Høgskulen på Vestlandet

Master Thesis Report







By Simon FOUCAUD
From 11/08/2025 to 19/12/2025
Tutor: Professor Richard J.GRANT

Acknowledgements

*I would like to express my gratitude to **Richard J. Grant** and **Harald Moen** for their supervision, guidance, and constructive input throughout the course of this master's thesis.*

I would also like to thank all the people I had the opportunity to work with and meet in the basement laboratories, whose daily interactions, technical discussions, and friendly atmosphere greatly contributed to this experience.

Finally, I am grateful to HVL – Western Norway University of Applied Sciences for providing me with the opportunity to undertake my master's thesis at their university and for the academic and technical support made available during this work.

Table of contents

I. Introduction 4
A. Global energy context 4
B. Solar energy in Norway: context and potential 5
II. SolarLab presentation 6
III. Contribution to the project 7
A. Heater Automation 7
1. Programming Language Choice 8
2. Communication with Sunny Island: Modbus TCP 8
3. Communication with Shelly Devices: HTTP REST API 8
4. Description of the automatic SOC based program 9
5. Data acquisition from sunny Island and shelly control via HTTP 9
6. Manual mode 10
B. Storage capacity for the website 12
1. Context and system description 12
2. Problem identification 12
3. Initial Hypotheses and verification 12
4. Implemented solutions 13
C. Smart battery management for automated systems 15
1. Programming 16
2. Energy Level Concept 17
3. Electrical relay box 18
4. 3D Modelling 20
Conclusion 22
Next steps recommendations 23

Table of figures:

Figure 1: Global energy potential 5
Figure 2: Change in total final consumption of electricity in the world, 2012-2024 5
Figure 3: SolarLab system 7
Figure 4: 24h irradiance 13
Figure 5: 3 days irradiance 13
Figure 6: 168h data recovery 13
Figure 7: Hysteresis logic 16
Figure 8: Section diameter of a cable given the intensity 19
Figure 9: Relay box wiring 20
Figure 7: Hysteresis logicFigure 9: Relay box wiring 20
Figure 7: Hysteresis logicFigure 9: Relay box wiring 20
Figure 7: Hysteresis logicFigure 9: Relay box wiring 20
Figure 10: Shelly 3D printed support 20
Figure 11: 3D printed socket cover 21
Figure 12: Installed sockets cover 21

I. Introduction

As part of my final year of the master's program at ICAM, I was offered the opportunity to conduct my research thesis abroad. Among all the possibilities offered by my school I chose the Western Norway University of Applied Sciences (HVL) in Bergen. This experience was a unique opportunity to align my professional ambitions in the field of renewable energy with the discovery of a new culture and my passion for mountains. I was integrated into the department of Mechanical Engineering and Maritime Studies under the supervision of Professor Richard J. GRANT with whom I worked on the SolarLab, a passive house running on solar energy only.

In this report you will find the global and local context of solar energy and how the SolarLab monitors and manages solar production, an overview of my project detailing the tasks that I have led, including the automation of the heater and electric vehicle chargers, the design and implementation of the electrical relay box, and some recommendations for the next student.

A. Global energy context

Each year we consume more electricity due to new technologies, artificial intelligence and mass production. In 2024 the global electricity demand increased by 4.3% nearly double the average growth of total energy demand between 2010 to 2023 (2.7%) (IAE, Paris, 2025) (Figure 2). As fossil fuels are finite, it is essential to turn to renewable energy. Why that much interest in Solar energy?

Solar energy holds enormous potential. If we could harness just the direct solar radiation striking the Earth, we could meet the current annual global energy demand of approximately 1081.1 TW (Energie Use Per Person, 2025) in just one hour. Figure 1 compares the available energy resources, showing that even the sum of all fossil fuels remains far below the sun's potential. Since 2010 the solar power generation has grown exponentially worldwide. In 2010 we used to genereate 32.21TWh of ectricity from solar power whereas in 2024 we generated 2,128.37 TWh of it, with China coming on top with 839.04TWh produced in 2024. This puts solar energy in the 6th place of the electricity production sources just behind wind and nuclear in 2024. (Energy Institute, 2026)

This abundance, combined with its low environmental impact and decentralized deployment possibilities, explains the growing interest in solar energy worldwide.

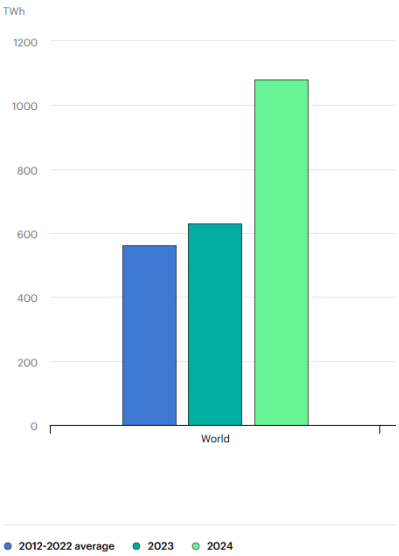


Figure 1: Change in total final consumption of electricity in the world, 2012-2024

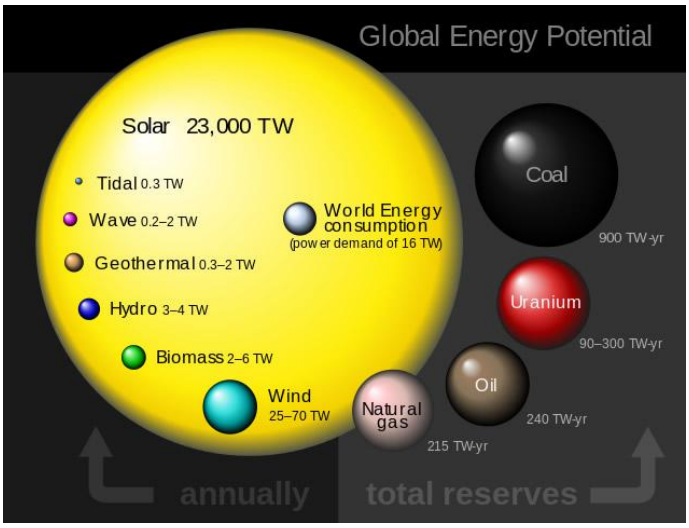


Figure 2: Global energy potential

B. Solar energy in Norway: context and potential

Norway is internationally recognized for its strong commitment to sustainable energy and technological innovation. The country stands among global leaders in renewable energy production as it has the highest share of electricity produced from renewable sources in Europe, 88% of its production comes from Hydropower plants and 11% from wind farms. (Norwegian Ministry of Energy, 2025). It is well known that Norway has Europe's most energy use per person. With not far from 100,000kWh of energy consumption per person, Norway stands far from Europe's average consumption of 39,255kWh (Rosado, 2025). The country needs a lot of resources to be able to answer consumer demands pushing them to diversifies their energy production sources.

Although Norway is mainly powered by hydropower, solar energy is increasingly recognized as a valuable complement. Installing solar farms is a big challenge in Norway as the country is mainly shaped by mountains; to overcome this issue solar panels are directly mounted on rooftops and facades. Study shows that solar energy could power half of the country's annual electricity demand with up to 87 gigawatts of possible installation (Gholmi, 2024). Despite its northern latitude, the cool climate improves photovoltaic efficiency and summer daylight provides significant production potential. This enormous amount of solar energy can not be fully exploited due to grid limitations showing the need for energy storage and smarter energy management systems.

Bergen, one of Europe's wettest cities, offers particularly challenging conditions to test solar technologies.

II. SolarLab presentation

The SolarLab is the result of four years of continuous work by ICAM students at HVL. Each semester, a new student contributes to improving the passive house and its energy systems. The aim is to provide students with hands-on experience on how solar energy can power a small building, while facing the challenging climate of Bergen.

