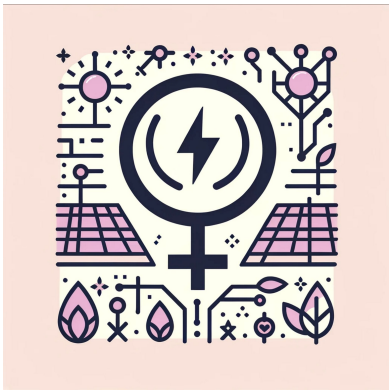


Smart Grid Project Report

The Slay Girlies

Group 18

Dhruv Ruda	CID: 0231321	EEE
Jennifer Emezie	CID: 02246573	EEE
Rares Yousif	CID:02266292	EEE
Arundhathi Pasquereau	CID: 02207669	EEE
Sophie Jayson	CID: 02254802	EIE
Adam El Jaafari	CID: 02204948	EIE

IMPERIAL

Imperial College London

ELEC50015 - Electronics Design Project 2

Marker: Philip Clemow — June 2024

Smart Grid Repository

Abstract

This year for the 2024 group projects each team was required to design and implement a system made up of modular elements and interfaces between them. The aim of the smart grid was to build an energy management system that connected a home to a smart grid. The use of a photovoltaic array and super-capacitors were provided for energy generation and storage. The system was required to balance energy supply and demand and use forecasting to minimise the cost of imported energy from the grid.

The project was designed to draw on different areas of the EEE and EIE curricula as well as independent research. General problem solving techniques and innovative designs were required to complete the project.

After this report We conclude that a smart grid reduces the cost of energy. This model can be used as to reduce energy importation and could be applied to real world buildings.

Word Count:

Contents

1 Introduction

1.1 Project Requirements

The smart grid is an energy management system designed to optimise energy production, storage, and consumption through embedded control systems. The system includes key functions:

- Use a photovoltaic array to generate electrical energy.
- Store energy in supercapacitors for later use.
- Buy and sell energy to and from the grid.
- Draw variable demand specified by a third-party web-server.
- Control how much energy is sold and bought to minimise the costs of importing energy.
- Store historic data on a data server and display data on a user interface on a web-server.

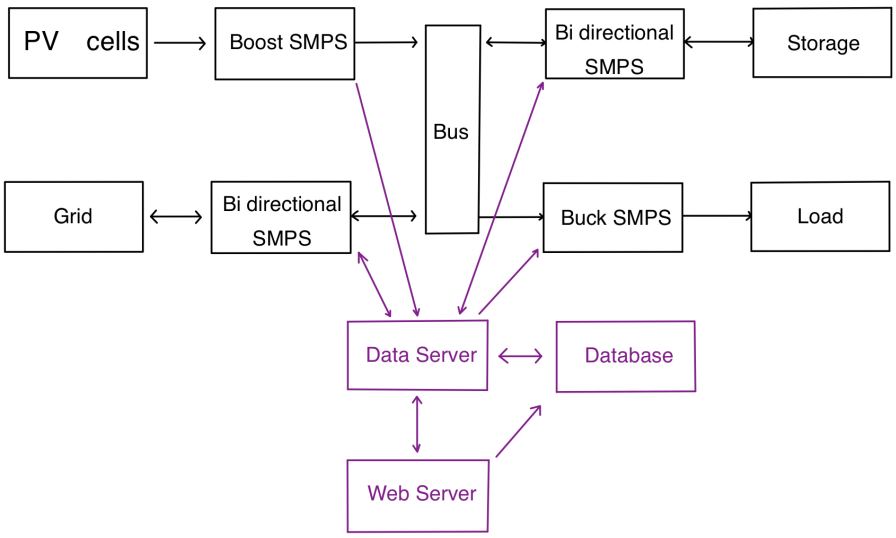


Figure 1: Smart Grid Block diagram

Each component was connected to a bus through a Switch-Mode Power Supply(SMPS), with a Wi-Fi-enabled Raspberry Pi Pico, to control the SMPS's power flow and facilitate communication with a web-server. Shown in figure 1

1.2 Team Management

Organisation was a key part in the completion of this group project. The team prioritised time management to ensure the completion of the project. The initial steps to start the project

was the creation of the Gantt Chart (seen in Appendix A). This was a visible plan that was accessible to all team members. It was used to keep track of 5 fundamental tasks - Research, Prototype, Presentation, Report, and the Demo.

In addition to this, the team ensured that communication was kept between members. A lot of the components required the EE Laboratory hence the team was able to work easily alongside one another. Some team members were able to work remotely due to the software side of the project. As a result group meetings were held every 3-5 days. These meetings acted as a check in point between members and ensured everyone was on track.

When dividing the project amongst members it felt appropriate to divide accordingly to the components within the smart grid. After an initial consultation, each team member felt more comfortable choosing a component due to the clarification of the project requirements. Additional organisations plans were in place such as shared folders, a team chat, and a GitHub repository which kept all work accessible to each other.

Team Member - Component	Description
Dhruv - PV Cell	This member look lead on the production element of the smart grid. They ensured that maximal energy was extracted from the PV array under various conditions, by developing numerous control algorithms to maximise production.
Rares Yousif - The Grid	This team member implemented the external Grid functionality for energy import and export, and researched ways to improve voltage control for the grid. Technical advisor, booked consultations
Jennifer Emezie - Storage	This member worked on the storage component of the smart grid. They were responsible for handling any excess energy in the system. Additionally this member was entrusted with the planning of both the presentation and report. They ensured both were professional and informative as well as grammatical and technical. Finally, they were in charge of team organisation and time management. The completed key tasks such as the creation of the Gantt chart and calling team meetings.
Arundhathi Pasquereau - Load	This team member worked on the load and the loop shaping to control it. They ensured that the correct power was drawn to the load and that the controller remained stable.
Sophie Jayson -Data Handling	This team member worked on the data server and the connection between hardware and software, creating a server to communicate and a database to store said information, ensuring data was sent timely can correctly across the entire system. Additionally this teammate aided in the report inorder to ensure spelling and grammatical correctness.
Adam El Jaafari - Web Server and Algorithms	This team member worked on creating a user interface that satisfies all the provided GitHub project requirements and designed an algorithm,which preforms better than the naive approach, that predicts the trend in export prices, deciding whether the smart grid should import or export energy, given the current demand, and designed another algorithm, with the help of Sophie Jayson, that ensures that deferrable demands are satisfied in an efficient manner.

2 Implementation

This section will discuss the design of each element and it's integration to the project

2.1 PV Cells

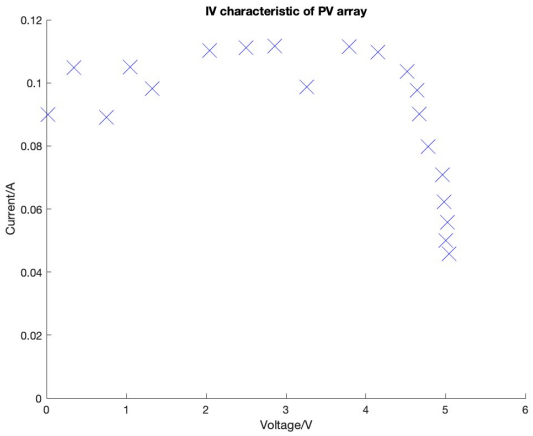
The Photo Voltaic (PV) cell sub-system consists of 4 PV cells connected in parallel to generate power. The operating current and voltage must be controlled to extract maximal power from the PV array, for any irradiance and temperature. This power must be measured and displayed on a user interface, then transferred to the remaining subsystems of the Smart Grid via the connecting bus.

2.1.1 Requirements

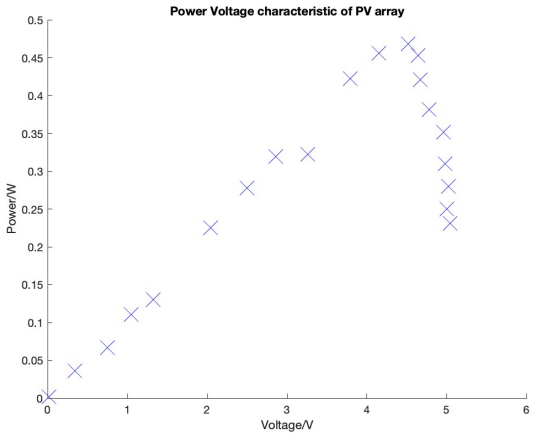
- Generate the maximum amount of power from the PV array, under a variety of conditions.
- Transfer the power generated from the PV array, to the system.

2.1.2 Research

The first step in the development of the PV subsystem was characterising the PV array. During this step, the array was modelled as a power supply with series and parallel resistors. It was better to model the non-ideal qualities of the PV array operating in constant voltage and constant current modes. The team characterised the PV array under natural light on an overcast day, whilst in the shade.If the array was characterised under direct sunlight, there would be more current flow, and therefore more power. The team found that the fluctuations caused by environmental factors, such as clouds, heavily impacted the consistency of the irradiance. Subsequently, this would lead to abnormal characteristic curves, leading to difficulties in modelling the array.



(a) Shows the measured IV characteristic of the PV array



(b) Shows the measured PV characteristic of the PV array

Figure 2: Shows the behaviour of the PV array under overcast conditions

From the characterisation of the PV array, and using the characteristic curve from GitHub [1], the team modelled the PV array for use within the laboratory setting. Their aim was to collect reliable and reproducible results. The team calculated resistor values using equations 10 and 2 which were used to create the model array with series resistance. They decided to use, R_S, of 0.6Ω and parallel resistance, R_P, of 41.7Ω, using $V_M \approx 4.5V$ and $I_M \approx 800mA$.

$$R_P = \frac{V_O}{I_S - I_M} \tag{1}$$

$$R_S = \frac{V_O - V_M}{I_M} \tag{2}$$

For the Maximum Power Point Tracker(MPPT), the team opted for a hybrid algorithm with a shoot for calculated operating range. Additionally, a Perturb and Observe(PAO) algorithm would vary the current flowing out of the PV array to control the operating point. The team chose this approach as the PAO model has a fast response and long-term stability. Alongside its easy to implement it had reliability and was the standard approach for PV and solar cells [3].

2.1.3 Final Design

Using R_S, R_L and the SMPS with a fixed load resistor, the team was able to characterise the model PV as shown in figure 3. The team modelled different irradiances, by using varying current limits as our research showed that the shape of the IV characteristic did not change.

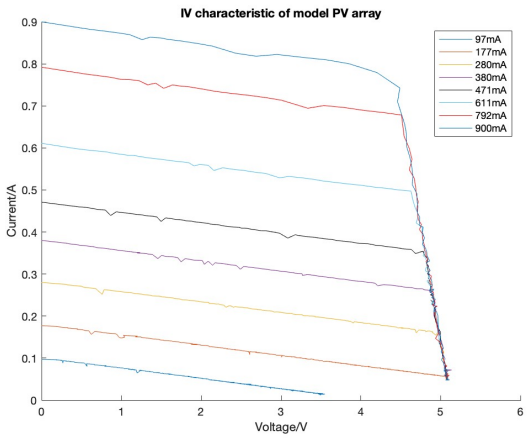
However, the height of the curve varied with irradiance [2]. Using this knowledge, and the data from characterising the model PV, the team used a straight-line approximation for all voltages less than V_M. At that point, the short circuit current was calculated using equation 3.

$$I_S = I_L - 0.02V_{in} \tag{3}$$

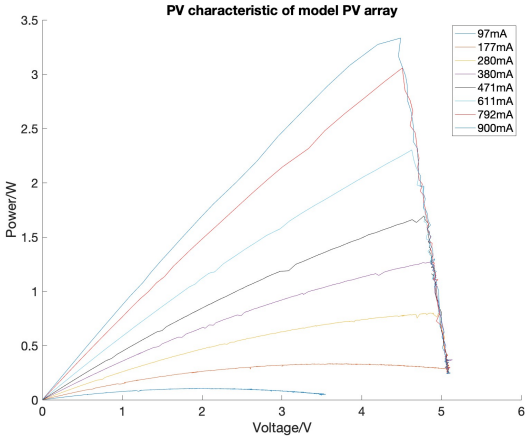
$$I_M = 0.8972I_S - 0.002 \tag{4}$$

$$I_L = -1.03V_{in} + 4.8 \tag{5}$$

The team estimated the value of the current where maximal power is produced, I_M (as shown in equation 4) This point was used as the estimate to begin finding the true maximum power. The team created a straight line approximation, 5, for $V_{in} > V_M$. This allowed the team to check if the maximum power point had been overshot. If this condition was satisfied, the operating current was reduced until $V_{in} < V_M$, to recalculate the operating point.



(a) Shows the measured IV characteristic of the modelled PV array



(b) Shows the measured PV characteristic of the model PV array

Figure 3: Shows the behaviour of the mdoel PV array under varying model irradiences

A large amount of the design phase was spent on creating an efficient maximum power point tracker for the control of power flow through the SMPS. To do this the team chose to control the operating point by using a current reference, I_{ref}, which was then passed to a control system, using proportional, K_i, and integral gains, K_p. This ensured the PV array was operating at this value.

The reference controller can be broken down into 2 sections, creating an estimate for I_{ref}, and the PAO for adjusting I_{ref} when near the maximum power point, as shown by Figure 4. This approach was chosen, as traditional PAO systems can take time to enter a region of maximal power. However, this ensured operation was within a narrow region on either side of V_M.

To ensure that the time of operation in the maximal power region was maximised, the team implemented a temporary aggressive controller, so this region could be operated as quickly as possible. This increased overall output power over long periods but had a significantly greater effect when first starting up, as the difference in power is at its maximum.

The team tested a PAO with a fixed step size, which led to long rise times or large oscillations in the steady-state response. A PAO with a variable step size, that was proportional to the difference in power between observations, was then tested which was promising for high irradiance conditions as the difference in power was large, for low irradiances settling time was significantly longer, causing the rejection of this version of PAO.

With the current control approach, the SMPS was configured to be a boost. The bus was connected to port B and the PV array connected to port A, as the bus voltage, which was chosen to be 15V, is greater than the max PV voltage as shown by figure 3. The direction of current flow was confined to ensure that current would flow from port B to port A. It limited the range of duty cycles to prevent flow in the wrong direction.

