IA 3204-Data Acquisition Systems (21/22)

Faculty of technology

Universuty of Colombo

Multi-Channel Voltage/Current Data Logger

Name: R.L.A.T.P.M.Rajakaruna

Index: T01343

Date: 25/ 01/ 2026

Introduction

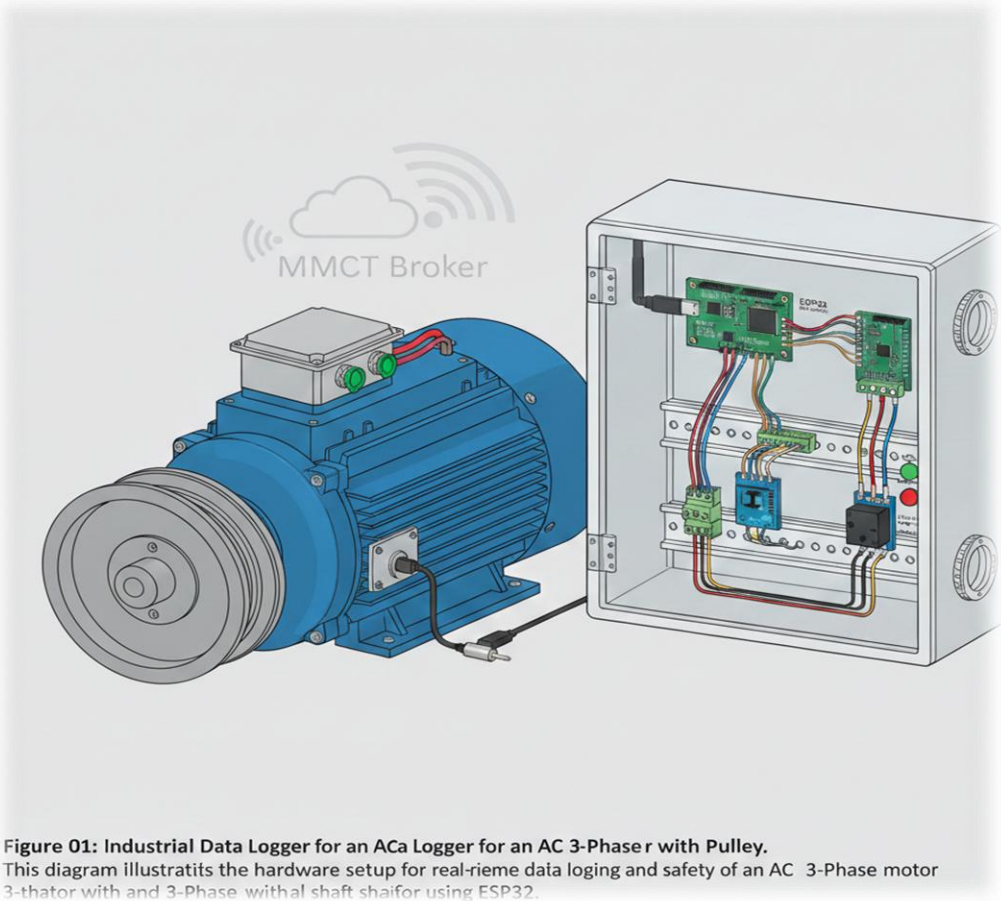

Figure 01: Industrial Data Logger for an ACa Logger for an AC 3-Phase r with Pulley.
This diagram illustratits the hardware setup for real-rieme data loging and safety of an AC 3-Phase motor
3-thator with and 3-Phase withal shaft shaifor using ESP32.

figure 01: Boiler system SCADA.

This is an industrial-grade **Smart Data Logger** designed for real-time monitoring of electrical systems and AC motors. The system acquires critical data such as **Voltage, Current, Power, and Temperature** using an ESP32 microcontroller. These parameters are transmitted via the **MQTT protocol** to a customized Python-based GUI dashboard for visualization and logging. For enhanced safety, the system features an automated protection logic that deactivates the load through a relay if current or temperature thresholds are exceeded, ensuring the protection of the connected industrial equipment.

