PRESENTATION OF MASTER THESIS:

“INITIAL DESIGN AND IMPLEMENTATION OF A CONTROL SYSTEM FOR A HYDROGEN-BASED MICROGRID”

Name: Jan Moritz Dehler

Institutions: HAW Hamburg - Faculty of Life Sciences & University of Hamburg - Hamburger Sternwarte

1st Supervisor: Prof. Dr. Carsten Frank (HAW Hamburg)

2nd Supervisor: Prof. Dr. Robi Banerjee (Hamburger Sternwarte)

Submission Date of Thesis: 1st of May 2025

Presentation Date at the INET Office: 10th of July 2025

Universität Hamburg

DER FORSCHUNG | DER LEHRE | DER BILDUNG

HAW HAMBURG

OUTLINE

HAW HAMBURG

A BRIEF INTRODUCTION ABOUT MYSELF

- Name
  - Jan **Moritz** Dehler
- My career so far
  - HAW-Student since 2018 & completed the BSc. Umwelttechnik (Environmental Engineering) in 2022
  - Completed my Master Thesis on the Major Renewable Energy Systems in May 2025
  - Previous technical background in the wind industry and later specialization in the field of embedded systems
- Experiences
  - ESP32/8266, Arduino DUE, Raspberry Pi 4/5
  - Python, C++, VBA, MATLAB
- Hobbies and Interests
  - Sports (football, volleyball, badminton, running & hiking), traveling
  - Board game nights
  - Currently learning Korean

Passionate about topics revolving around the Internet of Things!

Universität Hamburg
DER FORSCHUNG | DER LEHRE | DER BILDUNG

HAW HAMBURG

MOTIVATION & OBJECTIVES FOR THE THESIS

- Use Case: Observatory in Mexico
  - TIGRE Observatory in La Luz relies on stable power for remote telescope operation
  - Grid instability affects telescope performance and data transmission
  - Goal: make observatory autonomous using PV + H_2 + batteries
  - Prototype testing in Hamburg, designed for future deployment in Mexico
  - Lithium Ion Battery in combination with hydrogen storage ensures long time storage

- Thesis Objective
  - Develop a low-cost, modular control system for a hydrogen-based microgrid
  - Coordinate PV, battery, electrolyzer & PEM fuel cell
  - Implement using open-source hardware & software (Raspberry Pi, Arduino, MQTT

Universität Hamburg
DER FORSCHUNG | DER LEHRE | DER BILDUNG

HAW HAMBURG

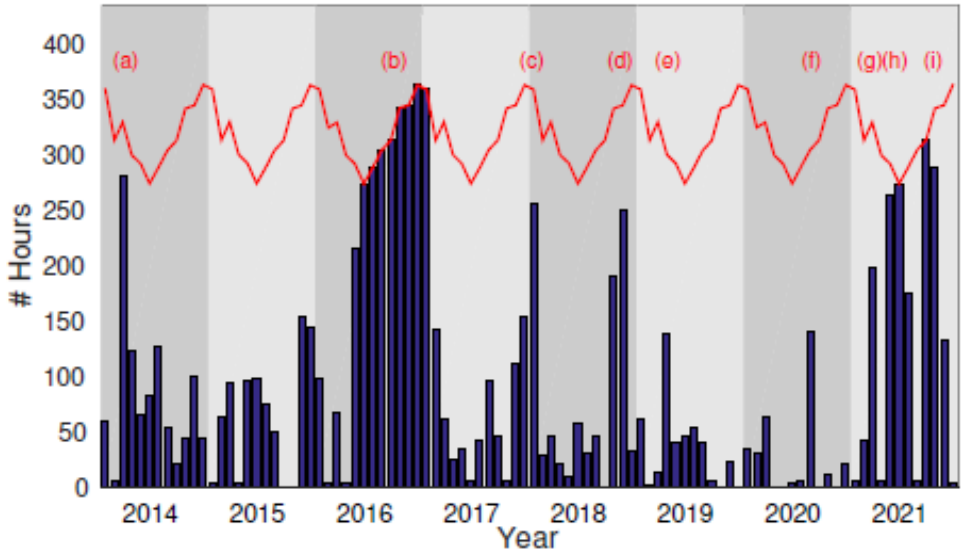


Figure 7. Number of hour per month lost due to technical issues (bars). The red line display the total number of night hours. Specific technical problems caused the events labeled (a) to (i) as detailed in the text.

Both images are taken from [1]

Universität Hamburg
DER FORSCHUNG | DER LEHRE | DER BILDUNG

HAW HAMBURG

SUMMARY OF THE THEORETICAL BACKGROUND

- What is a Microgrid?
  - Decentralized energy systems that can run independently or grid-connected
  - Improve energy resilience in remote, unstable, or off-grid environments
  - Typical components: PV, batteries (short-term), hydrogen systems (long-term)
- Why Intelligent Control?
  - Subsystems (e.g. inverter, electrolyzer) often have their own local logic
  - Central control unit monitors status and manages switching via relays
  - Fast decision-making based on current system conditions
- Communication & Protocols
  - CAN Bus, Modbus RTU/TCP → reliable communication with energy hardware
  - MQTT: lightweight publish–subscribe protocol (application layer over TCP/IP)
    → Common in IoT & embedded systems for distributed module communication
- Hardware Platforms
  - Raspberry Pi 4 → central logic, communication via various interfaces possible, data logging
  - Arduino Due → potential for real-time sensor acquisition

Universität Hamburg
DER FORSCHUNG | DER LEHRE | DER BILDUNG

HAW HAMBURG

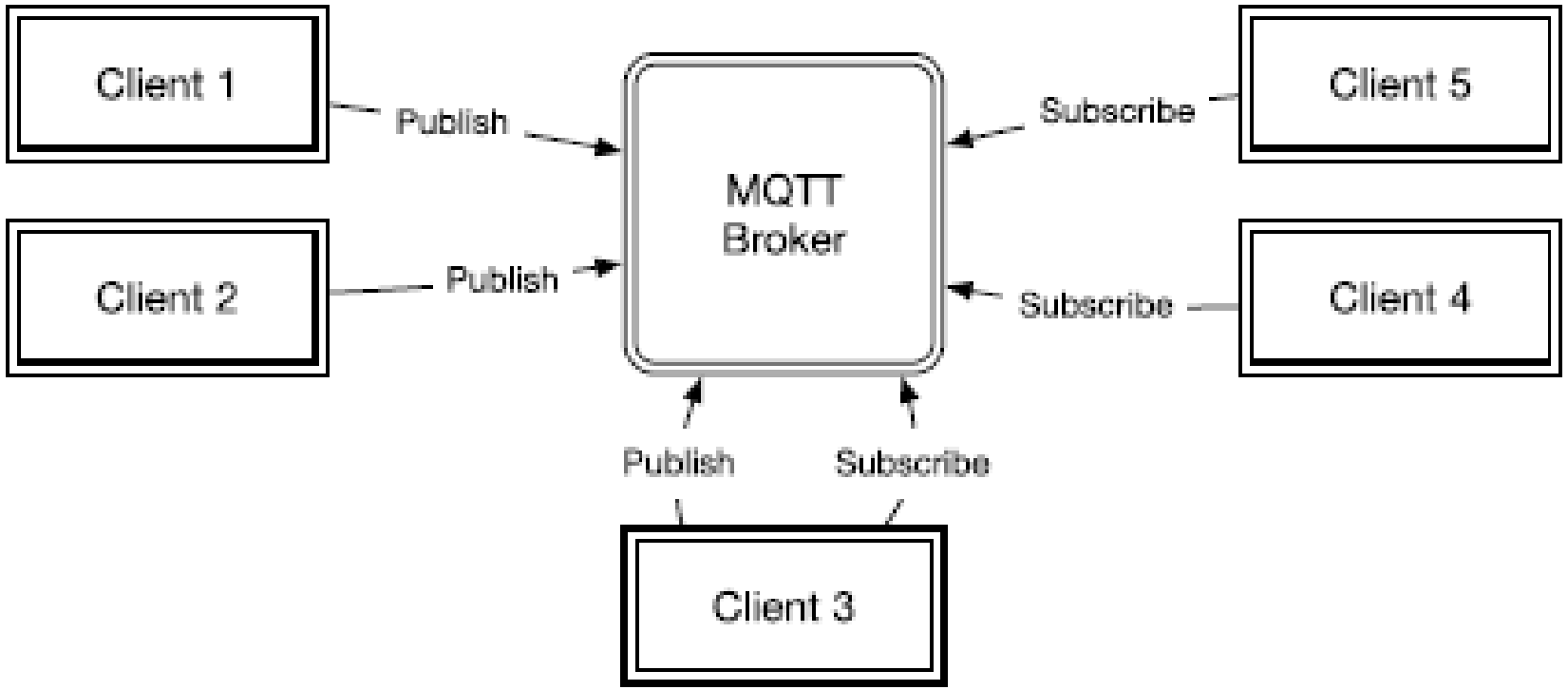


Image taken from [2]

