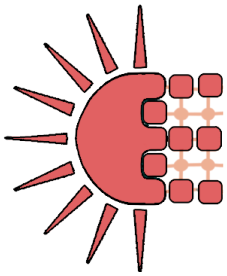
Imperial College London



ELECTRONICS DESIGN PROJECT

IMPERIAL COLLEGE LONDON

DEPARTMENT OF ELECTRICAL AND ELECTRONIC ENGINEERING

QUINTUPLE E

Smart Grid: Advanced Energy Control

Authors:
Ajith Kurian - 02136779
Intishar Misbahul - 02223194
Jay Mistry - 02223338
John Yeo - 02025314
Noam Weitzman - 02049854
Vishesh Mongia - 02288840

Supervisor:
Dr. Edward Stott

Second Marker:
NONE

September 27, 2024

Abstract

This report presents the design and implementation of an energy management system to optimise energy supply within a smart grid context. The project integrates a photovoltaic (PV) array for renewable energy generation, supercapacitors for energy storage, an external grid connection, and various LED loads. The primary goal is to efficiently manage energy usage while minimising reliance on the external grid, driven by demands provided by a third-party web server. Key components include maximum power point tracking (MPPT) algorithms for the PV array, cost-minimising algorithms utilising historical data, reinforcement learning for energy demand prediction, and multiple control systems to ensure proper functionality. Communication within the system is facilitated via MQTT, data is stored using a NoSQL database, and a user interface (UI) provides real-time and historical views of power flows. The project demonstrates effective energy management and offers a scalable model for integrating renewable energy sources into existing infrastructure, enhancing overall energy efficiency.

Contents

Chapter 1

Introduction

This project aims to design and implement a smart energy management system to power loads according to demands provided by a third-party web server. The system addresses the optimisation of energy management by efficiently balancing energy supply and demand.

Energy is extracted from a PV-array emulator configured with a bench power supply to mimic real-world solar conditions and employs switch-mode power supplies (SMPS) for energy conversion. Excess energy is stored in supercapacitors, ensuring a battery-free circuit, while energy discrepancies are mitigated by importing from or exporting to an external grid.

An algorithm is implemented to minimise the cost of imported energy and schedule the deferrable demands accordingly. The algorithm leverages historical data and Reinforcement Learning to predict and respond to energy demands dynamically. The system is tested using MQTT for communication, a NoSQL database for data storage, and a user interface for real-time monitoring of energy flows. The project provides a model for integrating renewable energy sources into existing infrastructures, optimising for overall efficiency.

This report lays out the progression of design, implementation and testing processes relating to both hardware and software forming the overall system.

Chapter 2

Preliminary Design and Planning

2.1 Requirement Analysis

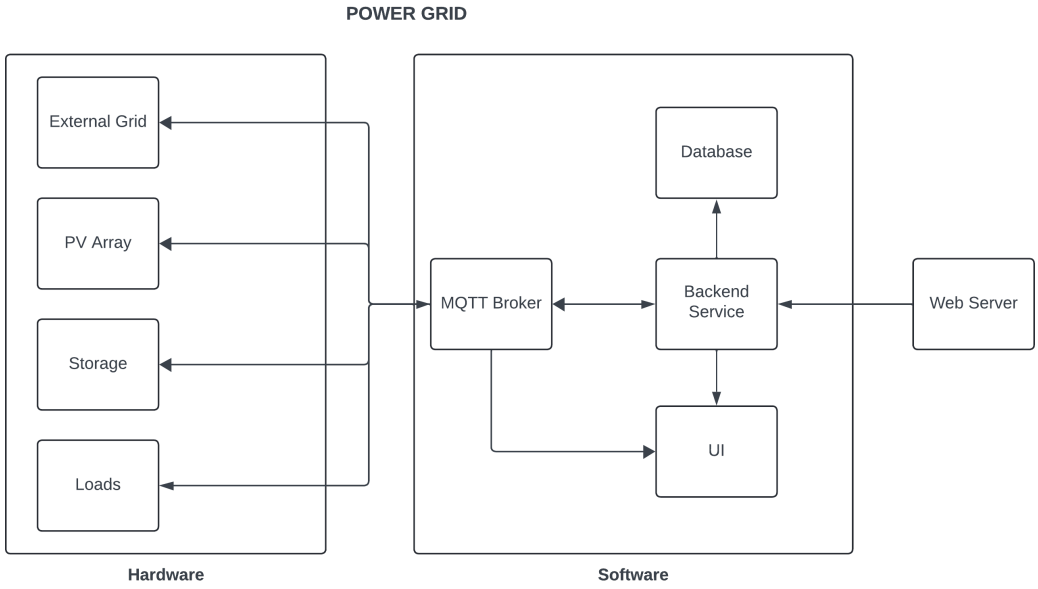


Figure 2.1: Overall schematics of the system

Figure 2.1 shows a top-level overview of the energy management system. The system should satisfy the following data flow:

1. The software receives data about the sun, price and demand requirements from the web server
2. The software processes the data to determine actions for each hardware component
3. The software successfully communicates these actions to each hardware component
4. The hardware components should be able to execute these actions with minimal error
5. The hardware components should be able to send the result of these operations back to the software
6. The software should be able to store the requirements, actions and results in the database and display them on the UI
7. The UI should allow the user to manually provide actions to control the hardware

SmartGrid Project Schedule

Project Start Date 5/21/2024 (Tuesday) Display Week 1

Project Lead

WBS	TASK	LEAD	START	END	DAYS	% DONE
1	**Emulation/Characterisation**			-		
1.1	Characterise PV Array	Intishar	Tue 5/21/24	Sun 5/26/24	6	100%
1.2	SuperCapacitor	Ajith/Intishar	Sat 2/03/18	Thu 2/08/18	6	60%
1.3	LED Drivers	Ajith	Wed 5/29/24	Fri 5/31/24	3	100%
1.4	External Supply	Ajith	Mon 5/27/24	Mon 6/03/24	8	75%
1.5	Emulate PV Array	Intishar	Tue 5/28/24	Thu 5/30/24	3	100%
2	**Grid Design**			-		
2.1	Theoretical Layout	Ajith/Intishar	Thu 5/23/24	Fri 5/24/24	2	100%
2.2	Tune the controller	Ajith	Fri 6/07/24	Sun 6/09/24	3	0%
2.3	Automation and optimisation	Ajith/Intishar	Wed 6/12/24	Tue 6/18/24	7	0%
2.4	Implement subsystem integration	Intishar	Mon 6/03/24	Sat 6/08/24	6	0%
2.5	Integrate communications code	Ajith	Wed 6/05/24	Sat 6/08/24	4	0%
3	**Webserver & Database**			-		
3.1	Research	ALL EIE	Tue 5/21/24	Sat 5/25/24	5	100%
3.2	Sever and Communication Design	John/Jay	Mon 5/27/24	Fri 5/31/24	5	100%
3.3	Database Design	Jay	Fri 5/24/24	Fri 5/24/24	1	100%
3.4	Server & Pico Implementation	Vishesh/Jay	Mon 5/27/24	Sun 6/02/24	7	80%
3.5	Database Implementation	Vishesh	Wed 5/29/24	Thu 5/30/24	2	100%
3.6	UI Design and Implementation	Jay	Wed 5/29/24	Thu 6/06/24	9	50%
4	**Trading Algorithm**			-		
4.1	Researching algorithm	John/Noam	Tue 5/21/24	Wed 5/22/24	3	100%
4.2	Price Prediction Model	John/Noam	Wed 5/29/24	Thu 6/6/24	1	30%
4.3	Reinforcement Learning	John	Tue 5/25/24	Fri 5/31/24	14	75%
4.4	Deferable Demand Scheduling	Jay	Mon 5/27/24	Mon 5/27/24	1	75%
5	**Presentation/Report**			-		
5.1	Brand Design	Vishesh	Mon 5/27/24	Thu 5/30/24	4	100%
5.2	Presentation	ALL	Mon 5/27/24	Mon 6/03/24	8	100%
5.3	Report	ALL	Tue 6/04/24	Wed 6/12/24	9	0%

Week 1 — 20 May 2024 | Week 2 — 27 May 2024 | Week 3 — 3 Jun 2024 | Week 4 — 10 Jun 2024 | Week 5 — 17 Jun 2024

Figure 2.3: Gantt Chart

2.2 Milestones, Contingencies & Planning

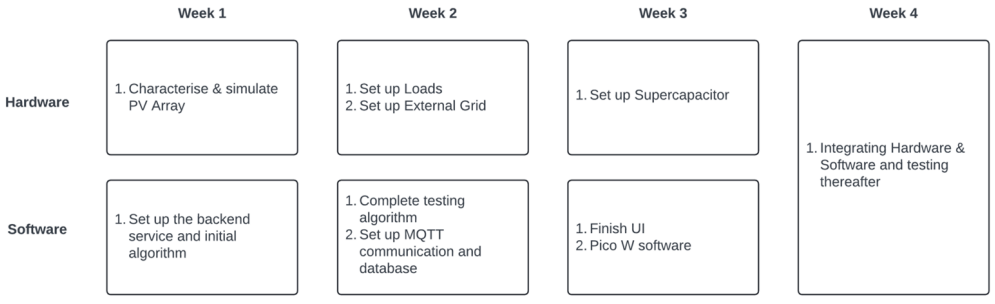


Figure 2.2: Planning and Milestones

During the design and implementation of the process, the system was primarily split between hardware and software, and therefore these two functions had individual milestones to satisfy for each week, before finally integrating them together in week 4. On the hardware side, the primary goals were to characterise, emulate and integrate the hardware components – PV array, loads, external grid, and supercapacitor. On the software side, the primary targets were to set up the following software systems – cost minimising algorithm, MQTT communication, database integration and UI. Each of these were split across the first three weeks. From week 4 onwards, integration of software and hardware was done, and thereafter testing was conducted. A more specific breakdown of the work done by each team member over the weeks is shown in Figure 2.3.

Chapter 3

System Design and Implementation

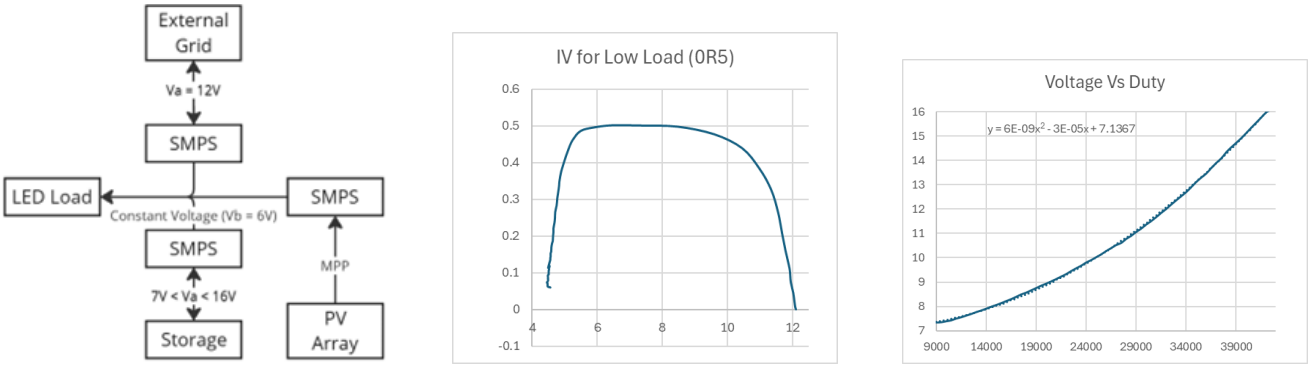
3.1 Hardware

3.1.1 Overview

The requirements for hardware are to implement a reliable power system for the LED loads whilst minimising the cost of importing energy from the external grid. A photovoltaic array with Maximum Power Point Tracking (MPPT) is used to efficiently power the loads. Meanwhile, supercapacitors store excess energy. The chosen configuration can satisfy energy demands whilst being flexible in maintaining grid integrity. For hardware details, check Appendix A.1

3.1.2 Grid Configuration

The Grid system must facilitate optimal power delivery to meet demand requirements efficiently. When transferring power from to the load, the voltage from energy generation modules (V_a) must be greater than the bus voltage (V_b). Given the voltage constraints of the Switch Mode Power Supply (SMPS) employed, the bus voltage (V_{bus}) is set at $6V$. This value is ideal as it can satisfy the LED demands and provide significant headroom for SMPS signals.



(a) Grid Functionality Design **(b)** Buck 0R5 IV Plot **(c)** Voltage vs Duty

Figure 3.1: Collection of hardware-related figures

- **PV Array**: Employing an incremental conductance (IncrCond) MPPT algorithm ensures that the PV array consistently supplies power close to its theoretical maximum, irrespective of irradiance levels.
- **Storage**: The SMPS operates within the voltage range of $[6V, 17V]$, unlike the LED drivers, which function between $[4.3V, 17V]$. Hence, the storage supercapacitors are limited between $[7V, 16V]$ to allow equal headroom from the max limits of the hardware for safety and maximising range. Storage utilisation is

fixed to 90% of max capacity to prevent overcharging and discharging beyond safe limits. Thus stability is ensured with constant power PID controller and minimal power flow oscillations.

- **External Grid Connection**: SMPS regulates (V_{bus}) at constant 6V, adjusting power import / export as necessary to stabilise the voltage.
- **LED Control**: The LED output power is regulated using a PID to manage the system's respective load and power consumption allowing for custom power requirements.

From this initial plan, a robust and efficient grid system was designed, satisfying demand requirements whilst ensuring the stability and longevity of components.

3.1.3 Photovoltaic Array Characterisation and Emulation

Due to unpredictable weather and the impracticality of testing the system with real sunlight, the PV array behaviour was characterised and emulated using data from live PV array testing. This emulation will allow for the testing of MPPT algorithms in a controlled lab environment.

Characterisation of Photovoltaic Array

PV array characterisation was broken down into the following steps:

1. Data Collection at Maximum Irradiance
   - Measure the maximum possible power point, open circuit voltage (V_{oc}), and short circuit current (I_{sc}) of the PV array at max irradiance
2. Determine MPP Values
   - Identify the maximum current (I_{m}) and voltage (V_{m}) at the maximum power point
   - Simulate the PV array behavior based on these parameters
3. High Load Characterisation
   - For large loads (such as 75Ω and 120Ω), boost mode is used to acquire the I-V data due to their high voltage requirements
   - Boost mode facilitates direct measurement of input power, which is more accurate than buck mode (used for lower loads)
4. Avoid Data Errors
   - Use a multimeter to accurately measure V_{oc} and I_{sc}
   - Ensure proper isolation of the SMPS from the measurement device to prevent power leakage and data inaccuracies

The following equations were used to characterise the array:

- Current at Maximum Power Point (I_{m}):

$$I_m = I_{sc} - \frac{V_{oc}}{R_P} \tag{3.1}$$

- Voltage at Maximum Power Point (V_{m}):

$$V_m = V_{oc} - I_m \cdot R_s \tag{3.2}$$

where R_P and R_s are parallel and series resistances respectively, to be used for emulation calculated from the equations above.

By correctly characterising and emulating the PV array, we ensure that the MPPT algorithms can be tested accurately, optimising the power output of the PV system despite the variability in natural sunlight conditions. The code used to acquire data via duty sweep can be found on GitHub by **DutySweepgithubcode**

Parameter	Value
Voc	12
Isc	0.47
Rp	400
Rs	37
Im (75Ω)	0.44
Vm (75Ω)	9.683
Pm (Exp)	4.26
Pm (Act)	4.1

Table 3.1: Characterisation Readings

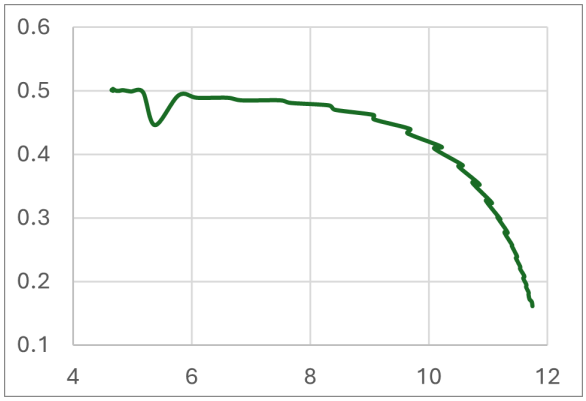


Figure 3.2: I-V plot for 75Ω

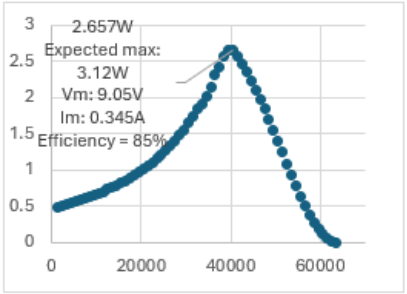
Emulation of Photovoltaic Array

To emulate the PV array and test MPPT algorithms, a PSU was configured with the measured open circuit voltage (V_{oc}) and short circuit current (I_{sc}), connected to SMPS in Buck Closed Loop mode. Control of the SMPS's duty cycle was achieved using a modified duty sweep programme recording voltage and current data.

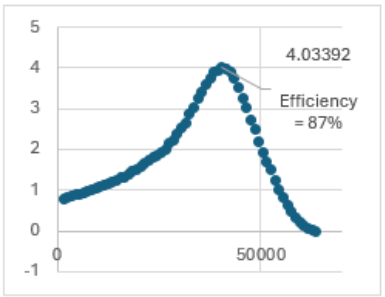
The plots below confirm near-identical relationship between the actual and emulated I-V behaviour, validating the accuracy of the experimental setup. Output Power versus Duty Cycle was considered for a more precise comparison to address inaccuracies observed in the I-V graph from Buck Mode which arise from inefficiencies. This approach avoids assumptions about input current, which is not directly measurable. Focusing on output power data allows us to validate the emulation process accurately. Series and parallel resistances were tuned to account for changes in the maximum power point using appropriate equations. Despite these adjustments, the MPP remained consistent at the same duty cycle, proving the reliability and effectiveness of the emulation approach.

