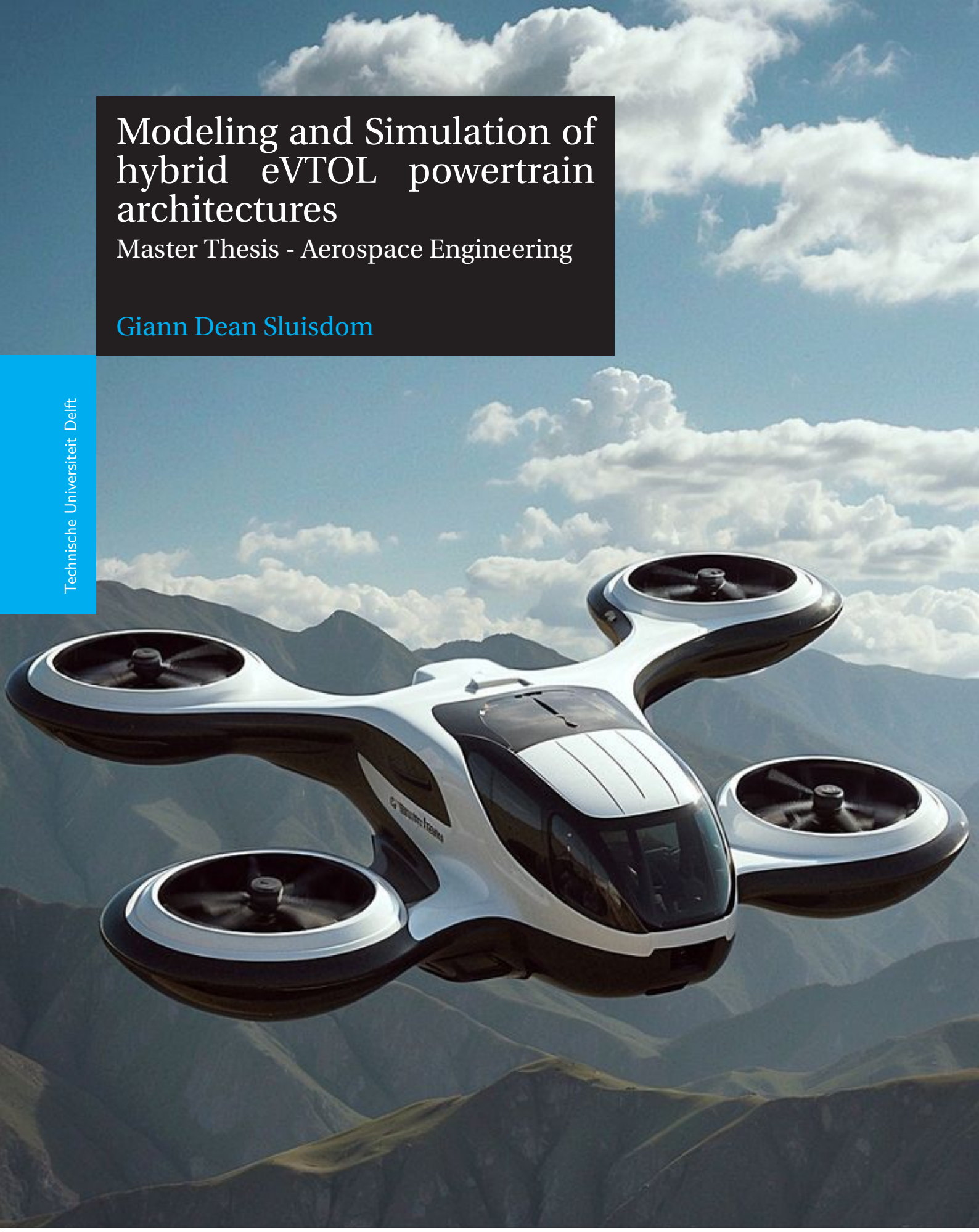
Modeling and Simulation of hybrid eVTOL powertrain architectures

Master Thesis - Aerospace Engineering

Giann Dean Sluisdom

Technische Universiteit Delft



TU Delft Delft University of Technology

Challenge the future

MODELING AND SIMULATION OF HYBRID eVTOL POWERTRAIN ARCHITECTURES

MASTER THESIS - AEROSPACE ENGINEERING

by

Giann Dean Sluisdom

in partial fulfillment of the requirements for the degree of

Master of Science
in Aerospace Engineering

at the Delft University of Technology,
to be defended publicly on Tuesday June 24, 2025 at 10:00 AM.

Supervisor:	dr. ir. C. Falsetti,	TU Delft
	dr. ir. W. P. J. Visser,	TU Delft
	Dipl.-Ing. A. Lautenschläger,	AYED-ENGINEERING GmhB
Thesis committee:	Prof. dr. ir. P. Colonna,	TU Delft
	dr. ir. C. Falsetti,	TU Delft
	ir. P. Roling,	TU Delft
	dr. ir. W. P. J. Visser,	TU Delft
	Dipl.-Ing. A. Lautenschläger,	AYED-ENGINEERING GmbH

An electronic version of this thesis is available at http://repository.tudelft.nl/.

Cover Image: eVTOL RALF WEBER design, www.ralfweber.design/

TU Delft Delft University of Technology

ABSTRACT

The development of Urban Air Mobility hinges on efficient, low-emission propulsion for electric Vertical Take-Off and Landing aircraft. This thesis presents a modular performance calculation tool for comparing three powertrain architectures; battery-only, fuel cell–battery, and fuel cell–turbogenerator, under a representative mission profile including take-off, climb, cruise, descent, landing, and emergency segments.

Component models employ a second-order Thevenin equivalent circuit for the lithium-ion battery, the Amphlett static approach for the polymer-electrolyte-membrane fuel cell, and precomputed zero-dimensional maps from the Gas Turbine Simulation Program for the turbogenerator. Steady-state 0-D system assumptions and constant-efficiency parasitic loads (compressor, humidifier, thermal management system) balance fidelity with computational speed. A non-causal power-management algorithm allocates base load to the primary source and peak demands to secondary storage.

Using Differential Evolution, the tool optimizes battery voltage, fuel-cell voltage, and fuel-cell power limit to minimize operational empty weight subject to a maximum take off weight of 3175 kg. The all-battery design incurs prohibitive mass. The fuel cell-battery configuration reduces battery mass by sizing the fuel cell for steady segments and the battery for transients, yet still exceeds maximum take off weight limit. The fuel cell-turbogenerator hybrid yields the lowest operational empty weight, demonstrating the potential of integrating a turbogenerator for high-power flight segments.

Sensitivity studies reveal that doubling the current cell capacity of the Panasonic NCR18650 could render the fuel cell/-battery architecture feasible, while switching from compressed to liquid hydrogen storage can yield significant mass savings. The sensitivity studies further clarify trade-offs between component voltages, power split strategies and aircraft weights

Although current battery and fuel-cell technologies fall short of strict weight limits, targeted advances in energy density, discharge rates, and hydrogen storage, along with refined thermal and power-electronics modeling, offer a clear pathway toward practical hybrid and fully electric eVTOL systems. Future work should integrate higher-fidelity thermal dynamics, aging effects, multi-objective optimization, and coupled aerodynamic–structural–propulsion analyses to accelerate sustainable urban flight.

PREFACE

Looking back on my years as a master student at Delft University of Technology, I realise how much this period has shaped me, both as an aerospace engineer and as a person. My path was not free of obstacles. Yet every setback turned into a lesson, and every lesson added a small measure of resilience. Today, holding this thesis in my hands, I feel proud—not because the journey was flawless, but because I learned to rise stronger each time I stumbled.

This growth would have been impossible without the people who guided, challenged, and encouraged me. First and foremost, I would like to thank my TU Delft supervisors, Chiara Falsetti and Wilfried Visser. From AYED–ENGINEERING, Alexander Lautenschläger and Johannes Kleinert. Your critical questions and relentless standards pushed me to sharpen my thinking and expand my horizons. You showed me that research is not about chasing perfect answers but about asking better questions.

My time at TU Delft would have been far less colourful without my friends Bram and Can. Whether we were debugging MATLAB scripts at midnight or grabbing a celebratory chocolate after a presentation, you reminded me that teamwork and laughter go hand in hand.

To my girlfriend Romy, thank you for the unwavering support that carried me through late-night writing sessions and early-morning frustration. Your patience, love, and gentle reminders to take breaks kept me balanced when the workload felt overwhelming.

My sisters, Bobbee and Amira, you are my inspiration. Wanting to set a good example for you both pushed me to finish what I started. You both will get the opportunity to study in the future and I will be right there supporting you in your choices and your dreams.

My father, Henk. You showed me what it means to work hard and stay curious. From helping me with homework as a child to proofreading draft chapters, your steady support has been my foundation.

Finally, to my mother Ruth. Thank you for believing in my dreams. You carried me through the hard times, raised me with love and discipline, and, together with Dad, gave me the opportunity to study at Delft University of Technology.

Giann Dean Sluisdom
Rotterdam, June 2025

Contents

List of Figures

LIST OF TABLES

NOMENCLATURE

Acronyms

BAT	Battery-only powertrain
BMS	Battery-management system
BOP	Balance-of-plant
CC	Constant-current
CV	Constant-voltage
ECM	Equivalent-circuit model
eVTOL	electrical vertical take-off and landing
GSP	Gas Turbine Simulation Program
HPS	Hybrid propulsion system
MAPE	Mean absolute percentage error
MCC	Multi-stage constant-current charging
MTOW	Maximum take-off weight
NCR18650	Panasonic NCR18650 Li-ion cell
OCV	Open-circuit voltage
OEW	Operational empty weight
PEMFC	Polymer-electrolyte-membrane fuel cell
PEM	Polymer-electrolyte membrane
RMSE	Root-mean-square error
SOC	State-of-charge
SOH	State-of-health
STOL	Short take-off and landing
UAM	Urban Air Mobility
VTOL	Vertical take-off and landing

Greek symbols

Δt	Integration time step (s)
η	Efficiency (–)
κ	Ratio of specific heats c_p/c_v (–)
λ	Stoichiometric or humidity ratio (–)
ρ_m	Specific resistivity of PEM membrane (Ω cm)

φ Relative humidity (–)

Roman symbols

$\dot{m}$ Mass-flow rate ($\mathrm{kg\,s^{-1}}$)

$\dot{m}_{\mathrm{air,in}}$ Inlet-air mass-flow rate to compressor ($\mathrm{kg\,s^{-1}}$)

$\dot{Q}$ Heat to be dissipated by thermal management (W)

$\dot{V}$ Volumetric flow rate ($\mathrm{m^3\,s^{-1}}$)

A Active area of a single PEM fuel cell ($\mathrm{cm^2}$)

C_1 Capacitance of first RC branch in battery ECM (F)

C_2 Capacitance of second RC branch in battery ECM (F)

c_p Specific heat of air at constant pressure ($\mathrm{J\,kg^{-1}\,K^{-1}}$)

F Faraday constant ($96\,485\ \mathrm{C\,mol^{-1}}$)

I Electric current (A)

i_{R_1} Current through first RC branch (A)

i_{R_2} Current through second RC branch (A)

m Mass (kg)

N Total number of cells in the fuel-cell stack (–)

n_{series} Number of electro-chemical cells in series (–)

P Power (W)

P_{comp} Compressor shaft power (W)

P_{total} Net stack electrical power (W)

Q Battery nominal capacity (A h)

R_0 Ohmic resistance in battery ECM (Ω)

R_1 Resistance of first RC branch (Ω)

R_2 Resistance of second RC branch (Ω)

$R_{\mathrm{electronic}}$ Electronic resistance in fuel-cell plates (Ω)

R_{proton} Protonic resistance of membrane (Ω)

T Temperature (K)

T_{t0} Total temperature at compressor inlet (K)

T_{t1} Total temperature at compressor outlet (K)

V Voltage (V)

V_{cell} Single-cell voltage (V)

z Battery state-of-charge (SOC) (–)

Subscripts

$_{\mathrm{air}}$ Air

$_{cell}$	Single electro-chemical cell
$_{FC}$	Fuel-cell quantities
$_{in}$	Inlet condition
$_{out}$	Outlet condition
$_{pack}$	Battery pack
$_{t0}$	Station 0 (compressor inlet)
$_{t1}$	Station 1 (compressor outlet)
$_{w}$	Water

Superscripts

$\left(^{cell}\right)$	Per individual cell
$\left(^{is}\right)$	Isentropic value
$\left(^{nom}\right)$	Nominal/reference value
$(^{max})$	Maximum value

1

INTRODUCTION

Electric Vertical Take-Off and Landing (eVTOL) and Electric Short Take-Off and Landing (eSTOL) systems have become pivotal elements in the Urban Air Mobility (UAM) framework, which has gained notable interest in recent years [13]. Urban air mobility proposes the implementation of airborne vehicles in urban settings for transporting people and goods [14]. This innovative method of transportation aims to tackle significant issues connected to typical road travel, including greenhouse gas emissions and traffic congestion in cities, with the former being a challenging concern.

A study from 2022 suggest that hybrid and fully electric propulsion system for eVTOL aircraft are expected to have similar performance in terms of environmental impact as their electrical road vehicle counterpart, given that the electricity generation continues in the same trend as in moving towards lower carbon intensity [15]. This projection underscores the importance of the development of the hybrid or fully electric powertrain configurations to further the concept of Urban Air Mobility.

Other significant advantages of hybrid-electric propulsion systems for eVTOL aircraft include energy cost savings and reduction of fuel usage. It was shown in a study that hybrid electric powertrain architectures can achieve a decrease of up to a 27% in total energy costs and a 28 % reduction in fuel consumption compared to conventional gas turbine engine propulsion [16]. This efficiency is achieved by utilizing battery power during high-demand flight segments such as takeoff and climb, while relying on fuel-based power sources for sustained cruise efficiency. This hybrid approach allows for optimized energy distribution, reducing overall fuel dependency and operational costs.

Despite the number of studies and research on hybrid and/or electric propulsion systems for eVTOL aircraft, a flexible framework and standardized tool for comparing different powertrain architectures remains underdeveloped. Most studies focus on specific configurations rather than providing a tool capable of assessing multiple hybrid-electric powertrain variations under different operating conditions. Addressing this gap is necessary for identifying the most efficient and feasible hybrid-electric propulsion solutions for eVTOL aircraft.

Modeling and simulations are essential in the design phase of hybrid propulsion systems for advanced aircraft. Computational techniques and simulation tools are used to evaluate performance, optimize designs, and analyze various configuration options. A framework for assessing and evaluating hybrid electric propulsion systems for eVTOL/eSTOL aircraft was developed through a collaboration between Delft University of Technology and AYED-ENGINEERING [9]. This project led to the creation of a performance calculation tool capable of predicting the performance of a fixed hybrid electric powertrain architecture, composed of a turbogenerator and a hydrogen fuel cell, specifically designed for eVTOL aircraft.

While this tool shows promising results in predicting propulsion system performance, it remains limited in its ability to compare multiple hybrid powertrain configurations, thereby hindering the determination of an optimal final design. The objective of this research is to develop a performance calculation tool that allows for the comparative assessment of multiple hybrid-electric propulsion architectures for eVTOL aircraft. By incorporating multiple power sources, this tool will provide insights into the trade-offs between different configurations, ultimately aiding in the selection of an optimal propulsion system. This tool is intended for steady-state calculations and may also be used to predict the propulsion system's performance in off-design conditions. The findings of this study will contribute to the advancement of hybrid-electric propulsion technologies and support the broader adoption of sustainable urban air mobility solutions.

The introductory chapter of this thesis is followed by chapter 2 which provides background information on eVTOL fundamentals, hybrid propulsion architectures, and the key power sources, including batteries, fuel cells, and turbogenerators. Additionally, it reviews power distribution strategies and existing modeling approaches in literature, setting the foundation for the methodology. Chapter 3 details the methodology used in this study, outlining the mission profile, system modeling approach, and component models for the fuel cell, battery, and turbogenerator. This chapter also describes the assumptions, design variables, and software implementation of the performance calculation tool. Chapter 4 presents the results of case studies evaluating two different hybrid powertrain configurations: a fuel cell-battery system and a fuel cell-turbogenerator system. The findings focus on component sizing, power distribution, and overall system performance. Chapter 5 summarizes key insights, discusses limitations, and suggests future research directions for further refinement and application of the performance calculation tool.

1.1. RESEARCH OBJECTIVE

The development of hybrid-electric propulsion systems is essential for the advancement of eVTOL aircraft within the UAM framework. While various hybrid-electric powertrain configurations have been explored in open literature, a standardized and flexible performance calculation tool remains underdeveloped. Existing studies typically assess specific configurations, limiting their applicability for comparative analysis across multiple architectures. To address this gap, this thesis focuses on the development of a robust performance calculation tool that enables the evaluation of multiple hybrid-electric propulsion system architectures under different operational conditions.

The objective of this research conducted within a collaborative framework between the Power and Propulsion group at the faculty of aerospace engineering of TU Delft and AYED ENGINEERING is formulated as follows:

> *To develop a robust performance calculation tool for hybrid propulsion systems in eVTOL aircraft, capable of simulating and evaluating the performance of multiple powertrain configuration variations.*

This tool is designed to accommodate different power sources, and will allow for comparative analysis of their performance in various hybrid-electric configurations. The insights derived from this tool will contribute to optimizing hybrid-electric propulsion for eVTOL applications and support the broader adoption of sustainable aviation technologies.

1.2. RESEARCH QUESTION(S)

The central research question guiding this thesis is:

> *How can a performance calculation tool be developed to accurately simulate and evaluate multiple hybrid-electric propulsion system configurations for eVTOL aircraft?*

The answer to this research question will be explored through multiple subquestions to provide a more structured and comprehensive analysis.

- How can individual propulsion system components be accurately modeled to reflect their dynamic interactions in a hybrid-electric powertrain?
- What level of fidelity is required for battery modeling to ensure realistic power distribution and state-of-charge predictions during flight?
- How can the tool be applied to evaluate different powertrain architectures under various mission profiles and operating conditions?
- What insights can be gained from comparative simulations of different hybrid-electric configurations, and how can these insights support eVTOL powertrain design decisions?

This structured approach ensures that the thesis systematically develops, implements, and validates the performance calculation tool while demonstrating its applicability for hybrid-electric eVTOL propulsion systems.

2

Background Information and Literature Review

This chapter establishes the theoretical foundation on which the performance-calculation framework is built. Section 2.1 traces the evolution from classical VTOL to distributed-electric propulsion and classifies the main hybrid-propulsion architectures. The discussion then turns to candidate on-board energy sources and associated power-management strategies, outlining how each option influences hybrid-propulsion sizing and control. Section 2.5 reviews state-of-the-art modeling techniques at both component and system level, highlighting the assumptions and fidelity choices adopted in this thesis. The chapter closes with the key takeaways that guide the methodology presented in Chapter 3.

2.1. eVTOL Fundamentals

The research on the development of aircraft that ascend and descend like a helicopter but flies as fast as an airplane has been around since the fifties [17]. The VTOL aircraft satisfies this requirement, while the STOL aircraft differs from this condition by having the need for small runway for take off and landing. The earlier VTOL aircraft were limited to a power plant design that either consists of a piston engine or a jet, wherein the efficiency of both engine types increases proportionally with engine size. The design choices for that period of time were limited to: having one power source for both lift and cruise functions or opting for distinct power sources dedicated to each application. The former was chosen for the first VTOL aircraft that was put in operation, the Harrier. Since the presence of multiple power plants on an aircraft results in diminished overall efficiency relative to a fixed total mass [17].

The Harrier can be categorized as a Vectored Thrust VTOL aircraft, a design in which the same propulsion system is used for both lift and cruise phases of the aircraft by manipulation of the thrust direction by its engine [13] [17]. This category can be further subdivided based on the component that is responsible for the thrust control: Tilt-wing, tilt-rotor and tilt-duct design, wherein the tilt-wing operates by employing a horizontal wing for forward motion, which rotates upward during vertical take-off and landing maneuvers. The tilt-rotor operates by having powered rotors that both deliver the thrust and the lift. The tilt-duct operates in a similar matter, but the rotor is inside a duct in order to prevent blade tip-losses [13] [18]. Another category of VTOL aircraft are the Lift + Cruise Aircraft. This type of design utilizes two different propulsion systems for the lift and cruise phase. The aircraft often possess a vertical propeller mounted on top a fixed wing. This fixed wing makes the Lift + Cruise design efficient at cruise phases [19]. The final category of VTOL aircraft is the wingless design. This configuration employs multiple rotors for both lift and cruise applications. The absence of a fixed wing results in relatively low weight, while the large rotor disk area contributes to lower disk loading, enhancing the aircraft's efficiency in hovering applications. The aircraft is less efficient in cruise also due to absence of the fixed wing [19]. An overview of the different VTOL/STOL configurations can be seen in figure 2.1.

Additionally, some future-oriented studies investigate high-temperature superconducting (HTS) materials to boost power density and reduce electrical losses in aviation [20]. Achieving and sustaining cryogenic temperatures, however, involves significant complexity, making practical HTS applications for eVTOL designs likely a few decades away [21, 22].

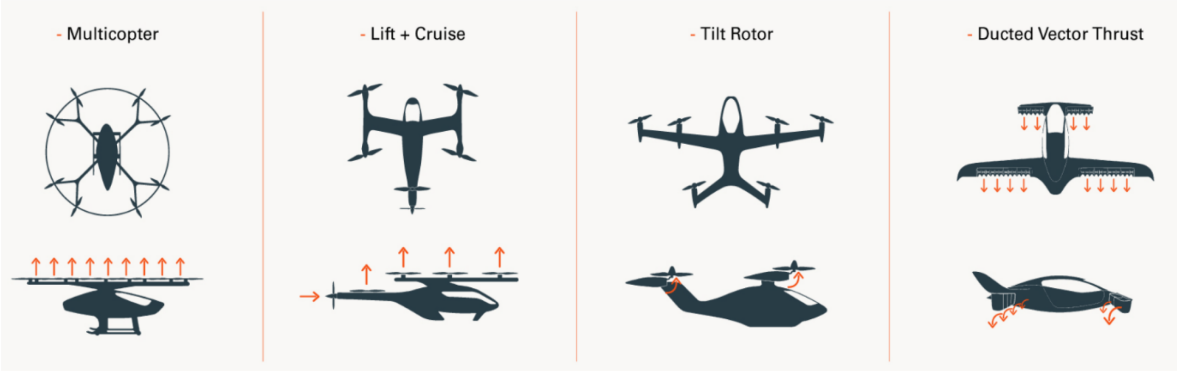


Figure 2.1: The different VTOL/STOL configurations, excluding the tilt-wing design [1].

2.1.1. DISTRIBUTED ELECTRIC PROPULSION

Distributed Electric Propulsion (DEP) has emerged as a promising architectural choice for eVTOL aircraft. Multiple electrically driven propulsors, arranged along the wings or fuselage, can reduce local wing loading, improve lift at low speeds, and potentially enhance fault tolerance [23]. While DEP may increase structural complexity and require additional power electronics, it supports boundary layer ingestion, wake filling, and powered yaw control, thus reducing required runway length or even enabling full VTOL in certain designs. Ongoing research focuses on balancing the aerodynamic gains of DEP with the mass penalties of extra motors and controllers [24].

Each concept faces technical hurdles in energy storage, mass constraints, noise abatement, and regulatory compliance. Batteries still impose range limits, while hydrogen fuel cells raise storage and dynamic response concerns. Even small turbogenerators add weight, but can extend endurance. Thus, propulsion architecture (selection of series, parallel, or fully electric design) profoundly affects the *hover-to-cruise* trade-offs, urban noise profiles, and overall mission viability. The subsequent chapters focus on these architecture differences and the key modeling approaches required to size and evaluate hybrid eVTOL powertrains.

2.2. HYBRID PROPULSION SYSTEMS

A non-DEP aircraft that has a hybrid propulsion system (HPS) utilizes two or more power sources to generate thrust. Similair to the DEP Aircraft there are three sorts of hybrid propulsion architectures, with all their own advantages and drawbacks: Series, parallel and series/parallel.

The series hybrid architecture is defined by the fact that the power generated by the power source is used to power the electric motors and subsequently the fans. The power source does not have a direct mechanical connection with the electric motor. For instance, if the two power sources present are a gas turbine engine coupled with an electric generator and a battery, both electrical outputs are connected with the electric motor. The main advantage of this architecture type is that the generator can always run at optimal speed to generate power since it is not mechanically coupled with the propulsor. The drawback of this system is the higher mass of the electric motor since it is individually responsible for the propulsive power [25].

The parallel hybrid architecture is characterized by two mechanically coupled propulsion shafts. The fan is driven by a shaft connected to both an electric motor and a gas turbine engine. Unlike the series hybrid architecture, where the fan is directly coupled to an electric motor and an electric generator is used for the gas turbine engine, the parallel hybrid architecture allows for simultaneous power generation from both power sources. The primary advantage of this architecture is that the aircraft can be powered by either or both power sources. However, the drawbacks include a more sophisticated propulsion control system and potentially less efficient operation of the gas turbine engine compared to the series architecture [25].

A combination of the aforementioned architectures leads to the series/parallel hybrid architecture. In this configuration, a power source, such as a turboshaft engine, can power both the electric generator and the electric motor simultaneously or separately. The advantages of this configuration include system flexibility and enhanced redundancy. Different flight phases require varying power demands, which can be effectively managed by this architecture. However, the major drawback is the increased complexity of the system. A schematic overview of the powertrain architectures can be seen in figure 2.2, including the turbo-electric concepts.

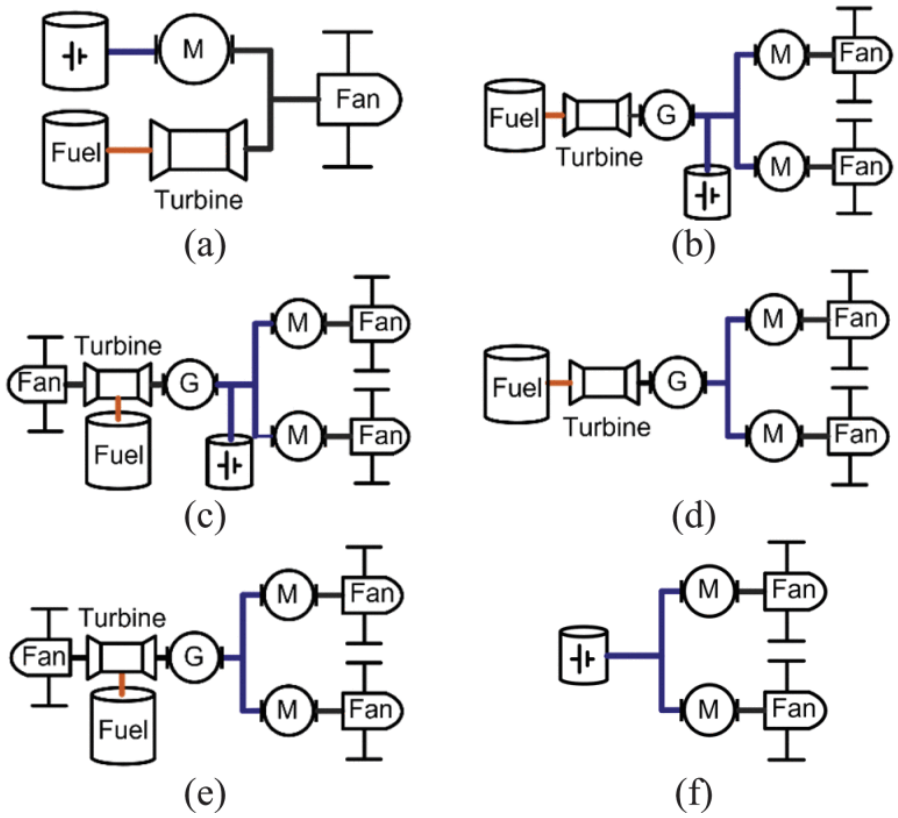


Figure 2.2: Six different hybrid powertrain architecture types. (a) Parallel hybrid (b) Series hybrid (c) Series/parallel hybrid (d) Fully turbo-electric (e) Partial turbo-electric (f) All electric [2].