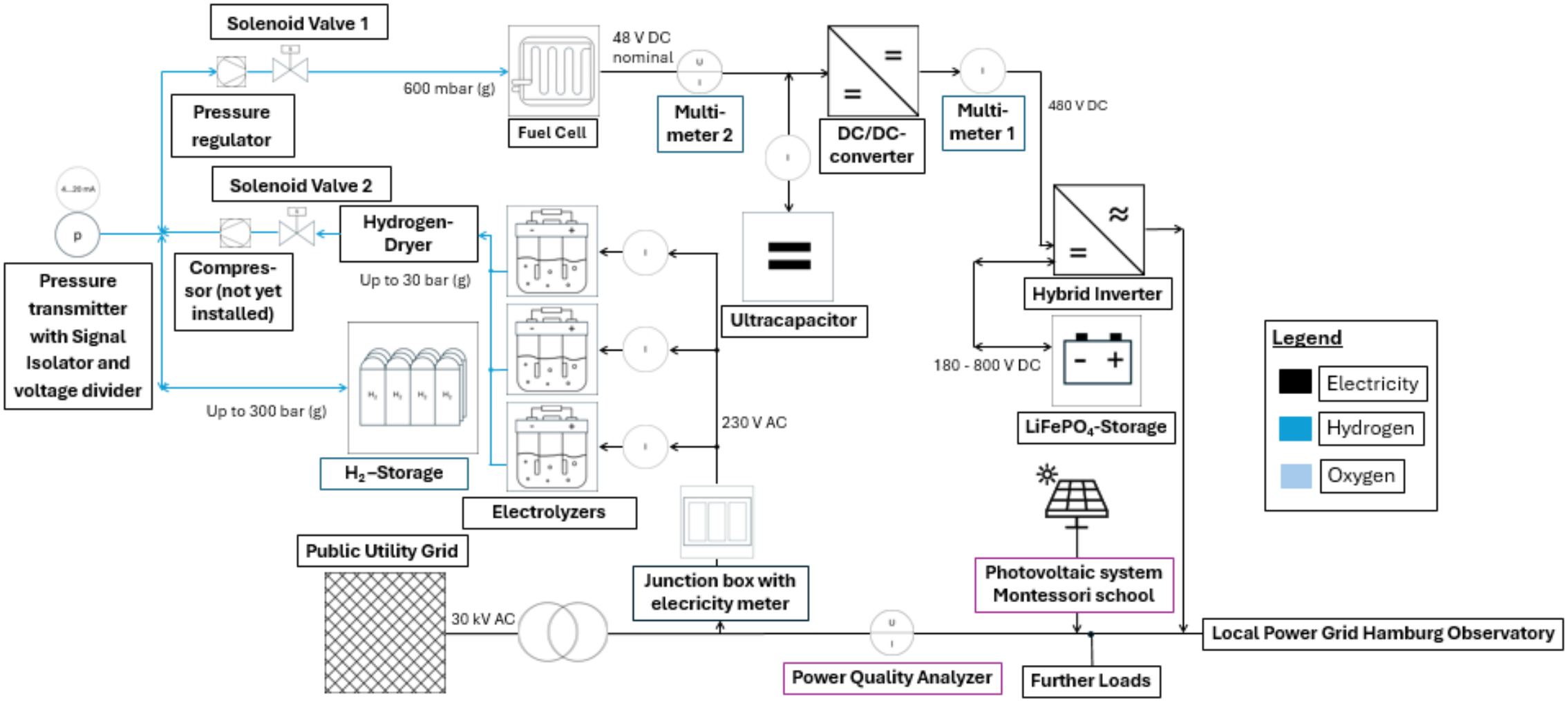
MICROGRID INFRASTRUCTURE: HARDWARE & COMMUNICATION

- Central Control Unit: Raspberry Pi 4
  - Acts as the main controller and decision-maker
  - Runs Python-based logic, energy management, data logging & MQTT broker
  - Coordinates all subsystems: energy flows, error detection, and message aquisiton
- Sensor & Device Communication Protocols
  - CAN Bus: For communication with fuel cell stack (e.g., voltage, current, status) usage of RS485 CAN HAT mounted on Raspberry Pi
  - Modbus: Industrial-standard protocol to interact with components like the inverter (RTU) usage of RS485 CAN HAT mounted on Raspberry Pi; Interaction with electrolyzers over Modbus TCP protocol
  - Ethernet (TCP/IP): For Data Acquisition from Multimeters, and external MQTT connection
  - MQTT Protocol: Lightweight publish–subscribe system connecting all software modules
- Time-Critical Sensor Acquisition via Arduino Due
  - Handles fast, low-level sensor tasks independently of the Pi
  - Reads data via:
    - I^2C: Digital sensors (e.g. temperature, humidity)
    - Analog voltage: Electrochemical gas sensors, loudness sensors
  - Sends preprocessed sensor data to Raspberry Pi via a custom GPIO interface with handshake logic
  - Ensures real-time responsiveness and system safety

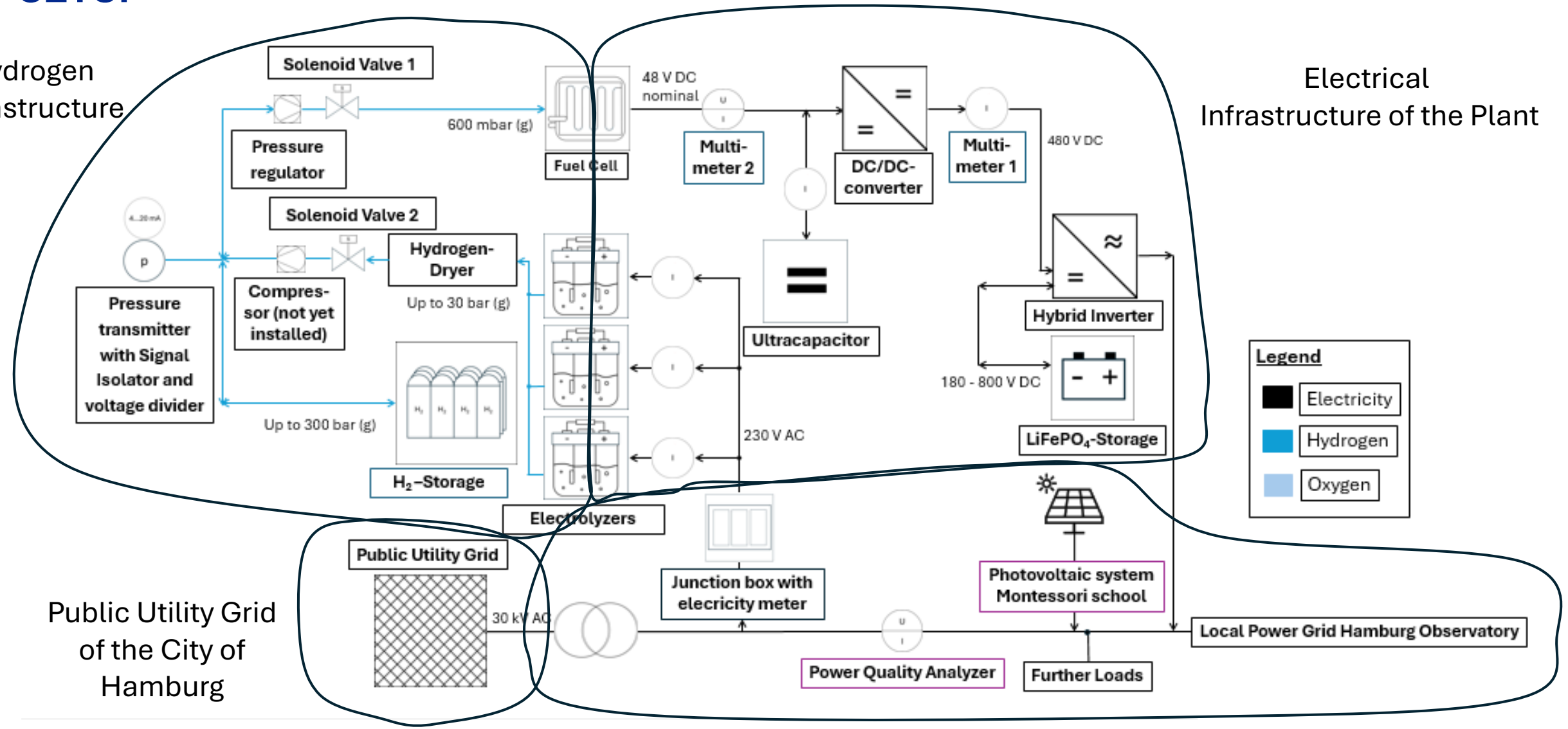
Universität Hamburg
DER FORSCHUNG | DER LEHRE | DER BILDUNG