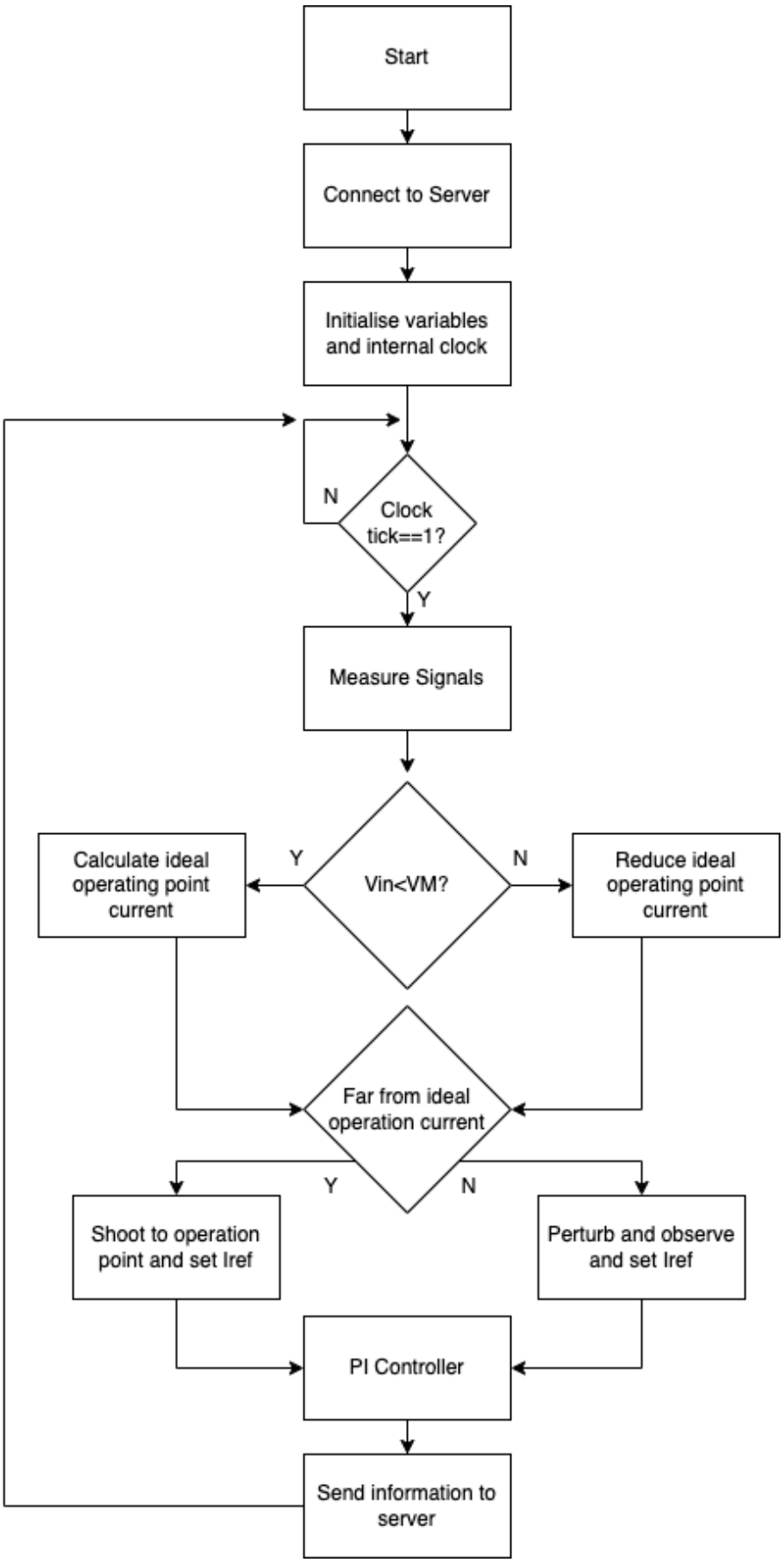


Figure 4: Shows a flowchart, of the implemented MPPT code

2.1.4 Implementation

The PAO controller was implemented using a $K_P = 10$ and a fixed step size of 1mA, as provided stability without compromising speed. Speed could have been increased, at the cost of oscillations, but it was found that the difference in the current operating point and the ideal was not large, so increasing, step-size and K_P was not worth the trade-off. The SMPS controller code had limits of $-1A < I_{ref} < -0.1mA$ as the maximum expected current was not larger than 920mA, which is the maximum current that can be produced, according to the datasheet (as seen in Appendix B), ensured no current flowed into the PV array.

The aggressive controller was implemented using $K_P = 20$ and with $I_{ref} = 1$ or $I_{ref} = -2$, as we found this provided a fast response with little overshoot. We removed the I_{ref} limits from the PAO, as the single aim for this section of the controller was speed, and increasing the I_{ref} limits greatly increased response time. We also found that despite the increase in I_{ref} limits, the current flowing always satisfied $1A > I_{in} > 0.1mA$.

The I_{ref} value from the controllers was then passed to a PI controller, with $K_I = 50$ and K_P set from the I_{ref} controllers, as this allowed for the speed and stability to be prioritised where needed.

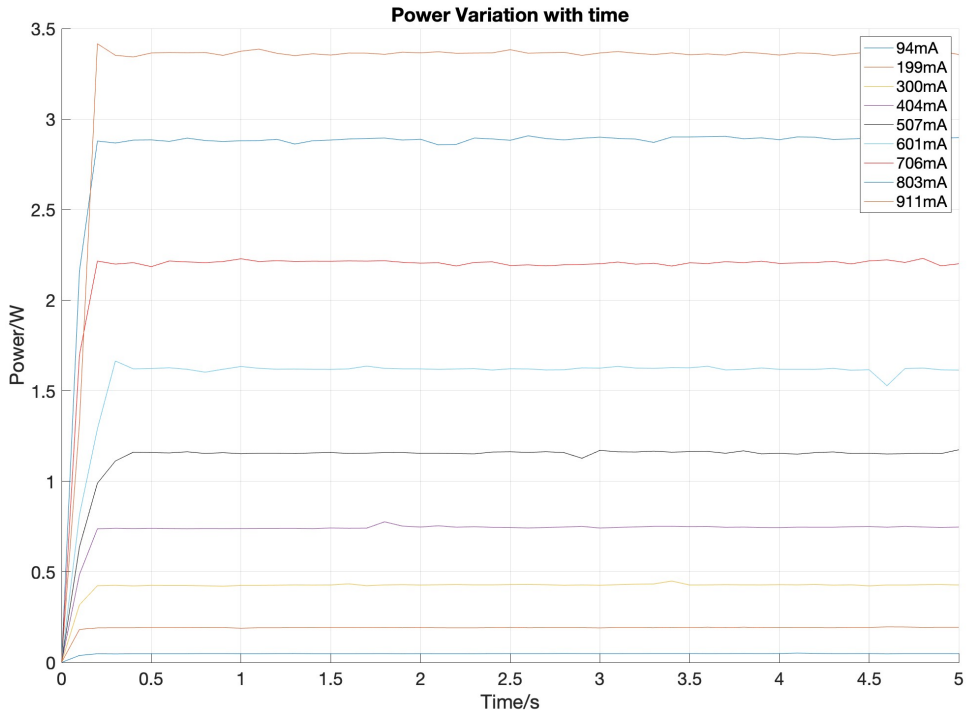


Figure 5: Shows the power variation of the PV cell, whilst under the MPPT, at various simulated irradiances

To test the effectiveness of the MPPT, the tracker was run with a fixed maximum current to simulate a steady irradiance. Data was recorded at an interval of 0.1s, for approximately 10s, however, the initial 5s were the priority as this, is how often the irradiance would change. Figure 5 shows the variation of power against time, while running the MPPT, for a range of irradiances. The rise time was consistently observed to be less than 0.5s for all irradiances, so the MPPT will find the maximum power, within the 5s before irradiance changes as shown by Figure 6.

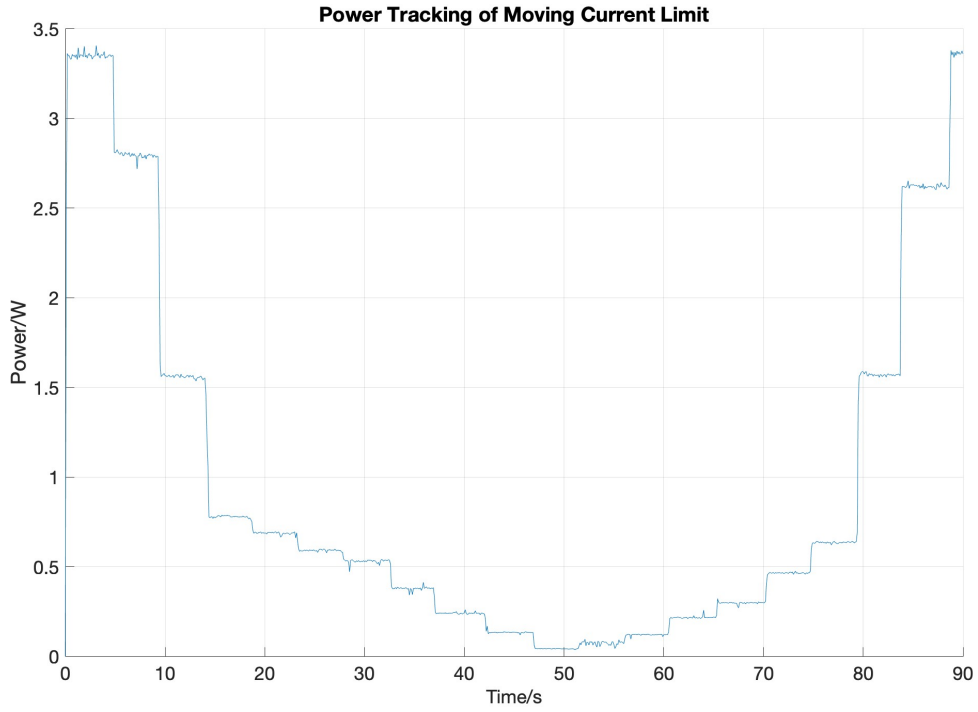


Figure 6: Shows the power variation of the PV cell, whilst under the MPPT, through a varying current limit to confirm ability to track the maximum power

2.2 The Grid

The Grid is chosen to regulate the bus voltage because it will have the ability to continuously source or sink current to and from its power supply reliably at all times without disruptions to maintain the constant bus voltage. The Grid must ensure there is little to no oscillation of the bus voltage to ensure stability and minimal power losses to load. Voltage control must be designed to reach equilibrium fast and quickly respond to changes in voltage. Then it must communicate with the web-server to send its energy import and export values with minimal delay.

2.2.1 Requirements

Use minimal current from external power supply, accurately send data fast to the web server with minimum delay, stable and fast voltage control for bus voltage, sensitive to changes in bus voltage.

2.2.2 Research

The main focus of Grid is to regulate bus voltage with minimal current delivery to minimise power needed to buy, as well as fast response to changes in voltage in bus. In order to do this, sliding mode control has been researched for the grid. According to Ben K, Sahbani A and Benrejeb M (2011) Published by InTech, sliding mode control can provide good control on fluctuations in a non-linear design, where a sliding surface is chosen to change the control input when it lies on the sliding surface. In the text they use the sliding surface: " S = ke + $\dot{e}$ ", where k is a constant to be tuned and the control input will have two components: a continuous and a discontinuous component.[5] To determine stability they use Lyapunov's function:

$$"V = \frac{1}{2}S^2 \text{ (Lyapunov, 1892)"[5]} \tag{6}$$

and find its derivative and then they find its range of values for which " $\dot{V} < 0$ " to ensure stability, leading to the discontinuous control input needed to be equal the sign of the sliding surface to ensure stability. From Ben K, Sahbani A and Benrejeb M (2011) Published by InTech, this leads to the discontinuous component of control input to include the sign of the sliding surface S. When S is positive, discontinuous control input must be positive and when S

is negative, so the discontinous control input. Therefore, using this sliding surface to control the duty cycle of the SMPS for the grid, the team initially chose a PI controller to be the continuous control input that will regulate the voltage at every point in time, and a discontinuous control that will be proportional to the sliding surface that will be added on or subtracted whether the sliding surface is positive and negative. The team implemented the condition as follows: if the sliding surface > 0.1, the duty cycle must increase as the voltage sensed by the Raspberry Pi Pico is smaller than the reference voltage (the bus voltage) and the error is great enough for the sliding surface to act to increase the voltage. Similarly, if the sliding surface < -0.1, the duty cycle will be decreased to reduce the voltage detected by the Raspberry Pi Pico because it is higher than the reference voltage.

Based on concepts from sliding mode control, the team chose initially a tuned PI controller together with a discontinuous control input changing the duty cycle based on the proportional error and the integral error of the PI controller. Inspired from Sliding Mode control, the following function is used to help the PI controller reach the reference voltage (the bus voltage) faster:

$$slide = v_{err}^2 + v_{errint}^2 - 2 \cdot verr \cdot v_{errint} \tag{7}$$

where v_{err} and v_{errint} are the proportional voltage error and integral voltage error respectively. If the voltage error is small this slide function will converge faster to 0 and will have minimal effect on the duty cycle, so that the PI controller can regulate the voltage without assistance. However, when the error is large the slide function will cause a significant change to the duty cycle which will cause a high rise or fall to reach the the bus voltage. The stability of this initial slide 7 chosen can be found intuitively, because it is directly related to the proportional and integral error. When the slide function is added to the pwm based on whether the slide function is positive or negative, you would respectively add or subtract from the pwm to change the duty cycle accordingly. For small errors, the slide function will converge to 0 and based on the if statements for which the slide function is activated to change pwm, that is: if slide < -0.1 decrease pwm, and if slide > 0.1, increase pwm using directly the slide function from 7. This means that the slide in 7 will never be used for small errors and the stable tuned PI controller will control the voltage only to ensure no oscillation added from the slide in 7. As error increases, the slide function will then act to increase or decrease the pwm with higher

impact than the PI controller when the error is higher due to the two higher power terms in the slide function. The output of slide function is then saturated to appropriate limits to ensure the pwm never goes unbounded.

