Imperial College
London

Imperial College London

Department of Electrical and Electronic Engineering

Group 17: Smart Grid Report

June 2024

Anson Chin CID: 02194736	Samuel Khoo CID: 0226491	Ilan Iwumbwe CID: 02211662
Eddie Moualek CID: 02029589	Justin Lam CID: 02210573	Lucas Lasanta CID: 02236286

With special thanks to Doctor Philip Clemow and Doctor Edward Stott

Contents

1 Abstract

In the evolving landscape of energy management, integrating renewable energy sources with advanced grid systems presents a significant opportunity to enhance efficiency and sustainability. Our project aims to develop a smart grid system that utilizes the bi-directional quality of a Switch Mode Power Supply (SMPS) to control the flow of energy, while leveraging machine learning and optimization algorithms to predict and manage energy trading with an external grid. This system will incorporate an internal solar array to harness renewable energy and make real-time decisions on energy storage, usage, selling, and buying. The smart grid model will simulate a single day, with each minute represented by a 5-minute interval, enabling detailed analysis and optimization of energy flows. The construction of the smart grid involves the integration of both hardware and software, as it will be demonstrated through a physical circuit which is controlled by the decisions made by the algorithm.

This project not only addresses the current challenges of energy management but also sets the stage for future advancements in smart grid technologies. By implementing this model, we aim to demonstrate the potential for intelligent, data-driven energy systems to revolutionize how energy is produced, distributed, and consumed.

2 System Specification and Design

2.1 Concept and Initiation

Project Definition: build an energy management system for connecting a home to a smart grid. The system must balance energy supply and demand and use forecasting to minimize the cost of imported energy from the grid.

Project Apparatus: A photo-voltaic array will be used to convert solar energy into electrical energy for the system, while a mechanical flywheel will be provided for energy storage through the conservation of angular momentum, along with two 0.25F super-capacitors. The grid itself will be mimicked by a Power Supply Unit (PSU), which has a set internal voltage and current limiter. The operation of supplying energy to the home is mimicked by connecting the smart grid to 3 LED loads. Additionally, in this project, each group is given 3 bi-directional and 3 Buck Switch Mode Power Supplies (SMPS), where the Buck SMPSs are attached directly to each LED load. Each SMPS comes with a Raspberry Pi Pico. Lastly, a 6-port central bus voltage is provided to make parallel connections within the circuit. The key to creating a successful smart grid is to utilize the given SMPSs to control and balance the flow of energy throughout the circuit.

Project Specification:

1. The PV array input should be determined through Maximum Power Point Tracking (MPPT)
2. The system should be able to import and export energy to the grid
3. The smart grid should supply energy to the LED loads based on its total demand, through the PV array and the PSU grid.
4. The circuit should use the bi-directional and Buck SMPSs provided, along with the 6-port central bus and a flywheel/capacitor for energy storage
5. The charging and discharging of energy in the flywheel/capacitor should be controlled.
6. The micro-controllers within each SMPS should be used to receive, and if necessary, send data to a laptop. The data received should then be used to control the operation of the circuit.
7. The total demand, solar energy generated, naive algorithms profit, optimized algorithms profit and storage level should be displayed in a website.
8. The design should specify the budget requirements of the project.

2.2 Assumptions

- The PV array will always be at its maximum power point, using MPPT.
- There is negligible energy loss in PV array and will be assumed to be purely dependent on solar irradiance.
- The energy in storage doesn't degrade over time.
- If API call to 3rd party webserver fails, assume last ticks data values applies.
- Base demand should never exceed 4 Watts.

2.3 Experimental Methodology

MPPT input of PV array: The energy input from the PV array is dependent on the solar irradiance, which causes its power to vary significantly. However, given the assumption that the full smart grid circuit is operated when the PV array is always at its maximum power point, a PSU can be used instead with its voltage and current limiters set at the maximum power point. This makes testing more convenient and reliable as the limiters on the PSU always ensure that the voltage and current drawn from it will never exceed its limits.

Flywheel as two 0.25F super-capacitors: Due to technical issues with the flywheel, it was not provided during the project design stage. Therefore, two 0.25F super-capacitors were used instead to create the energy storage function. By connecting them in parallel in a breadboard, a total of 0.5F capacitance is obtained. And according to the data sheet, its maximum potential is approximately 14V, which means the maximum energy that it can store is $50J$, from the equation $E = \frac{1}{2}CV^2$. This breadboard also has a cable port connection with the potential and ground, which is used to connect with the ports of the SMPSs.
Use of laboratory equipment: Throughout the project, especially from the hardware perspective, oscilloscopes and multi-meters will be frequently used to monitor the overall state of the circuit and its individual components. For instance, the multi-meter bench will be constantly connected to the bus voltage in order to ensure that it is operating at its desired voltage. On the other hand, the oscilloscope is more versatile and can be connected with the various pins on the SMPS or on the breadboard to measure the charging and discharging state of the super-capacitors.

2.4 Development from Initial to Final Design

The top-level design is arguably the most important section to decide early on during the project timeline as all of the hardware and software implementations revolve around this design. This section will cover what our initial designs were, and how it was translated the project specification requirements into our final design.

2.4.1 Hardware Design Choices

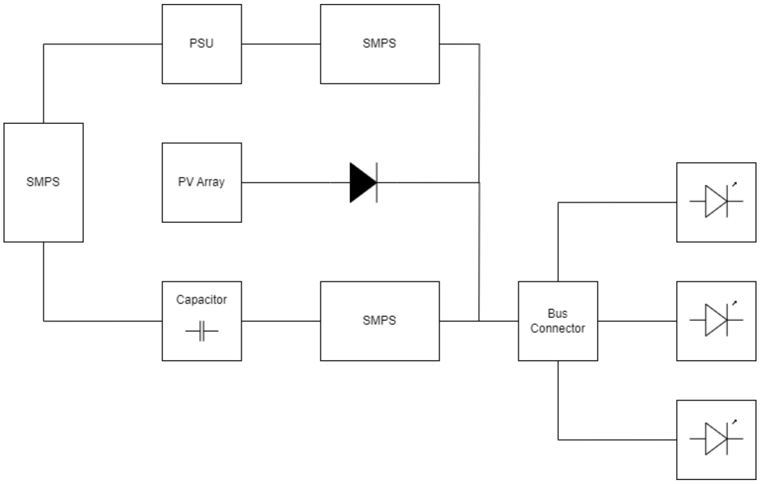


Figure 1: Initial Top-Level Circuit Design

During the initial stages of planning, the group members in charge of the hardware wanted to connect the PSU from the grid directly to the capacitor flywheel with an intermediate bi-directional SMPS. This is then connected to the bus voltage, in parallel with our PV cell input via another bi-directional SMPS, shown in the Figure 1. Although this design seemed quite straightforward and it met all of the project requirements of importing and exporting energy, there were a few problems that came up during our brainstorming process, which limited the practicality of our design. Firstly, when the flywheel is connected to both the PSU and the bus voltage, it is difficult for maintain, or even define the node voltage of the capacitor. This is because there is no common ground in the node, which means the system does not have a defined reference point and therefore any potential difference with that node does not exist. Secondly, the ability to control the charging and discharging of the capacitor becomes very difficult, particularly when having to synchronize the operation of both SMPSs in a precise manner. This is also due to the fact that both of the SMPSs are bi-directional.

Therefore, in order to design a system that can meet all of the hardware specifications, while preventing the aforementioned design issues, the group decided to arrange all of the components in parallel to the central bus voltage at 7V, as shown in Figure 2 below. It was established within the group that each SMPS should only be used for one specific function. Subsequently, each SMPS should only be assigned in series with one component of the circuit. It is important to mention, for future reference, that the bi-directional SMPS on the PSU acts as a voltage regulator while the bi-directional SMPS on the capacitor controls the current flow of the entire circuit.

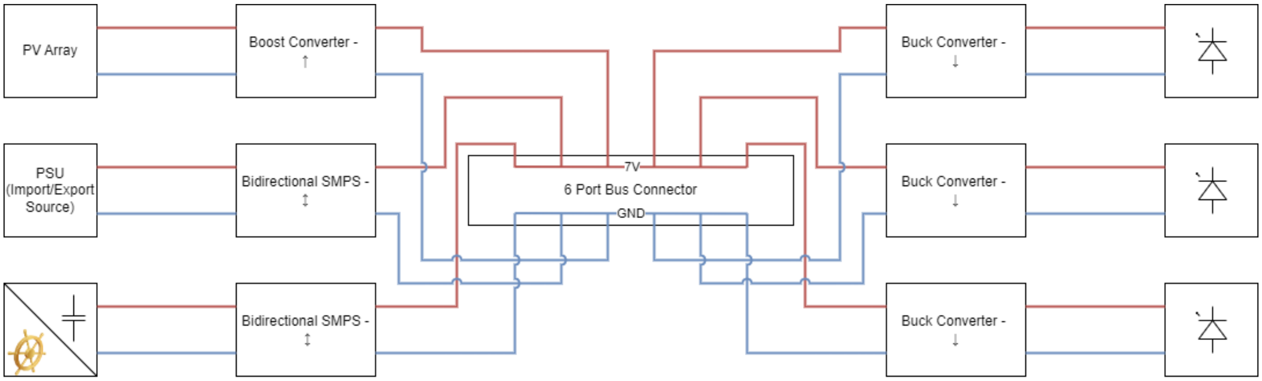


Figure 2: Final Top-Level Circuit Design

2.4.2 Software Design Choices

Although most of the design choices will be covered in future sections, the primary high level design choice that was decided to make, was to host the webserver and the buy-sell algorithm on an external laptop instead of hosting within the micro-controller. This was due to several reasons since the ML model required large amounts of local storage, and the optimization algorithm and webserver needed a lot of memory to run; hence was worried about *overworking* our microcontroller. Additionally, hosting it on our laptop allows the computations to be completed in a shorter amount of time and removes the limitation of using a simple small-sized front-end UI.

On the side of storing data and fetching data from the cloud, the decision was made to use API `Posts` and `Get` functionality to push and pull from the cloud. Data would primarily be stored in 2 separate tables for rapid access with each correlating to a different page within our web application. When it came to decided a cloud service, AWS was chosen due to the abundance of AWS documentation available, as well as the credit available to use provided by the department. Other services considered included Heroku, Google Cloud and Azure, however, these were either too cost-inefficient to use or lacked the features which AWS offered. With regards to features and services within AWS, DynamoDB was chosen due to its scalability compared to hosting S3 buckets or AWS RDS and AWS Aurora. The ease of managing DynamoDB and features like Time To Live (TTL) were also core factors which led to the use of DynamoDB as our database provider.

2.4.3 Hardware and Software Integrated Design

After finalizing the hardware and software top-level designs separately, a full schematic was designed to demonstrate the integration between hardware and software shown below. The main component which acts as the connection between the two is through TCP (Tranmission Control Protocol), with the laptop acting as the host and the micro-controllers acting as the clients. This will be further elaborated below in Section 6.

2.5 Final Design

After planning out our subsystems, here is our simplified final diagram for how our system will operate.

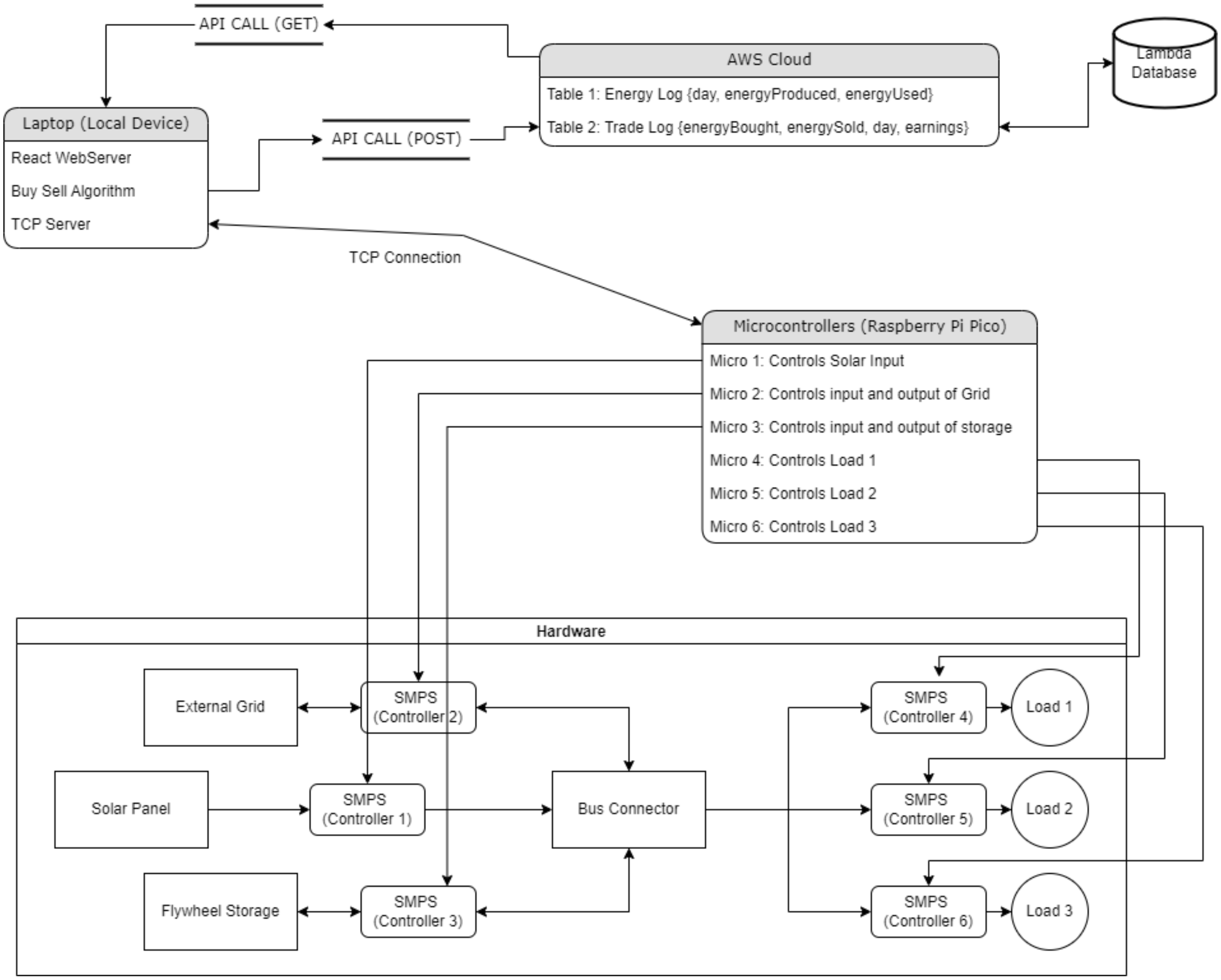


Figure 3: High Level Schematic Diagram of System including Design Choices

3 Project Management

3.1 Planning Methods and Team Communication

Various channels of communications were used to maximize work efficiency, with each of the channels serving a specific purpose. The following are some examples of the aforesaid channels and their respective functions:

- Notion - for general planning and organisation
- Github - for code sharing and explanations
- WhatsApp - for every day quick communication
- Microsoft teams - for group video meetings

Although team members often worked together in the laboratory for the majority of the week, we preferred to organize weekly catch-up meetings to ensure that all members of the team were on track and on the same page. As we have mainly divided tasks between EEE and EIE members, these weekly meetings have turned out to be extremely useful for making final design and data communication decisions. This follows our group principles of working together as a team, taking into account all the different opinions and evaluating ideas. Outside of our working times in the laboratory, and during the weekends, we were able to communicate effectively using WhatsApp due to its convenience.

