200 mA ($\pm 2\%$ for 20 A) provide reliable measurement results. Furthermore, the validation process based on the commercially available measurement instrument DI-145 was covered

in our previous paper [39]. To avoid repetitiveness, the details of the data verification are omitted in this paper. On 11 July, the PV module voltage dropped from 16.25 V to 0, at around 20:18 and 21:00. This is due to the decrease in the solar irradiance at St. John's, Canada. The PV voltage remained at zero until the next morning. At 04:50 on 12 July, the PV voltage started to increase due to the sunrise, to a value of 16.3 V at 07:13. The battery voltage was decreasing until 05:46, due to the self-discharge current. After that, the battery voltage and the PV module current increased rapidly, showcasing that the PV module was charging the battery. The PV current entered a stable stage, which also kept the battery voltage invariant. There are two peaks for the PV current, one is for charging the battery and the other is for charging the load.

Figure 11. Arduino Cloud dashboards on 11 and 12 July, with the 1D time scale.

To test if the relay can be controlled by ESP32-E, after 17:38 on 12 July, the load and the relay were integrated into the system. The relay was designed to turn on when the battery voltage is larger 13.0 V and turn off below 12.5 V. Figure 12 is the 1H time scale data to present the details. The load current varied between 4.42 A to 0, corresponding to the ON/OFF state of the relay. Due to the relay mechanism, the battery voltage was oscillating between around 13.1 V and 12.3 V. The battery current was negative, indicating that it was charging the load.

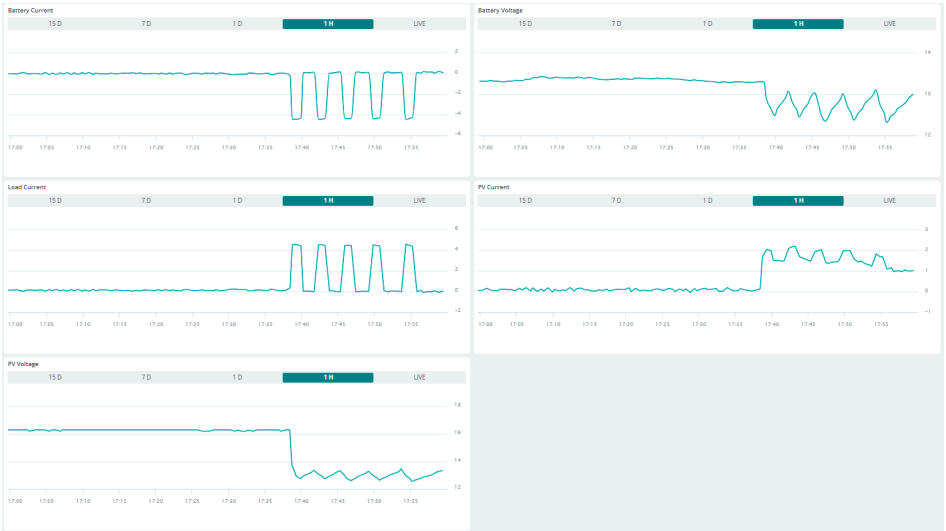

Figure 12. Arduino Cloud dashboards on 12 July, with a 1H time scale.

Furthermore, the data logging results were also displayed on ThingSpeak dashboards in Figure 13. The readings were presented in real time in the right column. A clear periodic pattern of the PV voltage and the battery voltage can be observed. The battery voltage was 12.55 V, which is larger than 12.5 V. The relay was still cut off, since the battery voltage must be larger than 13.0 V to turn it on. The relay state did not change when the battery voltage fell within 12.5 V and 13.0 V.