2.3. POWER SOURCES OVERVIEW

The performance of varying HPS architectures is highly dependent on the power sources used in the configuration. The most popular electric power sources are the battery and the hydrogen fuel cell, while a HPS for aircraft often includes an internal combustion engine or gas turbine engine to meet the specified energy and power demands of the aircraft.

The level of hybridization of the aircraft gets denoted with a degree with respect to power H_P or energy H_E, which can be seen in the following equations.

$$H_P = \frac{P_m}{P_{tot}} \tag{2.1}$$

$$H_E = \frac{E_b}{E_{tot}} \tag{2.2}$$

The electric motor power gets denoted with P_m and the battery energy with E_b. An all electric vehicle will have a 1 for both H_E and H_P, while a turbofan engine will have a zero for both degrees. An aircraft that relies on turbo-electric propulsion with electric motors will have a 1 for H_P and a zero for H_E, since it does not store any electric energy in the system [26]. The most common power sources for HPS architectures are revised in the following subsections.

2.3.1. GAS TURBINE ENGINE

A power source commonly found in HPS is gas turbine engine. The hybrid system typically combines an electrical power source with a gas turbine engine, the conventional power source in modern aviation. Turbo-electric propulsion also falls under the gas turbine engine category. In this configuration, a gas turbine engine combined with an electric generator produces electrical energy. This electrical energy can then be used to charge the batteries in the hybrid system or to power the electric motors. The high power and energy density of the turbo-electric propulsion system allows for a greater variety of mission profiles. However, the emissions from this system are not conducive to achieving zero-emission goals[19].

2.3.2. BATTERY

The first electrical power source would be the battery, a device that has a similar built as a hydrogen fuel cell. It consists out of a two electrodes, a diaphragm and an electrolyte that allows the ions to pass through. Unlike a hydrogen fuel cell, a battery does not require an external fuel supply since the energy comes from the difference in Gibbs free energy between the reversible reactions that take place at its electrodes, making it a closed system. This free energy difference of the reactions at its two electrodes determine how much chemical energy is converted into electrical energy. Lithium ion batteries are the most popular choice for electric vehicles and aircraft [10] [6]. It has good electrochemical stability, a long battery life, high specific power and a high conversion rate [3]. Moreover, lithium, being the lightest metal and the most electropositive element, contributes significantly to the high energy density of lithium-ion batteries [20].

WORKING PRINCIPLE

The four aforementioned components of the battery each have distinct functions in the operation of a lithium-ion battery. The anode materials, such as lithium cobalt oxide, lithium manganate, and lithium nickel cobalt manganate, provide the lithium ions. Graphite is typically utilized as the cathode material. The diaphragm prevents electrons from moving freely between the cathode and the anode, while the electrolyte facilitates the passage of lithium ions.

During the charging process, an oxidation reaction occurs at the anode, where lithium atoms are oxidized to lithium ions by losing electrons. These ions migrate through the electrolyte and embed into the cathode, thus increasing the battery's state of charge. Conversely, during the discharge process, a reduction reaction takes place at the anode, enabling the lithium ions to return to the cathode via the electrolyte. Figure 2.3 illustrates the working principle of the lithium-ion battery in a schematic representation.

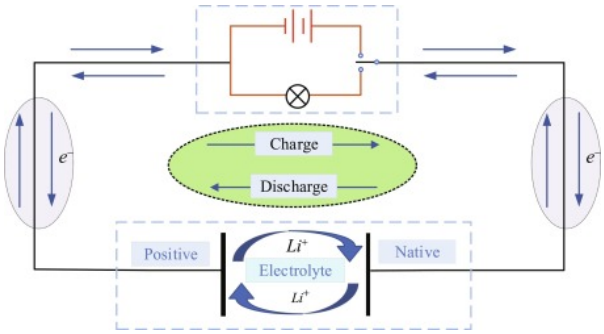


Figure 2.3: The working principle of a lithium-ion battery [3].

The relatively high power density characteristics of the battery make it more suitable for higher power flight segments such as take off and climb. However the energy density of the battery falls is comparatively low. A hybrid powertrain architecture that consists of a fuel cell and a battery is better suited for longer range mission profiles, in which the fuel cell is used as the main energy source and the battery energy storage is used supplementary during peak energy demand hours [19].

2.3.3. FUEL CELLS

Hydrogen fuel cells are also considered an electrical power source. They are electrochemical devices that convert the chemical energy of hydrogen atoms into electrical energy. A fuel cell consists of two electrodes that sandwich an electrolyte. The electrolyte blocks any electrons that try to pass through and allows hydrogen ions (or protons) to pass. Hydrogen gas separates with the help of a catalyst into hydrogen protons and electrons, the half reaction can be seen in equation 2.3. This process happens at the electrode called the anode. The hydrogen protons flow through the electrolyte towards the other electrode called the cathode, while an external circuit is used for the transportation of the electrons towards the cathode, generating electricity. Water is formed at the cathode from the hydrogen ions and the electrons. This half reaction can be seen in equation 2.4.

$$2H_2 \Rightarrow 4H^+ + 4e^- \tag{2.3}$$

$$O_2 + 4H^+ + 4e^- \Rightarrow 2H_2O \tag{2.4}$$

The output voltage of one hydrogen fuel cell is typically around 0.7 volts. However, multiple cells can be stacked in series to increase the total output voltage of the entire fuel cell stack [27].

The high energy density and low power density characteristics make it challenging for the fuel cell to be the sole propulsion mechanism for a VTOL. The hydrogen delivery system of the fuel cell associate the system with relatively slow dynamic responses [28][29]. The fuel cell's type is usually categorized by the sort of electrolyte, with the polymer electrolyte membrane (PEMFC) being the most widely used fuel cell for propulsion systems. These kind of fuel cells have a solid electrolyte, a long service life and are not so prone to corrosion. It operates in the temperature range of 50 to 100 degrees Celsius. It allows the hydrogen ions to pass through the membrane only when the membrane is sufficiently hydrated [27]. The fuel cell can easily function as the propulsion system for lower power flight segments such as descend and cruise of a VTOL/STOL aircraft.

BALANCE OF PLANTS COMPONENTS

A fuel cell is also accompanied by support systems that regulate the operation of the hydrogen fuel cell within the hybrid propulsion system. These support systems include a compressor that provides a continuous supply of air to the cathode, ensuring oxygen availability for the electrochemical reactions. The power density and reaction rate improve when the operation is performed at a high pressure level. Additionally, a humidifier is employed to maintain the necessary humidity levels in both the cathode's airflow and the anode's hydrogen flow. Furthermore, a heat exchanger is utilized to manage and reduce the temperature of air entering the cathode, thus enhancing the overall efficiency and longevity of the fuel cell system [27]. The auxiliary components are also known as the balance of plant components (BOP).

2.3.4. SUPERCAPACITORS

Supercapacitors (also known as ultracapacitors) can be considered as an electrical power source. However, due to their low energy storage capabilities, supercapacitors can only function effectively alongside other power sources within a hybrid propulsion system (HPS). For example, the shortcomings in a battery's power supply during high demand can be mitigated by using supercapacitors. They are characterized by their low specific energy and high specific power, making them suitable for mission profiles that require sudden transients in thrust. Supercapacitors can provide additional power during such events in a hybrid propulsion system [29]. The takeoff and landing phases are examples of these transient thrust applications. However, batteries generally have an adequate response time to handle these situations and can often meet the power demands during transient thrust events.

2.4. POWER DISTRIBUTION AND CONTROL

In hybrid propulsion systems, power management strategies are key to determining the optimal sizing and usage of multiple power sources. These strategies ensure that each power source is used efficiently throughout different operational phases, from takeoff to cruise and landing. The size and configuration of power sources—whether fuel cells, batteries, or other elements—are heavily dependent on the power management approach employed. Power management strategies can be broadly categorized into causal and non-causal approaches.

Non-causal strategies are most beneficial when the mission profile is known in advance. These strategies aim to optimize system performance across the entire mission, offering valuable insights for the design, sizing, and performance evaluation of hybrid propulsion systems. In contrast, causal strategies operate in real-time, adapting to the immediate conditions of the propulsion system. These strategies optimize power management at specific moments, dynamically responding to the system's needs based on current demand [30]. Both types of strategies influence how power is distributed between sources and, consequently, how power sources are sized, which directly affects the selection and performance of PE converters within the HPS.

The non-casual strategies are mainly based on computer algorithms that try to minimize the cost function, which stands for the fuel economy or emissions over a fixed mission profile. The casual strategies can be further categorized into rule based energy management strategies and optimization based strategies. The rule based energy management methods use a series of logic statements that determine which power source is used based on the state of the aircraft (State of Charge of the battery, Fuel Gauge, flight phase, etc.) The optimization based strategies consists of control methods, however system control falls out of the scope this research study.

Bindu and Thale proposed a power management and sizing methodology for hybrid electric vehicles, which is based on the energy and power demand per charge cycle [4]. Given that the mission profile, in this

case the Indian driving cycle, is predetermined, this approach can be classified as a non-causal strategy. The hybrid electric vehicle utilized two power sources: a battery and an ultracapacitor (UC). The battery was sized by modeling the energy and power requirements from the motor, based on the Indian driving cycle, which specifies the vehicle's velocity and acceleration as a function of time. As a result, the battery was dimensioned to meet the full energy demand of one complete driving cycle. To address the power demand, a power-sharing strategy was implemented between the battery and the ultracapacitor. The algorithm for power distribution is as follows:

- If the power requirement (PR) is less than zero (indicating regenerative braking), the UC is charged, and no power is supplied to the motor. The battery absorbs any remaining power once the UC's maximum charging capacity is reached.
- If PR > 0 and PR exceeds the battery power limit (P_{blim}), the battery supplies P_{blim}, with the ultracapacitor providing the remaining power.
- If PR > 0 and PR is less than P_{blim}, the battery supplies all the required power.

This strategy is illustrated in figure 2.4, which depicts the power demand throughout the driving cycle. The graph is color-coded to indicate the power source used at each moment, while also displaying the power limits of both the UC and the battery.

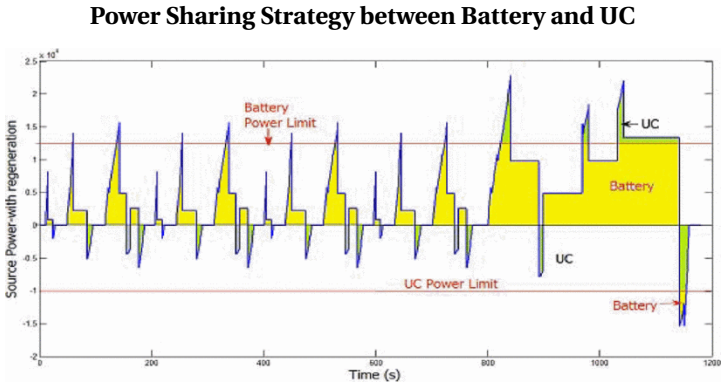


Figure 2.4: Visualization of the power sharing between the battery and ultracapacitor [4].

A similar strategy can be applied to hybrid eVTOL aircraft as well. By choosing a a primary power source that powers the aircraft up to a certain power limit. When the flight power requirement exceeds this power limit, the second power sources provides the additional power that is needed to satisfy the flight power requirement.

2.5. MODELING APPROACHES IN LITERATURE

Numerical modeling plays a critical role in the design process of hybrid propulsion systems (HPS). These systems are complex and require a structured approach to accurately simulate their performance and operational characteristics. This section explores the methodologies employed in modeling hybrid propulsion systems, emphasizing the significance of various components that comprise the overall system.

A well-structured system model is composed of multiple component models, each addressing a specific aspect of the overall system, such as gas turbines, electrical systems, and chemical reactions. Typically, these models involve combinations of ordinary differential equations (ODEs) and partial differential equations (PDEs) to manage the inherent complexities in such systems.

The modeling process begins with the selection of suitable models to adequately represent the physical behavior of each component. The next step is implementation, where software is developed to execute these models. This software uses variables that serve as placeholders for real-world values. The third step, parameterization, involves using real-world data to populate the model's variables. Finally, the simulation phase is conducted by running the software with these values to generate an output, approximating the physical world based on the chosen models [31].

2.5.1. BATTERY

The types of batteries can be categorized on their ability to recharge: Primary or secondary battery, Of which the primary battery is considered 'single use' and cannot be recharged and the secondary battery has the ability to recharge after the discharging process [10]. The latter category is the most interesting design choice in case of a hybrid propulsion system. The most prevalent battery types on the market are the Li-ion, lead acid, Ni-MH and Na-Cd batteries. These batteries, including their characteristics are shown in table 2.1.

Table 2.1: Popular battery types suitable for electric vehicles [10].

Battery Type	Service Life (cycles)	Nominal Voltage (V)	Energy Density (Wh/kg)	Power Density (W/kg)	Charging Efficiency (%)	Self-Discharge Rate (%/month)	Charging Temperature (°C)	Discharging Temperature (°C)
Li-ion	600 – 3000	3.2 – 3.7	100 – 270	250 – 680	80 – 90	3 – 10	0 to 45	-20 to 60
Lead acid	200 – 300	2.0	30 – 50	180	50 – 95	5	-20 to 50	-20 to 50
NiCd	1000	1.2	50 – 80	150	70 – 90	20	0 to 45	-20 to 65
NiMH	300 – 600	1.2	60 – 120	250 – 1000	65	30	0 to 45	-20 to 65

The inclusion of a battery pack inside a hybrid propulsion system also requires a battery management system (BMS), an essential unit that controls the management of energy and exposes the safety threats that happen within the battery [6]. A BMS consists of battery modeling, battery state estimation and a battery charging approach. The BMS captures the battery current, voltage and surface temperature via sensors, while the battery models further refine this data by filtering out noise. These battery models, in combination with state estimation models, collaboratively determine the State of Charge (SOC) and State of Health (SOH), all of which are crucial for monitoring and assessing battery life [5]. Additionally, the BMS can optimize the battery charging process by analyzing the thermal and electrical behavior of the battery [10]. It regulates the battery to maintain the desired temperature range by controlling the discharge rate and ensures balance by preventing overcharging and over-discharging of the battery cells.

BATTERY MODELS

Design of the BMS starts with finding the proper battery model to simulate the electric and thermal behaviour.

The most detailed models are known as electrochemical models, models that describe the internal electrochemical reactions and thermodynamics at cell level. These models are the most costly and require high computational power. The equivalent circuit models (ECM) capture the electric behaviour by making a simple circuit structure that consists of inductors, resistors and capacitors. Omitting high level detail makes the ECM faster than the electrochemical models. The analytical models are used to compute second or higher order differential equations to depict the battery operational effects. Analytical models are also used to predict the internal battery processes, grounded in the principles of thermodynamics, electrochemistry, and physics. The model does lack a physical basis, which makes physical interpretability relatively low. This is due to the fact that highly nonlinear electrochemical device storage devices is difficult to explain by mathematical equations [32]. Models such as the electro-thermal and the thermo-mechanical model are examples of combined models. These combines several sub-models coming from different disciplines [5].

The equivalent circuit model is the most used battery model in system simulation and management due to the representation of the model by relatively few components, making the mathematical state-space representation easier [32].

Another function of a battery management systems is the estimation of the state of the battery: The state of charge and state of health. The various SOC estimation methods are outlined in the following paragraphs.

BATTERY STATE ESTIMATION METHODS

The available battery capacity is represented by the SOC, where 100% SOC represents a fully charged battery and 0% describes a depleted battery. The battery's SOC can be portrayed as a sort of fuel gauge: the amount of energy that is available in the battery pack. There are several methods to estimate the battery's SOC. These methods can be categorized into three different types of methods: data driven, model based and direct methods [6].

Internal dynamics of the battery are learned via different measured data by a black box model in data driven methods. These kind of methods can only be utilized effectively if significant amount of data is available to train the deep learning tools inside the black box model. Data driven models are able to run real time simulations after nonlinear relationships are established between the battery SOC and factors such as, voltage, current and temperature [33]. Examples of data driven models are: Artificial Neural network and genetic

Model Nature		Model		Data	Physical Interpretability	Analogy	Accuracy	Complexity	Suited Application
Electro-chemical		Pure Electro-Chemical		P	H	White box	VH	H	Battery design
		ECM/Reduced order Electro-Chemical		SE	M	Grey box	H	M	
Electrical	Analytical	Peukert's model		E	L	Black box	M	M	Prediction
		Rakhmatov and Vrudhula		SE	M		M	M	
		Sheperd other iterations		SE	M		M	M	
		State-Space		E	L		M	M	
	ECM	Simples	Simple Rint	E	M	Grey box	L	L	Real time control, SoC estimation, ...
			Enhanced Rint	SE	M		L	L	
			RC	SE	M		L	L	
		Thevenin	1st order	SE	M		L–H	L–H	
			2nd order	SE	M				
			3rd order	SE	M				
			nth order	SE	M				
		PNGV	1st order	SE	M		L–H	L–H	
			2nd order	SE	M				
			nth order	SE	M				
		Noshin		SE	M		M	M	
		Neural nets		E	L		H	H	
	Impedance	Frequency domain		SE	L–M	Grey box	M	M	Characterization and real time operation
Thermal		Analytical Thermal		P–SE	H	White box	H	H	Real time
		ECM Thermal		SE	M	Grey box	M	M	
Mechanical/Fatigue		Fatigue/Mechanical		P–SE	H	Grey box		H	Design
Abstract model		Artificial Intelligence		E	L	Black box	M	M	Offline analysis
Combined models		Electro-Thermal		SE	M	Grey box	L–H	M	Real time
		Thermo-electrochemical		P	H			H	
		Thermo-Mechanical		SE	H			H	

* E: Empirical, H: High, L: Low, M: Medium, P: Physical, SE: Semi-Empirical, VH: Very High.

Figure 2.5: An overview of the battery models with their respective characteristics [5].

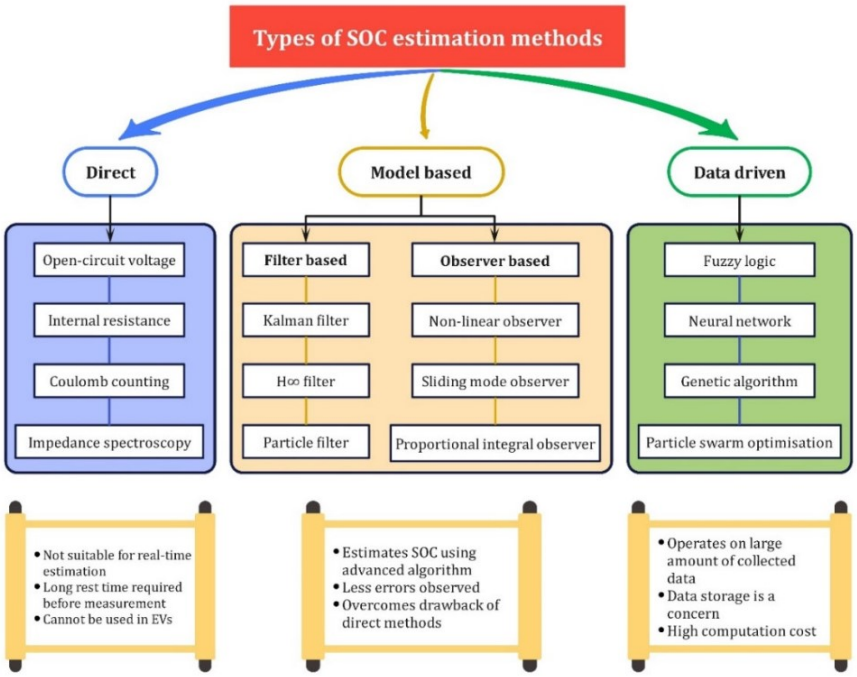


Figure 2.6: SOC methods and categorization [6].

algorithm. Direct methods rely on the measured data coming from the battery current and voltage sensors. A direct estimation method, the Ampere hour (Ah) method or Coulomb counting method is shown in equation 2.5.

$$SOC(k) = SOC(k_0) + \int_{k_0}^{k} \eta I(t) \mathrm{d}t / C_\mathrm{n} \tag{2.5}$$

Where the initial known SOC is depicted with $SOC(k_0)$. The efficiency of battery charging and discharging is denoted with η. C_n stands for the nominal capacity of the battery and the I(t) denotes the value of the current [34]. Equation 2.5 also shows that the Ah method is an integration of the current and therefore an efficient and fast method. However, a significant drawback of this method is the imprecise estimation of the initial SOC, which is highly dependent on the accuracy of the battery current sensors. Consequently, any errors in the battery current sensors accumulate during the integration process, leading to potential inaccuracies in the SOC estimation [33]. This method becomes very effective when the initial SOC is known and high precision sensors are used in the battery management system.

Another direct method approach is the Open Circuit Voltage (OCV) method. This method uses the nonlinear relation that exists between the SOC and OCV and is considered the most efficient method if an accurate OCV is known. The major drawback of this method is that the battery needs to reach internal equilibrium before the open circuit voltage of the battery can be measured. Generally speaking, this means that a Li-ion battery needs to rest for 10 hours before the OCV can be established and used to estimate the SOC. A way to tackle this limit of the OCV method is to combine a model based approach to calculate the OCV. After combining the OCV method with a model based approach, it becomes practical for real time applications [10] [33].

The model based approaches estimate the battery SOC by employing an established battery model in state-space representation, with one of the state variables being the battery's SOC. A closed loop estimation method is created with the measured voltage signals being its feedback. This makes the model based approaches more accurate than the most direct approaches and less complex than data driven approaches [35].

The deterioration of the battery's health is indicated by the battery's state of health. It reflects the current performance of the battery relative to that of a brand new battery. Degradation of the battery's capacity is a consequence of battery aging. This will cause an increased internal resistance and consequently affect the battery's performance. The most widely used definitions of the battery's SOH are the resistance form and the capacity form seen in equation 2.6 and equation 2.7, respectively.

$$SOH = \frac{R_{EOL} - R_\mathrm{aged}}{R_\mathrm{EOL} - R_\mathrm{fresh}} \times 100\% \tag{2.6}$$

$$SOH = \frac{C_\mathrm{aged}}{C_\mathrm{fresh}} \times 100\% \tag{2.7}$$

Where R stands for the internal resistance of the battery and C stands for the battery capacity. The subscript 'EOL' represents the end-of-life stage, 'aged' represents the current life stage and 'fresh' is used to represent the battery characteristics when the is at the start of its service life [36]. Similar to SOC estimation methods, SOH estimation methods can be categorized into three distinct groups: direct methods, data-based methods, and model-based methods. The charge distribution influences both SOC and SOH in similar ways, resulting in notable similarities in the methodologies used to predict these two states [6].

BATTERY CHARGING APPROACHES

Another function of the battery management system is to regulate the charging of the battery. This involves objectives like: reducing temperature rises during charging, shortening the charging time, maximising the battery lifespan and optimizing the charging efficiency. The most popular charging approaches can be categorized into four different methods: The constant-current (CC), constant-voltage (CV), hybrid CC-CV and the multi-stage constant current (MCC) method [10] [37].

The CC charging method employs a low constant current rate to charge the battery, ceasing the process once the SOC reaches a predetermined threshold. The primary challenge of this straightforward model lies in balancing a high current rate, which enhances charging speed but potentially reduces battery life, against a low current rate, which prolongs charging time, thereby rendering the battery impractical for EV applications.

The CV charging method excels in preserving battery lifespan by preventing irreversible reactions and over-voltage during charging. In this method, the current rate is initially high when the battery's SOC is low to

maintain a constant terminal voltage. As charging progresses, the current rate gradually decreases. However, the initial high current rate can be detrimental, leading to battery degradation.

The CC-CV method takes the best of both charging approaches and combines them into one charging method. This method comprises two phases: in the first phase, the CC method is applied to achieve a specific terminal voltage, enabling relatively rapid charging without excessively high current rates. In the second phase, the CV method is employed to prevent overcharging, thereby ensuring a safe and efficient charging process[37].

The MCC method charges the battery by using multiple constant current rate levels of different magnitudes. Determining the optimal charging profile remains a central focus in the application of the MCC method. Research has concentrated on using experimental design techniques and computer algorithms to identify the optimal charging profile, highlighting the complexity of this charging strategy. The advantages of the MCC method include reduced charging duration, extended cycle life, and high charging efficiency[37].

2.5.2. HYDROGEN FUEL CELL

A hydrogen fuel cell can be modeled by subdividing the stack into four submodels: cathode flow, anode flow, membrane hydration, and stack voltage. Although temperature variations can be captured by an additional thermal submodel, many studies (including [27]) treat the fuel cell temperature as uniform over short time horizons, given the slow stack thermal dynamics relative to electrical transients.

In the performance calculation tool by Khalil [9], the open-source OPEM tool [38] was employed to model a PEMFC using the Amphlett static model. This semi-empirical approach estimates the net stack voltage by accounting for the main voltage drops: activation, ohmic, and concentration losses.

The PEMFC operation is largely depended on the the balance-of-plant (BOP) components, which includes a compressor, humidifier and a thermal management system. The humidity level in the membrane is managed by adding or removing water vapor, ensuring the fuel cell remains within optimal operating conditions. Flooded fuel cells or dry membranes cause high polarization losses. Ohmic losses depend on membrane resistivity, which is influenced by temperature, pressure, and humidity [27].

To maintain correct air/fuel stoichiometry under varying loads, the reactant flow subsystem typically relies on separate loops for hydrogen and air supply [27]. As current is drawn from the fuel cell, the compressor must deliver sufficient oxygen at the cathode to keep excess ratios high and respond quickly to power demands. In high-pressure designs, the fuel cell itself may drive the compressor for compactness and efficiency, but this tight coupling can produce a temporary voltage drop, an inverse response, when power demand suddenly increases. Such non minimum phase behavior complicates closed-loop control, limiting how fast the system can react [27].

The thermal management system is there to dissipate the heat that is generated by the fuel cell when current is drawn from the stack. Around 50% of the chemical energy stored in hydrogen will convert into heat when using a fuel cell [39], while the PEM fuel cell is designed to operate at a temperature of 80 degrees Celsius [27]. The most common thermal management technique in PEMFC operation is liquid cooling [9] [7]. This technique relies on the passing of coolant through the channels between the cells of the fuel cell stack. A radiator cools the heated coolant, which can then be re-used to cool the fuel cell stack. A schematic of this cooling technique can be seen in figure 2.7.