Parameter	Value
Voc	12
Isc	0.47
Rp	100
Rs	10
Im (2R2)	0.35
Vm (2R2)	9.05
Pm(Exp)	3.12
Pm(Act)	2.7

(a) Emulation Readings



(b) Emulation using 2R2 as load for Power vs Duty



(c) Actual 2R2 load Power vs Duty using Sunlight

3.1.4 MPPT Algorithms

Incremental Conductance

The Incremental Conductance algorithm is commonly used for Maximum Power Point Tracking in photovoltaic systems. The operating point of a solar panel is continuously adjusted to maximise the power output under varying environmental conditions, especially changes in solar irradiance and temperature.

Instantaneous power P_{out} of the solar panel is given by:

$$P_{out} = V_b \cdot IL \tag{3.3}$$

where V_b is the voltage across the load and IL is the load current.

The IncrCond algorithm involves comparing the incremental conductance $\Delta IL/\Delta V_b$ with the instantaneous conductance $-IL/V_b$. This conductance is derived from the first derivative of the power-voltage (P-V) curve:

$$\text{Incremental Conductance} = \frac{\Delta IL}{\Delta V_b} \tag{3.4}$$

$$\text{Instantaneous Conductance} = -\frac{IL}{V_b} \tag{3.5}$$

MPPT Decision Logic: The algorithm adjusts the duty cycle of a DC-DC converter or the voltage applied to the load to ensure that the ratio of $\Delta IL/\Delta V_b$ matches $-IL/V_b$. Therefore, the operating point of the solar panel is always at the Maximum Power Point (MPP) on the P-V curve.

To prevent rapid oscillations around the MPP, a hysteresis band was introduced. This band determines how close the algorithm should steer towards MPP before adjusting, ensuring stability and reducing power fluctuations.

Implementation

Sensor Integration utilises current and voltage sensors (e.g., INA219) to measure IL and V_b respectively.

There is feedback control via a Proportional-Integral-Derivative controller or a similar control mechanism to dynamically adjust the duty cycle or voltage output based on the calculated conductance ratio:

$$u(t) = K_p e(t) + K_i \int_0^t e(\tau)\,d\tau + K_d \frac{de(t)}{dt} \tag{3.6}$$

It includes adaptive step size and delay adjustments to optimise the algorithm's response under varying solar conditions, ensuring no overshoot or oscillations when tracking MPP.

Advantages:

- Efficiency: Quickly converges to the MPP, maximising the energy extraction from the solar panel
- Robustness: Adapts well to changes in irradiance and temperature, maintaining stable operation

Perturb and Observe (P&O) MPPT Algorithm

The Perturb and Observe algorithm is an alternate method used for MPPT. It works by continuously perturbing the operating point of the PV panel and observing the resultant change in power output to determine the direction towards the Maximum Power Point (MPP).

The output power P_{out} of the solar panel was calculated using the equation:

$$P_{out} = V_b \cdot IL \tag{3.7}$$

where V_b is the voltage across the load (output voltage) and IL is the load current.

The algorithm perturbs the operating point by adjusting the duty cycle of a DC-DC converter or the voltage applied to the load. Then changes in output power are observed.

Implementation:

- If the change in power P_{out} increases, the algorithm continues to perturb in the same direction
- If the change in power decreases, the algorithm reverses the direction of perturbation

The goal is to continuously adjust the duty cycle or voltage to converge towards the MPP on the power-voltage curve of the solar panel.

To optimise convergence speed and stability, the algorithm adjusts the step size of perturbation dynamically based on the rate of change in power:

- If the power delta is small, the step size is reduced to fine-tune the adjustment
- If the power delta is significant, the step size is increased to speed the convergence towards MPP

Advantages:

- Simplicity: Easy to implement and understand, making it suitable for basic MPPT applications
- Effectiveness: Provides good tracking performance under steady-state and changing environmental conditions

Performance comparison of MPPT Algorithms

This section contrasts two distinct MPPT algorithms, Perturb and Observe (P&O) and Incremental Conductance (IncrCond). These algorithms were tested under varying load conditions to assess their performance in maximising power extraction from an emulated photovoltaic (PV) array.

Python scripts were developed to implement the MPPT algorithms and simulate varying load conditions. Key scripts are included in the repository:

- `MPPT_IncrCond.py`: Implements Incremental Conductance algorithm with adaptive step size and dynamic delay adjustment
- `MPPT_P&O.py`: Implements Perturb and Observe algorithm with oscillation handling and adaptive step size
- `MPPT_Irradiance.py`: Simulates varying irradiance conditions using a constant power algorithm, adjusting output power based on irradiance levels

The Perturb and Observe (P&O) algorithm demonstrated quick convergence to the MPP. This rapid response is advantageous for systems requiring fast adjustments to changing conditions.

However, P&O exhibited oscillations under certain load conditions, which impacted its stability. Sometimes, the algorithm became stuck at a lower power point, requiring periodic resets to function.

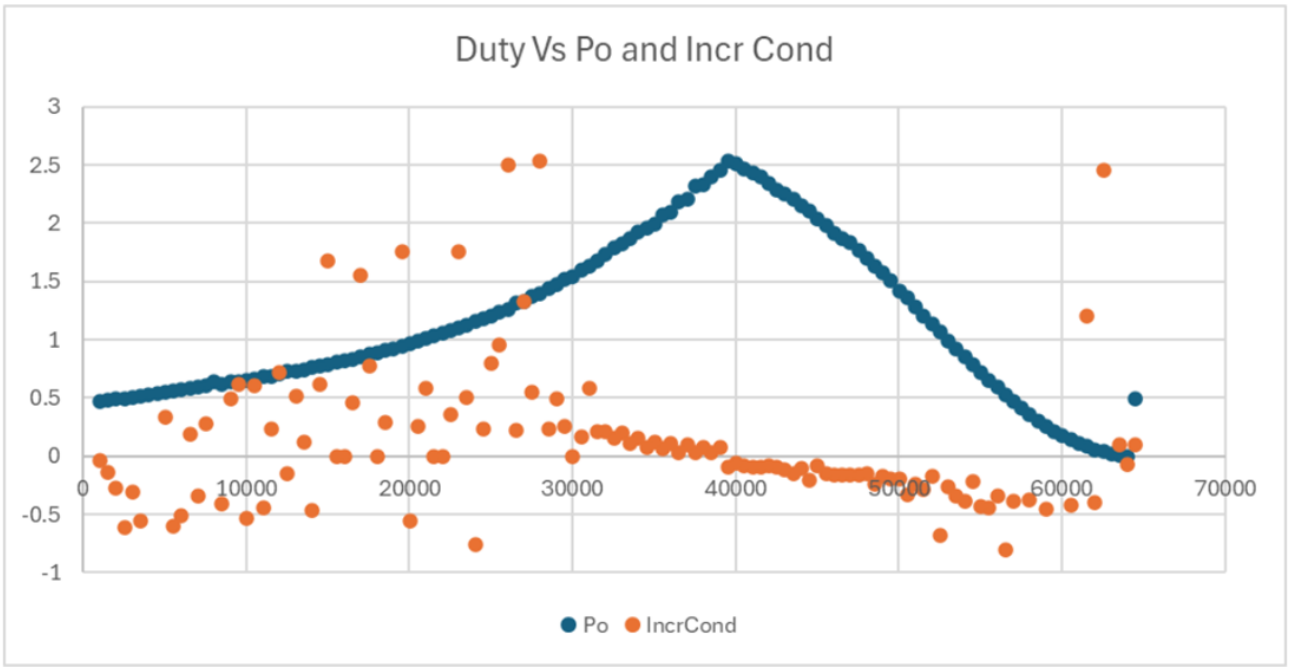


Figure 3.4: Data sweep result for Power and Conductance

The Incremental Conductance (IncrCond) algorithm encountered challenges due to hardware-induced noise affecting its stability. Implementing a moving average filter smoothed these fluctuations, enhancing stability.

Moreover, adaptive delay and step size adjustments were introduced. The adaptive delay function dynamically adjusts based on power changes, preventing rapid adjustments for small changes (up to 100 ms) and quicker responses for larger changes (down to 10 ms). Similarly, the adaptive step size function fine-tunes adjustments near the Maximum Power Point (MPP), reducing oscillations and maintaining stability.

These improvements made the IncrCond algorithm more robust, ensuring efficient power extraction from the photovoltaic array under dynamic load and irradiance conditions.

Average Power and Settling Time for P&O algorithm

The average power output for the P&O algorithm was recorded at 2.37 W.

The algorithm's response time, or settling time, was notably quick at 44 ms, reflecting its efficiency in promptly reaching the MPP.

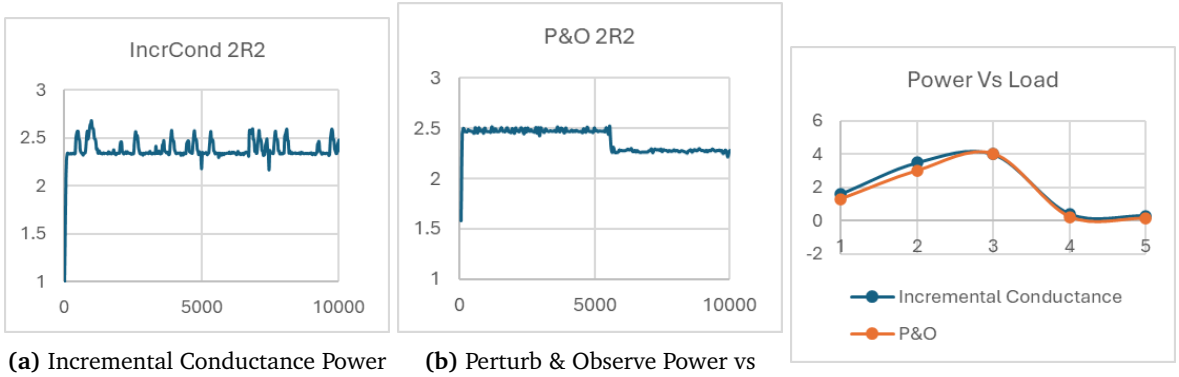
The Incremental Conductance (IncrCond) algorithm provided stable operation across various conditions. Although there were slightly more oscillations compared to P&O, these were minimal and did not significantly impact performance.

IncrCond adapted well to varying load conditions and effectively tracked the MPP across different scenarios without requiring any resets, demonstrating its robustness and reliability.

Average Power and Settling Time for IncrCond algorithm

The average power output for the IncrCond algorithm was slightly higher at 2.38 W.

The settling time was measured at 78 ms, which is longer compared to P&O but acceptable given its superior stability and consistent performance.



(a) Incremental Conductance Power vs Time

(b) Perturb & Observe Power vs Time

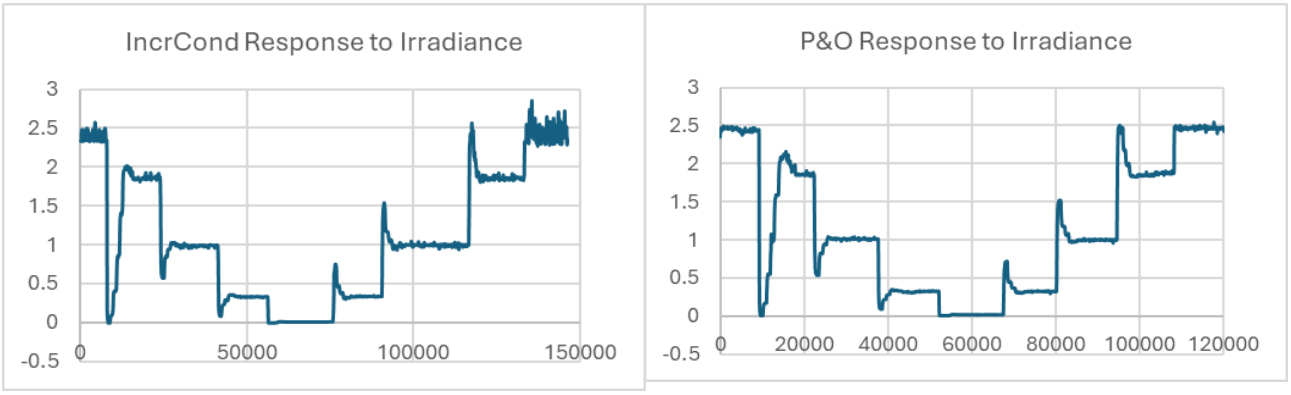
(c) MPPT vs Load

Figure 3.5: Various Characteristics

Irradiance Response Analysis

From the plots, it is evident that P&O algorithm performed poorly under changing irradiance conditions. This was simulated by varying PSU settings from 0.45A to 0.35A, 0.25A, 0.15A, 0.05A, and then back to 0.45A while keeping the voltage at 12V. The P&O algorithm struggled to maintain stable power output under these fluctuating conditions, often becoming stuck at suboptimal power points.

In contrast, the IncrCond algorithm maintained better performance and stability under the same varying irradiance conditions, as illustrated in the provided plots below. Demonstrates its superior adaptability and robustness in dynamic environments, making it a more reliable choice.



(a) Irradiance performance of IncrCond

(b) Irradiance performance of P&O

Demo Emulation of Irradiance

To emulate irradiance for testing purposes, a constant power controller was developed. This controller adjusts the power setpoint based on the maximum power point (MPP) characterised at 4.2W and irradiance values received from an external source. This approach allows for simulating different sunlight intensities.

The controller operates by adjusting the power setpoint using the formula $P_{\text{desired}} = \text{MPP} \times \text{Irradiance}$, where Irradiance is provided as a percentage. The implementation includes a PID controller to maintain stability and control the output power precisely during changing irradiance levels. This helps to test the overall system seamlessly inside the lab without depending on weather conditions.

Conclusion

While P&O offers quicker response times, IncrCond provides greater stability and consistency, particularly in environments with changing irradiance levels and load. Hence Incremental Conductance was implemented in the design of the grid.

3.1.5 LED Drivers

From the specifications, the LEDs, represent the end consumers and draw power from the grid corresponding to the values of the instantaneous and deferrable demands values supplied by the third-party web server.

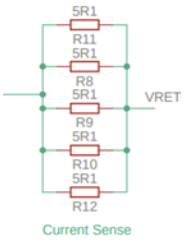
Characterisation

The I-V characteristics were identified through modifying the LED driver code to conduct a sweep of the LED for varying voltage levels. Output power was calculated through the current sense module in Figure 3.7a. Using the formula:

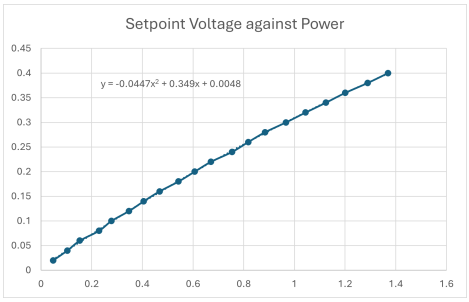
$$I_{out} = \frac{V_{RET}}{R_{total}} \tag{3.8}$$

with total impedance being calculated to 1.02Ω, the output power can then be calculated with the output voltage. The sweep was carried out for the various LED colours, but the results were virtually identical.

Figure 3.7b demonstrates the relationship between setpoint voltage and power follows the relationship $y = -0.447x^2 + 0.349x + 0.0048$. Safe operating voltage was found to be up to $4.5V$ and could output up to $1.4W$. Based on the restrictions showed in Figure 3.7c, a limit of $1W$ was implemented as power output did not increase significantly beyond the operating limits.



(a) Current Sense
LED_Driver_SCM.pdf



(b) Set-point Voltage Against Power

Specification	Value	Unit
Power Rating	1	W
Voltage Drop (varies with colour)	~3	V
Current Rating (also varies with colour)	~300	mA

(c) Restrictions
Appendix_Circuit_Specifications

Figure 3.7: Various Characteristics

Implementation

The LED's function is to output power at a certain power level depending on the requirements set by the web server. An initial algorithm was trialled where the set-point voltage of the PID controller is varied corresponding to the desired power demand using the relation between set-point voltage and power as shown in Figure 3.7b. Significant variations between desired power requirements and the actual output led to poor performance from the algorithm, as shown in the Table 3.2

Desired Power 1	Output Power
0.7	0.88
0.2	0.43
0.66	0.79
1	Nan

Table 3.2: Desired vs Output Power

An alternative design was implemented, keeping the setpoint at 0.3 thus maintaining the LED at the safe operating voltage. It was noted that the output power increased with higher duty values and hence was modified to obtain the required power value. This algorithm, shown in Figure 3.8, uses hysteresis to prevent the constant switching of output powers which can be seen in the program located in the **vishesh32quintuplee**.

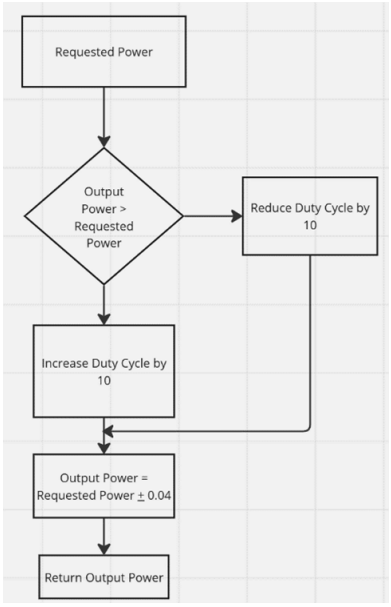


Figure 3.8: Power Control Algorithm

This program performed much better than the previous implementation, reaching accurate power values after an initial settling time represented in Figure 3.9. Increasing the values of kd and kp to 0.7 and 15 respectively, improved the power response of the LEDs. The PID parameters were initially tested at $0.46W$, and subsequently $0.71W$ once the adequate parameters were found through manual tuning.The average deviations from required power were found to be between ± 0.01, ± 0.02, which were found to be adequate for LED power demands.