The team then tried this new approach using the slide function in 7 and compared it with the slide mode control method learnt from Ben K, Sahbani A and Benrejeb M (2011) by testing each of the designs implemented in Python and plotting the voltage control against time:

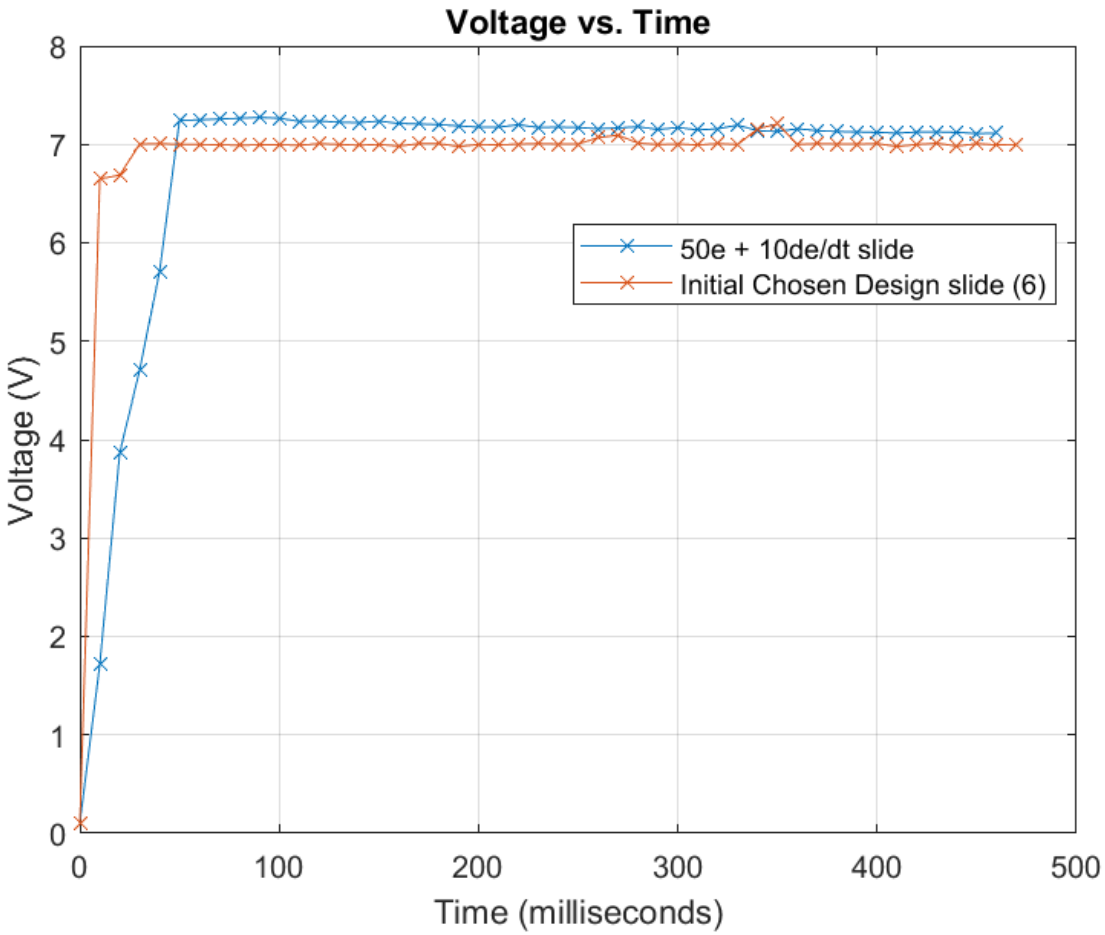


Figure 7: Shows the two choices of controllers regulating a reference voltage of 7 as a test

The slide in 7 has unexpected oscillations as shown in figure above which will still make bus voltage variable but around the equilibrium value of chosen reference voltage. The implementation of both choices is in the GitHub Grid folder as slideoptions.py file.

2.2.3 Final Design

Based on whether the current is positive or negative through the SMPS, the energy imported or exported will be calculated and summed up over a period of 5 seconds then sent to the web-

server using socket communication in Python. For the PI controller, proportional constant kp was chosen as 20, and the integral constant ki was chosen as 30, giving the open loop transfer function as:

$$C = \frac{20s + 30}{s} \tag{8}$$

As for the sliding surface, the following was chosen:

$$\text{slide} = 50e + 10\dot{e} \tag{9}$$

Where e is the voltage error. This slide equation in 9 has been chosen over the one in 7 because it has been found later that the initial slide chosen 7 has unpredictable undershoots to small voltages after some time after the desired voltage is reached, as it has trouble drawing in current from the power supply. Hence the final choice 9 is more reliable and can guarantee stability at all times, for a trade off in speed and predicted initial small overshoot of approximately 3.6% of desired voltage. Chosen Bus voltage as 15V which will be set for the final demo because a high enough bus voltage will increase efficiency of power delivered as well as. For analysis of voltage regulators implemented in the next section, a reference voltage of 7V has been chosen.

2.2.4 Implementation

The open loop transfer function of the controller has a zero in left hand plane and a pole in zero. The phase approximation shows that the phase will be always between -90 and 0 degrees, indicating phase margin stability. Shown below is the approximation bode plots of the PI controller in open loop as well as exact bode plots:

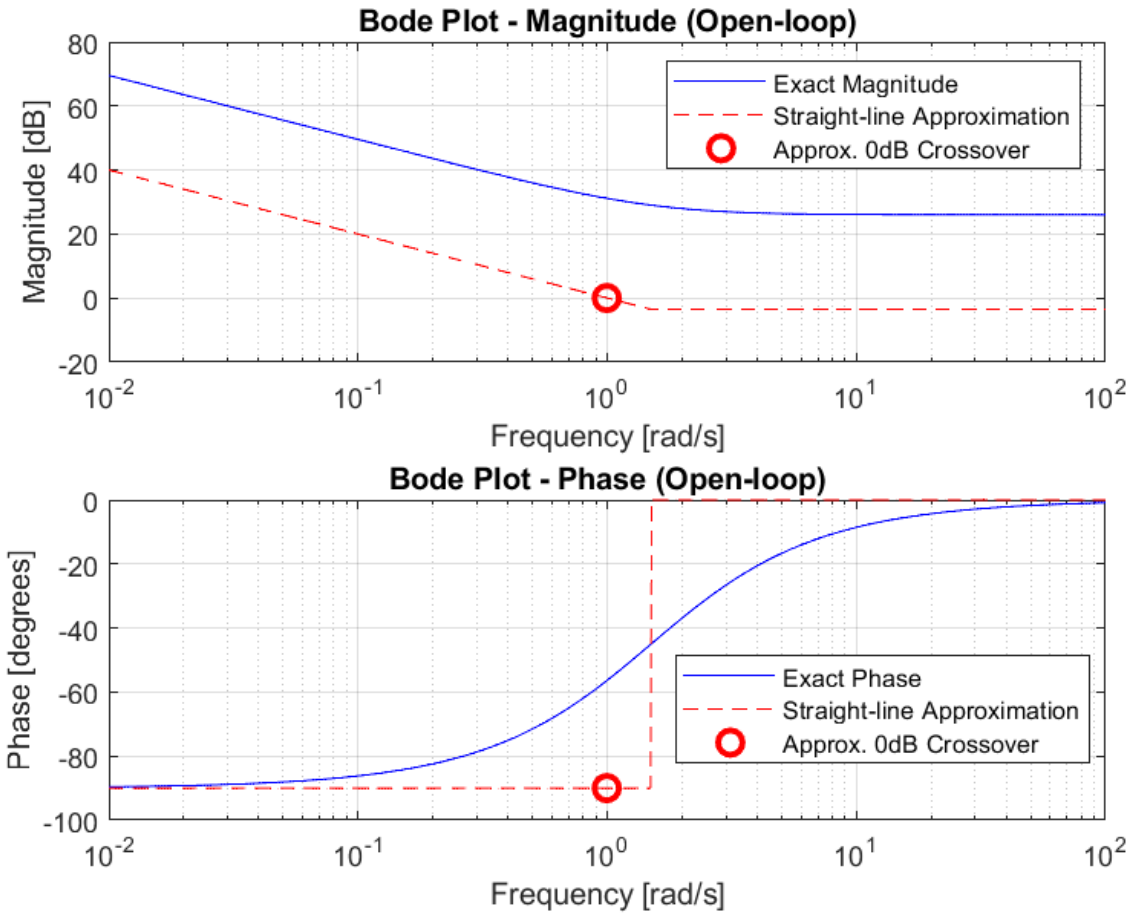


Figure 8: Shows the straight line approximations in open loop for PI controller and 0 dB crossover

From the straight line approximation, the phase margin can be estimated as 90 degrees (-90 - (-180)) which indicates stability. In closed loop the following expression is obtained:

$$\mathrm{GC} = \frac{20s + 30}{21s + 30} \tag{10}$$

one negative pole and one negative zero indicate closed loop is indeed stable. As for the slide function chosen the following expression has been selected:

$$\mathrm{slide} = 50e + 10\dot{e} \tag{11}$$

Below is a flowchart summarising how the code is executed:

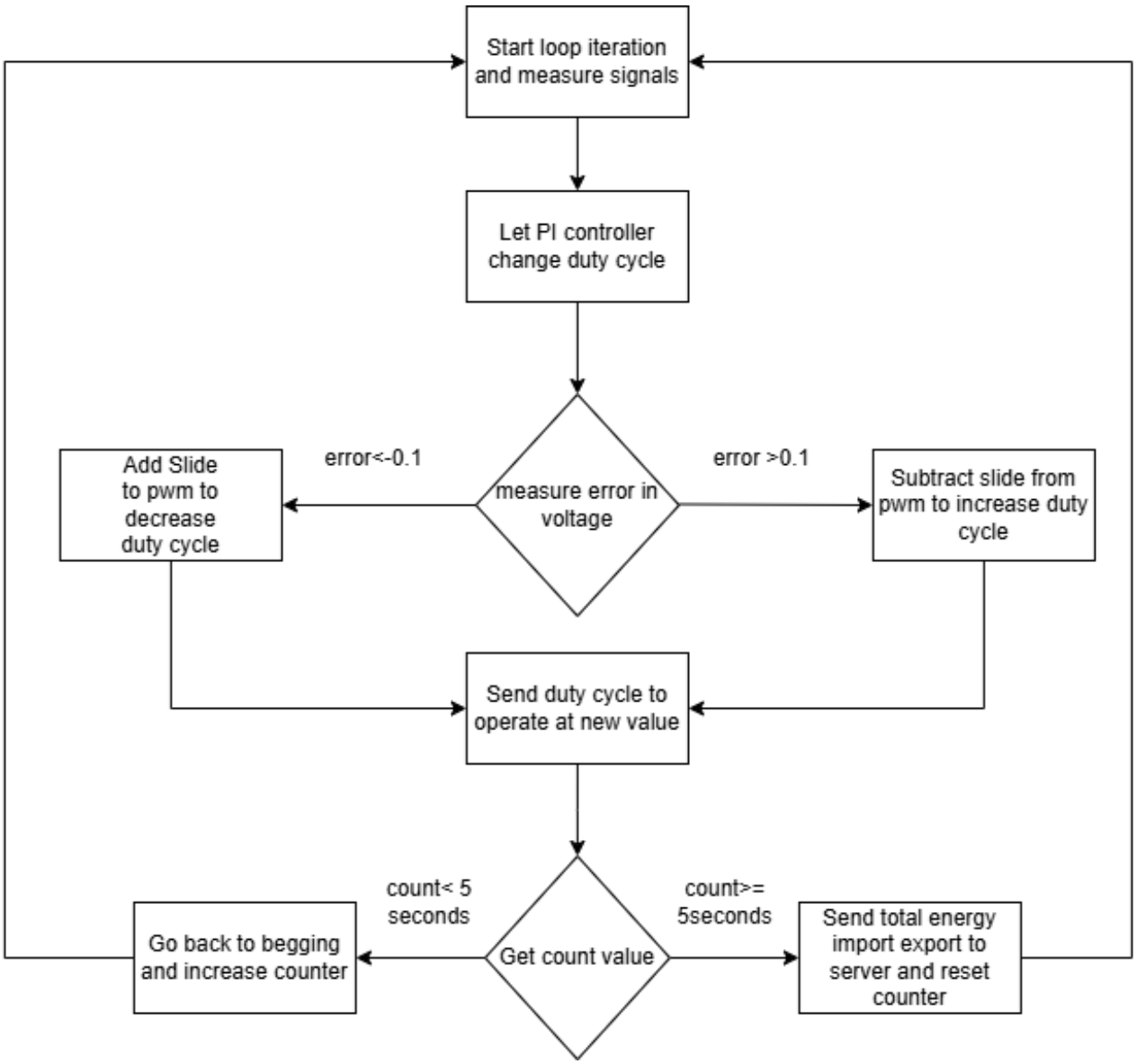


Figure 9: Shows the summary implementation of how grid code works

The full implemented code is on GitHub in the Grid folder.

2.3 Failure Modes Effect Analysis (FMEA)

The primary purpose of FMEA is to anticipate potential failures in a system, process, or product and mitigate them before they occur. By systematically identifying failure modes, assessing their effects, and prioritizing them based on their severity, occurrence, and detectability, FMEA helped the design to improve its reliability and cost due to power losses caused by small errors in the Grid, remove potential crashes in the code which could stop the functionality of the whole system, and improve the safety of the Grid design. One example based on the FMEA completed worksheet at the end of the report in the appendix that helped the design was identifying in the terminal a crash in the Raspberry Pi Pico due to memory allocation. This helped detect the cause of failure is due to an array not being able to clean itself at the end of the loop unexpectedly, which then caused memory in the Raspberry Pi Pico to be completely used up. This was then improved by replacing the array with a running total of the data and then stored and updated in a variable and then setting it to 0 after sending it instead of sending all the data every 0.1 seconds to the server in an array which was the potential cause of the error. More examples of FMEA included in the completed sheet at the end of the report in Appendix C.

2.4 Storage

This section will discus the storage component within the smart grid. This component aided the success of the smart grid as it provided the ability to store any excess energy. Additionally, it gave the ability to access that energy at a later time when desired.