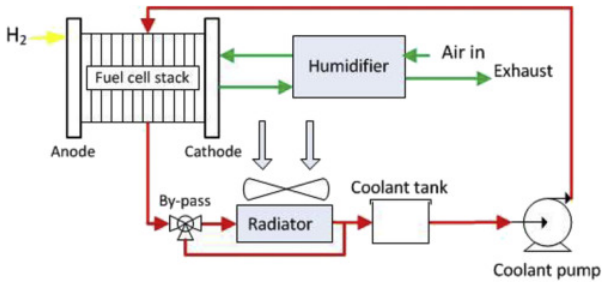


Figure 2.7: The technique of liquid cooling used in PEMFC operation [7].

Maintaining the correct air/fuel stoichiometry is critical for stable PEMFC performance; any mismatch in reactant supply can reduce efficiency or cause damaging fluctuations in stack voltage [27]. Practical implementations typically strive to avoid conditions leading to either under-supplied cathodes or over-powered

compressors at low load, ensuring a reliable and efficient fuel cell operation throughout a typical eVTOL or hybrid-electric mission profile.

2.5.3. TURBOGENERATOR

A turbogenerator is a power source used in turbo-electric propulsion systems. It consists of a gas turbine engine connected via a shaft to an electric generator, enabling the conversion of mechanical energy into electrical energy. As a result, the component model for the turbogenerator is divided into two submodels: one representing the gas turbine engine and the other representing the generator.

The gas turbine engine can be modeled by The Gas Turbine Simulation Program (GSP), a simulation program developed by the Delft University of Technology and the Netherlands Aerospace Centre (NLR) [40]. The program is able to simulate and evaluate the performance of gas turbine engines. GSP enables the modeling of various gas turbine configurations by integrating zero-dimensional component submodels, which calculate average pressure and temperature values at key points between components. The program uses turbomachinery performance maps in conjunction with aero-thermodynamic equations to assess both individual component performance and the overall performance of the gas turbine engine. A significant feature of GSP, especially in its integration with the HPS performance calculation tool, is its ability to accurately predict emissions across mission profiles and analyze the impact of alternative fuels on the gas turbine engine [40]. This capability is particularly valuable for assessing HPS configurations where the hydrogen fuel cell and turbogenerator share a common fuel tank.

The most commonly used AC generator in aircraft is the three-stage wound field synchronous generator (WFSG). This generator is favored for its robust safety features, particularly its ability to directly control the magnetic field within the rotor [41]. The permanent magnet synchronous generator (PMSG), which shares a similar design with the WFSG, is also frequently used in aviation. However, for the purpose of modeling a hybrid propulsion system, a more generalized component, known as a synchronous generator, will be employed.

A synchronous generator is a machine that consists of a rotor with field windings and a stator with armature windings. The rotor gets rotated by the prime mover, creating a rotating magnetic field. An electromotive force is induced by this rotating magnetic field inside the armature windings of the stator, consequently an alternating current is generated as output of the synchronous generator [8]. A schematic of a synchronous generator can be seen in figure 2.8. It shows the rotational direction of the rotor, the field windings inside the rotor, the stator and its armature windings and the axes of the three phases. The electrical rotational speed of the rotor is directly linked with the frequency of the generated AC power, hence why the generator is called "synchronous".

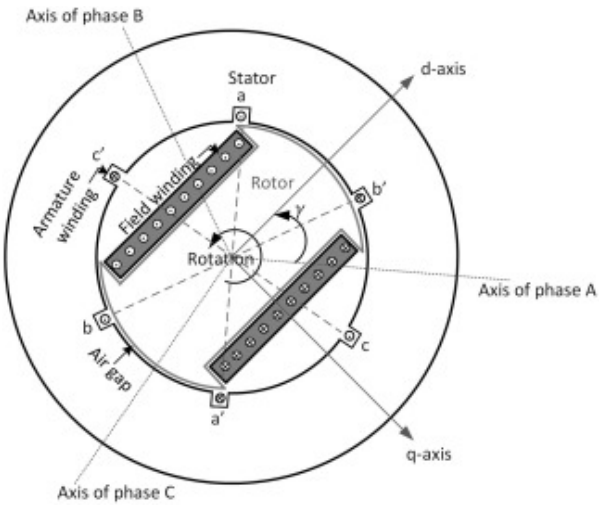


Figure 2.8: Schematic of a three phase synchronous generator [8].

The relationship between the mechanical rotational speed, ω_m, and the electrical rotational speed, ω, is defined by the equation

$$\omega = \omega_m n$$

where n represents the number of pole pairs. In a generator, the armature windings are arranged so that the three load phases are spaced 120 electrical degrees apart, as shown in Figure 2.8. As a result, when the rotor of a generator with a single pole pair rotates at electrical speed ω, the induced voltages in the windings will have a phase difference of 120 degrees.

As the rotor rotates relative to the stator, time-varying inductances are generated, making it challenging to model the behavior of a synchronous machine. Figure 2.8 also depicts the three armature windings in the stator, corresponding to the abc axes, along with the two axes of symmetry for the rotor field windings, corresponding to the d-axis and q-axis.

Synchronous generator modeling is typically done in either the abc reference frame or the $dq0$ reference frame. The $dq0$ reference frame simplifies integration into hybrid propulsion system (HPS) performance calculation tools by eliminating the complexity of time-varying inductances in the three-phase load. However, using the $dq0$ frame requires a mathematical transformation between the two reference frames, converting the machine's behavior from the abc system to the more manageable $dq0$ representation [8].

2.6. SYSTEM MODELING

System modeling integrates the above components into a coherent framework, respecting flight mission profiles, control constraints, and real-time power demands. After each component (battery, fuel cell, turbogenerator) is chosen or parameterized, the entire HPS can be simulated at a range of operating conditions to assess efficiency, mass, and emissions trade-offs.

2.7. SOFTWARE IMPLEMENTATION

Literature has shown that MATLAB in combination with Simulink is a popular choice to perform system level simulations for hybrid propulsion systems. Bell and Litt [42] have developed such a simulation toolbox that enables hybrid propulsion system modeling within the MATLAB Simulink environment. The Electrical Modeling and Thermal Analysis Toolbox (EMTAT) is a Simulink library that contains electrical components to be used in conjunction with the known turbomachinery toolbox T-MATS.

One of the primary strengths of Simulink lies in its graphical user interface, which enables users to construct models of physical systems by intuitively arranging block diagrams. This graphical and modular approach not only facilitates the reconfiguration of systems but also simplifies the addition or removal of components and submodels. EMTAT itself is composed of two types of models: physics-based component models and power flow component models. The physics-based models, grounded in first-principles equations, deliver a more accurate representation of electrical outputs compared to the semi-empirical power flow models. Conversely, the power flow models represent electrical components as equivalent circuits, enabling significantly faster simulation times relative to the physics-based models. However, the discrepancy between the electrical dynamics and turbomachinery dynamics allows for the physics based models to be treated as already having reached steady state at each integration step of the turbomachinery, meaning that the integration step may be increased to ensure faster simulation times [42].

Another notable simulation tool developed within the MATLAB environment for modeling hybrid propulsion systems is the Aircraft Propulsion System Simulation (APSS) developed by Schmitz and Hornung [43]. APSS is designed to perform both on-design and off-design performance calculations for user-defined propulsion configurations, including fully electric and hybrid-electric systems. The decision to implement APSS in the MATLAB environment is driven by the availability of built-in functions that are essential for complex engineering analyses. These functions include optimization algorithms, numerical integration techniques, multidimensional interpolation, and the ability to perform scalar-based calculations efficiently through MATLAB's vectorization capabilities. These features collectively enhance the tool's computational performance and flexibility, making it well-suited for the rigorous demands of hybrid propulsion system simulation.

While MATLAB, particularly Simulink, employs a modular approach to physical system simulations, the Modelica programming language provides an object-oriented framework for system modeling. Arzberger and Zimmer [44] developed a Modelica library specifically designed to facilitate the modeling of hybrid propulsion system (HPS) architectures in aviation. The choice of an object-oriented programming language, such as Modelica, was motivated by two key factors. Firstly, the Modelica library is designed to support interchangeable models, allowing users to adjust the level of fidelity based on the specific needs of different investigations. This flexibility is crucial because the computational time of a simulation is directly influenced by the complexity and level of detail in the modeling. Object-oriented programming enables the creation of models

with varying degrees of complexity and detail, which can be easily swapped or adjusted as required. Secondly, an object-oriented approach provides a flexible framework for various powertrain architectures, allowing for easy customization of the number of subsystems and their interconnections. The architecture framework also reduces the complexity of individual subsystems, which helps to lower computational time.

The Modelica programming language has also been used by Podlaski et al. [45] and the Center for High-Efficiency Electrical Technologies for Aircraft (CHEETA) to develop HPS for aircraft. Podlaski et al. shared the same reasons for using an object oriented programming language as Azberger and Zimmer.

Another programming language worth noting for system-level simulations of propulsion systems is Python. Python is a widely adopted, open-source, object-oriented programming language that is accessible to a broad spectrum of developers and researchers. This large and active community has contributed to a vast array of libraries and packages, covering areas from engineering to data science and machine learning, which can be leveraged in the development of a hybrid propulsion system HPS model.

One of the key advantages of Python is that it is open-source, making it freely accessible to everyone. This allows educational institutions, startups, and individuals to perform numerical computations similar to those done with MATLAB, without the need for costly licenses. Python's accessibility has also fostered the creation of specialized packages and toolboxes by developers, such as the Open-Source PEMFC Simulation Tool (OPEM) for the development of PEMFC models [38]. These toolboxes significantly reduce the time required to develop hybrid propulsion system models, enabling more efficient allocation of resources throughout the project.

2.8. KEY TAKEAWAYS

This chapter has provided an overview of eVTOL propulsion architectures, the various power sources available for hybrid propulsion, and the modeling approaches found in literature. Based on this review, the following methodological choices have been established for the development of the performance calculation tool:

- *Powertrain Architecture:* A series hybrid configuration is selected due to its relatively simple layout and efficient operation. As outlined in Section 2.2, the series hybrid architecture provides greater flexibility in power management and improved energy efficiency, making it a suitable candidate for eVTOL applications.

- *Power Sources:* The performance calculation tool models a hybrid propulsion system incorporating a hydrogen fuel cell, a battery, and a turbogenerator. These selections align with findings in Section 2.3, where the fuel cell is identified as an efficient primary energy source, the battery is well-suited for transient power demands, and the turbogenerator extends endurance for longer missions. Although supercapacitors offer high specific power and can enhance transient performance, modern high-power batteries generally meet these requirements, reducing the necessity for additional power buffering components.

- *Component Models:* The component-level modeling approach is informed by the literature review. The battery is modeled using a second-order Thevenin equivalent circuit model, balancing computational efficiency with adequate representation of transient electrical behavior (Section 2.5.1). The fuel cell is modeled using the Amphlett static model, which captures activation, ohmic, and concentration losses while remaining computationally feasible (Section 2.5.2). The turbogenerator model is precomputed using the Gas Turbine Simulation Program (GSP), a specialized tool for gas turbine performance analysis (Section 2.5.3).

- *Power Management Strategy:* The tool employs a modified non-causal power management strategy, based on the methodology proposed by Bindu and Thale [4]. The non-causal nature of this approach allows for optimization of system performance across the entire mission, offering insights into energy distribution, component sizing, and operational efficiency of the hybrid propulsion system.

- *Software Implementation:* Python is selected as the primary simulation platform due to its modularity, seamless integration with optimization algorithms, and robust support for system-level modeling. This choice is reinforced by the discussion in Section 2.7, which highlights Python's flexibility compared to MATLAB and Modelica, particularly in terms of adaptability for hybrid electric powertrain simulations.

Having established these design choices, the following chapter presents the methodology used to integrate these models into a unified system simulation framework, enabling performance evaluation and optimization of hybrid eVTOL propulsion architectures.

3

METHODOLOGY

This chapter presents the methodology employed in developing a modular performance calculation tool for hybrid propulsion systems in eVTOL and eSTOL aircraft. Building upon the theoretical framework established in Chapter 2, the methodology is structured to support the design, sizing, and performance evaluation of various hybrid propulsion architectures. The chapter begins with an overview of the flight mission and the corresponding modeling requirements. Subsequently, the system modeling approach is introduced, detailing the system boundaries, key variables, relevant physical phenomena, modeling assumptions, and component models. Finally, the chapter concludes with a discussion on the integration of the component models into a unified system-level simulation.

3.1. MISSION PROFILE AND REQUIREMENTS

The performance of the hybrid propulsion system will be assessed along an exemplary mission profile for eVTOL aircraft that departs under standard atmospheric conditions. Besides the ordinary flight phases such as, take-off, climb, cruise, descent and landing, the mission profile also includes an emergency segment during its final phase. The power profile of the eVTOL aircraft can be seen alongside the mission profile in figure 3.1. The flight characteristics of the mission can be seen in table 3.1.

Table 3.1: Flight characteristics of the exemplary eVTOL mission.

Flight Phase	Time (s)	Range (km)	Altitude (m)	Power (kW)	Energy (kWh)	Airspeed (m/s)
Idle	300	0	0.1	0	0	0
Take Off	50	0	50	1114	15	0
Ascend	500	20	3000	542	75	40
Cruise	3600	180	3000	207	207	50
Descend	1200	50	50	56	19	42
Landing	50	0	0.1	1111	15	0
Idle	300	0	0.1	0	0	0
Take Off Emergency	60	0	50	1111	19	0
Cruise Emergency	120	5	50	172	6	42
Landing Emergency	60	0	0.1	1107	18	0
Idle	300	0	0.1	0	0	0

The propulsion system must be able to provide for the entire net electric power and energy to the electric motor as can be seen in table 3.1. Eventually, the eVTOL aircraft will possess a maximum takeoff weight (MTOW) of 3175 kg, which includes the aircraft's structural weight of 1905 kg. Consequently, the complete hybrid propulsion system needs to be integrated into the eVTOL while adhering to these weight specifications.

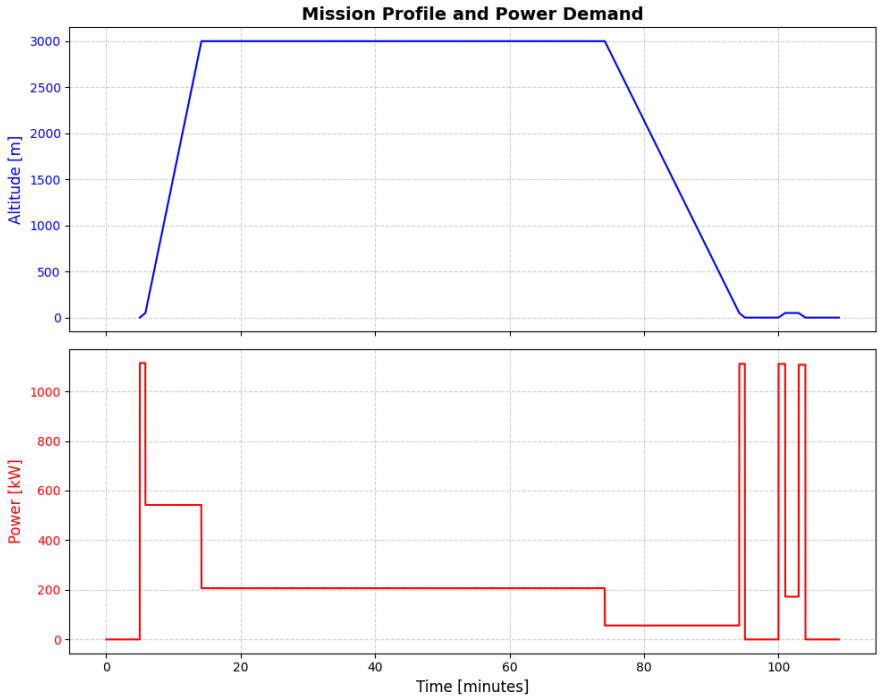


Figure 3.1: The mission profile and power demand for the eVTOL aircraft.

3.2. SYSTEM MODELING

The performance calculation tool is designed to evaluate multiple powertrain architectures, which requires flexibility in the system modeling approach. Each powertrain architecture features a unique configuration of propulsion components, and the tool adapts accordingly to simulate their performance. The system model for the hybrid propulsion system includes the following components:

- Hydrogen fuel cells
- Battery packs
- Turbogenerators: gas turbine engines & generators
- Power electronic (PE) converters: DC/DC, rectifier and inverter

It is important to note that not all components are present in every powertrain architecture, as configurations vary based on specific design objectives. The modeling of the power sources are described in Section 3.3. Three different powertrain architectures will be modeled with the use of the performance calculation tool, which are the following:

Hydrogen Fuel Cell and Battery Pack The first powertrain architecture (FC-BAT) consists of a hydrogen fuel cell accompanied by a battery pack in a series hybrid powertrain architecture. The power sources are used to power the electric motor, as can be seen in next following subsection in figure 3.2. Both sources are connected to the DC bus by the usage of DC/DC Converters and their power is converted to AC current by the inverter before it reaches the electric motor.

Battery Pack Another powertrain architecture that is be modeled utilizes a sole battery pack (BAT-Only) in the propulsion system, so there are no additional power sources that power the electric motors. The battery pack must be sized to meet the full power and energy demands of the mission. The battery output is routed through a DC/DC converter and then inverted to AC before reaching the electric motor.

Hydrogen Fuel Cell and Turbogenerator The last powertrain architecture (FC-GT) consists of a hydrogen fuel cell accompanied by a turbogenerator that runs on hydrogen. It is modeled in a series hybrid powertrain architecture. The turbogenerator is connected to a rectifier to convert AC to DC , while the hydrogen fuel cell connects to a DC/DC converter. This powertrain architecture can also be seen in figure 3.3 in the following subsection.

3.2.1. SYSTEM BOUNDARIES AND DESIGN VARIABLES

A simplified scheme for the FC-BAT can be seen in figure 3.2, while the system variables for the FC-BAT architecture are listed below:

INPUTS

- Mission profile (power profile, altitude)
- Ambient conditions (temperature. pressure)
- Battery parameters (battery characteristics, initial state of charge)

OUTPUTS

- Electrical power output
- Hybrid propulsion system mass (FC, battery, BOP components, PE converters)
- Waste heat from the hybrid propulsion system
- Parasitic power loss (compressor, thermal management system)
- Total hydrogen consumption

The primary distinction between the FC-BAT and BAT-Only architectures is the integration of the fuel cell system and its associated balance-of-plant components, all other aspects of the model remain identical.

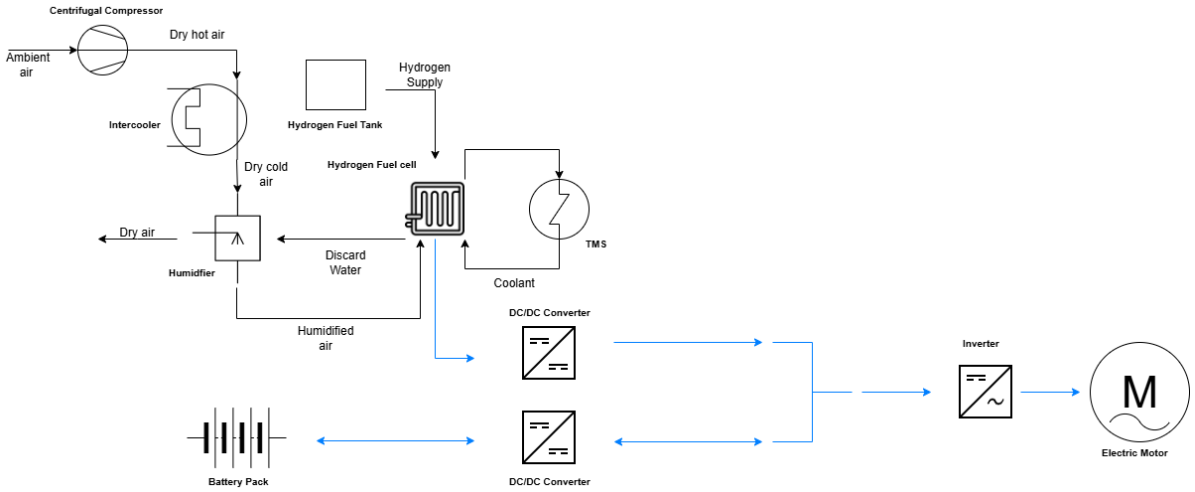


Figure 3.2: A simplified powertrain architecture for the FC-BAT propulsion system.

A more straightforward layout of the FC-GT architecture in figure 3.3. This powertrain layout had been evaluated in an earlier research project [9]. However, it will be re-evaluated to compare the outcomes from both performance calculation methods.

INPUTS

- Mission profile (power profile, altitude)
- Ambient conditions (temperature. pressure)
- System Voltage

OUTPUTS

- Hybrid propulsion system mass (FC, Turbogenerator, BOP components, PE converters)
- Waste heat from the hybrid propulsion system
- Parasitic power loss (compressor, thermal management system)
- Total hydrogen consumption

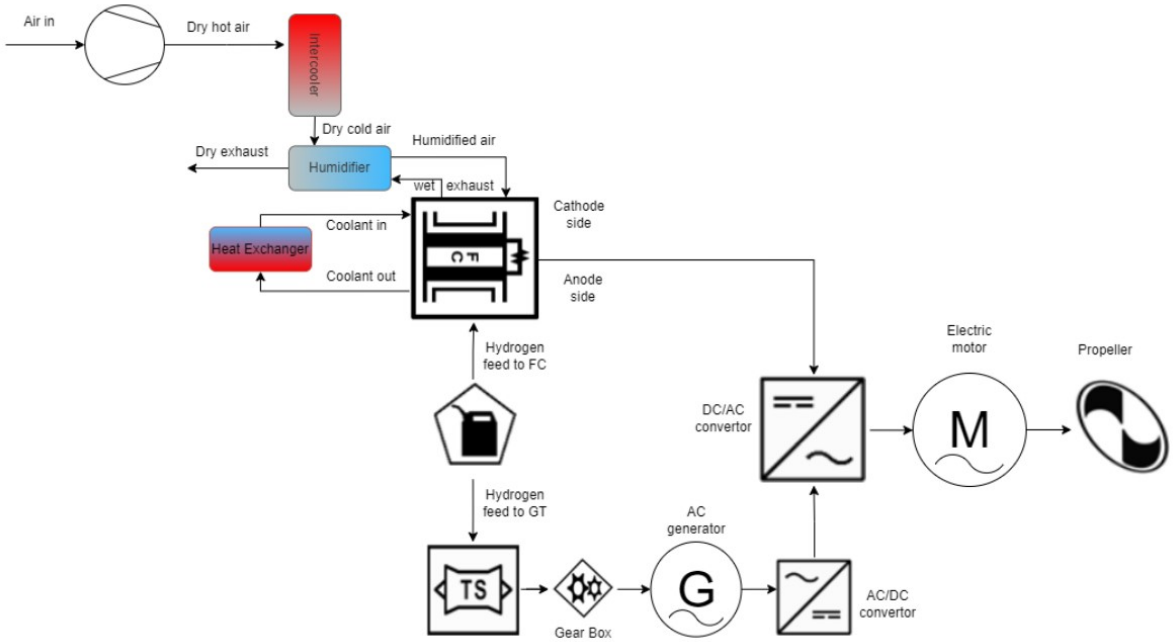


Figure 3.3: A simplified powertrain scheme for the fuel cell - turbogenerator hybrid evtol [9].

3.2.2. RELEVANT PHENOMENA

The section highlights the relevant physical phenomena that influence the performance of the hybrid propulsion system.

- There is variation in ambient conditions based on the altitude of the flight segment.
- There is variation in mass flow rate based on the flight speed.
- To execute the flight mission, the electric motors require energy from the hybrid propulsion system.
- In both cases, the hydrogen fuel cell acts as the main power source, while the secondary power source supplements during high power segments.
- The battery experiences heat generation due to the internal resistance under high current loads.
- The battery may get charged during flight segments where the fuel provides more power than the power demand of the electric motors.
- The hydrogen fuel cell is supported by additional balance of plants components that operate entirely on the energy provided by the hydrogen fuel cell.
- The thermal management system keeps the fuel cell operating at a temperature of 80 degrees Celsius.
- Compression of air takes place in compressor that supplies the hydrogen fuel cell.
- Waste heat from the fuel cell is being managed by the thermal management system.
- The supply of air to the hydrogen fuel cell ensures a hydrogen-oxygen stoichiometric ratio of two.

3.2.3. ASSUMPTIONS

The modeling of the hybrid propulsion system incorporates the following assumptions, categorized at the system level, component level, and specific subsystems:

SYSTEM-LEVEL ASSUMPTIONS

- The system is represented using a 0-D, steady-state model.
- Variations in potential energy between stations of different components are considered negligible.
- The weight of the PE converters is estimated using state of the art specific power value of 7500 Wkg^{-1} [46] [47].

BATTERY MODEL ASSUMPTIONS

- Internal resistances and capacitances are treated as constants and are assumed to be independent of temperature, state of charge (SOC), or discharge current.
- Cell-to-cell imbalances within the battery pack are not modeled. All cells are assumed to share identical SOC values.
- Battery aging effects are not included, and the nominal capacity of each cell remains constant regardless of the number of cycles or age.
- The weight of the battery management system and battery thermal management system is to be included in the 25% battery pack overhead mass [48].
- Overcharging or over-discharging does not occur. The SOC is constrained to remain between 0 and 1.

FUEL CELL ASSUMPTIONS

- The fuel cell stack is modeled without local temperature or concentration gradients.
- Accumulation of liquid water at the cathode is neglected.
- Chemical reactions are assumed to occur at ideal stoichiometric conditions.
- Both hydrogen and air are treated as ideal gases.
- The operating temperature of the fuel cell stack is maintained constant at 80 degrees Celsius.
- The hydrogen fuel utilization efficiency is assumed to be 95% [38].
- The pressure drop across the humidifier is considered negligible.
- The electronic resistance, $R_{\text{electronic}}$, is assumed to be much smaller than the protonic resistance, R_{proton}, and is therefore neglected [38].
- Membrane and electrode degradation effects are excluded, and their performance is assumed to remain constant.

TURBOGENERATOR ASSUMPTIONS

- The working fluid behaves like an ideal gas [49].
- The compression process in the compressor can be considered as adiabatic [49].
- The losses due to transmission of expansion power are accounted for in the mechanical efficiency. [49].

BALANCE-OF-PLANT COMPONENT ASSUMPTIONS

- The efficiencies of the generator, electric motor, and compressor are modeled as constants, with values of 90%, 90%, and 80%, respectively [50].
- The compression ratio adjusts with altitude to maintain a constant operating pressure for the hydrogen fuel cell.
- The inlet to the compressor is assumed to be ideal, with frictional effects ignored.