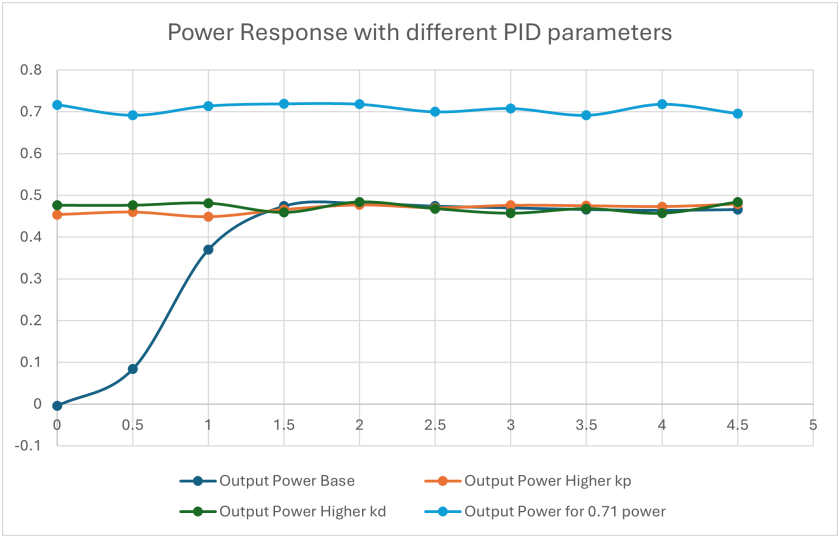


Figure 3.9: Settling Time

3.1.6 External Grid

This subsystem is responsible for modelling the import and export of energy from a separate power source, implemented with an additional PSU. The energy imported/supplied will be converted to a monetary value for use in the algorithm through server-side calculations.

Characterisation

A bidirectional SMPS was used to allow for power inflow and outflow to the external grid. The initial model for this implementation is presented in Figure 3.10. Furthermore, the energy imported/exported will be converted to monetary value for use in server-side algorithm calculations.

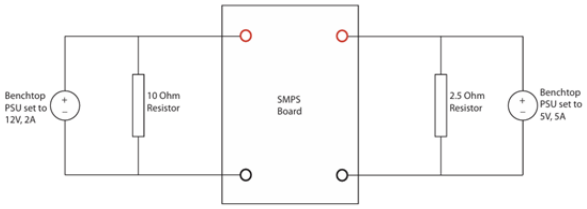


Figure 3.10: Bidirectional SMPS
Bidirectional_SMPS

With the setup in Figure 3.11, current flow can vary directions based on the duty cycle of the SMPS. The main requirement is to ensure the output voltage (V_b) was fixed to $6V$ (main bus voltage decided for the Power Grid system), the set-up was modified to fit these new parameters:

- External Grid PSU set to $12V$, $2A$
- Main bus PSU emulation configured to $6V$, $1A$

After trying boost mode open loop and realising it is unsuitable as the main bus voltage is stepped down from $12V$ to $6V$, Buck mode open loop was deemed more suitable for application to the external grid. The SMPS was manually configured to keep a constant bus voltage of $6V$, as seen in Figure 3.11 during intial testing.

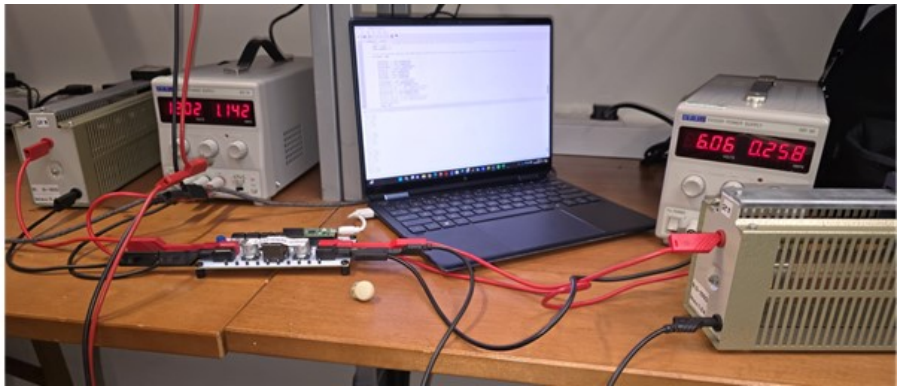


Figure 3.11: Buck Mode Open Loop Trial of External Grid

Implementation

To implement automated control an algorithm was devised following logic shown in Figure 3.12. The algorithm works to ensure duty is at a range between $5.9V$ and $6.1V$ and will increase or reduce duty respectively.

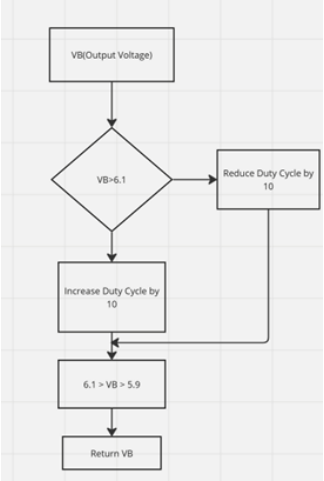


Figure 3.12: Voltage control automation

```python
# Saturation function for anything you want saturated within
def saturate(signal, upper, lower):
    return max(min(signal, upper), lower)

# This is the function executed by the loop timer, it simply
def tick(t):
    global timer_elapsed
    timer_elapsed = 1

def vb_at_bus_voltage(vb, pwm_ref):
    if vb > 6.1:
        pwm_ref = saturate(pwm_ref - 10, max_pwm, min_pwm)
    if vb < 5.9:
        pwm_ref = saturate(pwm_ref + 10, max_pwm, min_pwm)
    return pwm_ref
```

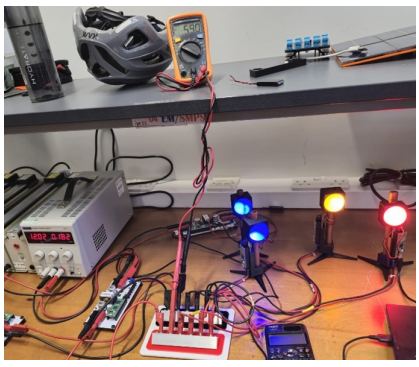
(a) Voltage Regulator Function

```python
# pwm_ref = saturate(65536-(int((vpot/3.3)*65536)),max_pwm,min_pwm) # convert the
pwm_ref = vb_at_bus_voltage(vb, pwm_ref)

# Energy calculation integration over 5 seconds
power = (va if iL < 0 else vb) * abs(iL)  # Power in watts
energy_accumulator += power / 1000.0  # Power in mW, loop runs every 1 ms

if time.time() - energy_start_time >= 5:
    energy = energy_accumulator  # Total energy in mWs over the 5-second period
    energy_accumulator = 0
    energy_start_time = time.time()
```

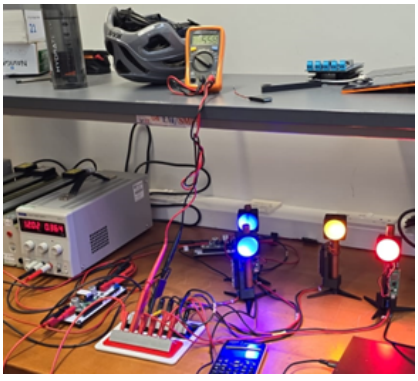
(b) Power calculation

Figure 3.13: Code

This program worked well to control the main bus voltage and further calculations for the imported / exported energy per tick were implemented as shown in Figure 3.13b. The finalised subsystem was tested with LED loads as shown in Figures 3.14. In Figure 3.14a, the LEDs were tested at 75% of max power demand and the main bus voltage was maintained at $5.9V$. In Figure 3.14b, the LEDs were at maximum power demand and the main bus voltage was found to initially drop to $5.6V$ before returning to $5.9V$.



(a) Bench



(b) Bench

Figure 3.14: Bench

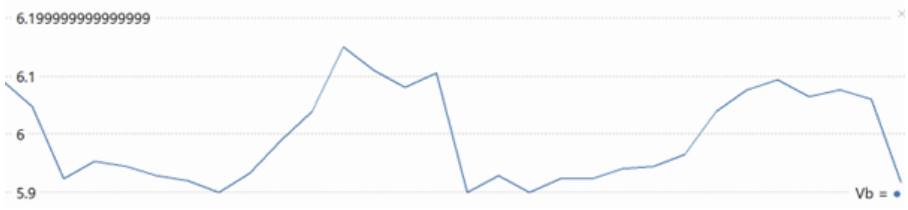


Figure 3.15: Initial Algorithm Voltage Response

The data in Figure 3.15 exhibits a voltage response which is constantly fluctuates to maintain the main bus voltage. This would place higher strain and requires importing $3.563J$ of energy to maintain constant voltage.

An alternate implementation of a PID controller was designed with K_p and K_i of 150 and 10 respectively, producing an output voltage response in Figure 3.16. The response is much more stable than the previous algorithm and only imports $2.447J$ to maintain a constant output voltage which is a 31.5% saving over previous implementation. The PID is centred higher than $6V$ to account for energy losses.

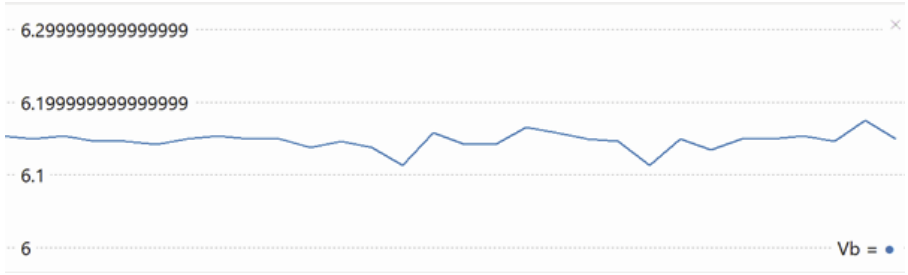


Figure 3.16: PID Voltage Response

3.1.7 Storage System and Control

To design the storage system, it was essential to characterise the relationship between the duty cycle, voltage and energy. This characterisation was performed by varying the duty cycle and measuring the corresponding voltage and energy levels. Two different approaches, energy control and power control were explored, to determine the most effective method for our system.

Characterisation and Equation Derivation

This characterisation was performed by varying the duty cycle and measuring the corresponding voltage and energy levels. The results are visualised in the following plots:

1. **Voltage vs Duty Cycle**

   From the characterisation, the equation that relates the voltage (V) to the duty cycle (D) was derived:

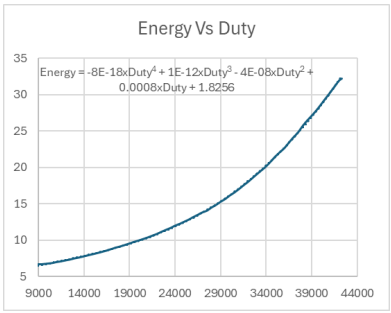
$$V = 6 \cdot 10^{-9} \cdot D^2 - 3 \cdot 10^{-5} \cdot D + 7.1367 \tag{3.9}$$

2. **Energy vs Duty Cycle**

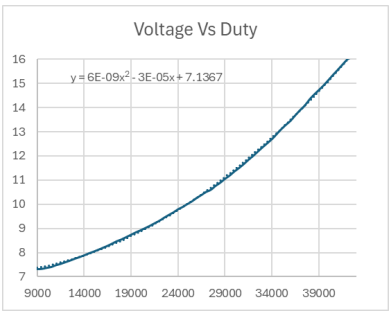
   Similarly, the equation relating energy (E) to the duty cycle (D) was found to be:

$$E = -8 \cdot 10^{-18} \cdot D^4 + 1 \cdot 10^{-12} \cdot D^3 - 4 \cdot 10^{-8} \cdot D^2 + 0.0008 \cdot D + 1.8256 \tag{3.10}$$

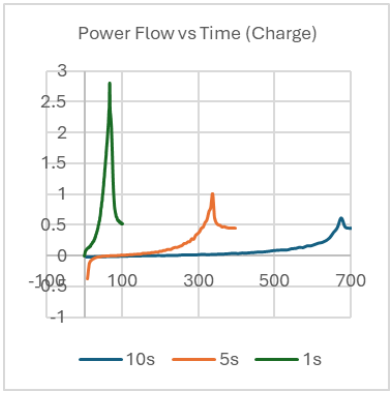
These equations were essential for initialising the system and adjusting the duty cycle to prevent current spikes and ensure smooth operation.



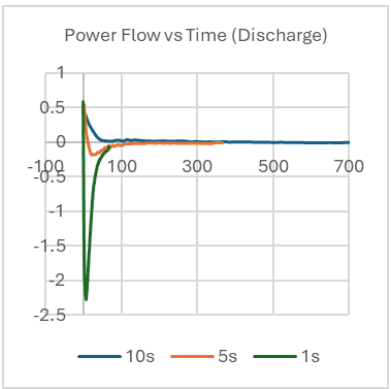
(a) Energy Vs Duty



(b) Voltage Vs Duty



(a) Charging Behaviour (500 steps)



(b) Discharging Behaviour (500 steps)

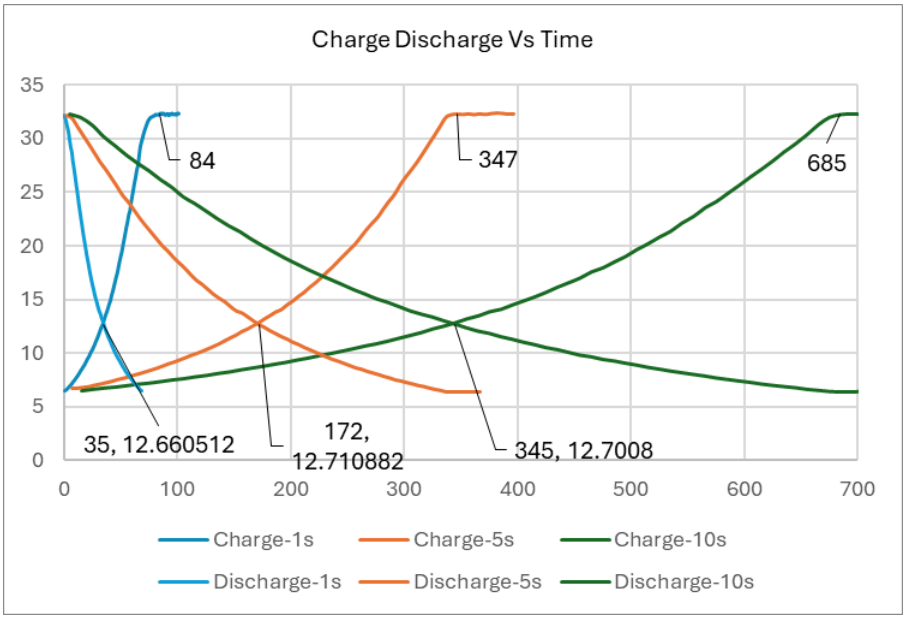


Figure 3.19: Enter Caption

Energy Control Approach

The energy control approach aimed to maintain a desired energy level in the storage system based on the server that sent energy transfer needed per tick. This method considered of various factors such as:

- **Leakage Current:** Supercapacitors inherently have leakage currents, which affect the stored energy over time. To account for this, a model that estimated the energy loss due to leakage and adjusted the control parameters accordingly was Incorporated.
- **Temperature Variations:** Temperature changes can influence the performance and capacity of supercapacitors. Due to the difficulty of tracking this, energy control was becoming unfeasible.
- **System Inefficiencies:** Inefficiencies within the system, such as resistance losses, can lead to discrepancies between the expected and actual voltage for a given duty cycle. Extensive testing to characterise these inefficiencies and integrate correction factors into our control algorithm was conducted but failed due to its non-linear nature of.

The energy control code involved complex calculations and adaptive adjustments to ensure the stored energy was accurately maintained at the desired level. This included real-time monitoring and PID control adjustments based on the measured energy levels and the system's dynamic characteristics.