2.4.1 Requirements

In this system, the power supplied by the PV cell always powered the load. When radiance levels were low, the power from PV Cell would be insufficient to meet the demand of the load. Ideally, during these conditions, energy would be bought from the grid. However, if the cost of that energy was high, energy would be discharged from storage instead (given that there was energy stored prior to this).

The energy transferred to storage would always be the energy that was no longer required from the load. This could be from a surplus of power from the PV cell (due to high levels of radiance) or energy bought from the grid when costs are low. Energy would always go to the load first then to storage.

Throughout the project the storage was modelled as two 0.25F capacitor. This meant there were practical restrictions, specifically a current limit of 0.38A and voltage limit of 18V (as shown in appendix D) Additionally, as stated in the earlier sections, the limit of the SMPS board with port A and B restricted to 17.5V.Initially the team had modelled the bus voltage at 17V to maximise energy storage space .Hence all tests in this section were conducted in the condition of 17 V at port A. Due to further testing explained in section 2.2, 15V was the most optimal voltage for the grid. When integrated into the full system, the storage SMPS ran at 15V causing a new maximum energy level of 54J.

2.4.2 Research

For energy to be stored and discharged, it was clear that a bidirectional SMPS would be required. This meant looking back at the ELEC50012 Power module specifically Lab 6 Bidirectional operation.

The bidirectional SMPS was current controlled. It allowed the movement of current in both

directions by :

1. Adjusted the PWM duty cycle based on the desired operation mode (buck or boost).
2. Used the INA219 sensor to measure current and ensured a safe operating limits.
3. Implemented control logic (open-loop or closed-loop) to regulate the power flow efficiently.[4]

For the purpose of this project, a closed-loop control logic was chosen as it provided a more stability with the feedback loop. The way it worked was a PI controller was used to regulate the current. The reference current i_{ref} was set based on the potentiometer value (further referred to as *vpot*).Then the current error i_{err} and its integral i_{errint} were calculated. Subsequently, the PI controller output i_{piout} determined the PWM duty cycle. Finally, the duty cycle was inverted and outputted to the PWM.

Appropriate adjustments were made to the original $SMPS_{2024}Bidirectional.py$ lab code. Firstly, adjusting the current limit to range at a saturation point between $\pm 0.3A$ due to the earlier stated current limits of the capacitors. The next step was to decide how to control the forward and reverse bias of the current when desired. This would give the ability to charge and discharge the capacitor. A way to quantify this increase and decrease was also beneficial. Furthermore, a zero current was need when no energy flow was required.

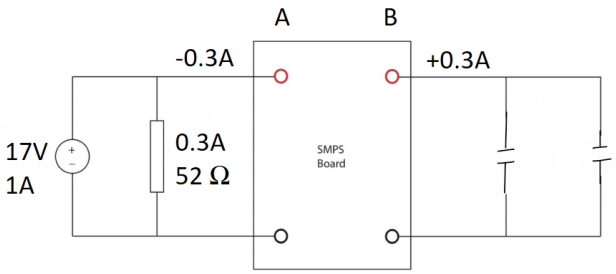


Figure 10: Circuit Diagram of Capacitor Connection

Figure 10 demonstrates the circuit design for capacitor testing. The team decided it was best to use the *vpot* to control the charging, discharging and storing of energy in the capacitor. Firstly, to track the energy stored in the capacitor the equation

$$E = \frac{1}{2}CV^2 \tag{12}$$

was used. Hence, the voltage at port B (further referred to as V_b) needed to be recorded from the SMPS board.

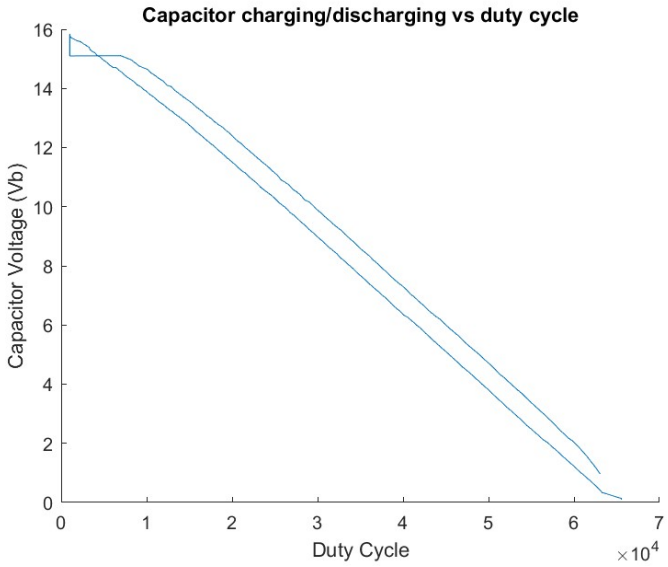


Figure 11: Variation of capacitor voltage in reference to duty cycle

After testing it was evident that at high duty cycles (min *vpot*) the current would be in reverse bias. This caused a decrease in V_b to stabilise the negative current which subsequently caused a discharge of energy from the capacitor. This variation in duty cycle and voltage can be seen in figure 11.

As a result the next changes to the code was to set *vpot* to 0V for discharging , 3.3V for charging and 1.66V to give a 0A. As a means to quantify this energy charge and discharge, the code calculated the new energy level of the capacitor based off of its current energy level and required amount from the sever. Equation 12 was used to calculate a corresponding voltage for this new energy level (referred to in code as V_c). $Vpot$ was set to charge or discharge till V_b reached V_c then *vpot* was set to 1.66V to keep this energy level.

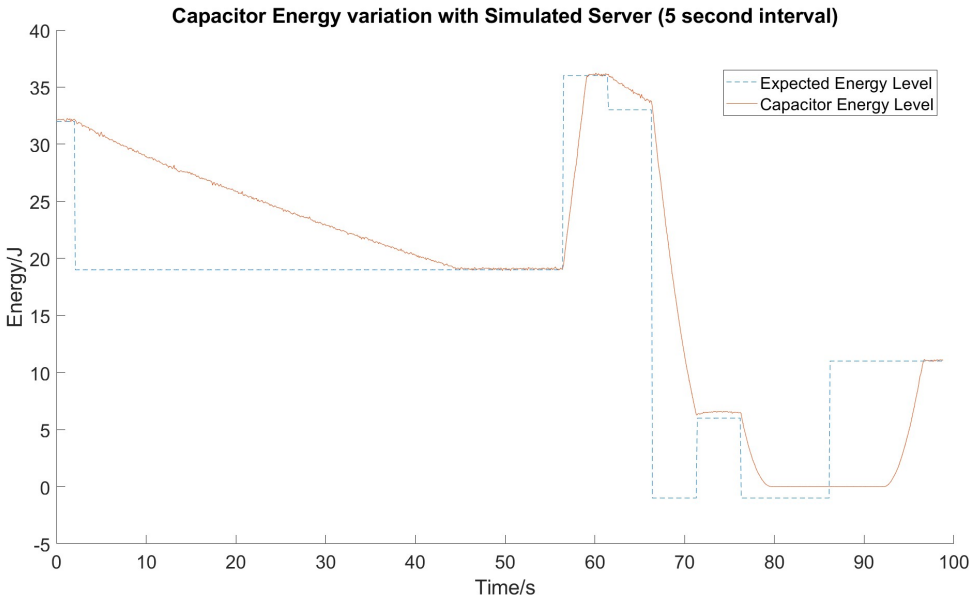


Figure 12: 5s interval testing of capacitance energy level

Figure 12 shows the initial testing results when data was being read by the SMPS board every 5 seconds. The potentiometer (that previously controlled *vpot*) was now simulating the server. Positive or negative values of energy represent the energy that should be stored or discharged by the capacitors respectively. Since the web-server updated every 5s it seemed appropriate to see that the capacitors would read the desired energy level at these intervals.

Evidently the capacitor was able to reach the required levels of energy. There was successful storage however the discharge time of the capacitor was too long. At first it was assumed that this was a physical limitation of the capacitance and the discharge time could not be improved. However it was discovered that this was a coding error that was quickly corrected. Even though this data was ran with incorrect code, this set of data highlighted important areas of concerns. Firstly, at around 70 seconds, the expected level of the capacitance energy was lower than 0. This was due to the modelled server asking for too much energy . As a result, code was been implement on the server side to not ask for (send negative values) when capacitance storage is low. On the flip side, energy would not be asked when the capacitors were at a max energy level. Another key area flagged was allowing the capacitance to reach the desired level before there was a change energy level (new energy sent by the server). As a result, the sending and receiving commands in the code were moved to only occur once the desired energy level was reached.

2.4.3 Final Design

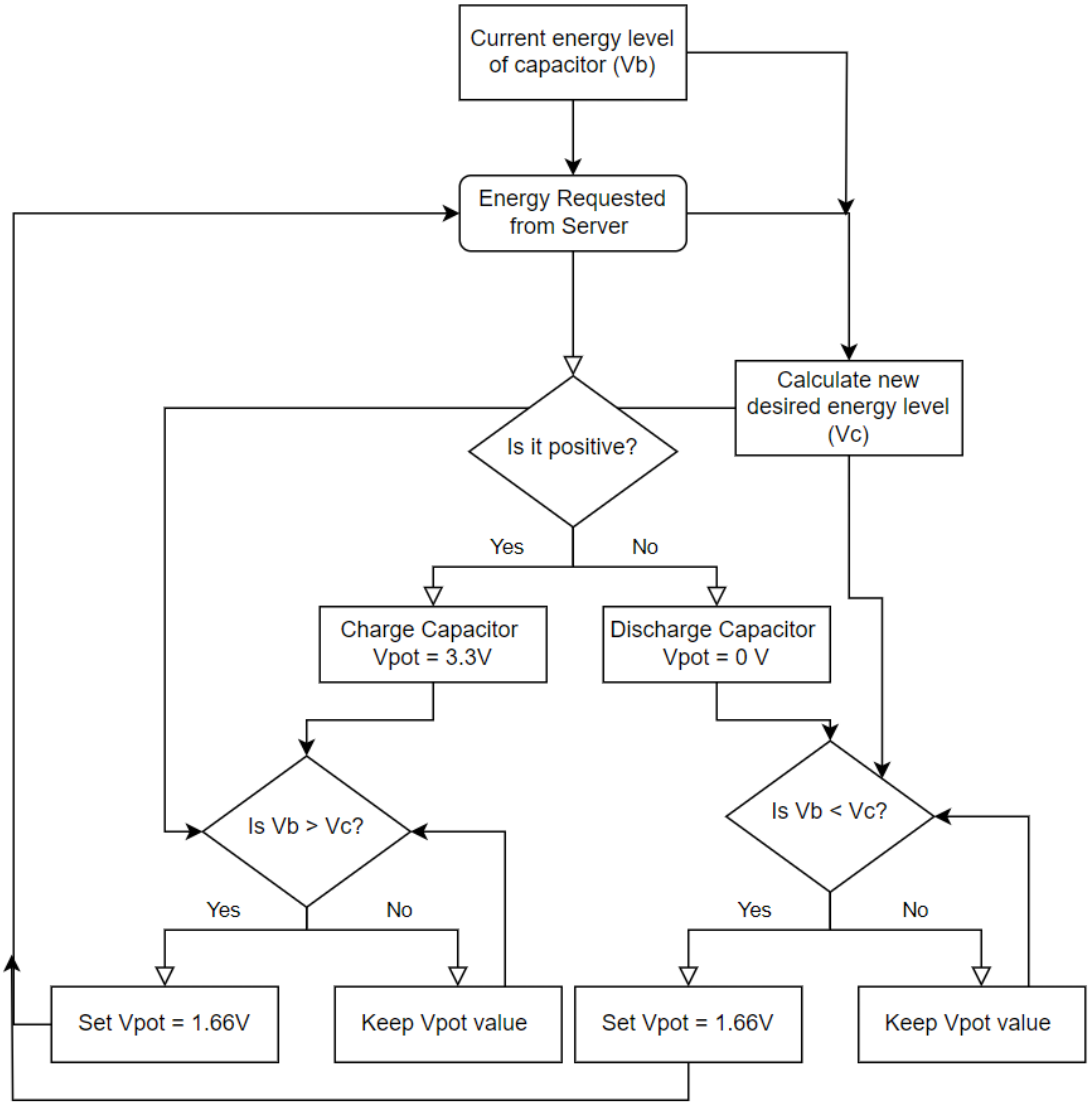


Figure 13: Flow Diagram of Capacitor Logic

The final logic can be demonstrated in figure 13. It is important to note that the new desired energy level was calculated by the addition of the current capacitor energy level with the requested energy from the server. Equation 12 allowed the energy to be controlled in terms of voltage.

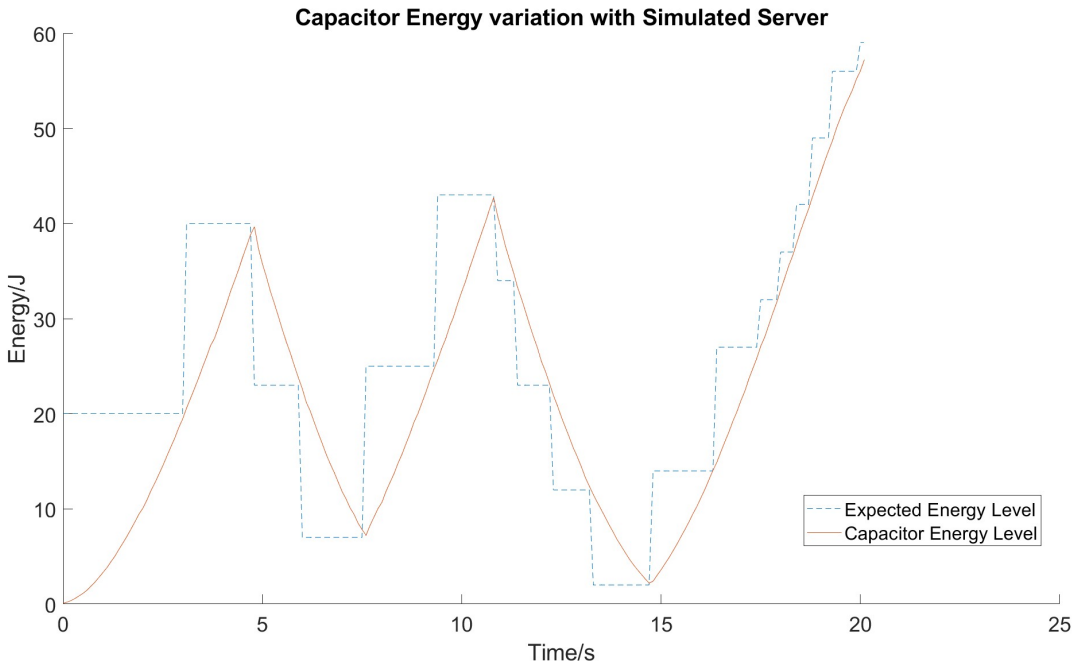


Figure 14: Energy variation of capacitor with a simulated server

This final logic was once again tested with a simulated server. As shown in figure 14 the capacitor responded perfectly to the modelled server.Both its charging and discharging time was quicker and would be able to handle the 5 second updates of the server.Even if not, the new code of collecting a new energy reading when desired energy was reaction is was still in place. This addition allowed stability and between readings.