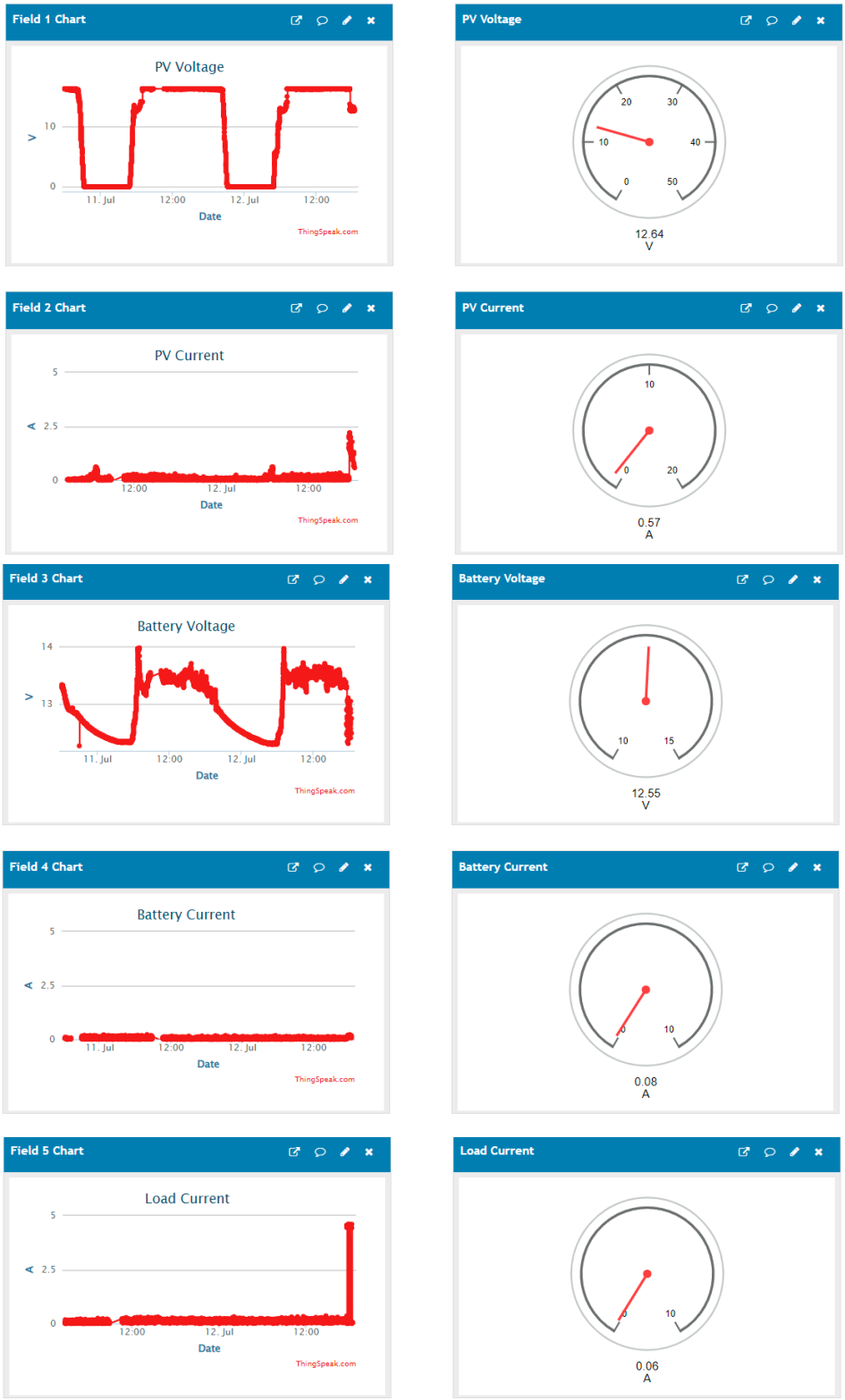

Figure 13. ThingSpeak dashboards displaying PV system parameters.

Figure 14 shows the lamp load to be turned on when the battery voltage is larger than 13.0 V.

