Imperial College London

SMART GRID

Electronics Design Project 2

Department of Electrical and Electronic Engineering

Smart Grid Report

Power Rangers-Green

https://github.com/SImoNJess/Power-Rangers-Green/tree/main

Authors:

Ang Li - 02381513

Haiyi Yu - 02403167

Ruizhu Tian - 02399096

Sam Hussey - 02383928

Ziqian Gao - 02373575

Date of Submission: 16 June 2025

Abstract

Solar panel (PV array) technology has become very popular in the last decade on a household scale for its ability to reduce energy bills and carbon footprint. However, there are challenges. Extra circuitry is required to connect PV arrays to household devices, and household power demand varies a lot depending on time of day or season. At the same time, the PV array output power also varies greatly.

In practice, this leads to extra energy being imported from an external grid to make up for shortfall, or energy being exported to the grid when the PV array is producing an excess. Modern household systems also integrate energy storage to smooth out the inconsistent output power.

This report details the full development of a simulated household solar energy system, and presents both the issues that such a system faces, and our solutions to said issues.

Table of Contents

1 Introduction ... 3

1.1 Project Brief ... 3

1.2 Requirements ... 3

1.3 Decomposition ... 4

1.4 Project Management ... 4

1.5 Basic Components ... 5

1.6 Hardware Overview and Grid Configuration ... 5

2 Hardware Development ... 7

2.1 PV Array ... 7

2.2 Supercapacitor ... 12

2.3 LED Drivers ... 16

2.4 External Grid ... 20

3 Software Development ... 23

3.1 MQTT Server ... 23

3.2 Web Server ... 27

3.3 Machine Learning Algorithm ... 34

4 Testing and Evaluation of the Integrated System ... 43

4.1 System Startup Procedure ... 43

4.2 Control Mode 1: Manual Control via UI ... 43

4.3 Control Mode 2: Auto Control with Cost-minimisation Algorithm ... 44

4.4 Evaluation ... 46

5 Conclusion ... 47

5.1 Potential Improvements ... 47

6 Appendix: Hardware Components ... 48

7 Reference ... 48

1 Introduction

As renewable power generation systems have become more common and widespread, particularly on an individual household scale due to favourable government policies and financial incentives, the problems that face such systems have also become clearer. Complex systems of both hardware and software that frequently require maintenance are needed in order to mitigate these problems. At a basic level, power generation must be optimised as much as possible while minimising financial losses and having a high degree of system uptime to power household needs.

1.1 Project Brief

Tasked with investigating these systems, we were given our initial stakeholder requirements as shown in *Figure 1.1.1*. (https://github.com/edstott/EE2Project/tree/main/smart-grid)

Requirements

1. The system shall provide energy to load LEDs to satisfy the demands given by a third-party web server.
2. The system shall extract energy from a bench power supply (PSU) set up to emulate the voltage-current characteristic of a PV array.
   i. The configuration of the PSU shall be determined by characterising a supplied PV array
   ii. The current and/or voltage of the PSU shall be manually modulated to emulate the effect of the day/night cycle
   iii. The system shall use a switch-mode power supply (SMPS) with variable duty cycle to maximise the energy extracted from the emulated PV array
3. The system shall store excess energy in a provided supercapacitor for use at a later time
   i. No batteries shall be used
4. A mismatch between supply and demand of power shall be accommodated by importing from or exporting to an external grid, which is emulated by a PSU with an energy sink.
   i. The energy imported or exported shall be metered and converted to a monetary value using variable prices specified by a third-party web server
5. The system shall minimise the overall cost of importing energy with an algorithm to decide when to store/release and import/export energy. It shall also choose when to satisfy demands that can be deferred
   i. The algorithm should perform better than a naive algorithm that always acts to minimise the amount of power imported or exported at a given moment, and delivers demands as soon as they are requested.
6. There shall be a user interface that displays current and historic information about energy flows and stores in the system.

Figure 1.1.1 - Initial stakeholder project requirements

1.2 Requirements

Our solution is a simulated household solar energy system, where we are provided simulated varying values of important data like sun irradiance or power demand from the ICElec server.

- 4 LEDs emulating household devices must be powered at varying levels, based on server data
- LEDs must operate at DC voltage and can be powered from 3 sources; the simulated PV array, a supercapacitor, and an external grid.
- A web server will display all information to the user with a simple UI that allows for user interaction
- The server will also run machine learning models, allowing the system to adapt to changes and providing flexibility compared to a naïve system, increasing system performance and minimising cost.
- Components will communicate through an MQTT server, providing rapid and lightweight communication, reducing latency and making the simulated system closer to the real one.

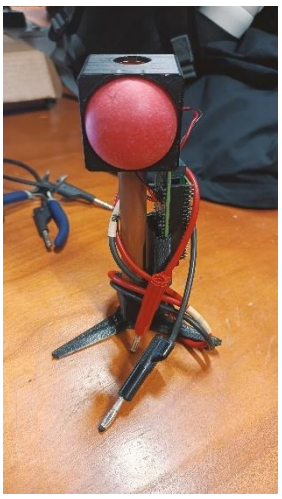

Figure 1.1.2 - Single LED system

1.3 Decomposition

We decomposed the project into separate modules so that each module can be allocated to different people, allowing modules to worked on simultaneously.

1. **PV Array** – an emulated PV array that provides varying power based on provided values of irradiance.
2. **Supercapacitor** – can charge or discharge energy
3. **LED** – can consume a set amount of power, representing household devices
4. **External Grid** – buy energy from or sell energy to the grid, at a variable price
5. **MQTT Server** – provides rapid communication between hardware and software
6. **Web Server** – provides a user interface to display data, and allow for user interaction
7. **Machine Learning Algorithms** – makes decisions in order to minimise cost

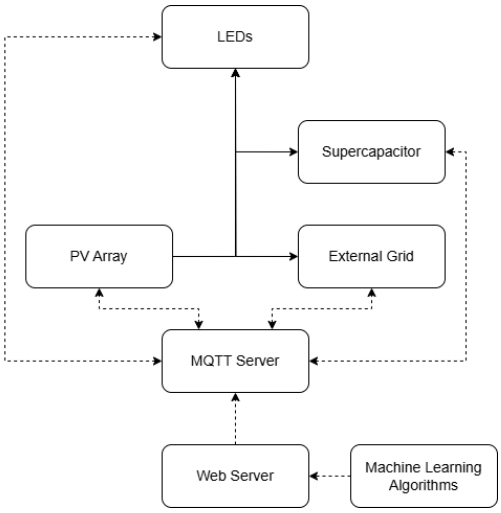

Figure 1.3.1 - System module overview

1.4 Project Management

After clearly defining modules, we assigned each one to several team members depending on their specialty. During the course of development, team members could help on other modules if it was needed.

The modules can be broadly split into "hardware" (1-4) and "software" (5-7) modules. We organised our project so that hardware and software are worked on concurrently, and therefore developed with each other in mind.

We also produced a Gantt chart to give us feedback on how fast we were progressing, and where extra time would be needed. We left slack so that if some tasks overran, it wouldn't cause issues.

Task	Sub-task Description	Members	Start Date	End Date	No. of Days
1. HARDWARE CHARACTERISATION	(a) Characterise PV array	Sam/Ruizhu/Ang	Mon 19/05/2025	Tue 20/05/2025	2
	(b) PV Array Emulator using PSU	Sam/Ruizhu/Ang	Wed 21/05/2025	Fri 23/05/2025	3
	(c) Design MPPT/Irradiance Algorithm	Sam/Ruizhu/Ang	Wed 21/05/2025	Fri 23/05/2025	3
	(d) Supercapacitor for Energy Storage	Ang	Mon 26/05/2025	Mon 02/06/2025	7
	(e) LED drivers	Sam	Mon 26/05/2025	Mon 02/06/2025	7
	(f) External Grid	Ruizhu	Mon 26/05/2025	Mon 02/06/2025	7
2. GRID AND SERVER INTEGRATION	(a) Design the Grid configuration	Sam/Ruizhu/Ang	Mon 02/06/2025	Tue 03/06/2025	2
	(b) Receiving Demand from server & Supply power to LEDs	Sam/Ruizhu/Ang	Mon 02/06/2025	Fri 06/06/2025	5
	(c) Testing of the integrated hardware & software system	Sam/Ruizhu/Ang	Mon 02/06/2025	Tue 10/06/2025	9
3. WEB SERVER AND DATABASE	(a) UI Design and Implementation	Haiyi	Mon 19/05/2025	Fri 23/05/2025	5
	(b) Display of Line Graphs	Haiyi/Ziqian	Fri 23/05/2025	Fri 30/05/2025	7
	(c) Server and Communcation Design	Haiyi	Fri 23/05/2025	Fri 30/05/2025	7
	(d) Database Design	Haiyi/Ziqian	Mon 02/06/2025	Fri 06/06/2025	5
4. TRADING ALGORITHM	(a) Research on Machine Learning Models	Ziqian	Mon 19/05/2025	Wed 21/05/2025	3
	(b) Price Prediction Algorithm	Ziqian	Mon 19/05/2025	Mon 26/05/2025	7
	(c) Optimisation Algorithm for Energy Import/Export	Haiyi/Ziqian	Mon 26/05/2025	Wed 04/06/2025	10
	(d) Communication with hardware (LEDs and Capacitor)	Haiyi/Ziqian	Wed 04/06/2025	Fri 06/06/2025	3
	(e) Testing of the software system	Haiyi/Ziqian	Mon 09/06/2025	Fri 13/06/2025	5
5. PRESENTATION AND REPORT	(a) Presentation	All	Mon 26/05/2025	Mon 02/06/2025	7
	(b) Report	All	Mon 02/06/2025	Sun 15/06/2025	14

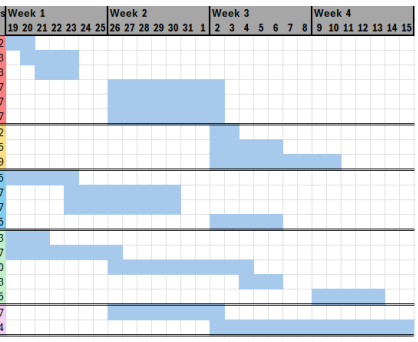

Figure 1.4.1 - Gantt Chart

We used an agile methodology of continuous improvements to each module until they are all in their ideal state. Once they were, we combined them together into the whole system to perform testing.

Module	Team Members
PV Array	Ang, Ruizhu
Supercapacitor	Ang
LED Drivers	Sam, Ruizhu
External Grid	Ang, Ruizhu
MQTT Server	Sam
Web Server	Haiyi
Machine Learning Algorithms	Ziqian

Figure 1.4.2 - Team member module allocation

1.5 Basic Components

Some components are common to several modules.

1.5.1 SMPS

All of the electrical components will be connected using an SMPS (Switch-Mode Power Supply) unit that is software-controlled using a Raspberry Pi Pico W. We are given 3 bidirectional SMPS units (*Figure 1.5.1*), and 4 buck SMPS units (connected to LEDs as shown in *Figure 1.1.2*).

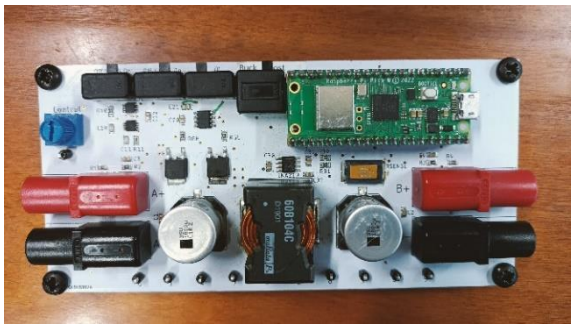

Figure 1.5.1 - Bidirectional SMPS unit with Raspberry Pi Pico W

The Raspberry Pi Pico W can control the duty cycle of the SMPS, which changes how much power can flow through it. It can also connect to servers using Wi-Fi, which is the main way that information will be transmitted between modules.

1.5.1.1 SMPS Efficiency

The SMPS introduces power losses that can be characterised by an efficiency that depends on output current. The efficiency curve depends on whether the SMPS is in buck or boost configuration.

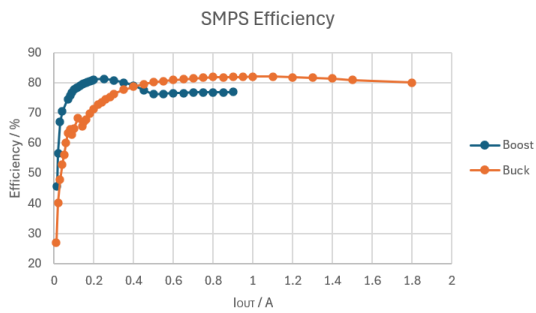

Figure 1.5.2 - Efficiency vs. current for Boost and Buck SMPS

In both cases, the efficiency is very low (≤50%) for low currents (≤50mA), and there is a maximum efficiency, after which the efficiency begins to decrease. For typical operating points, efficiency is on average ≈80%.

This introduces financial considerations that will be elaborated on in the conclusion.

1.5.2 PID Control

In order to control the output of the SMPS, we will commonly use a PID (proportional-integral-differential) controller. The controller is given a "setpoint" or target value that can be any measurable quantity, such as current or voltage.

The controller's general form is:

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

$e(t)$ is the error between the setpoint and actual value. The constant K_p is the proportional gain, K_i is the integral gain and K_d is the derivative gain. $u(t)$ is the controller output, or the SMPS's duty cycle in this case.

The proportional term affects response time, the integral term eliminates the steady state error, and the derivative term counteracts sudden fluctuations. The PID controller is used because it is very flexible and can be tuned for many situations.

1.6 Hardware Overview and Grid Configuration

The grid (comprised of the PV array, LEDs, supercapacitor and external grid) is built around a DC bus. For all SMPS modules, an inherent requirement is that $V_A > V_B$ due to diodes in the circuit. Therefore, we must choose both a bus voltage and SMPS polarity such that this requirement is always satisfied.

The PV emulator is interfaced with a boost SMPS to supply maximum PV power to the DC bus through MPPT. Four LED loads draw power directly from the DC bus, while a bidirectional SMPS interfaces with the external grid to maintain the bus voltage via import/export actions. Surplus energy is stored in a supercapacitor for later use. A key limitation of this grid setup is the ≈80% SMPS efficiency, causing internal power losses and increased grid power cost.

We choose a bus voltage of 7V. This is because it is close to the maximum PV voltage of 5.68V, minimising the amount of extra energy to be imported. It also allows the supercapacitor to have a greater range of voltage (above 7V) than if a higher voltage was chosen. A lower voltage was not chosen since the SMPS efficiency would be lower, causing greater power loss.

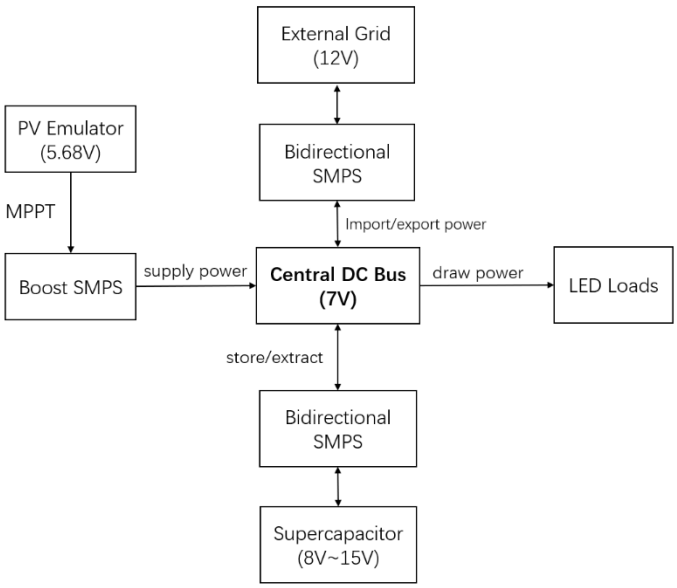

Figure 1.6.1 - Electrical grid configuration

2 Hardware Development

2.1 PV Array

2.1.1 Subsystem Specification

- The subsystem must accurately emulate the I-V characteristic of the real PV array outdoors under constant irradiance.
- The subsystem must maximise PV power with an MPPT algorithm.
- The subsystem must track different Maximum Power Points (MPP) corresponding to different irradiances.
- The subsystem's power output must have a 1% settling time of ≤250ms.

2.1.2 Design

Since the PV array exhibits degraded performance under artificial lighting inside the lab, a PSU is configured to emulate the I-V curve of the PV array. The PV emulator is subsequently interfaced with the input of a boost SMPS, whose output is linked to the 7V DC bus. This setup allows the PV output power to be efficiently transferred to the DC bus, enabling further control using maximum power point tracking (MPPT).

2.1.2.1 Characterisation of PV Array Outdoors

To characterise the I-V curve of a PV array, four PV cells were connected in parallel to form a non-ideal current source. This array was connected to a 100Ω rheostat, and placed under consistent outdoor irradiance. If we change the load resistance, the PV voltage changes which in turn changes the current: $I_{PV} = \frac{V_{PV}}{R_{LOAD}}$.