3.2 Task Assignment

In our first meetings, each member was assigned a specific task or area within the project that they should focus on, which is clearly shown in the figure below. This was to ensure that everyone had a clear responsibility and was in charge of a part of the project. This was a very effective strategy since everyone was keen in working on a wider range of tasks. This meant that most of the work was done collaboratively and everyone knew, to a certain degree, on the technical details of each aspect of the project. At the same time, everyone always had one person to go to for any confirmation or questions, for each individually aspect of the project.

Additionally, the members in charge of the hardware had an even more collaborative approach where the group members tackled tasks and challenges on the circuit together. This was because during the integration process which involved combining the entire circuit together, it was more effective to have multiple people cross-checking the hardware setup and analyzing the operation of the circuit.

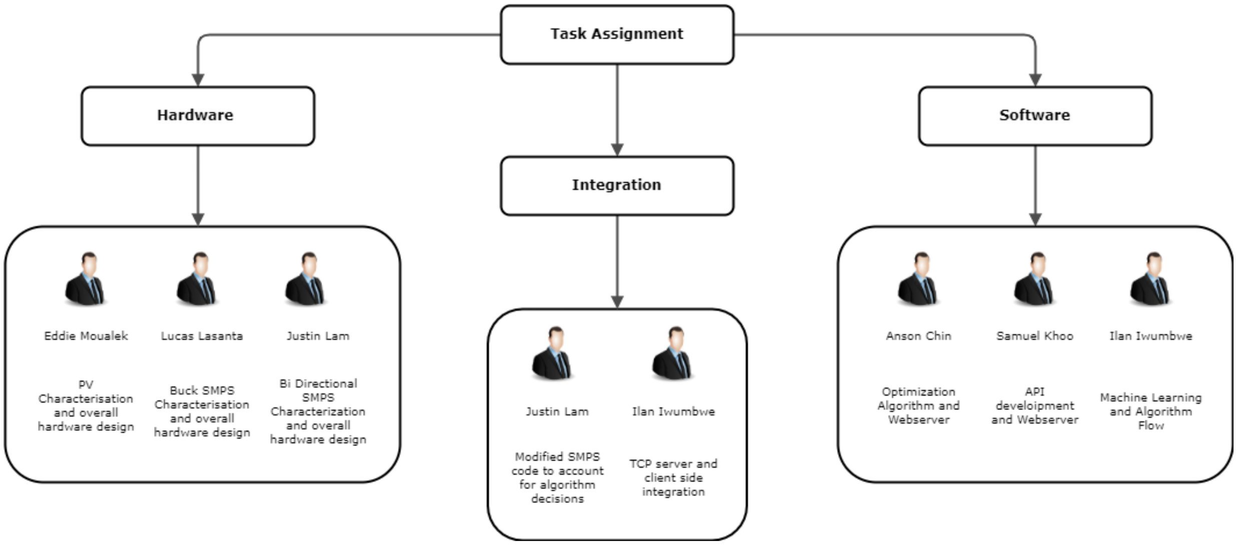

Task Assignment
Hardware
Integration
Software
Eddie Moualek
PV Characterisation and overall hardware design
Lucas Lasanta
Buck SMPS Characterisation and overall hardware design
Justin Lam
Bi Directional SMPS Characterization and overall hardware design
Justin Lam
Modified SMPS code to account for algorithm decisions
Ilan Iwumbwe
TCP server and client side integration
Anson Chin
Optimization Algorithm and Webserver
Samuel Khoo
API develoipment and Webserver
Ilan Iwumbwe
Machine Learning and Algorithm Flow

Figure 4: Overall task allocation for each member

3.3 Time Management

As mentioned in Section 3.1, the various methods of communication used within the team allowed us to discuss ideas and manage our responsibilities. In particular, it was found that creating a shared calendar in Notion with our meeting times and upcoming major deadlines was very helpful with our time management. Additionally, a shared Gantt Chart has been made in Google Sheets during our first few meetings, which was our more reliable planning system. This effectively decomposed the large project into smaller tasks and responsibilities, which allowed everyone in the group to have a clearer understanding on how much time they have for each task, therefore reinforcing better time management. As it was accessible by every member, the overall progress of the team was carefully monitored and regularly modified when there are any changes to our plans.

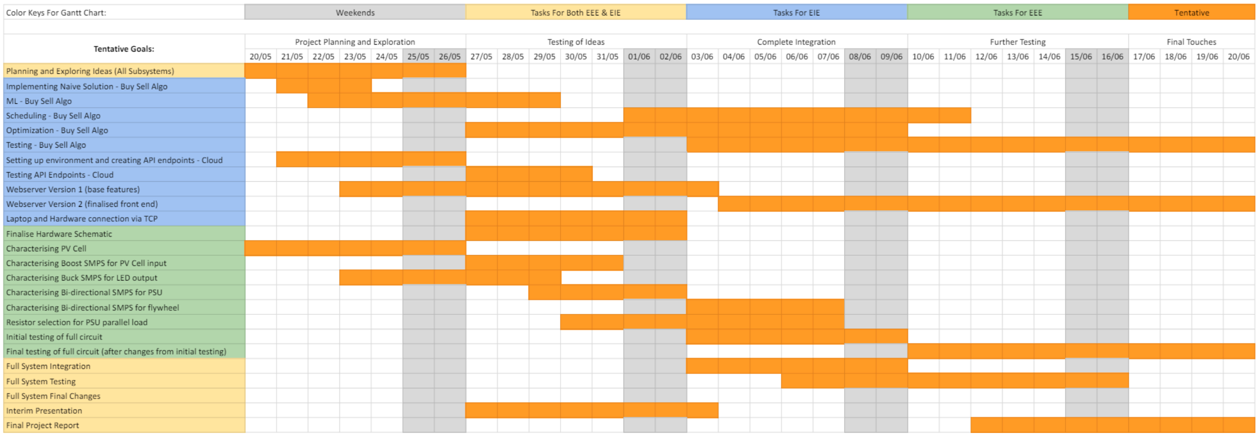


Figure 5: Gantt Chart

4 Hardware Design and Implementation

4.1 Hardware Specifications

Although the overall specifications of the project have already been mentioned in Section 3.1, the members in charge of the hardware wanted to go more in depth on the hardware specifications to obtain better understanding on how to design our circuit, which are shown as follows:

- The amount of current flowing in and out of the PSU should be controlled and monitored to mimick the buying and selling operation. This means at least one bi-directional SMPS must be used with a current reference controller.
- The bus voltage has to be kept constant during the buying and selling operation, and also the charging and discharging of the capacitor flywheel, which acts as an energy storage system. Hence, it is essential that one of the SMPSs acts as the voltage regulator.
- The power from the PV cell should always be prioritized on the loads rather than the PSU. There were uncertainties on how to achieve this, specifically on the balance of current, when there are multiple current sources flowing in and out of the bus voltage at the same time. However, as the PV cell only acts as an input, it was certain that a uni-directional Buck or Boost closed loop controller is required.
- The LED loads should be able to change how much power it draws from the circuit based on the data received from the main laptop. This implies that modification on the Buck SMPS on each LED load is required, particularly on parameters like duty cycle or output PWM. Nevertheless, the combined power drawn from the LED loads does not need to exceed 4W.
- All SMPSs will be ran using Thonny by cable connection, from the laptop containing the appropriate code to the Raspberry Pi Pico.

4.2 Integrating the PV array

The PV (Photovoltaic) array sources energy through light, based in the photoelectric effect. To properly integrate the PV array into the subsystem, for both use and emulation, the following steps need to be followed:

- Characterise the PV array
- Perform Max Power Point (MPP) identification
- Set up a power supply for emulation during agile development

Each subsequent step has been split into two sections; the why and the how.

4.2.1 IV Characterisation of the PV Cell

Characterisation of the PV array is crucial as it allows the identification of the maximum power point of operation, which will be used when designing the rest of the system, and more specifically, the control system used to maintain a voltage at the output concerning the safe operating limits of the SMPS.

The initial methodology revolved around using an SMPS to perform the characterisation. However, an issue was encountered where the SMPS was sourcing current and voltage from the USB connection, skewing the results. To rectify this, an ideal mathematical relation is used, and adapted to fit some of the successful data points obtained using the SMPS. This was done to follow the principles of agile development, where a crude initial design was to be assembled to test the modular stages of development. The mathematical relation between the voltage and current of an ideal PV array is defined as:

$$I = I_{SC} - I_O \left(e^{\frac{V+IR_S}{nV_t}} - 1 \right) \tag{1}$$

$$I_O = I_{SC} e^{\frac{V_{OC}}{nV_t}} \tag{2}$$

$$V_t = \frac{KT}{q} \tag{3}$$

I_{SC} and V_{OC} refer to the short-circuit current and open-circuit voltage, with values $0.23 Amps$ and $5 Volts$ respectively and are parameters made available through the datasheet. n and R_s are defined as the ideality factor, between 1 and 2, and the series resistance. Both are tuned to match existing/correctly recorded data points from the SMPS experiment. Due to the parallel connection of the PV cells, the PV array maintains a terminal voltage of $5V$, but through *Kirchoffs Current Law*, has a terminal current maximum of $0.92A$

4.2.2 Maximum Power Point Identification

The collected data set can then be processed in Excel to identify the maximum power point to enable PSU emulation. This was crucial as development was performed in the lab, and sunlight in both the lab and the UK as a whole are not reliable. The maximum power point is found by recording the voltages and currents at each point, finding the power and recording the maximum. The table and resulting maximum power point are included in the appendix. From here, a series and parallel resistance to make the emulation more realistic can be found using:

$$R_P = \frac{V_O}{I_S - I_M} \tag{4}$$

$$R_S = \frac{V_O - V_M}{I_M} \tag{5}$$

A summary of the key parameters are included below:

Parameter	**Value**	**Units**	**Description**
V_{OC}	5	V	Open-circuit Voltage
I_{SC}	0.92	A	Short-circuit current
I_M	0.829	A	MPP Current
V_M	4.875	V	MPP Voltage
R_S	0.15	Ω	Series Resistance
R_P	54.9	Ω	Parallel Resistance

Table 1: Key Parameters for PV Characterisation and Integration

4.2.3 PSU Emulation

There were 3 options for PSU emulation, each with advantages and drawbacks.

- Using V_{OC} and I_{SC}
- Using V_M and I_M
- Using V_M and I_M with R_S and R_P

The end choice was to use the first option; V_{OC} and I_{SC}, for a few key reasons:

- There was a limited selection of resistors that were useful for this task, and the ones provided had more pressing needs than being used for emulation.
- The group felt that due to the estimative nature of the initial characterisation, it was better to purely use data from the datasheet.
- The group did not want to encounter any power issues due to losses in long cables or connections, so setting a slightly higher voltage and current than V_M and I_M likely meant that the terminal connection was close to those values regardless.

- In reality, having the circuit operating at the current limit of the PV array may be dangerous, as it could continuously become saturated. Having a slightly higher current limit for the PSU emulation ensures the system remains in a safe current range, without continuously tripping the PSU.

As a result, the parameters for the PSU were chosen as shown in Table 2. These parameters will act as the maximum power point for the PV array by ensuring that the selected duty cycle for the SMPS is drawing the maximum power from the PSU.

Parameter	Value	Units	Description
V_S	5	V	Voltage supply
I_S	0.92	A	Current supplied and limiter

Table 2: PSU Emulation Parameters

4.3 Characterization of the SMPSs

After confirming the top-level circuit design, the group decided to separate the circuit into its subsystems and testing them independently, before integrating all of it methodically into one whole system. The following are the subsystems which will be independently characterized:

- PV array with the Boost SMPS
- PSU grid with the first bi-directional SMPS
- Capacitor with the second bi-directional SMPS
- 3 parallel LED loads with individual Buck SMPSs

During the independent testing phase, each SMPS within the subsystems were characterized to its desired function, while being connected to the bus voltage and the LED loads. The bus voltage, in particular, was constantly monitored by a digital multimeter. This was to make sure that the circuit was operating correctly when various changes were made to the subsystems. The details and intricacies will be explained in the subsections below.

In the beginning, the group wanted to set the central bus voltage to 10V. However, during the characterization of the Boost closed-loop SMPS, it was established that setting it to a lower voltage was better for the gain of the Boost. In other words, it was a safer decision to boost the PV array from 5V to 7V, rather than from 5V to 10V.

Nevertheless, a 10V bus voltage was not even possible, when the group tried to set the reference voltage at 10V. This was most likely due to the original design of the closed-loop controller, which seemed to stabilize more effectively around 7V. The choice of this bus voltage will also be further justified in Section 4.4.1.

4.3.1 For PV Array Subsystem

During the characterization of the PV array, the maximum power point obtained from Section 4.2.2 was then emulated by a regular PSU. This then allows the Boost SMPS to increase the voltage and match the 7V bus voltage. As mentioned earlier, since the PV Cell always acts as a supply, the SMPS does not need to be bi-directional and therefore simplifies it as a uni-directional Boost.

By referring to the hardware design specifications, a closed-loop controller was implemented to satisfy the requirement of keeping the bus voltage constant at 7V. This was accomplished by first adjusting the reference voltage to 7V while removing the function of the variable resistor which controls the duty cycle. As such, the error can be calculated by the difference between the reference voltage and the voltage in the output port (V_a). This error is then used to stabilize the output voltage through the Proportional-Integral (PI) controller. The PI controller includes important parameters that deal with properties such as overshoot and oscillations in voltage, and therefore should only be tuned near the end of the characterization process when all the subsystems are successfully integrated.

4.3.2 For PSU Subsystem

In this subsystem, the SMPS is in charge of regulating the bus voltage at 7V, as mentioned in the final top-level design. The group collectively agreed that the grid PSU was the most appropriate choice as a voltage regulator because it has a flexible supply which allows SMPS to vary the power drawn from it, based on the error between the actual voltage and 7V. Also, the PSU itself has a voltage and current limiter, which guarantees the safety of the circuit.

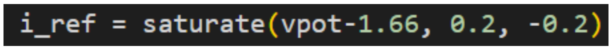
Additionally, the SMPS for the PSU has to be bi-directional to account for current flowing in and out, to import and export energy. During the testing process, it was confirmed that the current flowing out of the PSU should be an import or buying operation, while the current flowing into the PSU is an export or selling operation. This concept will also be further explained in Section 4.4.2.

Lastly, there will always be a resistor of around 40Ω which is connected in parallel to the grid PSU as it acts as a current sink, due to the fact that the PSU itself does not act as one. Slide resistors are used in this project as they contain a much higher power rating.

4.3.3 For Capacitor/Flywheel Subsystem

Similarly, after referring with the hardware design specifications, the group proceeded on utilizing a bi-directional SMPS with a current reference controller, to control the buying and selling operation of the whole circuit.

During the initial stages of testing, before integrating with the software, the current was referenced by the variable resistor on the SMPS board. The potential through the variable resistor is represented by the variable ”*vpot*” within the micro-controller code, which ranges from 0V to 3.3V. Using this variable, the current can be determined by the function shown in Figure 6, where $vpot = 0$ would give a -0.2A current, while $vpot = 3.3$ would give 0.2A. A maximum magnitude of 0.2A was chosen to ensure safety when testing out the circuit and can be easily adjusted within the micro-controller code if necessary. Setting the direction of the current is extremely important as it determines the entire operation of the circuit. The full operation of the circuit will be further explained in Section 4.4.2.



```
i_ref = saturate(vpot-1.66, 0.2, -0.2)
```

Figure 6: Current Reference Code

The buying and selling operations are directly determined by the direction of current through the SMPS, or whether the capacitor is charging or discharging. For instance, when the capacitor is charging, current will be flowing from the bus voltage into the capacitor, causing the bus voltage to momentarily drop. However, due to the voltage regulator SMPS, it will immediately supply more power to the bus voltage to compensate for its deficit and maintain it at 7V, therefore resulting in the buying operation.