HAW HAMBURG

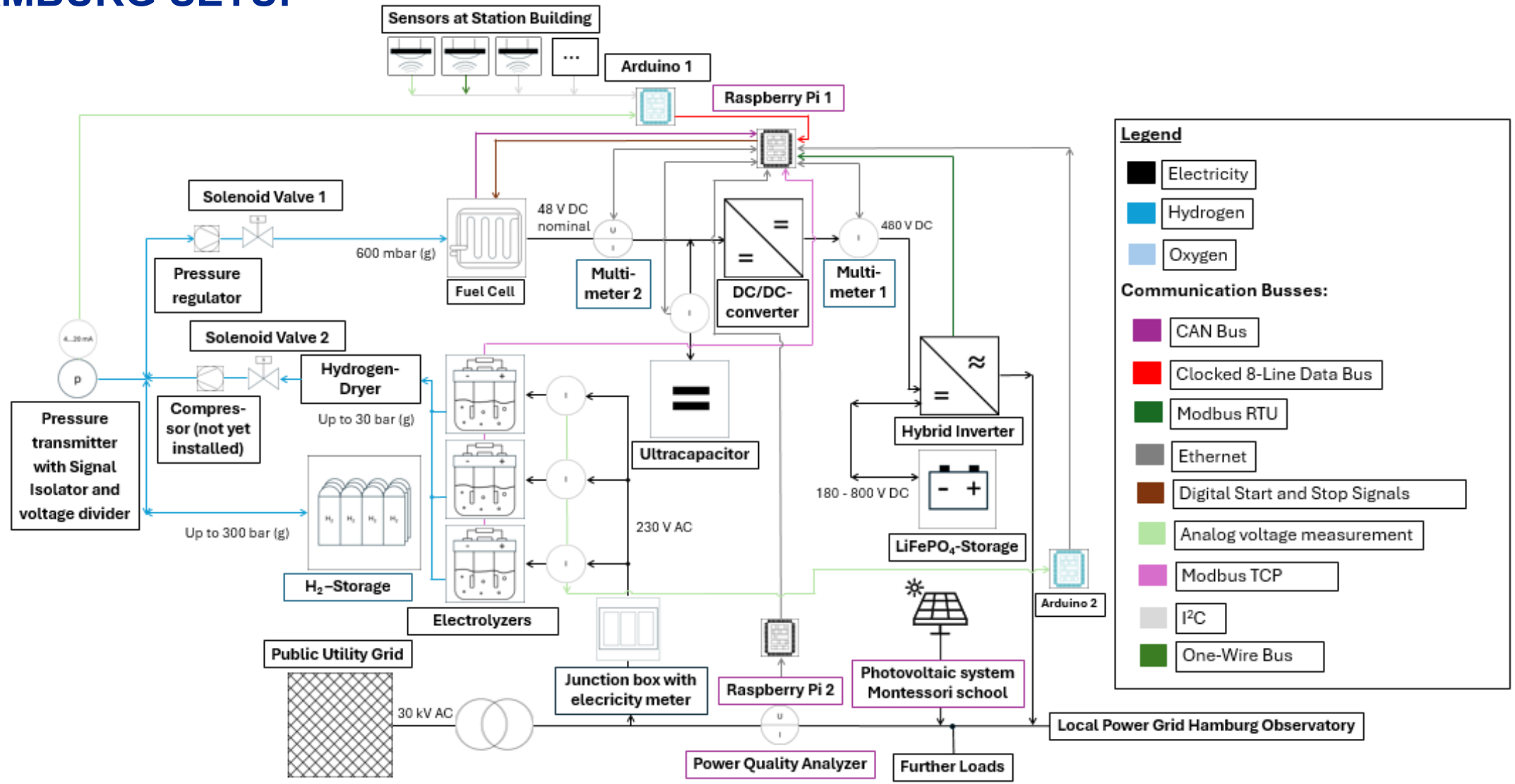
HARDWARE ARCHITECTURE OF THE MICROGRID CONCERNING THE HAMBURG SETUP



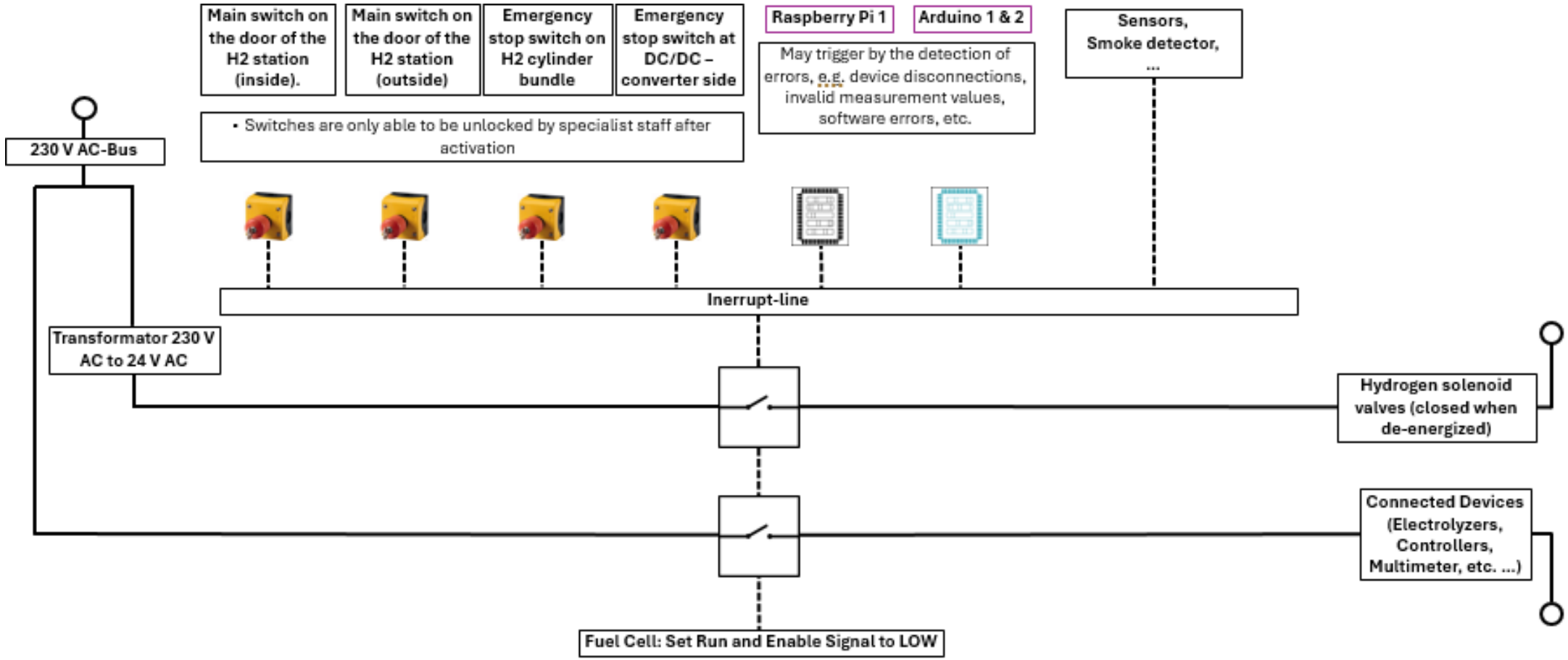
HARDWARE ARCHITECTURE OF THE MICROGRID CONCERNING THE HAMBURG SETUP



COMMUNICATION ARCHITECTURE OF THE MICROGRID CONCERNING THE HAMBURG SETUP



CONCEPT FOR THE EMERGENCY STOP CHAIN



PYTHON ARCHITECTURE ON THE RASPBERRY PI (I)

- 1. Python Script Structure on the Raspberry Pi
  - Class-based modules
    - Reusable functionalities (e.g., watchdog_class.py, datacheck_class.py) for timing control and data validation.
  - & Loop-based programs (*_loop.py)
    - Continuous while True loops handle:
      - Data acquisition (e.g., Arduino, Modbus)
      - MQTT messaging (publish/subscribe)
      - Logging and error detection
      - Plotting MQTT broker data with matplotlib
    - Script Roles:
      - Publishing Programs: e.g., arduino_loop.py, modbus_loop.py
      - Subscribers: e.g., for logging of mqtt data in csv files or data ploting, EMS logic
      - Also possible GUI (not implemented yet)
    - Startup Structure:
      - All modules are launched via main.py using subprocess. Popen().