The I-V and P-V curves ($P_{PV} = I_{PV}V_{PV}$) measured are shown in *Figure 2.1.1* and *Figure 2.1.2*. If we connect a low impedance load (0Ω), we will draw a large current but a low voltage, and the power yield will be poor. If we connect a high impedance load (100Ω), then the voltage will be high but little current will flow and the power yield will be low. Between these extremes, an intermediate load exists that maximises the power at 2.2W. This MPP can vary with irradiance and temperature.

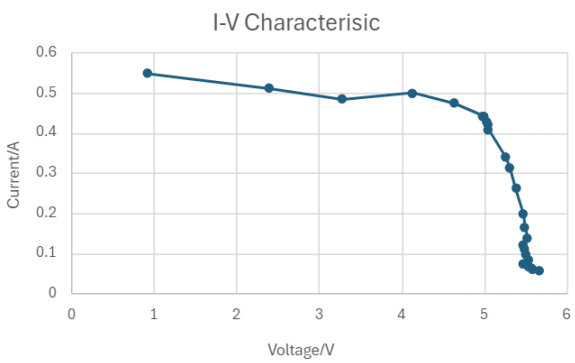

Figure 2.1.1 – I-V characteristic of PV array under intermediate irradiance

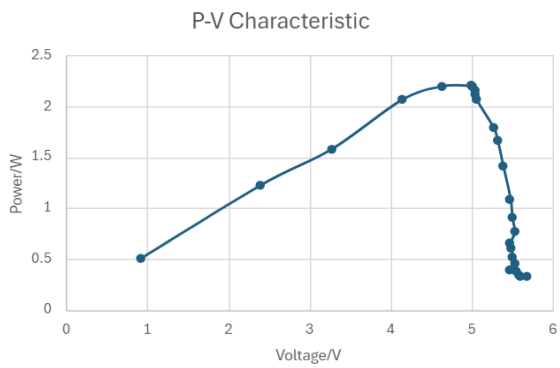

Figure 2.1.2 – P-V characteristic of PV array under intermediate irradiance

Parameter	Value
Open-circuit voltage V_O	5.68V
Short-circuit current I_{SC}	0.547A
Voltage at MPP V_M	4.99V
Current at MPP I_M	0.441A
Maximum power P_M	2.2W

Figure 2.1.3 - Summary table of measured parameters

2.1.2.2 PV Emulator using PSU

The PSU emulates the I-V curve of the PV array under maximum irradiance by increasing the current limit from 0.547 A to 0.8 A, shifting the MPP from 2.2 W to 3.5 W. Series (R_S) and parallel (R_P) resistors are added to shape the slopes of the curve realistically for MPPT testing.

$$Series\ Resistor: I_M = I_S - \left(\frac{V_O}{R_P}\right)$$

$$Parallel\ Resistor: V_M = V_O - I_M R_S$$

Experimental Setup:

- **Boost SMPS:** The boost converter steps up the input voltage (PV emulator = 5.68 V) to the output voltage (DC bus = 7 V).
- **Input (Port B)**: PV Emulator (5.68V PSU + R_S + R_P).
- **Output (Port A)**: 7V DC bus (PSU) with 10 Ω parallel resistor.

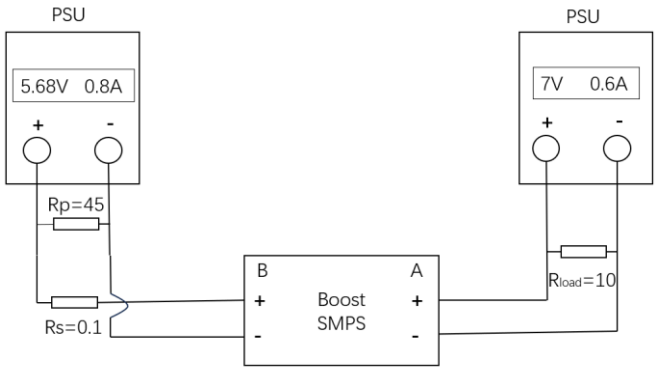

Figure 2.1.4 - The experimental setup for PV Emulator + Boost SMPS + Load (7V DC bus)

A duty sweep program was performed to vary the duty cycle and hence the voltage ratio of the Boost SMPS**:**

$$k = \frac{V_{\text{out}}}{V_{\text{in}}} = \frac{1}{1-\delta}$$

By changing the duty cycle δ, the referred load impedance R′ (resistance at the output of the SMPS referred to the input, see *Figure 2.1.5*) seen by the PV emulator changes as:

$$R' = \frac{R}{k^2}\text{ , Assuming ideal SMPS with } k_i = k_v = k \geq 1$$

This enables the PV emulator to observe a range of effective load impedances, from nearly short-circuit (0Ω) to open-circuit conditions (10Ω), and reproduce the complete I-V curve for the PV emulator. The resulting curve closely matches the real PV array behaviour and we can note the duty cycle and voltage at the MPP for testing MPPT algorithms later: At MPP, duty = 11400 and P_{max} = 3.64W.

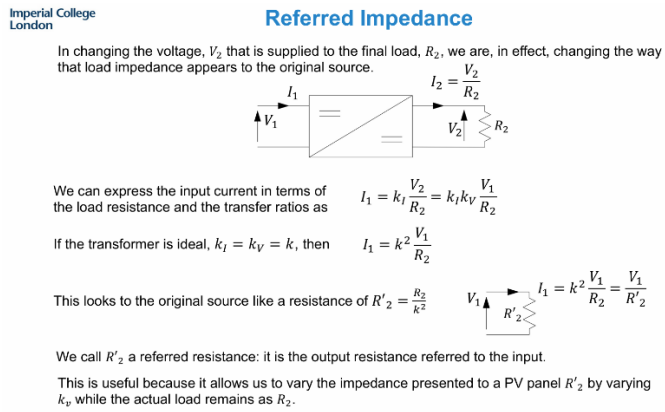

Figure 2.1.5 - Load impedance at output of SMPS referred to the input (Source: Electrical Power Engineering Lecture 3, Prof. Tim Green [1])

2.1.2.3 PV Emulator Results

Parameter	Value
Open-circuit voltage V_O	5.68V
Short-circuit current I_{SC}	0.8A
Series Resistor R_S	0.1Ω
Parallel Resistor R_P	45Ω
Voltage at MPP V_M	5.457V
Current at MPP I_M	0.667A
Maximum power P_M	3.64W

Figure 2.1.6 - Summary table of PV Emulator parameters

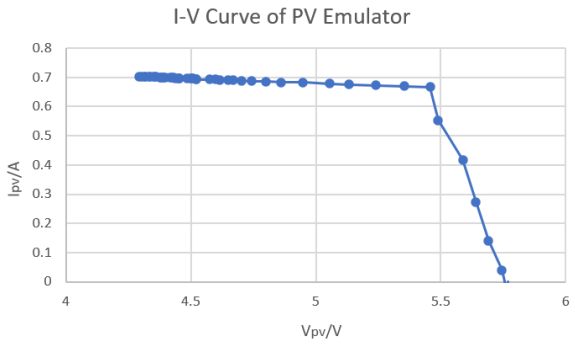

Figure 2.1.7 - I-V characteristic of PV Emulator assuming 100% irradiance

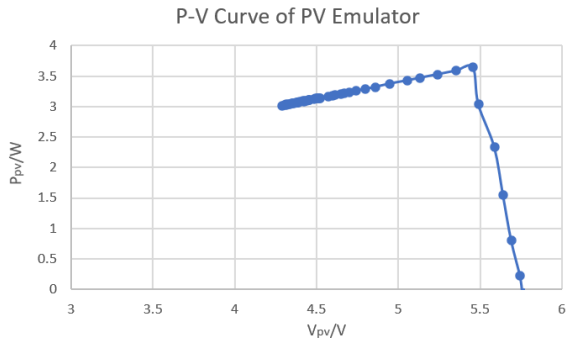

Figure 2.1.8 - P-V characteristic of PV Emulator assuming 100% irradiance

2.1.3 Implementation

2.1.3.1 MPPT (Maximum Power Point Tracking) Algorithm

Since the MPP varies with temperature and irradiance, we implement a MPPT algorithm that maximises PV power irrespective of irradiance conditions. A Boost Closed-Loop SMPS is used to step up the PV voltage and extract maximum power with MPPT (Refer to stage 1: DC/DC Conversion in *Figure 2.1.9*).

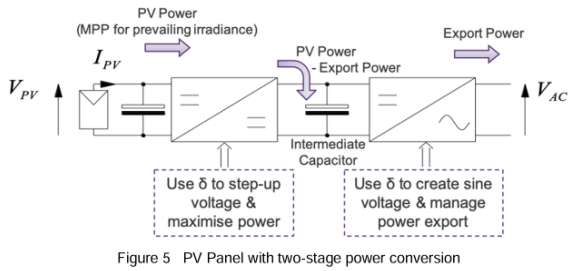

Figure 2.1.9 - PV interface using Boost SMPS with MPPT (only stage 1 is relevant in this scenario)

To track the MPP, a Perturb and Observe (P&O) algorithm is implemented, the PV power is calculated as:

$$P_{\text{pv}} = V_b \cdot I_L$$

The algorithm makes constant adjustments to the SMPS duty cycle, and measure the resulted PV power:

- **If** the power increases after the perturbation, the adjustment of duty cycle continues in the same direction.
- **Else if** the power decreases after the perturbation, the direction of adjustment is reversed. This allows fast convergence toward the MPP, and the MPP under maximum irradiance is noted.

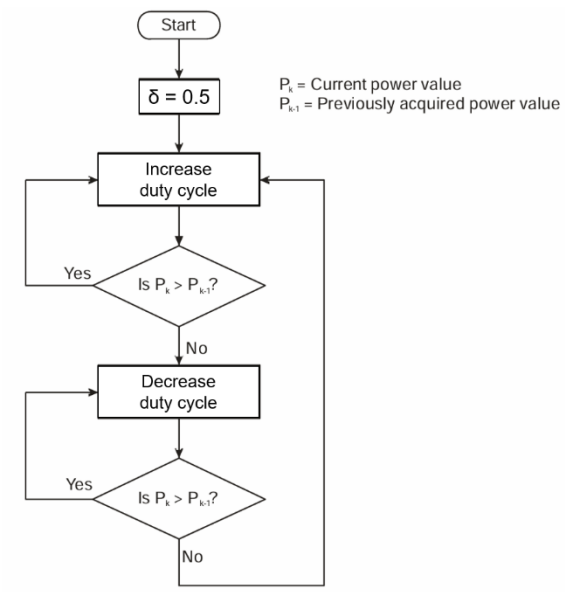

Figure 2.1.10 - Flowchart of P&O MPPT algorithm

```
# Perturb & Observe MPPT logic
if Ppv > prev_power:
    pwm_out += pwm_step * pwm_direction
else:
    pwm_direction *= -1
    pwm_out += pwm_step * pwm_direction

pwm_out = saturate(pwm_out, max_pwm, min_pwm)
pwm.duty_u16(pwm_out)

prev_power = Ppv  # update previous power
time.sleep_ms(2)
```

```
MPY: soft reboot
Duty = 20800
MPPT Vpv = 4.784 V, Ipv = 0.685 A, Ppv = 3.278 W

Duty = 19000
MPPT Vpv = 4.890 V, Ipv = 0.683 A, Ppv = 3.339 W

Duty = 18000
MPPT Vpv = 4.960 V, Ipv = 0.681 A, Ppv = 3.380 W

Duty = 15000
MPPT Vpv = 5.165 V, Ipv = 0.675 A, Ppv = 3.489 W

Duty = 13000
MPPT Vpv = 5.317 V, Ipv = 0.671 A, Ppv = 3.570 W

Duty = 11200
MPPT Vpv = 5.465 V, Ipv = 0.666 A, Ppv = 3.641 W

Duty = 11600
MPPT Vpv = 5.457 V, Ipv = 0.667 A, Ppv = 3.639 W

Duty = 11200
MPPT Vpv = 5.490 V, Ipv = 0.666 A, Ppv = 3.658 W

Duty = 11200
MPPT Vpv = 5.473 V, Ipv = 0.665 A, Ppv = 3.642 W
```

Figure 2.1.11 - P&O MPPT logic and its testing results showing quick convergence from mid-duty (20000) to the correct MPP at duty=11200 and P_{max}=3.64W

2.1.3.2 PID Power Control

The MPPT algorithm is used only for initial calibration to find $P_{\text{MPP, max irradiance}}$ at 3.6 W. During operation, a PID power control loop maintains PV output at the updated MPP, with irradiance values refreshed every 5 s from an external server to adjust the target power:

$$P_{\text{target}} = P_{\text{MPP, max irradiance}} \cdot \frac{\text{Irradiance}}{100}$$

The PID controller then adjusts the duty cycle to ensure the actual PV power tracks this target: Duty Adjustment $= k_p e(t) + k_i \int e(t)\ dt + k_d \frac{de(t)}{dt}, where\ e(t) = P_{target} - P_{pv}$

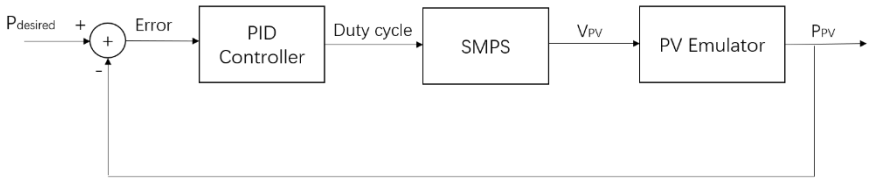

Figure 2.1.12 - Block diagram of the PID power control loop for PV Emulator

2.1.4 Testing

2.1.4.1 Procedure

The PID power control was tested with irradiance varying every 5-second tick, following the sequence: 0→25→50→75→100→75→50→25→0.

2.1.4.2 Results

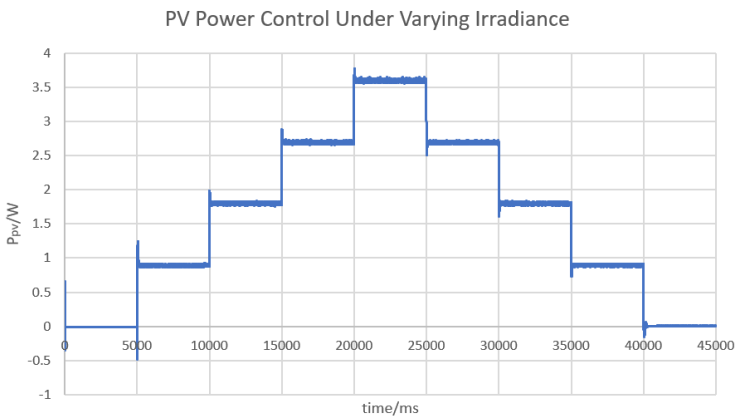

Figure 2.1.13 - PID-based PV power control under varying irradiance levels

1% Settling Time (Average) = 90 ms

1% Settling Time (Maximum) = 112 ms

2.1.4.3 Evaluation

The results, shown in *Figure 2.1.13*, demonstrate that the PID-based power control successfully tracks different Maximum Power Points as irradiance changes, as required.

The settling time of these power changes is on average 90ms, and even the maximum outlier at 112ms is well within the subsystem requirement of ≤250ms.

In addition, as shown earlier in *Figure 2.1.8*, the PV emulator provides an I-V characteristic approximating the real PV array as required. Finally, as shown in *Figure 2.1.11*, the MPPT algorithm, which can be enabled at will, successfully tracks the MPP with minimal oscillations.

Therefore, this subsystem specification is met.

2.2 Supercapacitor

2.2.1 Subsystem Specification

- The subsystem must store excess energy in a 0.5F supercapacitor for use at a later time.
- The subsystem must minimise the leakage/power loss due to ESR.
- The subsystem must prevent voltage from exceeding 17V to protect the SMPS and supercapacitor.
- The subsystem must be able to control the charge and discharge of set amounts of energy.

2.2.2 Design

The initial approach is to characterise a mathematical relationship between the stored energy and the SMPS duty cycle. This is followed by implementing a PID-based current control to regulate the stored energy by modulating the duty cycle.

An experimental setup was built to enable supercapacitor charging/discharging via a fixed 7 V DC bus:
• **SMPS**: A bidirectional SMPS with closed-loop control regulates bidirectional power flow.
• **Port B**: Connected to a 7 V DC bus with a 10 Ω parallel resistor to stabilise reverse power flow from the supercapacitor to the DC bus.
• **Port A**: Connected to two 0.25 F supercapacitors in parallel, storing up to 50 J ($E = CV^2$).