4.3.4 Buck SMPSs for LED loads

The final hardware specification which should be accounted for is the adjustable power usage in the LED loads. After looking at the LED Driver Schematic, the group found that the LED Driver had a shunt resistance of 5 identical resistances in parallel between V_{ret} and ground. Since each individual resistor is $5.1\ \Omega$, its parallel resistance drops to exactly $1.02\ \Omega$. Then, using Ohm's Law, the current I_L flowing into the LED load can be obtained. Therefore, the actual power drawn from the LED load can be calculated using $P = I_L \times V_{out}$, where V_{out} is the actual voltage drawn by the load. The values of V_{ret} and V_{out} were conveniently shown in the output of Thonny.

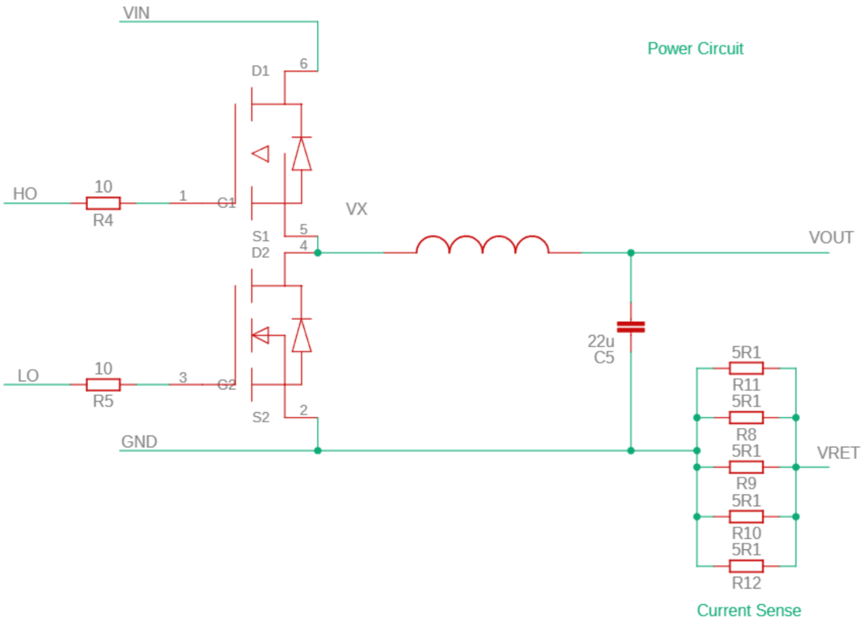


Figure 7: LED Driver Schematic

However, during the independent characterization of each SMPS, the group noticed that the brightness of each LED loads seemed to be different, even though they are all connected to the same bus voltage in parallel. This led to the understanding that the different LED colours draw different amounts of power. Therefore, each LED load was characterized individually. This was achieved by incrementing the output PWM of 1000 and determining its relationship with V_{ret}. Each LED load had a unique lower output PWM limit which causes it to draw almost zero power.

On the other hand, the upper output PWM limit was set when the LED load draws more than 1W of power. This acted as a safety limit since the combined power of the loads does not need to exceed 4W. All of these measurements were recorded in an Excel sheet, shown in Appendix B, and then the relationship between output PWM and power was plotted on a graph.

The initial idea for relating the required power level with the corresponding PWM was to use a mixture of a lookup table, for power levels having a directly corresponding PWM, and linear interpolation, for power levels that came in between two defined power levels.

The premise of linear interpolation operation looks at the existing power points just before and after the desired power level. From here, a straight line is formed between the two power points and their corresponding duty cycles. The gradient is found as $m = \frac{y_2 - y_1}{x_2 - x_1}$, where these points are the defined power points and the duty cycle at which this PWM occurs is found as:

$$y_0 = m(x_0 - x_1) + y_1 \tag{6}$$

Where x_0 is the desired power level, and y_0 is the PWM this occurs at. The figure below looks at the relation between a set of defined power points and the duty cycles at which they occur. This was done by manually varying the duty cycle, and calculating the power through:

$$P = \frac{V_{Ret} \times V_{Out}}{1.02} \tag{7}$$

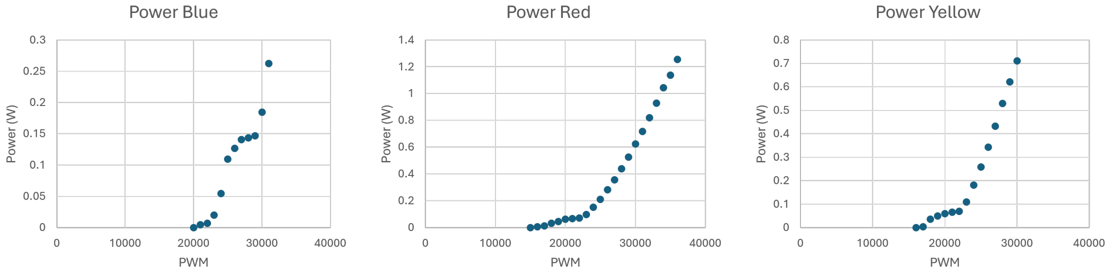


Figure 8: Output PWM against power for each LED load

An important note is that the figures are plotted with PWM on the x-axis and Power on the y-axis. This is done to stick to the convention that PWM will determine the output power, but the concept of linear interpolation will use this in reverse. This method, however, presented two key problems:

- There was non-linearity between the existing points, so making a linear assumption will have implications on accuracy.
- The trend for power points outside the defined range cannot be assumed to inherit earlier trends.

As a result, it was chosen to actively vary the PWM and monitor the corresponding power, then compare this to the desired power to know whether to increase or decrease the PWM. The power was calculated using equation (7).

This has the advantage of being extremely accurate due to it not relying on any mathematical assumptions. As a precursor to the testing stage, this was a successful method of varying the power draw in response to the server signal. The specifics regarding how the data received from the system is converted into individual output PWM inputs into each LED load will be further explained in Section 6.

4.4 Integration of the Circuit Subsystems

4.4.1 Integration of PV Cell and PSU subsystems

The first problem encountered was the stability of the SMPS, particularly in maintaining a stable voltage which adapted responsively to varying load requirements but also minimised oscillations. The assignment had warned that the PI values used were not designed for boost closed-loop control, and should be modified. Following this, two approaches were chosen to ensure a stable control system:

- Iterative design through trial and error.
- Confirmation through the use of Simulink®, a MathWorks® product."

The principles of iterative PI controller design start with setting $K_I = 0$ and incrementing K_P until the controller gets very close to the desired setpoint. Once the controller maintained a stable output of around $6.8V$, the steady-state error was dominating, and the integral controller was slowly modified to remove it, leading to a stable output voltage at $7V \pm 1\%$. This was further confirmed and tested using Simulink, where a crude boost converter was designed to observe the speed and oscillations present in the controller. The steady response is shown in the following figure:

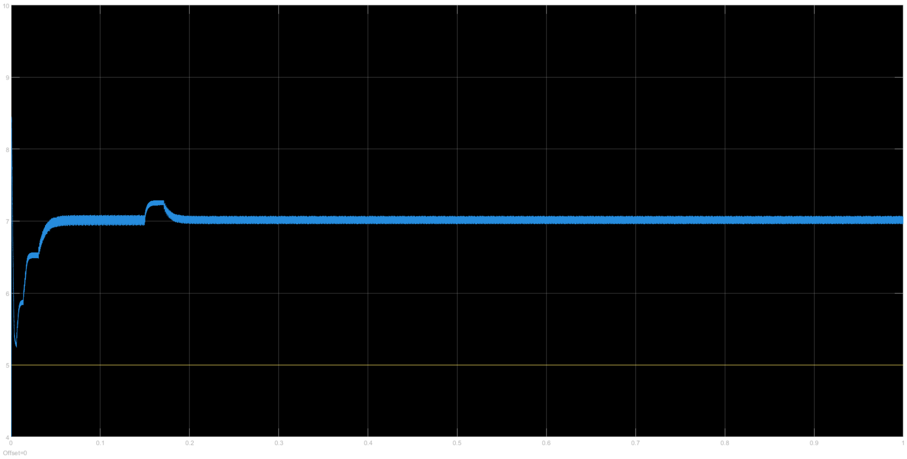


Figure 9: PI controller output with a focus on steady-state behaviour

The figure below focuses on the transient response of the controller, to observe the overshoot and ensure it falls within the safe operating range of the SMPS and the overall system.

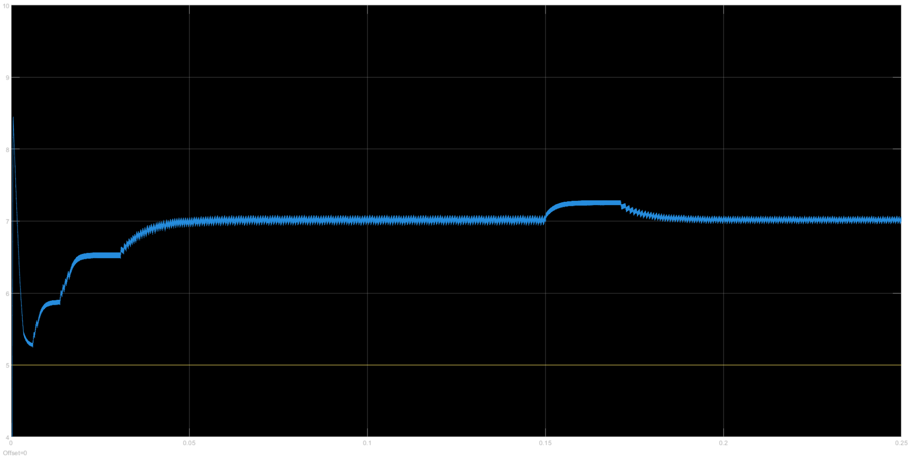


Figure 10: PI Controller output with a focus on the transient response.

As can be seen, the maximum overshoot is at $8.5V$, which is well within the safe operation for both the SMPS and the overall circuit. Lastly, for reference, the boost controller circuit is shown below the table, in Figure 6. This was created in MATLAB to assist with the circuit testing.

Parameter	**Value**	**Units**	**Description**
K_P	150	N/A	Proportional Controller Constant
K_I	25	N/A	Integral Controller Constant
V_{Bus}	$7 \pm 1\%$	V	Regulated Bus Voltage

Table 3: Key Parameters for PI Controller

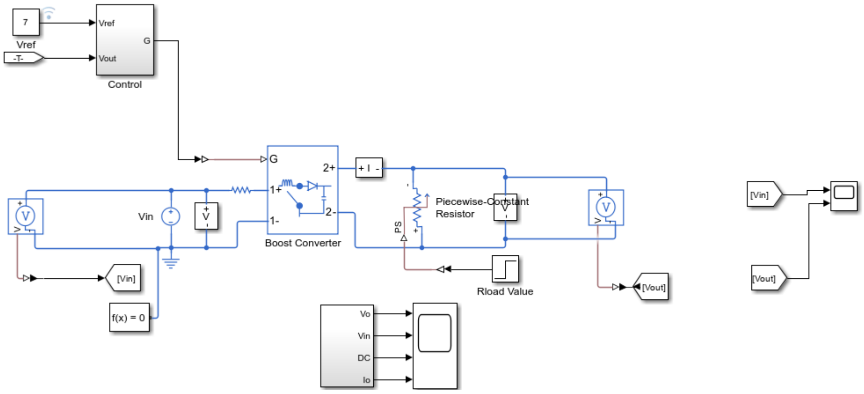


Figure 11: Boost Circuit Used to Test PI Values, based on MATLAB Model[1]

4.4.2 Integration of Full Circuit

When assembling the full circuit, there were a couple of issues regarding the saturation of the SMPSs, which were resolved after lowering the current limiters. However, there was a major hardware limitation which has a significant effect on the decision algorithm. When charging the capacitor, the circuit draws current from the bus voltage into the capacitor, which means the bus voltage would go below 7V. However, due to the operation of the voltage regulator, the SMPS would automatically draw power from the PSU to compensate for the charging capacitor and keep the bus voltage constant again. This implies that charging the capacitor would result in a buying operation and discharging it would result in a selling operation. The table below clearly shows what happens throughout the circuit when a positive or negative current is set throughout the SMPS.

Direction of Current	Capacitor State	Duty Cycle	Operation
Negative	Charging	Increasing	Buying
Positive	Discharging	Decreasing	Selling

Table 4: Circuit Operation from changing current direction

5 Software Design and Implementation

5.1 Software Specification Requirements

Similarly to the hardware specification, the members in charge of software decided to list out the requirements that need to be met on the software side:

- Display the findings of the hardware on an interactive UI
- Perform better than the naive algorithm and give operations/decisions to the microcontrollers. This includes decisions regarding the deferable demands
- Accurately predict the trends of the buy and sell prices, as well as the demand function

5.2 Timing and robustness of HTTP requests

A timing system was implemented, that allows decisions to be made for the correct data tick and data (most recently read) in a robust manner.

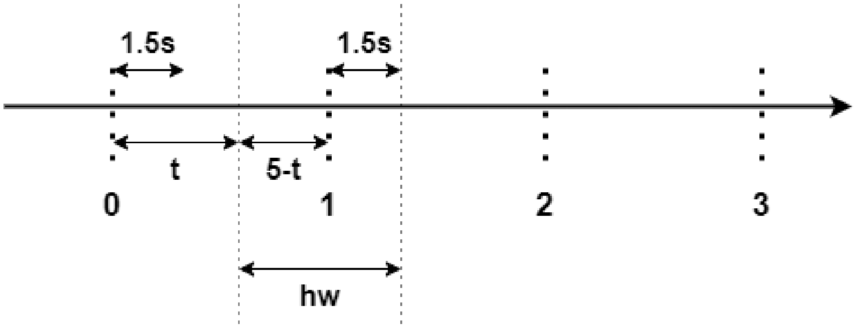


Figure 12: Timing diagram

Key

- t: time taken for computation
- hw: The window of time for sending the decision to hardware and for hardware to perform the decision
- $5 - t$: time to next tick

When the algorithm starts up, it makes sure to start at the first new tick it reads from the live server. This way, the first read is in sync with the changing tick on the live server. There is a set parameter called `computation_time = 1.5s` which specifies the minimum amount of time any computation needed to make a decision at a tick takes. This allows us to set a hardware window, which is the time given to the hardware to perform the decision before receiving a new decision from the algorithm.

In most cases, given good internet, the hardware window is 5 seconds. If not, it is usually a little less than that. For example, if some computation $t > 1.5$, then $hw = 6.5 - t < 5$. The time to $t + 1 = 5 - t$ is spent waiting for the live tick to change.

If $t > 5$, the tick finder will know to immediately increment the tick, and move on to the next decision. If polling the live server times-out, then cached data is used, and the tick is incremented.

HTTP requests to the live server for buy price, sell price, and demand are made asynchronously using Python's `asyncio` library.