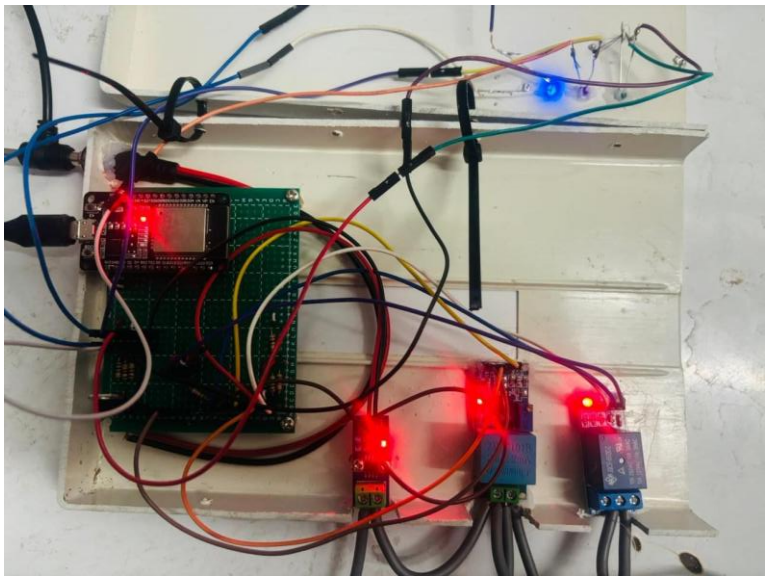

figure 02: Project Hardware.

There are three main analog inputs in this system: the Voltage sensor (ZMPT101B), Current sensor(ACS712), and Temperature sensor (10k NTC Thermistor). All of these provide analog signals to the ESP32's ADC (Analog to Digital Converter) pins.

The system features several main outputs, including a **relay module** for controlling industrial hardware and **indicating LEDs** for status monitoring. A green LED indicates normal operation, while a red LED signals a safety fault. Furthermore, the system outputs live telemetry data such as **voltage, current, and temperature** to a digital dashboard via MQTT for real-time logging.

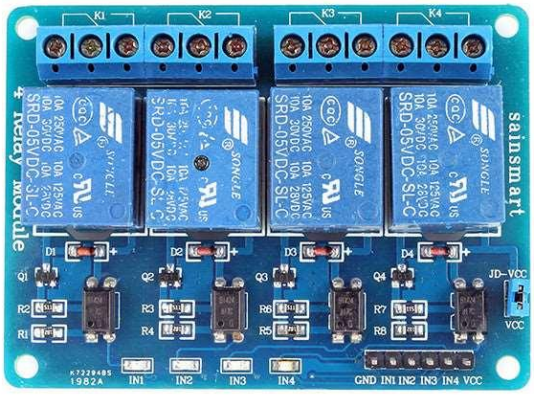

figure 03: Relay module.

https://www.sainsmart.com/products/4-channel-5v-relay-module?pr_prod_strat=e5_desc&pr_rec_id=8267493c5&pr_rec_pid=11091673300&pr_ref_pid=11091673364&pr_seq=uniform

Microprocessor

I used an ESP32 as the microcontroller for my project. Analog pins 32, 33, and 35 are used to receive inputs from the voltage, current, and temperature sensors. Digital pins 26 and 27 are used to control the status LEDs, while digital pin 25 is used to trigger the relay module. The ESP32 also uses its built-in Wi-Fi for MQTT communication. I used an external 12V power adapter with a voltage regulator to provide a stable power supply to the ESP32 and the sensors.

That is the wiring diagram of my project.