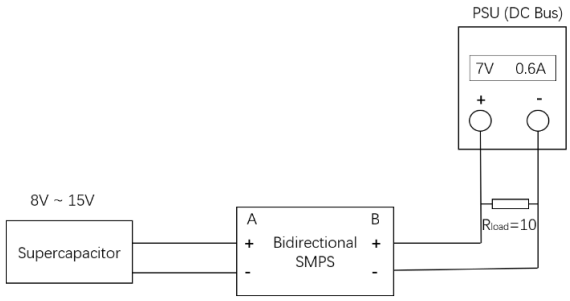

Figure 2.2.1 - Experimental setup of supercapacitor energy storage system

2.2.2.1 Characterisation

The characterisation of supercapacitor was performed through a duty sweep experiment, and two key plots were generated: Capacitor Voltage vs. Duty Cycle and Capacitor Energy vs. Duty Cycle.

Capacitor Voltage vs. Duty Cycle

As expected from boost SMPS theory, the capacitor voltage V increases nonlinearly as duty cycle increases:

$$\frac{V_A}{V_B} = \frac{1}{1-\delta}\ , where\ V_A: capacitor\ voltage, V_B: input\ voltage\ of\ 7V$$

In *Figure 2.2.2*, the relationship between capacitor voltage V and unnormalized duty cycle D (0~65534) is:

$$V = 7 \times 10^{-9} D^2 + 8 \times 10^{-5} D + 5.8172$$

Capacitor Energy vs. Duty Cycle

Since capacitor energy is a quadratic function of voltage, energy grows exponentially with respect to the duty cycle. In *Figure 2.2.3*, the relationship between capacitor energy E and unnormalized duty cycle D is:

$$E = 5 \times 10^{-8} D^2 - 1 \times 10^{-4} D + 9.2332$$

This demonstrates the feasibility of indirectly controlling the energy stored in the supercapacitor, by using a PID-based current control to modulate the SMPS duty cycle.

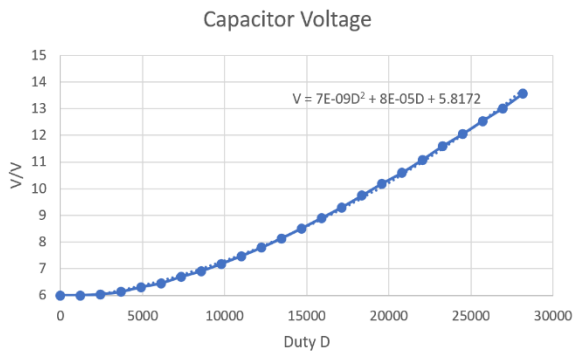

Figure 2.2.2 - Characterisation of capacitor voltage vs. duty cycle

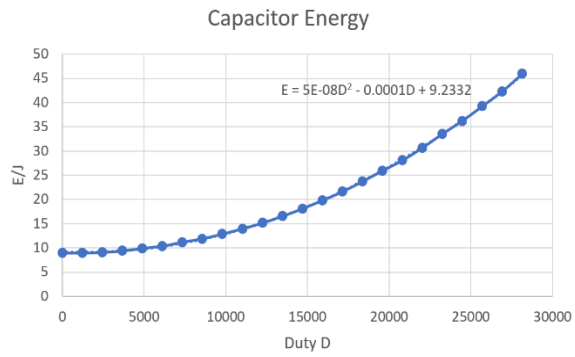

Figure 2.2.3 - Characterisation of capacitor energy vs. duty cycle

2.2.2.2 PID Current Control

A PID current control algorithm was implemented, which adjusts the duty cycle to ensure that current flowing into or out of the capacitor achieves energy target.

In every tick of 5 seconds, the system receives one of the three commands from the external webserver:

(a) Store 5J (S): Ensure positive charging current (+0.2A) from the DC bus to the supercapacitor.
(b) Extract 5J (E): Ensure negative discharging current (-0.2A) from the supercapacitor to the DC bus.
(c) Hold (H): Maintain the energy level by ensuring zero current flow.

Each command sets a target current $I_{desired}$, and a PID loop adjusts the duty cycle to drive measured current $I_{measured}$ toward it:

$$\text{Duty Adjustment} = k_p e(t) + k_i \int e(t)\ dt + k_d \frac{de(t)}{dt}, where\ e(t) = I_{desired} - I_{measured}$$

Hence the SMPS modulates capacitor voltage and energy accordingly. The action (Store/Extract) ends once the energy target is met by setting $I_{desired} = 0$.

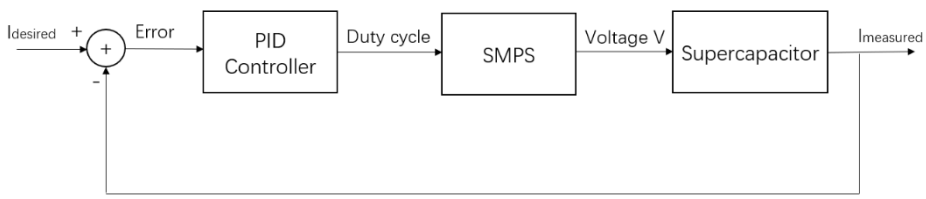

Figure 2.2.4 - Block diagram of the PID current control loop for supercapacitor

Command	$I_{desired}$	k_p	k_i	k_d
Hold	+0.02A	2000	0.1	0
Store 5J	+0.22A	2000	0.1	0
Extract 5J	-0.2A	2000	0.1	0

Figure 2.2.5 - Summary table of target current and PID gains for each command

```
# PID Current Control
err = I_target - IL
int_err += err
int_err = saturate(int_err, integral_max, integral_min)
der_err = err - prev_err

pid_output = kp * err + ki * int_err + kd * der_err
prev_err = err

duty += int(pid_output)
duty = saturate(duty, max_pwm, min_pwm)
pwm.duty_u16(duty)
```

Figure 2.2.6 - Code for PID current control of the supercapacitor

Practical Challenges and Solutions made to the PID Current control logic

(a) Energy loss during Hold due to parasitic resistance

During the Hold operation ($I_{desired}$ is nominally set to 0), capacitor energy slightly decreases due to ESR induced heat loss, and charge redistribution also lowers the capacitor voltage and energy.

Solution: Set $I_{desired}$ = 0.02A instead of 0A during Hold to compensate for energy loss and keep energy approximately constant.

(b) Asymmetry between Charging speed and Discharging speed

The charging/discharging speed is dependent on the current magnitude; supercapacitor charging is slightly slower than discharging, because the bidirectional SMPS operates in boost mode during charging and in buck mode during discharging. Boost is typically less efficient and delivers lower current than buck.

Solution: Use $I_{desired}$ = 0.22A for charging, and $I_{desired}$ = -0.2A for discharging. This compensation leads to increased charging current and charging speed.

(c) Overcharging Protection via State of Charge (SoC)

The SoC is estimated based on the current stored energy relative to max/min energy range:

$$\text{SoC} = \frac{E - E_{\text{min}}}{E_{\text{max}} - E_{\text{min}}} \times 100$$

If the SoC exceeds 90%, charging is automatically stopped. This prevents overcharging and ensures that the capacitor voltage never exceeds the rated 18 V, thereby enhancing the lifespan of the supercapacitor.

```
C = 0.25
max_capacity = C * 18**2
min_capacity = C * 7**2

def calculate_soc(E):
    return (E - min_capacity) / (max_capacity - min_capacity) * 100

# SoC-based Power Override
if soc >= 90 and P_desired > 0:
    P_desired = 0 # Prevents overcharging
```

Figure 2.2.7 - Overcharging protection code via SoC

2.2.3 Testing

2.2.3.1 Procedure

The PID control algorithm is tested with 8 operations (S, S, S, S, H, E, E, E), each lasting 5 seconds. The expected energy step per Store/Extract operation is 5 J; however, actual changes may vary slightly (e.g., 4.5 J) due to system dynamics.

2.2.3.2 Results

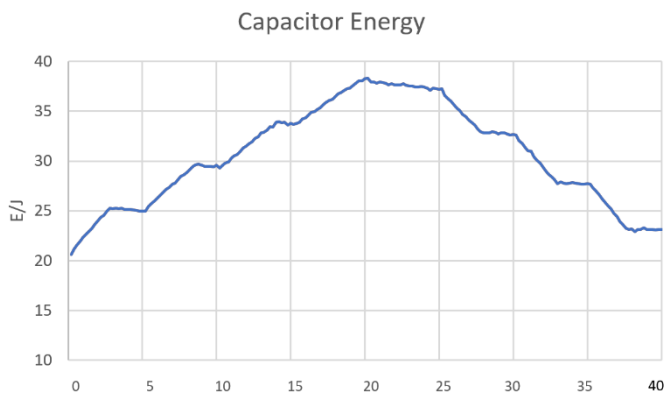

Figure 2.2.8 - Testing results for supercapacitor control in a duration of 40 seconds (8 operations)

2.2.4 Evaluation

Each energy step remains within an acceptable error margin of ±10% (see *Figure 2.2.8*), demonstrating reliable controllability in achieving the targeted energy storage. Since there is a linear current–duty cycle relationship, the use of current control simplifies PID tuning and improves system stability (whereas power control is non-linear). Current control also offers better start-up and low-voltage protection by limiting current during initial charging, preventing PSU stress when the capacitor voltage is below 7V.

Since the charging and discharging can be strictly controlled and voltage cannot exceed 17V, this subsystem's specification is met.

2.3 LED Drivers

2.3.1 Subsystem Specification

- The subsystem must consume a variable amount of power demanded by the user (±1% accuracy on average)
- The subsystem must be able to change its power value with a settling time of ≤100ms
- The subsystem must not consume more than 1W of power in each LED

The aim is to regulate current to satisfy the power demands from the user. Each LED driver has a buck SMPS controlled by a Raspberry Pi Pico, which is how the power consumption can be controlled.

2.3.2 Design

In order to control power consumption, we will use a PID controller since it can be easily tuned.

After the controller receives the power instruction from the MQTT broker, it calculates the actual power and uses the PID controller to make sure the actual power is set to the requested power. The actual power is published to MQTT broker every second. This is shown in *Figure 2.3-1*.

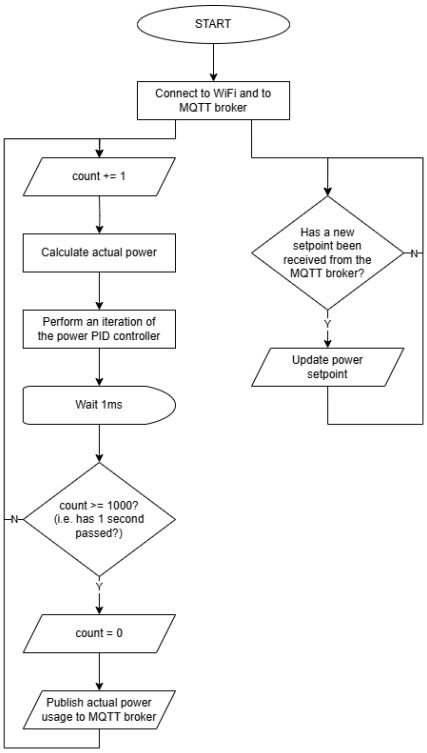

Figure 2.3.1 - Flowchart of LED driver

In order to further inform our design decisions, we characterise the LEDs we are given.

2.3.2.1 Characterisation

Each LED driver is characterized using the example code that varies the current setpoint of the PID controller. To calculate the current:

$$I = \frac{V_{ret}}{R_{total}}$$

is used. According to *Figure 2.3.2*, five 5.1Ω resistors are in parallel. Thus $R_{total} = 1.02\Omega$.

Each colour LED has different VI (voltage and current) and PI (power and current) characteristics. To make sure each LED doesn't exceed the power limit of 1W shown in *Figure 2.3.3*, the setpoint for the blue LED must be smaller than 0.3. The setpoint for the red and yellow LED can be around 0.35. From *Figure 2.3.4*, the power for the blue LED is slightly higher than others.

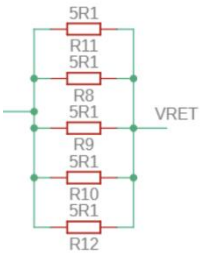

Figure 2.3.2 - SMPS current sensor

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

Figure 2.3.3 - LED specification

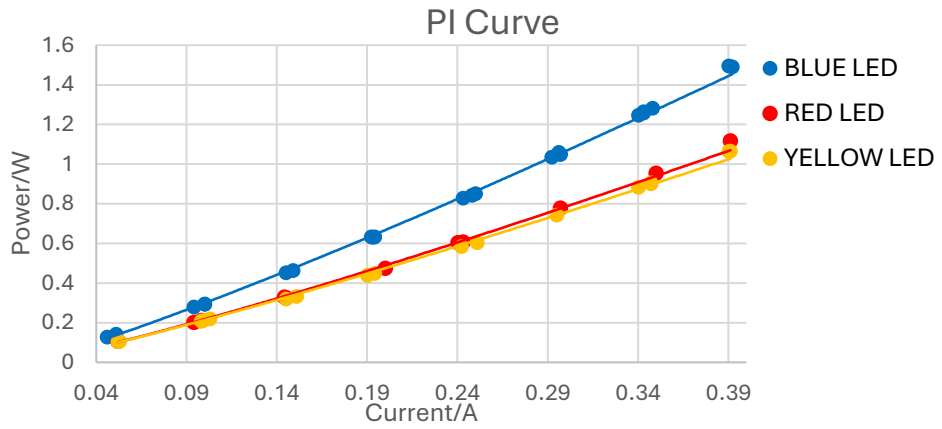

Figure 2.3.4 - PI characteristic of LEDs

2.3.2.2 Option 1 – Current Control:

To begin with, the relationship between power and current is roughly linearly proportional. Based on *Figure 2.3.4*, each LED has a line of best fit:

$$P = k \times I,$$

where k is the gradient. We use power divided by the gradient to get the setpoint current for a PID controller. Shown in *Figure 2.3.5*, this has a large inaccuracy because the line is not completely linear.

$P_{REQUEST}$(W)	P_{ACTUAL}(W)	Percentage error (%)
0.25	0.282	12.8

Figure 2.3.5 - Steady state error using current control

2.3.2.3 Option 2 – Power Control:

Alternatively, a PID controller for the power directly is another approach.

PID systems are mathematically designed for linear systems, and the relationship between power and duty cycle is not linear. However, the relationship is monotonically increasing, so while tuning may be difficult, the PID controller should work well.

2.3.2.4 Results

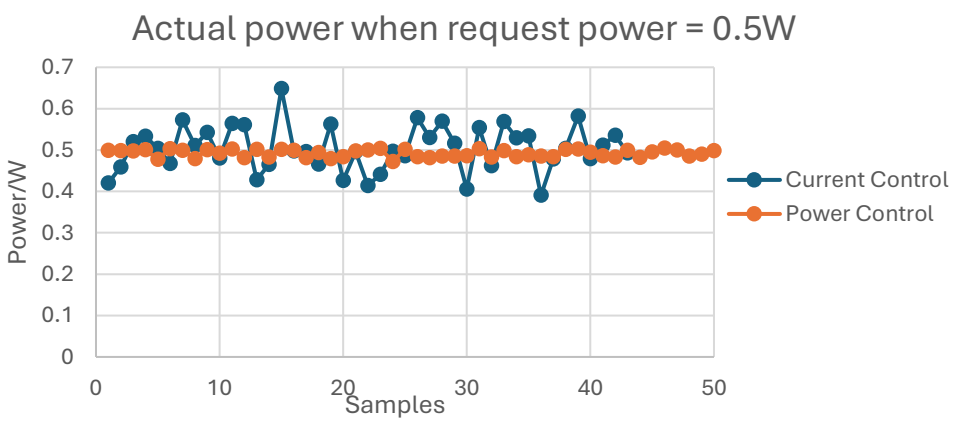

Figure 2.3.6 - Actual steady state LED power with current control and power control

2.3.2.5 Conclusion

Figure 2.3.6 shows the comparison between the current control and the power control. Power control has a higher accuracy, and fluctuates much less. This is because of difficulties in measuring the small current accurately, since in this case 0.5W corresponds to ≈150mA. The current measurement is therefore more susceptible to noise than the power measurement.

Therefore, we decided to use power control.

2.3.3 Implementation

Tuning K_p and K_i was done through trial and error, ensuring that the controller works at all reasonable setpoints (0 – 1W). Eventually, we settled on an optimal configuration that avoids overshoots while having a reasonable settling time: $K_p = 0.05, K_i = 10$. This is confirmed by testing (*Figure 2.3.8*).

We also added code so that the power setpoint is clipped at 1W, avoiding dissipating too much power.

We initially tested the driver by switching between 2 power setpoints every 5 seconds. However, we eventually implemented MQTT functionality as discussed in "MQTT Server".