The project has known several phases as described below:

1. Selection of the building that would become the SolarLab. The installation of **12 solar panels**, 6 monocrystalline (275Wp) and 6 polycrystalline (285Wp) panels for a total power of 3.36kWp as well as **6 bi-facial solar panels** mounted vertically connected to **Sunny Boy inverters** to convert the direct current produced by the solar panels into alternative current. Each inverter can convert up to 1.5kW but actual production is below 1 kW per hour due to panel limitations.
2. Real-time display system via the Raspberry Pi. To make the Raspberry communicate with the inverters the former student had to set up Modbus TCP protocol. The management of the power produced, and consumption is key in this project making **Sunny Island Inverter** a crucial component of the lab. Sunny Island inverter allows to manage energy flow between solar panels, batteries and electrical loads. **SMA Energy Meters** were installed to collect data consumption of the different equipment as well as an **SMA Home manager** that supervises the entire system and collects data.
3. Setting up **TIGO system** which allows the user to monitor the production of each panel in real time. The initial Power Oak battery was replaced with a **Tesvolt TS25**, offering better reliability and compatibility.
4. Modelling of the passive house and its surrounding landscape on Sketchup and Skelion to optimize solar panels orientation and to justify its location to HVL board.
5. Installation of a **pyranometer** (Kipp & Zonen SMP11) and development of a program on Python to collect, store and update the data on the website.
6. System optimization: **Flexit CI 600 double-flow** ventilation system was installed and a **backup generator** tethered to HVL main grid was connected to ensure continuous power to the passive house when solar panels are not producing. **A heater** was installed and connected to a remote-control app (Shelly) to dump the excess of energy produced by the solar panels.

Here is an overview of the system:

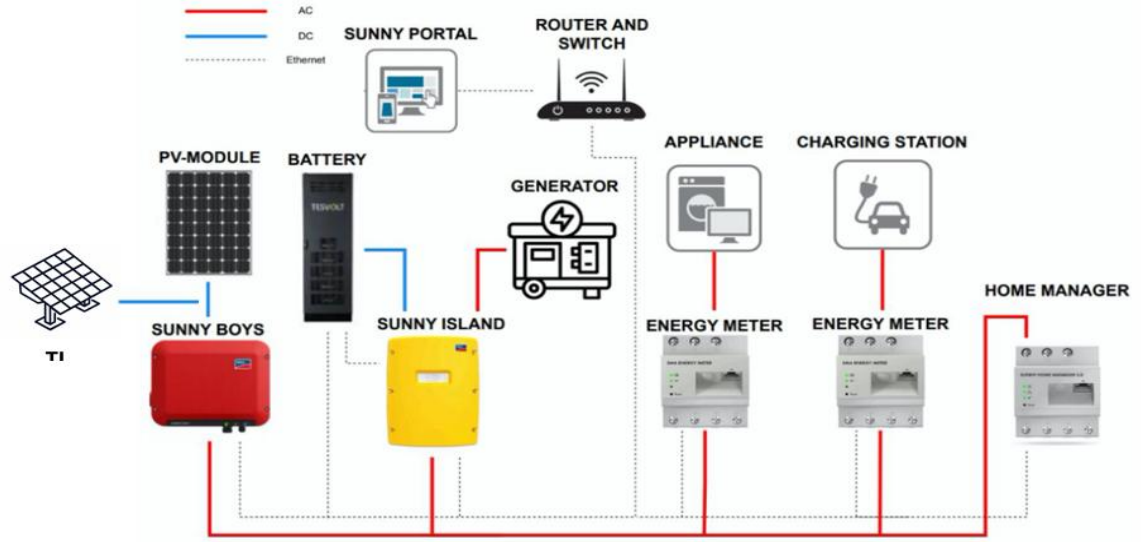


Figure 3: SolarLab system

III. Contribution to the project

My missions were multiple going from managing the passive house to its improvement and automation as well as fixing other minor problems that could be encountered during the semester. The main missions of this master thesis were the automation of the heater, fixing the storage capacity of the website and automating multiple systems to better manage the SolarLab battery.

A. Heater Automation

The heater and the car charger are our main source of energy dump; without them the battery would reach 98% of its capacity and the inverters would turn themselves down. This shut down means that during a short period of time the SolarLab cannot gather production data as the inverters are not producing anymore. This event mainly happens during summer season, during the week but more often during the weekends when no one is working implying that the car charger is not connected to any car, making the heater the only source of energy dump.

The previous student installed a Shelly[1] 1PM Gen 3 device with a relay switch allowing its user to control the heater remotely. This method was very demanding as the student or the tutor had to regularly look at their phone to check the State of Charge (SOC) of the battery and anticipate activating the switch. It also demanded access to Wi-Fi to be able to connect to the Shelly App.

Given these difficulties my first mission was to automate the heater. As explained before the shelly allows its user to remotely control a switch or any electrical devices

[1] A Shelly switch is an intelligent switching module that operates via a WI-FI network. It replaces traditional mechanical switches and allows you to control electrical devices remotely.

through its application but it can also be automated. The shelly has different communication protocols and support different scripts: MQTT, HTTP, inbound and outbound Web sockets. To control the heater given the State of Charge of the battery we had to be able to read it from the Sunny Island inverter and to do so we had to choose the correct communication protocol to use.

1. Programming Language Choice

The entire control system is implemented in Python, which was already used for the pyranometer, is supported by raspberry and is more familiar to me. Python allows rapid development while maintaining clear and maintainable code, which is particularly relevant in our context. In addition, Python provides libraries for both HTTP-based communication and industrial protocols such as Modbus TCP.

2. Communication with Sunny Island: Modbus TCP

The Sunny Island inverter is interfaced using the Modbus TCP protocol, through the *pymodbus* library. This choice is aligned with the communication capabilities officially supported by Sunny Island systems, which exposes data (such as battery State of Charge) via a Modbus TCP server over Ethernet.

Modbus TCP is widely used in industrial energy systems due to its reliability and standardized register-based data model. In this project, the State of Charge is read from dedicated holding registers.

Moreover, Modbus TCP operates entirely on the local network and does not require external services. This makes it particularly suitable for a control loop that must remain operational, predictable, and independent from internet connectivity.

As Sunny Island does not provide a native HTTP, MQTT, or WebSocket API for direct data access, Modbus TCP represents the most appropriate and officially supported solution for retrieving battery and system data.

3. Communication with Shelly Devices: HTTP REST API

As for the Shelly devices, they are controlled using HTTP requests through their built-in REST API, implemented in the code using the *requests* library. This choice was motivated by the fact that Shelly devices natively expose relay control and status information via simple HTTP endpoints.

HTTP allows:

- direct ON/OFF control of relays,
- retrieval of the current relay state,
- stateless and independent communication cycles.

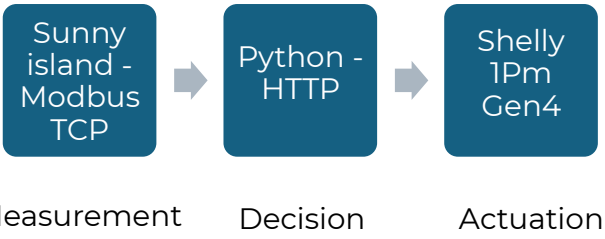
This stateless behaviour improves system robustness, as a temporary communication failure does not compromise the global execution of the control algorithm. Each request is independent and can be retried in the next control cycle.

Additionally, HTTP is easy to debug and maintain, and it integrates naturally with Python. Given that the Shelly devices are used exclusively for binary control (relay switching), HTTP satisfies the functional requirements without introducing unnecessary complexity.

To sum up, using HTTP and Modbus TCP avoids dependency on an intermediary service and results in a simpler and more transparent architecture.

4. Description of the automatic SOC based program

The developed program allows an automatic control strategy that manages an electrical load via a shelly, based on the SOC measured by the Sunny Island inverter.



5. Data acquisition from sunny Island and shelly control via HTTP

The battery SOC is retrieved from the Sunny Island using the Modbus TCP protocol. This functionality is implemented in the *SunnyIslandSOC* class.

The SOC value is stored in Modbus holding registers and decoded as a 32-bit unsigned integer:

```
result = self.client.read_holding_registers(30845, 2, unit=self.unit_id)
```

```
decoder = BinaryPayloadDecoder.fromRegisters(
    result.registers,
    byteorder=Endian.Big,
    wordorder=Endian.Big
)
soc = decoder.decode_32bit_uint()
```

Modbus TCP does not directly provide one 32-bit integer but provides two 16-bit modus registers which needs to be converted in one 32-bit unsigned integer allowing the user to read the percentage of battery.

This method ensures reliable access to energy parameter, which serves as the main input for the control logic.