      → Ensures process isolation and fault recovery.

Universität Hamburg
DER FORSCHUNG | DER LEHRE | DER BILDUNG

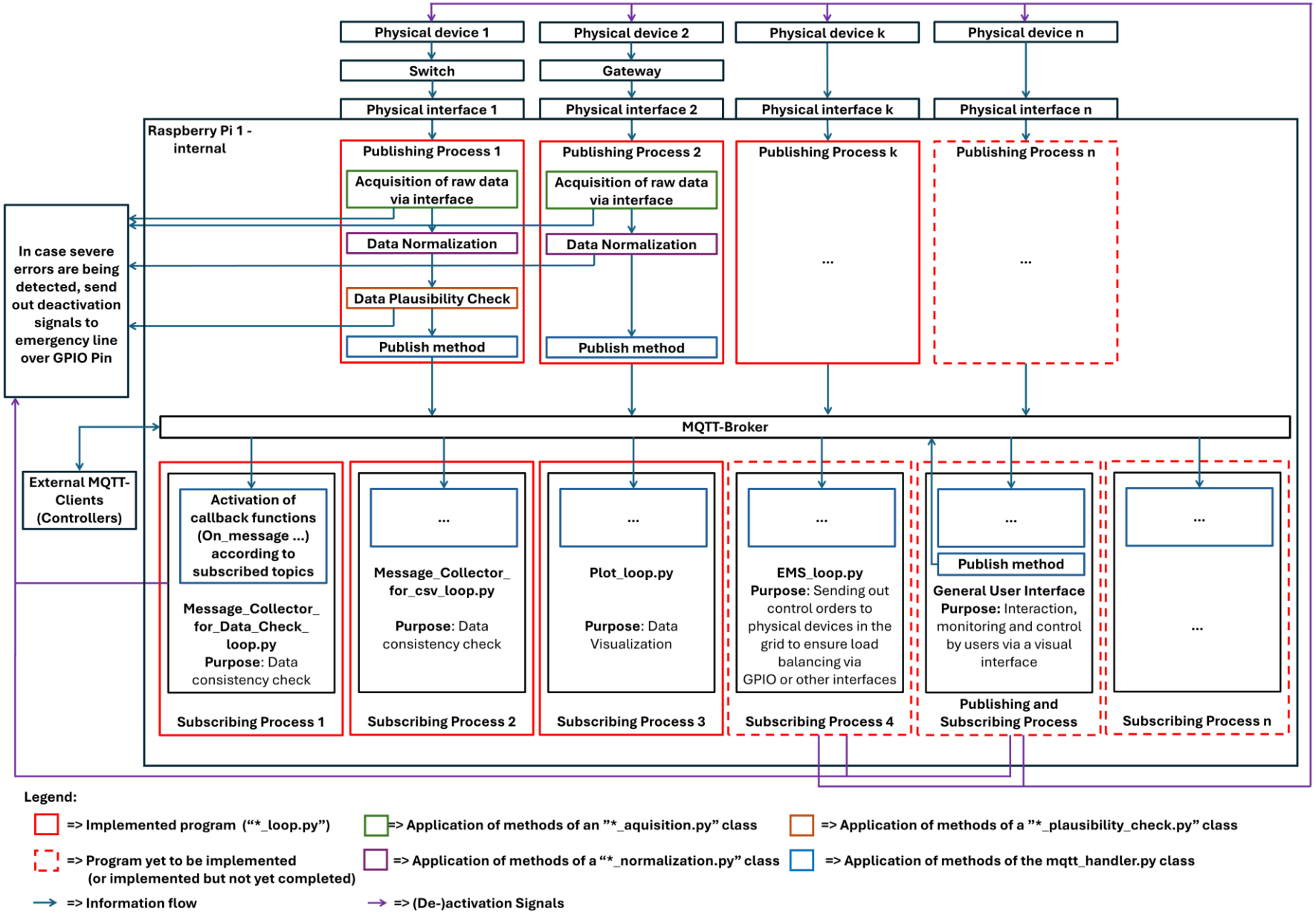
HAW HAMBURG

PYTHON ARCHITECTURE (II)

- 2. Inter-Process Communication: MQTT
  - Local Mosquitto broker on Raspberry Pi
  - Scripts communicate via paho-mqtt Python library
  - Decoupled messaging using publish/subscribe model
  - Advantages:
    - Loose coupling of modules
    - Independent reconnection and fault isolation
    - Scalable for additional components
- 3. Robustness Features
  - Watchdog timers in each loop: detect stalls or performance drops
  - Plausibility checks: sensor sanity checks (e.g. humidity ≤ 100%)
  - CSV logging: timestamped data for full traceability
  - Terminal & error logs: all outputs redirected to log files
  - Latency measurements of MQTT Messages

Universität Hamburg
DER FORSCHUNG | DER LEHRE | DER BILDUNG

HAW HAMBURG


Physical device 1
Physical device 2
Physical device k
Physical device n
Switch
Gateway
Physical interface 1
Physical interface 2
Physical interface k
Physical interface n
Raspberry Pi 1 - internal
Publishing Process 1
Publishing Process 2
Publishing Process k
Publishing Process n
Acquisition of raw data via interface
Acquisition of raw data via interface
Data Normalization
Data Normalization
Data Plausibility Check
Publish method
Publish method
...
...
In case severe errors are being detected, send out deactivation signals to emergency line over GPIO Pin
MQTT-Broker
External MQTT-Clients (Controllers)
Activation of callback functions (On_message ...) according to subscribed topics
Message_Collector_for_Data_Check_loop.py
Purpose: Data consistency check
Subscribing Process 1
...
Message_Collector_for_csv_loop.py
Purpose: Data consistency check
Subscribing Process 2
...
Plot_loop.py
Purpose: Data Visualization
Subscribing Process 3
...
EMS_loop.py
Purpose: Sending out control orders to physical devices in the grid to ensure load balancing via GPIO or other interfaces
Subscribing Process 4
...
Publish method
General User Interface
Purpose: Interaction, monitoring and control by users via a visual interface
Publishing and Subscribing Process
...
...
Subscribing Process n
Legend:
=> Implemented program ("*_loop.py")
=> Application of methods of an "*_aquisition.py" class
=> Application of methods of a "*_plausibility_check.py" class
=> Program yet to be implemented (or implemented but not yet completed)
=> Application of methods of a "*_normalization.py" class
=> Application of methods of the mqtt_handler.py class
=> Information flow
=> (De-)activation Signals