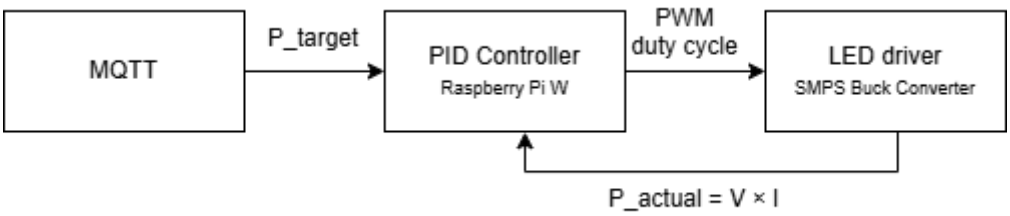

Figure 2.3.7 - Block diagram of LED power controller

The MQTT broker sends the target power value to the Raspberry Pi W, which then updates the setpoint of the PID controller. This can be done manually with an override, or automatically based on the machine learning algorithm.

2.3.4 Testing

2.3.4.1 Procedure

- Connect the LED to a 7V DC power supply
- Run the driver program
- Update the power setpoint (from 0W to 1W to show the full power range)
- Measure the 1% settling time
- Measure the accuracy of the steady-state power value

2.3.4.2 Results

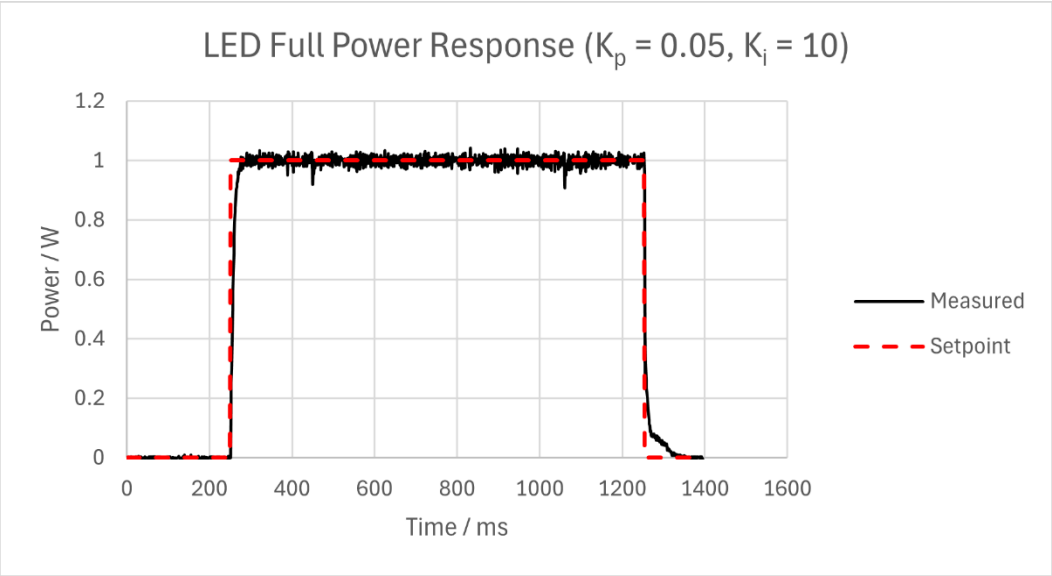

Figure 2.3.8 - LED full power response (between 0W and 1W)

1% Settling Time (0W → 1W) = 24 ms

1% Settling Time (1W → 0W) = 67 ms

Steady-State Power / W *(Note: After 1% settling time)*		
Mean	Maximum Deviation	Std. Dev
0.9992	0.093	0.015

2.3.5 Evaluation

The LED driver subsystem successfully responds to power demands by adjusting PWM duty cycle using a tuned PID controller. The settling times are rapid (≤100ms as required), and the steady-state average error is minimal (≤1%).

We did, however, observe occasional fluctuations around the steady-state value, which are typically within 15mW. For just 1-2 milliseconds, there are occasional undershoots of up to 93mW, which are immediately corrected. These fluctuations are very short-lived and should not cause any issues when operating, nor are they visible to the human eye.

There is also code that successfully clamps the power setpoint at 1W, so the actual LED power will never go above it.

For those reasons, we conclude that the LED driver subsystem meets its specifications.

2.4 External Grid

2.4.1 Subsystem Specification

- The DC bus voltage must be maintained at 7V (±5%).
- The DC bus voltage must have a 1% settling time after system startup of ≤200ms.
- The system must be able to export and import power bidirectionally.

2.4.2 Design

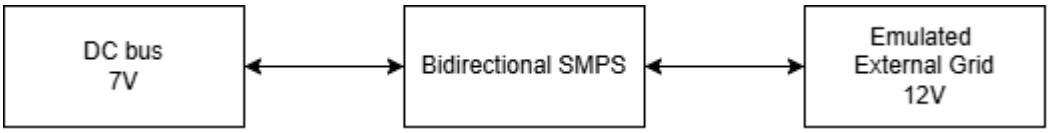

Figure 2.4.1 - Block diagram of external grid

As shown in *Figure 2.4.1*, one port of SMPS is connected to a DC power supply set to 12V in parallel with a resistive load of 50Ω, representing emulated external grid. The resistive load is essential because it acts as a current sink for exporting power. The other port of SMPS is connected to the DC bus.

2.4.2.1 Option 1 – Power Control

Using PID control of power, the amount of power being exported and imported can be controlled. However, it causes unstable DC bus voltage which doesn't satisfy the goal of maintaining DC bus voltage.

This is because each SMPS can only have one PWM signal output which is not possible to control power and voltage together.

2.4.2.2 Option 2 – Voltage Control

To address this, voltage control for DC bus voltage is required instead. The DC bus voltage needs to be controlled to be constant at 7V.

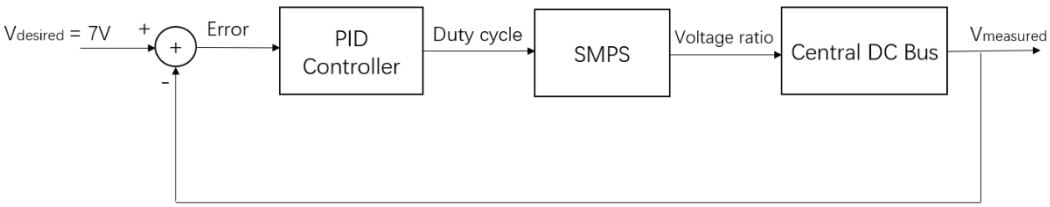

Figure 2.4.2 - Block diagram of voltage PID controller

The desired voltage is 7V. The input of PID controller is the error between the measured voltage and the desired voltage. Then, the PID controller output a duty cycle for PWM signal for SMPS. The SMPS outputs a voltage to the central DC bus to get the measured voltage to meet the desired voltage.

The external grid–interfaced SMPS maintains the DC bus voltage at 7V, automatically handling power import/export to balance the system. The resulting import/export energy exchange, combined with real-time buy/sell electricity prices, is used to calculate cost or profit, which is then displayed on the UI dashboard.

2.4.2.3 PID Tuning

A proper PID tuning is essential for achieving optimal system performance. We decided to prioritise fast convergence and lack of overshoot, since fast convergence prevents large currents from flowing and lack of overshoot prevents damage to other voltage-sensitive system components, especially the supercapacitor.

As shown in *Figure 2.4.3*, increasing integral gain leads to faster transient response, and changing proportional gain doesn't make a significant difference. For the best performance, we used $K_P = 150$ and $K_I = 300$. This provides a good balance between response speed and system stability.

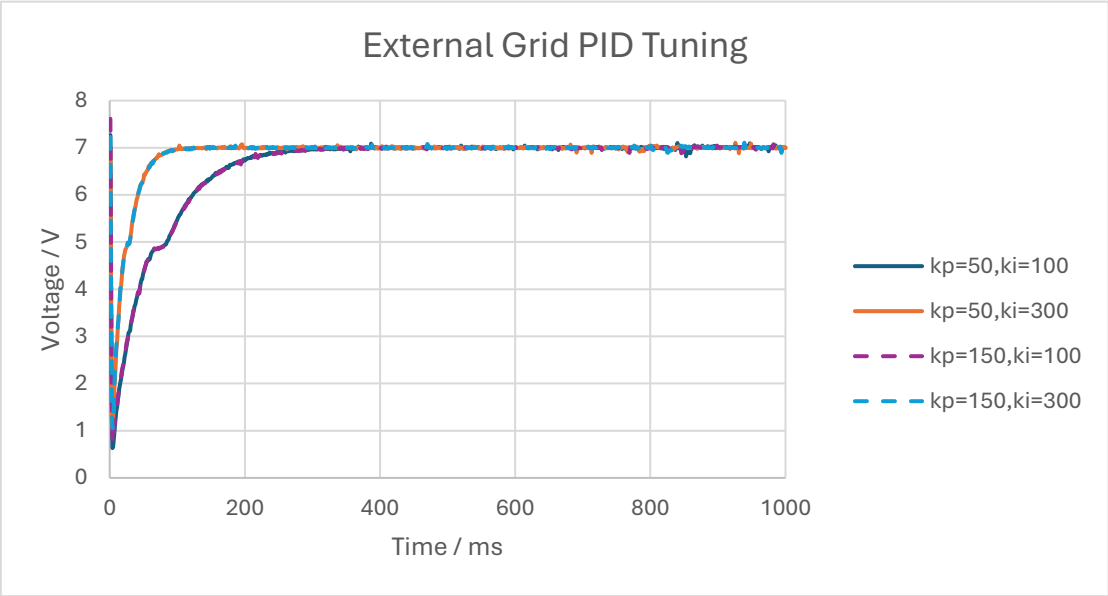

Figure 2.4.3 - Startup transient tuning for voltage PID controller

With these PID values, we observe a 1% settling time after startup of 89ms.

2.4.3 Testing

2.4.3.1 Procedure

In order to make sure that the DC bus voltage is maintained as constant, we must test the system together with other components to mimic the final configuration. We emulated the LEDs with a constant power load, and simulated the "worst-case scenario" of a sudden change of load from 0W to 4W.

- Connect the external grid and load to the DC bus
- Run the program
- Once in steady state, set the load to 4W, wait and set the load back to 0W
- Measure how the bus voltage changes with time

2.4.3.2 Results

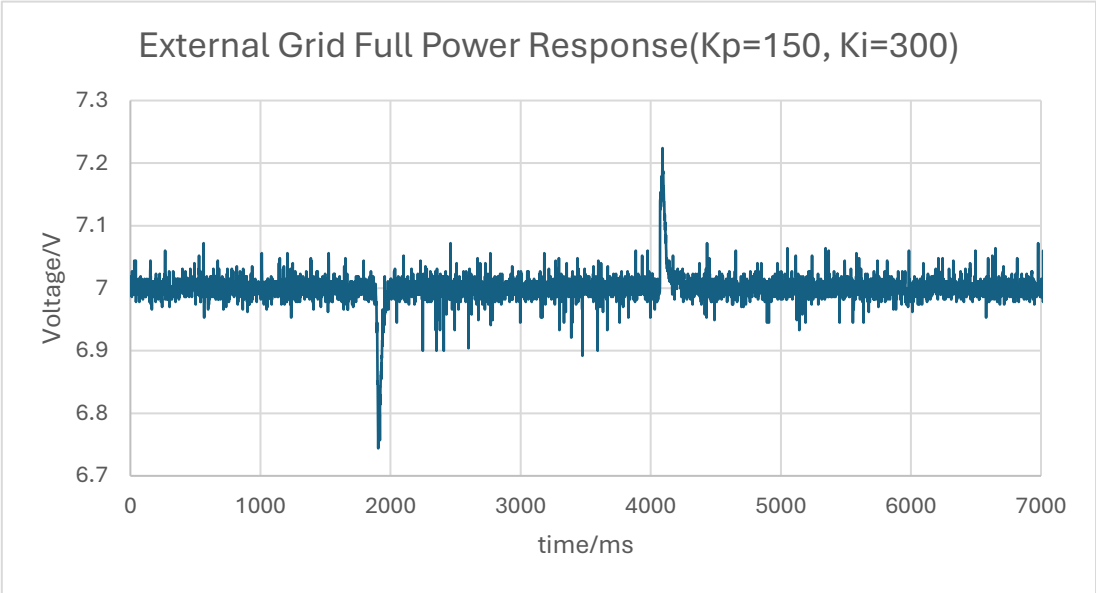

Figure 2.4.4 - External grid full power response (0W to 4W to 0W)

2.4.4 Evaluation

There are noticeable transients when the 4W load is both turned on and turned off. The error is corrected back to within 1% of 7V within just 19ms.

Even in this worst-case scenario, the bus voltage never leaves ±5% (±0.35V) of the 7V setpoint.

In addition, as shown in *Figure 2.4.3*, the startup transient (once Wi-Fi is connected) is 89ms, satisfying the requirement of ≤200ms.

Therefore, we conclude that this subsystem meets its specification.

3 Software Development

3.1 MQTT Server

3.1.1 Subsystem Specification

- The subsystem must be a server accessible by clients on a shared network.
- The subsystem must accept and store setpoint/decision data sent by a client.
- The subsystem must send relevant setpoint/decision data to specific clients.
- The subsystem must send data with an average latency of ≤500ms.
- The subsystem must be able to synchronise all setpoint changes within ≤200ms of one another.

3.1.2 Design

MQTT (Message Queuing Telemetry Transportation) is a network protocol that prioritises being lightweight. Its low latency and minimal required processing power makes it meet the requirements well.

It works using a "publish/subscribe" model, where clients can subscribe to, or publish messages to, a topic. Once a client publishes a message to a topic, then all subscribed clients will be sent the message.

Since we require setpoint/decision data to be sent to SMPS units, each SMPS will be a client of the central "broker" server, and the web server (that makes decisions) will be another client. This data flow is shown in *Figure 3.1.1*.

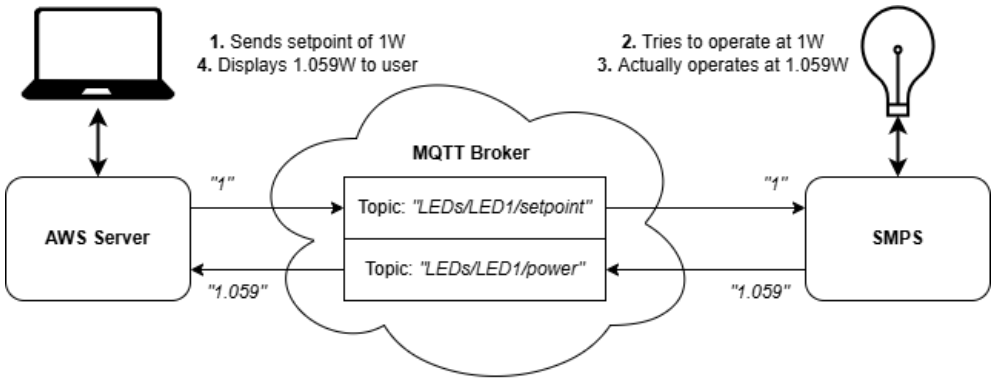

Figure 3.1.1 - Block diagram of MQTT setup with an example operation

3.1.2.1 Project Organisation

All SMPS units need to have MQTT functionality implemented, so this module was only developed once all other hardware modules were mostly complete. That way, only a few lines in each final program can be added or changed to implement MQTT.

3.1.2.2 Option 1 – Mosquitto (Self-Hosted Broker)

Our first option is a self-hosted MQTT broker. We decided to test *Mosquitto*, a free, open-source and lightweight MQTT broker.

3.1.2.3 Option 2 – HiveMQ (Cloud Broker)

Our second option is a remote, external cloud server. We set up a free HiveMQ account. It uses shared servers and has a 10 GB maximum data flow. If these limits become issues, then a paid upgrade is available.

3.1.2.4 Research & Investigation

3.1.2.4.1 Procedure

In order to test both options, we used a basic Python script that ran on a single Pico unit. The script connects to Wi-Fi and to the MQTT broker (either Mosquitto or HiveMQ), and then subscribes to 2 topics. The time taken to connect, and then to subscribe to each topic is measured to the nearest second.

3.1.2.4.2 Results

(With sample size of 15 each)

	Connection Time / s			Subscription Time / s		
	Mean	Maximum	Std. Dev	Mean	Maximum	Std. Dev
Mosquitto	2	6	2.07	0.233	1	0.43
HiveMQ	6.133	30	7.425	0.5	5	1.28

Figure 3.1.2 - Statistics on time taken for initial startup (connection and subscription)

3.1.2.5 Conclusion

As seen in *Figure 3.1.2*, the HiveMQ server has noticeably larger average latency with both connection and subscription compared to Mosquitto. HiveMQ is also inconsistent, with an outlier at 30 seconds.

However, the setup of Mosquitto is not viable from the end user's perspective, since it requires connecting additional hardware such as a laptop which must always run the server. Therefore, despite greater latency, practical considerations led us to choose the HiveMQ cloud server.

3.1.3 Implementation

The first step that all SMPS units must perform is to connect to the MQTT broker using an MQTT client. We used the *umqtt.simple* Python module, which is lightweight and designed for microcontrollers.

We started by implementing MQTT into the LED controller. This only required a small amount of code to be added at startup to connect to the MQTT broker, and then extra code to check if there are new messages within the main program loop (every millisecond).