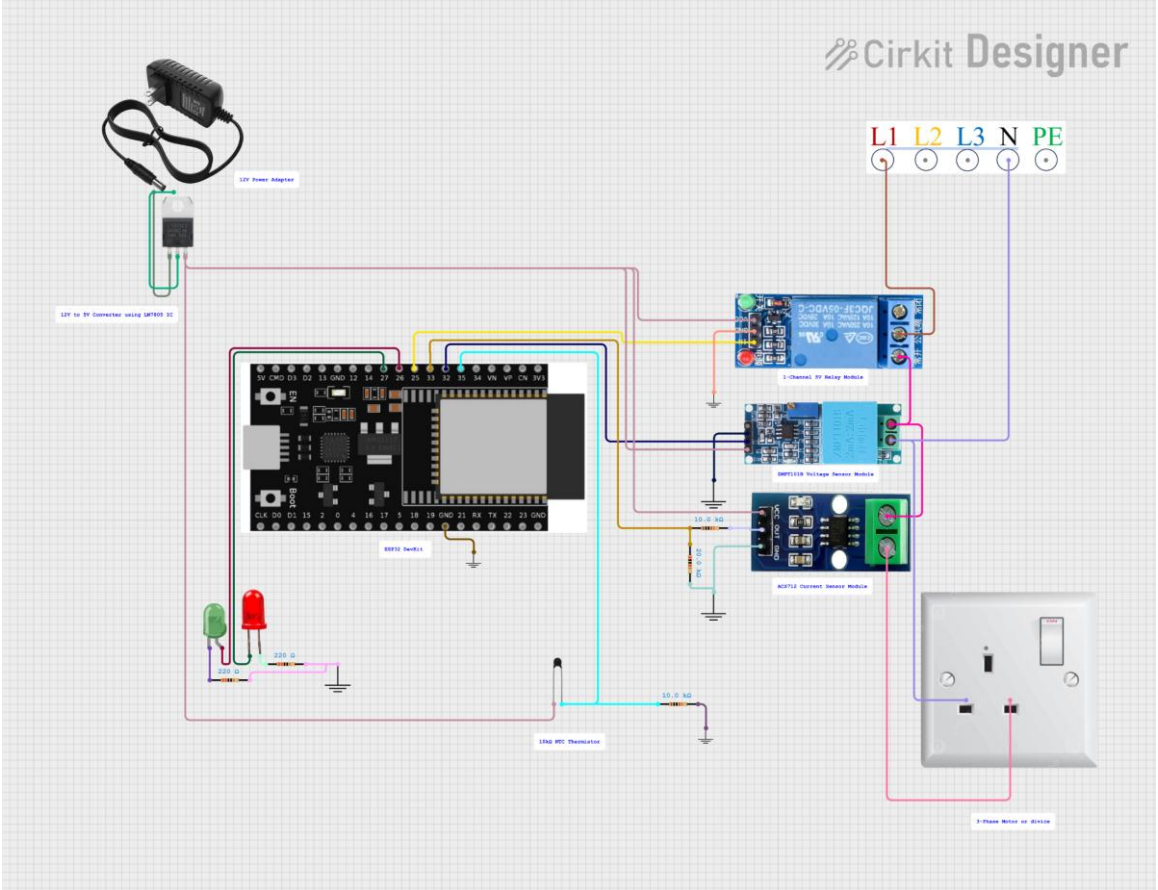

figure 04: Circuit diagram.

https://app.cirkitdesigner.com/project/c687fc0e-c4d7-459a-9a27-11c67bcd1ca5

Communication protocol

I used MQTT (Message Queuing Telemetry Transport) as my communication protocol. It is a lightweight messaging protocol ideal for IoT applications. I used the PubSubClient.h library in Arduino IDE to establish communication between the ESP32 and the broker. For the MQTT broker, I used broker.hivemq.com. The system publishes sensor data (voltage, current, etc.) to a specific "sensor topic" and subscribes to a "command topic" to receive ON/OFF instructions from the Python GUI dashboard.