2.4.4 Implementation

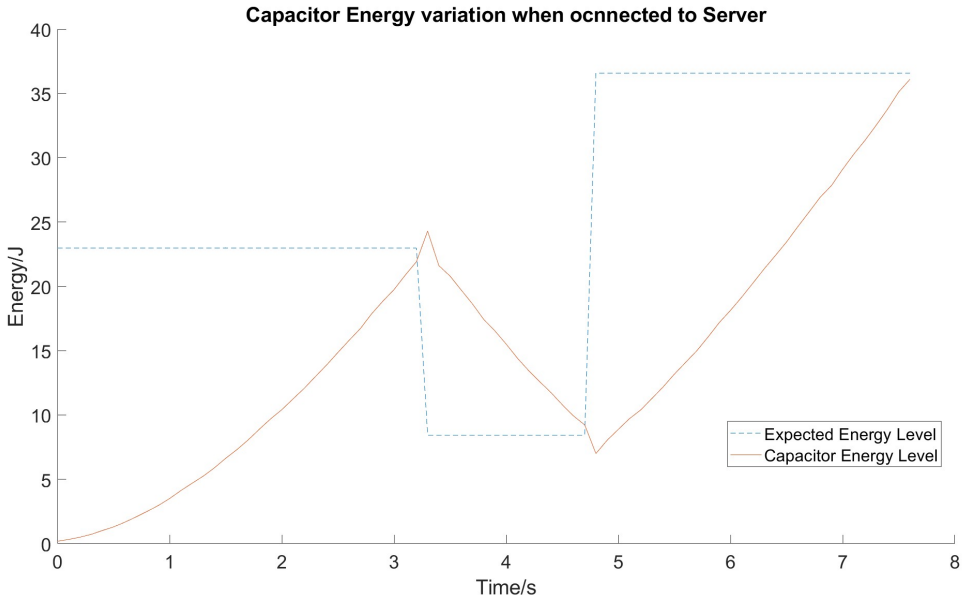


Figure 15: Energy variation of capacitor when connected to server

The final steps to prove the success of the storage unit was its connection to the server.

Figure 15 demonstrates the successful response of the capacitor to the server. The storage and discharge of the capacitors was smooth and fast. there is a slight over shoot in the expected values however this was expected due the code condition '$V_b > V_c$' and '$V_b < V_c$' This was kept as it is easier to implement a greater than and less than condition instead of equal to. Especially as the requested energy is a float variable type.

2.5 The Load

The Load consisted of four LEDs connected to a Buck SMPS which could each draw a maximum of 1W of power. The total power drawn by these LEDs should set by the algorithm, so they would be able to communicate with the Data Server (section 2.7) and receive the target power output and follow it closely. As the target power could change as fast as every 5s, the team aimed to implement a control system that achieves its target power and stabilises much faster than this.

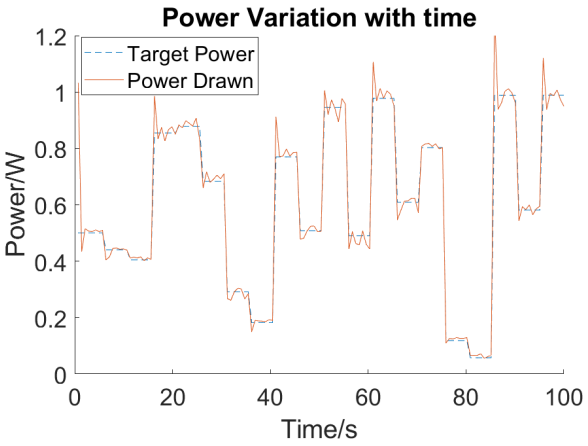
2.5.1 Research

The PID controller included in the setup for the pico has the 3 standard variable parameters: Proportional Gain , K_p; Integral Gain , K_i; and Differential Gain, K_d. As briefly mentioned in section 2.2.4, one has to tweak the different parameters to optimise the response of the controller. If K_p is too high, the controller will oscillate more and become unstable. A higher K_i increases the speed at which the target power is reached but also increases the overshoot [14]. The differential gain would normally help reduce overshoot, however adding non-zero K_d caused instability and broke the controller completely. Thus the team settled with a PI controller.

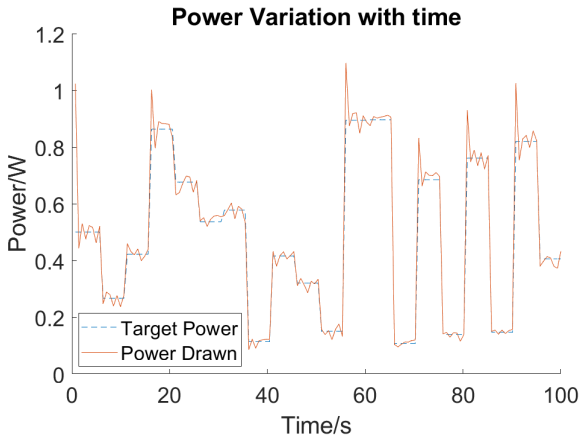
K_p	K_i	K_d	Average Error (W)	Maximum Overshoot
0.02	5	0	0.00019504	0.203782
10	5	0	0.000219881	0.2065935
0.02	7	0	0.000106003	0.2989254

Table 1: PID Parameter Testing

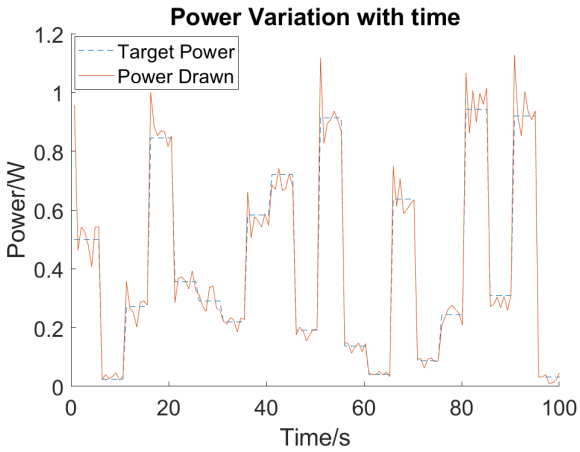
As seen in Fig. 16c, increasing the proportional gain in practice caused more oscillation so a lower proportional gain would be ideal. Increasing K_i also increases the overshoot (Tab. 1), so a lower integral gain is also more ideal.



(a) $K_p = 0.02$, $K_i = 7$, $K_d = 0$



(b) $K_p = 0.02$, $K_i = 5$, $K_d = 0$



(c) $K_p = 10$, $K_i = 5$, $K_d = 0$

2.5.2 Final Design

The final design included a closed loop PI control system which controls the output current through the LEDs. The team used trial and error to adjust the parameters of the PI controller. The ideal values found are in Tab. 2.

K_p	K_i
0.02	5

Table 2: PI controller Parameters

Although the controller does not follow the setpoint exactly and there is some oscillation, it was found that increasing K_i was not useful because it increased the overshoot and the decrease in average error was not worth it.

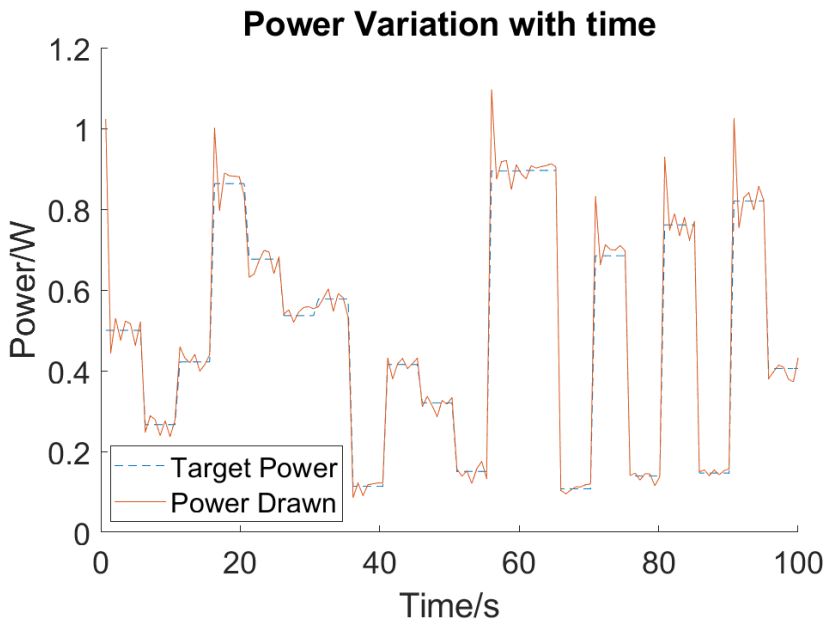


Figure 17: Final Controller Response

Finally, the team chose that instead of each LED drawing 1/4 of the power, instead LED1 would draw the first Watt of power, LED2 the second and so on. This is so that the person running the system can visually guess how much power is being drawn. If 1 LED is on the load is drawing 0-1 Watt, if 2 are on the load is drawing 1-2 Watts, etc.

2.5.3 Implementation

In order for the LED code to be as simple as possible, the team decided that they should simply, using a few if statements on each LED, draw their respective Watt of power. For example if we look at LED2, if the target power is greater than 2W, LED2 draws 1W, if the target power is less than 1w, LED2 draws 0W, otherwise it draws the target power minus 1. All the other LEDs work in a similar way but with different bounds so that the whole range of target powers (0 to 4 W) is covered. The LEDs all constantly connect to the server and demand a target power. The PI controlled designed previously makes sure that the actual power drawn is as similar to the target power as possible.

To improve the load system in the future, one could add a warning signal to the server if the load is unable to draw its target power which would help with troubleshooting the system as a whole.

2.6 Web Server

2.6.1 Requirements

The Web-Server must contain the front end which is used to display the UI showing *"current and historic energy stores in the system"*, and a back end that handles all HTTP requests to the third party web-server, whilst providing API endpoints for the web-server front end, to access the external third party webs server values, and also providing GET and POST endpoints for the algorithm and smart grid data server to update with any changes in the external web-server values, and to send data from the teams smart grid to the the teams website, to be displayed.

2.6.2 Research

The starting point was to decide what front and back-end frameworks to use for the teams web-server, React.js, and Flask where chosen due to their low barrier of entry and our teams familiarity with python and typescript. A CORS error was initially encountered when trying to fetch the third-party web-server.A CORS error is when "users attempt to access shared resources" [6] and so the solution to avoid this error is for our flask server to allow CORS and this allowed our front end to display the latest values from the external web-server, as seen in figure c. The Flask API runs on local host at port 4000, whenever the front end fetches the latest values from the Flask API, a HTTP GET request is sent from the react web-page to the API endpoints, there are 3 HTTP requests in the home page, one at the API endpoint *"/webdata"* to display all third party web-server information to the website, another HTTP GET is sent to *"/forwardcapdata"*, to display the latest value of energy that is currently in our capacitors and finally a HTTP GET request to the *"/forwardgriddata"* which allows us to display the current Imported and Exported value of energy from our Grid. Through testing, the total time required for the 3 HTTP GET requests was measured to range from 300ms-600ms,since the website is meant to keep up to data with the external web-server which updates every 5s (the time duration of a tick), the web-server team is well within this range and our website updates seamlessly.

2.6.3 Final Design

The final design of the web-server has 3 web-pages:

1. The Home page: This is where the sun irradiance, instantaneous demand, import and export energy cost, current imported and exported energy (within 5 seconds), and the current stored energy within our capacitors: are all displayed. Meeting the ”current information about energy flows and stores in the system” GitHub requirement. See Figure 18

2. The Historic information page: This page graphs the demand, import and export energy prices of the previous day. In addition graphing both the energy stored within our capacitors and the energy flows of our smart grid, during the present day , with values spaced by 5 second ticks. Meeting the ”historic information about energy flows and stores in the system” GitHub requirement. See Figure 19

3. The Energy Algorithm Performance page: This page displays the value of the energy that has been imported or exported within a 5 second period, and the total profit from our smart grid.See Figure 20

The profit is calculated by measuring the net energy that enters or leaves the grid within a 5 second period. This approach is the simplest and most accurate way, since the net energy measured by the grid directly describes the energy loss or gain of the smart grid system. If the measured energy is positive, this corresponds to the smart grid selling energy to the external grid, and so after multiplying the amount of energy exported by its export price, this value is added to the teams profit. If the measured grid value is negative, then this will correspond to our smart grid buying from the external grid and so after multiply the amount of energy imported by its import price, this vale is deducted from the teams profit. As the algorithm provides new information to the smart grid every 5 seconds, it is apt to require the grid to send the net energy measured in 5 second periods.

2.6.4 Implementation

Within the front end, useEffect hooks are used *”to “step out” of your React code and synchronize with some external system. This includes browser APIs ...”* [7] making it the perfect tool for our website to update with both the external web-server’s and our smart gird values.Values from both systems are fetched within 5 second intervals, and are either displayed in boxes, or graphed depending on the page the user is accessing.