The Shelly smart relay is controlled using its embedded HTTP REST API, which allows direct interaction through simple HTTP requests. The relay state is queried using:

```
r = requests.get(f"{self.base_url}/relay/0", timeout=5)
```

Relay activation and deactivation are done with this HTTP command:

```
requests.get(f"{self.base_url}/relay/0?turn=on", timeout=5)
requests.get(f"{self.base_url}/relay/0?turn=off", timeout=5)
```

These stateless requests ensure that each control action is independent, improving system robustness in case of temporary communication failures.

The program is in an infinite loop, executed every 60 seconds. At each iteration, the program reads the SOC and the relay state before applying the control logic. As the battery SOC increases or decreases slowly it does not require more frequent updates.

Furthermore, we chose that the shelly will activate the relay switch when the SOC reaches 90% and is turned off when it goes down to 85%. This hysteresis-based solution is described by the following program:

```
if soc >= self.soc_on and not self.current_state:
    self.shelly.turn_on()

elif soc <= self.soc_off and self.current_state:
    self.shelly.turn_off()
```

To activate the heater, we first verify that the state of charge is equal or greater than 90%, value that is stored in *soc_on* which has the same value as *self.soc_on* (the same goes for *self.soc_off = soc_off* = 85%) AND the *self.current_state* which is the current state of the shelly, ON or OFF in this case if it's off.

If both conditions are respected the heater is turned on, otherwise the heater is turned off.

To finish with, the program logs and displays in the command, key information such as the SOC values and relay state:

```
print(f"SOC: {soc}% | Shelly: {'ON' if self.current_state else 'OFF'}")
```

It allows the user to check the system by a simple look in the command.

6. Manual mode

The initial design of the system was based on a fully automated control. The heater was automatically activated when the battery State of Charge reached 90%, and deactivated when the SOC dropped back to 85%. However, this fully automatic system showed its limits as the manual activation of the heater was not possible. The program systematically overrode any user command. This constraint reduced the flexibility of the system.

To address this issue, a manual override mode was implemented. This mode allows the user to remotely activate the heater when required via the app, mainly to charge a car.

Here is the code that reads when the manual mode is activated.

```
if (self.previous_shelly_state is False and
    current_state is True and
    soc < self.soc_auto_on):

    self.mode = "MANUAL"
    self.soc_at_manual_activation = soc
    self.manual_stop_threshold = max(
        soc - self.manual_drop,
        self.soc_min_absolute
    )
```

The manual operating mode is designed to detect and manage user-initiated activations of the Shelly relay occurring outside the automatic control logic. Manual activation is identified when the relay state changes from OFF to ON while the battery SOC is below the automatic activation threshold. Once detected, the system switches to manual mode and stores the SOC value at the moment of activation. A dedicated stop threshold is then calculated, allowing a controlled battery discharge. During this phase, the automatic control logic is suspended to respect user intervention.

In manual mode, the system allows the battery SOC to drop of 15%, after which control is automatically returned to automatic mode. This provides a compromise between user flexibility and energy management constraints. See *Appendix 1* for the full program.

The control program was initially tested at lower SOC values to accelerate the validation process. First tests revealed communication issues between the controller and the Sunny Island inverter, which were resolved by correcting a Modbus configuration parameter based on external technical documentation.

Once communication was made, tests were conducted over an entire weekend. To ensure reliable long-term operation, the program was deployed as a systemd service on the Raspberry Pi, allowing automatic restart after power loss or system reboot.

B. Storage capacity for the website

1. Context and system description

The SolarLab has a website (SolarLab) where you can find information about the main use of the lab, its history but more importantly irradiance data from the pyranometer installed on the SolarLab and the one installed in the University of Bergen (UIB). This website therefore plays a key role in monitoring and sharing the laboratory's solar resource data.

Since one or two years the SolarLab has been equipped with its own Wi-Fi router to ensure continuous internet connection. Eduroam, the school Wi-Fi, occasionally stops working which would cause some data loss. Since the installation of the router, all the lab equipment's have been connected to it and as the internet subscription is around 5 GB, we consume it quite quickly.

2. Problem identification

We noticed that the irradiance graphs were not updated anymore to the website. Ensuring the availability and continuity of these data is essential for both monitoring purposes and external users of the platform.

3. Initial Hypotheses and verification

Initially, a power outage of the laboratory during the summer period was suspected to be the cause of the system malfunction. To verify this assumption, the production data available on Sunny Portal, the application used to collect the laboratory's energy production data, were analysed. The suspected shutdown period was cross-referenced with the corresponding production curves, which showed that the laboratory was still producing energy at that time. This observation ruled out a power supply interruption.

Another possible cause was related to network connectivity. An analysis of the mobile network operator's application revealed that the maximum data allowance of the Wi-Fi subscription had been reached between the 19th and 22nd of June and July, explaining why the graphs were not updated anymore. Further investigation showed that excessive data consumption was caused by the frequent transmission of images to the website. The large volume of uploaded images significantly increased data usage, ultimately exhausting the monthly data quota and leading to a loss of network connectivity.

The images are irradiance graphs, generated by a python script made by a former student. It collects the irradiance data, sends it to an excel file, makes 3 graphs of it (real time, last 3 days and 1 week) and sends it every 10 minutes to the website, which means that 432 images are sent to the website every day.

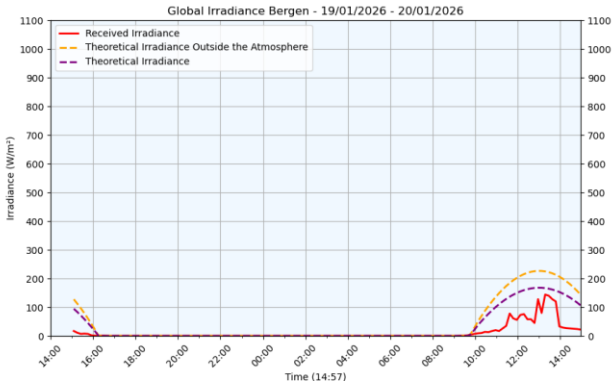


Figure 4: 24h irradiance

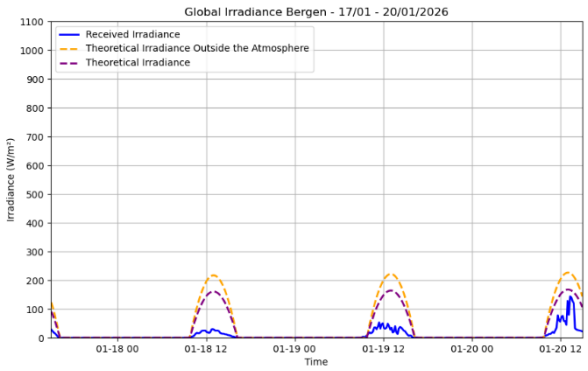


Figure 5: 3 days irradiance

4. Implemented solutions

To ensure that sufficient data allowance remained available until the end of each month, we decided to reduce the frequency of graph uploads to the website. The graphs are now sent every hour instead of every ten minutes, dividing by six the number of images transmitted per day and significantly reducing monthly data usage.

Additionally, it was observed that when the program was restarted, data from previous days were not included in the generated graphs. To address this issue, the script was modified to reload historical data at startup. Historical data are loaded once using the load_historical_data() function, which reads daily CSV files from the previous seven days, reconstructs complete timestamps, and stores the measurements in dedicated lists. During execution, new data are appended in real time, while the weekly graph is updated using a sliding 168-hour time window, as shown in Figure 6.

```
for days_back in range(7):
        target_date = current_time - timedelta(days=days_back)

        year = target_date.year
        month = target_date.month
        day = target_date.day

        desktop_path = os.path.expanduser("~/Desktop")
        file_path = os.path.join(
            desktop_path,
            "All data",
            f"{year:04d}",
            f"{month:02d}",
            f"{day:02d}_{month:02d}_{year:04d}.csv"
        )
```

Figure 6: 168h data recovery

The exact time points from the CSV file are then rebuilt

```
time_str = row[0]  # "HH:MM:SS"
hour, minute, second = map(int, time_str.split(':'))

data_datetime = target_date.replace(
    hour=hour, minute=minute, second=second, microsecond=0
```

To make sure that only the last 168 hours values are recovered we filter the values

```
if data_datetime >= start_time:
start_time = current_time - timedelta(days=7)
```

Then the values are uploaded to the lists used for the graphs

```
list_time.append(data_datetime)
list_radiation.append(radiation_value)
```

The same logic is applied to the 72 hours graph.