Each module has different requirements for what data must be sent and received by the SMPS, and this is shown in *Figure 3.1.4*. In each case, only a few lines of code must be added.

3.1.4 Testing

3.1.4.1 Procedure

- Run the control program on 4 LED units
- Record the difference in time (to the nearest millisecond) that an SMPS sends power info, and when it is received by the web server
- Send setpoint updates to all SMPSs simultaneously, measure the difference between earliest and latest update time
- Repeat until enough data is gathered

3.1.4.2 Results

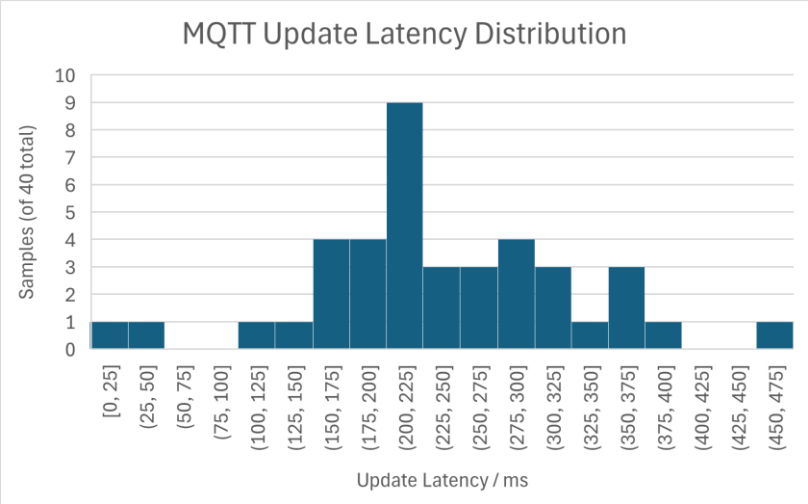

Figure 3.1.3 - Single component setpoint update latency distribution

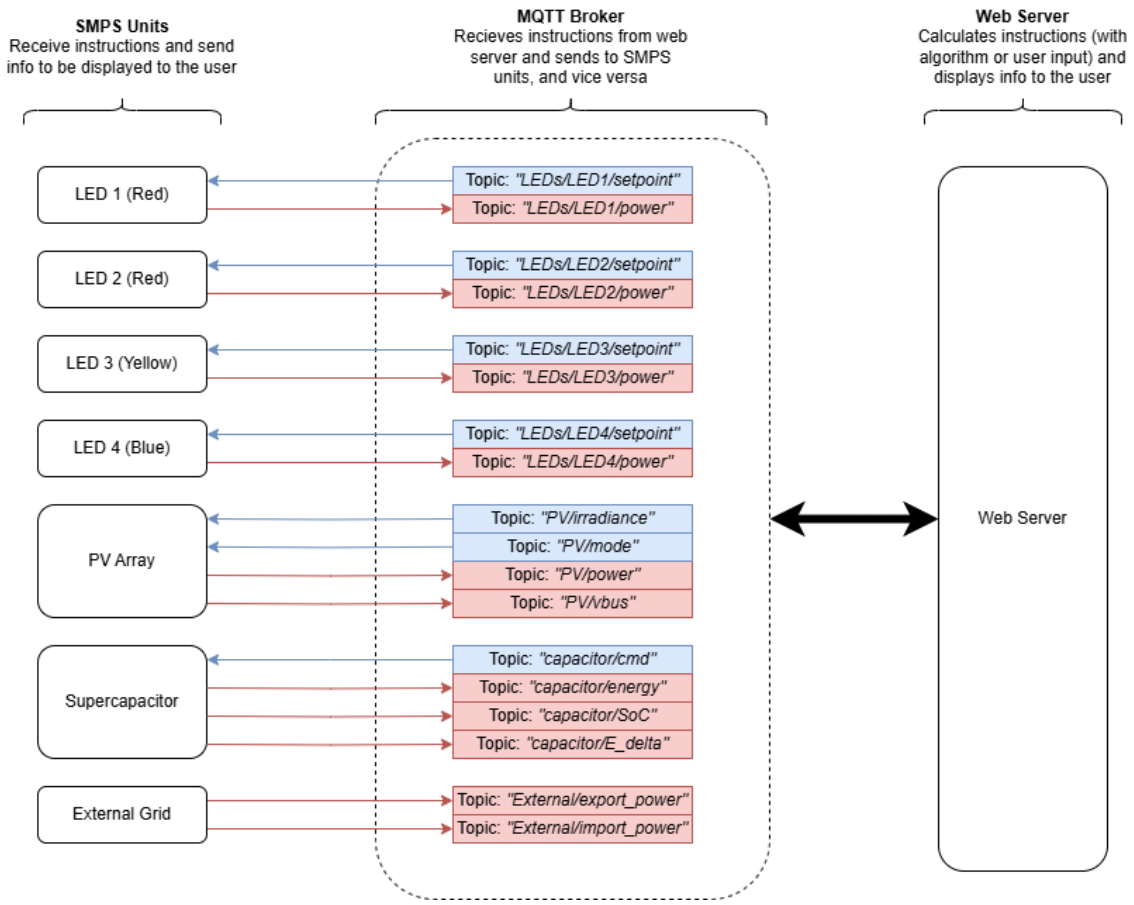

Figure 3.1.4 - Whole system MQTT setup, illustrating how each component is connected

Update Latency / ms			Update Spread / ms		
Mean	Maximum	Std. Dev	Mean	Maximum	Std. Dev
244.6	454.7	81.4	91.7	116.3	24.8

Figure 3.1.5 - Four-component update latency and spread statistics

3.1.5 Evaluation

As required quantitatively, the average latency is significantly less than 500ms, and even including outliers the latency is always below this point.

Setpoint updates always occur within less than 200ms of each other as required.

Data is sent and received as expected with manageable latency that should not impact electrical performance. Therefore, this module meets its specification well.

3.2 Web Server

3.2.1 Subsystem Specification

- The subsystem must provide a user interface (UI) to display real-time and historical system data.
- The subsystem must have user controls to monitor and manage hardware operations.
- The subsystem must be able to communicate with hardware modules.

3.2.2 Design

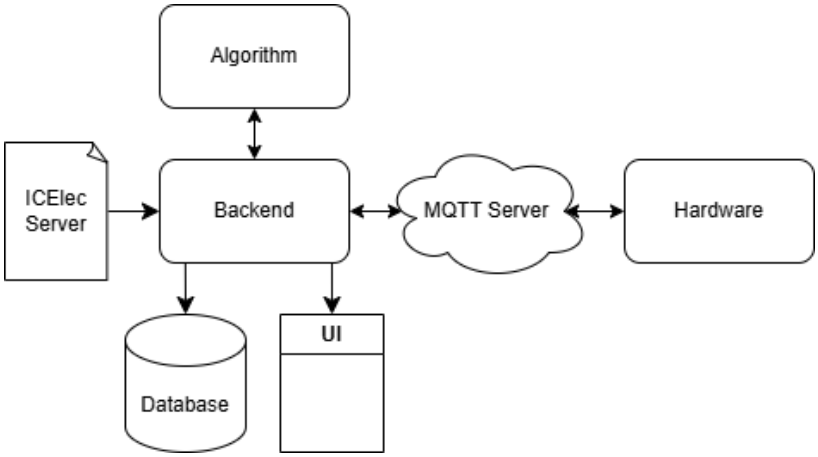

Figure 3.2.1 - Software system block diagram, illustrating web server position

The web server must act as the intermediary between many other components, with relationships as detailed below:

- **Backend:** Processes data to determine optimal decisions and serve as MQTT client.
- **Web Server → Backend:** Sends live irradiance, tick, price, and demand data.
- **Backend ↔ MQTT Server:** Exchanges action commands and hardware responses.
- **MQTT Server ↔ Hardware:** Communicate with Pico W modules and synchronize commands.
- **Backend → Database:** Stores data history and algorithms model prediction output.
- **UI↔ Database:** Retrieves data for real-time and historical visualization.
- **UI → Backend/MQTT:** Allows user's manual control of hardware directly.

3.2.3 Backend

The core of our backend is implemented in a single Flask/Python application (power_server.py). This service exposes API endpoints for dashboard clients (e.g., /status/grid, /status/days, /status/history) and also runs a background thread, which periodically fetches live grid data from ICElec and writes it into SQLite database.

3.2.3.1 Implementation

At startup, power_server.py initializes all algorithm models (LSTM, Transformer, etc.). It also serves as an MQTT client, establishing an MQTT connection to receive and publish hardware commands. All Flask routes and background share a single SQLite file (smartgrid.db), and the app is launched via app.run(...) to listen on port 5000 for HTTP requests.

3.2.3.2 REST API Endpoints

Endpoint	Functionality
/status/days	Return all distinct day values from smart_grid_data ordered by day DESC.
/status/history	Returns tick-based historical data with buy_price, sell_price, and demand.
/status/irradiance	Yields the latest sun irradiance value from smart_grid_data.
/status/deferrables	Returns the entire deferrables table as JSON.
/status/predict	Runs LSTM/ML models on last 60 ticks (from ICElec + our sun data) to produce forecasts.
/status/decision	Runs DQN policy model to simulate energy decision for next 60 ticks.
/status/grid	Bundles the latest day, tick, price_buy, price_sell, sun, demand, pv_power, and cost into one JSON.
/status/control	Returns the latest hardware telemetry and connection status.
/status/device	Fetches the latest LED status and current from device_data.
/controls/override	Allows manual override for LED power via POST.
/controls/capacitor	Receives capacitor commands ("S", "E", "H") via POST.
/controls/mppt	Enables or disables MPPT override and publishes the command.
/update/device	Stores new LED status and current in the database.
/download/history	Downloads tick data (price, demand) for a specific day in plain-text format.
/download/all	Downloads all available historical data in plain-text format.

All endpoints use get_db_connection() to open a new SQLite connection, execute their SELECT statements, then db.close() before returning jsonify(...).

3.2.3.3 Handling MQTT Messaging

```
mqtt_client = mqtt.Client(client_id="server1client", userdata = None, protocol=mqtt.MQTTv5)
mqtt_client.tls_set(tls_version=mqtt.ssl.PROTOCOL_TLS)
mqtt_client.username_pw_set("server1","Password01!")
mqtt_client.on_message = on_hardware_message
mqtt_client.connect("bd7c7a5e1c8e4513ba43dbbb7f288f38.s1.eu.hivemq.cloud", 8883)
mqtt_client.subscribe([
  ("LEDs/LED1/power",0),
  ("LEDs/LED2/power",0),
  ("LEDs/LED3/power",0),
  ("LEDs/LED4/power",0),
  ("PV/power",0),
  ("PV/vbus",0),
  ("capacitor/energy",0),
  ("capacitor/SoC",0),
  ("capacitor/E_delta",0),
  ("External/import_power",0),
  ("External/export_power",0)
])
mqtt_client.loop_start()
```

on_hardware_message(client, userdata, msg) parses msg.topic and msg.payload into floats, updating global hardware_state and last_msg_time. For example, if topic.endswith("LED1/power"), we do hardware_state['led1_power'] = float(payload) and stamp last_msg_time['LED1'].

Publishing Commands

In /controls/override (POST), we receive JSON like {"led3_power": 5.3}; we extract ledKey="led3_power", look up the corresponding MQTT topic "LEDs/LED3/setpoint", and mqtt_client.publish("LEDs/LED3/setpoint", "5.3"). Finally, we publish "sync" with payload "1" so all Picos know a new override is available.

Publishing Irradiance

In fetch_and_store_grid_data(), after fetching real sun_val from ICElec, if mppt_enabled is False we do mqtt_client.publish("PV/irradiance", str(sun_val)); if True we publish 1 at mqtt_client.publish ("PV/mode", str(cmd)), then we set it to MPPT mode and displaying the corresponding max sun irradiance at the control panel (100% (MPPT))

3.2.3.4 Algorithm Connection

To apply the data, we write a software intermedia program in the server to transfer the commands of the model to the actions of the devices.

1. Transfer the storage of the capacitor to the intermedia and the demand, buy price, sell price and PV power and deferrable tasks.

2. Input the sequence data of past 60 ticks, storage and deferable task to the machine learning model.

3. Machine learning model output the buy and sell amount of energy, and deferable demand for the next 60 ticks.

4. The intermedia translate the buy-sell amount to the command to the LED and capacitor.

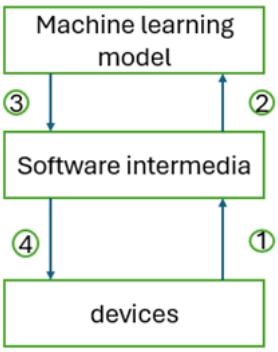

Figure 3.2.2 - Algorithm connection block diagram

3.2.3.5 Full Data Loop

- Fetch ICElec data,
- Running ML/policy to produce control outputs,
- Publishing outputs over MQTT
- Ingest hardware telemetry
- Expose a unified REST API for any UI to query live/historical data.

3.2.4 Database (SQLite)

3.2.4.1 Description

SQLite is used for its simplicity and ease of integration, providing quick and reliable data storage for historical and real-time data required by the frontend UI.

3.2.4.2 Schema

3.2.4.2.1 smart_grid_data

Per-tick energy data (prices, irradiance, demand, PV power).

Column Name	Type	Description
id	INTEGER	Primary key
day	INTEGER	Virtual day index
tick	INTEGER	Tick index (0–59)
timestamp	TEXT	timestamp
sun	REAL	Sun irradiance
price_buy	REAL	Buy price at tick
price_sell	REAL	Sell price at tick
demand	REAL	Instantaneous demand
pv_power	REAL	PV power output

3.2.4.2.2 deferrables

Tasks with flexible energy scheduling.

Column Name	Type	Description
id	INTEGER	Primary key
start_tick	INTEGER	Earliest tick to start
end_tick	INTEGER	Deadline tick to finish
energy	REAL	Required energy (J)
fetched_at	TEXT	When to execute fetch

3.2.4.2.3 yesterday_data

Historical data cache (tick-based).

Column Name	Type	Description
id	INTEGER	Primary key
tick	INTEGER	Tick index
buy_price	REAL	Buy price at tick
sell_price	REAL	Sell price at tick
demand	REAL	Demand at tick
fetched_at	TEXT	Retrieval timestamp

3.2.4.2.4 device_data

Recent hardware telemetry (e.g., LED1).

Column Name	Type	Description
id	INTEGER	Primary key
timestamp	TEXT	Entry timestamp
led1_status	TEXT	ON/OFF status
led1_current	REAL	Current in A

3.2.5 UI

3.2.5.1 Description

UI is separated into 3 sub-panels to serve the three fundamental utilities: Real-time data monitoring, historical data visualization, and hardware control functionalities.

3.2.5.2 Implementation

We aim to provide a clear and concise interface to the user. It uses HTML to structure pages, and JavaScript to fetch and display real-time data from the backend using JSON responses. For styling, we selected Tailwind CSS for its utility-first design approach, which simplifies layout and responsiveness, and DaisyUI to enhance visual consistency with ready-made UI components. This setup allowed us to build a clean, functional interface across three panels: live monitoring, historical review, and clean hardware control.