3.3. COMPONENT MODELING

The selection of component models in this study is directly informed by the literature findings outlined in Chapter 2. The battery model relies on an equivalent circuit representation to capture dynamic electrical behavior while ensuring computational efficiency. The fuel cell model follows the Amphlett approach to account for voltage losses and electrochemical characteristics. The turbogenerator model leverages the Gas Turbine Simulation Program (GSP) to integrate precomputed performance data. Each component model is integrated into a unified system-level simulation to evaluate hybrid propulsion architectures under real-world mission conditions.

3.3.1. BATTERY

The battery is modeled in Python using an equivalent electrical circuit model of a battery cell. The battery pack is then sized according to the power and energy demand of the eVTOL mission.

To accurately simulate the performance of the battery within the hybrid propulsion system, a second-order Thevenin model is implemented. The methodology for constructing and parameterizing the model began with fundamental equivalent circuit principles and progressed to a second-order configuration.

The modeling process started with a simple circuit representing the battery as an ideal voltage source delivering a constant output voltage at its terminals. The voltage source corresponds to the open-circuit voltage (OCV), which is the battery's voltage under equilibrium conditions, unaffected by load currents. The schematic of this basic model is shown in Figure 3.4. To improve this model, the OCV was modeled as a function of the state of charge (SOC), OCV($z(t)$), to account for the relationship between the battery's SOC and terminal voltage. The OCV and the state of charge follow a non-linear relationship, which can be accessed via a look up table. The SOC was calculated using the following differential equation:

$$\dot{z}(t) = -\frac{\eta(t), i(t)}{Q} \tag{3.1}$$

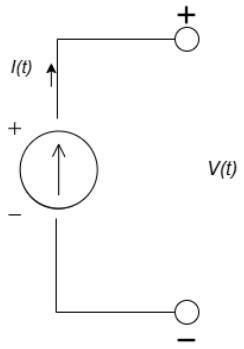


Figure 3.4: A simple equivalent electrical circuit model.

Here, $\eta(t)$ represents the Coulombic efficiency (assumed to be 1 during discharge and 0.95 during charging), $i(t)$ is the load current, where a positive values corresponds to discharge and a negative value to charging. Q is the battery capacity in ampere-hours (Ah). The SOC over time can be derived by integrating this equation:

$$z(t) = z(t_0) - \frac{1}{Q}\int_{t_0}^{t} \eta(\tau), i(\tau), d\tau \tag{3.2}$$

This improved model replaces the ideal voltage source with a voltage source dependent on SOC. The relationship between the SOC and the OCV is non-linear and is unique to every battery cell. The improved model is shown in Figure 3.5.

To incorporate the effect of internal resistance, R_0, the model was extended to account for the instantaneous polarization effect, where the terminal voltage $v(t)$ drops below the open-circuit voltage when the battery is subjected to a load current. The schematic can be shown in the relationship described by:

$$v(t) = \text{OCV}(z(t)) - i(t)R_0 \tag{3.3}$$

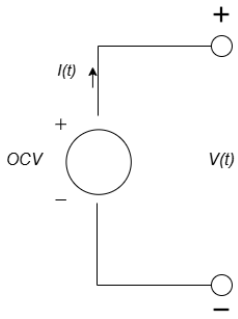


Figure 3.5: A simple equivalent electrical circuit model with an SOC dependent Open-Circuit-Voltage.

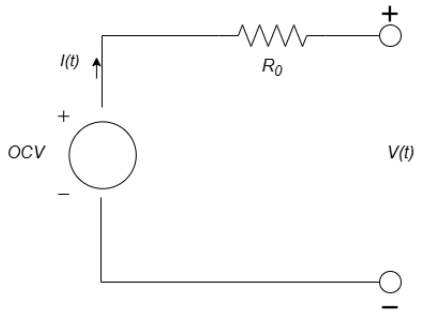


Figure 3.6: An ECM that includes a series resistor.

To capture the battery's transient response to load variations, a resistor-capacitor (RC) network was added in series with R_0. The RC network models the diffusion voltage effect, which reflects the slow change in voltage in response to current pulses. This first-order RC network includes a resistor R_1 and a capacitor C_1. The dynamics of the RC network are described by:

$$\frac{\mathrm{d}i_{R_1}(t)}{\mathrm{d}t} = -\frac{1}{R_1 C_1} i_{R_1}(t) + \frac{1}{R_1 C_1} i(t) \tag{3.4}$$

The terminal voltage equation is updated to:

$$v(t) = \mathrm{OCV}(z(t)) - R_1 i_{R_1}(t) - R_0 i(t) \tag{3.5}$$

Adding a second RC network (composed of R_2 and C_2) enhances the model's ability to capture more complex dynamic behavior. This results in the second-order Thevenin model, whose schematic is shown in Figure 3.7. The equations governing the second-order model can be seen in equation 3.6.

$$\begin{aligned}
\dot{z}(t) &= -\eta(t)\frac{i(t)}{Q} \\
\frac{\mathrm{d}i_{R_1}(t)}{\mathrm{d}t} &= -\frac{1}{R_1 C_1} i_{R_1}(t) + \frac{1}{R_1 C_1} i(t) \\
\frac{\mathrm{d}i_{R_2}(t)}{\mathrm{d}t} &= -\frac{1}{R_2 C_2} i_{R_2}(t) + \frac{1}{R_2 C_2} i(t) \\
v(t) &= \mathrm{OCV}(z(t)) - R_1 i_{R_1}(t) - R_2 i_{R_2}(t) - R_0 i(t)
\end{aligned} \tag{3.6}$$

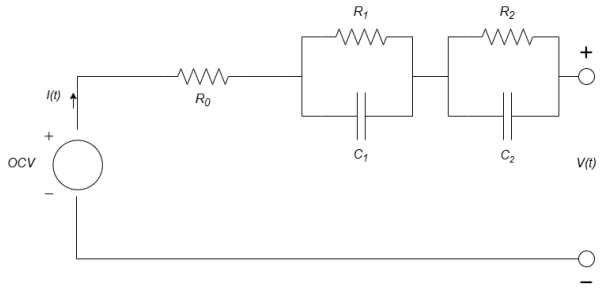


Figure 3.7: A schematic of the second-order Thevenin model.

The circuit constants R_0, R_1, R_2, C_1, and C_2 are identified using an offline parameterization approach for second-order Thevenin models as detailed in a research article [11]. The initial SOC is calculated through a non-linear relationship between SOC and the OCV found in a battery-specific lookup table. The equivalent circuit model employs a Panasonic NCR18650 battery cell. The datasheet for the battery can be seen in the appendix. The battery characteristics are stored in a Python dictionary.

Table 3.2: The battery cell characteristics for the Panasonic NCR18650 [11] [12].

Parameter	Value	Unit
R_0, discharge	0.0019	Ω
R_1, discharge	0.0017	Ω
C_1, discharge	5598.4	F
R_2, discharge	0.0139	Ω
C_2, discharge	352.253	F
R_0, charge	0.0019	Ω
R_1, charge,	0.0017	Ω
C_1, charge	5598.4	F
R_2, charge,	0.0139	Ω
C_2, charge	352.253	F
Δt	1	S
Q_{nominal}	2.7	Ah
Initial SOC	1	-
Max discharge current	9	A
Max charge current	1.925	A
Cell weight	0.0465	kg
Cell volume	1.76E-05	m^3

The input to such an equivalent electrical circuit model is the current flow, while the output of the model is limited to the SOC and the terminal voltage of the battery cell. Equation 3.6 is approximated using the forward Euler method in Python. An example of the output of this model is given in figure 3.8. It shows the terminal voltage and the SOC of the battery cell as a result of the input load current.

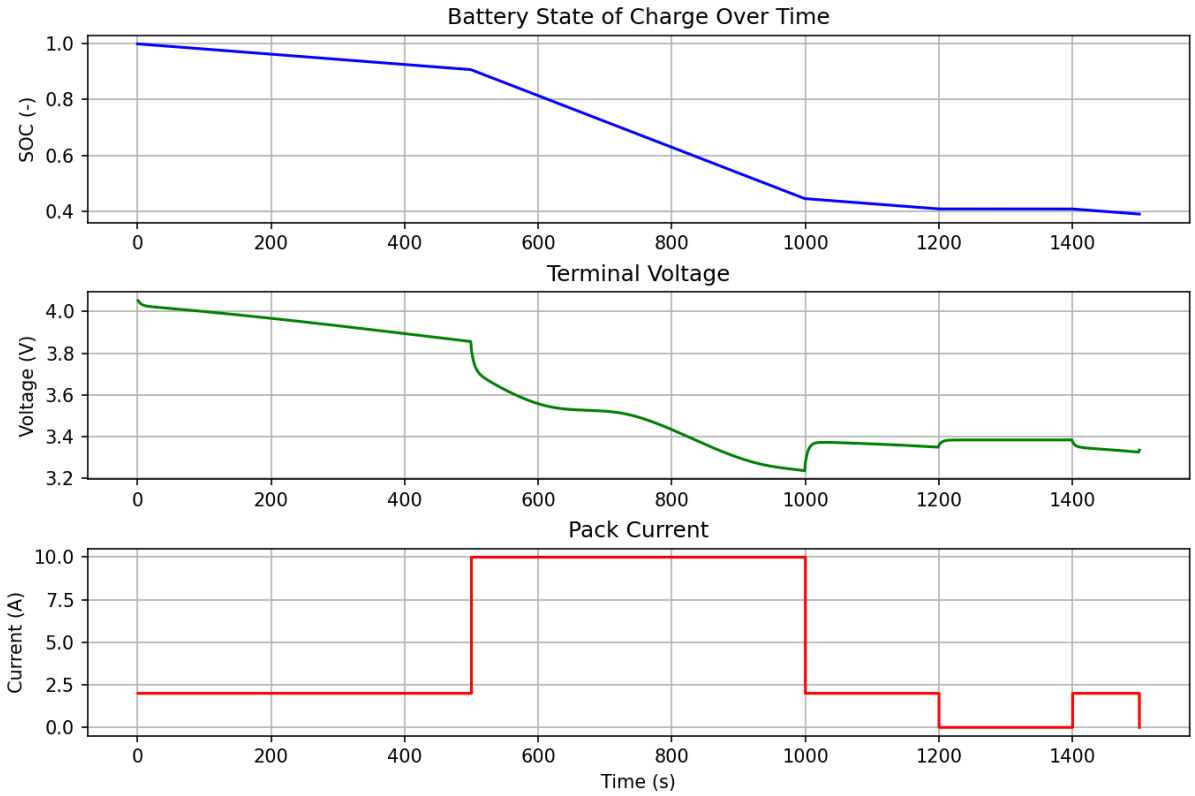


Figure 3.8: The SOC, terminal voltage and current flow over time for the Panasonic NCR18650 equivalent circuit model.

A battery pack consists of multiple individual battery cells, systematically arranged in a grid-like structure through series and parallel connections to achieve the desired voltage, capacity, and power output. Battery cells connected in series increase the total voltage of the pack while the capacity of the cells stays the same. Meanwhile adding more parallel battery cells increases the capacity of the battery pack while the pack voltage stays constant. The initial step in battery sizing is to ascertain the number of cells in series, denoted as n_{series}. This can be calculated using equation 3.7, where V_{pack} represents the voltage of the complete battery pack and V_{cell} is the nominal voltage of an individual cell.

$$\text{V}_{\text{pack}} = n_{\text{series}} \text{V}_{\text{cell}} \tag{3.7}$$

The characteristics of a cell, specifically its capacity Q and maximum power output $\text{P}_{\text{max, cell}}$, are used in the following inequalities to determine the number of cells in parallel, n_{parallel}.

$$P_{\text{max, cell}} = \text{V}_{\text{cell}} I_{\text{max, current}} \tag{3.8}$$

$$E_{\text{demand}} \leq n_{\text{series}} n_{\text{parallel}} \text{V}_{\text{cell}} Q \tag{3.9}$$

$$P_{\text{demand}} \leq n_{\text{series}} n_{\text{parallel}} P_{\text{max, cell}} \tag{3.10}$$

The larger n_{parallel} is selected to ensure the battery pack fulfills the energy and power requirements of the flight mission.

A portion of the power delivered by a battery is inevitably lost due to electrical heating within the system. This energy loss can be quantified through a battery efficiency calculation, which becomes feasible once the battery is properly sized. The power output of a battery cell is calculated using the same approach as outlined in equation 3.8, with the key distinction that the actual current flow is considered instead of the maximum discharge current of the cell.

The electric heating power loss is determined by the internal resistance of the battery pack, R_{pack}, and the actual current flow. The internal resistance of an individual battery cell is modeled using the second-order

Thevenin equivalent, which includes the series resistance R_0 and the polarization resistances R_1 and R_2. The total power loss due to heating, P_{heating}, is expressed as:

$$P_{\text{heating}} = I^2 R_{\text{pack}} \tag{3.11}$$

The total resistance of a single battery cell, R_{cell}, is given by:

$$R_{\text{cell}} = R_0 + R_1 + R_2 \tag{3.12}$$

For a battery pack consisting of n_{series} cells in series and n_{pack} parallel strings, the effective internal resistance of the pack is defined as follows:

$$R_{\text{pack}} = \frac{n_{\text{series}}}{n_{\text{pack}}} R_{\text{cell}} \tag{3.13}$$

The efficiency of the battery can then be quantified by equation 3.14, where the power output of the battery pack P_{output} is divided by the sum of the power output of the pack and the power loss due to electric heating. An example for the battery cell efficiency is also depicted in figure 3.9. This figure shows that the efficiency of the battery changes with the load current of the battery. The oscillating trend in the efficiency line is due to the polarization effects of the battery as result of the pulse current.

$$\eta_{\text{BAT}} = \frac{P_{\text{output}}}{P_{\text{output}} + P_{\text{heating}}} \tag{3.14}$$

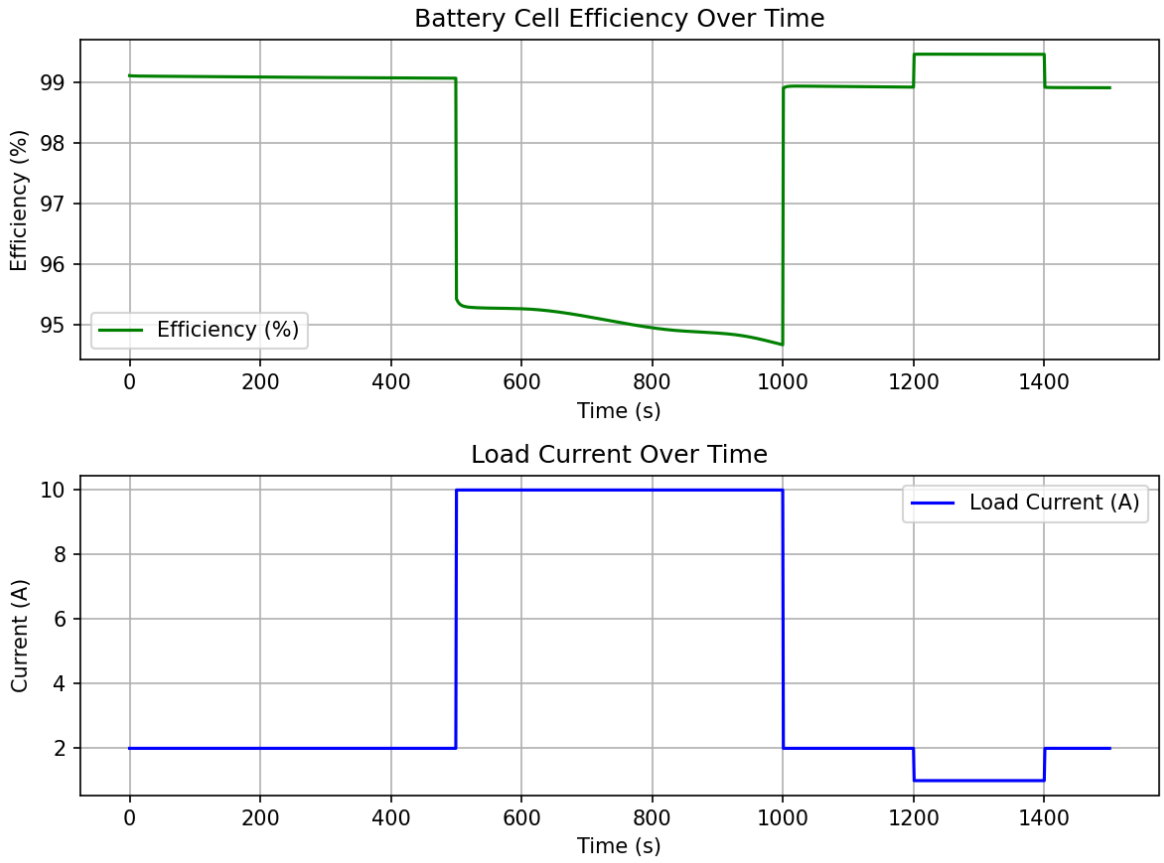


Figure 3.9: The efficiency of the battery cell and the current load over time.

3.3.2. FUEL CELL ANALYTICAL MODEL

The component model for the hydrogen fuel cell is based on the Amphlett Static Model, which was previously used in an eVTOL performance calculation tool with a similar objective as this research [9] [51]. The Amphlett Static Model is a combination of empirical and analytical modeling techniques used to predict the performance of a PEM fuel cell.

This performance is represented by the cell voltage E_{FC}, which is influenced by the cell's thermodynamic potential E_{thermo}, activation losses $\eta_{\text{activation}}$, concentration losses $\eta_{\text{concentration}}$, and ohmic losses η_{ohmic}.

$$E_{\text{FC}} = E_{\text{thermo}} - \eta_{\text{activation}} - \eta_{\text{ohmic}} - \eta_{\text{concentration}} \tag{3.15}$$

$$E_{\text{thermo}} = 1.229 - 0.85 \cdot 10^{-3}(T_{\text{FC}} - 298.15) + 4.3085 \cdot 10^{-5} T_{\text{FC}} \left[\ln p_{h_2} + \frac{1}{2} \ln p_{o_2} \right] \tag{3.16}$$

The activation losses are calculated using an equation that includes parametric constants ξ which are retrieved from experimental data [51]. The activation loss equation can be seen in equation 3.17, followed by the parameters in equation 3.18, where A denotes the active area of the fuel cell and T_{FC} denotes the operating temperature of the fuel cell.

$$\eta_{\text{activation}} = \xi_1 \xi_2 T_{\text{FC}} + \xi_3 T_{\text{FC}} \left[\ln C_{O_2} \right] + \xi_4 T_{\text{FC}} \left[\ln i \right] \tag{3.17}$$

$$\xi_1 = -0.948 \tag{3.18a}$$

$$\xi_2 = 0.00286 + 0.0002 \ln A + \left(4.3 \cdot 10^{-5}\right) \ln C_{H_2} \tag{3.18b}$$

$$\xi_3 = 7.6 \cdot 10^{-5} \tag{3.18c}$$

$$\xi_4 = -1.93 \cdot 10^{-4} \tag{3.18d}$$

$$C_{H_2} = \frac{p_{H_2}}{1.09 \cdot 10^6 \cdot \exp \frac{77}{T_{\text{FC}}}} \tag{3.18e}$$

$$C_{O_2} = \frac{p_{O_2}}{5.08 \cdot 10^6 \cdot \exp \frac{-498}{T_{\text{FC}}}} \tag{3.18f}$$

Ohmic losses depend on two types of resistance: the resistance to proton transfer, R_{proton}, within the solid polymer membrane, and the resistance to electron transfer, $R_{\text{electronic}}$, in the graphite collector plates and electrodes. For the limited operational range of the fuel cell, $R_{\text{electronic}}$ can be considered constant, whereas R_{proton} cannot be taken as constant. Equations 3.19, 3.20 and 3.21 show how the ohmic losses are calculated in the model. Observe that $\frac{i}{A}$ represents j, the current density, and note that the membrane's specific resistivity is denoted by ρ_m, while the thickness of the membrane is denoted by l. Equation 3.21 is applicable solely when the membrane in the fuel cell consists of Nafion.

$$\eta_{\text{ohmic}} = i \left(R_{\text{electronic}} + R_{\text{proton}} \right) \tag{3.19}$$

$$R_{\text{proton}} = \frac{\rho_m \cdot l}{A} \tag{3.20}$$

$$\rho_m = \frac{181.6 \left[1 + 0.03 \left(\frac{i}{A} \right) + 0.062 \left(\frac{T_{\text{FC}}}{303} \right)^2 \left(\frac{i}{A} \right)^{2.5} \right]}{\left[\lambda - 0.634 - 3 \left(\frac{i}{A} \right) \right] \exp \left[4.18 \left(\frac{T_{\text{FC}} - 303}{T_{\text{FC}}} \right) \right]} \tag{3.21}$$

Equation 3.22 shows the calculation for the concentration losses of the fuel cell, while J_{max} denotes the maximum current density of the fuel cell.

$$\eta_{\text{concentration}} = -B \cdot \ln \left(1 - \frac{J}{J_{\text{max}}} \right) \tag{3.22}$$

$$B = \frac{R T_{\text{FC}}}{nF} \tag{3.23}$$

Once all losses are identified, the voltage of the fuel cell can be determined by applying equation 3.15. Similar to a battery, the stack voltage of a hydrogen fuel cell, $V_{\text{FC,stack}}$, depends on both the number of cells connected in series, N, and the individual cell voltage, $V_{\text{FC,cell}}$. The stack voltage equation for a fuel cell is provided in equation 3.24. The power output of the hydrogen fuel cell is obtained by the product of the fuel cells current and the cell voltage.

$$V_{\text{FC,stack}} = N \cdot V_{\text{FC,cell}} \tag{3.24}$$

The fuel cell voltage and current density can then be used to draw the polarization and power density curves of the hydrogen fuel cell.

3.3.3. COMPRESSOR

The fuel cell is supported by balance-of-plant components. These elements ensure that the fuel cell stack operates under optimal conditions for achieving maximum efficiency. The key balance-of-plant components include the compressor, humidifier, and heat exchanger. The compressor's role is to elevate the pressure of the incoming air mass flow to the fuel cell's operating pressure. Equation 3.25 illustrates that the compressor power P_{comp} depends on the incoming air mass flow $\dot{m}_{\text{air}_{\text{in}}}$, as well as the total inlet temperature T_{t_0}, the total outlet temperature T_{t_1}, and the specific heat of the air c_p. The efficiencies for the motor η_m and the compressor η_c are assessed to be 0.9 and 0.8, respectively, as noted by [50]. Employing the pressure ratio alongside the isentropic relation shown in equation 3.26, leads to equation 3.27. Note that these equations are only applicable when assuming isentropic processes. The deviations from the real world are corrected by the η_{comp}.

$$P_{\text{comp}} = \dot{m}_{\text{air}_{\text{in}}} \frac{c_p \left(T_{t_1,is} - T_{t_0}\right)}{\eta_m \eta_{\text{comp}}} \tag{3.25}$$

$$\frac{T_{t_1,is}}{T_{t_0}} = \left(\frac{p_{t_1}}{p_{t_0}}\right)^{\frac{\kappa-1}{\kappa}} \tag{3.26}$$

$$P_{\text{comp}} = \dot{m}_{\text{air}_{\text{in}}} c_p \frac{T_{t_0}}{\eta_m \eta_{\text{comp}}} \left(\frac{p_{t_1}}{p_{t_0}}^{\frac{\kappa-1}{\kappa}} - 1\right) \tag{3.27}$$

The current flow through the fuel cell determines the incoming air mass flow rate through the compressor. This relation can be seen in equation 3.28.

$$\dot{m}_{\text{air}_{\text{in}}} = \frac{\lambda_{\text{air}}\, IM_{O_2}}{4F} = 3.57E-7 \cdot \lambda_{O_2} \frac{P_{\text{total}}}{V_{\text{cell}}} \tag{3.28}$$

3.3.4. HUMIDIFIER

An effective water management strategy is crucial to ensure the proton-exchange membrane (PEM) fuel cell operates under optimal conditions. The primary goal of the humidifier is to introduce sufficient water to the hydrogen fuel cell, thereby maintaining the relative humidity of the outlet flow at 100% [9]. The membrane is humidified through a combination of water generated by the electrochemical reaction within the cell and the water supplied by the humidifier. The output of the submodel specifies how much water will be supplied by the humidifier.

The required water mass flow rate for membrane humidification is obtained from the specific humidity, ω, shown in Equation 3.29. Here, p_W and p_{air} denote the partial pressures of water and air, respectively. With the molar masses being: $M_w = 18$ g/mol and $M_{\text{air}} = 28.97$ g/mol.

$$\omega = \frac{\dot{m}_w}{\dot{m}_{\text{air}}} = \frac{18 \cdot p_{W_{\text{inlet}}}}{28.97 \cdot p_{\text{air}}} = 0.622 \frac{p_{W_{\text{inlet}}}}{p_{\text{air}}}. \tag{3.29}$$

Since the partial pressure of dry air can be approximated by subtracting the partial pressure of water vapour from the total inlet pressure, rearranging Equation 3.29 yields Equation 3.30, which provides the necessary mass flow rate of water at the inlet of the fuel cell:

$$\dot{m}_w = 0.622 \frac{p_{W_{\text{inlet}}}}{p_{\text{inlet}} - p_{W_{\text{inlet}}}} \dot{m}_{\text{air}\ r_{\text{inlet}}} \tag{3.30}$$

The saturation pressure of water vapor at the fuel cell's operating temperature can be determined by regression. Meanwhile, the inlet water partial pressure $p_{w,inlet}$ can be calculated using the constant relative humidity ϕ_{inlet} and equation 3.31 [9].

$$\phi_{\text{inlet/outlet}} = \frac{p_{w_{\text{inlet/outlet}}}}{p_{\text{sat}}} \tag{3.31}$$

Within the fuel cell system, the cathode flow comprises both the incoming air (from the compressor) and the outgoing stream containing the water produced by the electrochemical reaction as well as any excess air. On the anode side, the flow submodel accounts for the incoming hydrogen stream and the exiting hydrogen flow. The consumption and production rates of the reactant gases (in both the anode and cathode flows) are determined using Equation 3.32, where $\dot{m}$ is the mass flow rate of the reactant gas, M is its molar mass, and F is the Faraday constant, representing the charge per mole of electrons. The term P_{stack} denotes the total power output of the fuel cell stack, while V_{cell} refers to the voltage of a single cell:

$$\dot{m} = \frac{M}{2F} \cdot \frac{P_{\text{stack}}}{V_{\text{cell}}}. \tag{3.32}$$

By incorporating the humidifier and accounting for the precise consumption and production rates of the reactants, the methodology ensures that the water content, the air flow, and the hydrogen flow are accurately modeled to maintain the optimum fuel cell performance. This water management approach complements the balance components described above, such as the compressor and guarantees stable operation across various mission segments.