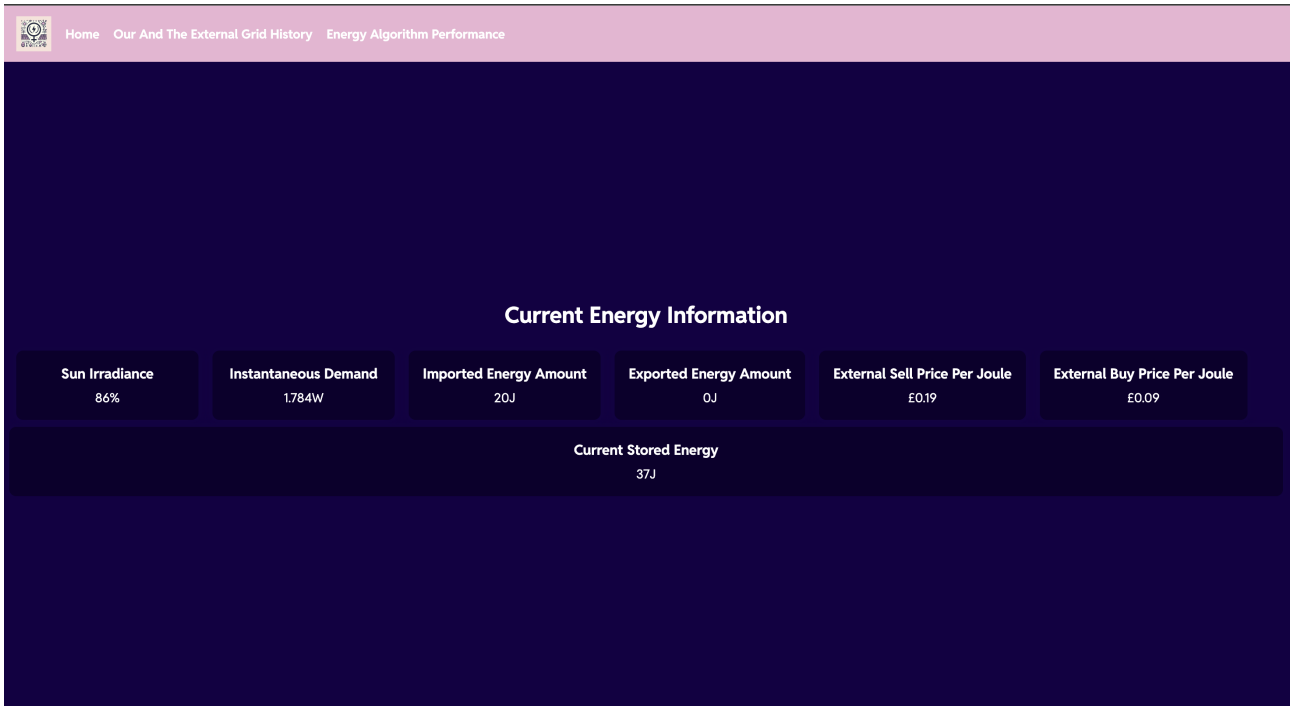


Figure 18: Shows the homepage displaying the external web-server and current grid values

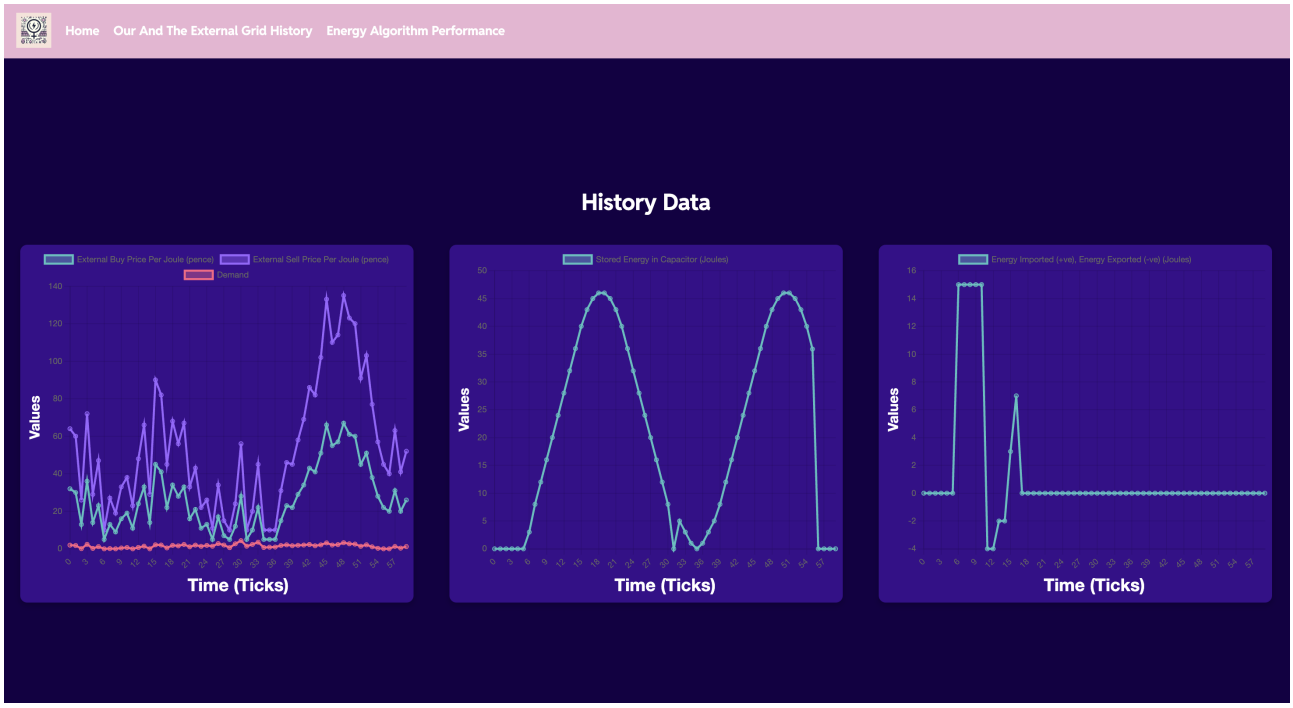


Figure 19: Shows the Historic Information page displaying the historic grid values and yesterdays data

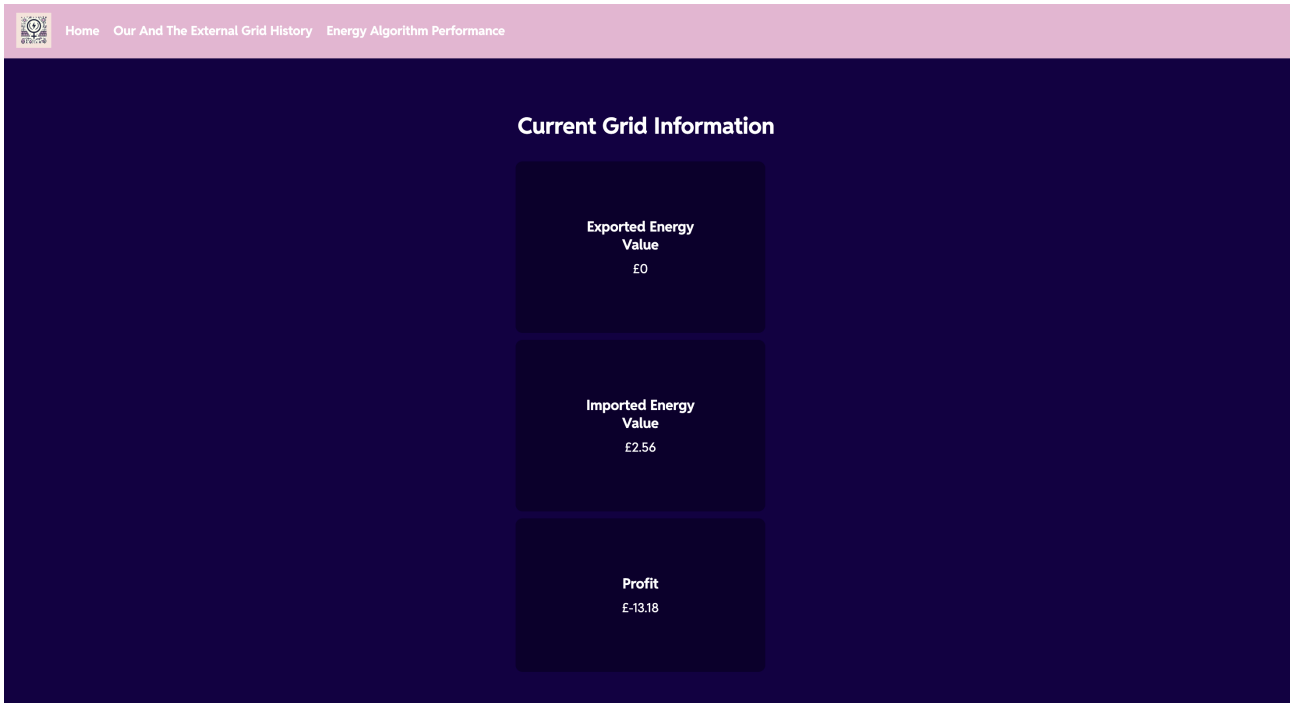


Figure 20: Shows the Energy Algorithm Performance page, displaying the current value of energy imported and exported and the current profit made

The Flask API has 8 HTTP GET and 2 HTTP POST endpoints: our socket server and front end, which send HTTP GET requests to the Flask API, to display and use data respectively. For example the endpoint *""* is used by the *"helper.py"* program, as the program contains both our machine learning and deferrable demand algorithms both requiring data from the external web-server. In order to link the socket server to the web-server and display our smart grid values, HTTP POST requests were used, HTTP requests have alot of overhead so they aren't the most efficient solution, however for this simple web site and due to fact our values only refresh every 5 seconds, they are appropriate for our use. The socket server will receive the values from the grid and capacitor, and send these values to their respective endpoints through a HTTP POST request, then the Flask server stores these values in a global variable each , sending these values to their respective endpoints, allowing our website can retrieve them through a HTTP GET request. A snippet of this and how a profit is calculated, has been provided below in figure 21.

In order to graph the historic values of the smart grid, the values that are sent from the smart grid, are placed inside an array of a fixed size, these arrays represents the values of the

smart grid (capacitor energy and grid import/export) within the present day. The index of the array is the current tick of the day, these arrays are then sent to two other API endpoints that are accessed by the Historic information page, using the **react-chartjs-2** library [8] these arrays can be plotted against the ticks in a full day, and the present days capacitor energy and grid import/export values can be seen on the "Our And The External Grid History" page as they update in real time.

```
@app.route('/send_grid_data', methods=['POST'])
def receive_grid_data():
    global grid_data
    try:
        received_data = request.json  # Data is sent in json format so got to handle
        grid_data = received_data
        response = requests.get('https://icelec50015.azurewebsites.net/price')
        tick = response.json()["tick"]
        grid_graph_data[tick]=int(grid_data)
        grid_graph_data[tick+1:]=len(grid_graph_data[tick+1:])*[0]


        return jsonify({'message': 'Data received successfully'}), 200
    except Exception as e:
        print(f"Error processing received data: {e}")
        return jsonify({'error': 'An error occurred while processing data'}), 500

@app.route('/forward_grid_data', methods=['GET'])
def forward_grid_data():
    global profit
    try:
        # Check if data is stored
        response = requests.get('https://icelec50015.azurewebsites.net/price')
        buyprice = response.json()["buy_price"]
        sellprice = response.json()["sell_price"]
        if(int(grid_data)>0):
            profit -= abs(int(grid_data)*sellprice)
        if(int(grid_data)<0):
            profit +=abs(int(grid_data)*buyprice)
        if grid_data:
            print("Sending",grid_data)
            return jsonify({
                'profit': profit,
                'value' : grid_data

            })
        else:
            return jsonify({'message': 'No data available'}), 404
    except Exception as e:
        print(f"Error forwarding data: {e}")
        return jsonify({'error': 'An error occurred while forwarding data'}), 500
```

Figure 21: Code showing how data from the grid is sent to an API endpoint to be accessed by our React web page

2.7 Data Server

To link the hardware to the software, data must be transmitted between the Pi's and the server, which handles all data processing. This data exchange must occur within a five-second window to ensure the information remains relevant. The data must be received, processed, and a response sent back within this time frame to ensure the system runs correctly. Furthermore, the data server needs to store the necessary data required for the website.

2.7.1 Research

To send information the team had two options: TCP (Transmission Control Protocol) or UDP (User Datagram Protocol): TCP allows for more reliable transfer between server and client however, UDP is faster. Furthermore, to increase the speed of data transmission, the server should employ parallelism, using either the multi-processing or threading Python libraries. [13] During the prototyping phase, the team explored the possibility of running the server remotely on AWS, rather than running the server locally on a private network. However, greater delays were found when AWS was in use; we attributed these delays to an increase in latency with a remote server and thus the team continued to use the local server.

2.7.2 Final Design

A threaded server listens on port 5001, until a client connects to it, at which point the request is split into its own thread. This design ensures that each client connection is handled in a separate thread, allowing concurrent processing of multiple requests. This decision was chosen due to the need for high responsiveness and reliability in real-time data processing, where each data exchange must be completed within the strict five-second window. The threading module was chosen due to previous experience using this module, thus allowing the creation of the server to be built quicker. In the context of our project, TCP was chosen over UDP due to several key advantages that align with the requirements for reliable and ordered data transmission. TCP ensures that all data packets are delivered in the correct order and without erroneous bits, which is vital for our project.

2.7.3 Implementation

The system starts with the creation of a TCP server, made using Python's socket library. The server is configured to listen on port 5001, awaiting connections from multiple Pi clients. Upon the establishment of a connection, a handshake protocol is initiated (as seen in the figure 22), the client identifies itself through sending its name, and a new thread is dynamically spawned for each client using Python's threading module. Within this thread the necessary data is traded between server client, and the the connection if closed. This multi-threaded approach allows the server to handle multiple client requests simultaneously without blocking, in order to ensure optimal performance and quick response times.