3.2.6 Dashboard Functionality

3.2.6.1 Live Dashboard (index.html)

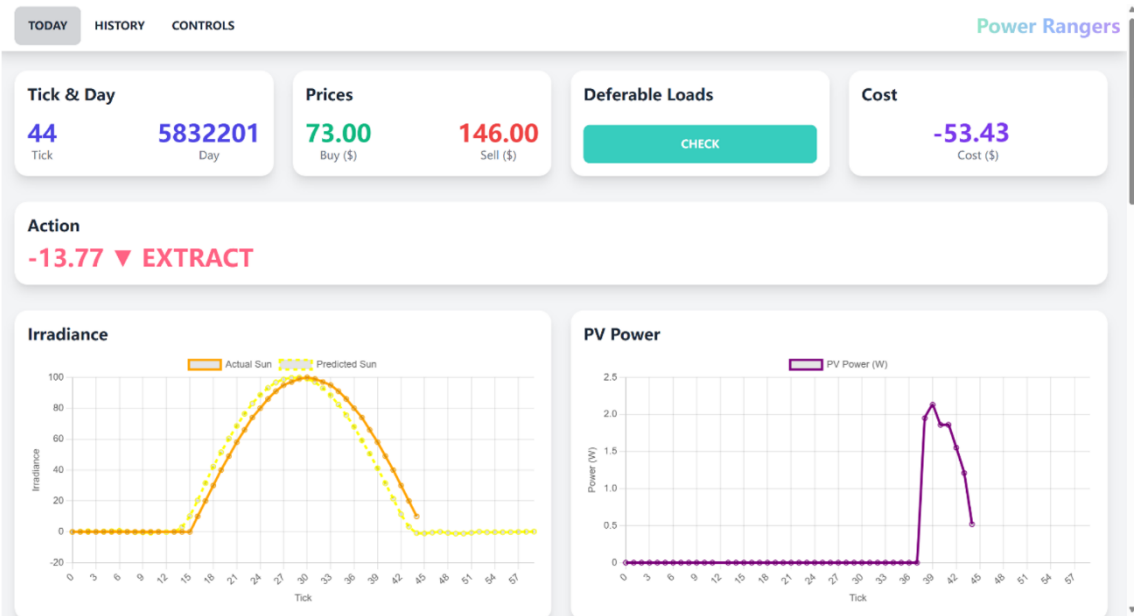

Figure 3.2.3 - Live data dashboard UI

Data Element	Description
Tick	Current time step (0–59)
Day	Current virtual day index
Buy Price	Real-time grid buy price (¢/J)
Sell Price	Real-time grid sell price (¢/J)
Cost	Cumulative energy cost (¢)
Sun Irradiance	Current solar input percentage (%)
PV Power	Output power from PV subsystem (W)
Demand	Instantaneous demand from web server (W)
Deferred Loads	List of deferrable load requirements (start, end, energy)
Forecasts	Predicted values for buy/sell prices, sun, demand (60 ticks)
Decision Energy	Energy decision output from DQN algorithm

3.2.6.2 Historical Page (yesterday.html)

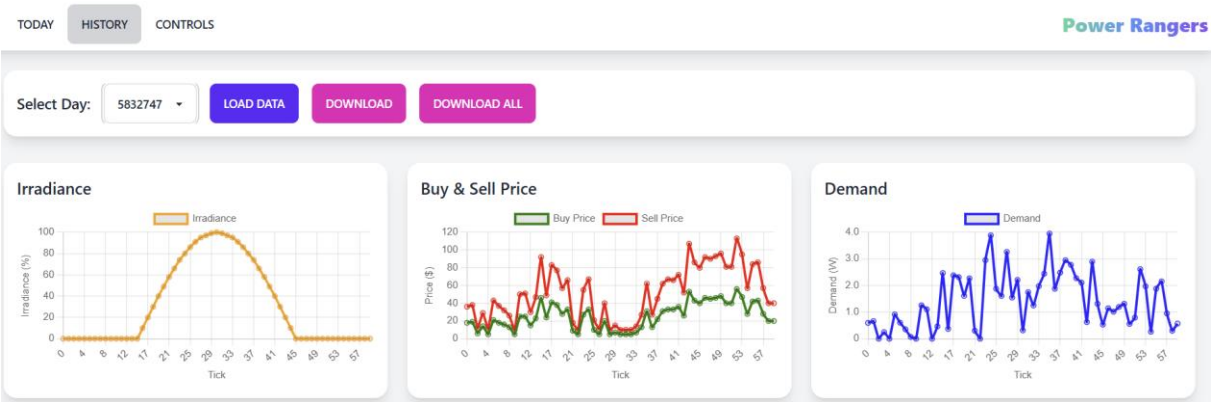

Figure 3.2.4 - Historical data dashboard UI

Data Element	Description
Available Days	Dropdown list of stored day indices
Buy Price History	Past buy prices (tick-wise)
Sell Price History	Past sell prices (tick-wise)
Demand History	Past demand values (tick-wise)
Download Buttons	Export current or all day data in .txt format

3.2.6.3 Controls Page (controls.html)

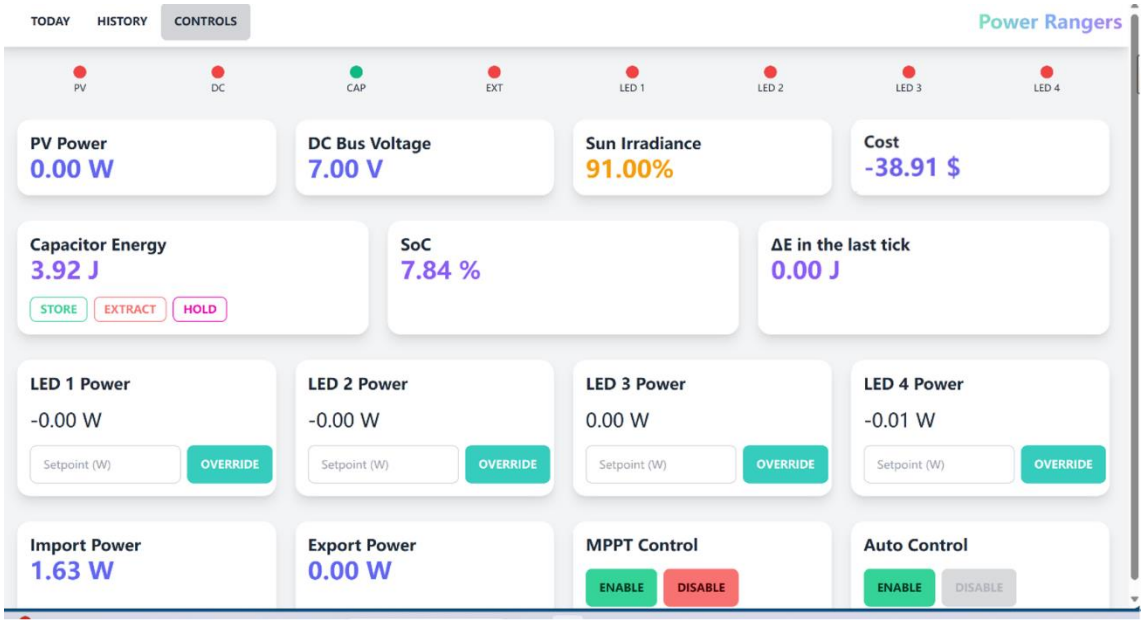

Figure 3.2.5 - Control dashboard UI

Data Element	Description
PV Power	Current PV output (W)
DC Bus Voltage	Bus voltage level (V)
Sun Irradiance	Live irradiance value or MPPT override
Cost	Real-time accumulated energy cost ($)
Capacitor Energy	Energy stored in supercapacitor (J)
SoC	State of charge of supercapacitor (%)
ΔE	Energy delta (change from last tick)
LED1–4 Power	Power drawn by each LED (W)
Import Power	Power imported from grid (W)
Export Power	Power exported to grid (W)
Manual Controls	Store/Extract/Hold capacitor, override LEDs, enable/disable MPPT
Connection Status	Connection indicator (green/red) for each hardware component

3.2.7 Evaluation

The web server successfully provides a user interface that shows the user real-time data (*Figure 3.2.3*), historical data (*Figure 3.2.4*) and controls to monitor and manage hardware (*Figure 3.2.5*). These pages all use high-contrast colours for accessibility, and prioritise being simple and easy to navigate.

The web server can also successfully communicate with hardware modules via MQTT.

Therefore, we conclude that this subsystem meets its specification.

3.3 Machine Learning Algorithm

3.3.1 Subsystem Specification

- The subsystem must attempt to minimise the total cost of imported energy.
- The subsystem must schedule when to satisfy deferrable demands.
- The subsystem must perform better than a naïve algorithm.

3.3.2 Design

The cost minimisation algorithm for the smart grid is divided into three machine learning components:

1. **Prediction**: Forecasts the buy/sell electricity prices, immediate demand, and sun irradiance over the next 60 ticks using a Long Short-Term Memory (LSTM) network.

2. **Scheduler**: Allocates energy to deferrable loads to minimise cost, using a Transformer-based architecture trained with reinforcement learning.

3. **Trader**: Determines the buy/sell quantity of electricity to/from the external grid using a PPO approach with ResNet.

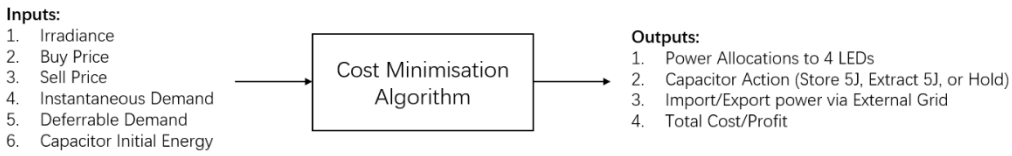

Figure 3.3.1 - Inputs and outputs of cost minimisation algorithm

3.3.3 Stage 1: Prediction

To predict the next 60 ticks of buy price, sell price, instantaneous demand, and solar irradiance, we employ a Long Short-Term Memory (LSTM) model. The network is trained on the past 60-tick sequence data using supervised learning with backward propagation.

3.3.3.1 LSTM Architecture

- The LSTM network mimics human memory through a gated structure, selectively forgetting irrelevant past data and retaining useful context.

- It uses three gates (input, forget, output) and a memory cell to manage the information flow.

- The training data consists of three years of historical records.

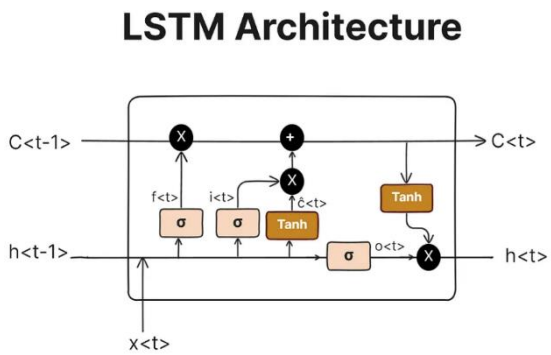

Figure 3.3.2 - Block diagram for LSTM architecture

Source: "Predicting the Future: LSTM vs Transformer for Time Series Modelling", MIT OpenCourse [2]

3.3.3.1.1 Linkage between LSTM and Kalman Filter

Kalman filter is a recursive filter used to denoise and smooth recent historical electricity prices, using a model-based average through prediction and correction. It estimates the underlying trend with reduced error and predicts the next day's prices.

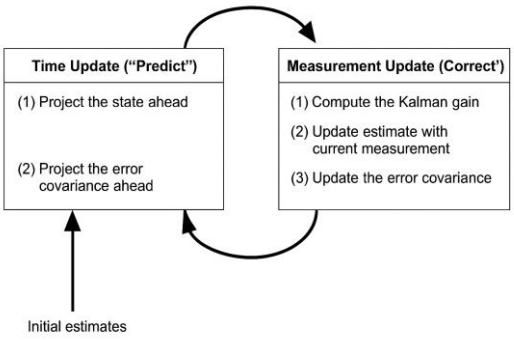

Figure 3.3.3 - Block diagram for Kalman filter

Source: An Introduction to the Kalman Filter, Greg Welch, UNC [3]

Kalman Filter and LSTM networks are both recursive models capable of state estimation and time-series prediction, and they share structural similarities rooted in signal processing and control theory.

At an abstract level, both can be interpreted as two-stage Infinite Impulse Response (IIR) filters:

- Kalman Filter, a linear recursive estimator designed for systems with Gaussian noise. It uses a model-based prediction-correction mechanism to smooth historical data and predict future states.
- In LSTM, is a nonlinear, trainable IIR filter that learns both dynamics and correction logic directly from data.

The recursive dependency in both Kalman Filter and LSTM mirrors the following difference equation, where the current output depends on previous outputs:

$$y[n] = f(y[n-1], y[n-2], \dots, x[n], x[n-1], \dots)$$

While the Kalman Filter remains effective for linear models with Gaussian noise, we decided to use LSTM which offers a superior approach for electricity price prediction due to its:

- Ability to learn complex, nonlinear temporal patterns directly from historical price data.
- Robustness to handling non-Gaussian noise.
- Capacity to utilise a larger dataset of historical data (LSTM predicts future prices using the last 60 ticks, whereas the Kalman Filter typically relies only on the most recent tick).

Thus, LSTM can be seen as a data-driven generalisation of Kalman Filter, capable of adaptively learning both system dynamics and correction logic- better suited to the complexity of electricity price forecasting.

3.3.3.1.2 LSTM Training and Deployment Strategy

1. The model is trained offline using supervised learning.

2. The LSTM effectively removes noise and provides smoothed forecasts (as shown in *Figure 3.3.4*), proving adequate for downstream decision-making.

3. During simulation and training of the subsequent models (Scheduler and Trader), we assume perfect prediction of the next 60 ticks to isolate their performance.

3.3.3.2 Results

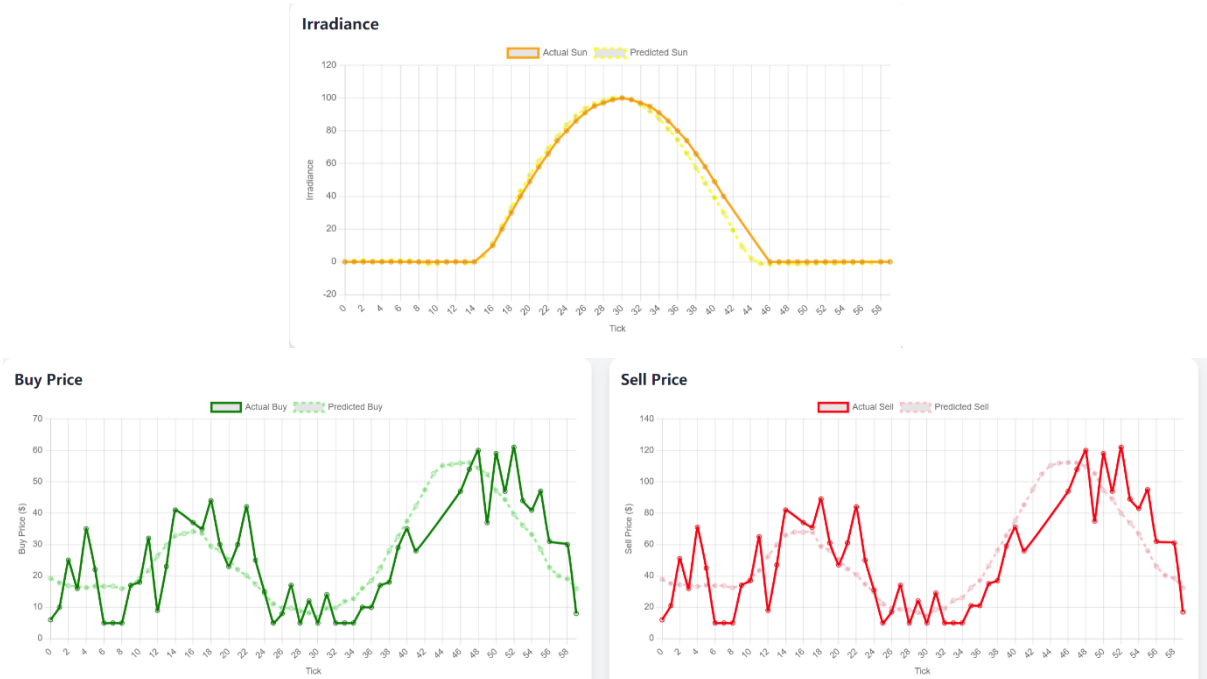

Figure 3.3.4 - Actual and predicted data (generated by LSTM) of irradiance, buy price, and sell price

3.3.3.2.1 Evaluation

As shown in *Figure 3.3.4*, the actual and predicted data values generally match well, with the addition of some inevitable random noise. Therefore, the system works as required.

3.3.4 Prerequisites

3.3.4.1 Energy Logic

Before proceeding to Stage 2 (Scheduling) and Stage 3 (Trading), the core Greedy logic governing both is outlined below. It minimises energy cost by scheduling deferrable loads through a scoring mechanism, while also managing electricity trading with the external grid:

1. **Deferrable energy demands are assigned to the LEDs at the most cost-effective ticks** based on a score that considers instantaneous buy price and task urgency. The energy is allocated greedily to the highest-scoring time slots.

2. **If surplus PV energy is available**, the system stores 5 J in the supercapacitor per tick (subject to capacity). If the supercapacitor is full, surplus energy is exported to the grid immediately.

3. **If the total energy demand exceeds the available PV energy**, the system prioritises extracting 5J from the supercapacitor to cover the shortfall. Any remaining unmet demand is then supplied by importing energy from the external grid.

4. **If buy price is high during the low-demand period and the supercapacitor has stored energy**, the algorithm extracts 5 J per tick from it and exports it to the grid for high profit.

5. **Import/export power cannot be directly controlled**, since the external grid's SMPS interface maintains a fixed 7 V DC bus (it performs PID voltage control instead of power control). Power flow is indirectly controlled by the scheduler through store/extract actions on the supercapacitor.

3.3.4.2 Self-Learning Controllers

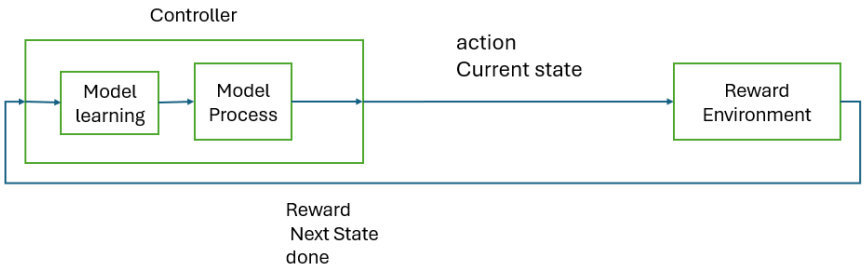

Figure 3.3.5 - Basic model of trader and scheduler from a control engineering perspective

The trader and scheduler act as self-learning controllers. They take the current state, choose an action (supercapacitor and LED commands), and receive feedback (reward, next state, done) from the environment. Using this feedback, the model improves itself through PPO training to make better decisions over time.