3.3.5. THERMAL MANAGEMENT SYSTEM

The thermal management system (TMS) used in the performance calculation tool made by Khalil [9] is based on a study focusing on the conceptual design of a thermal management system for PEM fuel cells written by Vonhoff [52]. This TMS model estimates the power P_{tms} needed to dissipate the heat $\dot{Q}$ coming from the operation of the PEM fuel cell, while also applying an correction factor f_{dt} [53]. The equation for this can be seen in equation 3.33.

$$P_{\text{tms}} = \left(0.371 \cdot \dot{Q} + 1.33\right) f_{dt} \tag{3.33}$$

The dissipation of heat is a function of the voltage efficiency η_{voltage} of the fuel cell and the nominal power of the fuel cell P_{FC}. Equation 3.34 shows this relationship, while equation 3.35 shows a correlation to specify f_{dt} [53].

$$\dot{Q} = \left(\frac{1}{\eta_{\text{voltage}}} - 1\right) P_{\text{FC}} \tag{3.34}$$

$$f_{dt} = 0.0038\left(\frac{T_0}{(T_{FC} - T_0)}\right)^2 + 0.0352\left(\frac{T_0}{(T_{FC} - T_0)}\right) + 0.1817 \tag{3.35}$$

3.3.6. TURBOGENERATOR

The component model for the turbogenerator differs from the other component models in the performance calculation tool since it uses the Gas Turbine Simulation Program (GSP), a third party software specialized in the modeling, simulation and analysis for gas turbine systems. The built for this component model is based on the performance calculation tool developed in [9]. The component model also differs from the other components by the fact that the performance of the turbogenerator is precomputed in GSP, the results of this computation are extracted and interpolated by the performance calculation tool to fit into the mission profile.

3.4. SYSTEM MODEL IMPLEMENTATION

This chapter shows the integration of all submodels to create three complete hybrid powertrain simulations. The FC-BAT combines a PEM fuel cell with a Li-ion battery pack, BAT-Only deploys a battery pack as sole power sources, while the FC-GT uses the same fuel cell alongside a turbogenerator. In all cases, the models are driven by flight mission data—altitude, velocity, and power demands—to compute fuel consumption, thermal loads, power flows, and overall efficiency. By comparing these two implementations, the chapter

highlights the key trade-offs between battery-based, fuel cell-based and turbogenerator-based hybrid propulsion for advanced VTOL aircraft.

3.4.1. HYDROGEN FUEL CELL AND BATTERY POWERTRAIN ARCHITECTURE

A hybrid propulsion architecture is developed for the FC-BAT configuration, integrating a Li-ion battery pack with a PEM fuel cell to satisfy the mission requirements of an advanced VTOL aircraft. The battery sizing is driven by peak power, total energy, and target voltage constraints (see Section 3.3.1). First, a nominal cell voltage is derived from a Thevenin-based simulation at representative discharge currents. Then, the required number of series cells is selected to achieve the target pack voltage, and parallel strings are added to ensure sufficient energy capacity and to limit each cell's discharge current.

Meanwhile, the fuel cell system relies on a static Amphlett model, which maps cell voltage to current density while accounting for activation, ohmic, and concentration losses. The stack is configured to meet the specified voltage and power levels, and an optimization routine minimizes the overall fuel cell mass while ensuring that the net power output meets flight demands after accounting for compressor and cooling loads. This entire process is orchestrated by an optimizer, as illustrated in the XDSM diagram of Figure 3.10.

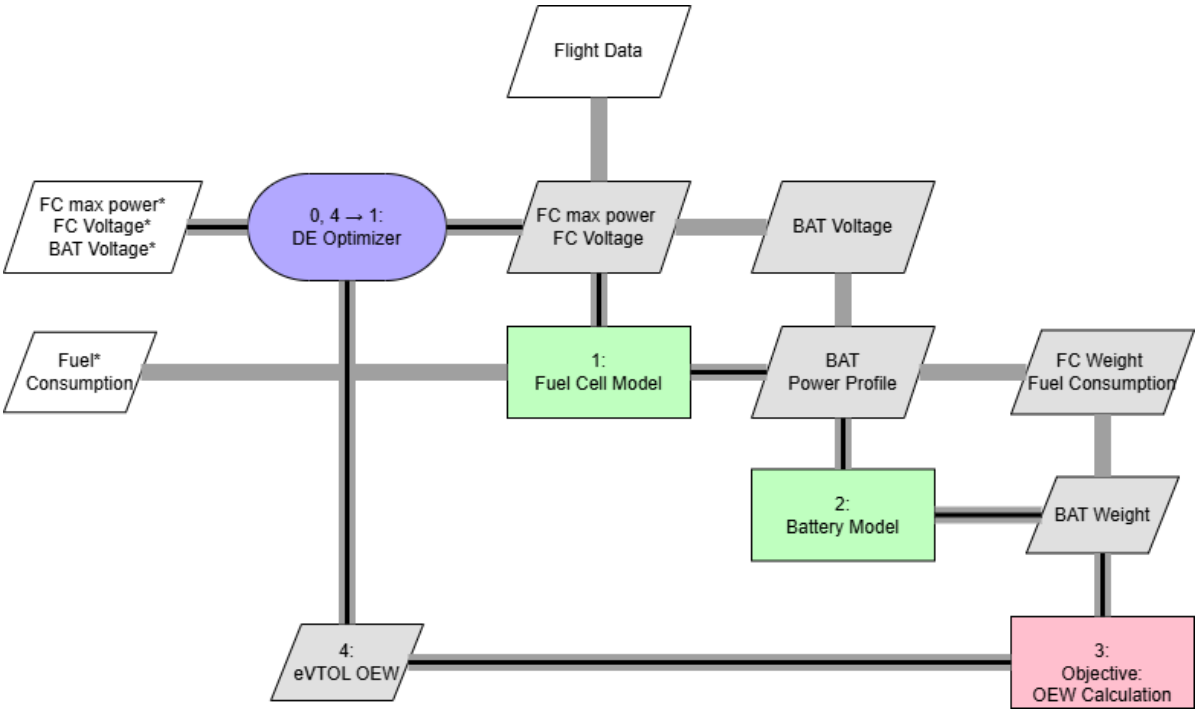


Figure 3.10: XDSM chart of the optimization scheme of the FC-BAT design.

The diagram shows how the optimizer coordinates the design variables: Fuel cell voltage, battery voltage, and the fuel cell power limit, and passes them to both the fuel cell model and the battery model. The fuel cell model also receives flight data (Table 3.1), including the power profile, altitude, and flight speed. Based on these inputs, the fuel cell is sized, and its net power output is calculated under off-design conditions after subtracting the cooling and compressor loads. The resulting fuel cell system weight and hydrogen consumption are then passed to the objective block. Meanwhile, any power gap between the flight demand and the net fuel cell output is passed to the battery model as an updated power profile. The battery model also takes in the battery voltage from the optimizer, using these inputs to compute battery mass and performance, which are then forwarded to the objective block. The objective block calculates the OEW of the eVTOL and returns this value to the optimizer, which iterates until convergence. This XDSM representation underscores the sequential yet loosely coupled nature of the fuel cell and battery models, while emphasizing that the entire scheme is driven at the top level by the optimizer.

Off-design performance is assessed for various flight phases by adjusting ambient conditions and compressor pressure ratios, and the resulting hydrogen consumption is integrated over the entire mission to determine total fuel requirements. In the combined simulation, both subsystems (fuel cell and battery) are updated step by step with flight data (including power demand, altitude, and Mach number). This approach

allows the model to track the battery state of charge and the net fuel cell output while accounting for overall system efficiency, thermal loads, and power flows. Consequently, factors such as eVTOL hydrogen usage, state of charge, component sizing, and parasitic power consumption are captured within a unified system model.

OPTIMIZATION: DIFFERENTIAL EVOLUTION

The final dimensions, weight, and fuel consumption of the FC-BAT architecture are determined by the execution of the previously described calculation scheme. To enhance the performance of this scheme, a heuristic optimization technique is employed to maximize the payload capacity of the FC-BAT system by minimizing the aircraft's Operational Empty Weight (OEW) and fuel consumption. This optimization is performed using Differential Evolution (DE), a population-based stochastic optimization method designed to minimize complex and non-differentiable cost functions over a set of parameters [54].

In this optimization problem, three design variables are considered: battery voltage, fuel cell voltage, and the maximum power limit of the fuel cell. Any arbitrary combination of these three variables defines an unique hybrid powertrain architecture design, or a so-called candidate solution $x_{r1,G}$.

$$x_{r1,G} = \begin{bmatrix} \text{Battery voltage} \\ \text{Fuel cell voltage} \\ \text{Fuel cell max power} \end{bmatrix}$$

The optimizer initializes a *population* of candidate solutions, each containing a unique combination of these design variable values. The population evolves iteratively through mutation, crossover, and selection, ensuring that the best-performing solutions are retained in successive generations.

The mutation step generates a new candidate solution, which is called the mutant vector $v_{i,G+1}$, by adding a weighted difference between two randomly selected vectors to a third vector. Diversity is introduced into the population of candidate solutions by the mutation process and it helps the optimizer escape local minima. Mathematically, the mutation process is defined as follows:

$$v_{i,G+1} = x_{r1,G} + F \cdot \left(x_{r2,G} - x_{r3,G}\right) \tag{3.36}$$

where $x_{r1,G}$, $x_{r2,G}$, and $x_{r3,G}$ are randomly selected vectors from the population, and F is a scaling factor that controls the magnitude of the mutation. Each vector represents a complete hybrid powertrain architecture.

Following mutation, the *crossover* step creates a new trial vector by combining elements of the mutant vector $v_{i,G+1}$ with those of another vector from the population, the target vector. This process is governed by the crossover rate (CR), which determines the proportion of information inherited from the mutant vector. The resulting trial vector is then evaluated and compared with the original target vector. The selection step ensures that only the vector with the better objective function value advances to the next generation, thereby steering the population towards improved designs [54].

To ensure effective optimization, the population size should be at least five to ten times the number of design variables. The scaling factor (F) and crossover probability (CR) are typically chosen within the range of 0 to 1, balancing exploration and exploitation throughout the optimization process [54].

3.4.2. BATTERY ONLY POWERTRAIN ARCHITECTURE

The BAT Only architecture can be viewed as a reduced form of the FC-BAT configuration, with the primary difference being that the total power demand must be met solely by the Li-ion battery pack. Instead of splitting power requirements between a fuel cell and a battery, this architecture relies entirely on the battery to satisfy all flight phases in the mission profile.

In the simulation, the battery model (detailed in Section 3.3.1) is configured to receive the complete mission power profile, which includes both transient segments (e.g., takeoff, climb) and relatively steady phases (e.g., cruise). The model updates the battery's state of charge (SOC) at each timestep based on the commanded current, which is derived from the power requirements. Consequently, battery voltage and SOC trajectories are computed over the duration of the mission:

1. **Sizing Step:** The maximum discharge current and total energy draw across the mission determine the number of cells in series and parallel. A suitable nominal cell voltage is first chosen (for instance, from

the Thevenin based parameterization), and the total pack voltage is set by the number of series connections. Parallel connections are then added to ensure the pack can both supply the highest required power level and store sufficient energy for the full mission.

2. **State Updates:** At each timestep, the model calculates the instantaneous current from the known power demand and battery pack voltage. The resulting current updates the SOC via the Coulomb counting approach. The open circuit voltage for each timestep is interpolated from look up tables of SOC vs. OCV, and the model applies internal resistances (and polarization elements, if present) to determine the terminal voltage.

3. **Objective:** For studies targeting optimal mass or minimal operational empty weight, the total battery mass is computed by multiplying the individual cell mass by the total number of cells. Because there is no second power source, the optimization routine in the BAT Only case essentially reduces to finding a combination of series and parallel connections that minimizes pack weight while still covering the entire mission envelope.

As a result, the system model for the BAT Only case is simpler than for hybrid architectures. It does not need to iterate with a fuel cell model, nor does it account for parasitic losses such as compressor loads. The battery submodel continuously checks whether it can handle the peak power demands and total energy draw of the mission, while the simulation tracks real time voltage, SOC, and waste heat generation. This approach captures battery performance throughout the flight envelope, allowing for direct comparisons to more complex systems like FC-BAT and FC-GT without the added degrees of freedom from fuel cell interactions or turbogenerator constraints.

3.4.3. HYDROGEN FUEL CELL AND TURBOGENERATOR POWERTRAIN ARCHITECTURE

In the FC-GT architecture, a PEM fuel cell is once more simulated and dimensioned using both on-design and off-design methods to meet part of the aircraft's power needs. To satisfy any additional demand, a turbogenerator is incorporated. During each simulation timestep, the script initially calculates the net output of the fuel cell based on flight conditions such as altitude and Mach number and then determines if extra power is necessary. If required, it accesses a performance data lookup for the turbogenerator, derived from external GSP outputs (including metrics like shaft power relative to turbine inlet temperature, fuel consumption, and air properties), to assess the power the turbine can generate within the given altitude and temperature constraints. The script monitors hydrogen (or alternative fuel) usage, temperatures (such as compressor outlet and turbine inlet), and power flows (both gross and net) to generate a detailed operational timeline of the turbogenerator throughout the mission. By integrating the off-design capabilities of the fuel cell with the adaptable power output of the turbogenerator, the model guarantees that the total system output aligns with the propulsion needs dictated by flight data at all times, facilitating a thorough evaluation of fuel usage, system efficiency, and mass-flow balance within this novel hybrid setup.

4

Results and Discussion

This chapter presents and critically evaluates the simulation results generated with the performance-calculation tool. It begins by validating the battery model against reference discharge data, thereby establishing confidence in the accuracy of subsequent analyses. The discussion then compares the hybrid power-train configurations formulated in Chapter 3, focusing on their respective energy demands and mass penalties. Section 4.3 follows with a series of sensitivity studies that illuminate the trade-offs associated with component voltage selection and power-split strategies between the energy sources. The chapter concludes with forward-looking simulations that assess the influence of expected technological advances, such as higher battery specific energy and increased fuel-cell power density, on overall aircraft performance.

4.1. Battery Model Verification

4.1.1. Simulation Setup and Experimental Reference

In order to validate the battery model incorporated into the performance calculation tool, a constant-current discharge simulation was performed and compared against experimental data. The experimental test, as performed in [11], involved a 1C continuous discharge of a Panasonic NCR18650 battery cell using a BTS4000 battery testing system, which precisely imposed the current load while recording the corresponding voltage response. The C-rate denotes how quickly a battery can be charged or discharged relative to its nominal capacity, so a 1C rate corresponds to using the full capacity in one hour, 2C in half an hour, and so on [55].

To replicate these conditions in the simulation, the model was configured for an identical 1C discharge rate, starting from the same initial state of charge. This setup ensured that any differences between the simulated and measured voltage responses could be attributed to the battery model itself rather than discrepancies in operational conditions.

Figure 4.1 illustrates the voltage responses from both the experimental test and the simulation. The simulated voltage profile captures the overall discharge trend but exhibits a slightly oscillatory behavior that is not seen in the experimental data. While this oscillatory characteristic warrants further investigation, the simulation results provide a preliminary indication that the model's core dynamics resemble those of the physical battery.

To quantify the battery model's performance, two error metrics were employed: the root mean square error (RMSE) and the mean absolute percentage error (MAPE). The RMSE, as utilized in [11], offers an absolute measure of deviation from experimental voltages, while MAPE provides a relative measure, representing the error as a percentage of the measured voltage.

Figure 4.2 shows the difference between the experimental and simulated voltages over the course of the test. Table 4.1 summarizes the calculated RMSE and MAPE values. The simulation yielded an RMSE of 0.0506 V and a MAPE of 1.00 %. Considering the battery's operating voltage range of approximately 3.2 V to 4.0 V, this represents a relatively small fraction of the overall discharge voltage.

In [11], an RMSE of 0.0189 V was reported for a similar battery model, corresponding to 0.45 % in percentage terms—indicating that their approach outperforms the current battery model by a small margin. Still, an RMSE of 0.0506 V and an average relative error of 1 % suggest that the model adequately captures most of the critical battery discharge behavior for system-level aerospace applications. The relevance of this battery error metric is elaborated further on in section 4.3.1,

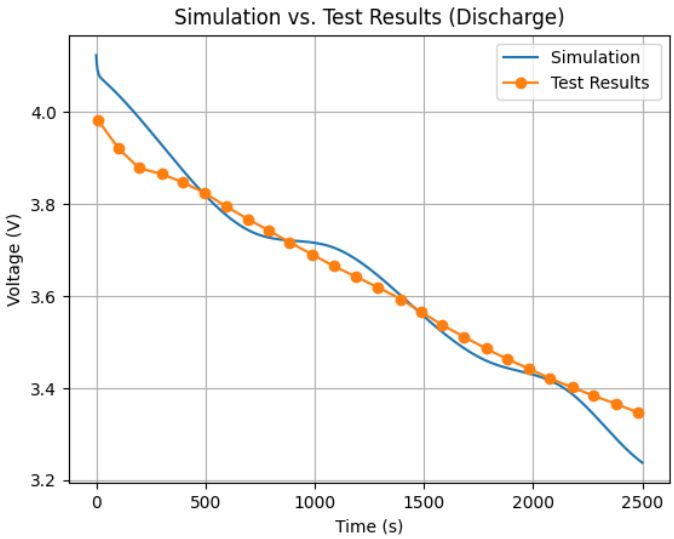


Figure 4.1: Comparison of the voltage responses from the battery test and the battery simulation under a continuous 1C discharge current.

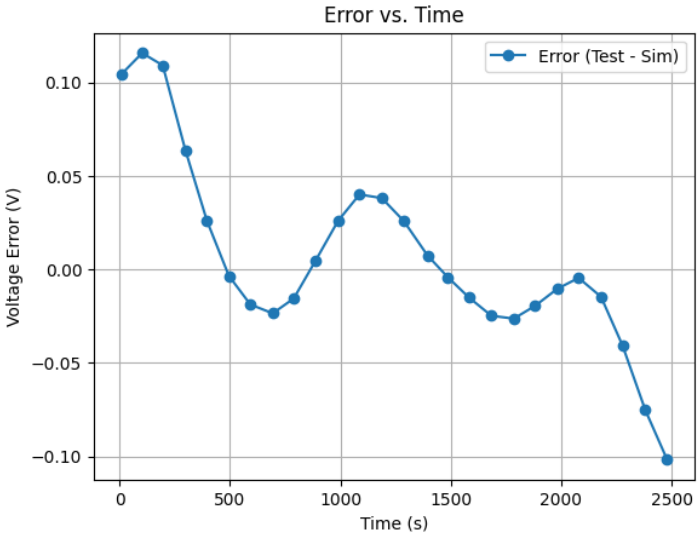


Figure 4.2: The voltage error of the continuous 1C discharge current simulation.

Table 4.1: Error metrics of the constant-current discharge simulation.

RMSE (V)	MAPE (%)
0.0506	1.00

4.1.2. OCV–SOC RELATIONSHIP IMPACT

Despite acceptable error metrics, the oscillatory behavior in the simulated discharge curve suggests that further refinement of the model's open circuit voltage vs. state of charge relationship may be necessary. Any discrepancies in the OCV–SOC definition propagate through the entire discharge cycle, influencing both steady-state and transient responses. To explore this further, an alternative OCV–SOC relationship from [56] was implemented, as shown in Figure 4.3. While the transient behavior was again reasonably captured, the overall voltage accuracy changed noticeably. Table 4.2 shows that the RMSE increased to 0.0900 V and the MAPE to 2.09 %, demonstrating a trade-off between matching transient dynamics and fitting the steady-state voltage profile.

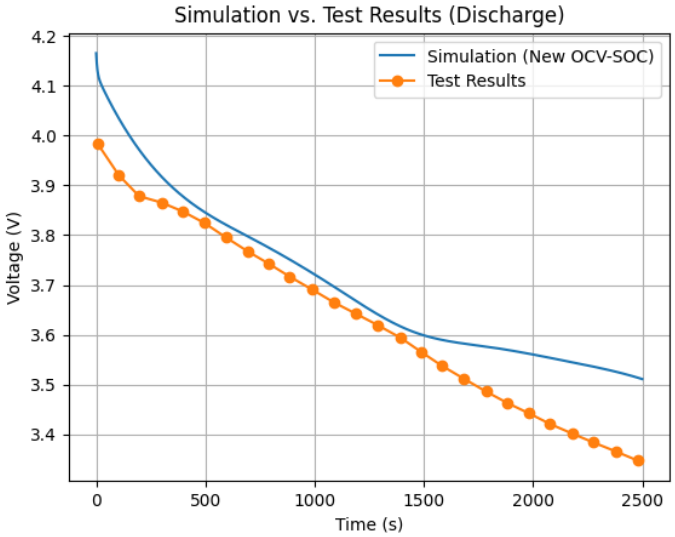


Figure 4.3: The discharge current test simulation with a new OCV–SOC relationship.

Table 4.2: The error metrics for the alternative OCV–SOC relationship.

RMSE (V)	MAPE (%)
0.0900	2.09

In conclusion, these tests highlight that although the model's transient discharge predictions are acceptable for many aerospace performance evaluations, a more accurate baseline OCV–SOC relationship could further reduce the residual errors. Nonetheless, given that an RMSE near 0.05 V and an average of 1 % MAPE are already within an acceptable range for feasibility assessments, the battery model can be considered valid for continued use in the performance calculation tool.

4.1.3. KEY TAKEAWAYS:

- The battery model replicates the overall discharge trend with small absolute and percentage errors.
- Minor oscillatory behavior indicates areas for improvement in the OCV–SOC relationship.
- The combination of RMSE and MAPE provides a robust validation approach, highlighting both absolute and relative error magnitudes.
- Further refinement of OCV–SOC data can enhance accuracy, although the current model is sufficiently reliable for system-level aerospace simulations.

4.2. COMPARISON OF HYBRID POWERTRAIN ARCHITECTURES

Two hybrid powertrain architectures is optimized using a Differential Evolution (DE) algorithm, with the objective of minimizing the Operational Empty Weight (OEW). A third powertrain architecture (FC-GT) was

modeled without the use of the optimizer. In these analyses, the aircraft's Maximum Take-Off Weight (MTOW) was held constant at 3175 kg, while the structural weight remained fixed at 1905 kg. Despite the sizing outcomes for each architecture aiming to achieve the lowest possible OEW, neither design satisfied the MTOW constraint, rendering them infeasible for actual flight operations. Nevertheless, the results offer valuable insight into the performance trends and trade-offs between all-battery and hybrid powertrain configurations.

4.2.1. BAT-ONLY ARCHITECTURE

The first architecture considered is an all-battery (BAT-only) design. The optimization centered on determining the number of cells in series and parallel to minimize total battery mass and thereby reduce OEW. Table 4.3 summarizes the sizing results for this configuration.

Table 4.3: Battery sizing optimization results.

Sizing Results: BAT-Only Architecture	
Battery Voltage (V)	1241.5
Number of cells (series)	313
Number of cells (parallel)	129
Number of cells (total)	40377
Battery Weight (kg)	2503.37
OEW (kg)	4714.68

Since current off-the-shelf battery systems typically have lower specific energy than conventional kerosene-based propulsion, the total battery mass in this architecture becomes disproportionately large. As shown in Figure 4.4, the battery pack and the structural weight dominate the overall aircraft mass, resulting in an OEW that exceeds the 3175 kg MTOW limit. By contrast, the mass of the PE converters remains relatively small, given their lower specific mass requirement.

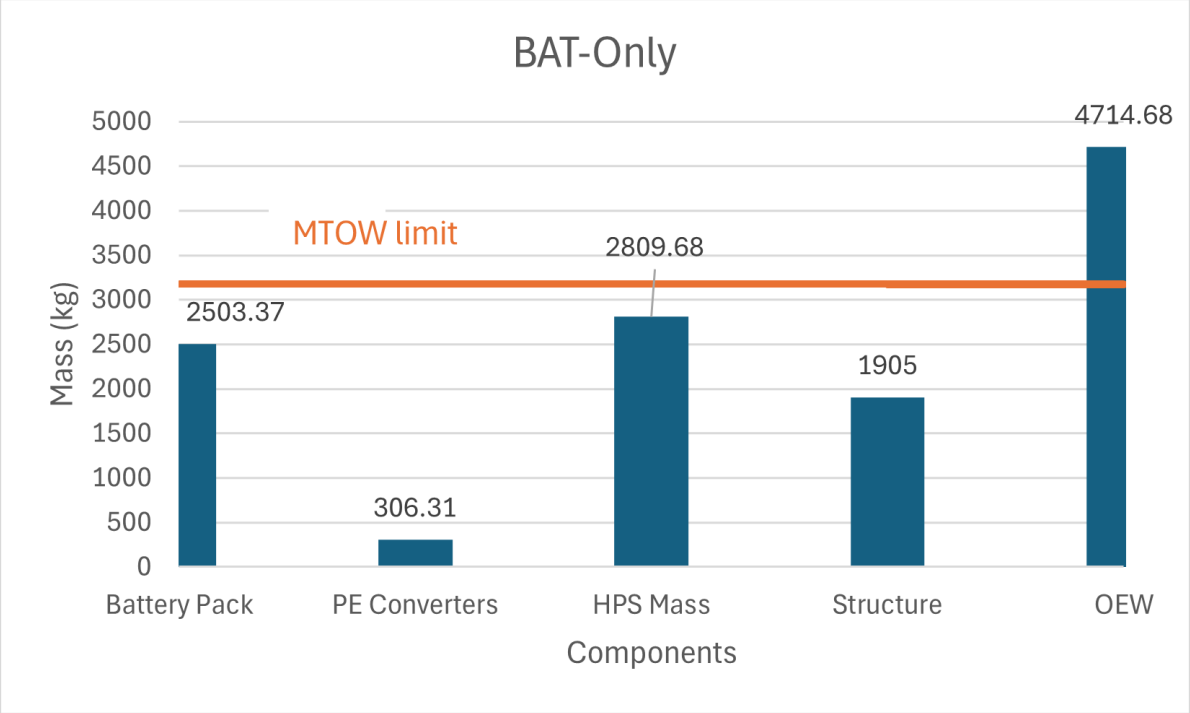


Figure 4.4: Weight component breakdown of the BAT-only architecture.

Time-history plots of the optimized BAT-only architecture are presented in Figure 4.5, illustrating the battery's SOC, terminal voltage, efficiency, and current flow during the entire mission profile. These plots reveal significant discharge levels, reflecting the challenge of relying on a low energy density source to meet mission

requirements. They also indicate a high discharge efficiency when the current draw is relatively low, since efficiency generally decreases at higher C-rates. As the mission progresses and the battery depletes, the terminal voltage drops, necessitating larger currents to supply the required power. This trade-off between voltage and current can be observed in both the battery current and discharge efficiency plots. The terminal voltage plot also shows the oscillatory pattern that was inherited from the SOC-OCV relationship of the battery model.