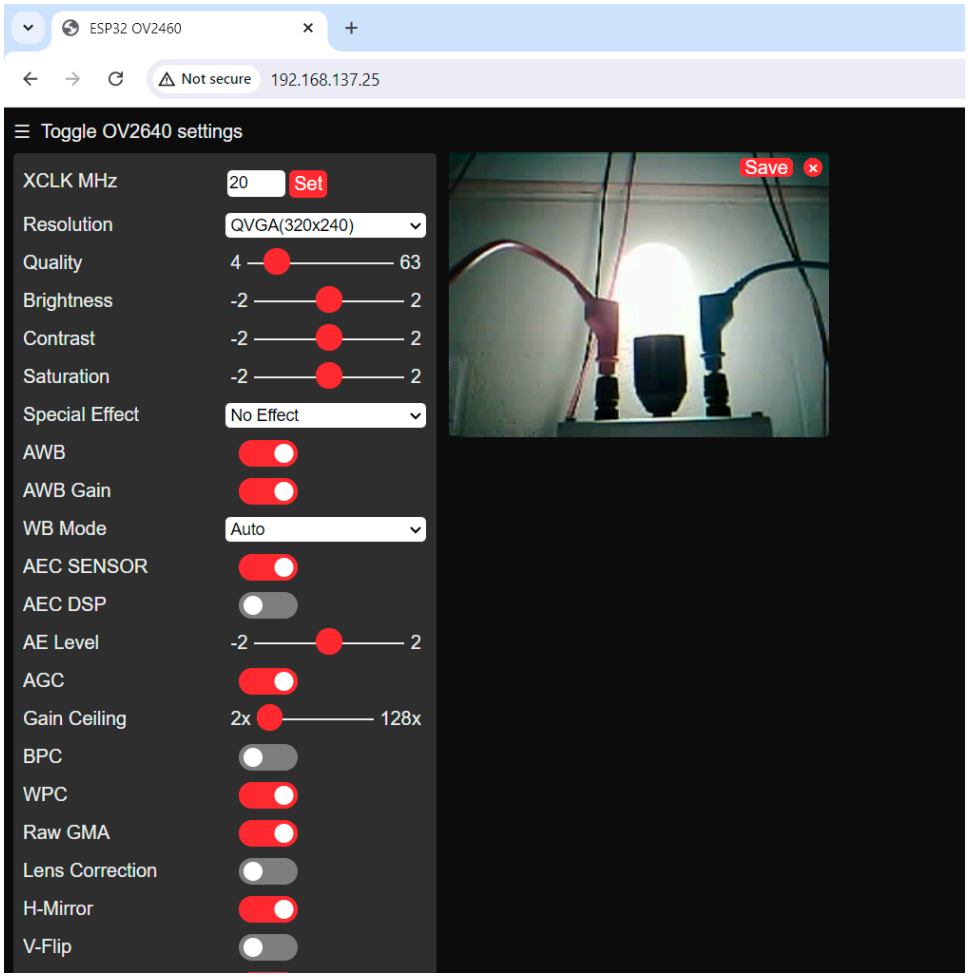

Figure 14. Load when battery voltage is larger than 13.0 V.

8. Discussion

Several features of the proposed IoT-SCADA system are listed below, providing that our system design is practical and innovative.

- **Integration of dual IoT Platforms:** The Arduino Cloud and ThingSpeak are two of the most popular IoT platforms, which facilitate easy and powerful IoT integration into SCADA systems. The communication layer, security layer, and application layer can all be implemented on the two platforms. The Arduino Cloud offers exclusive benefits over ThingSpeak, including the cloud programming environment, device management, and OTA updates. Comparatively, ThingSpeak provides advanced data analytics, customizable dashboards, integration with MATLAB, and powerful plugins. Both cloud services retain some data that could be accessed later. Using two cloud services at the same time adds highly desirable redundancy and diversity features to the design. This design also has great potential to develop new features in the future with the development and updates of the two IoT platforms.
- **Dual Microcontrollers:** ESP32-E focuses on collecting the PV system parameters and control, which requires electrical connections with the PV system circuit through sensors and the relay. Meanwhile, ESP32-S3 is not involved in the physical connection with the PV system but handles image collection and data transmission. This separation of duties improves the system's reliability at the hardware level. Furthermore,

the use of dual microcontrollers provides flexibility in monitoring tasks. While it is possible to mount the camera module on the microcontroller that collects the system parameters, this setup would significantly limit the range of inspection objects. This limitation arises from the need to align both the system's electrical ports and the inspection area simultaneously. For instance, if the inspected lamp load (or a PV panel in other scenarios) is located outdoors and the electrical ports are indoors, a single microcontroller would be insufficient to meet these requirements.