5.3 Buy-Sell Algorithm

5.3.1 Genetic Algorithm

A genetic algorithm was made from scratch, which trains a population of neural networks to predict buy price, sell price and demand. A genetic algorithm is inspired heavily by evolutionary principles. It tries to find a good local minima of loss on a problem by hand picking neural networks with good weights in a tournament pool. Start with a set of n neural nets, each with randomized weights. Run each neural net on the prediction problem, and by comparing its prediction with the actual values, work out the fitness of the neural network. The reciprocal of mean squared error was used as our fitness score. Create a new population of neural networks using this fitness information. The top $k\%$ (elitism) of neural nets are passed directly to the new population. The rest is filled by doing a crossover and mutation step. Crossover refers to picking weights from 2 parent neural nets to form a child neural net. Mutation refers to adding random noise to the weights. This process is repeated over multiple epochs. All parameters are easily tunable via a top-level interface.

```
self.trainer = Train(elitism=0.0, mutation_prob=0.08, mutation_power=0.1, max_epochs=20,
    num_of_histories=5, pop_size=60, nn_batch_size=4, parsed_data=self.serve.parsed_data,
    conc=True)
```

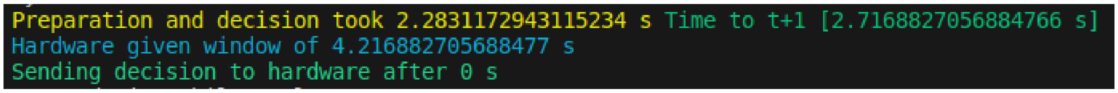
Listing 1: Training parameters

5.3.2 Future Predictions using a genetic algorithm

Predictions are made for the entire cycle for buy price, sell price, and demand. Given that in this model, the *buy:sell* ratio is 0.5, the prediction for the buy price is simplified by multiplying the sell price by this ratio. The system is designed for learning to occur over the course of the cycles, with no hard-coded values. Instead, the prediction models adapt and learn during the cycles.

- Setup a population of neural networks with random weights, or load a previously saved population
- On each 15th tick, train each neural net to predict the last 15 values given as input the past 5 values at that tick.
- Get the best performing neural net, and have it predict the next cycle's value at that tick given as input the 5 previous values, now including the value at the current cycle.

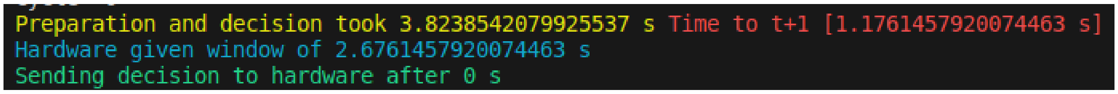
The process above takes at most 3 seconds, vastly reducing in time due to future populations having a head-start (not having to start from neural networks with random weights). This is implemented by ensuring that the best-performing populations are saved, and a fitness threshold multiplied by 5 to avoid limiting the neural networks.



```
Preparation and decision took 2.2831172943115234 s Time to t+1 [2.7168827056884766 s]
Hardware given window of 4.216882705688477 s
Sending decision to hardware after 0 s
```

Figure 13: Training 225 neural networks, batch size 15, 20 epochs, from random population, *multiple cores*

To help reduce computation time, a population of neural networks is batched up, and each batch runs in a separate process on its own core. This allowed for an increase in neural network populations with little overhead, which led to get better results from the training loop.



```
Preparation and decision took 3.8238542079925537 s Time to t+1 [1.1761457920074463 s]
Hardware given window of 2.6761457920074463 s
Sending decision to hardware after 0 s
```

Figure 14: Training 225 neural networks, batch size 15, 20 epochs, from random population, *one core*

The algorithm was implemented on a tick-by-tick basis. On tick 0, predictions for the current cycle were set up, having been prepared during the training loops in the previous cycle. The value at tick 0 is then added to a data buffer, and the optimization algorithm is executed. At each 15th tick, predictions for the next cycle are prepared, the value at that tick is added to the data buffer, and the optimization algorithm is run. On every other tick, the value is added to the data buffer, followed by the execution of the optimization algorithm.

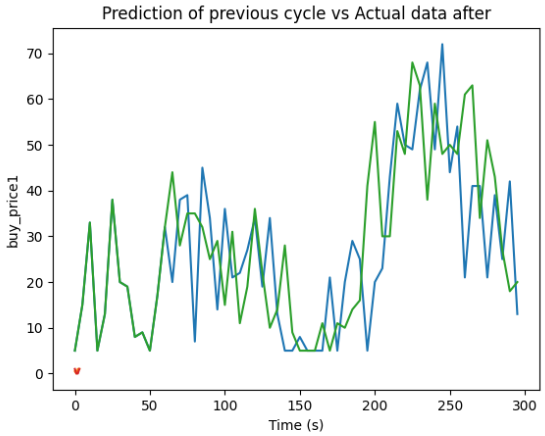


Figure 15: Buy price prediction made after one cycle of training

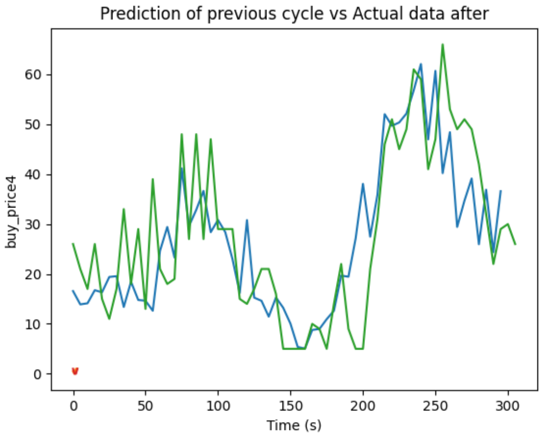


Figure 16: Buy price prediction made after 4 cycles of training

In Figure 15 and Figure 16, green is the actual data from the cycle and blue is the prediction that was prepared for that cycle. In Figure 15 up until about 70s (tick 14), the graphs are superimposed. This is because when the algorithm started up, the external server was at tick 14, so the next predictions and data buffers were filled with historical data.

Using the timing system, testing showed that all components of the algorithm run fast enough, such that at least 90% of the time (provided a stable connection) upon reading a tick t from the live server, successful decision making was seen on the hardware at tick $t+1$. So the hardware is exactly one tick out-of-phase with the live server. In all other cases, the system handles points of failure and knows how to self-correct back to this optimum.

5.3.3 Integration between Machine Learning and Optimization

A driver was developed to read live data from the external server and store it in buffers used by the machine learning and optimization algorithms.

- On tick 0, data, the predictions to be used in the new cycle are retrieved from a pre-prepared (from the previous cycle) buffer, all data and next prediction buffers are cleared, and deferrable demands are stored. In the event of the algorithm starting when the server is also at tick 0, a set of 5 historical data points is synthesized for the machine learning algorithm. At the start of the next cycle, one of the 5 historical data points is dropped and replaced with the newest data point. Once this is done, the optimization algorithm runs using the newest live values.
- On every 15th cycle, the live values are added to data buffers, and the optimization algorithm is run. Thereafter, the predictions for the next cycle are prepared.
- On every other tick, only the optimization algorithm gets run

For each of these computations, the time taken is measured and used to calculate the hardware window as explained in Section 5.1. Decisions made by the algorithm are added to a queue, which can be accessed by the TCP server running in a separate thread.

5.3.4 Optimization Algorithm

The naive approach for the algorithm was defined by a set of simple if-else statements as shown in Figure 3. As seen below, the current naive algorithm does not make use of future predictions generated by the machine learning model and instead makes the decision by a tick-by-tick basis. This can be further improved by: considering algorithms that can *look ahead in future time*, and by making better decisions than a simple if-else statement.

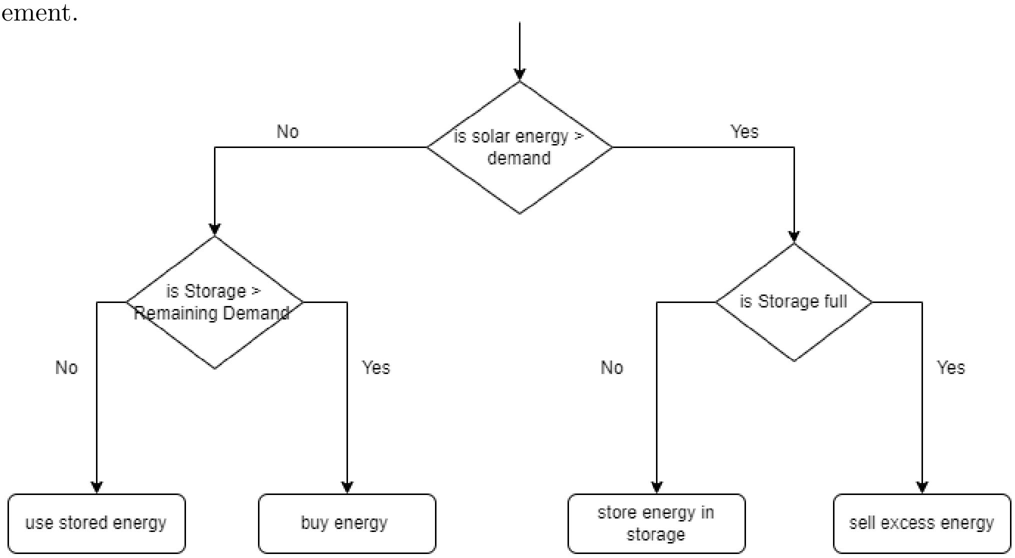


Figure 17: Naive algorithm flowchart

Considering the look-ahead feature, there were 2 possibilities that were considered: *heuristics* and *model predictive control*. By using heuristics, it was possible to manually 'hard code' the solution in this scenario since base functions remained the same. However, it was decided to use the MPC algorithm, since its rigorous constraint handling and non problem-specific nature would have better flexibility and performance overall. When MPC is coupled with MILP, it would yield an optimal solution whereas heuristics may have sub-optimal solutions.

Since the predictions obtained by the machine learning model only changes every 60 ticks, the MPC prediction window was decided to be a length of `60-current tick`. This means that the algorithm would constantly be generating the most optimal set of solutions each step taken

Once the method of using future predictions was devised, the focus shifted to making the best possible decision at each given tick. For this choice, Mixed Integer Linear Programming was used over other methods such as dynamic programming and non linear program since integer linear programming can handle the amount of non correlated constraints required to solve this problem. Another key reason I chose this option is because of the pre-made library Coin or Branch and Cut module which is an open source library that is explained by in Figure 18

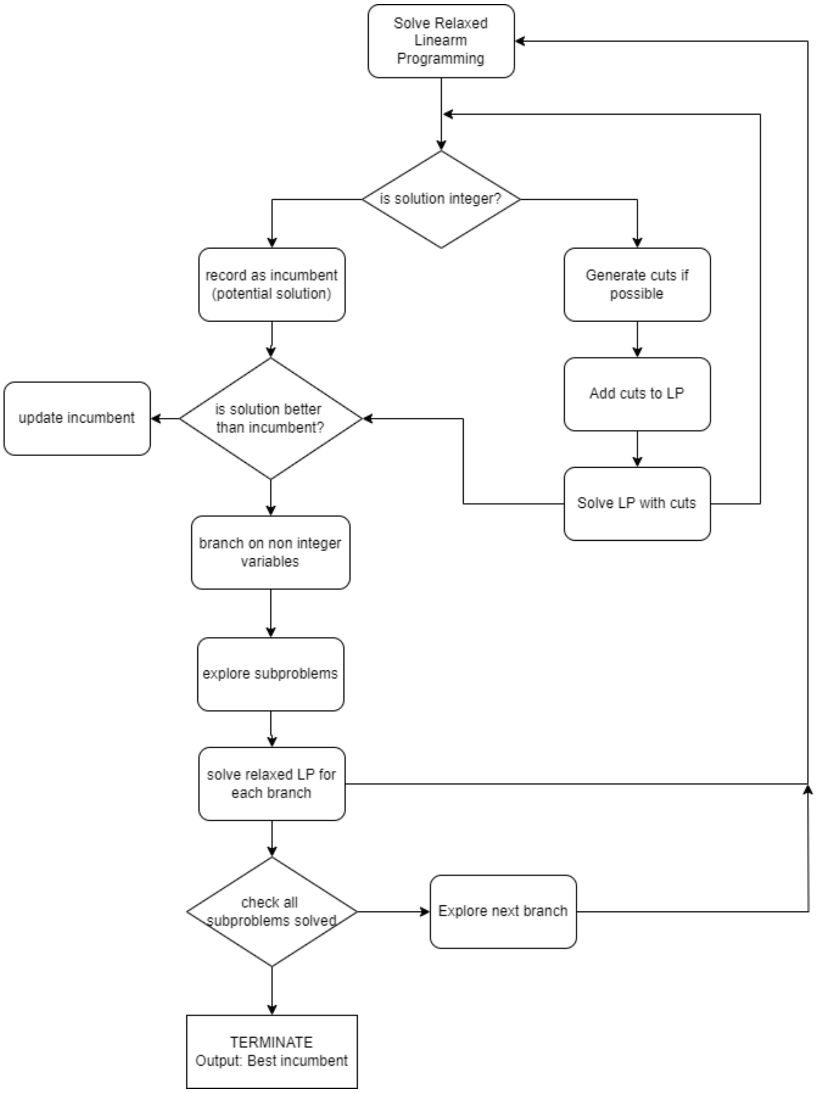

Solve Relaxed Linearm Programming
is solution integer?
record as incumbent (potential solution)
Generate cuts if possible
update incumbent
is solution better than incumbent?
Add cuts to LP
Solve LP with cuts
branch on non integer variables
explore subproblems
solve relaxed LP for each branch
check all subproblems solved
Explore next branch
TERMINATE Output: Best incumbent

Figure 18: Coin and branch or cut flowchart

When performing Linear Integer Programming to find the maximal/minimal situation, a set of sensible constraints are required to create a bounded problem. Listing 2 describes a snippet of code used when declaring some of the constraints in this scenario.

```
    M = 1e5 # arbituarily high number constraint
    constraints = [
        storage_level[0] == storage,
        solar_energy[0] == energy_in,
        demand[0] == energy_used,
        energy_transactions == pos_energy_transactions - neg_energy_transactions,
        storage_transactions == pos_storage_transactions - neg_storage_transactions,
        energy_transactions + storage_transactions + solar_energy - demand >= 0,
        pos_energy_transactions <= M * (1 - x),
        pos_storage_transactions <= M * x
    ]
```

Listing 2: Example of constraints

When testing the optimized model against the naive algorithm, the optimized model performed better than the naive algorithm in the long run. However, in the first cycle the naive and the optimized algorithm performed similarly, and it is only as the Machine learning predictions improve, better results becomes increasingly obvious in the optimization algorithm.

5.4 API development

5.4.1 Brainstorming

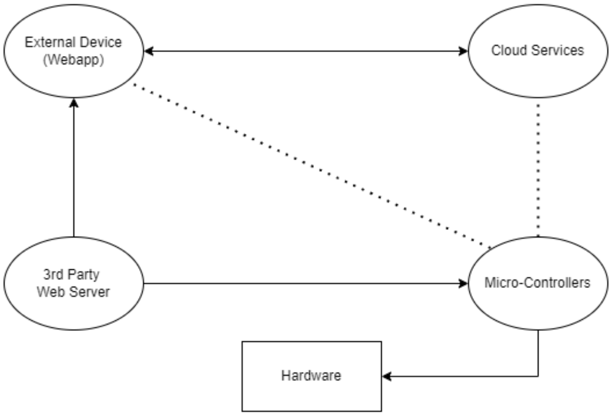


Figure 19: Initial API Design Concept

The design for the storage of historical data was heavily influenced by the initial software system map drafted from brainstorming sessions. A core decision was whether calls to our external cloud services would be made by an external device or the actual microcontrollers on the hardware itself. It was eventually concluded that an external device acting as a centralized meeting point for all points of data was easier to manage and maintain, as well as easier for us to collaborate and work on.

The next dilemma was handling the intervals for calling and sending data to and from this external database. Since the decisions and historical values for energy were alternating every 5 seconds, processing all these calls on a cloud service every 5 seconds would be relatively expensive due to the costs being determined by the active time of cloud services. Furthermore, calls every 5 seconds were more prone to data loss and scaling issues. This led us to take inspiration from industry standards to handle the storage of historical data daily (so every 5 minutes instead). This would mean that daily data would be kept locally until the end of a day cycle before being sent to the cloud. The user would still be able to see current insights into their historical data but only for the previous day and before.