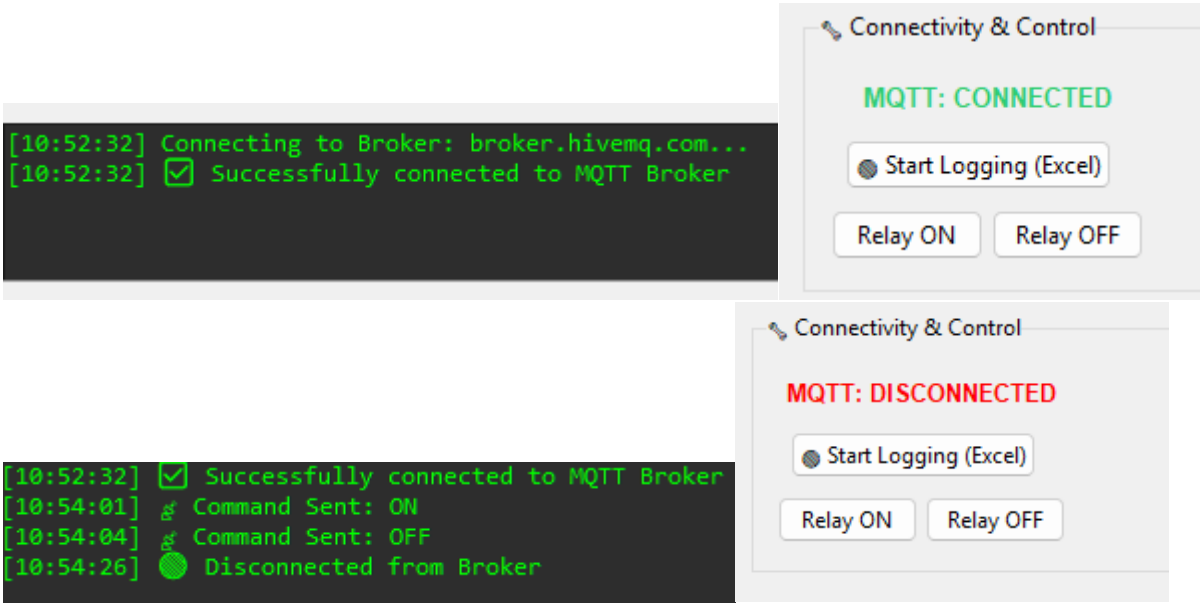

Figure 05: Verifying MQTT communication status on Python GUI dashboard (Connected and Disconnected states).

MQTT Broker Configuration

I used broker.hivemq.com as the central cloud bridge. The configuration in my Python code is as follows:

- Broker: broker.hivemq.com
- Port: 1883 (Standard MQTT port)
- Sensor Topic: tharusha/esp32/sensors (To receive Voltage, Current, and Temp)
- Command Topic: tharusha/esp32/commands (To send Relay ON/OFF signals)

2. Communication Testing and Verification

To ensure the system is reliable, I implemented "Callback" functions to monitor the connection status.

Figure 06: Verification of Connectivity and Control

As shown in Figure 06, the system was tested for two main states:

- Connected State: When the Python app connects to the broker, the status label turns green and displays "MQTT: CONNECTED". The console logs confirm successful data synchronization.
- Disconnected State (Fault Handling): If the internet connection fails, the on_disconnect function is triggered. The status label immediately turns red and displays "MQTT: DISCONNECTED", informing the user of the communication failure.

Data Logging and Storage

The ESP32 Smart IoT Monitor dashboard includes a built-in feature to record and store system performance data. By clicking the "Start Logging" button, the system automatically saves all live data into a CSV file.

```
Timestamp,Voltage,Current,Power,Temp,Relay,Status
2026-01-25 21:33:18,240.5,0.0,0.0,24.9,ON,NORMAL
2026-01-25 21:33:19,198.8,0.21,41.8,25.0,ON,NORMAL
2026-01-25 21:33:20,233.6,0.0,0.0,25.0,ON,NORMAL
2026-01-25 21:33:21,233.7,0.259,60.5,24.9,ON,NORMAL
2026-01-25 21:33:22,227.1,0.0,0.0,24.8,ON,NORMAL
2026-01-25 21:33:23,239.3,0.224,53.6,24.9,ON,NORMAL
2026-01-25 21:33:24,237.7,0.259,61.6,24.9,ON,NORMAL
2026-01-25 21:33:25,236.9,0.231,54.7,24.9,ON,NORMAL
2026-01-25 21:33:26,240.3,0.273,65.6,25.0,ON,NORMAL
2026-01-25 21:33:27,194.1,0.231,44.8,24.9,ON,NORMAL
2026-01-25 21:33:28,233.6,0.245,57.2,24.9,ON,NORMAL
2026-01-25 21:33:29,240.1,0.21,50.4,24.9,ON,NORMAL
2026-01-25 21:33:30,194.9,0.0,0.0,24.9,ON,NORMAL
2026-01-25 21:33:31,239.6,0.224,53.7,24.9,ON,NORMAL
2026-01-25 21:33:32,239.2,0.21,50.2,24.9,ON,NORMAL
2026-01-25 21:33:33,193.0,0.224,43.2,24.9,ON,NORMAL
2026-01-25 21:33:34,238.0,0.224,53.3,24.9,ON,NORMAL
2026-01-25 21:33:35,235.5,0.203,47.8,24.9,ON,NORMAL
2026-01-25 21:33:36,212.2,0.203,43.1,24.9,ON,NORMAL
2026-01-25 21:33:37,236.3,0.0,0.0,24.8,ON,NORMAL
2026-01-25 21:33:39,237.5,0.224,53.2,24.7,ON,NORMAL
2026-01-25 21:33:40,240.5,0.245,58.9,24.7,ON,NORMAL
2026-01-25 21:33:40,238.0,0.238,56.7,24.7,ON,NORMAL
2026-01-25 21:33:41,237.1,0.0,0.0,24.8,ON,NORMAL
2026-01-25 21:33:42,233.7,0.0,0.0,24.8,ON,NORMAL
```