3.3.4.2.1 Mathematical description of optimisation process (PPO) for Scheduler and Trader

Model structure

- **Policy head** outputs action probabilities for decision-making.
- **Value head** predicts the expected return of a state- both branching from a shared neural network in actor-critic methods.

Here is the description of training process:

- **Action advantage A_t** tells the agent how much better an action was compared to what it expected.
- **Probability ratio $r_t(\theta)$** compares how likely the new policy thinks an action is, versus the old one.
- We take the smaller of:
  - The straightforward reward improvement,
  - And a "clipped" version that prevents the update from being too large.

$$L^{\text{CLIP}}(\theta) = E_t\left[\min\left(r_t(\theta)\widehat{A_t},\ \text{clip}(r_t(\theta), 1-\epsilon, 1+\epsilon)\widehat{A_t}\right)\right] \ (1)$$

$$A_t = \delta_t + (\gamma\lambda)\delta_{t+1} + (\gamma\lambda)^2\delta_{t+2} + \cdots \ (2)$$

$$where\ \delta_t = r_t + \gamma V(s_{t+1}) - V(s_t) \ (3)$$

The advantage function tells us **how much better an action is compared to average behaviour**. We use a method called **Generalised Advantage Estimation (GAE)**, which blends rewards from the future with a "decay factor" to emphasize nearer rewards more. Value network loss L_{value} is computed as MSE between predicted value and the target return R_t, where R_t is the sum of immediate reward and next-state value.

The value network learns to **predict how much reward** an agent can expect from any state.

We train it using **Mean Squared Error (MSE)**—comparing its prediction with the actual return the agent received:

$$L_{\text{value}} = \frac{1}{N}\sum_t (V_\theta(s_t) - R_t)^2$$

Where $V_\theta(s_t)$: Value predicted by the critic, R_t: Actual return or bootstrapped reward target (GAE).

3.3.4.2.2 Pre-Training

Sometimes, when the machine learning model is too large, directly placing it into the reinforcement learning environment can cause it to lose direction. This often results in the loss failing to converge.

To address this issue, we first design a **heuristic policy** and perform supervised training based on it. Then, we place the model into the reinforcement learning environment, where the loss is able to converge successfully.

3.3.5 Stage 2: Scheduling

The scheduling module is responsible for allocating energy to deferrable demands over the next 60 ticks to minimise total cost, while satisfying time constraints. It computes the energy schedules based on the following inputs:

- **Sequence Data for last 60 ticks**: buy price, sell price, sun irradiation, and demand.
- **Deferrable Tasks**: for each deferrable load, a 61-dimensional vector is provided- the first element denotes energy requirement, the remaining 60 indicate availability (1 = available, 0 = unavailable).

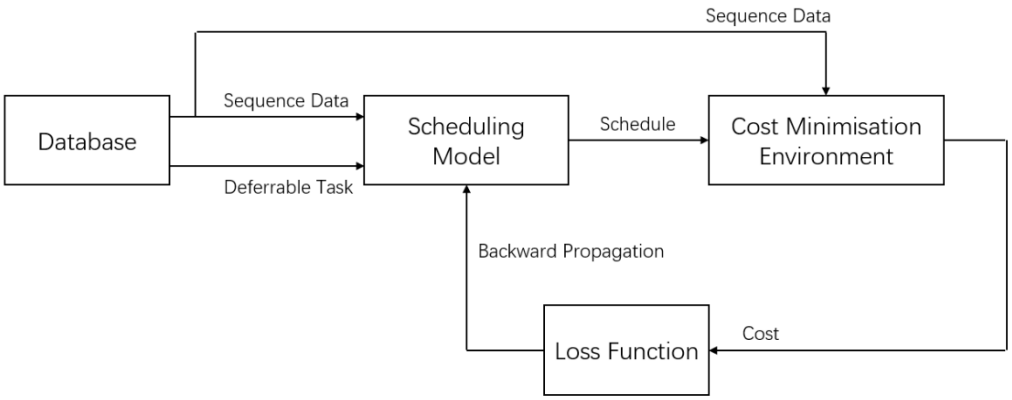

Figure 3.3.6 - Block diagram of Scheduler including inputs
Note: The cost minimisation environment is the trained trader model

3.3.5.1 Structure

1. **Self-Attention:** It computes dot products between input vectors to determine the attention weights, using learned Query (Q), Key (K), and Value (V) matrices. This reveals how each task interacts with others- When task availability masks are orthogonal, dot products become zero, meaning no interaction, indicating the tasks do not overlap in time domain. Stronger overlap leads to higher attention and greater coordination between tasks. All operations are matrix-based and efficiently parallelised on GPU.

2. **Position Encoding:** To preserve input order, we apply learnable position embeddings to each vector before processing. Though theoretically optional, position encoding improves practical performance- since later tasks are input later in the sequence, the model benefits from a clear sense of timing. This ensures the Transformer can differentiate task order and improves learning stability.

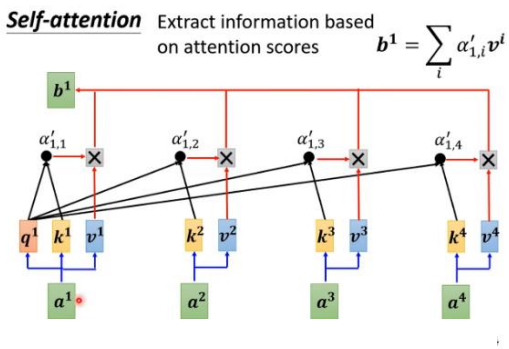

Figure 3.3.7 - Self-attention matrix operations

3. **Transformer Block:** Each Transformer block contains a multihead self-attention layer followed by a feedforward layer, both wrapped with normalisation and shortcut connections (this architecture was inspired by ResNet). Deferrable tasks and sequence data are first position encoded separately by two Transformers, then fused through a cross-attention layer before transferring to a ResNet-style decision head to make final decision.

3.3.5.2 Results

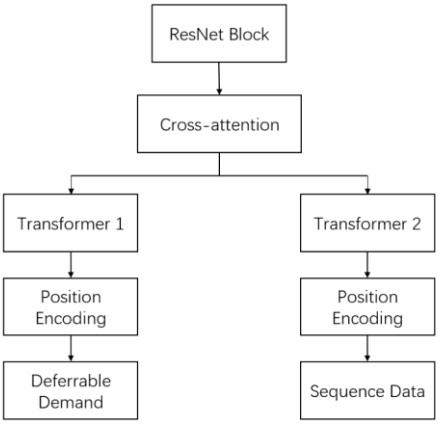

Figure 3.3.8 - Main structure of Transformer-based Scheduler

3.3.5.2.1 Pre-Training

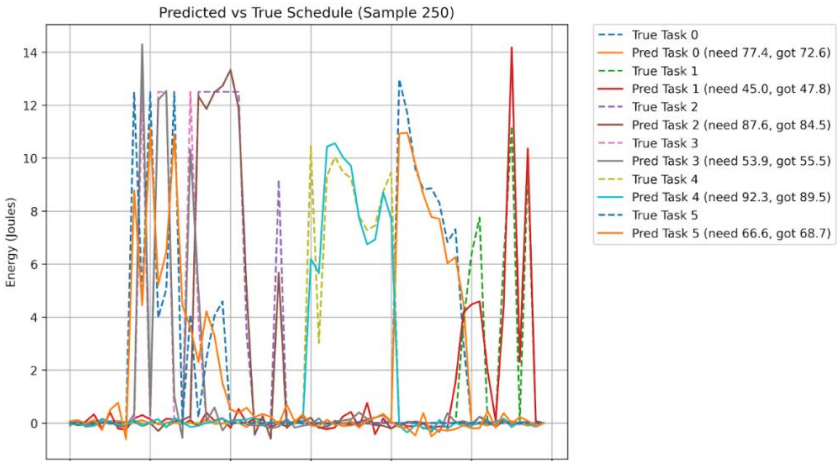

Figure 3.3.9 - Pre-training results: Predicted schedules (of deferrable demands) vs true schedules

As shown in *Figure 3.3.9*, the predicted schedules of deferrable demands closely match the reference schedules across six tasks. All prediction errors are under 5%, confirming that the model has successfully performed the scheduling logic.

3.3.5.2.2 Reinforcement learning

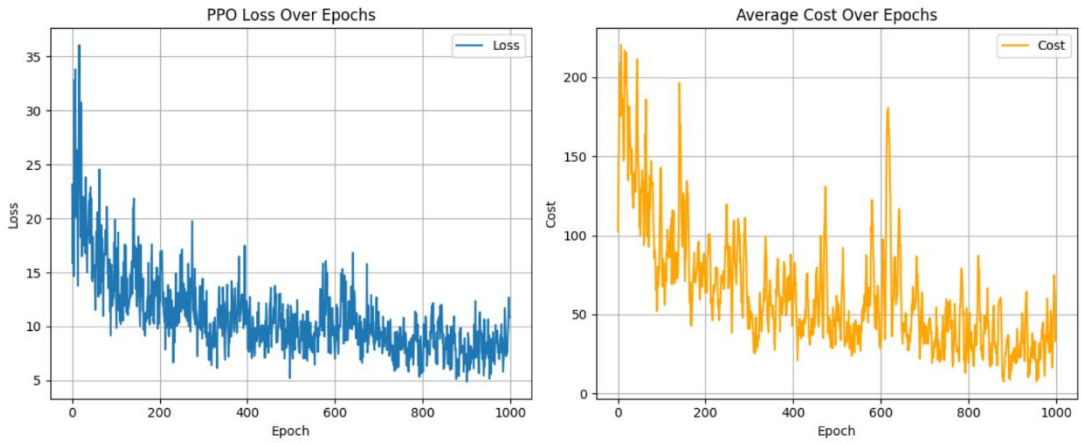

Figure 3.3.10 - PPO training results of Loss and Cost over epochs (training iterations)

Training results in *Figure 3.3.10* show effective convergence, with loss function dropping steadily and cost decreases (from ~150 to ~30) across epochs. This indicates the model improves reliably as PPO updates the policy toward more cost-effective schedules.

3.3.5.2.3 Evaluation

Final results (*Figure 3.3.11*) show an average energy error of 2.58 J (11.45%). PPO slightly outperforms the pre-trained model by trading off strict energy fulfilment for cost savings. In some cases, the model prefers to delay or shift energy usage to the next few ticks if the cost is too high at the target tick. This trade-off can be tuned via the energy deviation weight in the cost function.

```
✅ Average absolute energy error per task: 2.58 J
✅ Average percentage energy error: 11.45 %

--- Sample 1 ---
Task 1: Required=26.7J | Scheduled=27.0J | Error=0.31J | Error%=1.15%
Task 2: Required=46.2J | Scheduled=43.4J | Error=2.83J | Error%=6.13%
Task 3: Required=43.8J | Scheduled=46.4J | Error=2.61J | Error%=5.96%
Task 4: Required=27.5J | Scheduled=28.6J | Error=1.16J | Error%=4.23%
Task 5: Required=27.3J | Scheduled=28.4J | Error=1.04J | Error%=3.80%
Task 6: Required=26.0J | Scheduled=24.8J | Error=1.12J | Error%=4.31%
--- Sample 2 ---
Task 1: Required=36.3J | Scheduled=32.9J | Error=3.36J | Error%=9.26%
Task 2: Required=32.3J | Scheduled=34.1J | Error=1.82J | Error%=5.63%
Task 3: Required=12.3J | Scheduled=15.4J | Error=3.16J | Error%=25.82%
Task 4: Required=16.2J | Scheduled=18.4J | Error=2.29J | Error%=14.16%
Task 5: Required=35.4J | Scheduled=40.1J | Error=4.71J | Error%=13.31%
Task 6: Required=34.1J | Scheduled=26.7J | Error=7.38J | Error%=21.66%
--- Sample 3 ---
Task 1: Required=9.5J | Scheduled=12.4J | Error=2.93J | Error%=30.88%
Task 2: Required=25.8J | Scheduled=24.2J | Error=1.64J | Error%=6.34%
Task 3: Required=26.8J | Scheduled=26.6J | Error=0.15J | Error%=0.55%
Task 4: Required=5.2J | Scheduled=9.1J | Error=3.94J | Error%=76.52%
Task 5: Required=17.4J | Scheduled=16.2J | Error=1.15J | Error%=6.60%
```

Figure 3.3.11 - Error between scheduled and predicted energy

3.3.6 Stage 3: Trader

The trader module optimises electricity trading with the external grid by controlling supercapacitor actions (Store/Extract). The cost-minimisation environment is first set up, as shown in the flowchart in *Figure 3.3.12*. Additionally, a price threshold strategy is used for supervised pretraining:

- Sell when the predicted sell price exceeds a set threshold
- Buy when the predicted buy price falls below a lower threshold.

This strategy exploits price fluctuations to shift energy between the supercapacitor and grid economically.

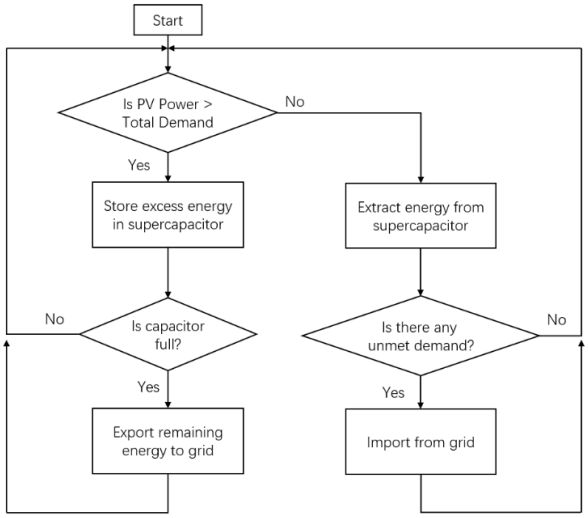

Figure 3.3.12 - Flowchart showing trader logic

3.3.6.1 Option 1 – Naïve Algorithm

The naïve algorithm is a strategy that disables supercapacitor actions, so that it always trades with the grid. It is an option; however we require that our system has better performance than this.

3.3.6.2 Option 2 – Price Threshold Strategy

The price threshold strategy decides when to charge of discharge the supercapacitor based on a threshold of energy buy/sell price.

The plot in *Figure 3.3.13* compares the two strategies across 100 evaluations of 30-day episodes.

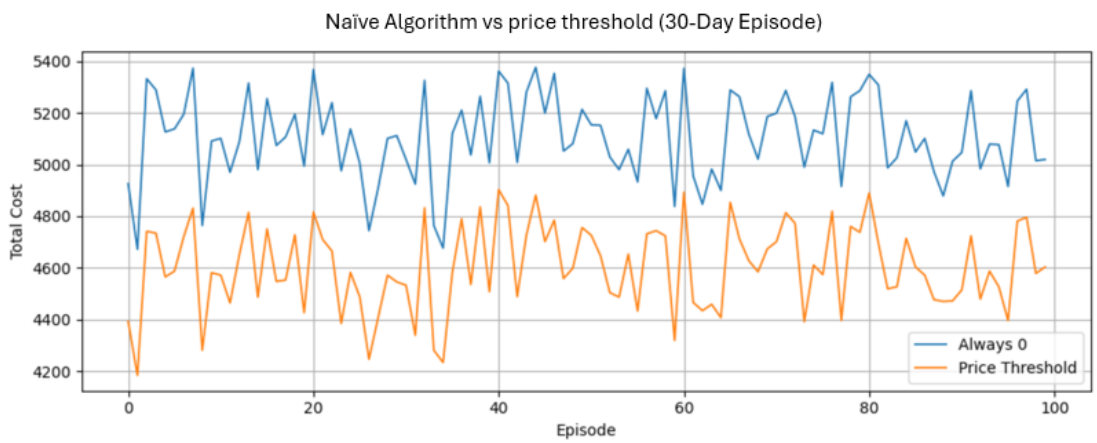

Figure 3.3.13 - Cost comparison between 2 trader logics: Price Threshold (Orange), Naive (Blue)

The results show a clear improvement with price threshold control: the average total cost decreases by approximately 12%, from $52.00 to $46.00.

3.3.6.3 Option 3 – PPO Model

Then we design a PPO model like this for the trader:

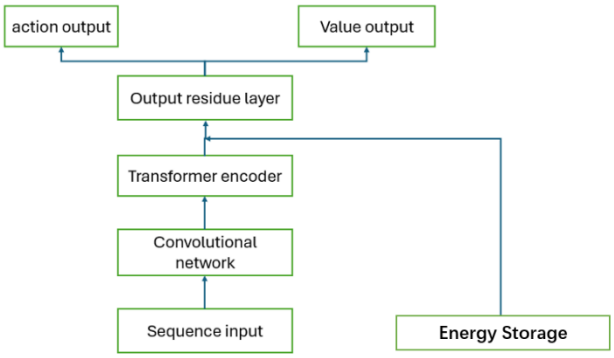

Figure 3.3.14 - PPO model block diagram

The convolutional network itself works like a noise filter here that allows the transformer to work in an environment with significant noise.