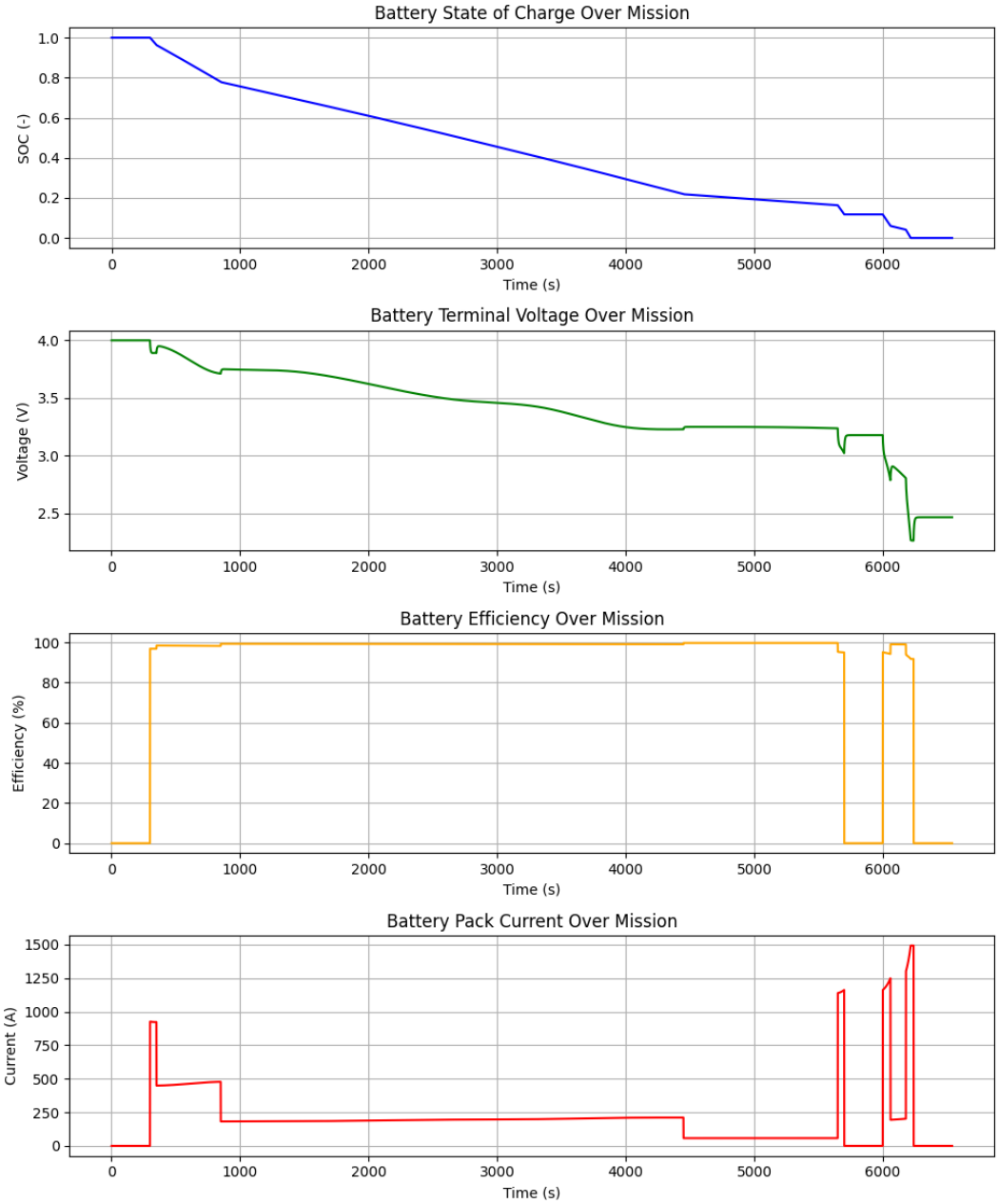


Figure 4.5: Time-history of the battery's SOC, terminal voltage, efficiency, and current for the BAT-only powertrain.

Overall, while the BAT-only architecture is conceptually simpler, its mass penalty underscores the limitations of current battery technology in meeting aerospace requirements within the specified MTOW constraints.

4.2.2. FC-BAT ARCHITECTURE

Table 4.4 summarizes the optimized sizing results for the hybrid fuel cell–battery (FC-BAT) architecture. Compared to the BAT-only design, the addition of a second power source reduces the overall OEW because

the power demand during the mission is partially supported by the fuel cell.

Table 4.4: Sizing results of the optimized FC-BAT design.

Sizing Results: FC-BAT Architecture	
Battery Voltage (V)	1207.4
Fuel Cell Voltage (V)	2204.8
Fuel Cell Power Limit (kW)	39.322
FC Number of Cells (series)	3564.0
FC Active Area (cm^2)	123.5
FC Current Density ($A\,cm^{-2}$)	0.22
FC Stack Mass (kg)	84.2
FC System Mass (kg)	260.3
FC Power Density ($W\,kg^{-1}$)	157.3
BAT Number of Cells (Series)	304
BAT Number of Cells (Parallel)	103
BAT Number of Cells (Total)	31312
Battery Mass (kg)	1941.3
Aircraft OEW (kg)	4405.0

In this hybrid configuration, the fuel cell's nominal power output is set at 39.3 kW, serving as the baseline power source throughout most phases of flight. Any power demand that exceeds 39.3 kW is met by the battery. This power-sharing strategy is illustrated in Figure 4.6, demonstrating how the hybridization helps reduce overall battery mass compared to an all-electric setup. Figure 4.7 highlights the weight distribution, showing that the battery mass decreases from its value in the BAT-only design, while the fuel cell system adds a new, but comparatively manageable, mass component.

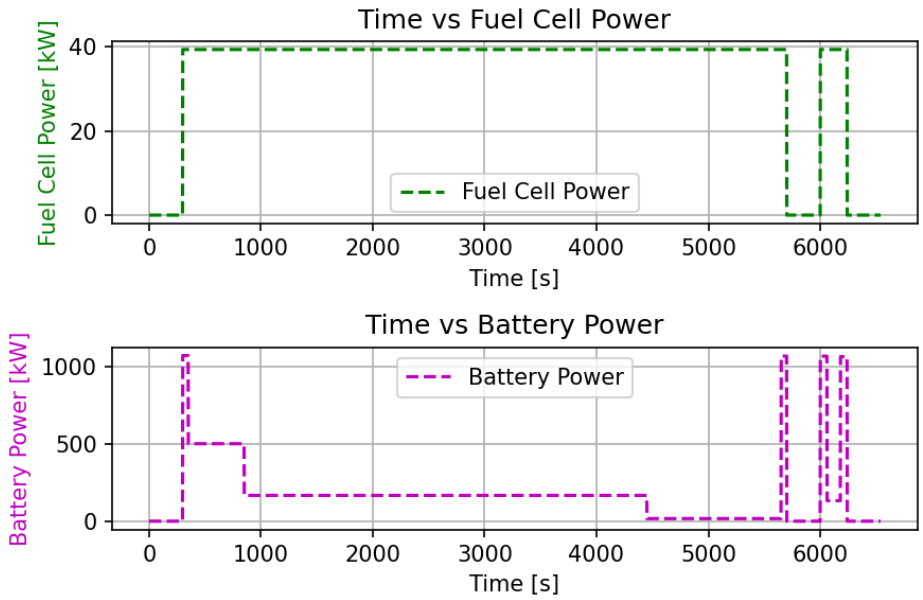


Figure 4.6: Optimized power management strategy in the FC-BAT architecture.

Figure 4.8 presents the time-history of battery SOC, voltage, efficiency, and current flow. The results confirm that sizing the fuel cell for moderate power and relying on the battery for peak loads can help avoid excessively large battery mass. Nevertheless, the OEW still overshoots the 3175 kg MTOW.

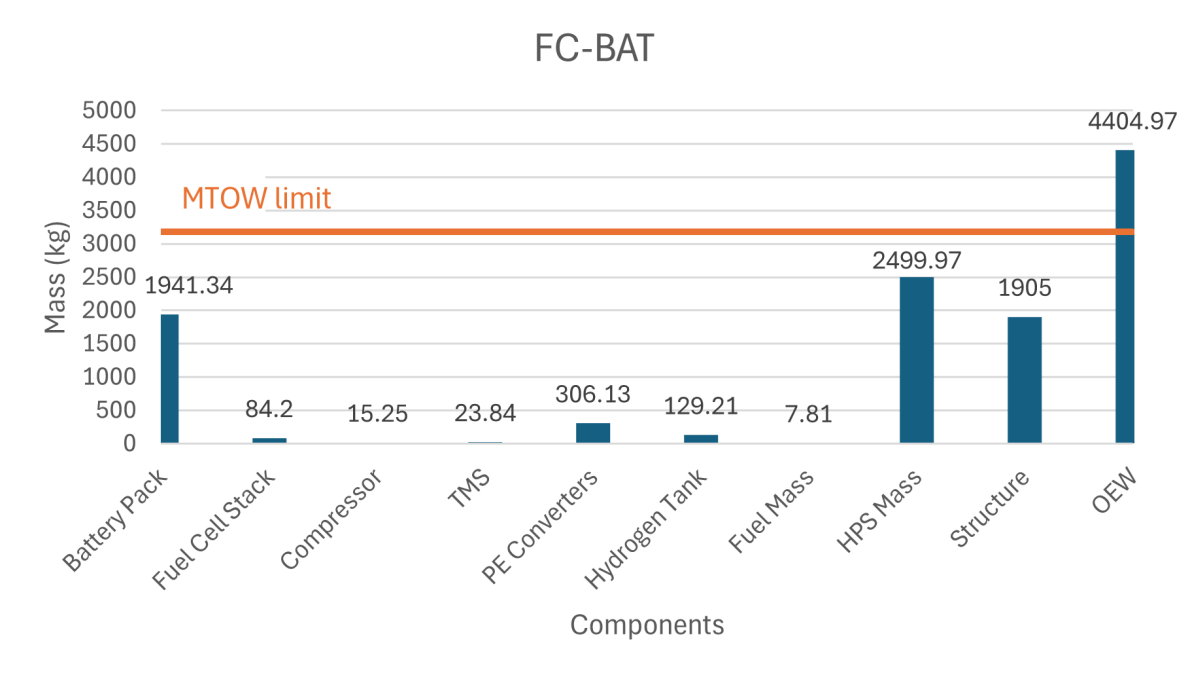


Figure 4.7: Weight component breakdown of the FC-BAT architecture.

4.2.3. FC-GT ARCHITECTURE

A third architecture investigated in this work, here referred to as FC-GT, was derived from [9]. The powertrain couples a fuel cell with a turbogenerator for a hybrid propulsion system. However, integrating the Gas Turbine Simulation Program data into the Python-based optimization code poses additional complexities, limiting the application of the Differential Evolution method in a direct manner.

Despite these challenges, a preliminary sizing result for the FC-GT architecture is provided in Table 4.5, where the turbogenerator and hydrogen fuel tank share the same fuel supply. Notably, for ease of comparison with the FC-BAT architecture, the computed power density of the fuel cell subsystem includes the mass of the hydrogen tank. Although adding the turbogenerator subsystem increases overall complexity, the resulting OEW of 3991.7 kg is slightly lower than that of the BAT-only or FC-BAT designs, pointing to potential benefits of incorporating a turbogenerator.

Table 4.5: Sizing results of the FC-GT architecture (including fuel tank mass in fuel cell power density).

Sizing Results: FC-GT Architecture	
Fuel Cell Voltage (V)	800
Fuel Cell Power (kW)	206.6
FC Number of Cells (series)	1538
FC Active Area (cm^2)	1337
FC Current Density (Acm^{-2})	0.32
FC Stack Mass (kg)	423.6
FC System Mass (kg)	1544.2
FC Power Density (Wkg^{-1})	138.6
Turbogenerator Mass (kg)	212.9
Aircraft OEW (kg)	3991.7

Figure 4.9 presents the weight distribution for each hybrid architecture configuration, allowing for direct comparison between the three powertrain designs.

In conclusion, while each architecture exceeds the MTOW constraint in its current form, the sizing and performance trends underscore the trade-offs between different power source combinations. The BAT-only design suffers from high battery mass, whereas the FC-BAT configuration benefits from a more distributed

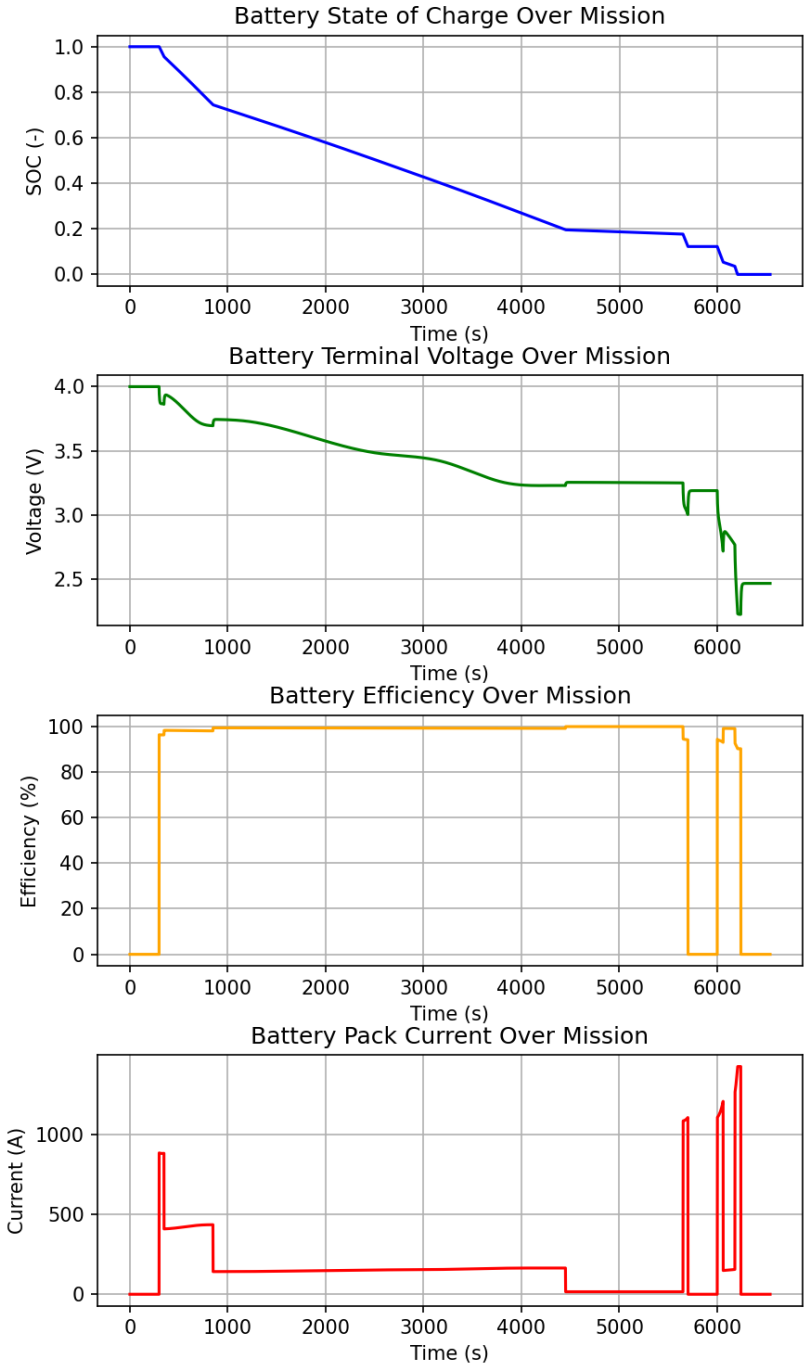


Figure 4.8: Battery performance plots over the mission profile of the FC-BAT architecture.

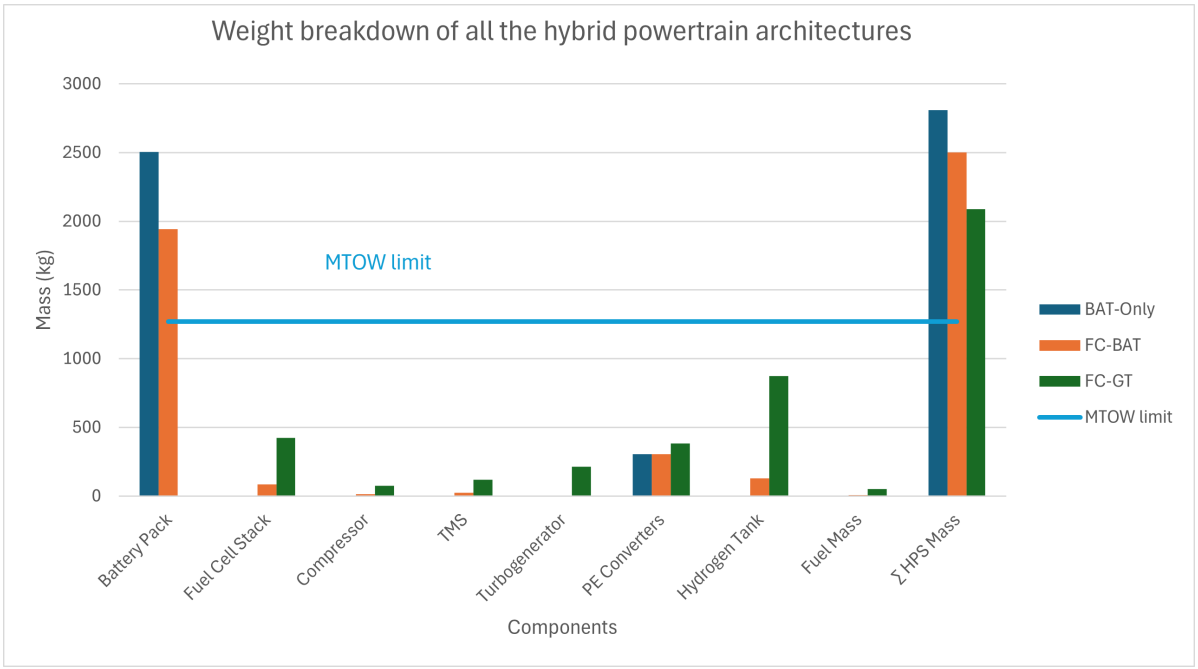


Figure 4.9: The weight breakdown plot of all the hybrid architecture types.

power supply but remains overweight. The FC-GT setup shows promise, although the complexities of optimizing this architecture with the performance calculation tool warrants further research.

4.2.4. KEY TAKEAWAYS:

- Pure battery systems (BAT-only) face prohibitive mass penalties given current battery technologies.
- Adding a fuel cell (FC-BAT) helps reduce battery mass but can still exceed MTOW.
- Coupling a fuel cell with a turbogenerator (FC-GT) shows lower OEW compared to BAT-only or FC-BAT, but introduces additional optimization challenges.
- Although none of the architectures meet MTOW constraints at present, these optimization results establish performance baselines and highlight future paths for technology improvements.

4.3. SENSITIVITY ANALYSES

In addition to architecture-level optimization, a series of sensitivity analyses were performed to explore how key design variables influence overall powertrain weight and performance. By systematically varying parameters such as battery voltage, fuel cell power limit, and fuel cell voltage, these analyses provide deeper insight into the trade-offs that arise when sizing electric and hybrid-electric propulsion systems.

4.3.1. BATTERY VOLTAGE VARIATION IN THE BAT-ONLY ARCHITECTURE

For the BAT-only architecture, the primary design variable is the battery voltage, which in turn fixes the number of cells in series (Section 3.3.1). The remaining degree of freedom is the number of parallel cell strings needed to meet both power and energy demands. Figure 4.10 illustrates how varying the battery voltage affects the aircraft's weight. This pattern arises from the fact that cells are discrete, so whenever the target voltage crosses a cell-voltage threshold the model adds or removes an entire cell in each series branch. To keep the same overall energy capacity, the number of parallel branches then also steps up or down in whole-string increments. Each time series or parallel counts change, the battery pack mass jumps by the weight of one or more battery cells.

Although the curve in Figure 4.10 exhibits an oscillatory pattern, the absolute magnitude of this variation is relatively small. Over a 2000 V range in nominal battery voltage, the maximum weight difference is approximately 15 kg. Consequently, while battery voltage is a relevant parameter for electrical design, its impact on overall aircraft weight remains modest in this particular mission profile.

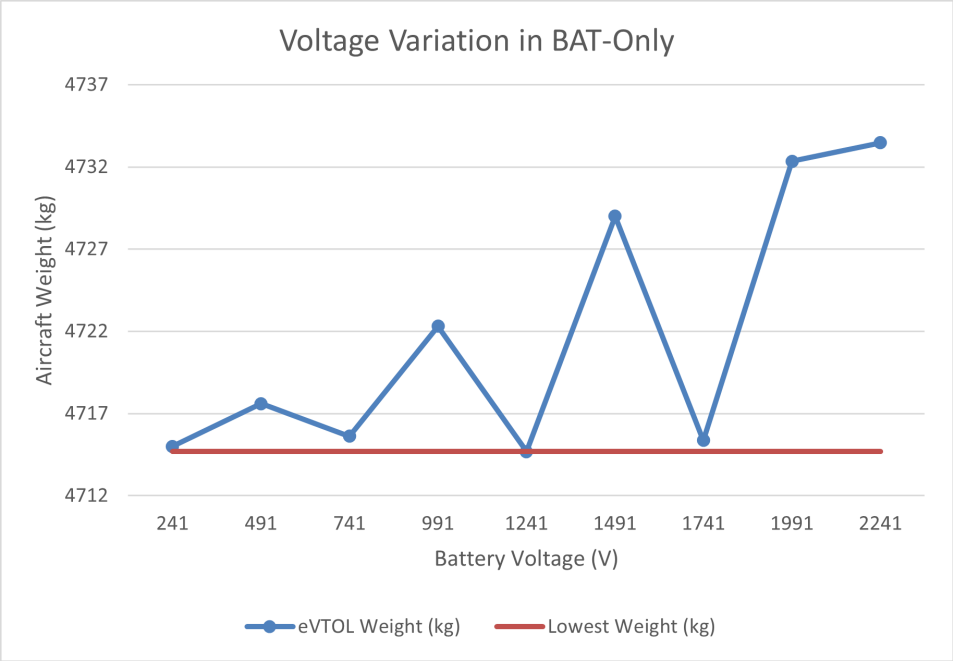


Figure 4.10: Effect of battery voltage on aircraft weight in the BAT-only architecture.

To assess whether the cell-level voltage fit is adequate for system-level sizing, consider a worst-case scenario in which every cell exhibits the root-mean-square voltage error $\delta V_{\text{cell}} = 0.050$ V. With $n_{\text{series}} = 313$ cells connected in series, the resulting pack-voltage bias is

$$\Delta V_{\text{pack}} = n_{\text{series}}\, \delta V_{\text{cell}} = 313 \times 0.050\ \text{V} = 15.65 V.$$

Figure 4.10 shows that the optimised aircraft operating empty weight (OEW) varies by approximately 17.5 kg for a 2000 V change in pack voltage, i.e. 0.00875 kgV^{-1}. Propagating the worst-case voltage bias therefore shifts the OEW by at most

$$\Delta OEW \approx 15.65 V \times 0.00875 kg V^{-1} \approx 0.14 kg$$

which is only about 0.004% of the maximum take-off weight (3175 kg). Hence the present battery-model accuracy is demonstrably sufficient for system-level trade-studies.

4.3.2. FUEL CELL POWER LIMIT IN THE FC-BAT ARCHITECTURE

In the FC-BAT architecture, the addition of a fuel cell introduces several new variables, the most influential of which is the fuel cell power limit. This parameter dictates how much of the required power is supplied by the fuel cell versus the battery. Figure 4.11 shows how varying the fuel cell power limit affects total aircraft weight.

A roughly linear trend emerges, indicating that as the fuel cell power limit increases, overall aircraft weight also rises. However, the minimum point in the graph indicates that the FC-BAT architecture is still lower than the BAT-Only architecture that corresponds with a FC power limit of 0 kW. This linear behavior can be attributed to the relatively low specific power of the fuel cell system; to meet higher power demands with the fuel cell alone, the stack and associated hardware mass must increase. Figure 4.11 further illustrates the effect of varying the fuel cell power limit on total energy consumption. It shows that the total energy consumption of the BAT-Only is the lowest among the remaining FC-BAT configurations. Sizing the fuel cell for high-power segments such as take-off and climb may reduce the battery's mass requirements, but it leads to higher energy use, as the low specific power of the fuel cell reduces overall efficiency in those segments.

4.3.3. VOLTAGE SENSITIVITY IN THE FC-BAT ARCHITECTURE

In the FC-BAT configuration, both the battery pack voltage and the fuel cell stack voltage influence the aircraft's operating empty weight through their impact on component sizing. Rather than examining each volt-

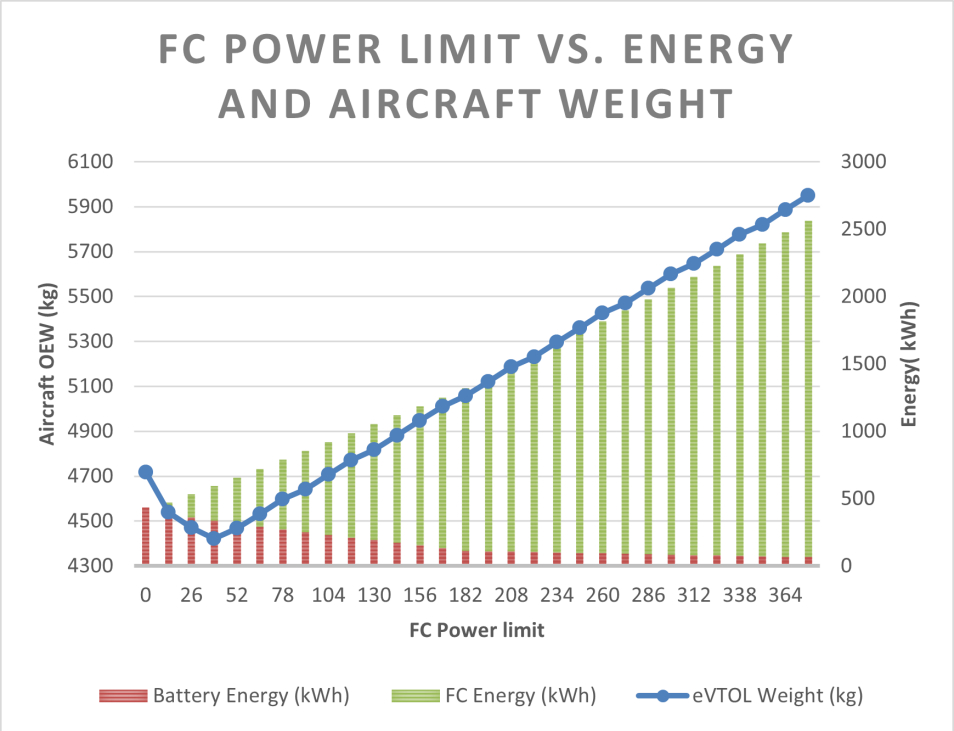


Figure 4.11: Effect of the fuel cell power limit on aircraft weight and energy consumption in the FC-BAT architecture with fixed BAT and FC voltages: 1207 V, 2204 V, respectively.

age in isolation, Figure 4.12 presents a two-dimensional carpet plot that simultaneously varies pack voltage and fuel cell voltage. Within this combined design space, increasing the fuel cell voltage shifts the carpet contours toward lower weight: higher cell voltage allows the same stack power to be delivered with a smaller active membrane area, reducing stack mass. Likewise, variations in battery voltage affect the number of cells in series and parallel as earlier mentioned, however these two effects only contribute modestly to the aircraft weight over the range considered. By overlaying both effects, the carpet plot clearly shows how battery and fuel cell voltages interact to define broader regions of near-optimal OEW, rather than isolated minima along a single axis. The selected operating point lies within a gently sloping trough, indicating some flexibility in voltage selection without large weight penalties. This integrated view underscores the importance of co-designing voltage levels across both electrochemical subsystems to minimize overall aircraft weight.