The Implementation Details are as follows:

1. **Initial Setup:**
   - We used an ADC (Analog-to-Digital Converter) to measure the voltage across the supercapacitor (V_a)
   - An INA219 current sensor was utilised to measure the current flow (iL), which was essential for calculating the power and energy stored
   - The PWM (Pulse Width Modulation) signal was generated to control the duty cycle, influencing the charging and discharging rates of the supercapacitor
2. **PID Control:**
   - PID gains were dynamically adjusted based on the energy levels to ensure stability and responsiveness
   - The integral and derivative terms were tuned to minimise overshoot and oscillations, ensuring smooth energy control

3. **Stability Checks:**
    - A stability check mechanism was implemented to ensure the voltage levels were stable before making further adjustments. This involved averaging multiple readings and ensuring the variations were within a predefined threshold
4. **Characterisation Process:**
    - We characterised the supercapacitor's performance by conducting extensive tests under various conditions. This included measuring the voltage, current, and temperature to comprehensively profile the supercapacitor's behaviour
    - The data obtained from characterisation was used to fine-tune the control algorithms, ensuring they were tailored to the specific characteristics of the supercapacitor used in our system

Challenges with Energy Control

While the energy control method provided precise control over the energy stored, it presented several challenges:

- **Complexity:** Considering multiple variables (leakage current, temperature effects) added complexity to the control algorithm
- **Stability:** Ensuring the system's stability with varying environmental conditions required sophisticated tuning and constant monitoring. Maintaining constant energy flow was nearly impossible
- **Implementation Difficulty:** The intricate nature of energy control made it more difficult to implement and maintain

Power Control Approach

To address these challenges, a power control approach was devised. Power control focuses on maintaining a desired power output, simplifying the control process. By regulating the power directly, we could achieve efficient and stable operation without requiring extensive adjustments based on multiple variables. The key advantages of power control include:

- **Simplicity:** Power control is inherently simpler as it directly adjusts the power output via the duty cycle, reducing the need to account for various factors affecting energy storage
- **Ease of Implementation:** The control algorithm for power control is more straightforward, making it easier to implement and maintain
- **Efficiency:** By focusing on power output, we can quickly respond to changes in the desired power level, ensuring smooth operation and minimising the risk of system instability

Implementation of Power Control

The power control algorithm utilises a PID controller to adjust the duty cycle of the PWM signal, thereby controlling the power output. The process involves:

- **Reading Sensor Data:** Continuously measure the voltage and current to calculate the power output
- **PID Control:** Adjusting the duty cycle based on the desired and actual power output difference
- **Dynamic Adjustment:** Adapting the PID gains based on the power range to ensure optimal performance across different operating conditions

The steps for the PID are as follows:

1. **Initial Setup:**
   - Proper initialisation of the duty cycle was critical. An algorithm estimating the initial duty cycle based on the measured voltage was included to provide a stable starting point. This involved
     (a) **Voltage Measurement:** Taking an initial reading of the supercapacitor voltage
     (b) **Duty Cycle Estimation:** Using a formula to estimate the appropriate duty cycle for the measured voltage, ensuring the system starts within a stable operating range
2. **PID Gains Adjustment:**
   - The PID gains were tailored to different power ranges to ensure optimal control performance. This dynamic adjustment helped maintain precise control over various power outputs
3. **State of Charge (SoC) Calculation:**
   - We introduced a mechanism to calculate the supercapacitor's State of Charge (SoC), assisting to make informed decisions about charging and discharging cycles
   - SoC was used to prevent overcharging and deep discharging, thereby protecting the supercapacitor and enhancing its lifespan
4. **Addressing Power Leakage:**
   - To solve the power leakage problem, leakage current compensation was included in the control algorithm. The duty cycle could be varied to compensate for any unexpected losses by continuous monitoring of current and accounting for expected leakage, ensuring consistent power output
   - The characterisation data helped accurately model the leakage behaviour under different conditions, allowing control algorithm to make precise adjustments
5. **User Input for Desired Power:**
   - The system allowed for real-time user input to set the desired power output, making it flexible to varying requirements
   - Input validation ensured the desired power was within safe and feasible limits

Advantages of Power Control

The advantages of power control are:

- **Simplified Control Logic:** The power control approach reduced the complexity of the control logic, making it easier to implement
- **Improved Stability:** By focusing on power output, the system could quickly stabilise and maintain the desired power level without recalibrating
- **User Flexibility:** The ability to dynamically set and adjust the desired power output provided greater flexibility for different use cases

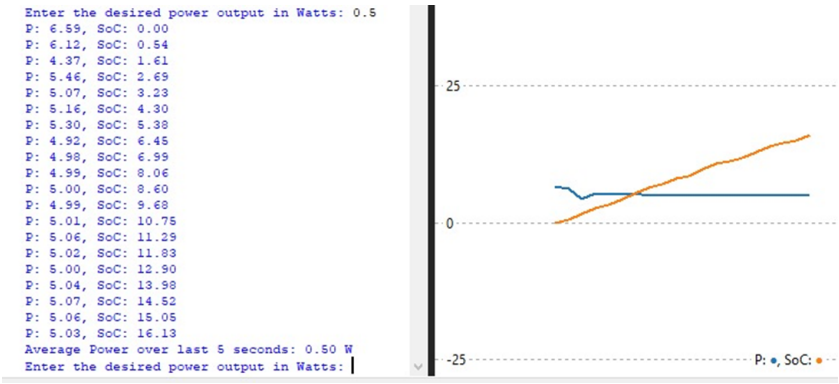


Figure 3.20: Constant Power PID
(scaled by x10) and State of Charge of storage

Leakage Control and SoC Limitation

Certain procedures were introduced to manage power leakage effectively and ensure stable operation of the storage system:

- **Leakage Control:** The desired power flow was cappedto ±3, based on testing that identified peak currents up to 0.35A, as specified in the datasheet. This control was implemented to mitigate power losses due to leakage under varying State of Charge (SoC) conditions
- **SoC Limitation:** The storage system's State of Charge (SoC) was limited to between 5% and 90%. This restriction provided headroom and stability, particularly at high SoC levels where performance issues were observed
- **Implementation Detail:** To address power leakage directly, dynamic control strategy in our code was implemented

```
if P_desired == 0:
    if 5 <= soc < 50:
        P_desired = 0.05  # Leakage control for SoC between 5% and 50%
    elif 50 <= soc < 70:
        P_desired = 0.06  # Leakage control for SoC between 50% and 70%
    elif 70 <= soc < 80:
        P_desired = 0.065  # Leakage control for SoC between 50% and 70%
    elif soc >= 80:
        P_desired = 0.07  # Leakage control for SoC above 80%

elif P_desired > 0:
    if soc >= 90:
        P_desired = 0.07  # Limit charging when SoC is 90% or higher

elif P_desired < 0:
    if soc <= 5:
        P_desired = 0.05  # Limit discharging when SoC is 5% or lower
```

Figure 3.21: Power leakage control to maintain SoC

This strategy adjusts the desired power output (P_desired) based on the current State of Charge (SoC), effectively compensating for power losses due to leakage and ensuring consistent operation of the storage system.

Conclusion

Overall, the transition from energy control to power control was a strategic decision that improved our storage system's performance and manageability. This approach streamlined the implementation and ensured reliable and efficient operation in various conditions.

3.1.8 Server-Hardware Communication

To facilitate seamless communication between our server and hardware components, a robust MQTT-based integration was introduced. More detail about this is provided in the communication section of the software.

MQTT Client Initialisation and Configuration

To establish communication with our MQTT broker, a Python module named `mqtt_client.py` was developed. This module includes the `MClient` class, which serves as our MQTT client interface. Here are the steps followed:

1. **Initialization:** Upon instantiation, the `MClient` class connects the Pico W to the internet, checks that the required libraries are installed (installs them if not) and connects to the MQTT broker. The "umqtt.robust" library is used to ensure the Pico W tries to reconnect to the MQTT broker if it disconnects **micropythonumqtt**.
2. **Subscription:** Each Pico connects to the correct topic, which depends on the hardware subsystem. For example, the PV array subscribes to "pico/pv-array". More detail about the topics used is provided in the communication part of the Software section.

Message Handling and Data Exchange

Message Reception: Using a callback function (`on_mqtt_msg`), the client listens for incoming messages. These messages are parsed as JSON objects to extract target devices and their corresponding payload data. A set of methods, with names prefixed with `get`, such as `get_desired_power`, returns the last power value received from the broker.

Data Publication: The client can publish data to the server using methods like `send_pv_power(power)` and `send_external_grid(import_p, export_p)`. These methods encode data into JSON format and publish it to the "server" topic.

3.2 Software

3.2.1 Requirements

- Design an algorithm to determine the energy to release / store, import / export, and allocate to each deferrable demand per tick
- Optimise the algorithm to minimise costs from importing / exporting energy
- Fetch requirement data from an external web server and pass it to the algorithm
- Communicate algorithm decisions to the appropriate hardware subsystems
- Develop a UI to display the current and historical state of the systems (web server requirements, algorithm actions, and hardware results) and enable manual control of the hardware subsystems

3.2.2 Overview

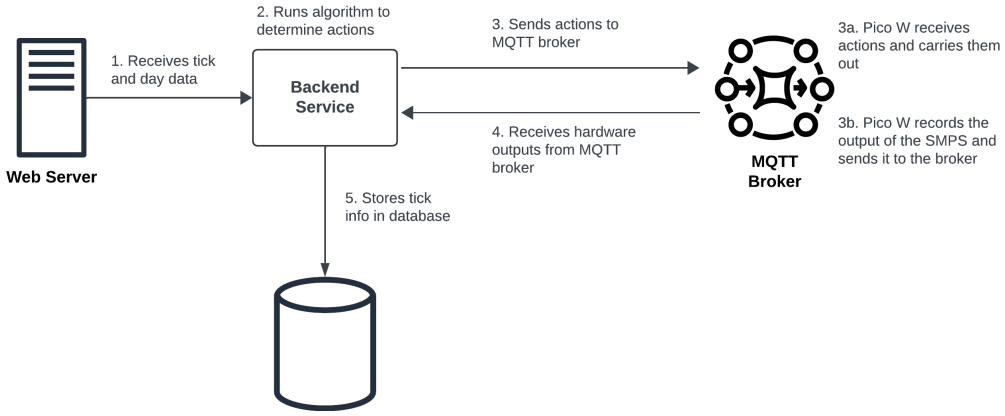


Figure 3.22: High-Level Overview of the Software

The software component comprises the following subsystems:

1. Backend service and database
2. Communication between backend service and hardware subsystems via a message broker
3. Dashboard (UI)
4. Optimisation algorithm

The high-level data flow per tick is shown in Figure 3.22. The UI is excluded from this figure to avoid complexity but it pulls data from both the backend service and MQTT broker.

3.2.3 Backend Service

The backend service serves as the central hub connecting various modules within the software ecosystem. It is comprised of three parts:

1. A main function executed every tick
2. A function monitoring power levels from the hardware components
3. A REST API for the UI to request data

The REST API is implemented using Next.JS server actions for simplicity. Appendix B details the functions built for each server action.

Main Function

The main function runs a loop, executing once per tick, and handles the following tasks:

1. Making HTTP requests to the external web server to fetch requirements data
2. Running the optimisation algorithm to generate hardware actions and sending the data to the message broker
3. Saving tick information to the database

A primary requirement for the main function is to minimise the time elapsed between the tick change on the web server and the transmission of actions to the Pico Ws, ensuring the Pico Ws have sufficient time to execute the actions.

A. Fetching Requirements from the Web Server

The first step in the loop is to fetch the following data points from the external web server:

- Sun data
- Buy and sell prices
- Instantaneous demand
- Deferrable demands

The main function achieves this by making an HTTP GET request to the specified endpoint. The requirements for this process are:

1. Minimise the request latency (RTT)
2. Minimise the delay between the tick change on the web server and the initiation of the fetch request by the backend service – this ensures locally accurate price information

The first requirement was met by sending requests to the `/sun`, `/price`, and `/demand` endpoints in parallel, and to the `/deferrables` endpoint every 60 ticks. For the second requirement, two implementations were considered, with the second one being used in the actual system:

Implementation 1: A naive approach is to constantly poll the web server as soon as a response is received for the previous request. Although this method performs well for the requirement, it strains memory resources due to the high frequency of requests.

Implementation 2: The chosen approach involves synchronising the backend service with the web server. It involves initially determining the start of a tick through polling. Once the tick changes, the main loop begins, and the backend service resumes polling 4.5 seconds after the last tick change.

Testing

To test the request latency, the RTT of the parallel requests was measured over 120 trials. The average time recorded was 330ms.

To test the delay between the tick changing on the web server and initiation of the fetch request on the backend service, the following was performed:

1. Assume implementation 1 has 0 delay
2. Run implementation 1 and 2 in parallel and record the timestamps for each tick change
3. Compare the difference between the timestamps of implementation 1 and implementation 2's

Across 120 trials, the average difference was 42ms. This was assumed to be the average delay between the tick changing on the web server and initiation of the fetch request on implementation 2.

B. Running the algorithm and sending actions to the message broker

The next step in the loop is to run the algorithm to generate hardware actions and send these actions to the message broker. Both steps are discussed in detail in dedicated sections later. At a high level, the following data processes occur in this step:

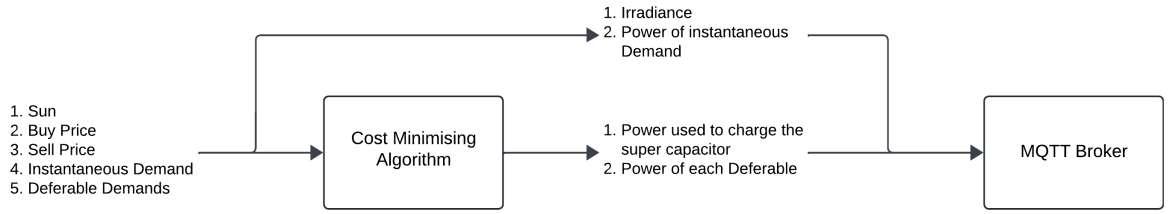


Figure 3.23: High-level overview of algorithm and broker

As shown in Figure 3.23, the algorithm determines the energy to release/store, import/export, and allocate to each deferrable demand for the given tick. This information is communicated to the Pico Ws by publishing to the appropriate topics in the MQTT broker. The values are sent in terms of power, with each energy value divided by the tick length, set to 5 seconds.

Testing

The algorithm runtime was tested over 1000 ticks, averaging 0.2ms per tick. Considering the delay from web server tick change to fetch request initiation, the RTT for parallel requests, and the algorithm's execution time, the total time taken is 372.2ms. This leaves 4.63s for the Pico Ws to receive and execute the action data, meeting the target mentioned earlier.

C. Saving tick data to the database

For each tick, there are three sets of data being saved:

1. Tick requirements fetched from the web server
2. Actions generated by the algorithm
3. Result of hardware operations

Additionally, the deferrable demands for each day are saved.

Monitoring hardware components and computing results per tick

The output of each hardware component was monitored and averaged over the tick. This process aimed to compare the actual output with the action communicated by the algorithm to measure the difference.

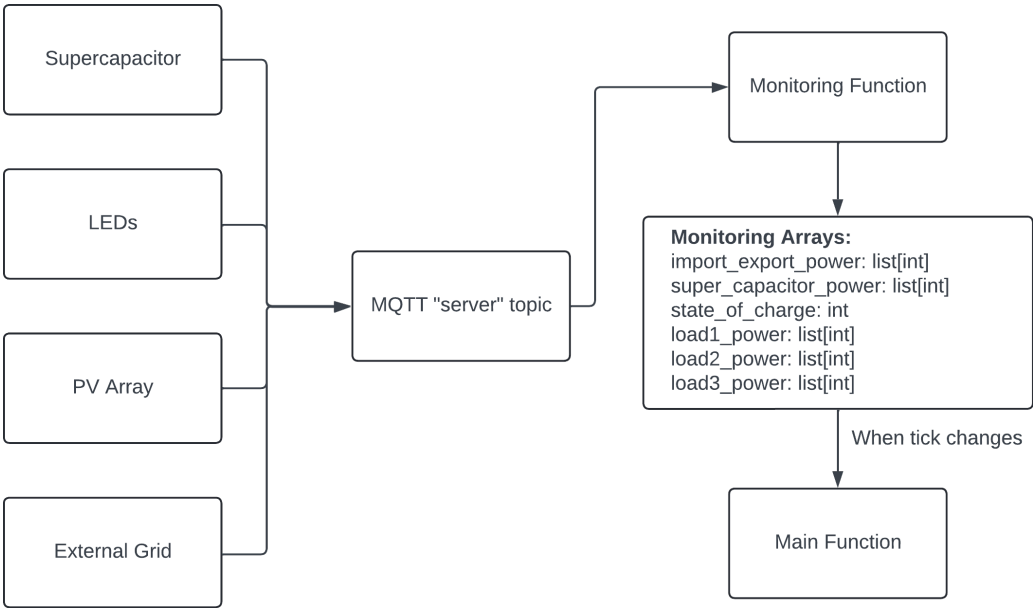


Figure 3.24: Hardware output monitoring flow

Note: Monitoring arrays are stored as global variables in the backend service

The general flow for monitoring and saving the output of the hardware components is shown in Figure 3.24. The following data was monitored:

1. Supercapacitor power charged (watts)
2. Supercapacitor state of charge (percentage)
3. Power supplied to each LED (watts)
4. Power output of SMPS simulating PV array (watts)
5. The import and export power of the external grid

The Pico W on each hardware component periodically publishes its output values to the MQTT "server" topic. The monitoring function subscribes to this topic and appends the received values to the relevant arrays.

In the main function, whenever the tick changes, signalling the end of the previous tick, the average of each array is calculated and stored in the database –these represent the results of the hardware operations for that tick. At this time, the arrays are also cleared. For the state of charge, the value is monitored as a single integer rather than as an array.

To compute the cost, the formula $(import\_power \times sell\_price) + (export\_power \times buy\_price)$ is used.