- **Real-time Monitoring and HMI design:** The Arduino Cloud and ThingSpeak receive voltage and current data every 30 s. Fast responses to incidents can be made to prevent potential losses. Furthermore, the two platforms provide customizable and easy-to-use dashboards for HMI.
- **Data Redundancy:** When the Arduino Cloud was offline, the IoT-SCADA system continued to log data on ThingSpeak, which ensures the overall system reliability and accessibility.
- **Image-based Load Monitoring:** A low-cost camera web server allows users to capture images of the load. Visual surveillance ensures that no lamp is burned out and the surrounding environment remains safe, and it provides more intuitive feedback than electrical parameters alone. By observing the load image, operators obtain awareness of the load status without being onsite, saving on unnecessary technical service calls. Live feedback is a great tool to remotely oversee the monitored system.
- **Automatic Control and alert:** A control method is employed in ESP32-E that controls the relay and the load locally. This operation protects the battery from over-discharging automatically. This novel design also has an auto low battery alert though email, which is a great tool to avoid battery dead discharge and extend the battery life.
- **System Security:** Devices must provide their corresponding keys assigned by the Arduino Cloud to connect to it. Moreover, ThingSpeak requires the channel read/write API key to allow the read/write operation. This method guarantees that only authenticated devices have access to the platforms.
- **Cloud Data Storage:** In our design, 5 messages are sent to ThingSpeak every 30 s. Since ThingSpeak supports 3 million messages per year free of charge, it can store up to 208 days of data.
- **Open-source:** IoT platforms, microcontrollers, and actuators are all open source. Free software guides and hardware at a low price are available on the Internet and the market. This removes the barrier of replicating this design, facilitating its promotion.
- **Low power consumption:** The average power consumptions of the ESP32-S3 with the camera and ESP32-E during working conditions are 0.92 W and 0.81 W, respectively. The total power consumption of the system is merely 2.38 W.

9. Conclusions

The proposed IoT-SCADA system design comprised a collection of sensors and the relay, ESP32-E and ESP32-S3 as the RTU and the MTU, various communication protocols, including HTTP and MQTT, and the Arduino Cloud with ThingSpeak as the IoT platforms. This design is demonstrated to have exceptional performance through experimental verification. The PV system was continuously monitored for 20 days, showcasing the robustness of the developed IoT-SCADA system even when Arduino Cloud was offline during 1 July 2024. Real-time monitoring of the PV system parameters, including the PV module voltage and current, the battery voltage and current, and the load current, was achieved on both the Arduino Cloud and ThingSpeak platforms. The load images were also available on the camera server built on ESP32-S3. On 11 and 12 July, the recorded parameters were displayed and analyzed. The periodic pattern of the parameters can be observed on a

1D time scale. When the load was turned on, the detailed changes of system parameters were analyzed on a 1H time scale. Additionally, ESP32-E also controls the load to turn off when the battery voltage is below 12.5 V, ensuring the safety of the battery. The email reporting the battery status was also received from ThingSpeak. The complete system realization proves its robustness, effectiveness, versatility, and precision against its rigorous design. Moreover, the use of two IoT platforms enhances systems robustness and the visual load surveillance, and verification of the load status detects the anomalies that may not be identified in traditional systems using only voltage and current sensors.

10. Future Work

This design lays a solid foundation for the development of advanced fault diagnosis for PV systems using artificial intelligence (AI). Future research could explore the integration of AI and computer vision algorithms to analyze the data collected by the camera, enabling predictive maintenance, real-time fault detection, and automatic alerts to the end user.

Author Contributions: W.H.: system design, experimental results, and writing of the manuscript. M.J.A.B. Supervised the research as a whole, review and revised the manuscript based on reviewers' comments. M.T.I.: Supervised, contributed research ideas throughout the research, funding acquisition, provided the required resources, and components. All authors have read and agreed to the published version of the manuscript.

Funding: This research was funded by the Natural Sciences and Engineering Research Council of Canada (NSERC) and the School of Graduate Studies (SGS) at Memorial University.

Data Availability Statement: Data are contained within the article and references.

Conflicts of Interest: The authors declare that there is no conflict of interest regarding the publication of this paper.