Figure 06: Sensor data saved in a CSV log file.

Communication Architecture

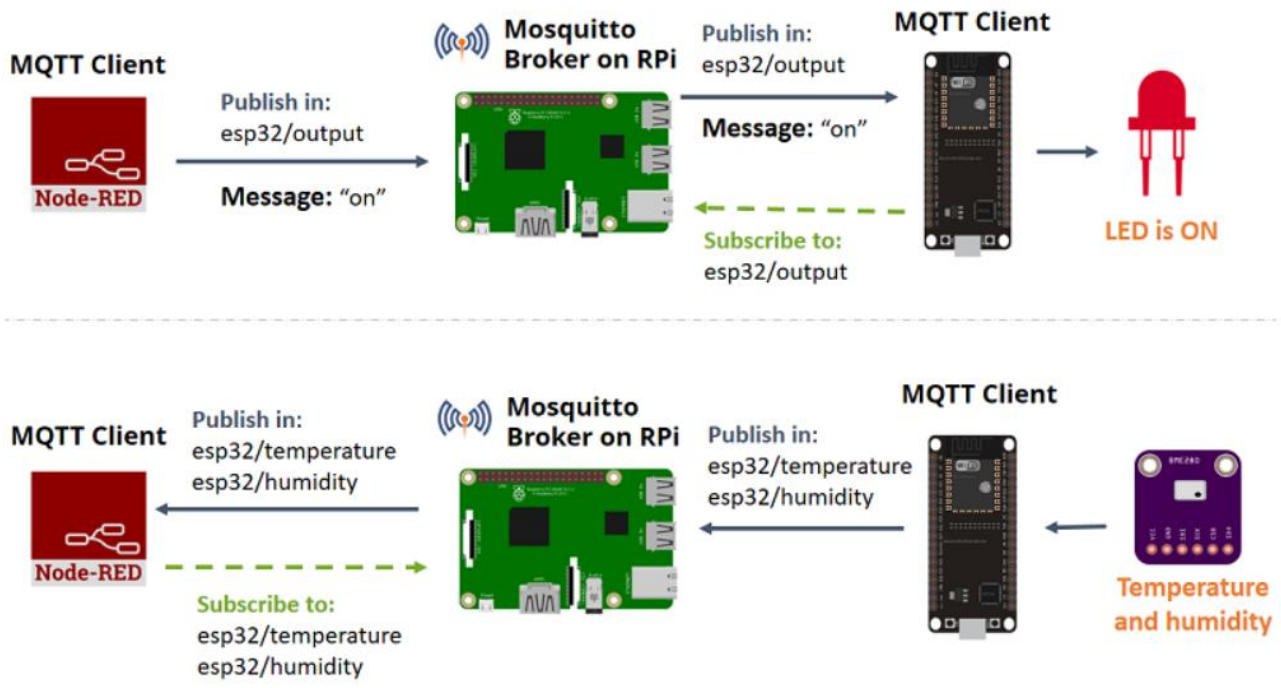

Figure 07: MQTT System Communication Architecture

https://randomnerdtutorials.com/esp32-mqtt-publish-subscribe-arduino-ide/

In this project, I used a Cloud-based MQTT Broker (HiveMQ) instead of a local physical broker like Raspberry Pi. This allows the system to work from any location with an internet connection. The ESP32 connects to the HiveMQ cloud server via Wi-Fi to publish sensor data. Simultaneously, the Python GUI subscribes to the same cloud broker to receive and display telemetry in real-time.