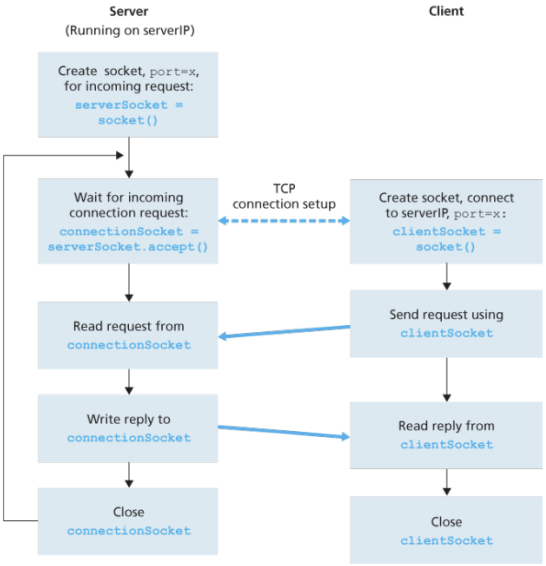


Figure 22: TCP Connection [12]

To validate the server's functionality, rigorous testing was conducted. A series of stress tests were executed where client requests bombarded the server with data queries. The response time for each request was measured using Python's time module, confirming that all responses were delivered within the five-second time-frame. Furthermore, debug print statements in the server's output showed the simultaneous execution of multiple threads, showing the correct operation of the threading mechanism.

```
Sent: PV
Received: PV
Data recieved ok
Sent: -1.886866e-05
40
closing
--- 3 seconds ---
-1.886866e-05,40,-0.1824,0.0002
Connecting to 192.168.203.234 5001
here
Sent: PV
Received: PV
Data recieved ok
Sent: -0.0
49
closing
--- 2 seconds ---
-0.0,49,-0.3648,0.0
Connecting to 192.168.203.234 5001
here
Sent: PV
Received: PV
Data recieved ok
Sent: 0.0
66
closing
--- 2 seconds ---
0.0,66,-0.44688,-0.0
Connecting to 192.168.203.234 5001
here
Sent: PV
Received: PV
Data recieved ok
Sent: -0.0
74
closing
--- 3 seconds ---
-0.0,74,-0.60192,0.0
Connecting to 192.168.203.234 5001
here
Sent: PV
Received: PV
Data recieved ok
Sent: 1.02547e-05
```

Figure 23: PV cell runtime during a whole system run

2.8 Algorithm

The algorithm consists of two aspects, the machine learning algorithm that is used to predict the trend of the next 2 prices, to be used to decide whether our grid should import or export energy, and the deferrable greedy algorithm, that works to complete the deferrable worth the most amount of power first(energy per tick).

2.8.1 Requirements

As mentioned in the Github, the algorithm *"decides when to store/release and import/export energy. It shall also choose when to satisfy demands that can be deferred"*.

2.8.2 Research

The machine learning algorithm required reading regarding, what type of machine learning models are used in financial markets to predict the movement of stocks and shares. The decision to use Random Forest Regression, was due to the fact that Random Forest Regression and Long short-term memory (LSTM) are both machine learning techniques used to predict stock prices. Building a LSTM model would require the learning of PyTorch and due to the project deadline, it was not feasible, whereas thanks to the python library **scikit learn** [9] Random Forest Regression is much easier to understand and implement in code. A issue encountered when training our Random Forest Regression model was over fitting, this is when the ml algorithm fits extremely well for (known) training data and performs poorly on unseen data (test data). The effect posed by over fitting can be mitigated by using a large dataset and also training the model on important features. The dataset our model is trained on has, 540 rows where each row has 4 columns. This dataset is good enough for the team, though the more data the better. In order to satisfy the important features requirement, our model was trained on the ticks, demands and import prices, and the previous 3 export prices. By inputting the previous export prices as training data, our model can find the relationship between previous export prices, with the eventual goal to predict the next export prices [15]. The researched approach in the teams algorithm, in order to meet the GitHub requirement regarding satisfying demand, was to follow a greedy algorithm in order to meet all the deferrable demands.

2.8.3 Final Design

The final implementation of our algorithm contains 2 functions within a *"helper.py"* file which aids the Data Server, in connecting the web-server and Smart Grid to each other:

1. The energyAlgorithm function within the helper, fetches all the current values that are needed for the model, producing an array of next 2 export prices. If the model predicts an upwards trend then the smart grid is told to buy energy, and should the model predicts a downwards trend,the smart grid is told to sell energy. The amount of energy that is exported or imported is described in more detail within the Implementation section below.See figure 24

2. The greedydeferrable function, within the helper, fetches the deferrables from either the local deferrables json database, if this file does not yet exist or if the current tick is equal to 0 (a new day) the deferrables are fetched from the Web-Server API. At the end of the function, the new values of the deferrables are stored and updated in a *"deferrable_db.json"* allowing the team to keep track of which deferrables have been satisfied and by how much, and returning a demand that the smart grid must satisfy, how this demand is calculated is mentioned in the Implementations section below. See figure 25.

```
def energyAlgorithm(MaxAmount):
    input,_,_,_,_,_=fetch_latest()
    predicted_prices=model.predict(input)
    test_list=predicted_prices[0]
    decr=0
    incr=0
    ratio=return_demand()/5
    res=0
    decr = np.array_equal(test_list, sorted(test_list, reverse=True))
    incr = all(i < j for i, j in zip(predicted_prices[0], predicted_prices[0][1:]) #if pred array increases the nprice trending up, buy

    if(decr):
        res=(46-MaxAmount)*ratio*-1
    if incr:
        res=MaxAmount*(1-ratio)
    return res
```

Figure 24: The algorithm the Smart Grid uses to predict the trend in prices and buy or sell accordingly

```python
def greedyDeferable():
    demand,tick,deferablelist=loaddeferable()
    freepower=4-demand
    ratiolist=[0]*3
    data=deferablelist
    # Check if deferablelist is empty and return early
    if not deferablelist:
        print("All deferables are processed list is empty")
        return demand # no more, just do demand
    for key,deferable in data.items():
        if deferable[2] <= tick:
           ratiolist[int(key)]=(deferable[1] / (deferable[0]-deferable[2]))
    max=0
    postion=0
    #if deferable now at 0, remove
    for i in range (0, len(ratiolist)):
        if ratiolist[i]>= max:
            postion=i
            max=ratiolist[i]

    if deferablelist[str(postion)][1]-5*freepower >=0: #We maximise how much of the deferable we do - take this away from the amount we are storing
        deferablelist[str(postion)][1]=deferablelist[str(postion)][1]-5*freepower
        print("Deferable at",deferablelist[str(postion)], "has ", deferablelist[str(postion)][1] )
        cleanandsave(data)
        return 4 #tell load to max out since we are maximising with greedy algo
    else:#if there is less deferable energy than the free room, then dont use all free room, just do deferable amount + demand
        deferablelist[str(postion)][1]=deferablelist[str(postion)][1]-5*freepower
        print("Deferable at",deferablelist[str(postion)], "has ", deferablelist[str(postion)][1] )
        cleanandsave(data)
        return (deferablelist[postion][1]/5)+demand #deferable value is in Joules so we need to convert it back to power by dividing by 5
```

Figure 25: Greedy algorithm used to satisfy deferrable demands

2.8.4 Implementation

In order to predict the trend in the export price,new information needs to be provided to the model. To do this, within the *"helper.py"* file , the *"fetch_latest"* function is called, and a new pandas data-frame [10] containing the current demand,tick and lags (previous 3 export values) is created, which is then inputted into the model,creating a prediction of the next 2 export prices. If the predicted prices are increasing, then our smart grid will buy energy now, as it can be sold for more money later on. The amount of energy that the smart grid buys is the amount of energy that our capacitors can store multiplied by the (current demand)/5. This constant of proportionality was chosen to satisfy the GitHub requirement that our algorithm *acts to minimise the amount of power imported or exported at a given moment*, as when demand is low and the model is selling, the smart grid sells less energy, and if demand is low and the teams model is buying, the smart grid buys less energy.

For our team to satisfy the deferrable demands,it became clear the need of a database, that stores all the deferrable demands and the current energy each have remaining, if any of the deferred demands are satisfied, then they need to be removed from the database, leaving only the remaining deferrables.Initially this database file does not exist and so the greedyDemand function, uses the deferrable demand values that are fetched from the */helper* API endpoint, the web-server provides, an example of this helper data can be seen in figure 26.

Subsequently, if the received data isn't empty and the deferrable is valid (the ability to satisfy the deferrable has begun) the power value of each deferrable that satisfies the criteria is calculated and stored in an array. The highest value of this array corresponds to the deferrable currently worth the most. The teams knapsack greedy algorithm then calculates the demand to be sent to the Load; this value is calculated by considering the "importance" of each deferrable (energy needed/time to die) and using the "free space" left within the load. Such "free space" exists due to the maximum load capping at four Watts, however the current instantaneous demand is usually lower than this capped amount, allowing the system to use this spare load to complete the deferrable demand.

$$E^{max}_{defferable} = 5 \times (4 - InstantaneousDemand) \tag{13}$$

If the energy of the most valuable deferrable is greater than this value, the Load is sent its maximum value of 4, and the value of max E_deferrable is deducted from the deferrable, and the new values of the deferrables are stored in the database.

If the energy of the most valuable deferrable is less than this maximum value, then this implies that the deferrable is going to be satisfied and so the Load receives the instantaneous demand added onto the power the remaining part of the deferrable demand requires (in a 5 second period).

Any deferrable demands that are met, are then removed from the JSON database so that the team's algorithm can perform as expected.

```
{
  "current_sell_price": 118,
  "deferables": {
    "0": [
      59,
      50.0,
      0
    ],
    "1": [
      59,
      39.81783866255207,
      46
    ],
    "2": [
      36,
      42.860031693791434,
      26
    ]
  },
  "demand": 2.4398105899703304,
  "lags": [
    115,
    118,
    118
  ],
  "sun": 0,
  "tick": 50
}
```


Figure 26: Helper endpoint values

3 Final Evaluation

(Top Level Overall System)

3.1 Technical

To allow data to pass between each system component, the team implemented a client-server communication using TCP to ensure ordered and reliable data exchange between components and server. The server runs on a specified IP and port, listens to incoming connections, and processes the data sent from clients. After establishing a connection, the clients send the server their name and then all appropriate and required data is shared. Our implementation uses TCPs as they have the benefits of error detection and flow control, making it suitable for applications requiring dependable and sequential data transmission. .Any advanced techniques applied to the overall system

3.2 Practical

When the whole system was turned on, each component successfully exchanged data with the server, adjusting its values within the designated time-frames. Our system effectively fulfilled both immediate and deferred demands (evidence shown in web server section). Considering our algorithms final output and monetary result, the team have been able to show our algorithms superiority over the naive approach.

3.3 Application

The smart grid project could be applied to any building, commercial or residential, to reduce the cost of energy being imported, by generating energy through the PV array. This project provided proof of concept, however, should this be applied to the real world, the super-capacitors should be switched to flywheels for longer-term storage, and a larger PV array would be needed, as the load demand is also likely to be greater.

4 Conclusion

The system efficiently manages and supplies energy to LED loads based on the instantaneous demand and the current deferrable tasks by simulating a PV cell and the varying irradiance throughout the day and night. Our system tries to maximise the profit by selling excess energy back to the grid using a machine-learning-based algorithm that predicts prices will fluctuate; allowing for more energy to be bought and stored in the capacitors when the price is low. Finally, all necessary data, such as current values and profit will be shown in real-time on a flask based website.

5 References

[1] E. Stott, P. Clemow, and G. Lu, "EEE2/EIE2 Electronics Design Project," GitHub. https://github.com/edstott/EE2Project/blob/main/doc/PV-real.svg

[2] Elektro-Automatik, "Application Note: Simulating Solar Cells." Accessed: Jun. 06, 2024. [Online]. https://www.eapowered.com/wp-content/uploads/2021/08/ea_application_note_simulating_solar_cells_ru.pdf

[3] I. Yahyaoui, Advances in Renewable Energies and Power Technologies Solar, Wind, Wave Energies and Fuel Cells. Elsevier Science Ltd, 2018.

[4] E. Stott, P. Clemow, and G. Lu, "EEE2/EIE2 Electronics Design Project," GitHub. https://github.com/edstott/EE2Project/blob/main/smart-grid/SMPS_2024_Bidirectional.py

[5] Ben K, Sahbani A and Benrejeb M (2011) Sliding Mode Control and Fuzzy Sliding Mode Control for DC-DC Converters. Sliding Mode Control. InTech,Andrzej Bartoszewicz (ed.). Available at: https://www.intechopen.com/chapters/15204

[6] Cors Error definition https://www.contentstack.com/docs/developers/how-to-guides/understanding-and-resolving-cors-error

[7] The useEffect documentation https://react.dev/reference/react/useEffect#usage

[8] The library used to graph data on UI https://react-chartjs-2.js.org

[9] The machine learning library used for the price trend prediction algorithm https://scikit-learn.org/stable/

[10] Pandas dataframe documentation https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.html

[11] Simoes, G.M. and Farret, F.A. (2017) 'Designing Power Electronic Control Systems', in Modeling Power Electronics and Interfacing Energy Conversion System. New Jersey: John Wiley Sons, pp. 83–116.