Abbreviations

The following abbreviations are used in this manuscript:

SCADA	Supervisory Control and Data Acquisition
RTUs	Remote Terminal Units
MTUs	Master Terminal Units
FIDs	Field Instrumentation Devices
HMI	Human–Machine Interface
IoT	Internet of Things
GPIO	General-purpose Input/Output
IDE	Integrated Development Environment
PV	Photovoltaic
MPPT	Maximum Power Point Tracker
RAM	Random Access Memory
UART	Universal Asynchronous Receiver Transmitter Pins
MQTT	Message Queuing Telemetry Transport
HTTP	Hypertext Transfer Protocol

Appendix A

```
% Store the channel ID for the PV monitoring channel.
channelID = 2570592;
readAPIKey = 'GW5DUUKETI09WJTU';

% Provide the ThingSpeak alerts API key.  All alerts API keys start with TAK.
alertApiKey = 'TAKzIUH98ogbRUuzpzu';

% Set the address for the HTTTP call
alertUrl="https://api.thingspeak.com/alerts/send";

% webwrite uses weboptions to add required headers.  Alerts needs a ThingSpeak-Alerts-API-Key header.
options = weboptions("HeaderFields", ["ThingSpeak-Alerts-API-Key", alertApiKey ]);

% Set the email subject.
alertSubject = sprintf("Low Battery Voltage");

% Read the recent data.
BV_Data = thingSpeakRead(channelID,'ReadKey',readAPIKey,'NumPoints',30,'Fields',3);

% Check to make sure the data was read correctly from the channel.
if isempty(BV_Data)
    alertBody = ' No data read from battery. ';
else
    % Get the most recent point in the array of battery voltage data.
    lastBV_Value = BV_Data(end);
    lastBV_Value

    % Set the outgoing message
    if (lastBV_Value <= 12.5)
        alertBody = ' The battery voltage is low! ';

    else (lastBV_Value > 12.5)
        alertBody = 'The battery voltage is normal. ';
    end
end

 % Catch errors so the MATLAB code does not disable a TimeControl if it fails
try
    webwrite(alertUrl , "body", alertBody, "subject", alertSubject, options);
catch someException
    fprintf("Failed to send alert: %s\n", someException.message);
end
```

Figure A1. MATLAB program for email alert when the battery voltage is below 12.5 V.

References

1. Aheleroff, S.; Xu, X.; Lu, Y.; Aristizabal, M.; Pablo Velásquez, J.; Joa, B.; Valencia, Y. IoT-enabled smart appliances under industry 4.0: A case study. *Adv. Eng. Inform.* **2020**, *43*, 101043. [CrossRef]
2. Stavropoulos, T.G.; Papastergiou, A.; Mpaltadoros, L.; Nikolopoulos, S.; Kompatsiaris, I. IoT Wearable Sensors and Devices in Elderly Care: A Literature Review. *Sensors* **2020**, *20*, 2826. [CrossRef]
3. Franco, J.; Aris, A.; Canberk, B.; Uluagac, A.S. A Survey of Honeypots and Honeynets for Internet of Things, Industrial Internet of Things, and Cyber-Physical Systems. *IEEE Commun. Surv. Tutor.* **2021**, *23*, 2351–2383. [CrossRef]
4. Wang, Q.; Zhu, X.; Ni, Y.; Gu, L.; Zhu, H. Blockchain for the IoT and industrial IoT: A review. *Internet Things* **2020**, *10*, 100081. [CrossRef]
5. Rejeb, A.; Rejeb, K.; Simske, S.; Treiblmaier, H.; Zailani, S. The big picture on the internet of things and the smart city: A review of what we know and what we need to know. *Internet Things* **2022**, *19*, 100565. [CrossRef]
6. Mohamed, R.; Behiri, M.A.M.; Mohammed, J. Al shammri, Hegazy Rezk. Energy Performance Analysis of On-Grid Solar Photovoltaic System-a Practical Case Study. *Int. J. Renew. Energy Res. IJRER* **2019**, *9*, 1292–1301. [CrossRef]
7. Yadav, G.; Paul, K. Architecture and security of SCADA systems: A review. *Int. J. Crit. Infrastruct. Prot.* **2021**, *34*, 100433. [CrossRef]
8. Alanazi, M.; Mahmood, A.; Chowdhury, M.J.M. SCADA vulnerabilities and attacks: A review of the state-of-the-art and open issues. *Comput. Secur.* **2023**, *125*, 103028. [CrossRef]
9. Sajid, A.; Abbas, H.; Saleem, K. Cloud-Assisted IoT-Based SCADA Systems Security: A Review of the State of the Art and Future Challenges. *IEEE Access* **2016**, *4*, 1375–1384. [CrossRef]
10. Babayigit, B.; Abubaker, M. Industrial Internet of Things: A Review of Improvements Over Traditional SCADA Systems for Industrial Automation. *IEEE Syst. J.* **2024**, *18*, 120–133. [CrossRef]