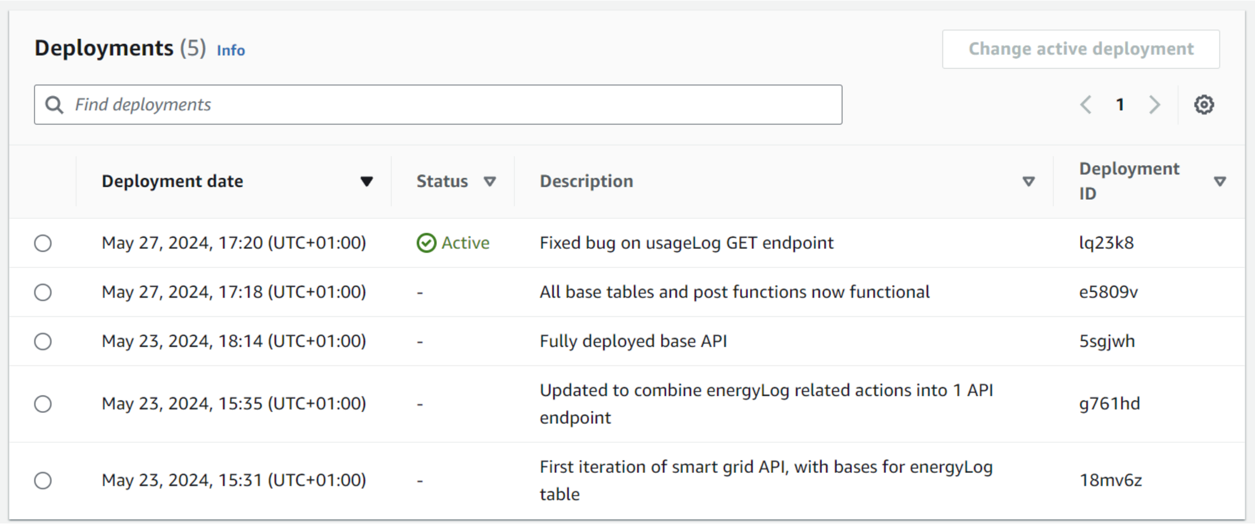
5.4.2 Production

During the production of the cloud services, different AWS consoles and configurations were experimented with to deploy our storage options. It eventually came to the conclusion of using the Lambda functions services paired with Cloud-Watch logs for monitoring our usage and error logs as well as the API gateway feature to enable the Lambda function triggers. After discussion, it was decided that AWS Lambda best suited our use case for this project as it was an easily scale-able serverless service that was pay-as-you-use. This means that the server would only be online for a few minutes at a time, this would significantly reduce the running costs for the daily long-term usage of back-end services within our web app and energy system.

The downsides to Lambda were occasional slow responses during the initial startup of the process and instance, but the speed of the response was not a major focus for the cloud services as the majority of the processing for the buy, sell algorithm and management of the energy systems would be done locally and on micro-controllers instead of the cloud service which rather acted as a log of past data and events.

When handling the code for the APIs and generating the respective endpoints, proper code ethics were used with camel casing, clear comments of packages imported and identification of core functions. Alongside this, industry standards of API keys and proxy integration were closely followed to ensure usability for all systems. Edge case detection was also implemented to better handle and accelerate debugging during events of incorrect data formats being sent. Implementation of the Time To Live feature was also added further down the line with aims to limit the volume of data stored on our services and prevent a reduction in performance.

A clear deployment pipeline was also followed to allow for clear tracking of the changes made to each stage and to allow for rollback in the event of unfixed bugs hindering the development and functionality of other API calls.



Deployments (5) Info

	Deployment date	Status	Description	Deployment ID
○	May 27, 2024, 17:20 (UTC+01:00)	✓ Active	Fixed bug on usageLog GET endpoint	lq23k8
○	May 27, 2024, 17:18 (UTC+01:00)	-	All base tables and post functions now functional	e5809v
○	May 23, 2024, 18:14 (UTC+01:00)	-	Fully deployed base API	5sgjwh
○	May 23, 2024, 15:35 (UTC+01:00)	-	Updated to combine energyLog related actions into 1 API endpoint	g761hd
○	May 23, 2024, 15:31 (UTC+01:00)	-	First iteration of smart grid API, with bases for energyLog table	18mv6z

Figure 20: API Deployment Pipeline

5.5 Integration between Algorithm and Web Server

The conventional approach to pass data to the front-end is through a web socket. However, since the algorithm and web server will both run locally on the same computer, it was decided on a simple solution that allows us to show data from our algorithm on the web server. On startup, create a `data.json` file that can be read by the web server. Then, add a JSON string with keys that denote data from the TCP server, and from the algorithm.

```python
def init_frontend_file():
    try:
        initial_data = {str(i): [] for i in range(60)}  # Create keys ’0’ to ’59’ with empty lists

        # Write the initial structure to the JSON file
        with open(json_path, "w") as f:
            json.dump(initial_data, f, indent=4)

        print(f"Initialized {json_path} with empty data for ticks 0 to 59.")

    except Exception as e:
        print(f"An error occurred during initialization: {e}")
```

Listing 3: Initializing JSON file

Then, to add data to the front-end server, a function is called that writes to the JSON file. Note that this is done with a `Lock()`. This is because the TCP server and algorithm run on separate threads, and is required to write to the same file safely.

```python
def add_data_to_frontend_file(data : Dict):
    with lock:  # only one thread can access this at any one time
        if ’tick’ not in data:
            raise ValueError("The ’data’ dictionary must contain a ’tick’ key.")

        tick_value = str(data[’tick’])  # Convert tick to string to match JSON keys

        try:
            # Check if the file exists
            if not os.path.exists(json_path):
                raise FileNotFoundError(f"The file {json_path} does not exist. Initialize it first.")

            # Read the JSON file
            with open(json_path, "r") as f:
                json_data = json.load(f)

            # Ensure the tick value exists in the JSON data
            if tick_value in json_data:
                # Append new data to the list for the specified tick
                json_data[tick_value].append(data)
            else:
                raise ValueError(f"Tick {tick_value} is not valid. Must be between ’0’ and ’59’.")

            # Write the updated JSON back to the file
            with open(json_path, "w") as f:
                json.dump(json_data, f, indent=4)

            if tick_value == "59":
                print("Tick 59 reached. Clearing data...")
                init_frontend_file()

        except json.JSONDecodeError as e:
            print(f"Error decoding JSON from the file {json_path}: {e}")
        except FileNotFoundError as e:
            print(e)
        except Exception as e:
            print(f"An unexpected error occurred: {e}")
```

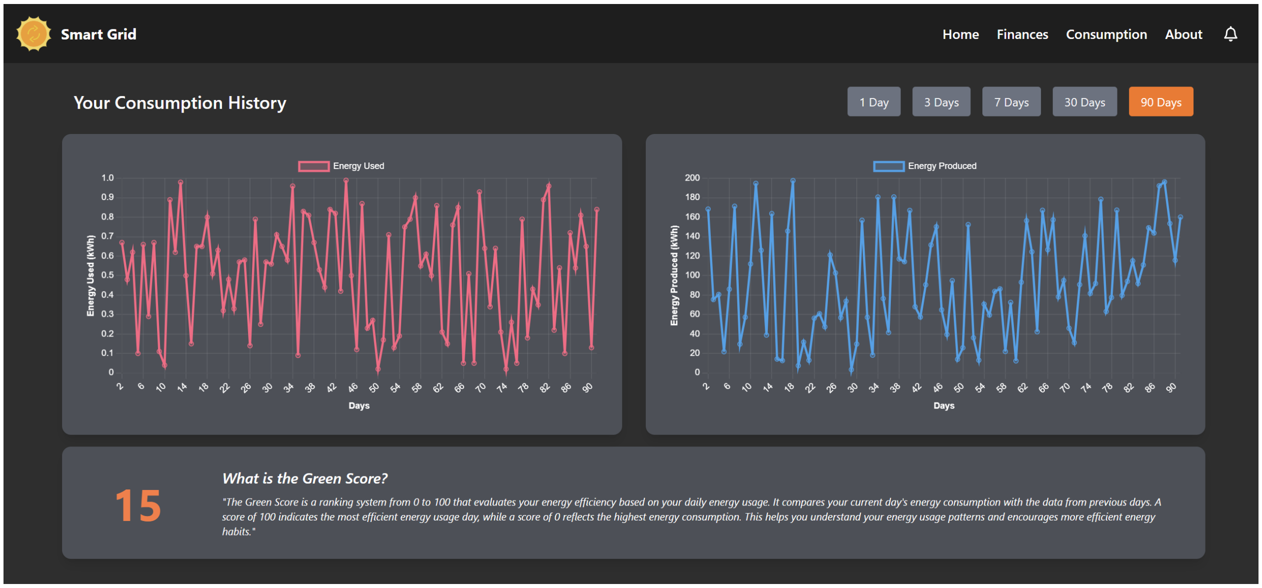
Listing 4: Adding data to JSON file

5.6 Web-Server Design

5.6.1 Initial Ideas and Concept

A successful web application is intuitive and straightforward for users whilst still effectively conveying all the essential information intended by its creators. React was selected as the platform to achieve this due to its balance of flexibility and efficiency as well as its popularity in developing modern web interfaces. This popularity meant an extensive presence of online libraries to reduce the work needed to achieve desired designs, as well as a large presence of online resources to assist in any difficulties during the development process.

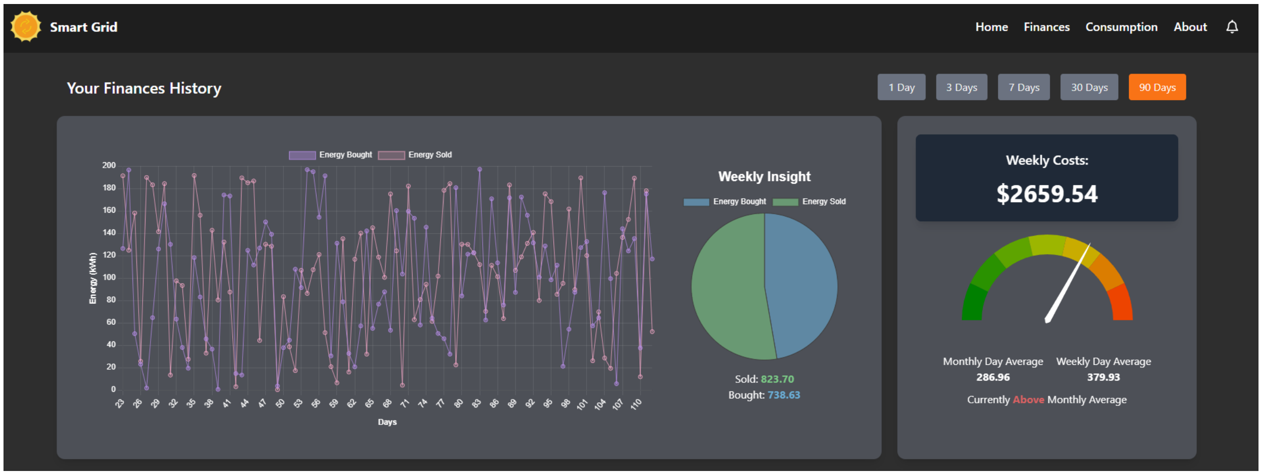
React’s component-based architecture allows for a modular approach, making it easier to scale and manage the application as needed. Minimalism was a core design principle from the initial drafting stages, an emphasis on clean and uncluttered interfaces to prioritize user experience was the end goal. To achieve the presentation of data in a manner that is both aesthetically pleasing and easy to understand a combination of visual graphs, charts and informative scores/statistics was integral. This visual-centric philosophy ensured that the most critical information was always front and centre, facilitating better decision-making for users interacting with the energy smart grid platform.



What is the Green Score?

"The Green Score is a ranking system from 0 to 100 that evaluates your energy efficiency based on your daily energy usage. It compares your current day's energy consumption with the data from previous days. A score of 100 indicates the most efficient energy usage day, while a score of 0 reflects the highest energy consumption. This helps you understand your energy usage patterns and encourages more efficient energy habits."

Figure 21: Consumption page



Sold: 823.70
Bought: 738.63

Weekly Costs:
$2659.54

Monthly Day Average
286.96

Weekly Day Average
379.93

Currently Above Monthly Average

Figure 22: Finance page

5.6.2 Development

The development phase focused on leveraging React's strengths with key components designed to handle data input, processing and visualisation efficiently. The backend was integrated with APIs from AWS to fetch and update historical data, live data was handled via parsing a JSON file, ensuring that the users would always have access to the most recent information from each tick.

Chart.js was utilized for creating data visualisations, this library allowed for the seamless integration of different types of charts and graphs. Extensive configuration options were also present to ensure that the data was presented in a manner which best suited the visualisation desired. Aside from libraries, proper development practices were followed with the application being divided into well-defined components for each page, with routes managed to ensure clear segmentation of sections and their respective elements. Helper functions were also created for global components to maintain consistency and reusability across the platform.

An assets folder was also maintained to store global images, icons and other static resources. A core UI CSS was developed to ensure a uniform design throughout the application, promoting a cohesive look and feel. Furthermore, the development process was guided by a milestone approach with specific components and features targeted for completion at each stage. This method allowed for systematic progress tracking and timely identification of potential bugs and integration issues.

Simulated data was also generated to visualise the graphs during the initial development phase, this approach facilitated early testing prior to integration and was core for the refinement of visual components. This data was then later overwritten and replaced with data fetched from either the cloud or from the local hardware sources with minimal hiccups.

5.6.3 Feedback and Improvements

Feedback from initial user testing highlighted several areas for improvement, many appreciated the overall design and functionality but suggested enhancements in data visualisation clarity and softer styling elements. In response, the design was iterated to better suit a pastel colour palette and to include more intuitive charts and graphs to provide clear insights into energy and financial trends.

Performance optimisation was also implemented to reduce load times and improve the responsiveness of the application. Lazy loading techniques were integrated to reduce the perceived delays by loading components and data only when needed. This approach significantly enhanced the user experience and was well-received by those who tested the application before and after its addition.

6 Hardware and Software Integration

The method of data communication for the entire Smart Grid system involves connecting to the micro-controllers or the Raspberry Pi Pico through a TCP connection to the main laptop which runs the algorithm. As each Raspberry Pi Pico comes with an unique IP address, the main laptop can connect to multiple micro-controllers simultaneously and send its decisions from the algorithm. In the hardware perspective, it is essential to communicate with the members who are in charge of the software charge, in terms of what type of data will be sent to which micro-controller within each tick.

6.1 Software-Hardware TCP Connections

The integration process between the hardware and software side is integral and hence will undergo thorough testing and fallback plans in the case of unstable internet.

6.1.1 Specification

Much like the previous sections, first explore the functionalities that the subsystem needs to perform.

- Be able to send data from the server to a specific client. This data will pertain information on the decisions made my the algorithm
- Be able to send data from the client back to the server. This will mainly pertain to ACKs as well as any other control data that the algorithm may require
- Be able to forcefully try to reconnect back to the server in case of disconnection
- Server should be able to handle multiple clients disconnecting and connecting
- After TCP client receives the data, the hardware-side code should be able to change according to the received data

6.1.2 Design Process

The algorithm and TCP server run on the same laptop on separate threads. These threads communicate via a queue, implemented using the `Queue` library in Python, which was chosen since it is thread safe.
On startup, both threads are started, and given access to the queue like so:

```python
q = Queue()

def main():
    init_frontend_file()
    algo = Algorithm()
    algo_thread = Thread(target=algo.algo_driver, args=(q, ))
    tcp_thread = Thread(target=m_tcp.run_server, args=(q, ))

    algo_thread.start()
    tcp_thread.start()

    algo_thread.join()
    tcp_thread.join()
```

Listing 5: Algorithm and TCP server startup

When the algorithm is done with computation to get a decision, it enqueues its decision, adding a name tag for the specific client it should be sent to.