Since these changes the graphs have always been updated to the website.

It should be noted that sending graphs to website was not the only cause of hight data consumption, the computers used in the lab were also connected to the local Wi-Fi. An alternative solution would have been to change the contract to have more Giga Bytes, however this solution would have been more expensive.

C. Smart battery management for automated systems

The SolarLab can now dump battery automatically through the heater, but the heater should not be our main source of energy dump, the car charger should be as it's a more useful way to use the battery instead of heating the planet.

The car charger is a plug-and-charge unit from the Amina 1 brand, meaning that no matter the battery state of charge of the lab it's going to charge the car. We usually use it in place of the heater during weeks, but it requires us to call the owner of the electric BMW, get the keys and plug the car, very demanding.

Following the same logic as the heater, we chose to automate all dumping battery equipment making it easier for battery management, this became my last mission.

The management of the battery was the most challenging task of the semester. Understanding how all the components are connected between them, which cable to choose, which contactor were adequate and how to program everything in one singular code.

First, we had to come up with the charging priority. Which car charger will be activated first? at what percentage? When will the heater be needed?

For the moment only one car charger is installed the second will be purchased later.

The first idea was to prioritize the car charger already installed CC1 which would activate itself when SOC would be between 60% and 75% then between 76% and 89% we would also activate the second car charger and finally the heater would be activated when the SOC would be greater or equal to 90% but we would deactivate power from one of the car charger in our case CC2 otherwise it would trigger the fuse (Appendix 2). CC1 is already set on 10A so if we were to install another car charger, we could set its intensity properly to make sure that if both car chargers are working simultaneously, it would not trigger the fuse.

This logic, although effective, shows certain limitations. In the absence of hysteresis, the battery can frequently oscillate between two states. For example, when the SOC is between 76% and 89%, both CC1 and CC2 are active. However, if there is a temporary drop in production (due to cloud cover), the SOC may fall back to 75%, which leads to the deactivation of CC2. As soon as solar production becomes sufficient again and generated energy exceeds consumption, the SOC quickly rises back to 76%, triggering the reactivation of CC2.

These repeated transitions between the CC1 and CC2 states can occur over short periods and result in frequent on/off cycling of the charging process. This behaviour is damaging to electric vehicle charging, as it can cause premature wear of both the charger and the vehicle's charging system. Implementing hysteresis is therefore necessary to stabilize state transitions and protect both the charger and the vehicle.

Given this information a few changes were made in order to protect the cars and their chargers. A hysteresis system with different energy levels was put in place as presented in Figure 7. The detail of the energy levels will be presented in III.C.5: energy level concept

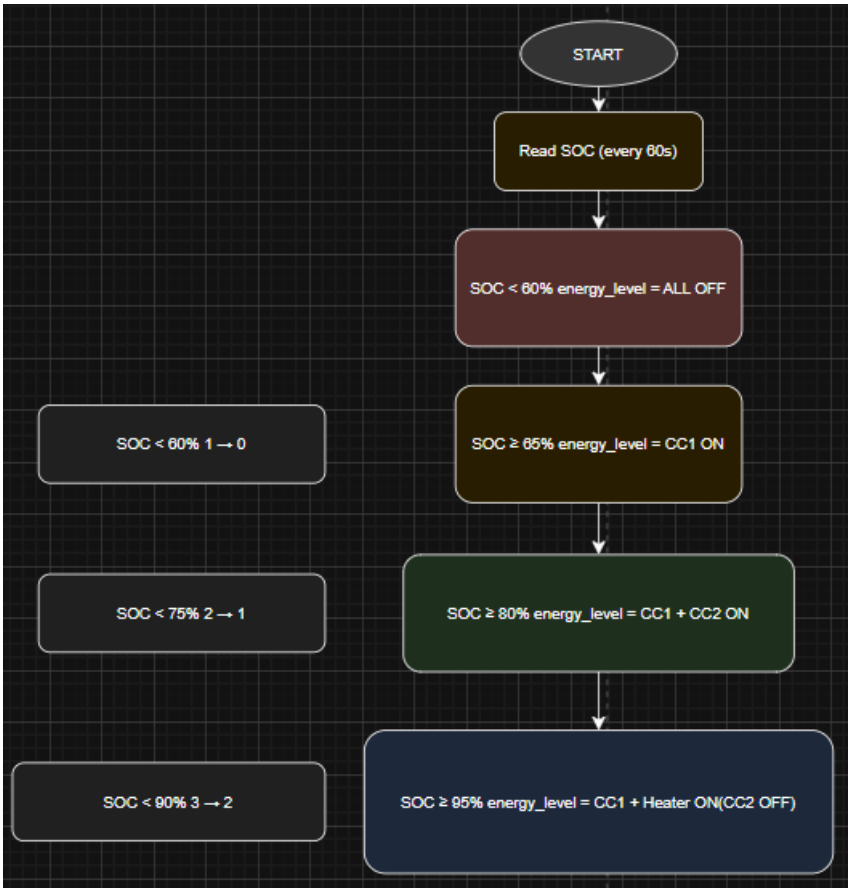


Figure 7: Hysteresis logic

1. Programming

This Python program is designed to automatically control electrical loads (two chargers and one heater) using Shelly 1PM Gen4 smart relays, based on the battery State of Charge provided by a Sunny Island inverter. The programming is based on the heater automation program as communication is the same (HTTP and Modbus TCP), but the decision logic is more complex as we now have 3 shelly. An internal energy state (energy_level) was defined and the use of hysteresis thresholds to manage state transitions were added.

Each Shelly device is represented by an instance of the ShellyGen4 class. This abstraction allows the program to treat each load in a uniform way, regardless of its physical function.

```
shelly_cc1 = ShellyGen4("Shelly CC1", "192.168.1.107")
shelly_cc2 = ShellyGen4("Shelly CC2", "192.168.1.105")
shelly_heater = ShellyGen4("Shelly Heater", "192.168.1.111")
```

The relay command logic and the battery state of charge acquisition are the same as the program described in III.A.5

2. Energy Level Concept

Instead of directly controlling loads based on SOC thresholds, the program introduces an intermediate variable called energy_level

```
energy_level = 0
```

This variable represents the current energy state of the system and simplifies the control logic.

Energy level	Active loads
0	All loads OFF
1	Charger 1
2	Charger 1 + Charger 2
3	Charger 1 + heater

This approach improves readability and makes the system easier to extend or modify. These variables are implemented in the program. It defines different SOC thresholds for increasing and decreasing energy levels.

Increasing level logic:

```
if energy_level == 0 and soc >= 65:
    energy_level = 1
elif energy_level == 1 and soc >= 80:
    energy_level = 2
elif energy_level == 2 and soc >= 95:
    energy_level = 3
```

Decreasing level logic:

```
elif energy_level == 3 and soc < 90:
    energy_level = 2
elif energy_level == 2 and soc < 75:
    energy_level = 1
elif energy_level == 1 and soc < 60:
    energy_level = 0
```

This hysteresis creates buffer zones[2] that prevent rapid state oscillations and ensure stable system behaviour.

[2] A buffer zone is a range of values intentionally left between two decision thresholds where no action is taken, in order to ensure stability and robustness of a control system

Once the energy level is determined, the corresponding Shelly commands are applied:

```
if energy_level == 0:
    shelly_cc1.turn_off()
    shelly_cc2.turn_off()
    shelly_heater.turn_off()
elif energy_level == 2:
    shelly_cc1.turn_on()
    shelly_cc2.turn_on()
    shelly_heater.turn_off()
```

The structure links each energy level to a specific load configuration.

The program is run every 60 seconds to ensure a progressive response of the system. For the entire program see Appendix 3.

3. Electrical relay box

Building the electrical relay box proved to be a challenging task, as it required a good understanding of how to organize, integrate, connect, and protect multiple electrical and electronic devices. The correct selection and dimensioning of components were crucial to ensure both the safety and the reliability of the system.

The relay box is installed after the main fuse box and is designed to supply power to two electric vehicle chargers (CC1 and CC2) as well as a heater. All these loads are monitored by a single SMA Energy Meter, which imposes a key constraint on the overall system design.

The dimensioning of the relay box had to consider the maximum measurable current of the energy meter, as the combined power consumption of both chargers and the heater is measured on the same device. According to the manufacturer's datasheet, the SMA Energy Meter can measure up to 63 A per phase conductor, corresponding to approximately 14.5 kW in a single-phase 230 V system.