4.3.4. KEY TAKEAWAYS:

- **Battery Voltage (BAT-only)**: Varying the battery voltage in the all-electric architecture has a relatively small impact on total weight, suggesting that other parameters (e.g., energy density or capacity) may be more critical in mass-driven applications.

- **Fuel Cell Power Limit (FC-BAT)**: Increasing the fuel cell power capacity generally raises aircraft weight, reflecting the low specific power of current fuel cell technology; however, it can reduce the required battery mass. A careful balance is needed based on mission-specific power demands.

- **Fuel Cell Voltage (FC-BAT)**: Although less influential than the power limit, operating at a higher fuel cell voltage can trim total system mass by reducing stack size.

Overall, these sensitivity analyses confirm that small changes in certain parameters (like battery voltage) produce limited effects on weight, whereas more fundamental attributes (e.g. fuel cell power limit) can substantially shift both the weight and energy performance of the system. These findings only hold for the design assumption that the aircraft has a fixed MTOW limit, regardless of the weight. These insights can guide future optimizations and highlight where emerging technologies—such as higher-power-density fuel cells or improved battery chemistries—might offer the greatest impact.

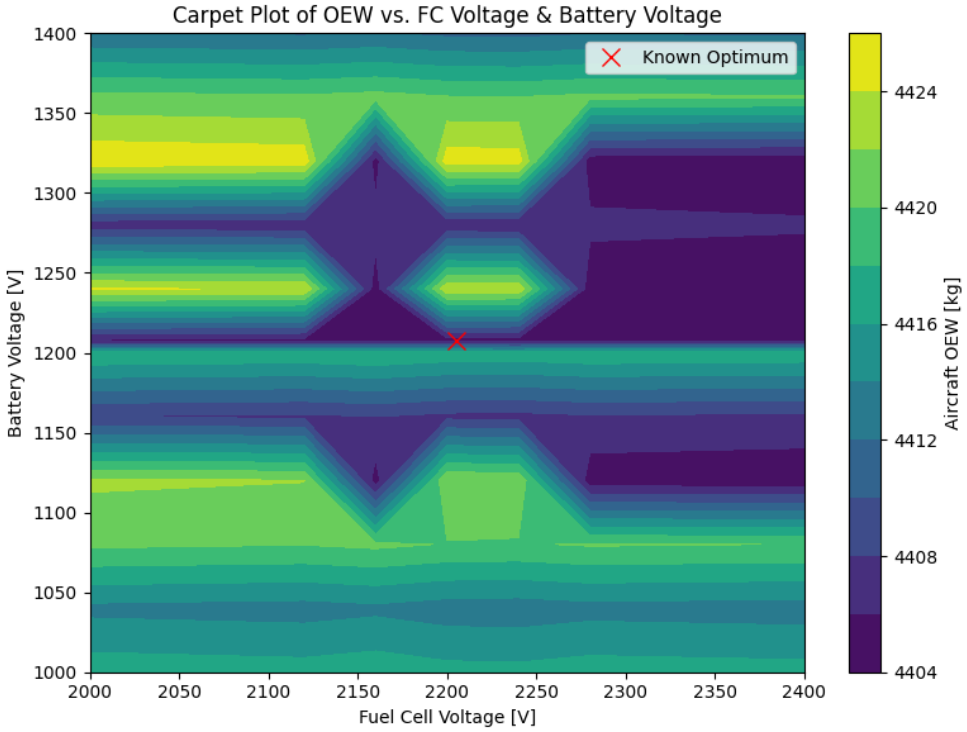


Figure 4.12: Combined influence of battery pack voltage and fuel cell stack voltage on operational empty weight in the FC-BAT architecture at FC Power limit = 39 kW.

4.4. TECHNOLOGICAL DEVELOPMENT

The preceding results demonstrate that current off-the-shelf battery technology is insufficient for direct implementation in the eVTOL concept studied, as the resulting aircraft weight exceeds the predefined MTOW. This raises an important question: *What battery specifications are required to enable a feasible eVTOL design?*

This section addresses potential advancements in both battery technology and hydrogen storage systems. Through sensitivity analyses and parametric studies, it explores how higher battery capacity, increased discharge rates, and improved hydrogen tank gravimetric efficiency could collectively yield an aircraft configuration that meets MTOW constraints.

4.4.1. BATTERY TECHNOLOGY IMPROVEMENTS

Key battery cell characteristics influencing propulsion system sizing and performance include the cell's capacity, maximum discharge current (C-rate), and nominal voltage. To understand the technological advancements necessary for feasibility, an additional sensitivity study examined how improving these parameters could enhance the overall energy and power density of a hybrid propulsion system.

Figure 4.13 shows the effect of varying both the nominal cell capacity (from 2.7 Ah to 13.5 Ah in increments of 2.7 Ah) and the C-rate (from 1C to 4C) on eVTOL weight. The cell capacity was then converted to specific energy to allow for easier comparison with different energy sources. The results indicate a strong dependence on discharge rate for lower-capacity cells; increasing the C-rate significantly reduces battery mass by enabling higher power output per unit mass. As cell capacity (or specific energy) grows, improvements in discharge rate have a diminishing impact on total weight.

Of particular note is that the difference between 3C and 4C discharge rates becomes marginal at higher capacities, implying that a C-rate of 3 is adequate to meet most high-power mission segments without necessitating additional parallel strings in the battery pack. Although these data points do not reflect fully optimized powertrain designs, they highlight the importance of both capacity and C-rate in achieving viable eVTOL configurations.

Figure 4.14 illustrates a further investigation where each battery capacity level underwent full optimization (e.g., adjusting other design variables accordingly). The findings show that a battery capacity of approximately 7.0 Ah, combined with a 3C discharge rate, is sufficient to drive the OEW below the 3175 kg MTOW

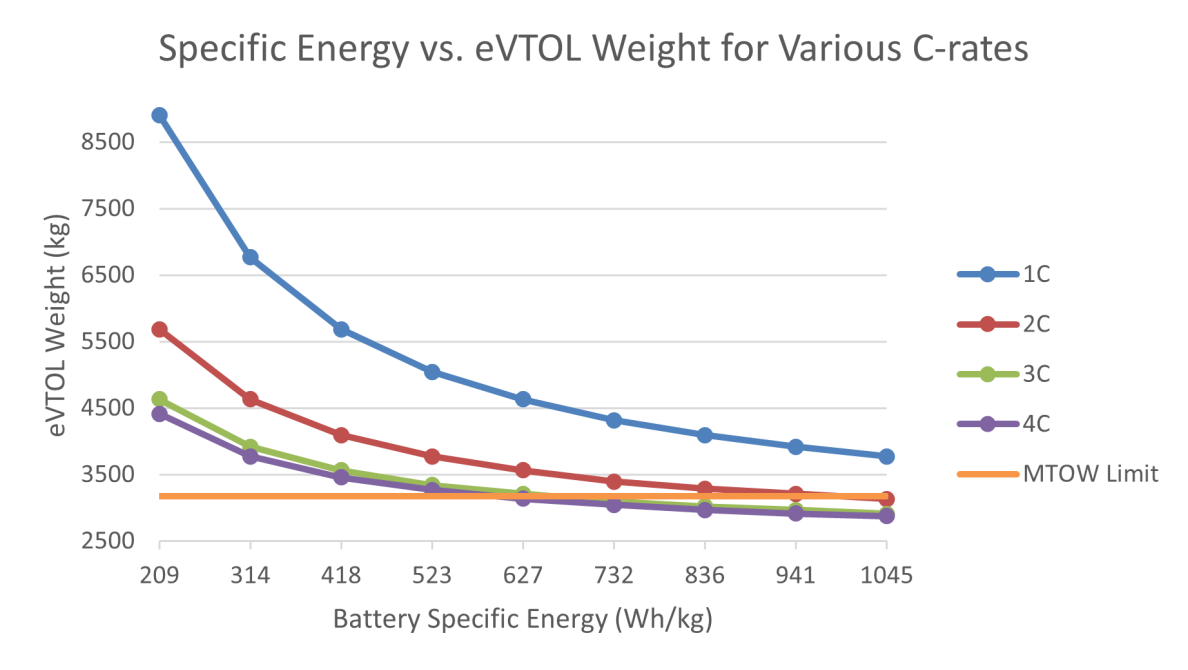


Figure 4.13: Effect of varying cell capacity and discharge current on eVTOL weight in the FC-BAT architecture with fixed BAT and FC voltages: 1207 V, 2204 V, respectively.

threshold. In contrast, the initial optimization employing a 2.7 Ah battery demonstrates that the battery capacity must be at least doubled to satisfy the requirements of this representative eVTOL mission.

Such results underscore the role of future battery cell innovations—both in energy density and discharge performance—in determining whether an all-electric or hybrid-electric eVTOL can meet aerospace weight limits.

4.4.2. ALTERNATIVE HYDROGEN STORAGE

Another critical factor influencing aircraft weight is the choice of fuel for the hybrid powertrain. Although compressed hydrogen gas was assumed in earlier analyses, liquid hydrogen offers much higher gravimetric efficiency. While compressed hydrogen tanks achieve only 5 to 10% gravimetric efficiency, liquid hydrogen tanks can exceed 60% [57]. This difference strongly suggests that shifting from gaseous to liquid hydrogen could unlock substantial weight savings.

Figure 4.15 highlights the weight breakdown of the FC-BAT architecture when employing a liquid hydrogen tank, where a gravimetric density ρ_{tank} of 1.5 kg_{H_2} per kg of tank mass was assumed. By contrast, the baseline compressed hydrogen tank was modeled with $\rho_{\text{compressed,gas}} = 0.0604\ kg_{\text{H}_2}$ per kg of tank mass. The notable reduction in system weight with the liquid hydrogen tank improves feasibility for meeting or approaching the MTOW requirement. The power split strategy also changes as a result of the changed power density of the hydrogen fuel cell. The FC power limit shifts from 39 kW towards 45 kW.

Although liquid hydrogen storage poses practical challenges such as cryogenic handling and infrastructure requirements [58] [57], its significantly higher gravimetric efficiency makes it an attractive avenue for future development of hybrid-electric aviation concepts.

4.4.3. KEY TAKEAWAYS:

- **Battery Technology**: Increasing cell capacity and discharge rates can substantially reduce battery mass, with a 5.4 Ah cell operating at 3C identified as sufficient to meet the MTOW constraint in one optimized scenario. Research into higher-energy-density chemistries could further expand the design space for eVTOL applications.

- **Hydrogen Storage**: Transitioning from compressed gaseous hydrogen to liquid hydrogen tanks may yield major weight savings, though cryogenic systems introduce new engineering and operational challenges.

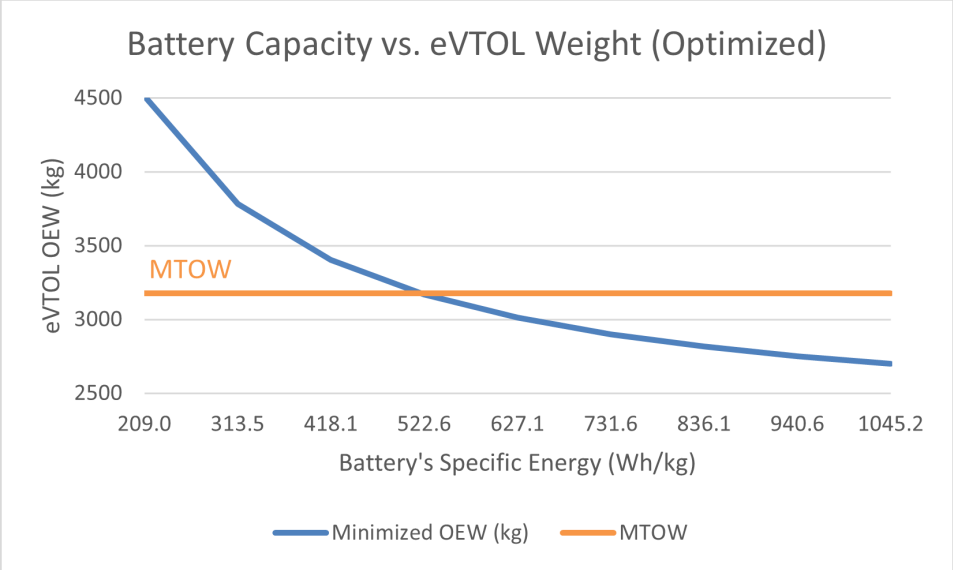


Figure 4.14: Minimizing eVTOL weight with varying cell capacity (constant C-rate) in the FC-BAT architecture.

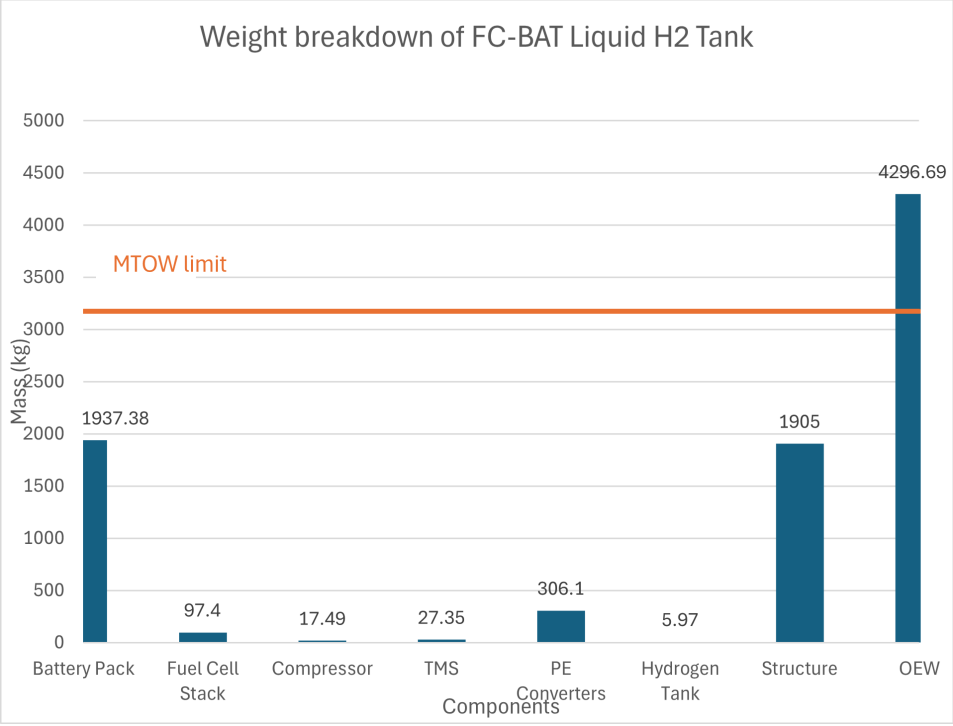


Figure 4.15: Weight breakdown of the FC-BAT architecture with a liquid hydrogen tank.

- **Integration and Validation**: Future work should combine refined system modeling (e.g., improved OCV–SOC curves, advanced aerodynamic considerations) with experimental data to validate next-generation battery packs and hydrogen storage solutions.
- **Roadmap to Feasibility**: Given the margins by which current concepts exceed MTOW, closing technology gaps in both battery and hydrogen infrastructure is essential for achieving practical hybrid-electric or fully electric eVTOL aircraft.

5

Conclusion and Recommendations

This chapter concludes the thesis while also providing recommendations for future research. It first summarises how the work answers the research questions set out in the introductory chapter, showing what the simulation tool revealed about different hybrid-electric eVTOL power-train options. It then explains what these insights mean for aircraft design and operation, before offering practical suggestions for future research. Together, these elements highlight the progress made and the steps still needed to turn hybrid-electric eVTOL concepts into viable aircraft for Urban Air Mobility.

5.1. Conclusion

This thesis set out to develop and validate a robust performance calculation tool capable of simulating and evaluating multiple hybrid electric eVTOL powertrain architectures, thereby addressing the research objective of enabling a systematic, comparative assessment of advanced propulsion configurations. By integrating detailed submodels for the battery, fuel cell, and turbogenerator and coupling these with a modular framework designed to accommodate varying power management strategies, this work provides an important step toward a deeper understanding of how hybrid propulsion systems can fulfill the requirements of Urban Air Mobility.

The research questions formulated in Chapter 1 focused on (i) identifying which modeling approaches and levels of fidelity are appropriate for component level simulations, (ii) The dynamic interactions between the component models in the powertrain architectures. (iii) applying the tool to examine distinct powertrain architectures. Throughout the study, each subquestion was systematically addressed. Equivalent circuit modeling proved sufficiently accurate to represent the dynamic and steady state voltage responses of lithium ion batteries, while the static Amphlett model offered a well balanced approach to capturing the electrochemical and thermodynamic processes of a PEM fuel cell. Performance maps generated by GSP allowed for integration of gas turbine results, demonstrating that properly calibrated off design simulations can be effectively be modeled alongside battery and fuel cell systems.

In comparing the different architectures, the following key findings emerged:

- An all battery powertrain (BAT only) currently faces prohibitive mass penalties given the limited energy density of available commercial cells. Even with an optimization routine targeting minimal empty weight, the overall system mass significantly exceeded the assumed MTOW, underscoring the need for battery technology advances before purely electric eVTOL solutions become feasible.

- Introducing a hybrid architecture that combines a fuel cell with a battery (FC-BAT) resulted in a lower operational empty weight than the BAT only scenario. By sizing the fuel cell to meet moderate power demands and delegating peak loads to the battery, the total battery capacity and thus mass was reduced. However, this configuration remained above the targeted MTOW, illustrating that incremental improvements in key parameters such as fuel cell power density and battery discharge capability are still necessary.

- A fuel cell turbogenerator arrangement (FC GT) showed promise in lowering operational empty weight, mainly because the turbogenerator can cater to high power phases without relying on a large battery

pack. Although practical challenges, especially in integrating hydrogen storage and ensuring dynamic response, remain, the FC-GT architecture highlights a tangible pathway for extended range eVTOL applications.

Taken together, these results confirm that while progress has been made in modeling, sizing, and optimizing hybrid propulsion systems for eVTOL missions, near term solutions will likely hinge on targeted technology advancements. The sensitivity analyses illustrated that improvements in battery specific energy, increased discharge rates, and more efficient hydrogen storage (for example, liquid hydrogen) are especially influential in bridging the gap between conceptual designs and MTOW limits. Additionally, the research confirmed that accurate power management, particularly non causal algorithms that exploit known mission profiles, can significantly influence component sizing and overall mass.

Reflecting on the research objective and questions, this thesis has demonstrated that:

- A flexible, open ended calculation tool can effectively model and compare the performance of multiple hybrid electric eVTOL propulsion systems.
- Appropriate fidelity for each component (for example, second order ECM for battery, Amphlett model for fuel cells) is critical to capturing both steady state and transient behaviors in an efficient manner.
- Distinct performance metrics, such as net system efficiency, battery State of Charge evolution, or the mass specific power of fuel cell stacks, reveal valuable trade offs for eVTOL designs that might otherwise be overlooked in single point analyses.
- Applying the tool to several candidate architectures clarifies the path forward: while none of the configurations evaluated met the strict MTOW requirement with current off the shelf technologies, the investigation identifies concrete technology roadmaps that, if pursued, could enable feasible hybrid and fully electric aircraft in the near future.

In conclusion, this thesis successfully meets its stated objective of creating and testing a performance calculation tool for multiple hybrid electric powertrain configurations, thereby addressing the fundamental research questions of how best to model, simulate, and compare advanced eVTOL propulsion systems. The findings highlight both the near term challenges in battery mass and fuel cell power density and the most promising directions for technological progress, paving the way for cleaner, more efficient aviation in the urban environment.

5.2. RECOMMENDATIONS

Future investigations should delve more deeply into higher fidelity thermal modeling for the battery and fuel cell systems, since thermal behavior directly impacts performance, reliability, and safety. Detailed thermal analyses would help optimize cooling strategies and ensure that peak operating conditions do not compromise component lifespans. In parallel, continuing refinements to power electronics models is essential for capturing the nuances of converter based transient dynamics, which become increasingly important as architectures grow in complexity and voltage requirements.

Further experimental data on next generation battery chemistries and hydrogen fuel cell stacks is also vital, because any uncertainty in input parameters can have a cascading effect on the accuracy of system level forecasts. Building a more extensive empirical dataset would allow for more precise parameterization and validation of the propulsion system models. One example is to implement the effects of the battery's temperature, battery age (state of health), and C-rate on the ECM parameters. An additional opportunity for expanding the models lies in incorporating more detailed aerodynamic and structural considerations. Right now, the modeling of the hybrid propulsion system is decoupled from the flight performance or structural requirements. Coupling the powertrain modeling to these two modules would align propulsion simulations with overall aircraft design criteria and yield a more comprehensive, integrated approach to eVTOL development.

Lastly, to better capture mission variability, implementing multi objective optimization frameworks could help balance trade offs such as total weight, cost, emissions, and noise. This would be especially important for real time power management systems that operate beyond a single, known profile. By exploring these areas, future research can build on the foundation established in this thesis to accelerate the design of practical, efficient, and sustainable eVTOL aircraft.

BIBLIOGRAPHY

[1] Lilium, *Different VTOL configurations,* (2020).

[2] T. C. Cano, I. Castro, A. Rodríguez, D. G. Lamar, Y. F. Khalil, L. Albiol-Tendillo, and P. Kshirsagar, *Future of Electrical Aircraft Energy Power Systems: An Architecture Review,* IEEE Transactions on Transportation Electrification **7**, 1915 (2021), conference Name: IEEE Transactions on Transportation Electrification.

[3] *Battery System Modeling* (2021).

[4] R. Bindu and S. Thale, *Sizing of hybrid energy storage system and propulsion unit for electric vehicle,* in *2017 IEEE Transportation Electrification Conference (ITEC-India)* (2017) pp. 1–6.

[5] G. Saldaña, J. I. San Martín, I. Zamora, F. J. Asensio, and O. Oñederra, *Analysis of the Current Electric Battery Models for Electric Vehicle Simulation,* Energies **12**, 2750 (2019).

[6] B. Ashok, C. Kannan, B. Mason, S. D. Ashok, V. Indragandhi, D. Patel, A. S. Wagh, A. Jain, and C. Kavitha, *Towards Safer and Smarter Design for Lithium-Ion-Battery-Powered Electric Vehicles: A Comprehensive Review on Control Strategy Architecture of Battery Management System,* Energies **15**, 4227 (2022).

[7] A. Fly and R. H. Thring, *Temperature regulation in an evaporatively cooled proton exchange membrane fuel cell stack,* International Journal of Hydrogen Energy **40**, 11976 (2015).

[8] L. Kunjumuhammed, S. Kuenzel, and B. Pal, *Chapter Three - Synchronous machine modelling,* in *Simulation of Power System with Renewables*, edited by L. Kunjumuhammed, S. Kuenzel, and B. Pal (Academic Press, 2020) pp. 39–80.

[9] A. A. Khalil, *Performance Modeling of a Hybrid Electric Propulsion System for eVTOL Aircraft,* Delft University of Technology (2024).

[10] K. Liu, K. Li, Q. Peng, and C. Zhang, *A brief review on key technologies in the battery management system of electric vehicles,* Frontiers of Mechanical Engineering **14**, 47 (2019).

[11] B. Arabsalmanabadi, N. Tashakor, S. Goetz, and K. Al-Haddad, *Li-ion Battery Models and A Simplified Online Technique to Identify Parameters of Electric Equivalent Circuit Model for EV Applications,* in *IECON 2020 The 46th Annual Conference of the IEEE Industrial Electronics Society* (2020) pp. 4164–4169, iSSN: 2577-1647.

[12] M. Camas-Náfate, A. Coronado-Mendoza, C. J. Vega-Gómez, and F. Espinosa-Moreno, *Modeling and Simulation of a Commercial Lithium-Ion Battery with Charge Cycle Predictions,* Sustainability **14**, 14035 (2022), number: 21 Publisher: Multidisciplinary Digital Publishing Institute.

[13] L. F. Krull and B. Muhammad, *Urban Air Mobility: Insights into Potentials and Challenges,* in *2022 25th International Symposium on Wireless Personal Multimedia Communications (WPMC)* (2022) pp. 267–272, iSSN: 1882-5621.

[14] A. Straubinger, R. Rothfeld, M. Shamiyeh, K.-D. Büchter, J. Kaiser, and K. O. Plötner, *An overview of current research and developments in urban air mobility – Setting the scene for UAM introduction,* Journal of Air Transport Management **87**, 101852 (2020).

[15] T. Donateo, A. Ficarella, and L. Surdo, *Energy consumption and environmental impact of Urban Air mobility,* IOP Conference Series: Materials Science and Engineering **1226**, 012065 (2022), publisher: IOP Publishing.

[16] G. L. Thomas, B. P. Malone, J. W. Chapman, and J. T. Csank, *Electrical Energy Storage Design Space Exploration for a Hybrid-Electric Six Passenger Quadrotor,* in *2022 IEEE Transportation Electrification Conference & Expo (ITEC)* (2022) pp. 319–324, iSSN: 2377-5483.

[17] A. Bacchini and E. Cestino, *Electric VTOL Configurations Comparison,* Aerospace **6**, 26 (2019).

[18] P. D. . R. G. A. C. J. S. S. K. Colleen Reiche, *Urban Air Mobility Market Study,* (2018), 10.7922/G2ZS2TRG, publisher: [object Object].

[19] N. Swaminathan, S. R. P. Reddy, K. RajaShekara, and K. S. Haran, *Flying Cars and eVTOLs—Technology Advancements, Powertrain Architectures, and Design,* IEEE Transactions on Transportation Electrification **8**, 4105 (2022), conference Name: IEEE Transactions on Transportation Electrification.

[20] R. Blockley, W. Shyy, R. K. Agarwal, F. Collier, A. Schäfer, and A. G. Seabridge, *Green aviation*, Encyclopedia of aerospace engineering (John Wiley & Sons, Inc., Chichester, West Sussex, United Kingdom, 2016).

[21] B. A. Adu-Gyamfi and C. Good, *Electric aviation: A review of concepts and enabling technologies,* Transportation Engineering **9**, 100134 (2022).

[22] W. Luo, *Design of large-scale battery pack for regional electric aircraft,* .

[23] M. T. Fard, J. He, H. Huang, and Y. Cao, *Aircraft Distributed Electric Propulsion Technologies—A Review,* IEEE Transactions on Transportation Electrification **8**, 4067 (2022).

[24] T. Zhao, *Propulsive Battery Packs Sizing for Aviation Applications,* .

[25] M. A. Rendón, C. D. Sánchez R., J. Gallo M., and A. H. Anzai, *Aircraft Hybrid-Electric Propulsion: Development Trends, Challenges and Opportunities,* Journal of Control, Automation and Electrical Systems **32**, 1244 (2021), company: Springer Distributor: Springer Institution: Springer Label: Springer Number: 5 Publisher: Springer US.