11. Pliatsios, D.; Sarigiannidis, P.; Lagkas, T.; Sarigiannidis, A.G. A Survey on SCADA Systems: Secure Protocols, Incidents, Threats and Tactics. *IEEE Commun. Surv. Tutor.* **2020**, *22*, 1942–1976. [CrossRef]
12. Duair, J.J.; Majeed, A.I.; Ali, G.M. Design and Implementation of IoT-Based SCADA for a Multi Microgrid System. *ECS Trans.* **2022**, *107*, 17345. [CrossRef]
13. Alhasnawi, B.N.; Jasim, B.H.; Alhasnawi, A.N.; Sedhom, B.E.; Jasim, A.M.; Khalili, A.; Bureš, V.; Burgio, A.; Siano, P. A Novel Approach to Achieve MPPT for Photovoltaic System Based SCADA. *Energies* **2022**, *15*, 8480. [CrossRef]
14. de Arquer Fernández, P.; Fernández Fernández, M.Á.; Carús Candás, J.L.; Arboleya Arboleya, P. An IoT open source platform for photovoltaic plants supervision. *Int. J. Electr. Power Energy Syst.* **2021**, *125*, 106540. [CrossRef]
15. Silva, F.M.Q.; Filho, B.J.C.; Pires, I.A.; Maia, T.A.C. Design of a SCADA System Based on Open-Source Tools. In Proceedings of the 2021 14th IEEE International Conference on Industry Applications (INDUSCON), Sao Paulo, Brazil, 15–18 August 2021; pp. 1323–1328.
16. Melo, G.C.; Torres, I.C.; Araújo, Í.B.; Brito, D.B.; Barboza, E.D. A Low-Cost IoT System for Real-Time Monitoring of Climatic Variables and Photovoltaic Generation for Smart Grid Application. *Sensors* **2021**, *21*, 3293. [CrossRef]
17. Hoarcă, I.C. Energy management for a photovoltaic power plant based on SCADA system. In Proceedings of the 2021 13th International Conference on Electronics, Computers and Artificial Intelligence (ECAI), Pitesti, Romania, 1–3 July 2021; pp. 1–9.
18. Vujović, I.; Mladen, K.; Željko, Đ. Centralized controlling of distributed PV systems using cloud and IoT technologies. *Telfor J.* **2023**, *15*, 6. [CrossRef]
19. Sarkar, S.; Rao, K.U.; Bhargav, J.; Sheshaprasad, S.; C.A, A.S. IoT Based Wireless Sensor Network (WSN) for Condition Monitoring of Low Power Rooftop PV Panels. In Proceedings of the 2019 IEEE 4th International Conference on Condition Assessment Techniques in Electrical Systems (CATCON), Piscataway, NJ, USA, 21–23 November 2019; pp. 1–5.
20. Dupont, I.M.; Carvalho, P.C.M.; Jucá, S.C.S.; Neto, J.S.P. Novel methodology for detecting non-ideal operating conditions for grid-connected photovoltaic plants using Internet of Things architecture. *Energy Convers. Manag.* **2019**, *200*, 112078. [CrossRef]
21. Seflahir, D.; Ahmad Faisal Mohamad, A.; Aliashim, A.; Abdurahman; Nurul, H.; Rivaldi, K.; Ojak Abdul, R. Real-time Analysis of Inverter Performance via SCADA Haiwell Online Monitoring. *J. Adv. Res. Appl. Sci. Eng. Technol.* **2024**, *37*, 99–114. [CrossRef]
22. Mellit, A.; Benghanem, M.; Kalogirou, S.; Massi Pavan, A. An embedded system for remote monitoring and fault diagnosis of photovoltaic arrays using machine learning and the internet of things. *Renew. Energy* **2023**, *208*, 399–408. [CrossRef]
23. Khelil, A.; Germanus, D.; Suri, N. Protection of SCADA Communication Channels. In *Critical Infrastructure Protection: Information Infrastructure Models, Analysis, and Defense*; Lopez, J., Setola, R., Wolthusen, S.D., Eds.; Springer: Berlin/Heidelberg, Germany, 2012; pp. 177–196.
24. Aghenta, L.O.; Iqbal, M.T. Low-Cost, Open Source IoT-Based SCADA System Design Using Thinger.IO and ESP32 Thing. *Electronics* **2019**, *8*, 822. [CrossRef]