The Amina 1 car charger has a maximum current rating of 32 A, delivering a peak power of 7.4 kW. The heater has a nominal power of 2.5 kW, corresponding to a current of approximately 11 A. Under these conditions, operating one car charger at full power simultaneously with the heater would exceed the maximum measurable current of the energy meter.

To address this limitation, the already installed car charger (CC1) was configured to operate at a reduced current of 10 A. Combined with the heater operating at 11 A, this results in a total current of 21 A. This configuration theoretically allows the second car charger (CC2) to operate at full power while remaining below the energy meter's 63 A limit.

However, an additional constraint arises from the protection devices. Each contactor within the relay box is protected by a rated fuse of 20A for both car

charger and 16A for the heater. The fuse initially protecting the energy meter was rated at 20 A and was replaced with a 25 A fuse. The recovered 20 A fuses was then used to protect one of the contactors. As a result, only 25 A remain available for the simultaneous operation of multiple devices on the protected lines.

To prevent nuisance tripping of the fuses and ensure safe operation, the second car charger was also configured to operate at a reduced current of 10 A. This configuration guarantees that neither the energy meter limits nor the fuse protection thresholds are exceeded under normal operating conditions.

I was asked to dimension the contactor as if we were to use the car charger at full capacity. This decision was made so if one day we wanted to use more intensity we didn't have to change the contactor. As said before max intensity of an Amina 1 car charger is of 32A meaning that the contactor should be of 32A or more. The only contactor available was of 40A, perfect for our needs. Each contactor is then wired to a specific shelly allowing automatic control. The electrical schematics can be found in Appendix 4.

Cables had to be bought in order to add the new car charger, around 14 meters of 3G6mm^2 cable. The section was chosen given the maximum power that could be delivered to the car charger and to respect regulations. To determine the correct section, we used the table Figure 8.

MONOPHASÉ		Section										
Puissance	Intensité	1,5 mm²	2,5 mm²	4 mm²	6 mm²	10 mm²	16 mm²	25 mm²	35 mm²	50 mm²	70 mm²	95 mm²
500 W	2,3 A	100 M	165 M	265 M	395 M							
1 000 W	4,6 A	60 M	84 M	135 M	200 M	335 M	530 M					
1 500 W	6,8 A	33 M	57 M	90 M	130 M	225 M	355 M	565 M				
2 000 W	9 A	25 M	43 M	68 M	100 M	170 M	265 M	430 M	595 M			
2 500 W	11,5 A	20 M	34 M	54 M	80 M	135 M	210 M	340 M	470 M	630 M		
3 000 W	13,5 A	17 M	29 M	45 M	66 M	110 M	180 M	285 M	395 M	520 M		
3 500 W	16 A	14 M	24 M	39 M	56 M	96 M	155 M	245 M	335 M	450 M		
4 000 W	18 A	-	21 M	34 M	49 M	84 M	135 M	210 M	295 M	395 M	580 M	
4 500 W	20 A	-	19 M	30 M	44 M	75 M	120 M	190 M	260 M	350 M	515 M	
5 000 W	23 A	-	-	27 M	39 M	68 M	105 M	170 M	235 M	315 M	460 M	630 M
6 000 W	27 A	-	-	23 M	32 M	56 M	90 M	140 M	195 M	260 M	385 M	530 M
7 000 W	32 A	-	-	-	28 M	48 M	76 M	120 M	170 M	225 M	330 M	460 M
8 000 W	36 A	-	-	-	-	42 M	67 M	105 M	145 M	195 M	290 M	400 M
9 000 W	41 A	-	-	-	-	38 M	60 M	94 M	130 M	175 M	255 M	355 M
10 kW	45 A	-	-	-	-	34 M	54 M	84 M	120 M	155 M	230 M	320 M
12 kW	55 A	-	-	-	-	-	45 M	70 M	98 M	130 M	190 M	265 M
14 kW	64 A	-	-	-	-	-	38 M	60 M	84 M	110 M	165 M	230 M
16 kW	73 A	-	-	-	-	-	-	53 M	74 M	99 M	145 M	200 M
18 kW	82 A	-	-	-	-	-	-	47 M	65 M	88 M	125 M	175 M
20 kW	91 A	-	-	-	-	-	-	-	59 M	79 M	115 M	160 M

Limite d'échauffement

Droits d'auteur enregistrés © Reproduction interdite

Figure 8: Section diameter of a cable given the intensity

The result after wiring all components is shown in Figure 9.

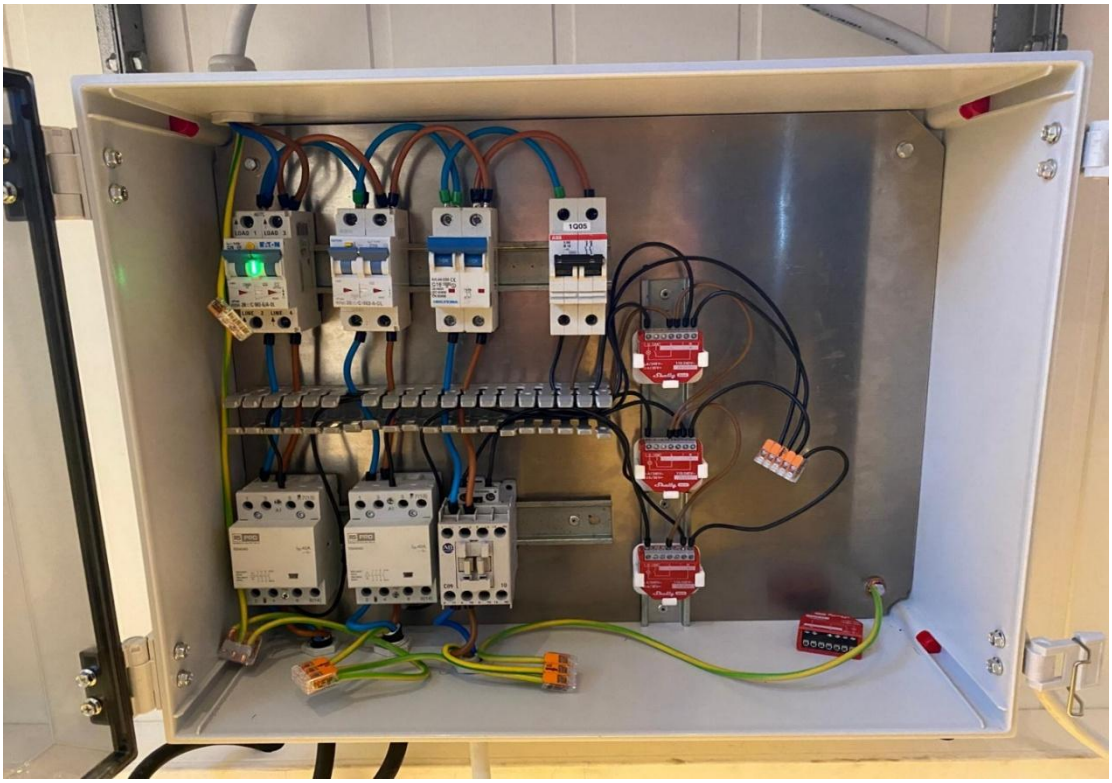


Figure 9: Relay box wiring

4. 3D Modelling

3D support for the shelly were designed on SolidWorks and printed with HVL 3D printer. They are easy to mount on a din rail as we install them using rivets that goes through the support to the aluminium plate. The support shape allows the user to install without difficulty the shelly device while maintaining it properly.