Name of Program	Categorization	Purpose
CAN_BUS_01_ loop.py	Validated Publishing Program	Communication with Fuel Cell
HMC8012_loop.py		Communication with Multimeter 1
HMC8015_loop.py		Communication with Multimeter 2
get_resource_usage_loop.py		Acquisition of System Resource Data
Arduino_Communication_loop.py		Communication with Arduino 1
Modbus_RTU_loop.py	Publishing Program (Dummy)	Communication with Hybrid Inverter
modbusTCP_elektrolyseur_loop.py		Communication with Electrolyzers
SolaxHybrid_loop.py		Communication with Hybrid Inverter
Message_ Collector_for_csv_loop.py	Validated Subscribing Program	Periodic logging of the MQTT server content in a csv file
Message_Collector_for_Data_Check_loop.py		Data integrity validation over data coming from MQTT-server
Message_Collector_for_plot_loop.py		Visualization of specific data series of MQTT-server in a live-plot
EMS_loop.py	Subscribing Program (Dummy)	Energy management system
main.py	main.py	Initialization of all other programs
blink_loop.py	blink_loop.py	Visualization of system activity via blinking of the status LED
get_svg_files_loop.py	get_svg_files_loop.py	Performance profiling through generation of .SVG files

```
# Unix-Zeitstempel in ein lesbares Format konvertieren
readable_time = time_conversion.unix_to_readable_with_ms(current_time)

payload = {
    "software_timestamp": str(readable_time),
    "error_observed": error_observed_to_publish,
    "error": error_to_publish,
    "connected": HMC8015_Receiver.connected,
    "raw_data": str(HMC8015_Receiver.data),
    "voltage": HMC8015_Normalizer.processed_values[1],
    "current": HMC8015_Normalizer.processed_values[3]
    }

mqtt_client.publish(topic = f"{script_name}_data", message = json.dumps(payload), private_debug = False)
```

```
###### Alle im System vorgesehenen Topics ######

self.topics_for_mqtt_names_to_subscribe = [
        "Message_Collector_for_csv_loop_data",
        "Message_Collector_for_Data_Check_loop_data",
        "Message_Collector_for_plot_loop_data",
        "HMC8015_loop_data",
        "HMC8012_loop_data",
        "Arduino_Communication_loop_Sensors_H2_Station_data",
        "Arduino_Communication_loop_Sensors_for_testing_1_data",
        "CAN_BUS_01_loop_data",
        "Modbus_RTU_loop_data",
        "modbusTCP_elektrolyseur_loop_data",
        "SolaxHybrid_loop_data",
        "EMS_loop_data",
        "Hioki_data",
        "get_resource_usage_loop_data",
        "main_resource_usage_data",
        "Message_Collector_for_csv_loop_resource_usage_data",
        "Message_Collector_for_Data_Check_loop_resource_usage_data",
        "Message_Collector_for_plot_loop_resource_usage_data",
        "HMC8015_loop_resource_usage_data",
        "HMC8012_loop_resource_usage_data",
        "Arduino_Communication_loop_resource_usage_data",
        "CAN_BUS_01_loop_resource_usage_data",
        "Modbus_RTU_loop_resource_usage_data",
        "modbusTCP_elektrolyseur_loop_resource_usage_data",
        "SolaxHybrid_loop_resource_usage_data",
        "EMS_loop_resource_usage_data",
        "get_resource_usage_loop_resource_usage_data",
        "blink_loop_resource_usage_data"
        ]
```

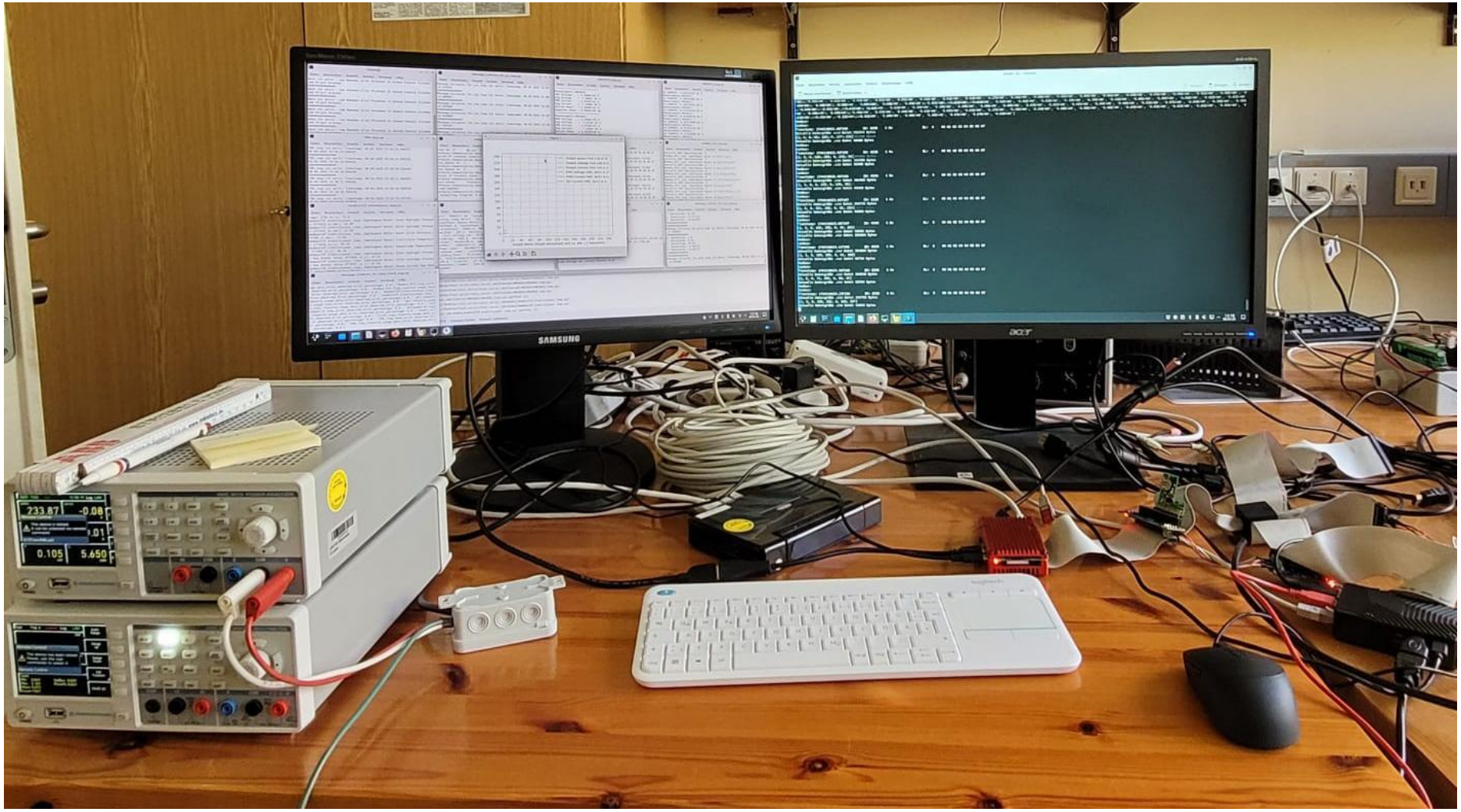
```python
def on_message_HMC8015_loop_data(self, client, userdata, message):

    try:
        payload = message.payload.decode("utf-8")  # Nachricht als String
        if not payload.strip():  # Leerzeichen entfernen und prüfen, ob leer
            print(f"{client}: Leerer Payload erhalten von Topic '{message.topic}', Verarbeitung wird abgebrochen.")
        else:
            # JSON-Daten parsen
            data = json.loads(payload)
            self.HMC8015_loop_software_timestamp = data['software_timestamp']
            self.HMC8015_loop_error_observed = data['error_observed']
            self.HMC8015_loop_error = data['error']
            self.HMC8015_loop_connected = data['connected']
            self.HMC8015_loop_raw_data = data['raw_data']
            self.HMC8015_loop_voltage = data['voltage']
            self.HMC8015_loop_current = data['current']

    except json.JSONDecodeError as e:
        print(f"{self.client_name}_MQTT_handler (on_message): Fehler beim Parsen des JSON: {e} - Payload war: {repr(payload)} für das Topic: {message.topic}")
    except Exception as e:
        print(f"{self.client_name}_MQTT_handler (on_message): Ein unerwarteter Fehler ist aufgetreten: {e} - Payload war: {repr(payload)} für das Topic: {message.topic}")


    if self.allow_calculation_of_delta_t and self.HMC8015_loop_software_timestamp!=self.HMC8015_loop_software_timestamp_from_before:
        self.HMC8015_loop_software_timestamp_from_before = self.HMC8015_loop_software_timestamp
        index = self.topics_for_mqtt_names_to_subscribe.index(message.topic)
        self.calculate_delta_t(message, index, False)
```

Program Name	*t_delay* in seconds*	*t_delay_for_ publishing* in seconds	*print_interval_ time* in seconds
CAN_BUS_01_loop.py	(-)**	1	1
HMC8012_loop.py	0.1	1	1
HMC8015_loop.py	0.1	1	1
Modbus_01_loop.py	(-)**	1	1
Arduino_Communication_loop.py	0.1	1	1
modbusTCP_elektrolyseur_loop.py	0.1	1	1
get_resource_usage_loop.py	(-)***		

Program Name	*t_delay* in seconds*	*t_delay_for_ publishing* in seconds	*print_interval_ time* in seconds
Message_Collector_for_csv_loop.py	0.1	1	1
Message_Collector_for_Data_Check_loop.py	0.1	1	1
Message_Collector_for_plot_loop.py	1.5****	1	1
EMS_loop.py	0.1	1	1