25. He, W.; Baig, M.J.; Iqbal, M.T. An Open-Source Supervisory Control and Data Acquisition Architecture for Photovoltaic System Monitoring Using ESP32, Banana Pi M4, and Node-RED. *Energies* **2024**, *17*, 2295. [CrossRef]
26. Naderi, E.; Asrari, A. A Deep Learning Framework to Identify Remedial Action Schemes Against False Data Injection Cyberattacks Targeting Smart Power Systems. *IEEE Trans. Ind. Inform.* **2024**, *20*, 1208–1219. [CrossRef]
27. Naderi, E.; Aydeger, A.; Asrari, A. Detection of False Data Injection Cyberattacks Targeting Smart Transmission/Distribution Networks. In Proceedings of the 2022 IEEE Conference on Technologies for Sustainability (SusTech), Virtual, 21–23 April 2022; pp. 224–229.
28. Naderi, E.; Asrari, A. Toward Detecting Cyberattacks Targeting Modern Power Grids: A Deep Learning Framework. In Proceedings of the 2022 IEEE World AI IoT Congress (AIIoT), Seattle, WA, USA, 6–9 June 2022; pp. 357–363.
29. Cui, Y.; Liu, M.; Li, W.; Lian, J.; Yao, Y.; Gao, X.; Yu, L.; Wang, T.; Li, Y.; Yin, J. An exploratory framework to identify dust on photovoltaic panels in offshore floating solar power stations. *Energy* **2024**, *307*, 132559. [CrossRef]
30. Yang, M.; Javed, W.; Guo, B.; Ji, J. Estimating PV Soiling Loss Using Panel Images and a Feature-Based Regression Model. *IEEE J. Photovolt.* **2024**, *14*, 661–668. [CrossRef]
31. David, M.; Alonso-Montesinos, J.; Le Gal La Salle, J.; Lauret, P. Probabilistic Solar Forecasts as a Binary Event Using a Sky Camera. *Energies* **2023**, *16*, 7125. [CrossRef]
32. Espressif Systems. ESP32-S3-WROOM-1 ESP32-S3-WROOM-1U Datasheet. Available online: https://www.espressif.com/sites/default/files/documentation/esp32-s3-wroom-1_wroom-1u_datasheet_en.pdf (accessed on 3 July 2024).
33. Espressif Systems. ESP32-WROOM-32E ESP32-WROOM-32UE Datasheet. Available online: https://www.espressif.com/sites/default/files/documentation/esp32-wroom-32e_esp32-wroom-32ue_datasheet_en.pdf (accessed on 4 July 2024).
34. Geekstory Voltage Tester Sensor Measurement Detection Module DC 0-25V Terminal Sensor Module for Arduino UNO Mega Robot Smart Car Geekstory (Pack of 10). Available online: https://www.amazon.ca/Voltage-Measurement-Detection-Arduino-Geekstory/dp/B07FVVSYYH (accessed on 5 June 2024).
35. SparkFun Electronics. ACS712 Datasheet. Available online: https://www.sparkfun.com/datasheets/BreakoutBoards/0712.pdf (accessed on 5 July 2024).

36. OmniVision. OV2640 Color CMOS UXGA (2.0 MegaPixel) CameraChip with OminiPixel2 Technology. Available online: https://www.uctronics.com/download/cam_module/OV2640DS.pdf (accessed on 5 July 2024).
37. ArduCam. Arducam OV2640 Camera Module, 2MP Mini CCM Compact Camera Modules Compatible with Arduino ESP32 ESP8266 Development Board with DVP 24 Pin Interface. Available online: https://www.arducam.com/product/arducam-ov2640-camera-module-2mp-mini-ccm-compact-camera-modules-compatible-with-arduino_m0031esp32-esp8266-development-board-with-dvp-24-pin-interface_/ (accessed on 5 July 2024).
38. Arduino Cloud. Scheduled Maintenance for Arduino Cloud. Available online: https://status.arduino.cc/incidents/3zbncwd7k7vl (accessed on 12 July 2024).
39. He, W.; Iqbal, M.T. Power Consumption Minimization of a Low-Cost IoT Data Logger for Photovoltaic System. *J. Electron. Electr. Eng*. **2023**, *2*, 241–261. [CrossRef]