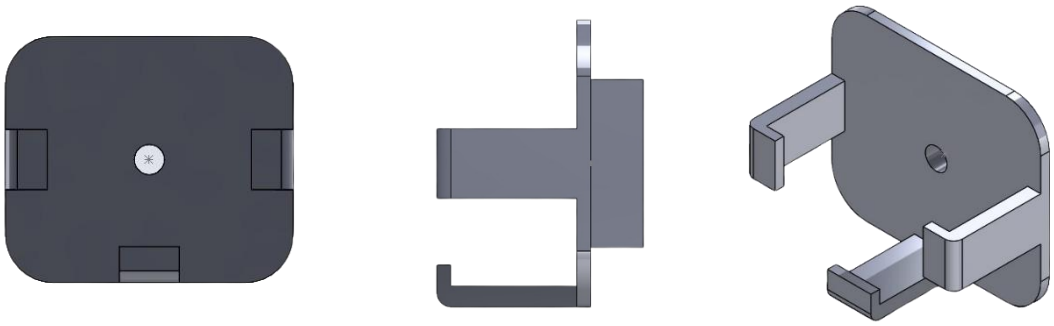


Figure 16: Shelly 3D printed support

Once the electrical relay box was completed, additional outdoor equipment had to be installed. New power cables were pulled and protected using flexible conduits to shield them from rain and environmental exposure. During the installation

phase, it was observed that the existing electrical socket was exposed to water ingress, despite being certified IP44 (water-resistant). To address this issue, a custom protective cover was designed using SolidWorks and 3D printed.

The cover provides effective protection against water while maintaining ease of use for the end user when plugging or unplugging electrical equipment. Particular attention was paid to both functionality and aesthetics: the mounting system is concealed inside the cover, resulting in a clean and unobtrusive appearance while remaining simple to install. The attachment mechanism consists of two screws fixed to the wall. The cover is mounted by aligning its internal slots with the screw heads and sliding it downward to lock it in place. To ensure the stability of the cover, two additional screws are fixed at its bottom as shown in Figure 11.

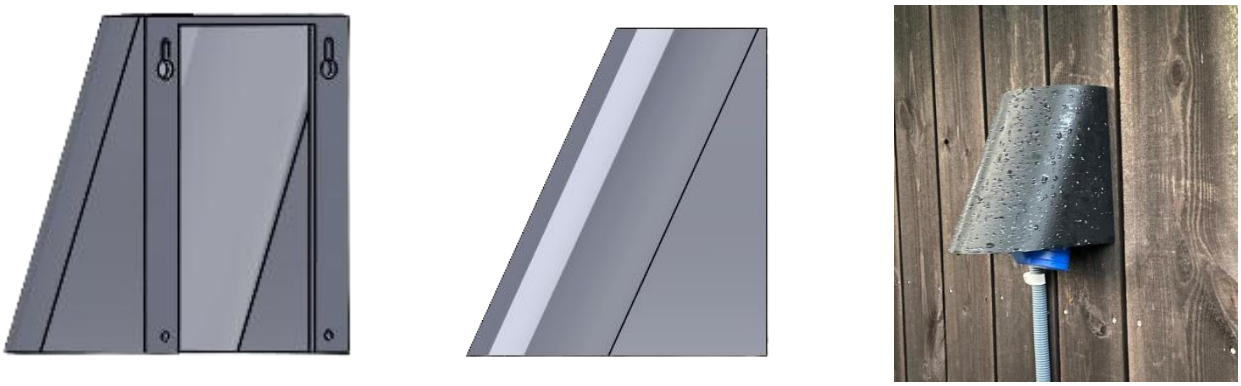


Figure 17: 3D printed socket cover

This mounting solution allows for quick installation and removal of the cover, making maintenance operations easier while ensuring reliable protection of the electrical connections in outdoor conditions. We had a lot of spare flexible water-resistant tubes and as I was outside, I used it to protect the cable of the pyranometer that was hanging along the wall as my final task.



Figure 18: Installed sockets cover

Conclusion

This project allowed me to automate the SolarLab, starting with the heater and then integrating the electric vehicle chargers. Setting up the energy states and hysteresis thresholds made the system more stable, protected the equipment, and prioritized charging the cars first. Overall, the lab is now much closer to running on its own, which was really satisfying to see in action.

On a personal level, I learned a lot about programming, electrical design, procurement of materials, manufacturing and installation of an electical system. But it wasn't just technical, working in Norway was a completely new experience. The way people work, the environment, and even small daily routines were different from what I'm used to in France. It made me adapt, communicate differently, and see how a team can solve problems in a different culture. Living and working there gave me a lot of perspective, both personally and professionally, and I think it's something I'll carry with me for a long time.

Next steps recommendations

Test the new battery management program in real condition and put some tags on the box so that people can know which shelly control which device.

It could be interesting to create some logs to see when the power is being drawn from the lab to the car chargers and the heater. As all of them are connected to one energy meter we will know the overall power taken. The energy values are displayed on sunny portal but knowing by a single look at the curve color or any distinctive markers which device consume how much power could be interesting. Using the energy meter would be the first option.

A second option would be using the shelly 1PM Gen 4 as they are meant to measure current up to 16A. The devices are mostly used around 10A it could, in theory, be possible to get power consumption from them.

Bibliography

Energie Use Per Person. (2025, 07 7). Retrieved from Our World in Data: https://ourworldindata.org/grapher/per-capita-energy-use?tab=discrete-bar&time=2024

Energy Institute. (2026, 01 9). *Statistical Review of World Energy*. Retrieved from Explore Data: https://ourworldindata.org/grapher/solar-energy-consumption?tab=line&time=earliest..2024

Gholmi, H. (2024). *Technical potential of solar energy in buildings across Norway: capacity and demand.* Retrieved from ScienceDirect: https://www.sciencedirect.com/science/article/pii/S0038092X24004535

IAE, Paris. (2025). *Global Energy Review 2025*. Retrieved from IAE: https://www.iea.org/reports/global-energy-review-2025

Norwegian Ministry of Energy. (2025). *Energy facts Norway*. Retrieved from energifaktanorge: https://energifaktanorge.no/en/norsk-energiforsyning/kraftproduksjon/#solar-power

Rosado, H. R. (2025). *Energy*. Retrieved from OurWorldinData: https://ourworldindata.org/profile/energy/norway

List of appendices

Appendix 1: Python program SOCShellyHybridAutomation 26
Appendix 2: Flowchart 3 Shelly automation 30
Appendix 3: FullSystem3Hyste python program 31
Appendix 4: Electrical schematics relay box 33

Appendix 1: Python program SOCShellyHybridAutomation

```python
#!/usr/bin/env python3
"""
Automatic Shelly controller based on Sunny Island SOC
- AUTO mode: Turns ON at 90% SOC, turns OFF at 85% SOC
- MANUAL mode: If manually activated, allows discharge down to 60% SOC (25% margin)
- PROTECTION: SOC never drops below 20% (absolute minimum SOC)
"""

import requests
import time
from datetime import datetime
from pymodbus.client.sync import ModbusTcpClient
from pymodbus.payload import BinaryPayloadDecoder
from pymodbus.constants import Endian

class ShellyController:
    def __init__(self, shelly_ip, username=None, password=None):
        self.base_url = f"http://{shelly_ip}"
        self.auth = (username, password) if username and password else None

    def get_status(self):
        try:
            r = requests.get(f"{self.base_url}/relay/0", auth=self.auth, timeout=5)
            if r.status_code == 200:
                return r.json().get("ison", False)
        except Exception as e:
            print(f"Shelly read error: {e}")
        return None

    def turn_on(self):
        try:
            r = requests.get(f"{self.base_url}/relay/0?turn=on", auth=self.auth, timeout=5)
            if r.status_code == 200:
                print(" Shelly plug TURNED ON")
                return True
        except Exception as e:
            print(f"Shelly ON error: {e}")
        return False

    def turn_off(self):
        try:
            r = requests.get(f"{self.base_url}/relay/0?turn=off", auth=self.auth, timeout=5)
            if r.status_code == 200:
                print(" Shelly plug TURNED OFF")
                return True
        except Exception as e:
            print(f"Shelly OFF error: {e}")
        return False

class SunnyIslandSOC:
    def __init__(self, ip_address="192.168.1.10", port=502, unit_id=3):
        self.client = ModbusTcpClient(ip_address, port=port)
        self.unit_id = unit_id

    def read_soc(self):
        try:
            result = self.client.read_holding_registers(30845, 2, unit=self.unit_id)
            if not result.isError():
                decoder = BinaryPayloadDecoder.fromRegisters(
                    result.registers, byteorder=Endian.Big, wordorder=Endian.Big
                )
                soc = decoder.decode_32bit_uint()
                return soc
        except Exception as e:
            print(f"SOC read error: {e}")
        return None