Database Implementation

The database used in this project is MongoDB. It was chosen for its familiarity, ease of implementation due to the flexible schema, and general reliability. MongoDB Atlas, a managed service, was used. There are two collections for data storage: `days` and `ticks`. The data schema for each collection is shown below:

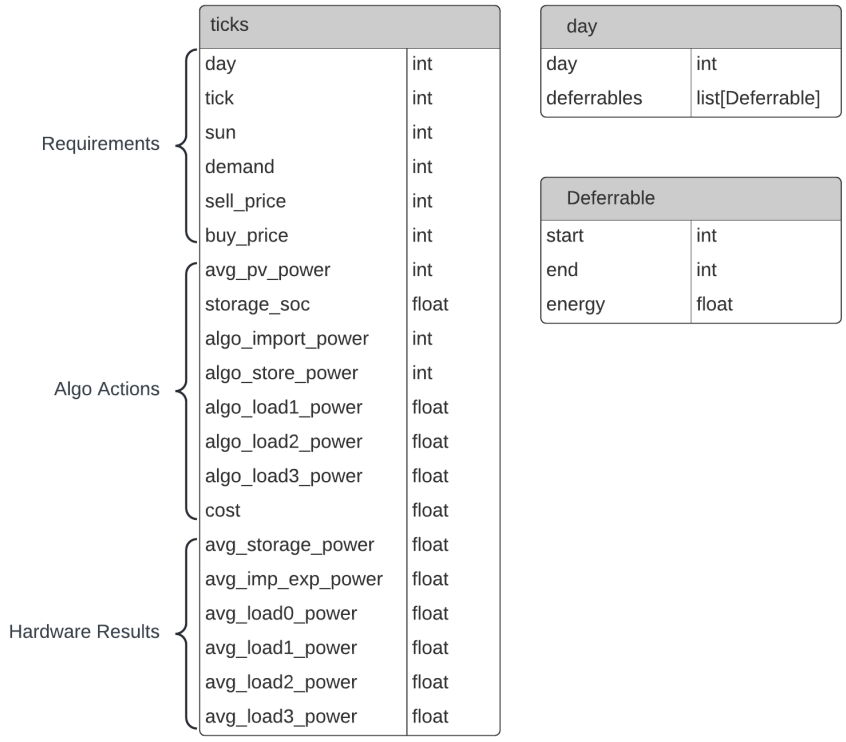


Figure 3.25: Database schema

Note: Deferrable is not a collection, but a data model for the Deferrable object stored within the day collection

3.2.4 Communication between backend service and hardware

When choosing a communication protocol, the following two requirements were set:

1. It should be lightweight and use minimal memory, as the Pico W only has 2MB of on-board flash memory **raspberrypi2024**

2. It should support duplex communication, allowing the backend service to send algorithm decisions and the Pico Ws to send hardware outputs

The following protocols were considered:

1. HTTP

2. Raw TCP

3. MQTT

HTTP was deemed too memory-intensive **raspberrypiforum2024**, and raw TCP was unnecessarily complex compared to MQTT. Therefore, the MQTT protocol was selected. MQTT is a lightweight, publish, subscribe based messaging protocol, making it ideal for these requirements.

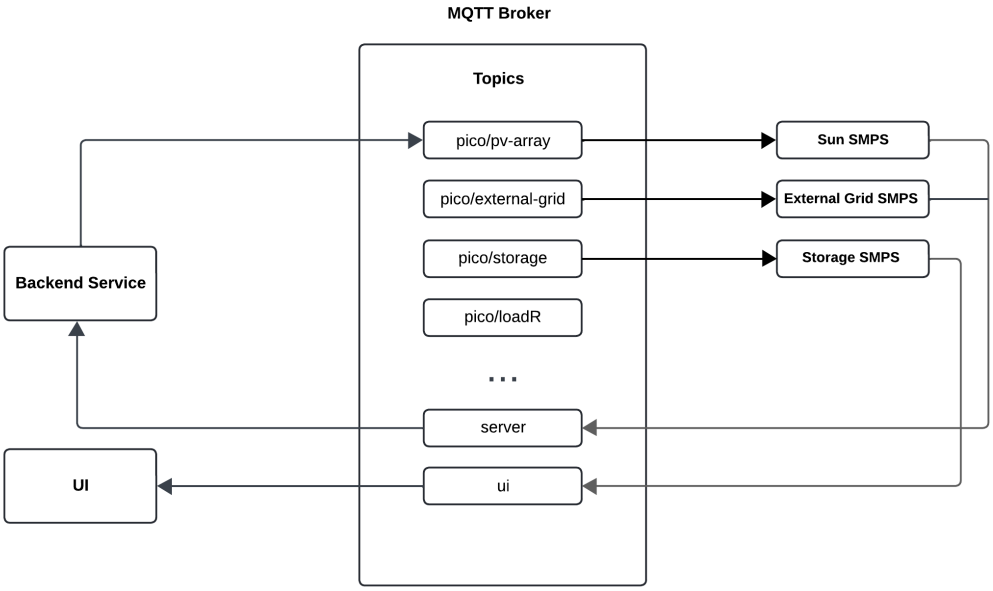


Figure 3.26: MQTT Communication

The data flow for the MQTT communication is illustrated in Figure 3.26. Each SMPS module subscribes to its own topic, where it receives algorithm actions and publishes hardware outputs to the "server" topic. The backend service and web UI subscribe to the "server" and "ui" topics respectively. The transmitted data follows the JSON schema: {"target": <string>, "payload": <any>}, where 'target' refers to the Pico W that the data is being sent to or from, and 'payload' refers to the actual data value being sent. Appendix 4 details all the messages sent between each device and how they should be formatted. The MQTT broker is hosted as an Eclipse Mosquitto program running on an EC2 t2.micro instance and the user interface and backend service is run locally on our computer and made publically available using ngrok, this is done due to the hardware limitations of the t2.mico.

3.2.5 Dashboard / UI

The requirements of the dashboard are as follows:

- Display a live view of the power consumed by each hardware subsystem
- Display historical information about the power consumed by each hardware subsystem
- Display information to verify the system is functioning as instructed by the algorithm
- Have manual control of each hardware subsystem

Based on the requirements, the UI can be broken down into 3 parts:

- Home: Provide only the most recent information about the system
- Historical Information: Display past data of the system
- Control: Provide a live view of the systems and manual control of each component of the system

To verify each hardware component's functionality, the algorithm's power instructions are plotted alongside the average power output for each component. Additionally, plotting the buy and sell prices helps to observe the algorithm's decisions. Each graph is limited to two units on the left and right Y axes and a maximum of four plots to maintain clarity.

The data points available to plot are:

1. Sell Price (Sell Price)
2. Buy Price (Buy Price)
3. Cost for the Tick (Tick)
4. Sun Irradiance from web server (Sun)
5. Average power of PV SMPS (PV Power)
6. Algorithm generated import/export power (Import/Export)
7. Average import/export power (EG Power)
8. Algorithm generated release/store power (Release/Store)
9. Average release/store power (SC Power)
10. State of Charge (SOC)
11. Instantaneous demand from web server (Demand)
12. Algorithm generated deferrable allocations (DD X)
13. Average power supplied to each load (Load X Power)

The terms in brackets will be used to refer to each data point in subsequent sections.

Plot	Data Point 1	Data Point 2	Data Point 3	Data Point 4
1	Tick	Demand	-	-
2	Sun	PV Power	-	-
3	Import / Export	EG Power	Buy Price	Sell Price
4	SOC	SC Power	Release/Store	-
5	Release / Store	SC Power	Buy Price	Sell Price
6	Demand	Load 0	-	-
7	DD0	Load 1	Buy Price	Sell Price
8	DD1	Load 2	Buy Price	Sell Price
9	DD2	Load 3	Buy Price	Sell Price

Table 3.3: Data Points for Each Plot

Home Page

The home page displays the most recent hardware behavior, enabling verification through the plots listed in Table 3.3. This is shown in Figure 3.27. It updates by retrieving the current day and tick from the external web server, then fetching data for the previous tick from the database. Subsequently, it updates every five seconds with data for the next tick.

Historical Page

The historical page shows the same plots as the home page, except with data taken from the database. Additionally, it shows a table (Figure 3.28) detailing the algorithm's energy allocation for each deferrable load, the actual energy supplied, the energy required, and the remaining ticks to fulfill the requirement. This information helps verify whether each deferrable load is satisfied.

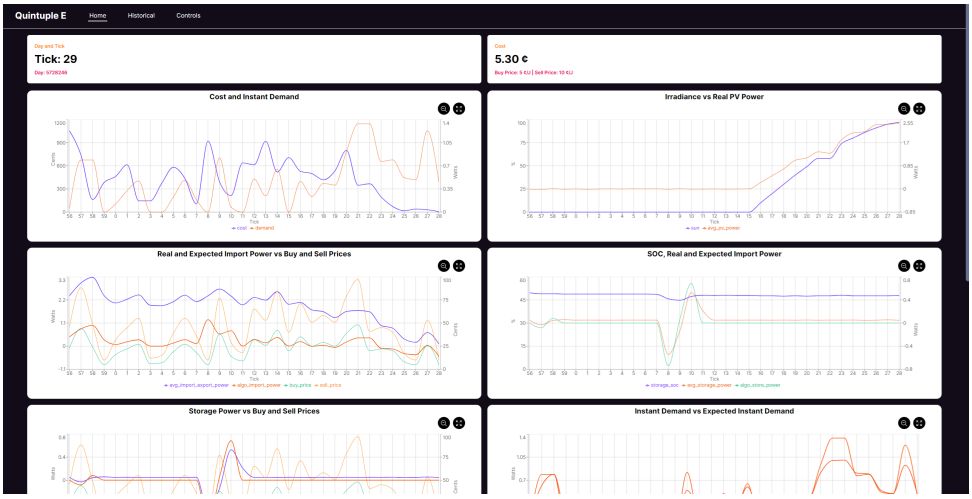
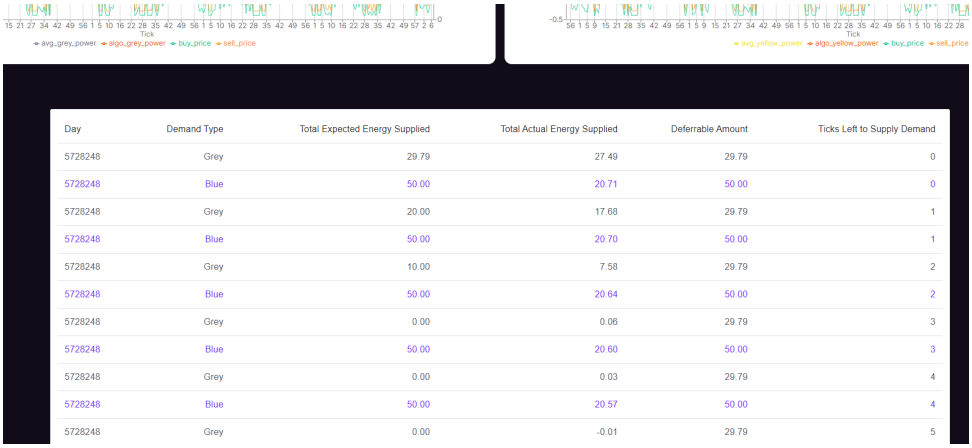


Figure 3.27: Example of plots shown in the UI



Day	Demand Type	Total Expected Energy Supplied	Total Actual Energy Supplied	Deferrable Amount	Ticks Left to Supply Demand
5728248	Grey	29.79	27.49	29.79	0
5728248	Blue	50.00	20.71	50.00	0
5728248	Grey	20.00	17.68	29.79	1
5728248	Blue	50.00	20.70	50.00	1
5728248	Grey	10.00	7.58	29.79	2
5728248	Blue	50.00	20.64	50.00	2
5728248	Grey	0.00	0.06	29.79	3
5728248	Blue	50.00	20.60	50.00	3
5728248	Grey	0.00	0.03	29.79	4
5728248	Blue	50.00	20.57	50.00	4
5728248	Grey	0.00	-0.01	29.79	5

Figure 3.28: Example of algorithm deferrable allocations display - Historical Page

Controls Page

The controls page (Figure 3.29) provides a real-time view of the hardware status. It shows connected devices, the power output of each component, and allows control of each component's output. Devices are marked as disconnected if no message is received within five seconds, given the 1Hz message rate from each Pico W. The "server" topic is subscribed to for receiving hardware power outputs – this is done instead of sending the same data to the "ui" topic to prevent data redundancy. To manually control the inputs, the dashboard ensures the backend service isn't sending data to the Pico W's by syncing a boolean "override status" between the two devices using the MQTT protocol. On initialization, the dashboard requests the current override status, and when the user changes it, the dashboard sends the new value to the backend service.

3.2.6 Optimisation Algorithm

Figure 3.30 shows the inputs and outputs of the cost minimizing algorithm. It is responsible for deciding each of the actions taken by the hardware components. Specifically:

1. Release / Store amount is communicated to the supercapacitor

2. Energy allocation for each deferrable, represented as a size 3 array, is communicated to loads 1, 2 and 3 respectively.

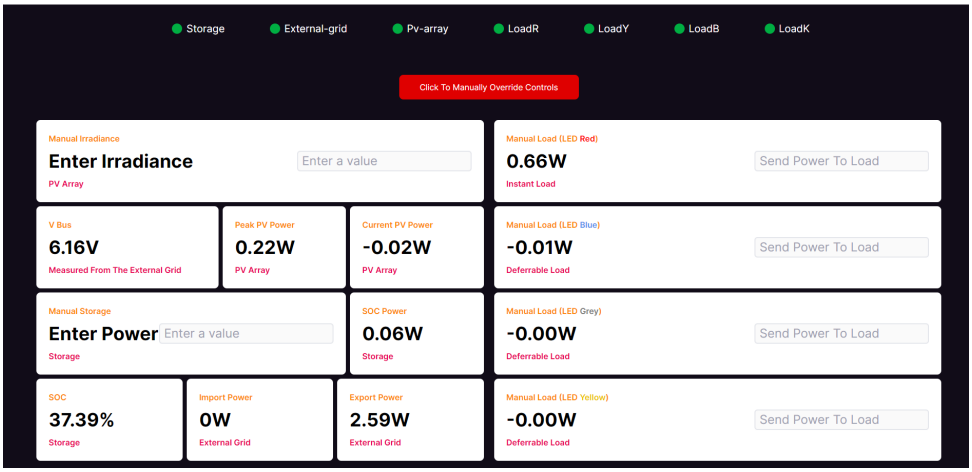


Figure 3.29: Controls Page

Figure 3.30: Top level overview of optimisation algorithm

Note: The import/export energy is not communicated because the external grid automatically balances the energy once all other hardware operations are complete, thereby fulfilling the import/export action. Additionally, for all algorithms discussed below, given the sun energy, instantaneous demand, deferrable allocations, and the import/export amount, any excess energy will be stored in the supercapacitor. When the supercapacitor is full, the remaining energy will be exported. Consequently, only the import/export amount and deferrable allocations can be adjusted to optimise cost, while the release/store amount is implicitly determined by the import/export balance.

The algorithm runs once per tick and its requirements are as follows:

- Energy must be conserved at each tick
- Instantaneous demand must be satisfied at each tick
- Deferrable demands must be satisfied by the end tick specified
- Cost over time should be minimised

Additionally, the following hardware constraints were respected:

- Supercapacitor energy capacity of 25J
- Maximum of 50J to be imported or exported per tick

Implementation

Three different methods to implement the algorithm were explored, each with increasing complexity and aimed at improving upon the previous one. To test the algorithms, they were simulated across 1000 days, and the following aspects were analysed:

1. Plot of the exponential moving average (EMA) of cost using $N = 100$, as shown in Equation 3.11.
2. The average cost per day.

$$\text{EMA}_i = \alpha \cdot \text{data}_i + (1 - \alpha) \cdot \text{EMA}_{i-1} \quad \text{for } i = 1, 2, \ldots, \text{len(data)} - 1, \text{ where } \alpha = \frac{2}{N+1} \tag{3.11}$$

An EMA is used because the variation in cost per day is quite large, and using an EMA helps us better compare and understand trends. N in the equation indicates how past values are taken into account when computing the EMA.

Approach 1: Naive Algorithm

The first and simplest algorithm implemented is the naive algorithm, which follows this logic:

1. Any deferrable demands are satisfied as soon as possible.

2. If there is not enough energy provided by the sun and supercapacitor, import energy up to the maximum amount.

3. If there is excess energy, store as much as possible in the supercapacitor, and export the remaining energy to the external grid.

Two additional variants of this algorithm were tested to see if the results could be improved:

- **Variant 1:** Export excess energy at the end of the tick instead of storing it in the supercapacitor.

- **Variant 2:** Satisfy demands as late as possible instead of as early as possible.

For Variant 2, due to the limit on the amount of energy available for import, it was implemented as follows:

1. Initialise a maximum energy value that can be allocated per day, calculated as the maximum import energy minus the maximum instantaneous demand, approximately 10J.

2. Based on energy requirements and deadlines, count backward from the deadline and assume the maximum energy value (from step 1) is allocated each day until the deadline.

3. Allocate the maximum energy value if the current tick is within the required range (from step 2).

The three variants were tested and compared, as shown in Figure 3.31. As can be seen, the algorithm that uses the supercapacitor and satisfies deferrable demands as early as possible performs the best (Naive SC Start).