Universität Hamburg
DER FORSCHUNG | DER LEHRE | DER BILDUNG

HAW HAMBURG

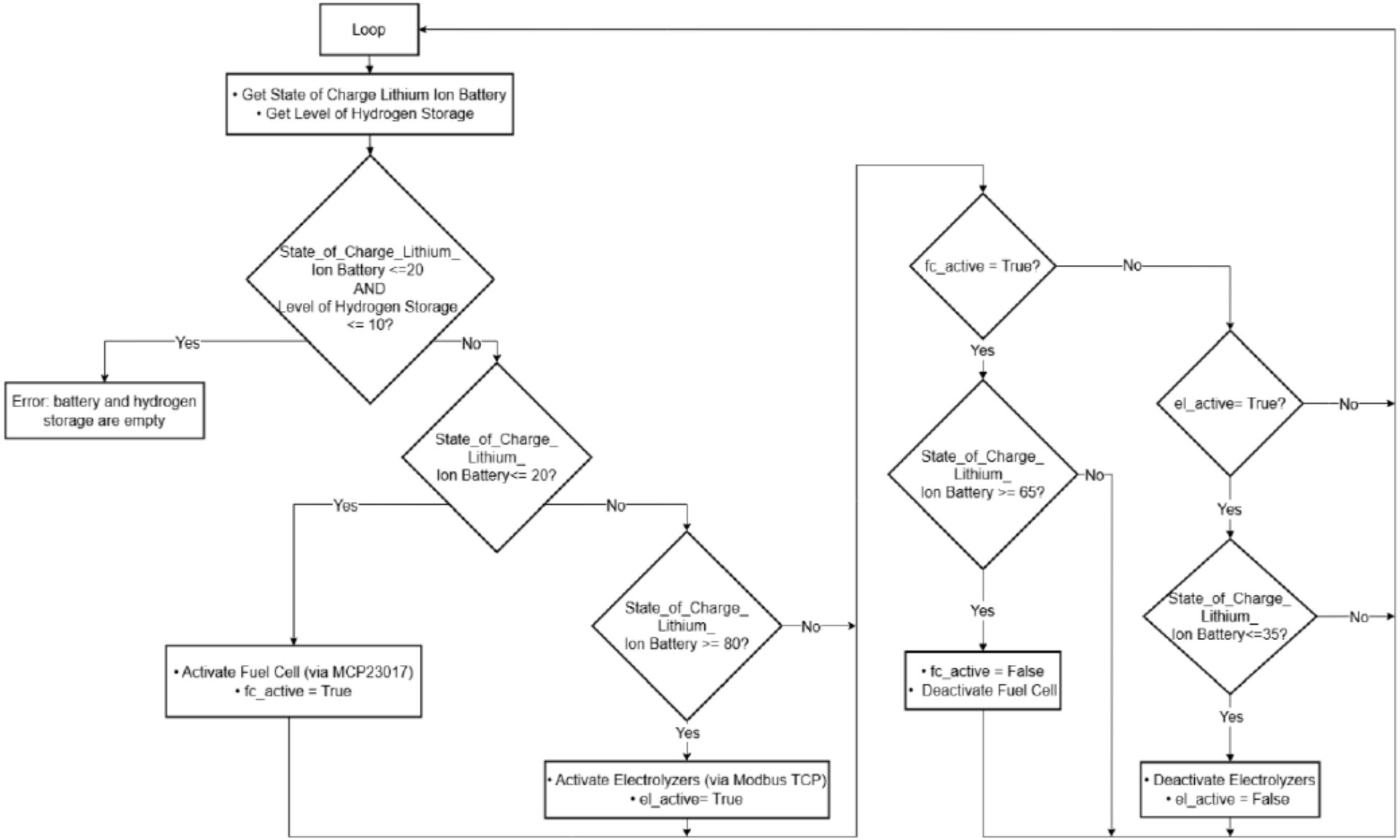

SAMSUNG
acer

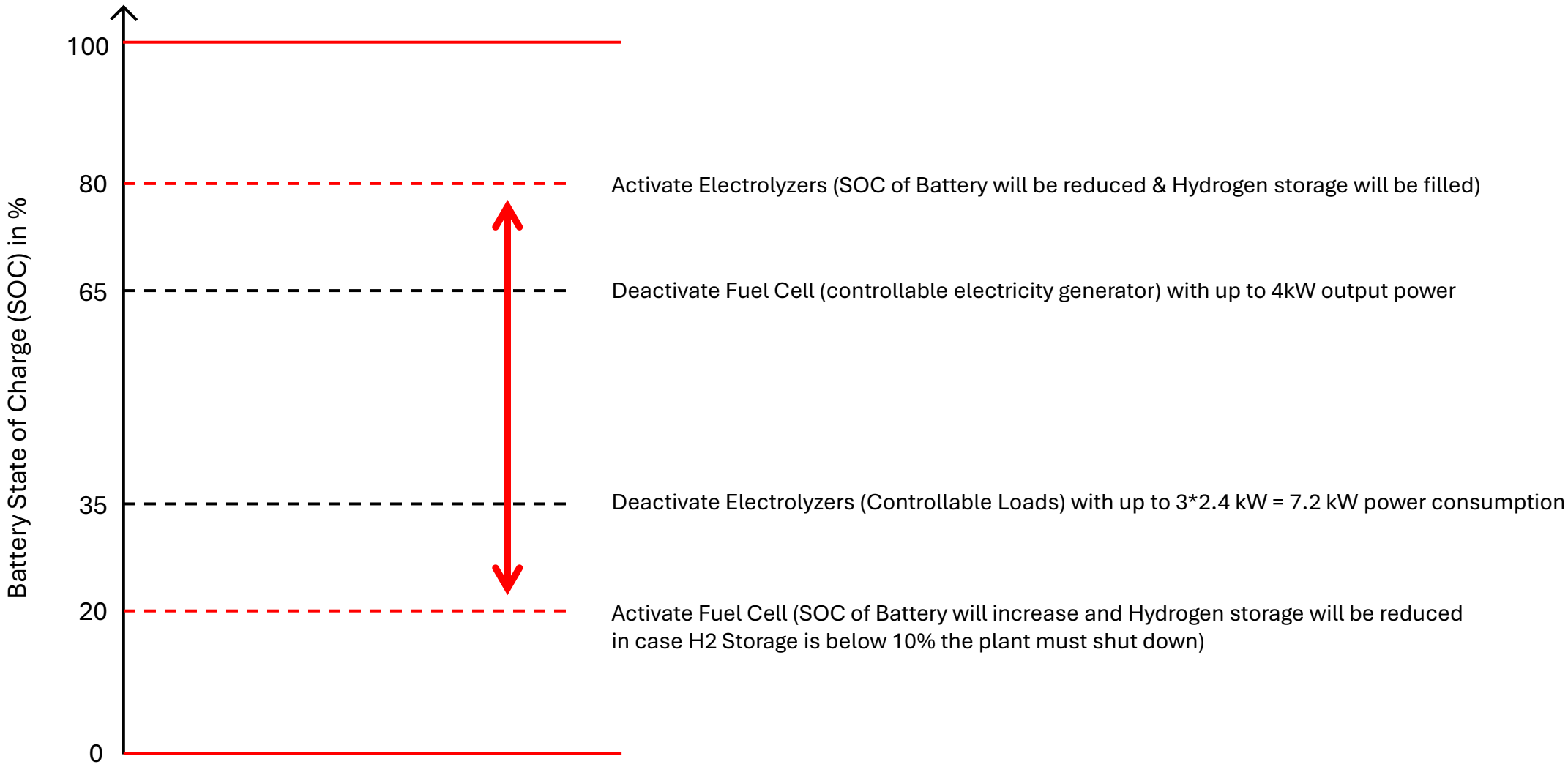
ENERGY MANAGEMENT SYSTEM (EMS): LOGIC & SIMULATION

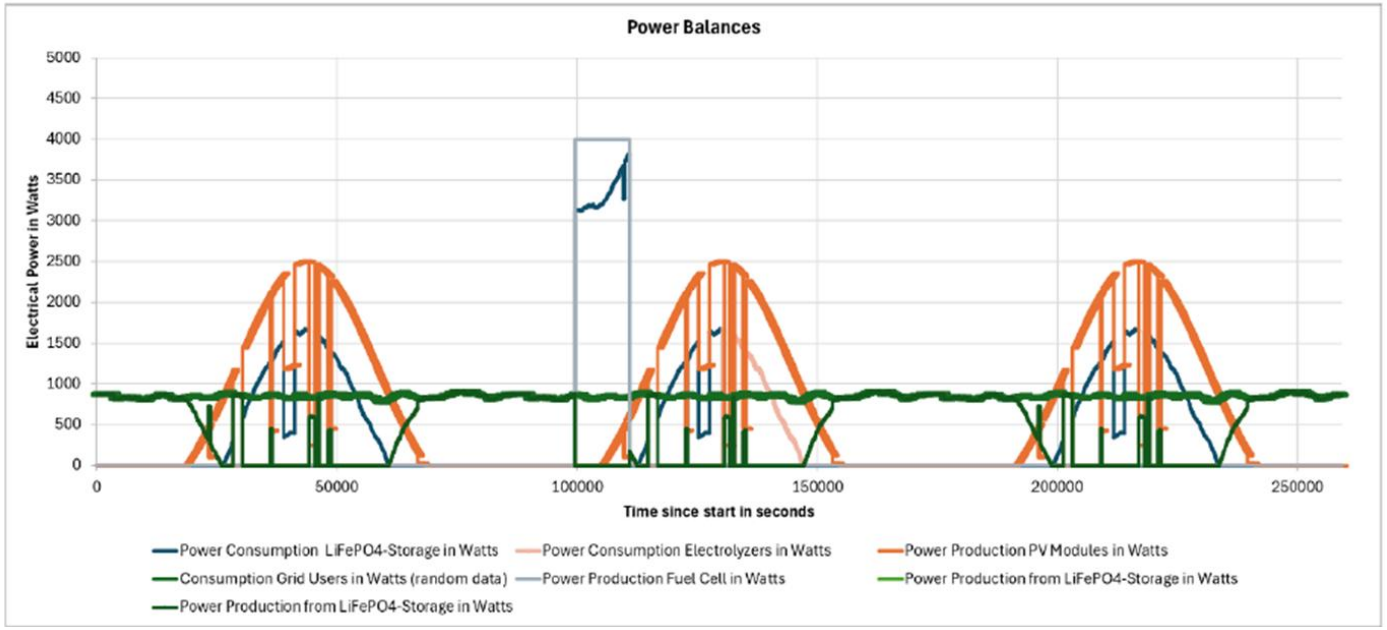
- Central control unit of the microgrid
  - Decides when to store, supply, or switch components on/off
- Control parameters
  - Battery State of Charge (SoC)
  - Hydrogen storage level
    - → Values acquired from respective *_loop scripts via MQTT
- Control logic
  - Battery prioritized for energy supply (efficient & fast)
  - Hydrogen system used only when battery is insufficient
  - Prevents wear from frequent switching
- Simulation model (Excel/VBA)
  - Simulates PV input, load, battery, and hydrogen behavior
  - Helps tune thresholds before hardware deployment
- Control signal pathways
  - Fuel Cell: Activation via relay board
  - Electrolyzers: Controlled via Modbus TCP
- Design goals
  - Simple, robust, and modular structure
  - Upgrade-ready for future predictive or adaptive EMS logic

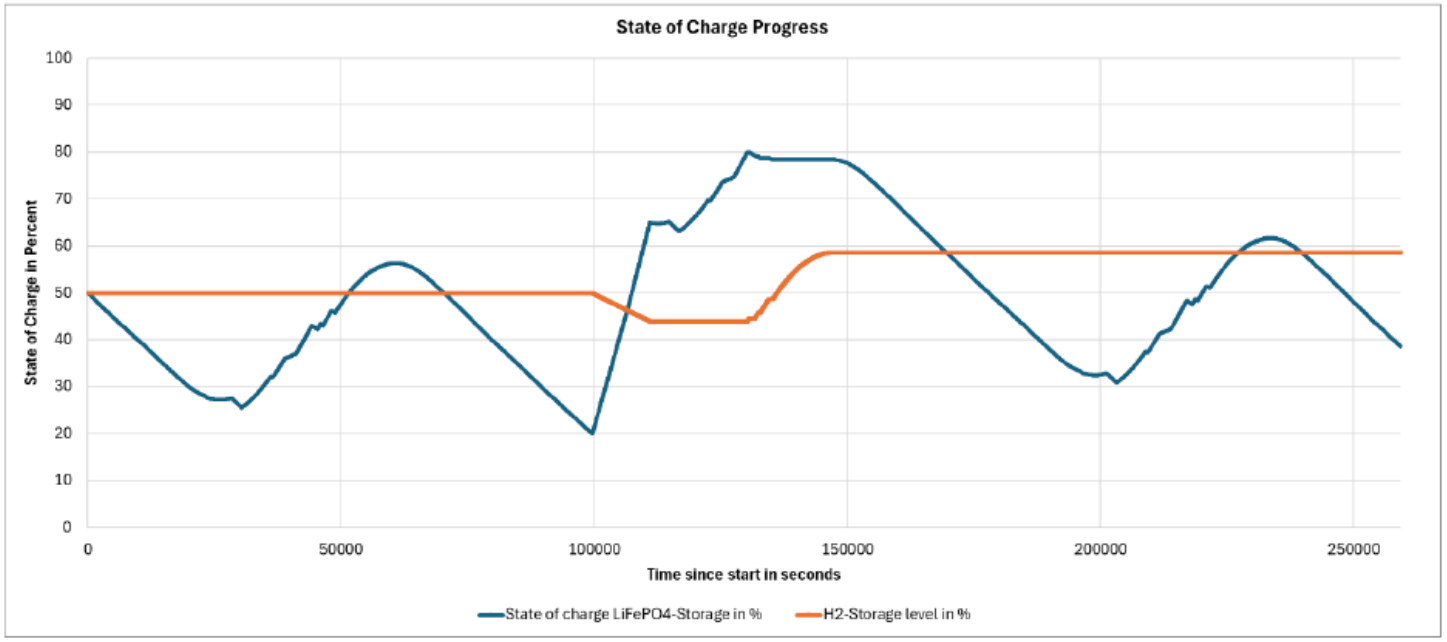
Universität Hamburg
DER FORSCHUNG | DER LEHRE | DER BILDUNG

HAW HAMBURG


Loop
• Get State of Charge Lithium Ion Battery
• Get Level of Hydrogen Storage
State_of_Charge_Lithium_ Ion Battery <=20 AND Level of Hydrogen Storage <= 10?
Yes
No
Error: battery and hydrogen storage are empty