```

```python

class SOCShellyHybridAutomation:
    def __init__(self, si_ip, shelly_ip, soc_auto_on=90, soc_auto_off=85, manual_drop=20, soc_min_absolute=20):
        """
        Args:
            soc_auto_on: SOC threshold for automatic ON (%)
            soc_auto_off: SOC threshold for automatic OFF (%)
            manual_drop: Allowed SOC drop in manual mode (%)
            soc_min_absolute: Absolute minimum SOC - never go below (%)
        """
        self.si = SunnyIslandSOC(si_ip)
        self.shelly = ShellyController(shelly_ip)

        # Automatic thresholds
        self.soc_auto_on = soc_auto_on
        self.soc_auto_off = soc_auto_off

        # Manual parameters
        self.manual_drop = manual_drop
        self.soc_min_absolute = soc_min_absolute  # NEW: absolute minimum SOC

        # States
        self.current_shelly_state = None
        self.previous_shelly_state = None
        self.mode = "AUTO"  # AUTO or MANUAL
        self.soc_at_manual_activation = None  # SOC at manual activation
        self.manual_stop_threshold = None     # Computed stop threshold

    def log(self, msg):
        timestamp = datetime.now().strftime('%Y-%m-%d %H:%M:%S')
        print(f"[{timestamp}] [{self.mode}] {msg}")

    def detect_manual_activation(self, current_state, soc):
        """
        Detects whether the Shelly was activated manually
        (activation when SOC < automatic threshold)
        """
        if (self.previous_shelly_state is False and
            current_state is True and
            soc < self.soc_auto_on):

            # Check if manual activation is possible
            theoretical_stop = soc - self.manual_drop
            if theoretical_stop < self.soc_min_absolute:
                self.log(" MANUAL ACTIVATION detected but LIMITED by minimum SOC")
                self.log(f"   Current SOC: {soc}%")
                self.log(f"   Normal drop: {self.manual_drop}% → {theoretical_stop}%")
                self.log(f"   Applied limit: minimum SOC {self.soc_min_absolute}%")
            else:
                self.log(" MANUAL ACTIVATION detected")
                self.log(f"   SOC at start: {soc}%")
                self.log(f"   Allowed drop: {self.manual_drop}%")

            self.mode = "MANUAL"
            self.soc_at_manual_activation = soc
            # MODIFICATION: Use the maximum between normal drop and minimum SOC
            self.manual_stop_threshold = max(soc - self.manual_drop, self.soc_min_absolute)

            self.log(f"    Final stop threshold: {self.manual_stop_threshold}%")
            return True
        return False

    def run(self):
        self.log(" Starting SOC-Shelly HYBRID controller with protection")
        self.log(f"   AUTO: ON at {self.soc_auto_on}%, OFF at {self.soc_auto_off}%")
        self.log(f"   MANUAL: Allows {self.manual_drop}% drop")
        self.log(f"   PROTECTION: Absolute minimum SOC at {self.soc_min_absolute}%")

        # Initial state
        initial_state = self.shelly.get_status()
        if initial_state is not None:
            self.current_shelly_state = initial_state
```

```
            self.current_shelly_state = initial_state
            self.previous_shelly_state = initial_state
            self.log(f" Initial Shelly state: {'ON' if initial_state else 'OFF'}")

        try:
            while True:
                # Read SOC
                soc = self.si.read_soc()
                if soc is None:
                    self.log(" SOC read error")
                    time.sleep(30)
                    continue

                # Current Shelly state
                current_state = self.shelly.get_status()
                if current_state is not None:
                    self.previous_shelly_state = self.current_shelly_state
                    self.current_shelly_state = current_state

                # Detect manual activation
                if self.detect_manual_activation(current_state, soc):
                    pass  # Mode already changed in detect_manual_activation

                # Display status
                shelly_status = "ON" if self.current_shelly_state else "OFF"

                if self.mode == "MANUAL" and self.manual_stop_threshold is not None:
                    current_drop = self.soc_at_manual_activation - soc
                    max_theoretical_drop = self.soc_at_manual_activation - self.manual_stop_threshold
                    self.log(
                        f"SOC: {soc}% | Shelly: {shelly_status} | "
                        f"Drop: {current_drop:.1f}% / {max_theoretical_drop:.1f}% "
                        f"(limit: {self.manual_stop_threshold}%)"
                    )
                else:
                    self.log(f"SOC: {soc}% | Shelly: {shelly_status}")

                # === CONTROL LOGIC ===
                action_needed = None

                # ABSOLUTE PROTECTION: Immediate shutdown if minimum SOC reached
                if soc <= self.soc_min_absolute and self.current_shelly_state:
                    action_needed = "OFF"
                    self.log(f" MINIMUM SOC REACHED ({self.soc_min_absolute}%) → IMMEDIATE SHUTDOWN")

                elif self.mode == "AUTO":
                    # Automatic mode
                    if soc >= self.soc_auto_on and not self.current_shelly_state:
                        action_needed = "ON"

                    elif soc <= self.soc_auto_off and self.current_shelly_state:
                        action_needed = "OFF"

                elif self.mode == "MANUAL":
                    # Manual mode: shutdown when computed threshold reached
                    if soc <= self.manual_stop_threshold and self.current_shelly_state:
                        action_needed = "OFF"
                        actual_drop = self.soc_at_manual_activation - soc
                        if self.manual_stop_threshold == self.soc_min_absolute:
                            self.log(f" Absolute minimum SOC reached: {soc}%")
                        else:
                            self.log(f" Drop of {actual_drop:.1f}% reached")

                    # Return to AUTO mode when off
                    elif not self.current_shelly_state:
                        self.mode = "AUTO"
                        self.soc_at_manual_activation = None
                        self.manual_stop_threshold = None
                        self.log("Returning to AUTOMATIC mode")
```

```
                # === EXECUTE ACTIONS ===
                if action_needed == "ON":
                    self.log(f"SOC {soc}% ≥ {self.soc_auto_on}%  AUTO TURN ON")
                    if self.shelly.turn_on():
                        self.current_shelly_state = True

                elif action_needed == "OFF":
                    if soc <= self.soc_min_absolute:
                        self.log(f" SAFETY SHUTDOWN (minimum SOC: {self.soc_min_absolute}%)")
                    elif self.mode == "AUTO":
                        self.log(f" SOC {soc}% ≤ {self.soc_auto_off}%  AUTO TURN OFF")
                    elif self.mode == "MANUAL":
                        self.log(" Manual limit reached  SHUTDOWN")

                    if self.shelly.turn_off():
                        self.current_shelly_state = False
                        if self.mode == "MANUAL":
                            self.mode = "AUTO"
                            self.soc_at_manual_activation = None
                            self.manual_stop_threshold = None
                            self.log(" Returning to AUTOMATIC mode")

                # Wait before next check
                time.sleep(60)

            except KeyboardInterrupt:
                self.log(" Stop requested by user")
            except Exception as e:
                self.log(f" Error: {e}")

if __name__ == "__main__":
    # CONFIGURATION
    SUNNY_ISLAND_IP = "192.168.1.10"
    SHELLY_IP = "192.168.1.100"

    # Automatic thresholds
    SOC_AUTO_ON = 90    # Automatic ON at 90%
    SOC_AUTO_OFF = 85   # Automatic OFF at 85%

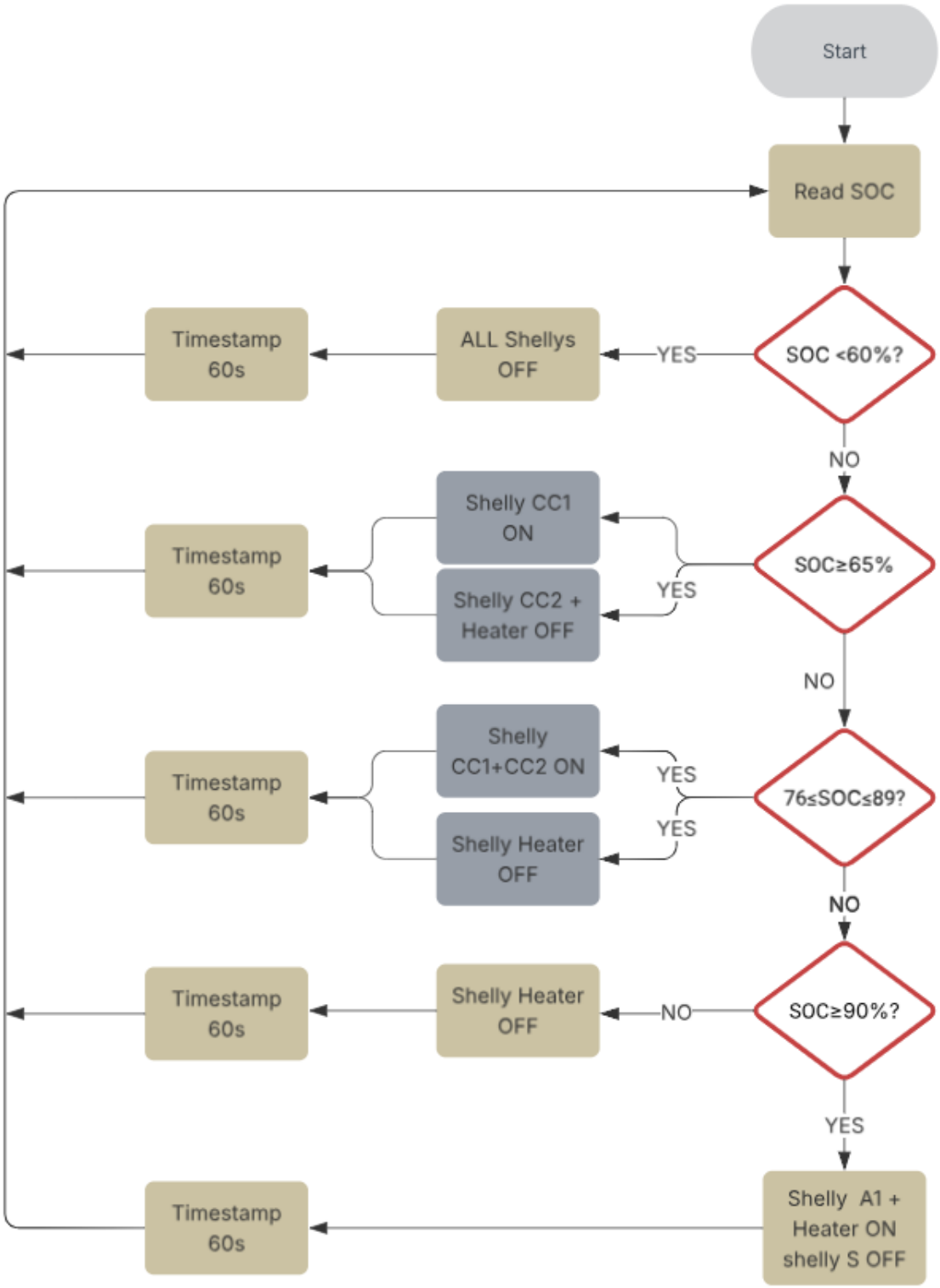
    # Manual mode parameters
    MANUAL_DROP = 20       # Allows 20% drop
    SOC_MIN_ABSOLUTE = 20  # NEW: absolute minimum SOC - never go below

    print("=" * 70)
    print("HYBRID SOC + SHELLY CONTROLLER WITH PROTECTION")
    print("=" * 70)
    print(f"AUTOMATIC mode: ON at {SOC_AUTO_ON}%, OFF at {SOC_AUTO_OFF}%")
    print(f"MANUAL mode:    Allows {MANUAL_DROP}% drop")
    print(f"PROTECTION:     Absolute minimum SOC: {SOC_MIN_ABSOLUTE}%")
    print(f"Sunny Island:   {SUNNY_ISLAND_IP}")
    print(f"Shelly:         {SHELLY_IP}")
    print("=" * 70)

    controller = SOCShellyHybridAutomation(
        si_ip=SUNNY_ISLAND_IP,
        shelly_ip=SHELLY_IP,
        soc_auto_on=SOC_AUTO_ON,
        soc_auto_off=SOC_AUTO_OFF,
        manual_drop=MANUAL_DROP,
        soc_min_absolute=SOC_MIN_ABSOLUTE
    )

    controller.run()
```