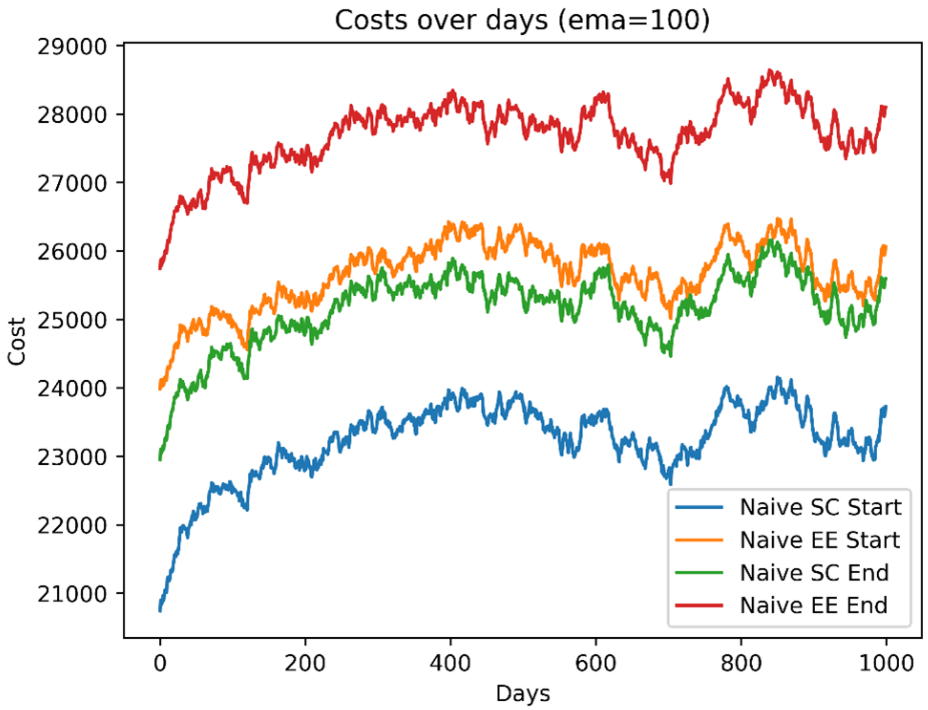


Figure 3.31: Comparison between Naive Algorithm and variants

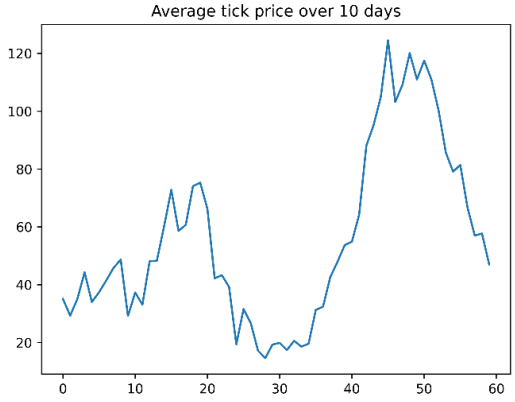
Something interesting to note is that when the max import energy is high enough such that all deferrable allocations can be satisfied at one go, the algorithm which allocates demands at the end performs better. Additionally, the average cost for the Naive SC Start algorithm was used as a benchmark to try and improve upon in the next algorithm.

Approach 2: Price Modelling and Heuristics

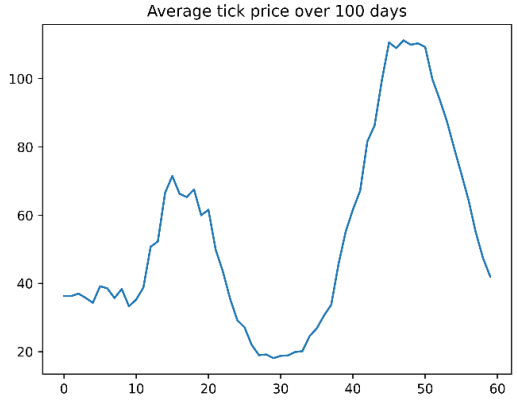
The next approach was to model the sell price per day and then use that to make heuristic decisions about deferrable allocations and import / export amounts.

Price Modelling

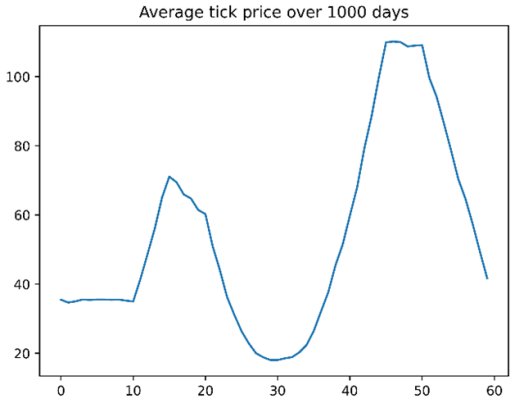
To model the price on each day, the average price across N number of days was taken for each tick in a day. The trials performed were for N = 10, 100, 1000 and 10,000.



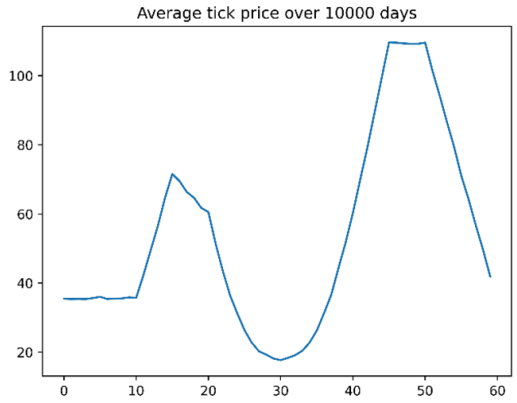
(a) Average tick price over 10 days



(b) Average tick price over 100 days



(c) Average tick price over 1000 days



(d) Average tick price over 10000 days

Figure 3.33: Comparison of average tick prices

As the number of samples increases, the tick price converges to a particular trend. This indicates that there is a general trend for the price, with some random noise added that has a mean of 0, according to the Law of Large Numbers.

Heuristics

Using this model, we developed the following heuristics for the algorithm:

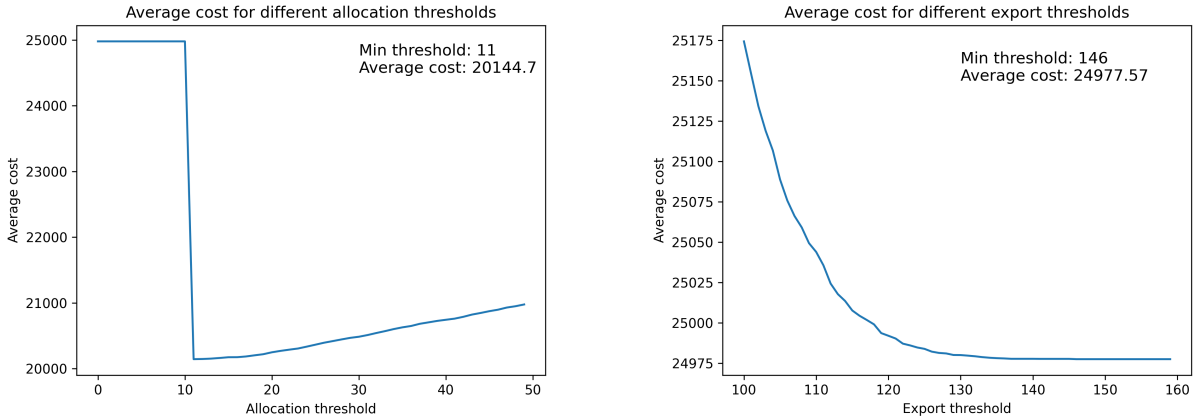
1. **Allocation Threshold:** Anytime the price is below this threshold, we allocate as much energy as possible to the deferrable demands. Additionally, to ensure that deferrable demands are satisfied by their deadlines, we allow the Variant 2 algorithm discussed above for the naive algorithm to take over once the current tick is within the required range.
2. **Export Threshold:** Anytime the price is above this threshold, we will export as much energy as possible, essentially draining the supercapacitor of stored energy.

Testing

Firstly, for the allocation threshold, we tested values between 0 and 50 inclusive. For each threshold, the average cost across 1000 days was calculated and compared. As seen in Figure 3.34a, the allocation threshold with the minimum average cost was 11. Secondly, a similar test was performed for the export threshold, with values

between 100 and 160 inclusive. As seen from the results in Figure 3.34b, the best value for the export threshold is 160.

Note: The export threshold heuristic was turned off during the allocation threshold test, and vice versa. Another method to find the optimal threshold would be to use a grid search approach to find the best combination of allocation and export thresholds. However, based on the results, it is assumed that the two thresholds are independent of each other.



(a) Allocation Threshold **(b)** Export Threshold

Figure 3.34: Results for testing threshold values

Results

As shown in Figure 3.35, the heuristics-based algorithm outperforms the naive algorithm, with an average cost of 20418.45 cents across 1000 trials, compared to 23408.47 cents for the naive algorithm. This demonstrates the effectiveness of the price modelling and thresholds.

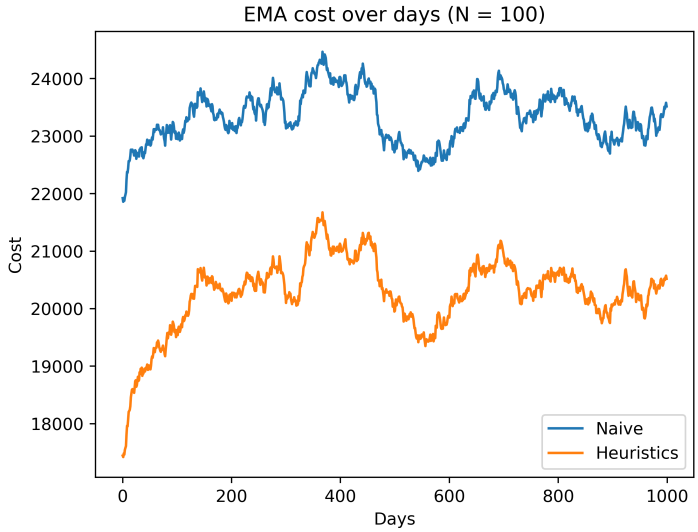


Figure 3.35: Comparison of heuristic-based algorithm and naive algorithm

Approach 3: Reinforcement Learning

The final algorithm implemented was a reinforcement learning (RL) algorithm, specifically the REINFORCE algorithm, which is a basic policy gradient RL method. The rationale behind using this approach was to enable a

neural network, referred to as the policy network, to dynamically learn optimal actions at each time step based on the current state (e.g., sun, import price, etc.). This aims to surpass the performance of static heuristics shown in the previous algorithm.

The setup for the RL algorithm is illustrated in Figure 3.36. At each time step, the policy network receives a state as input and generates an action as output. The state is represented as an (N,) tensor, with the meaning of each dimension detailed in Figure 3.37. The action is a scalar value that denotes the amount of energy to import or export at that time step. This action results in an observation, reflecting the outcomes of the hardware components executing the specified actions – such as the remaining energy needed to satisfy each deferrable demand and the new state of the supercapacitor.

Note: Technically, the action could have been a vector with additional dimensions representing the amount of energy to release or store and the energy allocation to each deferrable demand. However, two main reasons justified using a scalar action:

1. As mentioned above, the release/store amount is implicitly determined by the import/export amount.
2. Heuristic methods from our second algorithm proved more effective in deciding allocations for each deferrable demand. Testing showed that the policy network could not converge to a better solution for deferrable allocations than the heuristic approach used in the second algorithm.

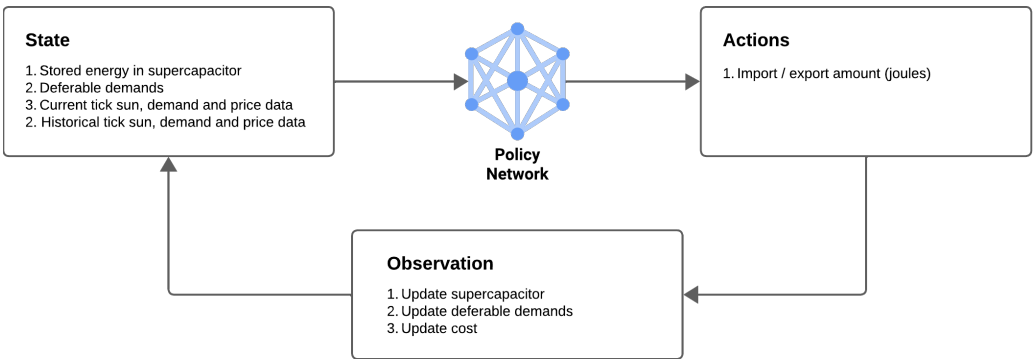


Figure 3.36: Policy Network Diagram

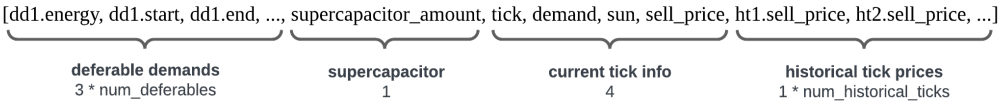


Figure 3.37: State Vector

Training the policy network

Mathematically, the goal of the policy network is to optimise the expected return (sum of rewards) of a trajectory — a set of state-action pairs. This is expressed through the objective function shown in Equation 3.12 **rlequations**. The gradient of the objective function, which is used to update the policy network parameters during backpropagation, is shown in Equation 3.13 **rlequations**. This can be interpreted as follows:

1. For each trajectory, calculate the log probability of each action taken and multiply it by the sum of rewards from that time step to the end of the trajectory.
2. Average the result of the above calculation across N trajectories.

$$J(\theta) = E_{\tau \sim p_\theta(\tau)} \left[\sum_t r(s_t, a_t) \right] \tag{3.12}$$

$$\nabla_\theta J(\theta) \approx \frac{1}{N} \sum_{i=1}^{N} \sum_{t=1}^{T} \nabla_\theta \log \pi_\theta(a_{i,t}|s_{i,t}) \left(\sum_{t'=t}^{T} r(s_{i,t'}, a_{i,t'}) \right) \tag{3.13}$$

> **Definition:** ***Trajectory (τ)***
> We define one day of simulation (60 ticks) as a trajectory or episode.

> **Definition:** ***Reward***
> We define the reward per tick as the negative of the expenditure due to the import/export amount.

Using these definitions, one epoch of the training process is as follows:

1. Simulate the policy across one day.
2. For each tick, calculate the return of that tick—the sum of rewards from that tick to the last tick of the day.
3. Calculate the gradient of that trajectory (day) using Equation X.
4. Repeat steps 1 to 3 N times using the same day and average the gradient across each episode sample.
5. Update the weights of the network through backpropagation.

This process is then performed across M number of days to train the policy network. For our final model, N (number of samples per trajectory) was set to 50, while the M (number of epochs / days) was set to 5000. As seen in Figure 3.38, which plots the EMA of the training cost for each epoch, the model learns to minimise the cost, reaching a minimum of 18191.93 cents at epoch 3478 (red point).

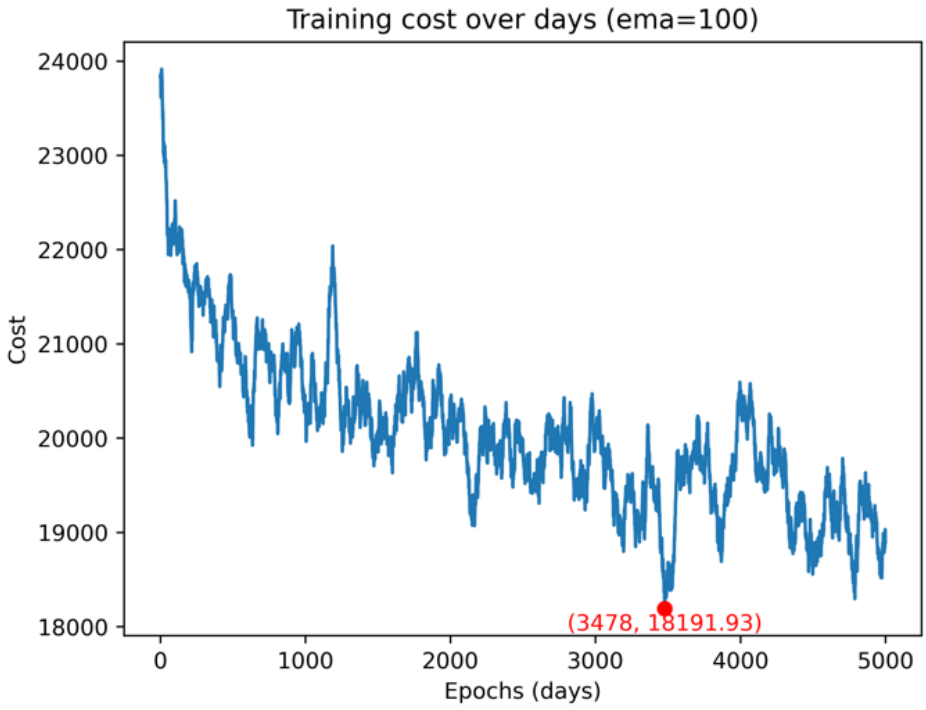


Figure 3.38: Training Cost Over Days

Hyperparameters

To ensure optimal learning and convergence rates for the policy network, the following hyperparameters were tuned:

1. **Import/Export Multiplier:** This value is multiplied by the action scalar (representing the energy to import or export for that tick) to keep the action values within a reasonable range. For instance, if the import/export amount is around 0–1J, small changes have less impact compared to a range of 10–15J. A 10x multiplier was thus used to maintain appropriate action magnitudes.

2. **Number of Trajectories per Epoch:** According to Equation 3.12, the objective function is the expected return for a trajectory. Sampling multiple trajectories and averaging their returns provides an unbiased estimate of this expectation. Empirical tests showed that using only one trajectory per epoch caused high oscillations in the training cost. Increasing the number of trajectories stabilised training and improved convergence rates. However, beyond 50 trajectories, the computation time increase outweighed the benefits.

3. **Number of Epochs:** The high variance in gradient meant that training convergence varied. Thus, an appropriate range for the number of epochs is typically between 1000 and 5000. The epoch with the lowest EMA cost was saved. For hyperparameter testing, 1000 to 3000 epochs were used to increase training speed. The final model, as shown in Figure 3.38, was trained for 5000 epochs.

4. **Discount Factor and Learning Rate:** The discount factor (γ) was set at 0.99 to emphasise future rewards. A learning rate of 1e-4 was used. These parameters were not extensively tested, as tunning them did not significantly affect convergence speed or model performance.

Import / Export Multiplier	1	5	8	10
Number of trajectories per episode	1	10	20	50
Number of epochs run	1000	2000	3000	5000
Discount Factor	0.99			
Learning Rate	1e-4			

Table 3.4: Hyperparameter values tested during training.