The TCP server runs separate client handler threads for each Pico that connects. Inside each handler, if the queue has some data (from the algorithm) and it is specifically for that client, it gets sent to the client. The TCP client (Raspberry Pico) connects to WiFi and to the server, then sends its name tag to the server. This is how the server knows which client decisions should go to.

Another important feature that was included in the design was rigorous error handling. Especially when connected to multiple picos at the same time, it was necessary to add things like a *re connection* in case of sudden disconnection and multiple try and except statements across both the client and server code.

```python
while client_socket.fileno() != -1:
    utime.sleep(1)

print("Reconnecting to the server...")
```

Listing 6: Reconnecting code

Figure 23 shows a clear visualisation of our final integration design.

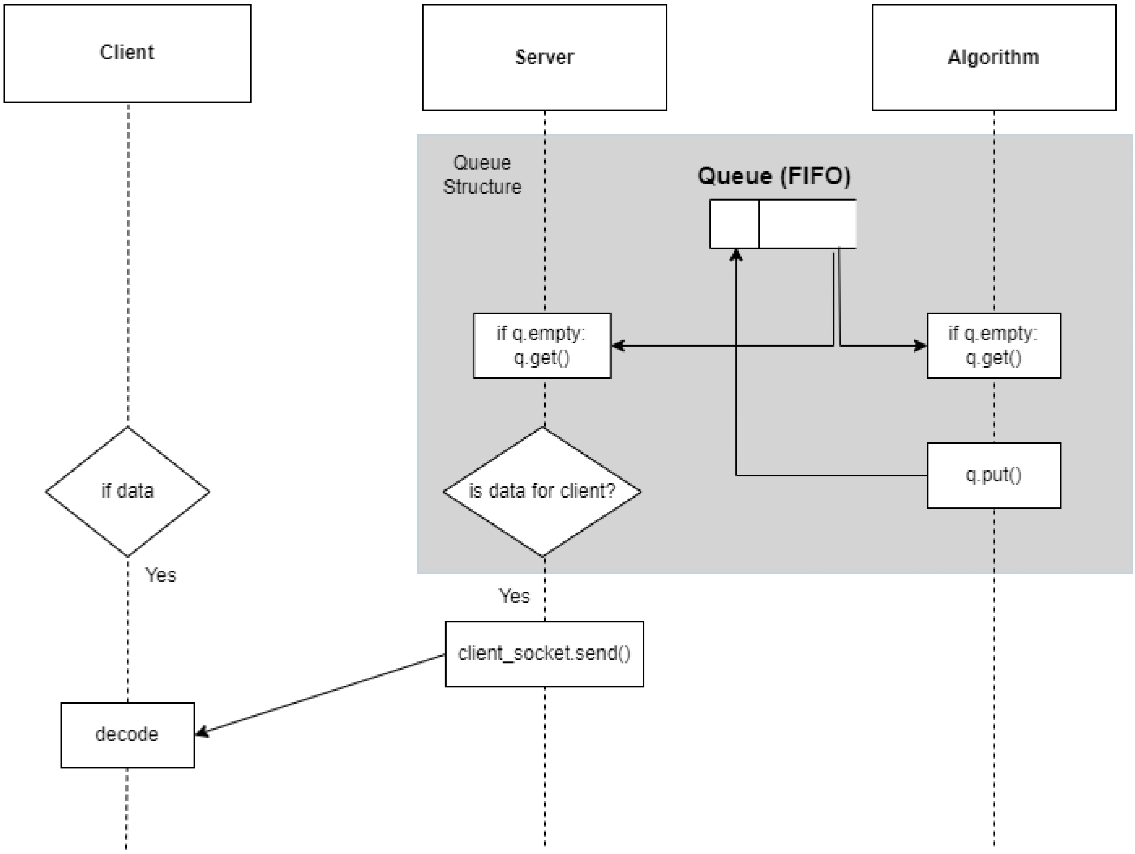


Figure 23: TCP General Idea

6.2 Integration of Decision and Circuit Operation

After receiving the decision from the algorithm through the TCP server, the next focus in the hardware perspective would be to convert the decision received to the corresponding duty cycles of the SMPSs to achieve the desired operation.

The general structure of the data which will be sent to the hardware is shown below.

6.2.1 Integration of Data and LED Loads

To control how much power is drawn by the LED loads, each Buck SMPS will receive an integer which represents the total demand of power from the loads, evenly divided by 4, which is the total number of LED loads. By referring to the characterization of the Buck SMPS in Section 4.3.4, the power demand of each LED load is achieved by trial and error, by increasing or decreasing the output PWM incrementally until it reaches its desired power.

6.2.2 Integration of Data and Current Control

As mentioned in Section 1.4.3, the SMPS on the capacitor alone is capable of controlling the current flow, the charging state of the capacitor and therefore the buying or selling operation. The conversion of data to setting the magnitude and direction of the current will have to be integrated in two different scenarios. The first scenario is when the power is positive, or when the capacitor is charging. Subsequently, the current required to achieve the specified power can be calculated by using Ohm's Law and dividing $\frac{Power}{V_b}$, where V_b is the voltage at port B of the capacitor SMPS. The second scenario, on the other hand, is when the power received is negative, which implies the discharging operation of the capacitor. In this case, the current required would similarly be calculated by $\frac{Power}{V_a}$, and V_a is used instead as it represents the voltage in the capacitor. It is important to remember that a negative current on the SMPS implies buying, while a positive current implies selling.

7 Testing Methods and Overall Performance Evaluation

7.1 Hardware Testing

PV Array Subsystem: The Boost closed-loop SMPS was tested to ensure it outputs 7V when the PSU mimicking the PV array is at its maximum power. Additional testing involved manually lowering the voltage and current limiters on the PSU to verify consistent 7V output.

- Conclusion: Test Passed

PSU Subsystem: Testing of the voltage regulator was completed with the full circuit connected. It was observed that the bus voltage varied during the charging and discharging of the capacitor. However, the voltage regulator did not function correctly when the initial bus voltage was at 0V. The PV array and Boost closed-loop subsystem needs to be operational at around 7V to allow the regulator to calculate and adjust the voltage properly. If the initial error is too large (e.g., bus voltage at 0V), the duty cycle saturates due to hitting the controller limit.

- Conclusion: Test Passed (only if the PSU for the PV array is turned on first)

Capacitor Subsystem: The current control function was tested with the full circuit. The objective was to ensure that once the capacitor is fully charged, no additional current is forced into it, thus avoiding hitting the SMPS safety limit. This also applied when the capacitor equals the bus voltage and cannot discharge further.

- Conclusion: Test Passed

LED Loads: The stability of the control system in the LED load subsystem was tested by monitoring the duty cycle to ensure it did not oscillate around the desired duty cycle under varying power demands.

- Conclusion: Test Passed

7.2 Software Testing

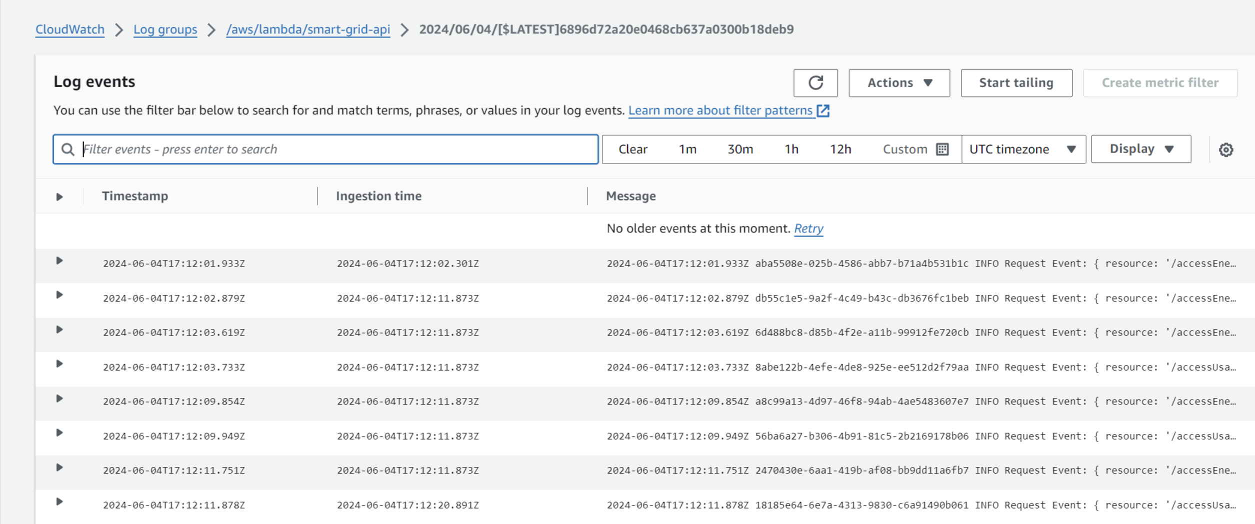
The software approach to testing was primarily via unit and integration tests. In special scenarios, usage of external tools was used to obtain extra information from a targeted subsystem. Numerous scripts for simulated data generation were also employed to analyse performance metrics. The table below details the different subsystems and the testing protocols followed.

System Requirements	**Testing Methods**	**Metrics**	**Result**
API Endpoints	POSTMAN and Unit Testing	Server Latency and Local Processing Times	Test Passed
Machine Learning Model	Comparing predictions with real data via python script. Integration tests with optimization algorithm	Mean Squared Average	Test Passed
Web Application	Python scripts for generating simulated data and React DevTools to analyse components and loading time	N/A	Test Passed
TCP Connections	Using commands such as ping, netperf and wireshark	Latency, Network Performance, etc	Test Passed
Optimization Algorithm	Used a python test script to compare naive and optimal, added breakpoints for further variable comparison	Difference in profit and core variable values	Test Passed

Table 5: Table of Testing

7.2.1 API Calls

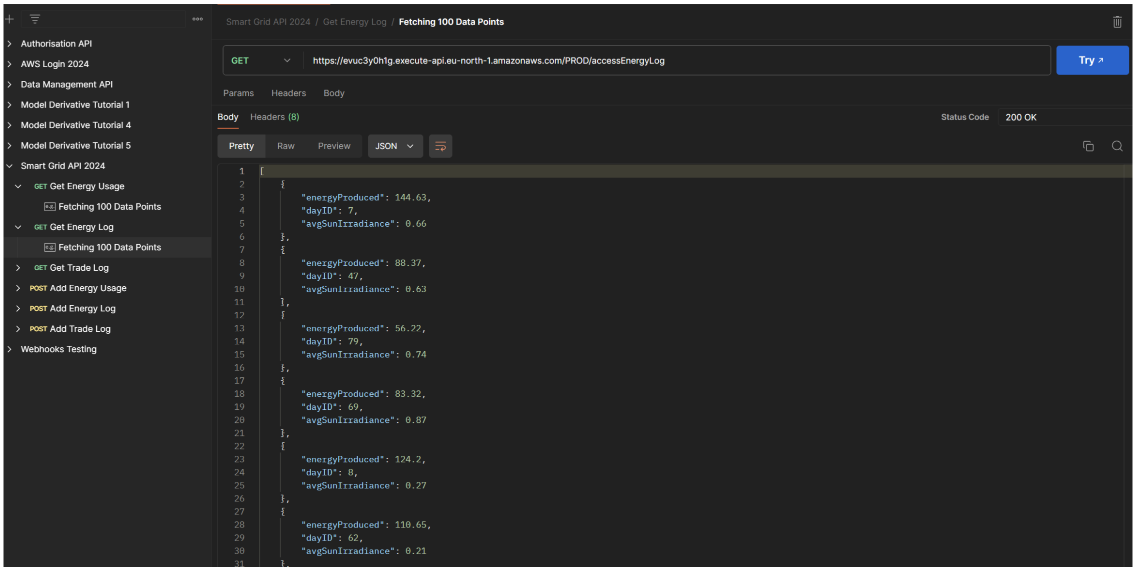
Testing of API calls was an especially integral part of the software system - since APIs are the main communication between different interfaces. Extensive testing was carried out for all cloud services to ensure full functionality as well as handling unexpected behaviour and predicting response times for regular calls to the API. This was done on multiple platforms to ensure vigorous coverage and to simulate different scenarios and points of failure. One core method was using the AWS console for testing and tracing the requests using Cloudwatch Logs. This accelerated debugging and allowed us to swiftly debug unexpected behaviour. Cloudwatch logs were also beneficial when it came to determining the active time of the AWS server and the costs for each API call (the time/resources used and how much of the actual response time on the AWS server was billable time).



CloudWatch > Log groups > /aws/lambda/smart-grid-api > 2024/06/04/[$LATEST]6896d72a20e0468cb637a0300b18deb9
Log events
Actions
Start tailing
Create metric filter
You can use the filter bar below to search for and match terms, phrases, or values in your log events. Learn more about filter patterns
Filter events - press enter to search
Clear
1m
30m
1h
12h
Custom
UTC timezone
Display

```
Timestamp                   Ingestion time              Message
                                                        No older events at this moment. Retry
2024-06-04T17:12:01.933Z    2024-06-04T17:12:02.301Z    2024-06-04T17:12:01.933Z aba5508e-025b-4586-abb7-b71a4b531b1c INFO Request Event: { resource: '/accessEne…
2024-06-04T17:12:02.879Z    2024-06-04T17:12:11.873Z    2024-06-04T17:12:02.879Z db55c1e5-9a2f-4c49-b43c-db3676fc1beb INFO Request Event: { resource: '/accessEne…
2024-06-04T17:12:03.619Z    2024-06-04T17:12:11.873Z    2024-06-04T17:12:03.619Z 6d488bc8-d85b-4f2e-a11b-99912fe720cb INFO Request Event: { resource: '/accessEne…
2024-06-04T17:12:03.733Z    2024-06-04T17:12:11.873Z    2024-06-04T17:12:03.733Z 8abe122b-4efe-4de8-925e-ee512d2f79aa INFO Request Event: { resource: '/accessUsa…
2024-06-04T17:12:09.854Z    2024-06-04T17:12:11.873Z    2024-06-04T17:12:09.854Z a8c99a13-4d97-46f8-94ab-4ae5483607e7 INFO Request Event: { resource: '/accessEne…
2024-06-04T17:12:09.949Z    2024-06-04T17:12:11.873Z    2024-06-04T17:12:09.949Z 56ba6a27-b306-4b91-81c5-2b2169178b06 INFO Request Event: { resource: '/accessUsa…
2024-06-04T17:12:11.751Z    2024-06-04T17:12:11.873Z    2024-06-04T17:12:11.751Z 2470430e-6aa1-419b-af08-bb9dd11a6fb7 INFO Request Event: { resource: '/accessEne…
2024-06-04T17:12:11.878Z    2024-06-04T17:12:20.891Z    2024-06-04T17:12:11.878Z 18185e64-6e7a-4313-9830-c6a91490b061 INFO Request Event: { resource: '/accessUsa…
```