[12] James F. Kurose and Keith W. Ross. 2020. Computer Networking: A Top-Down Approach (8th Edition) Pearson. https://medium.com/@arjunprakash027/threading-vs-multiprocessing-in-python-a-comprehensive-guide-cae3ce0ca6c1

[13] Prakash, A., 2020. Threading vs Multiprocessing in Python: A Comprehensive Guide. https://medium.com/@arjunprakash027/threading-vs-multiprocessing-in-python-a-comprehensive-guide-cae3ce0ca6c1

[14] PID Explained, "How to Tune a PID Controller • PID Explained," PID Explained, Nov. 24, 2018. Available: https://pidexplained.com/how-to-tune-a-pid-controller/. [Accessed: Jun. 17, 2024]

[15] https://www.ibm.com/topics/overfitting

6 Appendix

Appendix A

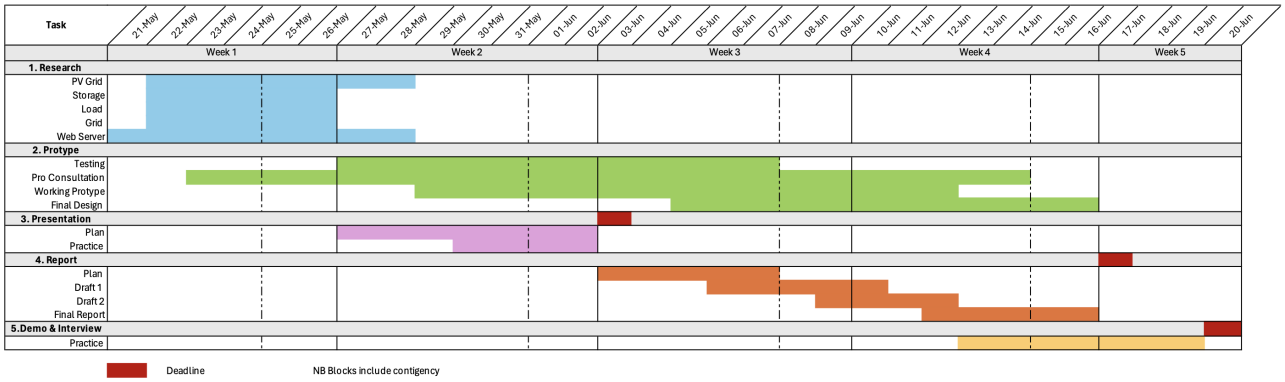


Figure 27: Gantt Chart

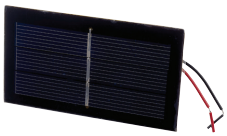
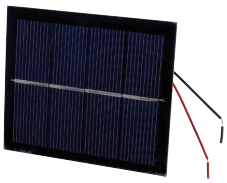
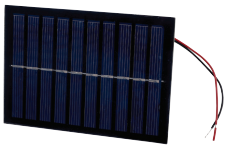
Appendix B

Solar Cell Modules

Features:

- Using crystal type silicon solar cells
- Small size
- Light Weight
- High efficiency
- Output with 2 solder Line, 24# AWG Red and black wires; length: 30 cm
- Long life and durable
- **IP65** ability

Specifications:

Order code	MPN	Solar Cell	Voltage (V)	Current (mA)	Power (W)	Dimensions (mm)	Power Tolerance (%)	Temperature Coefficient (%/°C)
55-0010	OPL15A25101		1.5	250	0.37	90x50x3	+/–3	–0.45
55-0012	OPL20A25101		2.0	250	0.5	90x60x3	+/–3	–0.45
55-0014	OPL20A50101		2.0	500	1	90x105x3	+/–3	–0.45
55-0016	OPL50A23101		5.0	230	1.15	90x125x3	+/–3	–0.45

Appendix C

Design FMEA																
Project	Smart Grid								**Version**							
Sub assembly	The External Grid								**Date**	17/06/2024						
Prepared by	The Slay Girlies								**Page____ of _____**							
Approved by									**Notes**							
Process Purpose	**Potential Failure Mode**	**Potential effect(s) of Failure**	**Severity**	**Potential Causes of Failure**	**Occurrence**	**Process Control**	**Detection**	**RPN**	**Recommended Actions**	**Area / person responsible**	**Delivery date**	**Action taken**	**Severity**	**Occurrence**	**Detection**	**New RPN**
Control voltage of bus fast and responsive with minimal oscillations (The Grid)	unbounded error used in slide surface	oscillations of max and min duty cycle to change voltage	9	starting with a very high pwm or very low pwm that can cause a high error in voltage	7	Printing the duty cycle every 0.1 seconds	1	63	saturate slide function to half pwm	External Grid	17/06/2024	Ensuring stability of slide function	1	1	4	4
	two SMPS trying to control different voltages in same bus	Non constant bus voltage and errors in Raspberry Pi Pico	10	two voltage regulator controls acting on same bus voltage	10	Printing the voltages regularly	1	100	ensure only one SMPS is controlling bus voltage, have the others control current or have a variable voltage at least	External Grid	17/06/2024	Checking PI controller code on each SMPS and changing it accordingly	1	1	1	1
	Variable bus voltage	not enough current to control to desired bus voltage using SMPS	10	Not enough current drawn from power supply	10	Printing the voltages regularly	1	100	Set the power supply to a high enough current depending on deisred bus voltage	External Grid	17/06/2024	Choose a sensible bus voltage to balance between demand and power supply current drawn	10	1	7	70
	Unable to communicate with server	Not being able to send energy import and export	6	time delay and multiple circuit boards trying to communicate at same time	6	Monitoring web and information printed on program	4	144	Send information at different times for each board, at time difference larger than delay to close and open communication with the server	External Grid	17/06/2024	Ensure the circuit boards communicate at different times using sockets	6	5	4	120
	Memory allocation errors	Crash of microcontroller code	10	Using arrays and not cleaning up on time for next allocation	9	Monitoring terminal for unexpected crash	1	90	Do not use arrays to send all energy import export at all sample times, instead do a running total in one variable	External Grid	17/06/2024	Use arrays with constant size and ensure they get deleted after set amount of loops	10	1	1	10
	error in get globals	Crash of microcontroller code	10	Using too many global variables	6	Waiting for crash in terminal	9	540	Attempt not to use global variables	External Grid	17/06/2024	Only using global variables given in skeleton code	10	3	9	270
	Overshoot in bus voltage	High power losses in bus	6	Controller not being tuned properly	10	Monitor bus voltage regularly	1	60	Start at an initial duty cycle that should be close to desired bus voltage	External Grid	17/06/2024	Further tuning the PI controller	6	6	1	36
	High power loss in resistor sink	Not enough power left to use SMPS to control voltage using Power supply	9	Resistor is too small and sinks too much current	7	Wait for malfunction in bus voltage	8	504	Use a larger resistor in parallel with power supply	External Grid	17/06/2024	A larger resistor used	9	1	8	72
	Microcontroller overheating	Program terminating early	10	High currents in sensor	1	Wait for program crash	1	10	Ensure currents are small enough	External Grid	17/06/2024	Current limits in code used	1	1	1	1

Figure 28: FMEA

Appendix D

Type DSM, Standard Supercapacitor Modules

9V, 18V, and 30V configurations with cable assembly

Type DSM Standard Supercapacitor Modules provide a wide range of capacitance values and voltages to enable simple, rapid implementation into any system. Each module features an insulated construction with integrated cell balancing, cable assembly with a Molex Mini-Lock connector, and 85 ºC rated supercapacitor cells.

Highlights

- Standard, stocked modules, great for prototyping.
- Wide operating temperatures compared to Li-batteries
- Voltages up to 30 Vdc, capacitance from 0.15 to 36.6 Farads
- Safe, green alternative to Li-ion batteries
- Each module has voltage balancing
- REACH and RoHS compliant
- UL 810A recognized cells
- Custom designs available

Specifications

Rated Voltage Range	9 Vdc to 30 Vdc
Capacitance Range	0.15 F to 36.6 F
Capacitance Tolerance	-10% +30% (20 ºC)
Operating Temperature Range	–40 ºC to +65 ºC 9V, 18V, 30V
Extended Temperature Range	–40 ºC to +85 ºC 7.5V, 15V, 25V
Storage Temperature	–40 ºC to +70 ºC
Life Time	1000 Hours at rated voltage 65 ºC
Shelf Life	1000 Hours at 65 ºC
Life Cycles	500,000 Cycles at 25 ºC, Vr to 1/2 Vr
Regulatory Information	

Part Numbering System

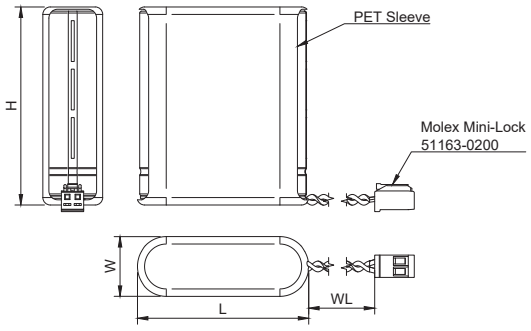
DSM Series	504 Capacitance	Q Tolerance	009 Voltage	W Teminal Type	075 Lead Length	PB Balancing Type
DSM	504 = 0.5 F 105 = 1.0 F 116 = 11 F	Q = -10% to +30%	009 = 9V 018 = 18V 030 = 30V	W = wire with connector	075 = 75mm	PB = passive balancing AB = active balancing

Appendix D

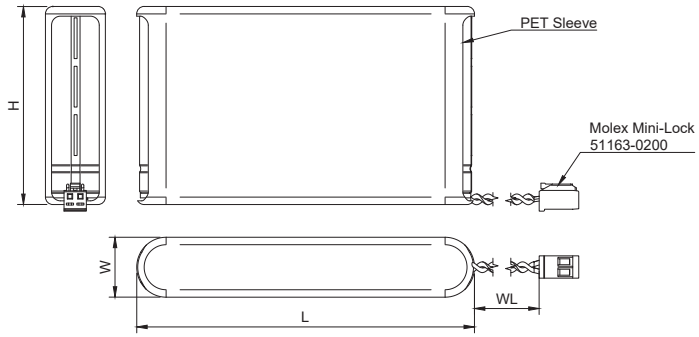
Type DSM, Standard Supercapacitor Modules

9V, 18V, and 30V configurations with cable assembly

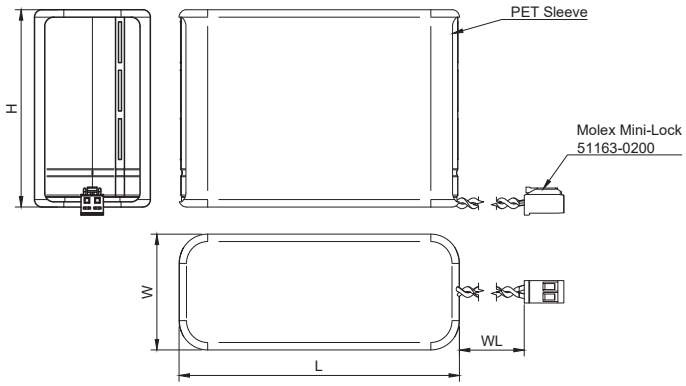
9V Standard Modules



18V Standard Modules



30V Standard Modules



Appendix D

Type DSM, Standard Supercapacitor Modules

9V, 18V, and 30V configurations with cable assembly

Part Numbers

Part Number	Nominal Capacitance	ESR DC	Peak Current 1s	Rated Current 5s	Leakage Current 72 hrs	Mass	L (max)	H (max)	W (max)	WL (typ)
	(F)	(mΩ)	(A)	(A)	(mA)	(g)	(mm)	(mm)	(mm)	(mm)
9 VDC (65 °C) | 7.5 VDC (85 °C)										
DSM504Q009W075PB	0.5	1940	1.14	0.38	0.20	3.5	21.5	24.5	7.5	75
DSM105Q009W075PB	1.0	455	3.09	0.83	0.30	5.5	27	27	9	75
DSM335Q009W075PB	3.3	546	5.30	2.20	0.90	11	33	30	11	75
DSM505Q009W075PB	5.0	210	12.26	3.86	1.10	13	41	30	14	75
DSM126Q009W075PB	11.6	152	18.89	7.76	3.00	32	51	39	17.5	75
DSM236Q009W075PB	23.3	122	27.29	13.41	9.00	48	57	57	19.5	75
DSM376Q009W075PB	36.6	110	32.84	18.35	11.00	63	57	67	19.5	75
18 VDC (65 °C) | 15 VDC (85 °C)										
DSM254Q018W075PB	0.25	3879	1.14	0.38	0.20	6	42	24.5	7.5	75
DSM504Q018W075PB	0.5	909	3.09	0.83	0.30	11	52	27	9	75
DSM165Q018W075PB	1.6	1091	5.24	2.20	0.90	20	64	30	11	75
DSM255Q018W075PB	2.5	334	12.26	3.86	1.10	26	79	30	14	75
DSM585Q018W075PB	5.8	303	18.93	7.76	3.00	64	100	39	17.5	75
DSM126Q018W075PB	11.6	243	27.34	13.41	9.00	96	112	57	19.5	75
DSM186Q018W075PB	18.3	220	32.66	18.08	11.00	126	112	67	19.5	75
30 VDC (65 °C) | 25 VDC (85 °C)										
DSM154Q030W075PB	0.15	6464	1.14	0.38	0.20	10	35	24.5	14.5	75
DSM304Q030W075PB	0.3	1515	3.09	0.83	0.30	15	44	27	18	75
DSM105Q030W075PB	1.0	1818	5.32	2.20	0.90	29	54	30	22	75
DSM155Q030W075PB	1.5	700	12.27	3.86	1.10	48	66	30	27	75
DSM355Q030W075PB	3.5	505	18.97	7.76	3.00	89	84	39	34	75
DSM705Q030W075PB	7.0	404	27.43	13.41	9.00	171	94	57	38	75
DSM116Q030W075PB	11.0	360	33.26	18.41	11.00	225	94	67	38	75

Appendix D

Notice and Disclaimer: All product drawings, descriptions, specifications, statements, information and data (collectively, the “Information”) in this datasheet or other publication are subject to change. The customer is responsible for checking, confirming and verifying the extent to which the Information contained in this datasheet or other publication is applicable to an order at the time the order is placed. All Information given herein is believed to be accurate and reliable, but it is presented without any guarantee, warranty, representation or responsibility of any kind, expressed or implied. Statements of suitability for certain applications are based on the knowledge that the Cornell Dubilier company providing such statements (“Cornell Dubilier”) has of operating conditions that such Cornell Dubilier company regards as typical for such applications, but are not intended to constitute any guarantee, warranty or representation regarding any such matter – and Cornell Dubilier specifically and expressly disclaims any guarantee, warranty or representation concerning the suitability for a specific customer application, use, storage, transportation, or operating environment. The Information is intended for use only by customers who have the requisite experience and capability to determine the correct products for their application. Any technical advice inferred from this Information or otherwise provided by Cornell Dubilier with reference to the use of any Cornell Dubilier products is given gratis (unless otherwise specified by Cornell Dubilier), and Cornell Dubilier assumes no obligation or liability for the advice given or results obtained. Although Cornell Dubilier strives to apply the most stringent quality and safety standards regarding the design and manufacturing of its products, in light of the current state of the art, isolated component failures may still occur. Accordingly, customer applications which require a high degree of reliability or safety should employ suitable designs or other safeguards (such as installation of protective circuitry or redundancies or other appropriate protective measures) in order to ensure that the failure of an electrical component does not result in a risk of personal injury or property damage. Although all product-related warnings, cautions and notes must be observed, the customer should not assume that all safety measures are indicated in such warnings, cautions and notes, or that other safety measures may not be required.