Note: A hyperparameter tuning algorithm like grid search could have been implemented to select the best values. However, one training run took up considerable time (~ 2 hours), and therefore performing grid search would probably take up to a day. Given that the requirements and constraints of the hardware were subject to changes even up to a day before the demo, it was not practical to constantly run a hyperparameter tuning algorithm.

Results

As shown in Figure 3.39, the RL algorithm performs the best out of the three. This shows that the learned decision making for import / export amount outperforms the heuristic decisions made in the second algorithm.

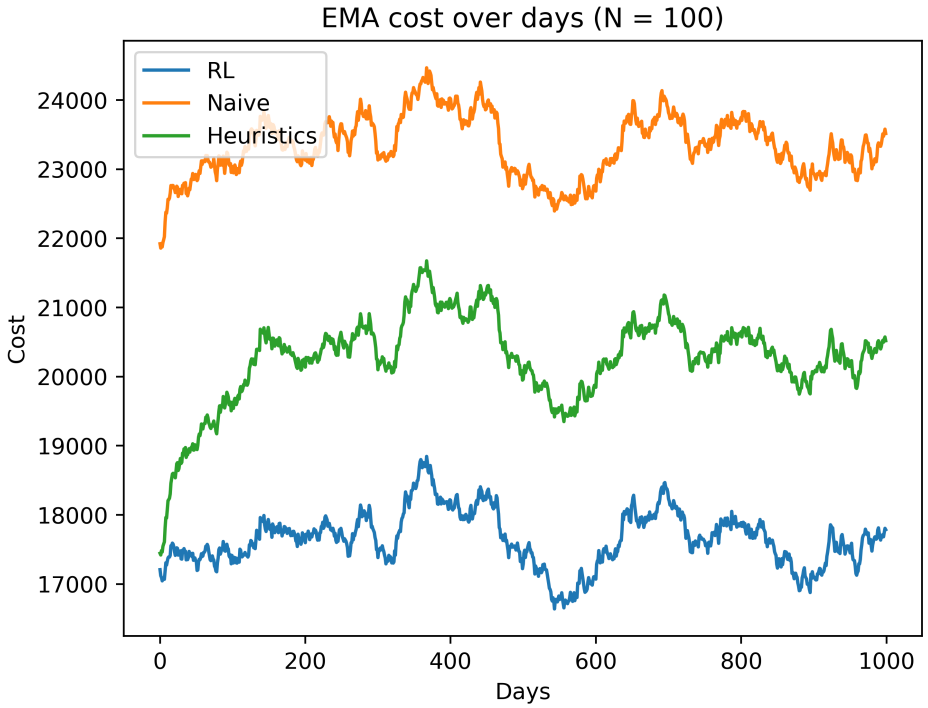


Figure 3.39: Comparison between RL and other algorithms

Algorithm	Average Cost over 1000 days (cents)
Naïve	17678.24
Heuristics	20418.45
Reinforcement Learning	23408.47

Table 3.5: Comparison of average cost over 1000 days for different algorithms.

Limitations

A major limitation of the RL model is its extremely high variance when estimating the gradient of returns. Additionally, there are many local optima, and there may not be a true global optimum. This makes it difficult for the model to converge to an optimal solution, as we encountered practically. This issue was evident when the model attempted to learn actions for allocating energy to deferrable demands—it did not outperform the heuristic decision using a price threshold. However, allowing the model to learn a single action (the amount of energy to import/export at each tick) proved much more effective.

To address the high gradient variance, several techniques could have been employed. One technique we implemented was using a baseline for the returns. A baseline shifts the evaluation of each action from "How well does it perform" to "How much better than the average does it perform" by subtracting a baseline value from each return. We tested two baselines: the mean of the rewards and the returns from the heuristic-based algorithm. The latter performed significantly better and resulted in higher model performance.

Other potential methods, not implemented in this study, include using an actor-critic algorithm instead of a policy gradient one, where the Value or Q function could be learned and used as the baseline.

3.2.7 Software Conclusion

In summary, the following systems have been successfully implemented, with relevant tests employed to ensure their effectiveness and compliance with requirements:

1. A backend service capable of:
    - Fetching requirements from the web server with minimal latency and synchronisation delay
    - Running an algorithm to determine hardware actions that minimise cost over time
    - Communicating algorithm actions to the hardware subsystems and monitoring output levels through a duplex communication protocol
    - Saving tick information to a database
2. A UI that:
    - Allows the user to override the algorithm and manually provide actions to the hardware
    - Fetches data from the backend service and message broker to display plots of relevant data points such as cost over time

Chapter 4

Testing and Results

This section details the testing and results from connecting the entire system together. This process was split into two different stages:

- Connecting and controlling the hardware subsystems through the dashboard
- Controlling the hardware subsystems with the optimisation algorithm and logging the data produced

4.1 Connecting the hardware and dashboard

This test aims to send information to each Pico W and display its real-time output power. The Pico W sends data to the server topic, validating its ability to transmit data. When the dashboard and hardware were started independently, some devices sent messages immediately, turning the connected indicator green, while others failed to connect. Figure 4.1 shows an example where data about power output and storage state of charge were being received, with values constantly changing, indicating new data was being sent.

Some devices did not connect because the external grid took too long to connect and maintain the bus voltage at 6V, leading to improper power supply and connection failures. To mitigate this, a five-second timeout was added to all devices except the external grid, ensuring that it connects first and establishes the 6V bus voltage. This allowed the other SMPSs to function correctly, as they depend on the 6V bus voltage V_{bus}.

When the external grid fails, no devices can connect to the server due to insufficient power, as the external grid is the only one directly powered by the PSU. This approach was necessary to ensure consistent operation.

After resolving this issue, it was possible to monitor the power output of all devices and the state of charge of

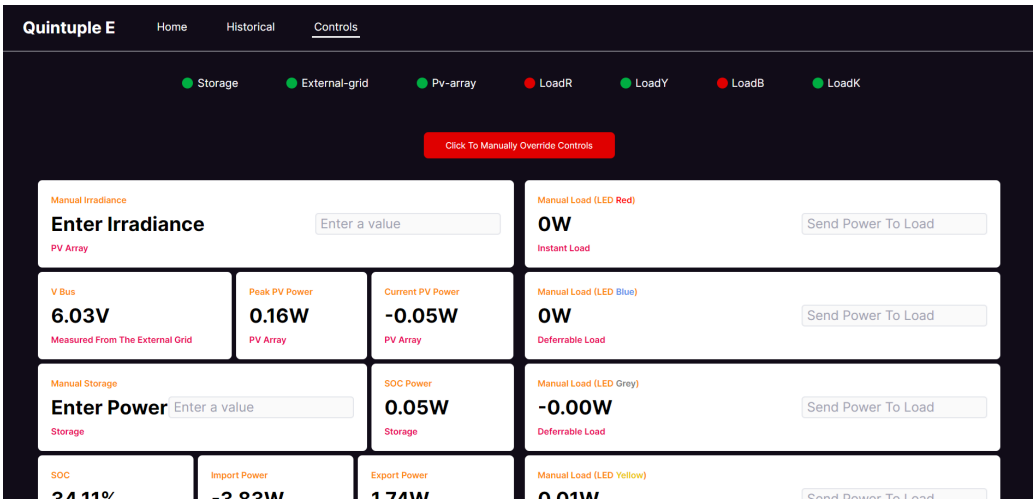


Figure 4.1: Not all devices are sending messages to the broker

the storage. Additionally, devices could be controlled via the dashboard. Figure 4.2 and 4.3 shows the effect of sending a positive power command to charge the storage to 60%, sending 1W to the red and yellow LEDs, 0W to the rest and 10% to the PV array, resulting in the output being 10% of the Maximum Power Point (MPP). The irradiance simulation code assumes that we always reach MPP, and the output has a linear relationship with irradiance.

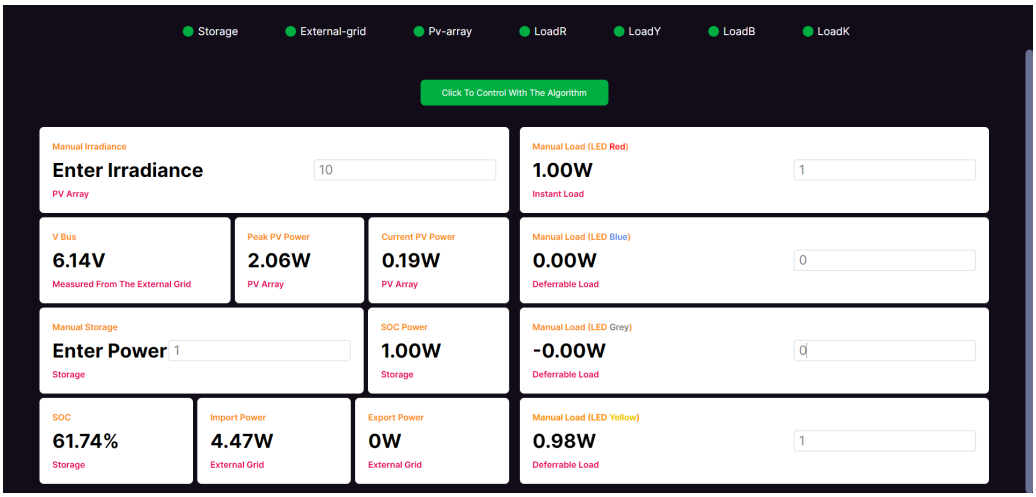


Figure 4.2: Controls page of the UI presenting the inputs and outputs

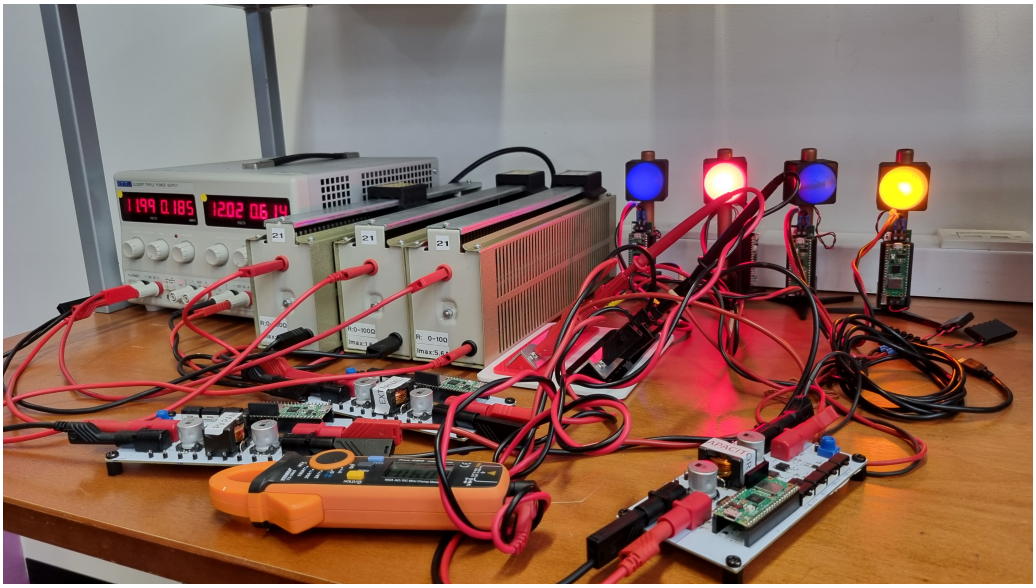


Figure 4.3: Hardware output given inputs shown above

These results satisfy requirement 3 (The software successfully communicates these actions to each hardware component), requirement 4 (the hardware components should execute these actions with minimal error) and requirement 7 (the UI should allow the user to manually provide actions to control the hardware).

4.2 Connecting the optimisation algorithm

With the broker, UI and hardware working correctly together, the backend service, logging with the database and the optimisation algorithm was the last thing left to connect. The demands sent from the external web server are too large for the loads to handle, since we are representing each load by the instantaneous demand and the other 3 deferrables. To account for this, the instantaneous demand and the deferrable powers sent to the load have been scaled down by a factor of 6, since the max value of the instantaneous demand is 6W.

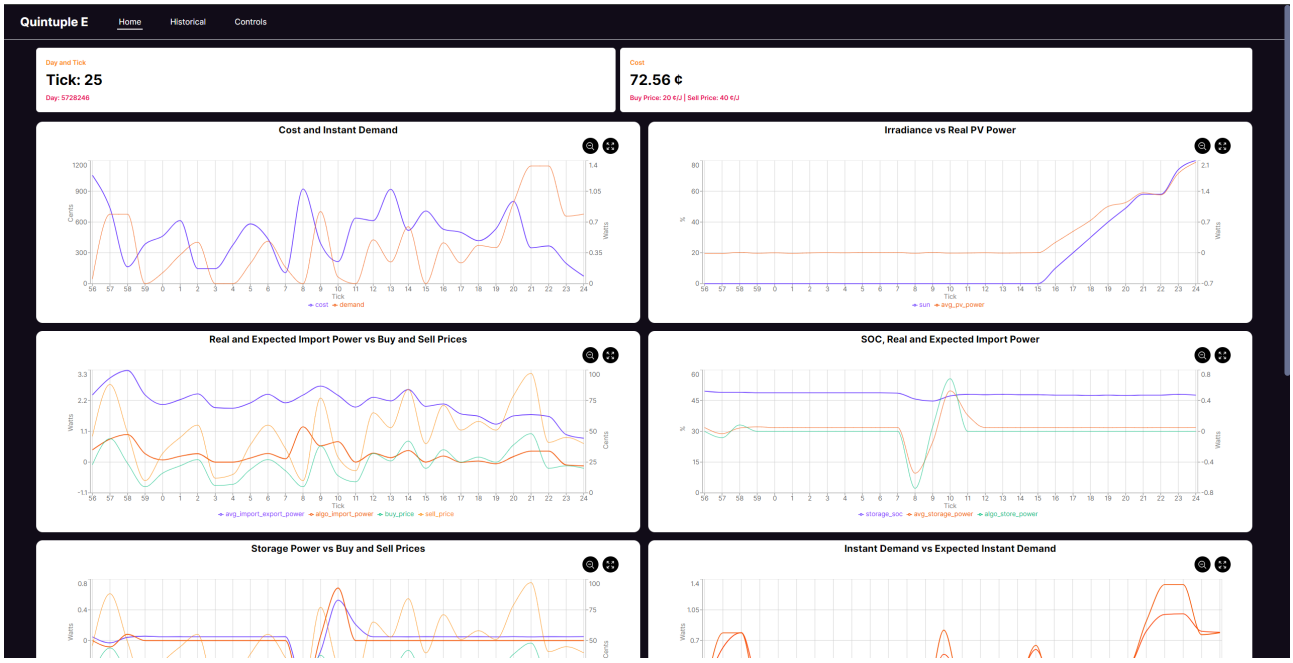


Figure 4.4: Home page data displayed after connecting the backend service

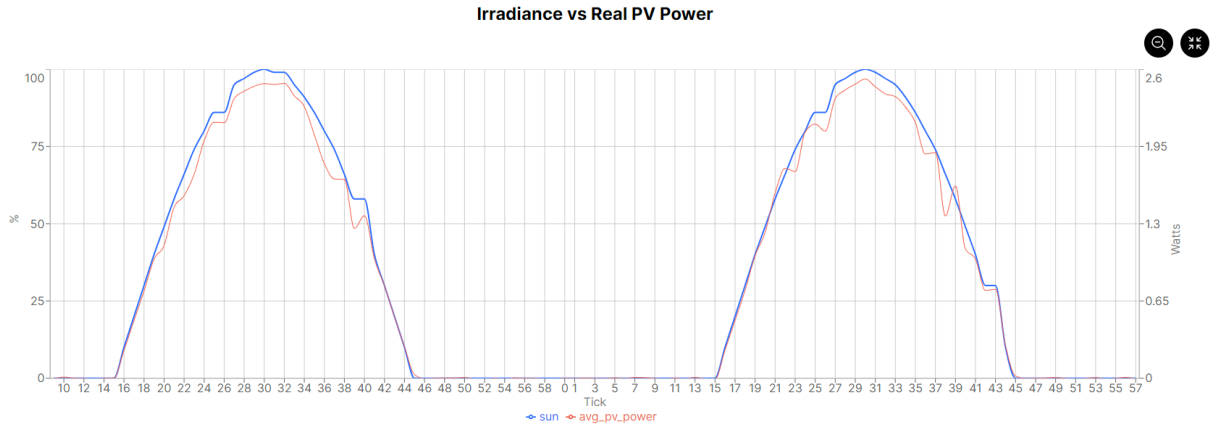


Figure 4.5: Plot of the irradiance sent to the PV array (blue) and the output of the PV array (red)

The backend service sent data to the Pico W's and the Pico W's were sending data to the backend service to process and store, this resulted in the data displayed on the Home page updating to Figure 4.4.

Figure 4.5 highlights that the PV array functions correctly, since a linear relationship is observed between irradiance value sent to the PV array and the actual output power of the PV array.

Figure 4.6 highlights the change in the state of charge and the power sent to the supercapacitor from the backend service. A problem we faced with the storage was that the output of the algorithm demands large changes of energy from the storage, resulting in large power values sent to the storage, hence, the power sent was halved. This enabled the power of the storage to follow the power it was sent from the backend service quite well, although it could not respond to fast changes in power demanded by the algorithm.

Figures 4.7, 4.8, 4.9 and 4.10 show the comparison of the powers sent to each of the loads the actual power output of the load.