State_of_Charge_ Lithium_ Ion Battery<= 20?
Yes
No
• Activate Fuel Cell (via MCP23017)
• fc_active = True
State_of_Charge_ Lithium_ Ion Battery >= 80?
No
Yes
• Activate Electrolyzers (via Modbus TCP)
• el_active= True
fc_active = True?
No
Yes
State_of_Charge_ Lithium_ Ion Battery >= 65?
No
Yes
• fc_active = False
• Deactivate Fuel Cell
el_active= True?
No
Yes
State_of_Charge_ Lithium_ Ion Battery<=35?
No
Yes
• Deactivate Electrolyzers
• el_active = False


Battery State of Charge (SOC) in %
100
80
65
35
20
0
Activate Electrolyzers (SOC of Battery will be reduced & Hydrogen storage will be filled)
Deactivate Fuel Cell (controllable electricity generator) with up to 4kW output power
Deactivate Electrolyzers (Controllable Loads) with up to 3*2.4 kW = 7.2 kW power consumption
Activate Fuel Cell (SOC of Battery will increase and Hydrogen storage will be reduced
in case H2 Storage is below 10% the plant must shut down)


Power Balances
Electrical Power in Watts
5000
4500
4000
3500
3000
2500
2000
1500
1000
500
0
0
50000
100000
150000
200000
250000
Time since start in seconds
Power Consumption LiFePO4-Storage in Watts
Power Consumption Electrolyzers in Watts
Power Production PV Modules in Watts
Consumption Grid Users in Watts (random data)
Power Production Fuel Cell in Watts
Power Production from LiFePO4-Storage in Watts
Power Production from LiFePO4-Storage in Watts