3.3.6.4 Conclusion

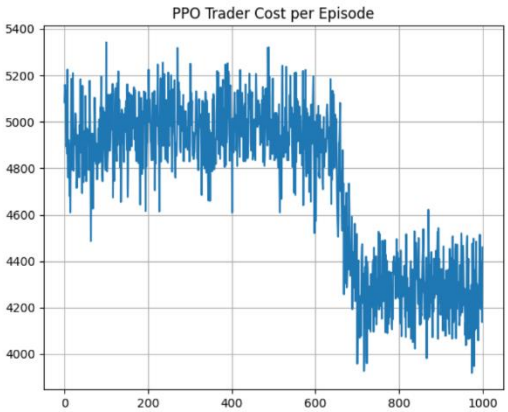

Figure 3.3.15 - PPO trader cost per day

As shown in *Figure 3.3.15*, the use of PPO achieves a final cost of around $**43.00** after 1000 episodes of training, a further decrease of $3.00 compared to the price threshold strategy, showing that the PPO-based third strategy not only improves upon the man-made rule but also develops its own trading logic beyond the naïve and threshold approaches.

3.3.7 Conclusion

The core idea behind both the scheduler and trader modules is simple yet widely applicable to optimisation problems- including smart grid control and CPU scheduling. The strategy follows a three-step process:

1. Design a rule-based algorithm (e.g., greedy or threshold-based) as a baseline.
2. Generate the target output for the supervised pretrain for the model and let it learn the strategy
3. Place the model in a reinforcement learning environment and let it optimise the cost autonomously.

Now that this has been concluded, we can see that the algorithm works well.

As shown in *Figure 3.3.15*, the PPO algorithm successfully optimises power cost so that the system performs better than the naïve algorithm by ≈20%, as required. Deferrable demands are also successfully scheduled and completely satisfied by the end of each day, as shown in *Figure 3.3.11*.

Therefore, we conclude that this subsystem meets its requirements.

4 Testing and Evaluation of the Integrated System

To ensure correct and stable operation of the smart grid system, a comprehensive testing procedure was performed on the UI dashboard, with two control modes: (1) manual control via UI, and (2) autonomous optimisation via cost-minimisation algorithm.

4.1 System Startup Procedure

To prevent instability that may drive the PSU into current limiting mode, a strict startup procedure must be followed prior to any testing or control of the smart grid system. Since the initial capacitor voltage (during startup) at port A of SMPS is lower than the 7V bus voltage at port B, directly connecting the supercapacitor causes a transient response: the external grid PSU voltage drops from 12 V to 6 V, then recovers and stabilises at 12 V within 5 seconds. To reduce this instability, the correct startup sequence is to:

1. First connect the external grid and PV emulator to establish and stabilise the bus voltage at 7 V.
2. Then connect the supercapacitor to the bus.

This procedure ensures smooth voltage regulation and prevents PSU undervoltage conditions.

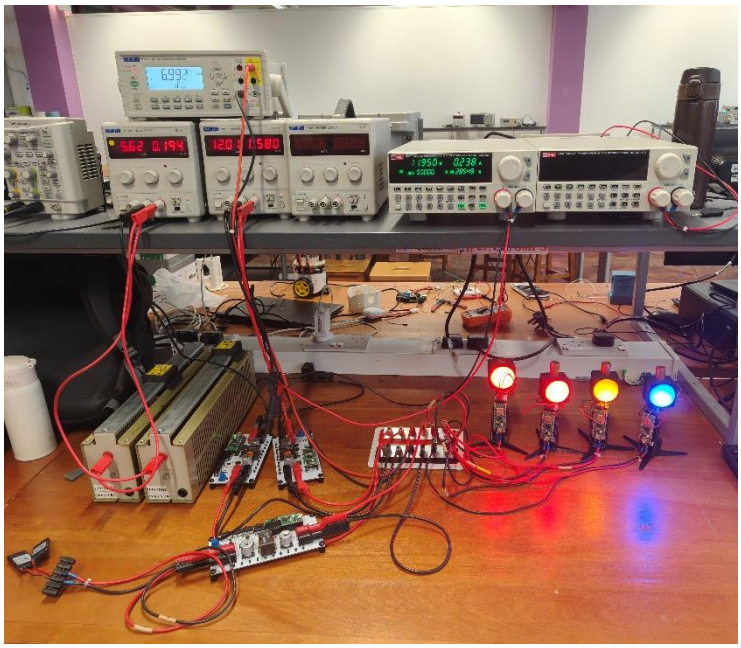

Figure 4.1.1 - Complete Smart Grid Configuration

4.2 Control Mode 1: Manual Control via UI

In manual control mode, all hardware components can be controlled via the web-based UI:

- **PV Control**: The user can toggle between PID power control and MPPT. When MPPT is enabled, the system rapidly converges to the maximum power point (at 100% irradiance) of 3.2 W within 0.2 seconds and maintains steady output at 3.2W. When MPPT is disabled, new irradiance levels (0–100) are sent to the PV emulator-interfaced SMPS every 5 seconds. PV power output is scaled and adjusted accordingly.
- **LED Power Control**: Users can manually assign power targets (e.g., 0.2 W) to each of the four LEDs.
- **Supercapacitor Control**: The user can issue discrete energy commands—**Store**, **Extract**, or **Hold**—with each Store/Extract transferring 5 J of energy.

Figure 4.2.1 - PV power corresponding to varying irradiance levels (MPPT disabled)

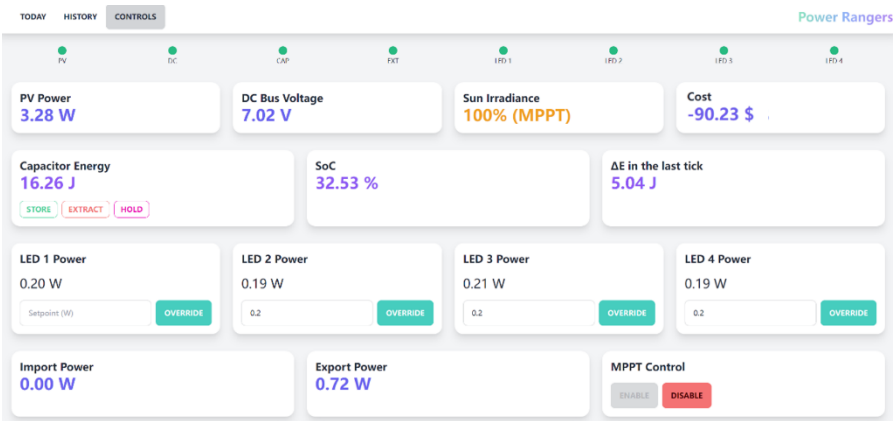

Figure 4.2.2 - Manual control panel for supercapacitor and LEDs (MPPT enabled in this case: 3.2W)

During testing, commands issued via the UI are executed by the hardware with minimal error and latency, demonstrating reliable hardware–software communication.

4.3 Control Mode 2: Auto Control with Cost-minimisation Algorithm

In automatic mode, the system uses a greedy machine learning algorithm to minimise operational cost by scheduling LED power and controlling energy storage.

4.3.1 Load Scheduling

Deferrable LED loads are automatically shifted to ticks with either high PV output or low electricity import cost. The total demand is the sum of the instantaneous demand at the current tick and the deferrable demand.

This total is then divided equally among the four LEDs by dividing it by 4. In cases where total power is above 4W (the maximum power of the LEDs) then the surplus demand is carried over to the next tick until it is fully satisfied. This is shown in *Figure 4.3.1*.

Figure 4.3.1 - Allocation of 3 deferrable demand tasks

4.3.2 Energy Storage Strategy

The supercapacitor actions are automatically derived from the energy decision graph (*Figure 4.3.2*), where blue bars indicate energy storage and red bars indicate energy extraction.

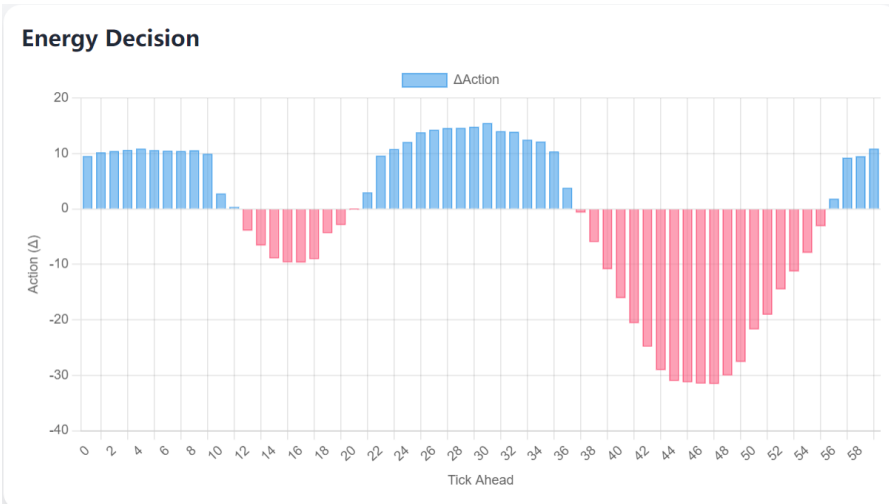

Figure 4.3.2 - Energy decision graph for automatic control of supercapacitor

- **Ticks 22–36 (noon period):** High solar irradiance results in peak PV generation. The surplus PV energy is stored in the supercapacitor.
- **Ticks 0–10 (midnight):** Due to low electricity prices during this period, the system purchases energy from the grid and stores it for later use.
- **Red regions overall:** These indicate periods where PV generation is insufficient to meet demand. In such cases, the system first extracts energy from the supercapacitor, and only then resorts to importing additional energy from the grid.

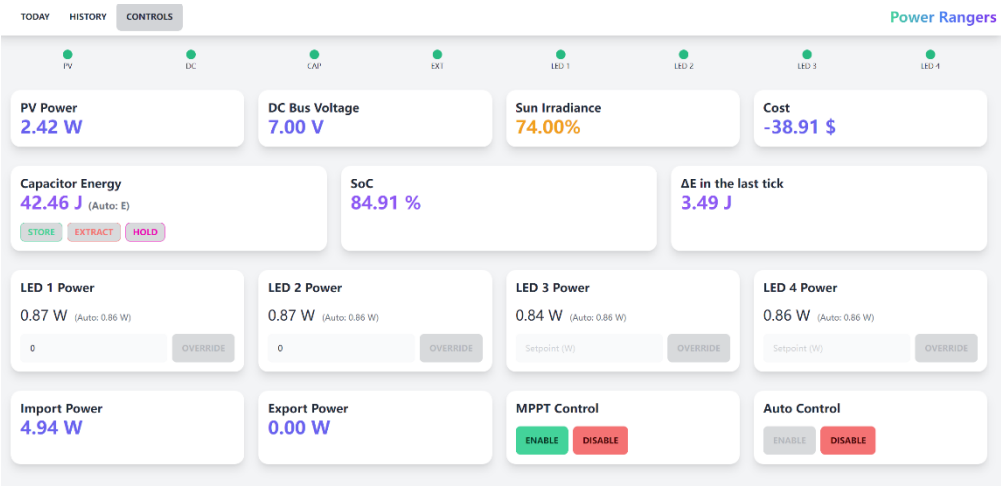

Figure 4.3.3 - Switch to Auto-Control: supercapacitor action and LED power shown in Auto mode

4.4 Evaluation

4.4.1 Stakeholder Requirements

We can use our testing to verify that the initial stakeholder requirements (reiterated in *Figure 4.4.1*) are met. Testing confirms that:

- The load LEDs fully satisfy the immediate demand given by the server (requirement 1, see *Figure 2.3.8*)
- The PV emulator consistently responds to irradiance inputs with correct scaled power output (requirement 2, see *Figure 2.1.13*).
- MPPT converges to the true MPP efficiently and reliably (requirement 2, see *Figure 2.1.11*).
- The supercapacitor can store energy when there is excess, and discharge in response to commands (requirement 3, see *Figure 2.2.8*)
- The control algorithms effectively defer demand and control supercapacitor store/extract actions to reduce daily energy cost (requirements 4 & 5, see *Figure 4.3.1*, *Figure 4.3.2*, *Figure 3.3.10*, *Figure 3.3.15*).
- The DC bus voltage remains at a constant 7V with only minor transients within ±5% (requirement 5, see *Figure 2.4.4*)
- The user interface displays current and historic information (requirement 6, see *Figure 4.2.2*, *Figure 3.2.4*)

Requirements

1. The system shall provide energy to load LEDs to satisfy the demands given by a third-party web server.
2. The system shall extract energy from a bench power supply (PSU) set up to emulate the voltage-current characteristic of a PV array.
   i. The configuration of the PSU shall be determined by characterising a supplied PV array
   ii. The current and/or voltage of the PSU shall be manually modulated to emulate the effect of the day/night cycle
   iii. The system shall use a switch-mode power supply (SMPS) with variable duty cycle to maximise the energy extracted from the emulated PV array
3. The system shall store excess energy in a provided supercapacitor for use at a later time
   i. No batteries shall be used
4. A mismatch between supply and demand of power shall be accommodated by importing from or exporting to an external grid, which is emulated by a PSU with an energy sink.
   i. The energy imported or exported shall be metered and converted to a monetary value using variable prices specified by a third-party web server
5. The system shall minimise the overall cost of importing energy with an algorithm to decide when to store/release and import/export energy. It shall also choose when to satisfy demands that can be deferred
   i. The algorithm should perform better than a naive algorithm that always acts to minimise the amount of power imported or exported at a given moment, and delivers demands as soon as they are requested.
6. There shall be a user interface that displays current and historic information about energy flows and stores in the system.

Figure 4.4.1 - Initial stakeholder requirements

Despite an average daily loss of approximately $100, attributable to the fact that average PV power is lower than the average LED demand, the cost is minimised relative to uncontrolled scenarios. Importantly, supply-demand balance is satisfied and grid stability is maintained throughout the operation.

5 Conclusion

This project successfully emulates a household photovoltaic (PV) electricity system connected to an external grid, aiming to meet the demand of LED loads while minimising electricity costs. A user-friendly UI dashboard enables real-time visualisation of predicted and actual data of the smart grid, with two control modes for manual override or automated optimisation to minimise cost. The developed system demonstrates robust and stable performance under testing and meets all design goals.

Real-world relevance is clearly reflected in the system's energy flow and economic outcomes. Typically, solar energy has a low load factor caused by weather dependency, and in this project's setup, it corresponds to lower average PV power compared to LED demand, thus the system relies heavily on grid imports, resulting in consistent monetary losses. Daily loss is estimated at around $100, which aligns with real-world household PV economics where break-even typically occurs after 15 years-longer than the average PV panel lifespan of 10 years. Furthermore, the ≈80% efficiency of SMPS modules introduces internal power losses, reducing the effective utilisation of PV power.

5.1 Potential Improvements

To enhance system performance, future projects could explore advanced control logics beyond PID control, such as Fuzzy Logic Control (FLC) and Model Predictive Control (MPC), to better handle dynamic constraints in hardware such as supercapacitor. Additionally, subtracting the power loss inside SMPS and Raspberry Pi from the import power could lead to more accurate energy accounting and economic evaluation.

6 Appendix: Hardware Components

1. Lab SMPS

Specification	Value	Unit
Circuit Type	Buck or Boost or Bidirectional	-
Controller Type	Raspberry Pi Pico W	-
PWM Frequency	100	kHz
Port A Voltage	0 - 17.5 ($V_A > V_B$)	V
Port B Voltage	0 - 17.5	V
Max Current	5	A
Efficiency	80% (average)	-

2. LED Driver SMPS

Specification	Value	Unit
Circuit Type	Buck or Boost	-
Controller Type	Raspberry Pi Pico W	-
PWM Frequency	100	kHz
Input Voltage	7 - 17.5	V
Output Voltage	0 - 17.5	V
Max Current	~500	mA

3. PV Panels (4 of them connected in parallel)

Specification	Value	Unit
V_{OC}	5	V
I_{SC}	230	mA

4. Supercapacitor

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

7 Reference

[1] Electrical Power Engineering Lecture 3: PV Interface, Tim Green, Imperial College London.

[2] Predicting the Future: LSTM vs Transformer for Time Series Modelling, MIT OpenCourseWare.

[3] An Introduction to Kalman Filter, Greg Welch, University of North Carolina at Chapel Hill.

[4] The Transformer, Daniel Jurafsky & James H. Martin, Stanford University.

[5] PPO: Local Optimization Achieves Global Optimality in Multi-Agent Reinforcement Learning, Yulai Zhao, Department of ECE, Princeton University.