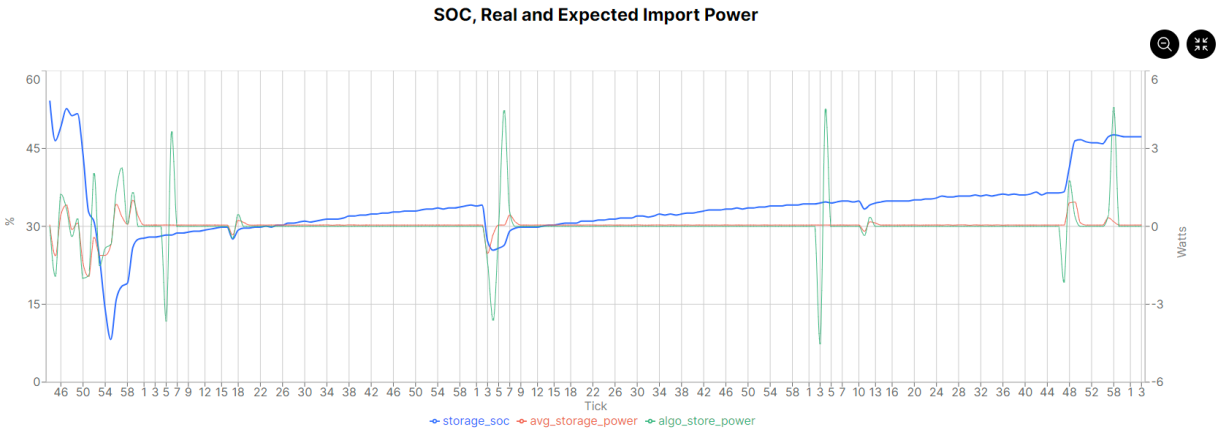


Figure 4.6: Plot showing the state of charge (Blue), the power sent to the storage (Green) and the actual power used in the storage (red)

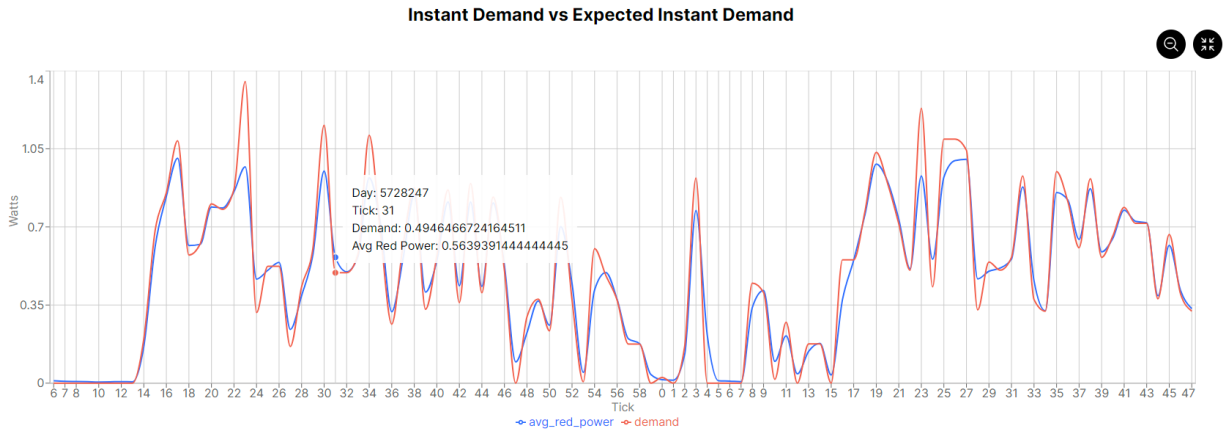


Figure 4.7: Plot of the instantaneous demand and the output power of the Red LED load, which is used to represent the instantaneous demand

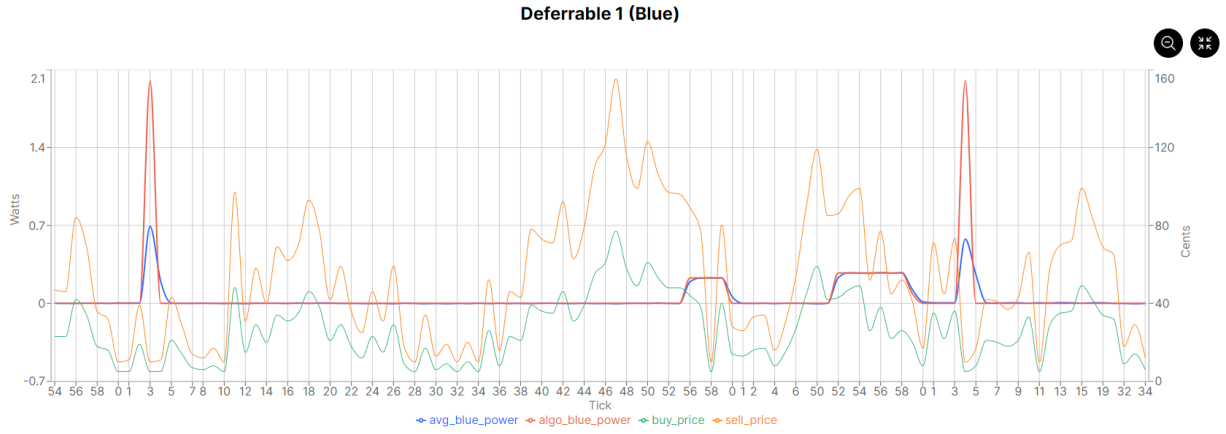


Figure 4.8: Plot of the power sent to deferrable 0 by the backend service (red) and the actual power consumed by the blue LED (blue)

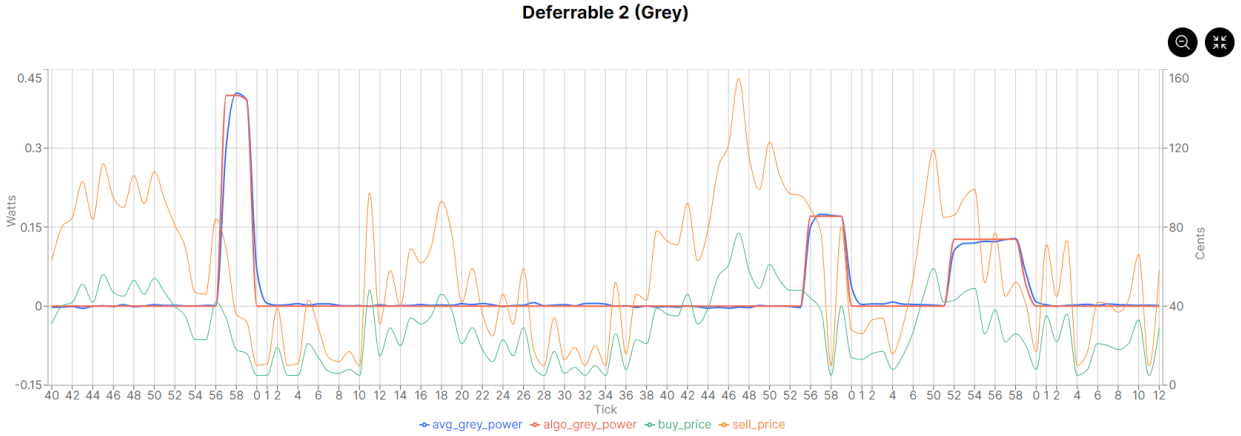


Figure 4.9: Plot of the power sent to deferrable 1 by the backend service (red) and the actual power consumed by the grey LED (blue)

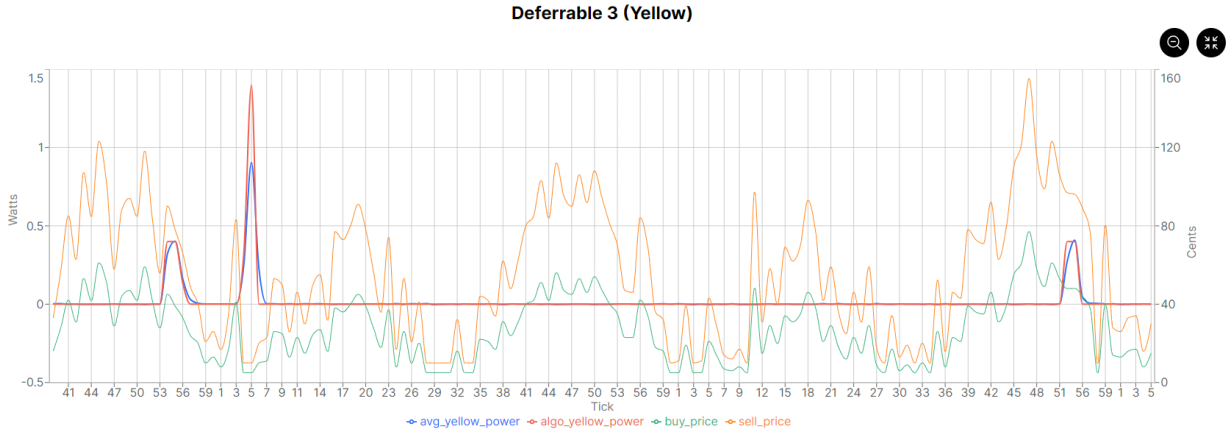


Figure 4.10: Plot of the power sent to deferrable 2 by the backend service (red) and the actual power consumed by the yellow LED (blue)

These results satisfy requirements 1 (The software receives data about the sun, price and demand requirements from the web server), 2 (The software processes the data to determine actions for each hardware component), 5 (The hardware components should be able to send the result of these operations back to the software) and 6 (The software should be able to store the requirements, actions and results in the database and display them on the UI). Based on this, all the requirements have been met.

Chapter 5

Conclusion

The purpose of this project was to design and emulate a household photovoltaic array system connected to an external grid, and to attempt to minimise energy costs when satisfying multiple demands in the form of load LEDs. We have also produced a user-friendly interface which interacts with a NoSQL database in order to display live and past performance of our smart grid. The user interface provides additional functionality with manual overrides for further control over the system. The final product has been thoroughly tested to confirm its stability in extreme simulated conditions. Alongside this, the smart grid is also equipped with contingency mechanisms to prevent complete failure, ensuring that in all scenarios the energy demand is met efficiently.

Based on the analysis conveyed, we conclude that Quintuple E's smart grid solution meets the required specifications and successfully reduces the daily energy costs compared to the naive methods mentioned earlier.

Recommendations

In future projects, the team would aim to conduct further research on the different types of algorithms that can be employed to minimise cost on a pseudo-random simulation as given in this smart grid project. This updated target would allow us to further optimise the current solution whilst still satisfying the project requirements.

Controllers were integral for stability handling in the various embedded systems such as SMPS modules for LEDs, External Grid, Storage, and PV Array. The majority controllers are simple PID controllers with modified kp/ki values which work well for current applications. Alternative controllers could have been researched and implemented especially in the External Grid subsystem to further minimise cost of keeping a stable main bus voltage.

Appendix A

Appendix

A.1 Appendix 1: Hardware Specifications

A.1.1 Lab SMPS

- **Specification**

Parameter	Value	Unit
Circuit Type	Buck or Boost	
Current Sensing	Bidirectional at Port B positive line	
Controller Type	Raspberry Pi Pico	
Controller Power Source	USB, Port A* or Port B*	
Base Code	SMPS_Bidirectional.py	
PWM Frequency	100	kHz
Port A Voltage*	0 - 17.5	V
Port B Voltage*	0 - 17.5	V
Max Current	5	A

*The Buck/Boost switch determines which port is used to power all the support circuitry on the board (Pico, Current Sensor, MOSFET drivers etc.), this port has a minimum voltage of 6-7V for this function. If the switch is set to Buck then Port A must meet the minimum voltage, if it's set to Boost then Port B must meet the minimum voltage.

A.1.2 LED Driver SMPS

- **Specification**

Parameter	Value	Unit
Circuit Type	Buck or Boost	
Current Sensing	Unidirectional at output negative line	
Controller Type	Raspberry Pi Pico	
Controller Power Source	USB or Input Port	
Base Code	LED_Driver.py	
PWM Frequency	100	kHz
Input Voltage	7 - 17.5	V
Output Voltage	0 - 17.5	V
Max Current	~500	mA

A.1.3 LED Loads

- **Specification**

Parameter	**Value**	**Unit**
Power Rating	1	W
Voltage Drop (varies with colour)	~3	V
Current Rating (also varies with colour)	~300	mA

A.1.4 PV Panels

Datasheet for PV Panel

- **Specification**

Parameter	**Value**	**Unit**
VOC	5	V
ISC	230	mA

A.1.5 Supercapacitor

Datasheet for Supercapacitor

- **Specification**

Parameter	**Value**	**Unit**
Max Voltage	18	V
Capacitance	0.25	F
ESR	~4	Ohms
Current Limit (5s Peak)	350	mA

A.2 Appendix 2: Next.JS Server Actions Created

Each server action is a REST API endpoint, the exact endpoints called have been abstracted away by the Next.JS framework into a set of function calls. Table A.1 is a list of the server actions created.

Function Name	Parameters	Description	Example Input	Example Output
getTick()	yValues	Gets all the documents in the ticks collection, getting only the values for the keys specified in the yValues object.	yValues = {demand: 1, sun: 1, avg_blue_power: 1}	{[day: 5768, tick: 2, demand: 2.1, sun: 10, avg_blue_power: 2.1], [day: 5768, tick: 3, demand: 2.8, sun: 30, avg_blue_power: 1.5],...}
getDayAndTick()	None	Gets the current day and tick from the external web server	None	{day: 5767, tick: 10}
getValuesOnTick()	yValues, day, tick	Gets the documents with the specified tick and day from the ticks collection, returning only the values for the keys specified in the yValues parameter	yValues = {demand: 1, sun: 1, avg_blue_power: 1}, day = 5768, tick = 2	{[day: 5768, tick: 2, demand: 2.1, sun: 10, avg_blue_power: 2.1],...}
getAllDays()	None	Returns all the documents in the day collection	None	{[day: 5768, deferrables: [{start: 0, end: 29, energy: 20}], ...],...}
getDeferrableData()	day	Gets all the documents in the ticks collection, getting only information about the deferrables allocated by the algorithm	day = 10	{[day: 10, tick: 0, algo_deferrables supplied: [0,0,0]], [day: 10, tick: 1, algo_deferrables supplied: [0,2,1]],...}

Table A.1: Description of Next.js actions and their example inputs and outputs.

A.3 Appendix 3: Failure Modes and Effects Analysis on Supercapacitor Storage

Design FMEA

Project	Smart Power Grid								**Version**	1							
Assembly	Storage								**Date**	06/12/2024							
Prepared by	Quintuple E								**Page____ of _____**								
Approved by	Ajith								**Notes**								
Process Purpose	**Potential Failure Mode**	**Severity**	**Potential Causes of Failure**	**Occurrence**	**Process Control**	**Detection**	**RPN**	**Recommended Actions**	**Area / person responsible**	**Delivery date**	**Action taken**	**Severity**	**Occurrence**	**Detection**	**New RPN**		
Store excess power or cheap power for discharge later when needed to supply demand at low sunlight ticks	Unable to charge/discharge	4	Failure of communication with server	2	Check real-time State of Charge (SoC)	7	56	Server tracks SoC and avoids power transfers reaching max/min capacity	Intishar Alam Misbahul	3rd June 2024	Limited the server power control values	2	1	9	18		
			SMPS board trips	1													
			Exceeding capacity limits														
	Inaccurate power transfer	3	Exceeding capacity limits	2		8	48	Improved constant power PID controller			Improved PID Controller	1					
			Oscillations in power line														
	Overcharge/Discharge	3	Large changes in power transfer	1	Limit duty cycle step size	9	27	Limit duty step size			Hard coded limit	1					

Figure A.1: Completed analysis to prevent various types of failure regarding the energy storage

A.4 Appendix 4: MQTT messages sent in the system

Table A.2: Table A.2: Message Descriptions

Sending Device	**Receiving Device**	**Topic**	**Description**	**Message**
Backend Service	UI	ui	Send the value of the override status stored by the backend service to the user interface	{"target": "override", "payload": <boolean>}
UI	Backend Service	server	Send the value of the override status stored by the user interface to the backend service	{"target": "override", "payload": <boolean>}
UI	Backend Service	server	Ask for the value of the override status stored by the backend service	{"target": "override", "payload": "req"}
UI or Backend Service	PV array Pico W	pico/pv-array	Send the value of the irradiance to the PV array	{"target": "pv-array", "payload": <int>}
UI or Backend Service	Storage Pico W	pico/storage	Send the power value to the storage	{"target": "storage", "payload": <float>}
UI or Backend Service	Red Load Pico W	pico/loadR	Send the power value to the red LED	{"target": "loadR", "payload": <float>}
UI or Backend Service	Blue Load Pico W	pico/loadB	Send the power value to the blue LED	{"target": "loadB", "payload": <float>}
UI or Backend Service	Yellow Load Pico W	pico/loadY	Send the power value to the yellow LED	{"target": "loadY", "payload": <float>}
UI or Backend Service	Grey Load Pico W	pico/loadK	Send the power value to the grey LED	{"target": "loadK", "payload": <float>}
PV array Pico W	UI or Backend Service	server	Send the output power of the PV array	{"target": "pv-array", "payload": <float>}
Storage Pico W	UI or Backend Service	server	Send the power used by the storage	{"target": "storage", "payload": {"type": "power", "value": <float>}}
Storage Pico W	UI or Backend Service	server	Send the state of charge of the storage	{"target": "storage", "payload": {"type": "soc", "value": <float>}}
Storage Pico W	UI or Backend Service	server	Send the value of the V bus measured from the storage	{"target": "storage", "payload": {"type": "v-bus", "value": <float>}}
Red Load Pico W	UI or Backend Service	server	Send the value of the instantaneous power consumed by the red load	{"target": "loadR", "payload": <float>}