Programming language/ IDE

I used Arduino IDE to develop the firmware for this project. Instead of traditional Modbus libraries, I utilized the PubSubClient.h library to enable the ESP32 to communicate with the MQTT broker as a client. While TIA Portal was used in older systems for SCADA, this modern implementation features a custom-built

Python-based GUI Dashboard to monitor and control the system remotely.

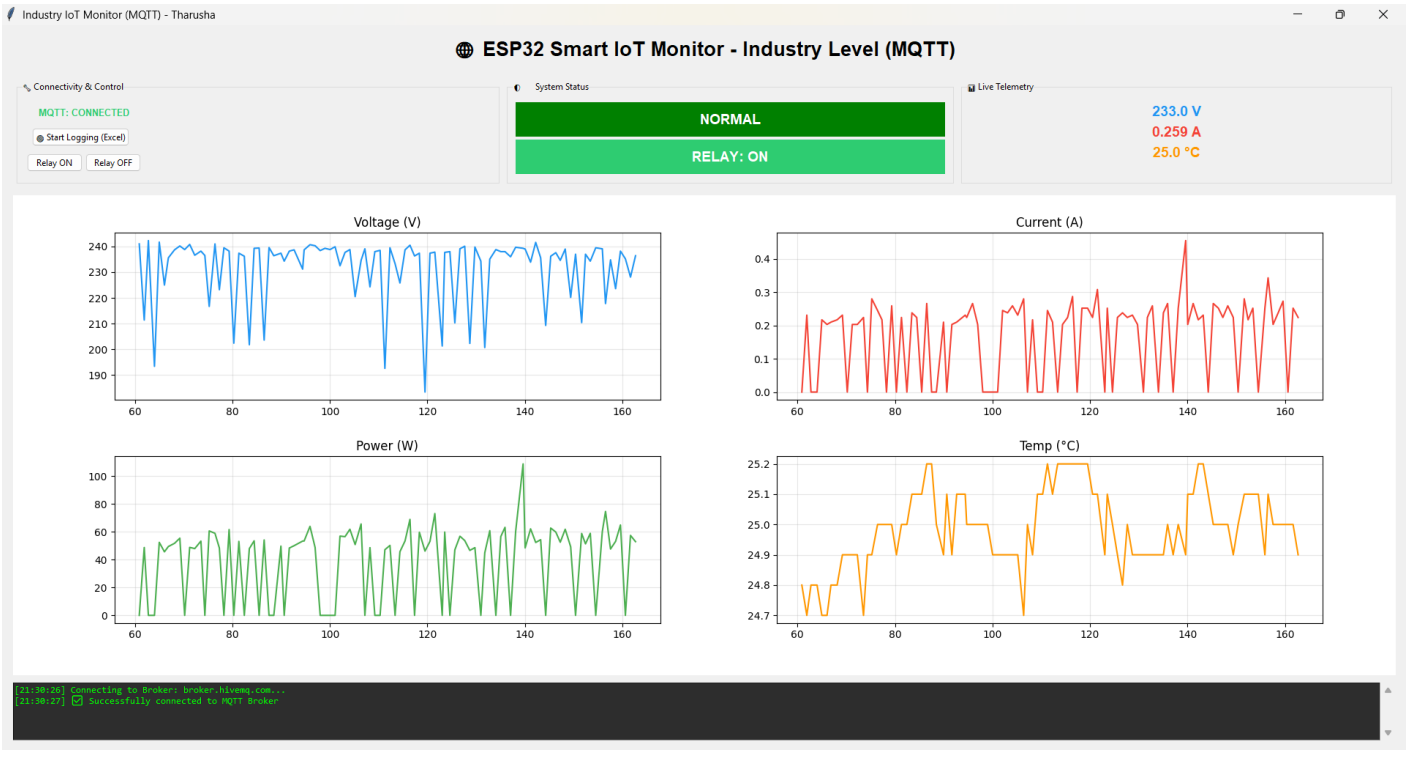

Figure 08: ESP32 Smart IoT Monitor Dashboard in active state (Relay ON).