Figure 24: Cloudwatch Logs



GET https://evuc3y0h1g.execute-api.eu-north-1.amazonaws.com/PROD/accessEnergyLog

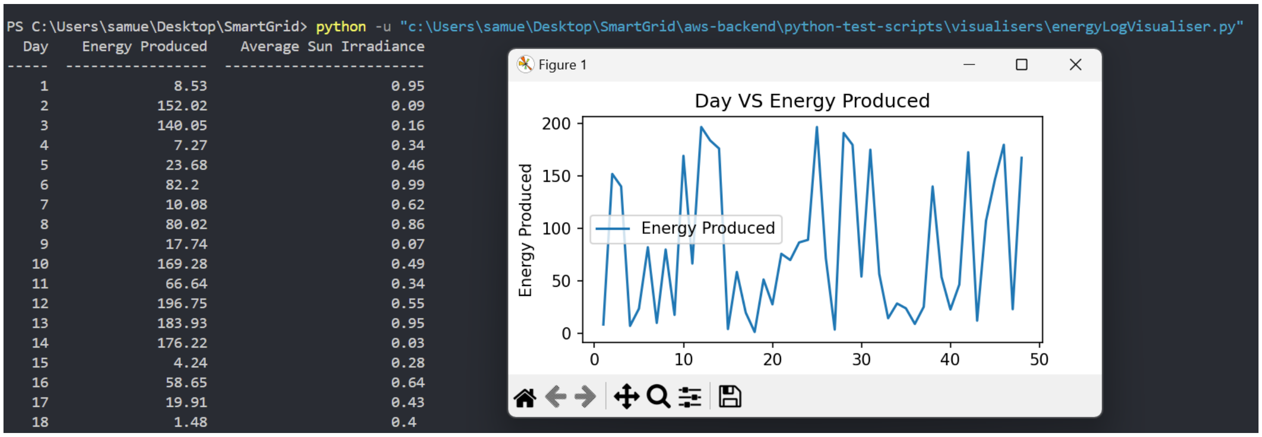
Status Code 200 OK

```
[
    {
        "energyProduced": 144.63,
        "dayID": 7,
        "avgSunIrradiance": 0.66
    },
    {
        "energyProduced": 88.37,
        "dayID": 47,
        "avgSunIrradiance": 0.63
    },
    {
        "energyProduced": 56.22,
        "dayID": 79,
        "avgSunIrradiance": 0.74
    },
    {
        "energyProduced": 83.32,
        "dayID": 69,
        "avgSunIrradiance": 0.87
    },
    {
        "energyProduced": 124.2,
        "dayID": 8,
        "avgSunIrradiance": 0.27
    },
    {
        "energyProduced": 110.65,
        "dayID": 62,
        "avgSunIrradiance": 0.21
    },
```

Figure 25: Postman API Testing

Following on from this, Postman testing was implemented to analyse the behaviour when sending incorrect formats of data or when rapidly making the API call multiple times. Postman also gave us a secondary timing source regarding AWS response times for fetching and sending data. Allowing us to cross-check for any timing discrepancies between the average time reported via Cloudwatch and those gathered from Postman. We found that on average the timing for a post request recorded using Cloudwatch logs for the server's true live time was between 70 ms and 90 ms, while on Postman they were between 100 ms and 130 ms instead. This difference is due to the connection time established between the Postman application and the regional AWS server and waiting for a response time. Cloudwatch fails to account for this as it only times the processing time on the actual server itself and not when the request was initially made and received by the end device.

The final round of testing involved a combination of unit testing and graph visualisation with generated data to simulate true usage conditions. Using the built-in unit testing, time and random libraries found in Python, we wrote a series of tests including a dummy data generator that simulated 5-second interval post requests to the database with an incrementing day counter and random values to be added for the remaining attributes. We chose 5-second interval testing as this demonstrated that the use of the API without data loss for 5-second intervals was possible if we decided to branch away to a different style of processing that involved sending data to the cloud in intervals at which the data was being updated in the 3rd party web server.



```
PS C:\Users\samue\Desktop\SmartGrid> python -u "c:\Users\samue\Desktop\SmartGrid\aws-backend\python-test-scripts\visualisers\energyLogVisualiser.py"
  Day    Energy Produced    Average Sun Irradiance
-----  -----------------  ------------------------
    1               8.53                      0.95
    2             152.02                      0.09
    3             140.05                      0.16
    4               7.27                      0.34
    5              23.68                      0.46
    6              82.2                       0.99
    7              10.08                      0.62
    8              80.02                      0.86
    9              17.74                      0.07
   10             169.28                      0.49
   11              66.64                      0.34
   12             196.75                      0.55
   13             183.93                      0.95
   14             176.22                      0.03
   15               4.24                      0.28
   16              58.65                      0.64
   17              19.91                      0.43
   18               1.48                      0.4
```

Figure 26: Simulated Data Generation

Testing the time needed to fetch, sort and filter the data retrieved was important for the final web app and scaling with the Time To Live feature. We had configured the expiration time for the Time To Live feature to be 6 months (converting to 180 days of data or 900 minutes) but our final web app only displayed a maximum of 90 days at one time to prevent overwhelming our viewer. This would mean we would need to access 180 points of data and sort it before returning it to the end user, which could lead to a significant perceived delay if not tested and configured appropriately. Using a combination of unit testing and graph visualisation we concluded that on average the time needed to fetch, filter and sort all the data was between 0.3 to 0.4 seconds. This was still within the desired human perceived range of 0.1 to 1 second for instantaneous and seamless response time on web pages. All in all, the testing was essential for macro optimizations to ensure a smooth and responsive experience for end users as well as to maintain the efficiency and reliability of data handling.

7.3 Evaluation of Hardware System

7.3.1 Power Losses

Bidirectional Switched-Mode Power Supplies (SMPSs)

The power losses in the bidirectional SMPSs primarily originate from the conduction and switching losses in the MOSFETs. Due to the synchronous rectification architecture, diode losses are minimal. Diodes are only used for free-wheeling during dead-time, approximately 200 ns, rather than as the main conducting switch, thus significantly reducing diode losses. There are also minor losses in the inductor due to its parasitic resistance and capacitance. When these SMPSs operate in closed-loop Boost mode, the architecture remains identical, and the notable losses are mainly from the MOSFETs.

Storage Capacitor
The storage capacitor, which is subject to continuous charging and discharging, incurs power losses due to its Equivalent Series Resistance (ESR), the dielectric material's polarization in response to the electric field, and the leakage current that flows through the dielectric material. The total loss from the storage capacitor is the cumulative effect of these factors.

Bus Connector
In the bus connector, the primary source of power loss is the electrical resistance of the conductors, which is proportional to the square of the current flowing through the bus connector and the resistance of the material. Additionally, there is some contact resistance at the connection points where the bus connector interfaces with other components, leading to further losses. The heat generated by these resistive and contact losses can increase the resistance of the bus connector material, thereby exacerbating power loss through thermal effects.

LED Loads
The power losses from the three LED loads, each with an individual Buck SMPS configured as an LED driver, are significant. These Buck SMPSs experience substantial conduction diode losses alongside the switching and conduction MOSFET losses. Additional losses are introduced by the control circuitry, where the control IC and other components in the feedback and control circuit draw some current. The LED loads themselves also contribute to power loss through junction losses, which include quantum efficiency loss and phosphor conversion loss, as well as thermal losses. Furthermore, optical losses due to inefficiencies in lenses, reflectors, and diffusers also play a role in the total power loss.

7.3.2 Inaccuracies and possible improvements

The overall results of the circuit measurements have been meticulously produced, as they are a hugely important part of our project, allowing us to continuously check the correct functioning of the circuit while helping us with the buying-selling decision making. With this said, there are certain inaccuracies that need to be mentioned. Our whole system is connected through cables with diameters large enough to generate losses, which despite being close to negligible, do make the measurements not totally accurate. Additionally, the power supplies used (EX355R POWER SUPPLY) can have an error of up to 6% in its displayed values, which also affects the overall accuracy in our system.

An area for improvement would be control system optimisation, employing integrator anti-windup or a suitable differentiator controller could have helped prevent the circuit from entering saturation and causing potential safety and/or operation problems. This could have been taken further using a more accurate model of the systems transfer function, and obtaining better tuned PID values.

7.3.3 Design FMEA abstract

Subassembly: Storage capacitor

We use a capacitor to store energy. The capacitor has a maximum charging voltage of 18V and we should not exceed this limit. For safety purposes we should not even enable the capacitor to be charged more than around 75-80% (at most) of its maximum capacity. Our aim is for overcharging to never happen, as it is vital to be in control of the current flow through the entire circuit at all times.

When directly connecting a 16V power supply to the 0.25F storage capacitor there is a chance the capacitor fails. In addition, the capacitor is also connected to a bus connector through a bidirectional SMPS that boosts the bus voltage (7V) towards the capacitor. The bidirectional SMPS is current controlled. A common rule of thumb is to choose a capacitor with a voltage rating 20-50% higher than the supplied voltage, and 2V/18V is around 11%. So from the beginning it can be seen why this might not be a good idea when trying to maximize energy storage. If we charge the capacitor too much, it will heat up and its performance will begin to be inconsistent, potentially leading to an unstable and unpredictable circuit. In extreme cases it can even get to a point where it blows up.

When a capacitor is charged over its rated voltage its leakage current increases. This can lead to higher power dissipation and unwanted current paths in the circuit, which can affect the performance of the circuit as a whole. The electric field across the dielectric in the capacitor increases too when it is overcharged. If this increase exceeds the maximum electric field that the dielectric can withstand (the dielectric strength) it causes the dielectric to fail and become conductive. This can potentially lead to the capacitor short circuiting. Even a slight overcharging can begin to degrade the dielectric material, leading to partial breakdown, which will create regions within the capacitor that are less effective at insulating, causing inconsistent capacitance and erratic behavior.

We were able to detect the failure mode by using the oscilloscope to display in its screen the reading of the voltage across the capacitor. This way we could clearly see when the capacitor voltage increased too much.

In order to prevent the capacitor from any kind of overcharge we firstly set the power supply voltage to 14V. This made sure that the power supply never charged the capacitor over that value and that the capacitor voltage rating was at least 22% higher than the maximum voltage from the power supply. Moreover, as the capacitor is also connected to the bus connector through a bidirectional SMPS, we changed the code that controls the SMPS in order to add a limit for how much voltage can be applied across the capacitor. These changes make the probabilities of the capacitor blowing up practically null and also dramatically decrease the possibilities of the capacitor slightly overcharging and therefore interrupting the correct functioning of the circuit.

Following the improvements made on the circuit the RPN decreased significantly (improved RPN = 14). It is now a perfectly safe number to carry on with our circuit design and keep optimizing the performance of our system. The full Design FMEA is shown in the Appendix.

7.4 Evaluation of Software System

The software system encapsulates the requirements quite well. Our system has been thoroughly tested and design choices carefully planned out. There were also additional features added that could be mentioned especially in how the algorithm and web design is carried out. Our algorithm runs completely in sync with the hardware model, not including any model that learnt in advance. Hence this makes the timing of the system alot more challenging, yet we devised a strategy to counteract timing delays in every subsystem, covering all edge-cases. The web design was also made to be really user friendly, with multiple different interacting components and alternating tips based on the users spendings.

However, there is a long way to go to make the software system an industry standard. A prime example of this is when deploying the WebApp, it would no longer be possible to send information via a JSON file locally and instead would be sent through websocket or an external API call. These are all important changes that can be made in the future to ensure a better fully functional and industry standard system

8 Conclusion

By referring to the testing results above, it can be concluded that the project design was very successful. Through the combination of expertise in both the hardware and software aspects of the project, along with effective collaboration and communication skills, the group has managed to integrate all the various components and subsystems into one working design. On the hardware side, the operation of the SMPS was fully utilized and acted as the backbone of the entire circuit. On the software side, machine learning functions, algorithms and the transmission of data to the hardware and the web application were all successfully implemented within the specified project time frame. Furthermore, all the project specifications were satisfied without any use of the allocated project budget.

Yet, it can be shown that there exists limitations to the current design of the circuit. Particularly on the hardware as the present configuration of the grid is unable to differentiate between the buying operation from the grid and the charging of the flywheel. If the project were to be taken to the next step, the use of switches or relays as well as a more efficient controller strategy with a central controller that acts as a master for the entire circuit could provide further improvements.

Nevertheless, the project itself is only a simulation and is not fully applicable in a real-life smart grid system. However, it acts as an excellent example to show how these systems can be created by decomposing what seems to be an extremely complex design, into various subsystems which can be designed independently and later integrated methodically into one whole system. At the same time, it demonstrates a solid foundation of key engineering principles and scientific concepts by the students.