Appendix 2: Flowchart 3 Shelly automation



Appendix 3: FullSystem3Hyste python program

```python
import requests
import time
from datetime import datetime
from pymodbus.client.sync import ModbusTcpClient
from pymodbus.payload import BinaryPayloadDecoder
from pymodbus.constants import Endian
import urllib3

urllib3.disable_warnings(urllib3.exceptions.InsecureRequestWarning)


# ======================================================
# Shelly 1 PM Gen4
# ======================================================
class ShellyGen4:
    def __init__(self, name, ip, relay_id=0, timeout=5):
        self.name = name
        self.ip = ip
        self.relay_id = relay_id
        self.timeout = timeout
        self.url = f"https://{ip}/rpc/Switch.Set"
        self.last_state = None

    def set_relay(self, state: bool):
        if self.last_state == state:
            return

        payload = {"id": self.relay_id, "on": state}

        try:
            r = requests.post(
                self.url,
                json=payload,
                timeout=self.timeout,
                verify=False
            )
            if r.status_code == 200:
                self.last_state = state
                print(f"{self.name} {'ON' if state else 'OFF'}")
            else:
                print(f"Error {self.name}: HTTP {r.status_code}")
        except Exception as e:
            print(f"Error {self.name}: {e}")

    def turn_on(self):
        self.set_relay(True)

    def turn_off(self):
        self.set_relay(False)


# ======================================================
# Sunny Island SOC
# ======================================================
class SunnyIslandSOC:
    def __init__(self, ip="192.168.1.10", port=502, unit_id=3):
        self.client = ModbusTcpClient(ip, port=port)
        self.unit_id = unit_id

    def read_soc(self):
        try:
            result = self.client.read_holding_registers(30845, 2, unit=self.unit_id)
            if not result.isError():
                decoder = BinaryPayloadDecoder.fromRegisters(
                    result.registers,
                    byteorder=Endian.Big,
                    wordorder=Endian.Big
                )
```

```
                return decoder.decode_32bit_uint()
        except Exception as e:
            print(f"Error reading SOC: {e}")
        return None


# ======================================================
# Main program with hysteresis
# ======================================================

def main():

    # ==========================
    # CONFIGURATION TO EDIT
    # ==========================
    shelly_cc1 = ShellyGen4("Shelly CC1", "192.168.1.107")   # ### IP TO EDIT ###
    shelly_cc2 = ShellyGen4("Shelly CC2", "192.168.1.105")   # ### IP TO EDIT ###
    shelly_heater = ShellyGen4("Shelly Heater", "192.168.1.111")  # ### IP TO EDIT ###

    si = SunnyIslandSOC("192.168.1.10")   # ### SUNNY ISLAND IP TO EDIT ###

    # ==========================
    # CURRENT ENERGY LEVEL
    # ==========================
    # 0 = everything OFF
    # 1 = CC1 ON
    # 2 = CC1 + CC2 ON
    # 3 = CC1 + Heater ON
    energy_level = 0

    while True:

        soc = si.read_soc()
        now = datetime.now().strftime('%Y-%m-%d %H:%M:%S')
        print(f"\n[{now}] SOC : {soc}% | Level : {energy_level}")

        if soc is None:
            print("SOC not read → waiting")
            time.sleep(60)
            continue

        # ==========================
        # HYSTERESIS MANAGEMENT
        # ==========================

        # Level increase
        if energy_level == 0 and soc >= 65:
            energy_level = 1
        elif energy_level == 1 and soc >= 80:
            energy_level = 2
        elif energy_level == 2 and soc >= 95:
            energy_level = 3

        # Level decrease
        elif energy_level == 3 and soc < 90:
            energy_level = 2
        elif energy_level == 2 and soc < 75:
            energy_level = 1
        elif energy_level == 1 and soc < 60:
            energy_level = 0

        # ==========================
        # APPLY STATES
        # ==========================
        if energy_level == 0:
            shelly_cc1.turn_off()
            shelly_cc2.turn_off()
            shelly_heater.turn_off()

        elif energy_level == 1:
            shelly_cc1.turn_on()
            shelly_cc2.turn_off()
            shelly_heater.turn_off()

        elif energy_level == 2:
            shelly_cc1.turn_on()
            shelly_cc2.turn_on()
            shelly_heater.turn_off()

        elif energy_level == 3:
            shelly_cc1.turn_on()
            shelly_cc2.turn_off()
            shelly_heater.turn_on()

        time.sleep(60)


if __name__ == "__main__":
    main()
```

Appendix 4: Electrical schematics relay box