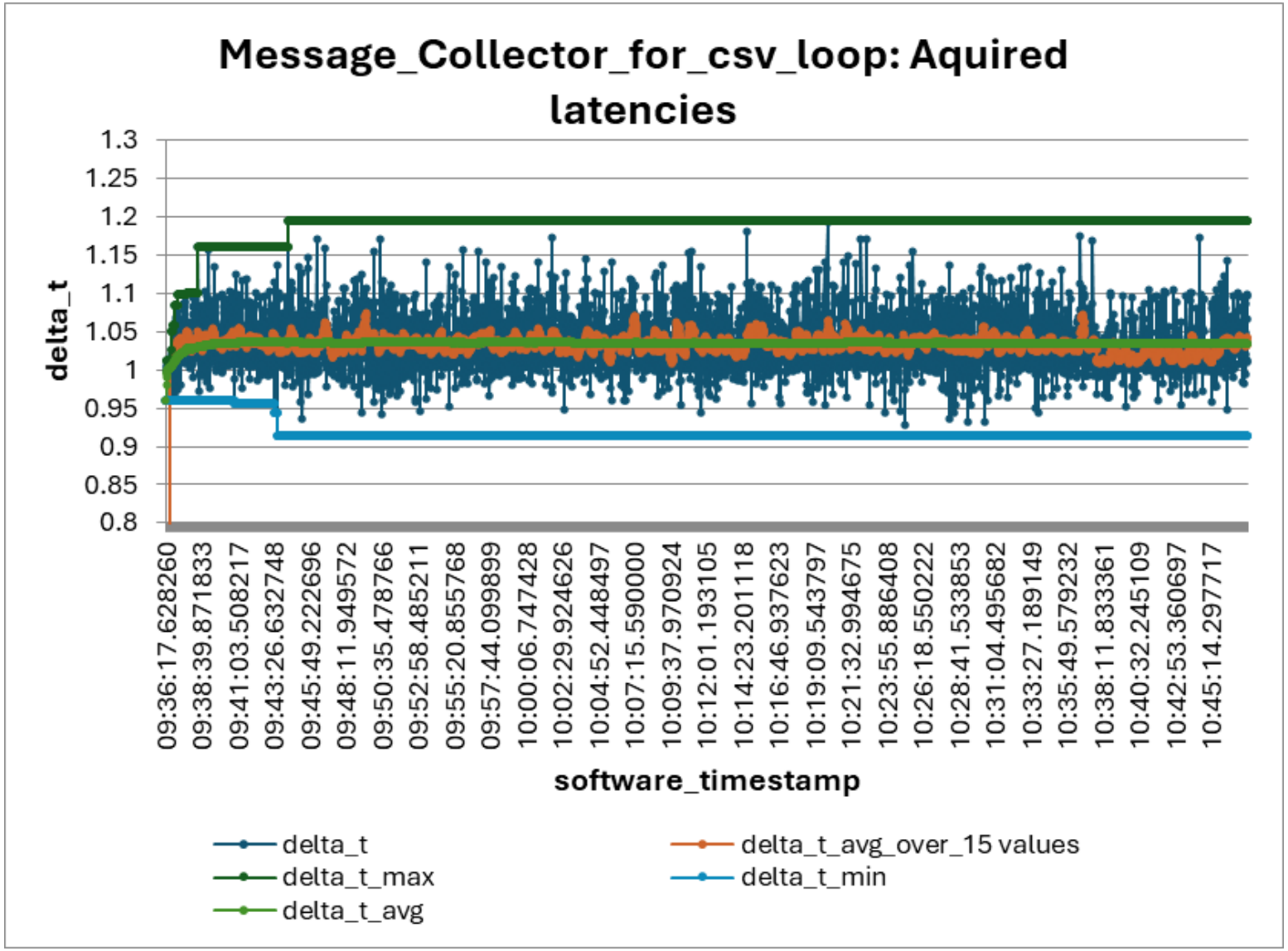

State of Charge Progress
State of Charge in Percent
100
90
80
70
60
50
40
30
20
10
0
0
50000
100000
150000
200000
250000
Time since start in seconds
State of charge LiFePO4-Storage in %
H2-Storage level in %

TEST RUNS & PERFORMANCE RESULTS

- Testing Period & Setup
  - Test phase of the software conducted over several months under lab conditions
  - Duration: October 2024 to April 2025
- Key Performance Metrics Analyzed
  - CPU, RAM & Swap Usage (Raspberry Pi) via psutils library in get_resource_usage_loop.py :
    - Load distribution across processes monitored over time
    - Ensured no memory leaks or bottlenecks
    - System remained stable even under extended runtime
- MQTT Message Latency
  - Measured round-trip delay between publisher and subscriber
  - Average latency consistently below 100 ms
  - Low message deviation rate (<0.2%) due to minor loss and occasional duplicates
- System Stability
  - Raspberry Pi ran continuously for >2 h without crash
  - All subprocesses launched via main.py remained active
  - Watchdog timers triggered correctly in simulated faults

Universität Hamburg
DER FORSCHUNG | DER LEHRE | DER BILDUNG

HAW HAMBURG


Message_Collector_for_csv_loop: Aquired latencies
delta_t
1.3
1.25
1.2
1.15
1.1
1.05
1
0.95
0.9
0.85
0.8
09:36:17.628260
09:38:39.871833
09:41:03.508217
09:43:26.632748
09:45:49.222696
09:48:11.949572
09:50:35.478766
09:52:58.485211
09:55:20.855768
09:57:44.099899
10:00:06.747428
10:02:29.924626
10:04:52.448497
10:07:15.590000
10:09:37.970924
10:12:01.193105
10:14:23.201118
10:16:46.937623
10:19:09.543797
10:21:32.994675
10:23:55.886408
10:26:18.550222
10:28:41.533853
10:31:04.495682
10:33:27.189149
10:35:49.579232
10:38:11.833361
10:40:32.245109
10:42:53.360697
10:45:14.297717
software_timestamp
delta_t
delta_t_avg_over_15 values
delta_t_max
delta_t_min
delta_t_avg

KEY ACHIEVEMENTS OF THE THESIS

- Functional Control & Monitoring System
  - Complete end-to-end prototype successfully implemented
  - Implementation of control logic, sensor data acquisition, visualization, logging & error management
  - Tested under lab conditions with stable performance
- Integration of Industrial Communication Protocols
  - Successfully combined:
    - CAN Bus for fuel cell stack monitoring
    - Modbus RTU/TCP for inverter and electrolyzer communication
    - Ethernet (TCP/IP) for dashboard and remote access
  - Demonstrated interoperability across diverse hardware components
- Custom High-Speed Arduino-Pi Interface
  - Implemented GPIO-based handshake logic for reliable sensor data transfer
- Modular & Scalable Software Design
  - 12 independent Python modules with an MQTT-based architecture enables easy extension or component replacement
  - Codebase is maintainable, open-source ready, and deployable in field conditions

Universität Hamburg
DER FORSCHUNG | DER LEHRE | DER BILDUNG

HAW HAMBURG

LIMITATIONS & FUTURE DEVELOPMENT

- Current Limitations of the Prototype
  - System not yet deployed in real-world conditions (lab-tested only)
  - No graphical user interface (GUI) → only file-based or terminal based visualization
  - EMS logic is rule-based and not predictive (static thresholds only)
- Planned & Recommended Future Work
  - Graphical User Interface (GUI)
    - Web dashboard for real-time monitoring and control
    - Improve accessibility and user interaction for non-developers
  - Advanced EMS Features
    - Dynamic or predictive energy management (e.g. using weather forecasts)
    - Load forecasting or priority-based load shedding
    - Integration of fault detection algorithms
  - Extended Sensor Network
    - Add sensors for pressure, hydrogen tank fill level, power quality (THD)
    - Enable better situational awareness and EMS decisions
  - Field Deployment at TIGRE Observatory (La Luz, Mexico)
    - Long-term outdoor testing in harsh climate
    - Evaluate system durability, error behavior, and maintenance effort
    - Gather real-world operational data for future improvements

Universität Hamburg
DER FORSCHUNG | DER LEHRE | DER BILDUNG
HAW HAMBURG

POSSIBLE FUTURE AREAS OF RESPONSIBILITY WITHIN MY ROLE AT INET

- Contributing to RIOT OS Core & Drivers
  - Extension and optimization of drivers for embedded platforms (e.g., ESP32, STM32)
  - Participation in RIOT kernel development in accordance with international community standards
- Energy-efficient protocol stacks
  - Integration or optimization of CoAP and MQTT-SN on RIOT OS
  - Benchmarking with alternative forms of communication such as NDN or 6LoWPAN
- Testbeds & field evaluation
  - Setup and support of urban/smart city testbeds (such as MONICA): Design of demonstrators for field validation of RIOT OS solutions

Universität Hamburg
DER FORSCHUNG | DER LEHRE | DER BILDUNG
HAW HAMBURG

REFERENCES TO FIGURES

[1] Schmitt, J. H. M. M. et al., 2014. TIGRE: A new robotic spectroscopy telescope at Guanajuato, Mexico. Astron.Nachr., 1 Oktober, 335(8), pp. 787-796.

[2] Plenk, V., 2024. Angewandte Netzwerktechnik kompakt - Dateiformate, Übertragungsprotokolle und ihre Nutzung in Java-Applikationen. 3 ed. s.l.:Springer Fachmedien. p. 117

Universität Hamburg
DER FORSCHUNG | DER LEHRE | DER BILDUNG

HAW HAMBURG