Appendix

Appendix A

Design FMEA

Project: Smart Grid

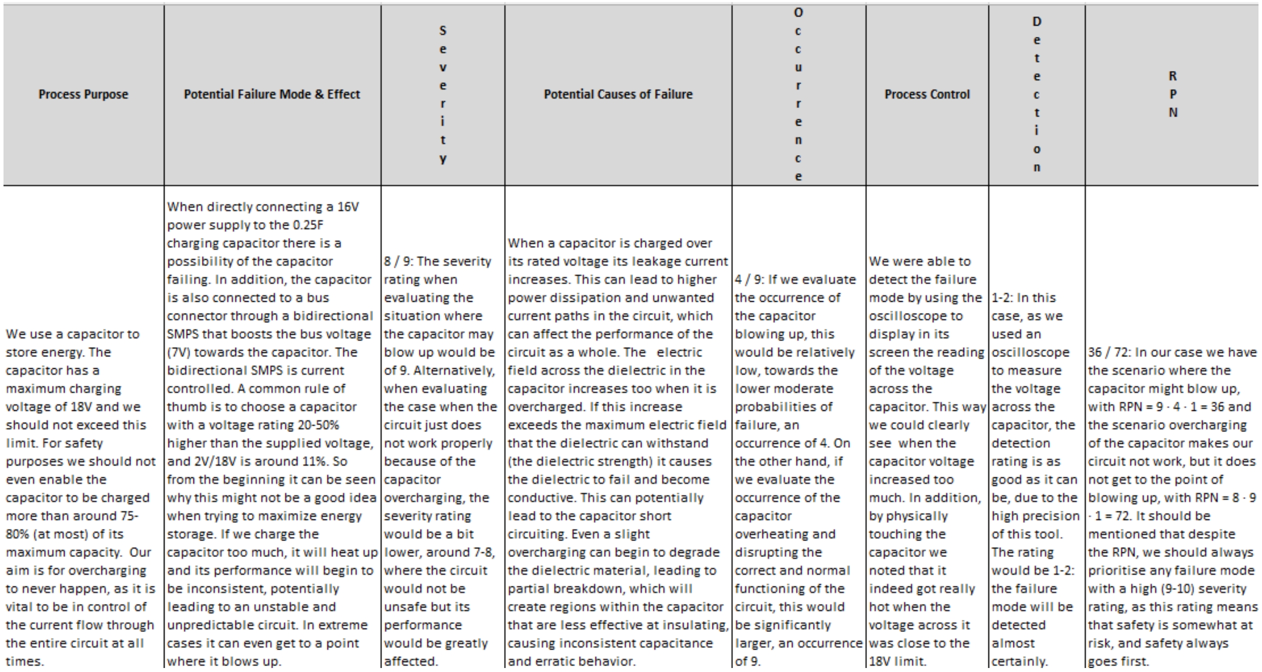
Sub assembly: Storage capacitor

Prepared by: Justin Lam, Eddie Moualek and Lucas Lasanta

Approved by: Samuel Khoo, Anson Chin and Ilan Iwumbwe

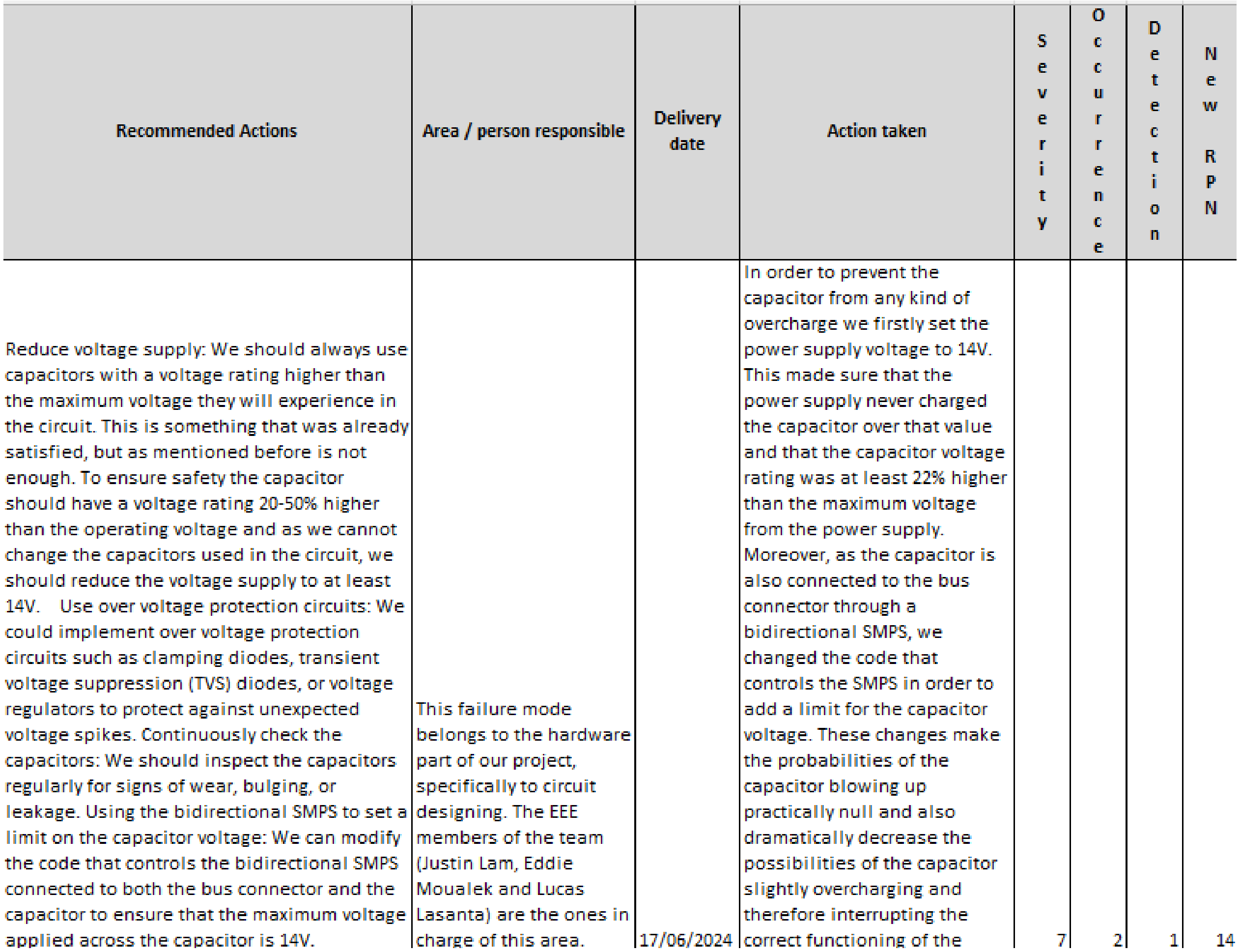
Version: 2/2

Date: 12/06/2024



Process Purpose	Potential Failure Mode & Effect	Severity	Potential Causes of Failure	Occurence	Process Control	Detection	RPN
We use a capacitor to store energy. The capacitor has a maximum charging voltage of 18V and we should not exceed this limit. For safety purposes we should not even enable the capacitor to be charged more than around 75-80% (at most) of its maximum capacity. Our aim is for overcharging to never happen, as it is vital to be in control of the current flow through the entire circuit at all times.	When directly connecting a 16V power supply to the 0.25F charging capacitor there is a possibility of the capacitor failing. In addition, the capacitor is also connected to a bus connector through a bidirectional SMPS that boosts the bus voltage (7V) towards the capacitor. The bidirectional SMPS is current controlled. A common rule of thumb is to choose a capacitor with a voltage rating 20-50% higher than the supplied voltage, and 2V/18V is around 11%. So from the beginning it can be seen why this might not be a good idea when trying to maximize energy storage. If we charge the capacitor too much, it will heat up and its performance will begin to be inconsistent, potentially leading to an unstable and unpredictable circuit. In extreme cases it can even get to a point where it blows up.	8 / 9: The severity rating when evaluating the situation where the capacitor may blow up would be of 9. Alternatively, when evaluating the case when the circuit just does not work properly because of the capacitor overcharging, the severity rating would be a bit lower, around 7-8, where the circuit would not be unsafe but its performance would be greatly affected.	When a capacitor is charged over its rated voltage its leakage current increases. This can lead to higher power dissipation and unwanted current paths in the circuit, which can affect the performance of the circuit as a whole. The electric field across the dielectric in the capacitor increases too when it is overcharged. If this increase exceeds the maximum electric field that the dielectric can withstand (the dielectric strength) it causes the dielectric to fail and become conductive. This can potentially lead to the capacitor short circuiting. Even a slight overcharging can begin to degrade the dielectric material, leading to partial breakdown, which will create regions within the capacitor that are less effective at insulating, causing inconsistent capacitance and erratic behavior.	4 / 9: If we evaluate the occurrence of the capacitor blowing up, this would be relatively low, towards the lower moderate probabilities of failure, an occurrence of 4. On the other hand, if we evaluate the occurrence of the capacitor overheating and disrupting the correct and normal functioning of the circuit, this would be significantly larger, an occurrence of 9.	We were able to detect the failure mode by using the oscilloscope to display in its screen the reading of the voltage across the capacitor. This way we could clearly see when the capacitor voltage increased too much. In addition, by physically touching the capacitor we noted that it indeed got really hot when the voltage across it was close to the 18V limit.	1-2: In this case, as we used an oscilloscope to measure the voltage across the capacitor, the detection rating is as good as it can be, due to the high precision of this tool. The rating would be 1-2: the failure mode will be detected almost certainly.	36 / 72: In our case we have the scenario where the capacitor might blow up, with RPN = 9 · 4 · 1 = 36 and the scenario overcharging of the capacitor makes our circuit not work, but it does not get to the point of blowing up, with RPN = 8 · 9 · 1 = 72. It should be mentioned that despite the RPN, we should always prioritise any failure mode with a high (9-10) severity rating, as this rating means that safety is somewhat at risk, and safety always goes first.

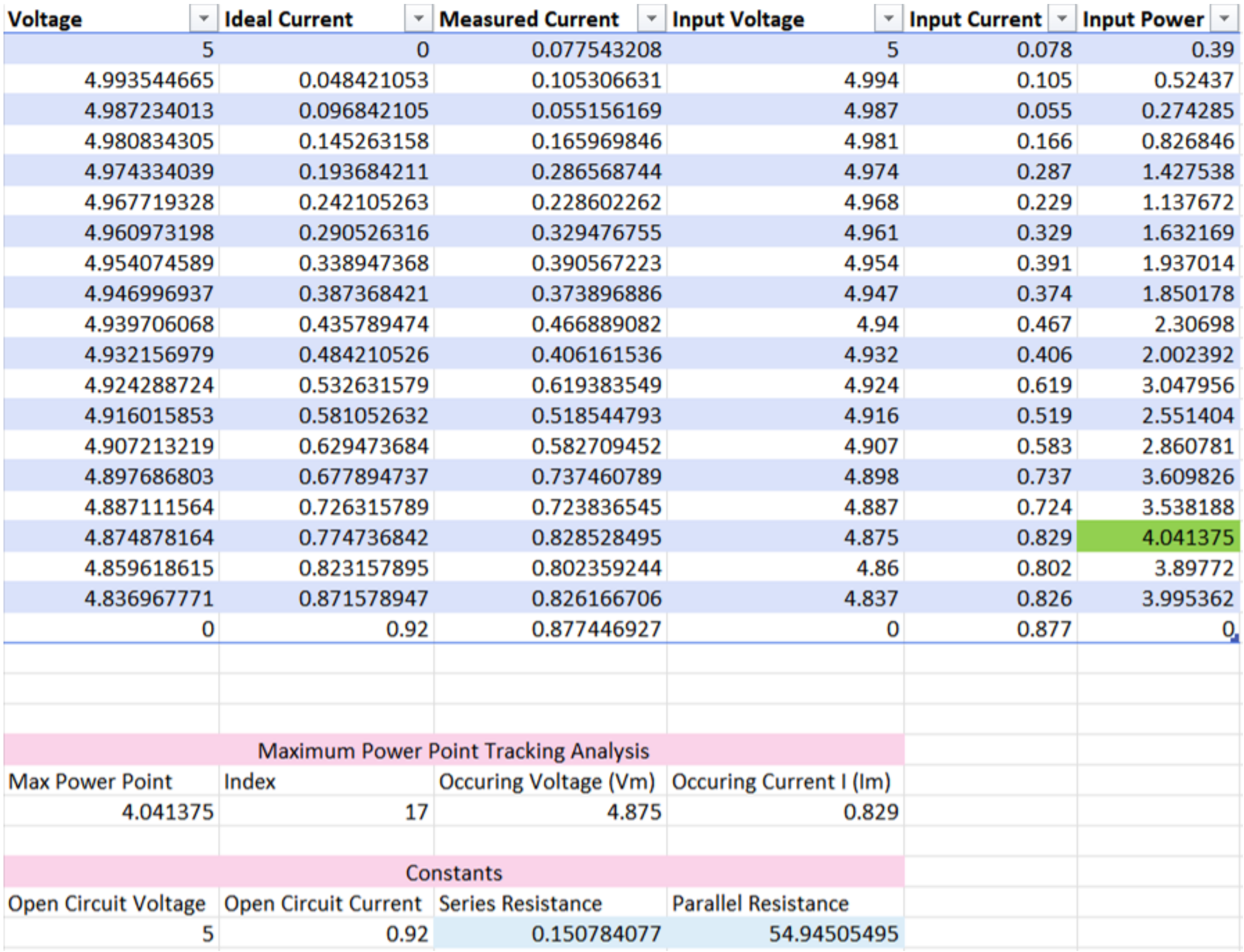
Figure 27: Design FMEA 1



Recommended Actions	Area / person responsible	Delivery date	Action taken	Severity	Occurrence	Detection	New RPN
Reduce voltage supply: We should always use capacitors with a voltage rating higher than the maximum voltage they will experience in the circuit. This is something that was already satisfied, but as mentioned before is not enough. To ensure safety the capacitor should have a voltage rating 20-50% higher than the operating voltage and as we cannot change the capacitors used in the circuit, we should reduce the voltage supply to at least 14V. Use over voltage protection circuits: We could implement over voltage protection circuits such as clamping diodes, transient voltage suppression (TVS) diodes, or voltage regulators to protect against unexpected voltage spikes. Continuously check the capacitors: We should inspect the capacitors regularly for signs of wear, bulging, or leakage. Using the bidirectional SMPS to set a limit on the capacitor voltage: We can modify the code that controls the bidirectional SMPS connected to both the bus connector and the capacitor to ensure that the maximum voltage applied across the capacitor is 14V.	This failure mode belongs to the hardware part of our project, specifically to circuit designing. The EEE members of the team (Justin Lam, Eddie Moualek and Lucas Lasanta) are the ones in charge of this area.	17/06/2024	In order to prevent the capacitor from any kind of overcharge we firstly set the power supply voltage to 14V. This made sure that the power supply never charged the capacitor over that value and that the capacitor voltage rating was at least 22% higher than the maximum voltage from the power supply. Moreover, as the capacitor is also connected to the bus connector through a bidirectional SMPS, we changed the code that controls the SMPS in order to add a limit for the capacitor voltage. These changes make the probabilities of the capacitor blowing up practically null and also dramatically decrease the possibilities of the capacitor slightly overcharging and therefore interrupting the correct functioning of the	7	2	1	14

Figure 28: Design FMEA 2

Appendix B



Voltage	Ideal Current	Measured Current	Input Voltage	Input Current	Input Power
5	0	0.077543208	5	0.078	0.39
4.993544665	0.048421053	0.105306631	4.994	0.105	0.52437
4.987234013	0.096842105	0.055156169	4.987	0.055	0.274285
4.980834305	0.145263158	0.165969846	4.981	0.166	0.826846
4.974334039	0.193684211	0.286568744	4.974	0.287	1.427538
4.967719328	0.242105263	0.228602262	4.968	0.229	1.137672
4.960973198	0.290526316	0.329476755	4.961	0.329	1.632169
4.954074589	0.338947368	0.390567223	4.954	0.391	1.937014
4.946996937	0.387368421	0.373896886	4.947	0.374	1.850178
4.939706068	0.435789474	0.466889082	4.94	0.467	2.30698
4.932156979	0.484210526	0.406161536	4.932	0.406	2.002392
4.924288724	0.532631579	0.619383549	4.924	0.619	3.047956
4.916015853	0.581052632	0.518544793	4.916	0.519	2.551404
4.907213219	0.629473684	0.582709452	4.907	0.583	2.860781
4.897686803	0.677894737	0.737460789	4.898	0.737	3.609826
4.887111564	0.726315789	0.723836545	4.887	0.724	3.538188
4.874878164	0.774736842	0.828528495	4.875	0.829	4.041375
4.859618615	0.823157895	0.802359244	4.86	0.802	3.89772
4.836967771	0.871578947	0.826166706	4.837	0.826	3.995362
0	0.92	0.877446927	0	0.877	0

Maximum Power Point Tracking Analysis			
Max Power Point	Index	Occuring Voltage (Vm)	Occuring Current I (Im)
4.041375	17	4.875	0.829

Constants			
Open Circuit Voltage	Open Circuit Current	Series Resistance	Parallel Resistance
5	0.92	0.150784077	54.94505495

Figure 29: MPPT Table

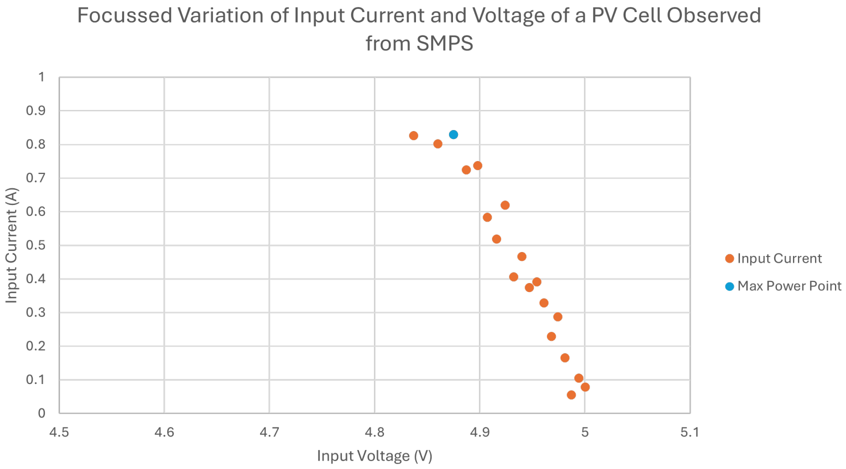


Figure 30: MPPT Graph

8.1 References

[1] MathWorks, ”Boost Converter Voltage Control,” MathWorks, [Online]. Available: https://uk.mathworks.com/help/sps/ug/boost-converter-voltage-control.html. [Accessed: 16-Jun-2024].