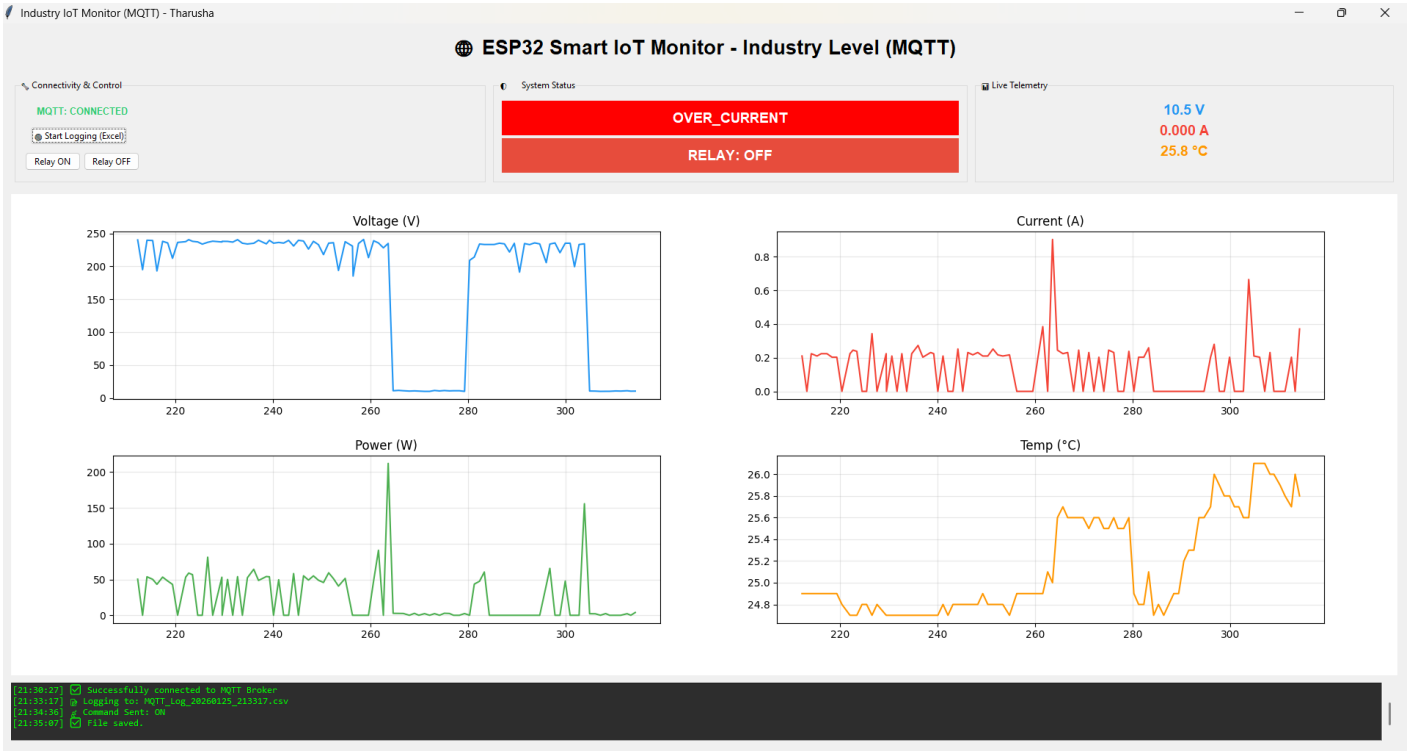

Figure 09: Dashboard during an Over-Current fault state with automatic Relay shutdown.

figure 11: GUI of boiler system when pump, mixer, chiller, and solenoid valve are on.

Reference

Knolleary (2020) *PubSubClient Library for Arduino/ESP32.* Available at: https://github.com/knolleary/pubsubclient (Accessed: 26 January 2026).

HiveMQ (n.d.) *MQTT Dashboard and Public Broker.* Available at: https://www.hivemq.com/public-mqtt-broker/ (Accessed: 26 January 2026).