[26] P. Wheeler, T. S. Sirimanna, S. Bozhko, and K. S. Haran, *Electric/Hybrid-Electric Aircraft Propulsion Systems,* Proceedings of the IEEE **109**, 1115 (2021), conference Name: Proceedings of the IEEE.

[27] J. T. Pukrushpan, A. G. Stefanopoulou, and H. Peng, *Control of Fuel Cell Power Systems*, edited by M. J. Grimble and M. A. Johnson, Advances in Industrial Control (Springer London, London, 2004).

[28] M. Ramezanizadeh, M. Alhuyi Nazari, M. Hossein Ahmadi, and L. Chen, *A review on the approaches applied for cooling fuel cells,* International Journal of Heat and Mass Transfer **139**, 517 (2019).

[29] T. Marzougui, K. Neuhaus, L. Labracherie, and G. Scalabrin, *Optimal sizing of hybrid electric propulsion system for eVTOL,* IOP Conference Series: Materials Science and Engineering **1226**, 012070 (2022).

[30] Y. Xie, A. Savvarisal, A. Tsourdos, D. Zhang, and J. Gu, *Review of hybrid electric powered aircraft, its conceptual design and energy management methodologies,* Chinese Journal of Aeronautics **34**, 432 (2021).

[31] R. Khoury and D. W. Harder, *Numerical Methods and Modelling for Engineering* (Springer International Publishing, Cham, 2016).

[32] *Equivalent modeling, improvement, and state-space description,* in *Battery State Estimation: Methods and models*, edited by S. Wang (Institution of Engineering and Technology, 2021) pp. 45–84.

[33] J. Meng, M. Swierczynski, D.-I. Stroe, A.-I. Stroe, and R. Teodorescu, *An Overview and Comparison of Online Implementable SOC Estimation Methods for Lithium-Ion Battery,* IEEE TRANSACTIONS ON INDUSTRY APPLICATIONS **54** (2018).

[34] S. T, C. S. S, C. V. N. S. Pavan Pusya, and R. Bhuvaneswari, *An Investigation into Battery Modelling for Electric Vehicles and Applications for Electric Power Systems,* in *2022 6th International Conference on Computing Methodologies and Communication (ICCMC)* (IEEE, Erode, India, 2022) pp. 519–525.

[35] J. Xu, B. Cao, J. Cao, Z. Zou, C. C. Mi, and Z. Chen, *A Comparison Study of the Model Based SOC Estimation Methods for Lithium-Ion Batteries,* in *2013 IEEE Vehicle Power and Propulsion Conference (VPPC)* (IEEE, Beijing, China, 2013) pp. 1–5.

[36] A. X. Liu, B. Kai Lyu, C. S. Chen, D. Y. Guo, E. S. Xing, F. Di Wang, and G. J. Sun, *SOH Estimation Method for Lithium-ion Batteries Based on Partial Charging Voltage Segments,* in *2023 11th International Conference on Power Electronics and ECCE Asia (ICPE 2023 - ECCE Asia)* (IEEE, Jeju Island, Korea, Republic of, 2023) pp. 1597–1602.

[37] G.-J. Chen and W.-H. Chung, *Evaluation of Charging Methods for Lithium-Ion Batteries,* Electronics **12**, 4095 (2023).

[38] S. Haghighi, K. Askari, S. Hamidi, and M. M. Rahimi, *OPEM : Open Source PEM Cell Simulation Tool,* Journal of Open Source Software **3**, 676 (2018).

[39] Y. Huang, X. Xiao, H. Kang, J. Lv, R. Zeng, and J. Shen, *Thermal management of polymer electrolyte membrane fuel cells: A critical review of heat transfer mechanisms, cooling approaches, and advanced cooling techniques analysis,* Energy Conversion and Management **254**, 115221 (2022).

[40] W. P. J. Visser and M. J. Broomhead, *GSP, a Generic Object-Oriented Gas Turbine Simulation Environment,* (American Society of Mechanical Engineers Digital Collection, 2014).

[41] V. Madonna, P. Giangrande, and M. Galea, *Electrical Power Generation in Aircraft: Review, Challenges, and Opportunities,* IEEE Transactions on Transportation Electrification **4**, 646 (2018).

[42] M. E. Bell and J. S. Litt, *An Electrical Modeling and Thermal Analysis Toolbox for Electrified Aircraft Propulsion Simulation,* in *AIAA Propulsion and Energy 2020 Forum* (American Institute of Aeronautics and Astronautics, VIRTUAL EVENT, 2020).

[43] O. Schmitz and M. Hornung, *Methods for simulation and analysis of hybrid electric propulsion systems,* CEAS Aeronautical Journal **6**, 245 (2015).

[44] M. J. Arzberger and D. Zimmer, *A Modelica-based environment for the simulation of hybrid-electric propulsion systems,* (2019) pp. 471–480.

[45] M. Podlaski, L. Vanfretti, H. Nademi, P. J. Ansell, K. S. Haran, and T. Balachandran, *Initial Steps in Modeling of CHEETA Hybrid Propulsion Aircraft Vehicle Power Systems using Modelica,* in *AIAA Propulsion and Energy 2020 Forum* (American Institute of Aeronautics and Astronautics, VIRTUAL EVENT, 2020).

[46] H. Hizarci, O. Demirel, K. Kalayci, and U. Arifoglu, *An Overview of Aircraft Electric Power System for Sustainable Aviation,* in *New Frontiers in Sustainable Aviation* (Springer, Cham, 2022) pp. 113–146, iSSN: 2730-7786.

[47] C. Hall, C. L. Pastra, A. Burrell, J. Gladin, and D. N. Mavris, *Projecting Power Converter Specific Power Through 2050 for Aerospace Applications,* in *2022 IEEE Transportation Electrification Conference & Expo (ITEC)* (IEEE, Anaheim, CA, USA, 2022) pp. 760–765.

[48] R. de Vries, R. E. Wolleswinkel, D. R. Jacobson, M. Bonnema, and S. Thiede, *BATTERY PERFORMANCE METRICS FOR LARGE ELECTRIC PASSENGER AIRCRAFT,* .

[49] D. S. Shakariyants, J. P. Buijtenen van, W. P. Visser, and F. Montella, *(Introduction, Ideal Cycles, Real Cycles, Shaft power Gas turbines, Aircraft Gas Turbines and Performance Characteristics),* (2021).

[50] J. Larminie, A. Dicks, and M. S. McDonald, *Fuel cell systems explained*, Vol. 2 (J. Wiley Chichester, UK, 2003).

[51] J. C. Amphlett, R. M. Baumert, R. F. Mann, B. A. Peppley, P. R. Roberge, and T. J. Harris, *Performance Modeling of the Ballard Mark IV Solid Polymer Electrolyte Fuel Cell: I . Mechanistic Model Development,* Journal of The Electrochemical Society **142**, 1 (1995), publisher: IOP Publishing.

[52] G. L. M. Vonhoff, *Conceptual Design of Hydrogen Fuel Cell Aircraft,* (2021).

[53] J. W. Chapman, S. L. Schnulo, and M. P. Nitzsche, *Development of a Thermal Management System for Electrified Aircraft*, Tech. Rep. E-19786 (2020) nTRS Author Affiliations: NASA Glenn Research Center NTRS Document ID: 20200001620 NTRS Research Center: Glenn Research Center (GRC).

[54] R. Storn and K. Price, *Differential Evolution – A Simple and Efficient Heuristic for global Optimization over Continuous Spaces,* Journal of Global Optimization **11**, 341 (1997), company: Springer Distributor: Springer Institution: Springer Label: Springer Number: 4 Publisher: Kluwer Academic Publishers.

[55] J. T. Warner, *Chapter 3 - Basic Terminology,* in *The Handbook of Lithium-Ion Battery Pack Design (Second Edition)*, edited by J. T. Warner (Elsevier, 2024) pp. 53–69.

[56] K. Omiloli, A. Awelewa, I. Samuel, O. Obiazi, and J. Katende, *State of charge estimation based on a modified extended Kalman filter,* International Journal of Electrical and Computer Engineering (IJECE) **13**, 5054 (2023).

[57] S. Tiwari, M. J. Pekris, and J. J. Doherty, *A review of liquid hydrogen aircraft and propulsion technologies,* International Journal of Hydrogen Energy **57**, 1174 (2024).

[58] P. Muthukumar, A. Kumar, M. Afzal, S. Bhogilla, P. Sharma, A. Parida, S. Jana, E. A. Kumar, R. K. Pai, and I. P. Jain, *Review on large-scale hydrogen storage systems for better sustainability,* International Journal of Hydrogen Energy **48**, 33223 (2023).

A

Battery Data Sheet

Panasonic

Lithium Ion NCR18650

Features & Benefits

- High energy density
- Long stable power and long run time
- Ideal for notebook PCs, boosters, portable devises, etc.

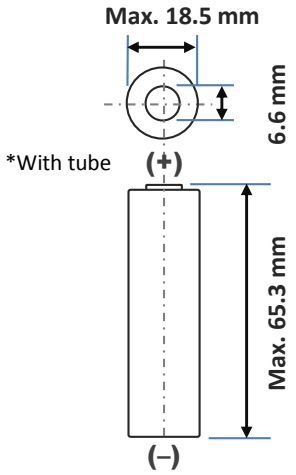
Specifications

Rated capacity(1)	Min. 2700mAh
Capacity(2)	Min. 2750mAh Typ. 2900mAh
Nominal voltage	3.6V
Charging	CC-CV, Std. 1925mA, 4.20V, 3.0 hrs
Weight (max.)	46.5 g
Temperature	Charge*: 0 to +45°C Discharge: -20 to +60°C Storage: -20 to +50°C
Energy density(3)	Volumetric: 577 Wh/l Gravimetric: 214 Wh/kg

* At temperatures below 10°C, charge at a 0.35C rate.

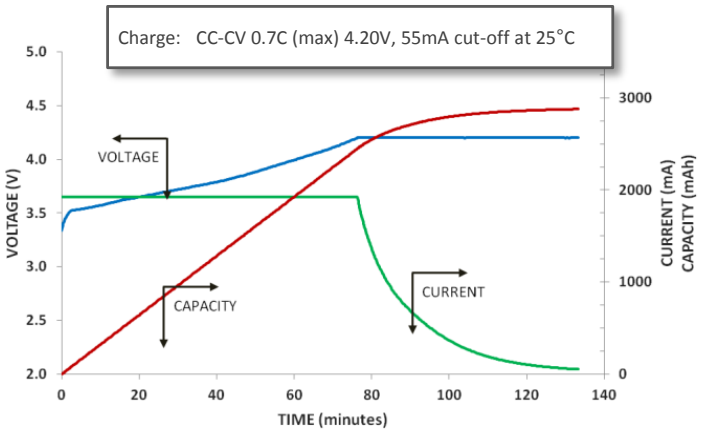
(1) At 20°C (2) At 25°C (3) Energy density based on bare cell dimensions

Dimensions

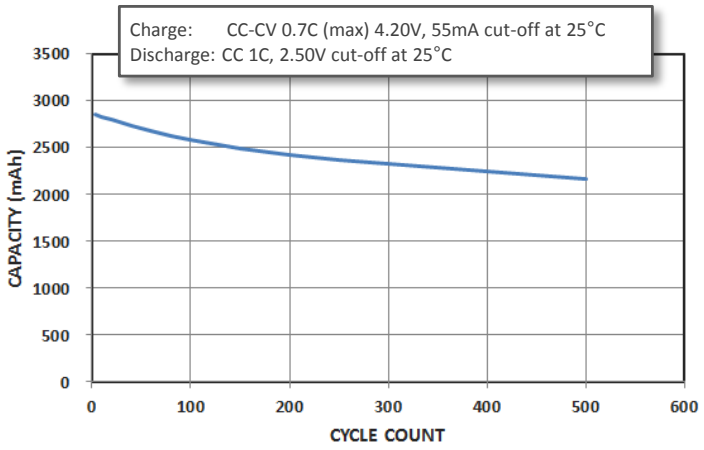


For Reference Only

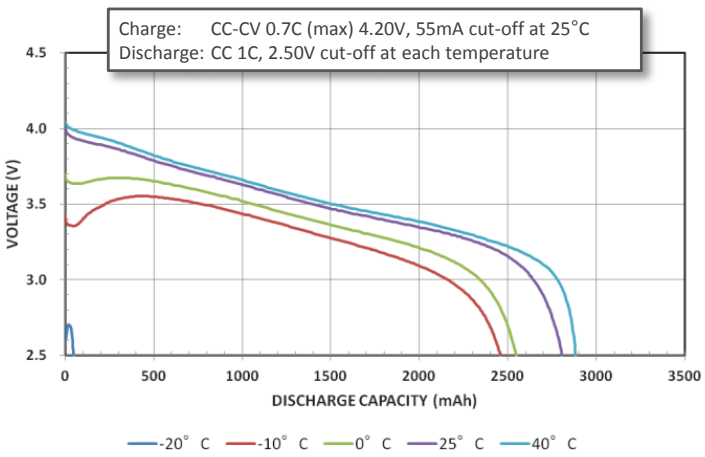
Charge Characteristics



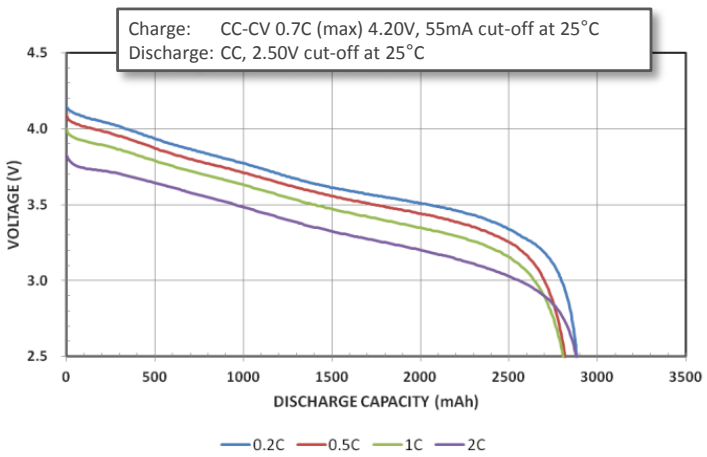
Cycle Life Characteristics



Discharge Characteristics (by temperature)



Discharge Characteristics (by rate of discharge)



B

Fuel Cell - Turbogenerator Simulation Output

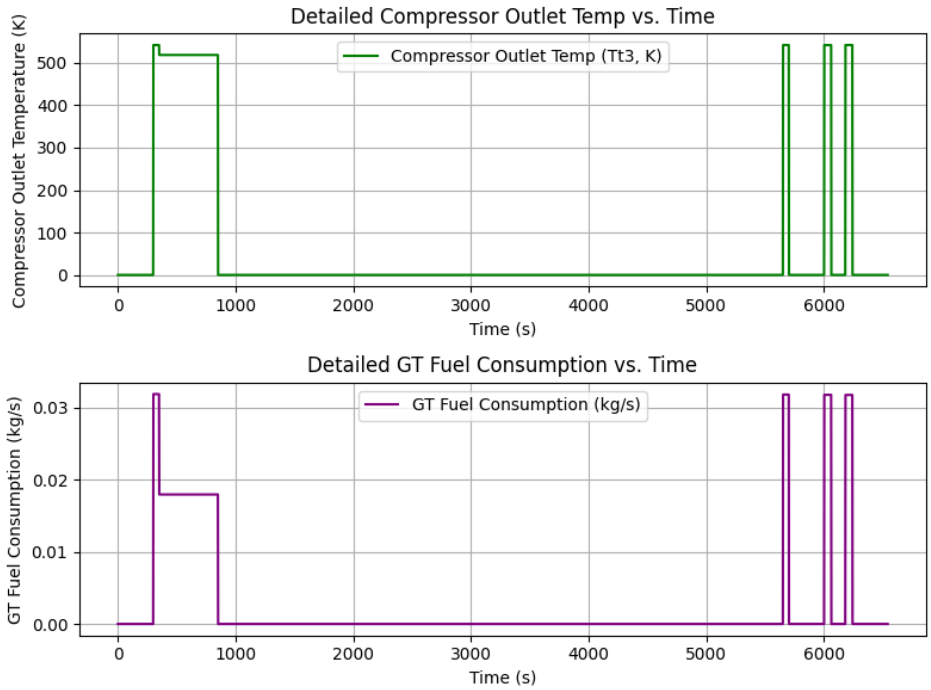


Figure B.1: The compressor outlet temperature and the fuel mass flow rate along the mission profile.

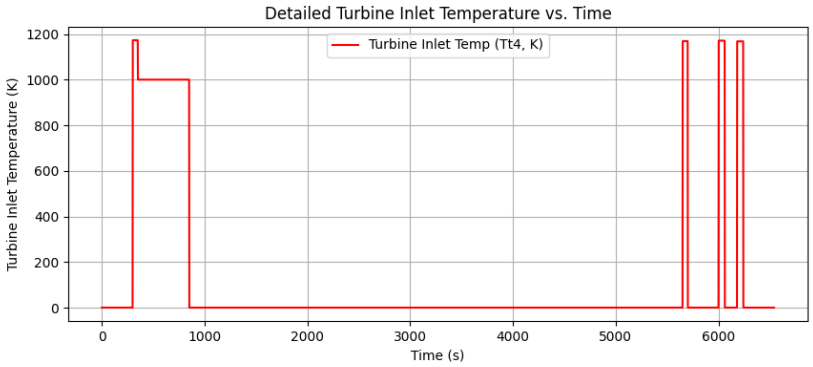


Figure B.2: The turbine inlet temperature along the mission profile.

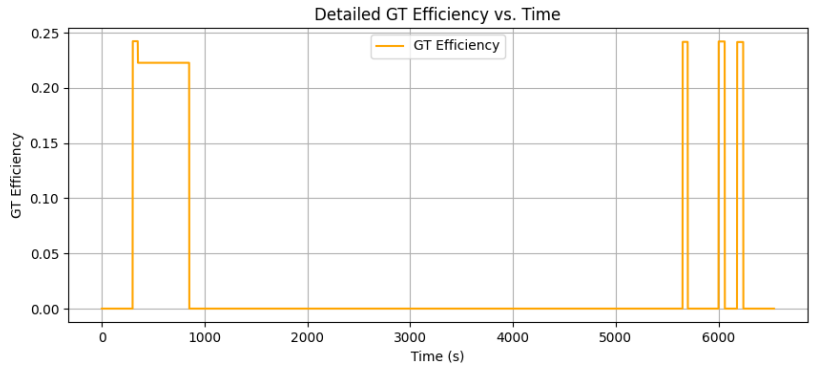


Figure B.3: The efficiency of the gas turbine engine along the mission profile.

C

Code Files

C.1. Flight Data Reader

```python
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from ambiance import Atmosphere

def read_data(file_name):
    df_flight = pd.read_csv(file_name)

    flight_condition = list(df_flight[df_flight.columns[0]])
    Zh = list(df_flight['Altitude [m]'])
    Pw_r = list(df_flight['Power [kW]'])
    Time = list(df_flight['Time [s]'])
    timeF = list(np.cumsum(Time))
    v_x = list(df_flight['v_x [m/s]'])
    v_z = list(df_flight['v_z [m/s]'])

    TimeP = np.linspace(0, timeF[-1], int(timeF[-1]))

    # Initialize the new power list
    Pw_P = []

    # Initialize the index for the timeF and Pw_r lists
    index = 0

    # For each time in TimeP
    for time in TimeP:
        # If the time is greater than or equal to the current time in timeF
        if index < len(timeF) - 1 and time >= timeF[index]:
            # Move to the next time and power
            index += 1
        # Add the current power to the new power list
        Pw_P.append(Pw_r[index])

    # Convert the new power list to a numpy array
    Pw_P = np.array(Pw_P)

    return timeF, Zh, Pw_r, TimeP, Pw_P, flight_condition, v_x, v_z, df_flight


def read_data_fc(file_name, fuel_cell_max_power=206.6):  # Example constant power for fuel cell in kW # New method
    # Read the CSV file
    df_flight = pd.read_csv(file_name)

    # Extract relevant columns
    flight_condition = list(df_flight[df_flight.columns[0]])
    Zh = list(df_flight['Altitude [m]'])
    Pw_r = list(df_flight['Power [kW]'])
```

```
    Time = list(df_flight['Time [s]'])
    timeF = list(np.cumsum(Time))
    v_x = list(df_flight['v_x [m/s]'])
    v_z = list(df_flight['v_z [m/s]'])

    # Define TimeP with a finer resolution
    TimeP = np.linspace(0, timeF[-1], int(timeF[-1]))

    # Initialize the new power lists
    Pw_P = []
    Pw_fuel_cell_P = []
    Pw_battery_P = []

    # Initialize the index for timeF and Pw_r lists
    index = 0

    # For each time in TimeP
    for time in TimeP:
        # If the time is greater than or equal to the current time in timeF
        if index < len(timeF) - 1 and time >= timeF[index]:
            # Move to the next time and power
            index += 1
        # Total power demand
        power = Pw_r[index]

        # Calculate the power provided by the fuel cell
        if power > 0:
            fuel_cell_power = fuel_cell_max_power
        else:
            fuel_cell_power = 0

        # Calculate the power provided by the battery
        if power > fuel_cell_power:
            # If power demand exceeds fuel cell power, battery provides the remaining power
            battery_power = power - fuel_cell_power
        else:
            # If fuel cell power exceeds power demand, battery absorbs the excess power (
    negative value)
            battery_power = power - fuel_cell_power

        # Append to respective lists
        Pw_P.append(power)
        Pw_fuel_cell_P.append(fuel_cell_power)
        Pw_battery_P.append(battery_power)

    # Convert lists to numpy arrays
    Pw_P = np.array(Pw_P)
    Pw_fuel_cell_P = np.array(Pw_fuel_cell_P)
    Pw_battery_P = np.array(Pw_battery_P)

    return timeF, Zh, Pw_r, TimeP, Pw_P, Pw_fuel_cell_P, Pw_battery_P, flight_condition, v_x,
     v_z, df_flight


def read_data_fc_bop(file_name, fuel_cell_max_power=206.6):
    """
    Reads mission data from a CSV and produces arrays for:
     - segment-based time (timeF), alt (Zh), power (Pw_r)
     - finer 1-second arrays (TimeP, Pw_P, Pw_fuel_cell_P, Pw_battery_P)
     - and extended 1-second arrays for altitude (Zh_P) and horizontal velocity (v_x_P).
    """

    # Read the CSV file
    df_flight = pd.read_csv(file_name)

    # Extract relevant columns
    flight_condition = list(df_flight[df_flight.columns[0]])
    Zh = list(df_flight['Altitude [m]'])     # segment-based altitude
    Pw_r = list(df_flight['Power [kW]'])     # segment-based power
    Time = list(df_flight['Time [s]'])       # segment durations
    timeF = list(np.cumsum(Time))            # cumulative time at each segment
```

```
    v_x = list(df_flight['v_x [m/s]'])      # segment-based horizontal velocity
    v_z = list(df_flight['v_z [m/s]'])

    # Define TimeP with a finer resolution (1-second increments)
    # We'll add +1 so we include the last integer second
    TimeP = np.linspace(0, timeF[-1], int(timeF[-1]) + 1)

    # Initialize lists for the 1-second power outputs
    Pw_P = []
    Pw_fuel_cell_P = []
    Pw_battery_P = []
    Zh_P = []    # 1-second altitude
    v_x_P = []   # 1-second horizontal velocity

    # Index to track which segment we're in
    index = 0

    # Loop over the 1-second time array
    for t in TimeP:
        # If the time t exceeds the boundary of the current segment, increment index
        if index < len(timeF) - 1 and t >= timeF[index]:
            index += 1

        # ----- Altitude at this time step -----
        alt = Zh[index]
        Zh_P.append(alt)

        # ----- Horizontal velocity at this time step -----
        vx_val = v_x[index]
        v_x_P.append(vx_val)

        # ----- Power demand logic -----
        power = Pw_r[index]
        # Fuel cell power
        if power > 0:
            fuel_cell_power = fuel_cell_max_power
        else:
            fuel_cell_power = 0
        # Battery power
        if power > fuel_cell_power:
            battery_power = power - fuel_cell_power
        else:
            battery_power = power - fuel_cell_power

        Pw_P.append(power)
        Pw_fuel_cell_P.append(fuel_cell_power)
        Pw_battery_P.append(battery_power)

    # Convert lists to NumPy arrays for convenience
    Pw_P = np.array(Pw_P)
    Pw_fuel_cell_P = np.array(Pw_fuel_cell_P)
    Pw_battery_P = np.array(Pw_battery_P)
    Zh_P = np.array(Zh_P)
    v_x_P = np.array(v_x_P)

    return timeF, Zh, Pw_r, TimeP, Pw_P, Pw_fuel_cell_P, Pw_battery_P, flight_condition, v_x,
     v_z, df_flight, Zh_P, v_x_P

def plot_data(timeF, Zh, TimeP, Pw_P):
    fig, ax1 = plt.subplots()

    ax1.plot(timeF, Zh, 'b-')
    ax1.set_xlabel('Time [s]')
    ax1.set_ylabel('Altitude [m]', color='b')
    ax1.tick_params('y', colors='b')

    ax2 = ax1.twinx()
    ax2.plot(TimeP, Pw_P, 'r-')
    ax2.set_ylabel('Power [kW]', color='r')
    ax2.tick_params('y', colors='r')
```

```
    plt.title('Time vs Altitude and Power')
    plt.show()

def plot_data_fc(timeF, Zh, TimeP, Pw_P, Pw_fuel_cell_P, Pw_battery_P):
    fig, ax1 = plt.subplots()

    ax1.plot(timeF, Zh, 'b-')
    ax1.set_xlabel('Time [s]')
    ax1.set_ylabel('Altitude [m]', color='b')
    ax1.tick_params('y', colors='b')

    ax2 = ax1.twinx()
    ax2.plot(TimeP, Pw_P, 'r-', label='Total Power')
    ax2.plot(TimeP, Pw_fuel_cell_P, 'g--', label='Fuel Cell Power')
    ax2.plot(TimeP, Pw_battery_P, 'm--', label='Battery Power')
    ax2.set_ylabel('Power [kW]', color='r')
    ax2.tick_params('y', colors='r')
    ax2.legend(loc='upper right')

    plt.title('Time vs Altitude and Power')
    plt.show()

def plot_data_fc_seperate(timeF, Zh, TimeP, Pw_P, Pw_fuel_cell_P, Pw_battery_P):
    fig, axs = plt.subplots(2, 1, figsize=(6, 4), dpi=150, constrained_layout=True)

    # Plot fuel cell power vs time
    axs[0].plot(TimeP, Pw_fuel_cell_P, 'g--', label='Fuel Cell Power')
    axs[0].set_xlabel('Time [s]')
    axs[0].set_ylabel('Fuel Cell Power [kW]', color='g')
    axs[0].set_title('Time vs Fuel Cell Power')
    axs[0].grid(True)
    axs[0].legend()

    # Plot battery power vs time
    axs[1].plot(TimeP, Pw_battery_P, 'm--', label='Battery Power')
    axs[1].set_xlabel('Time [s]')
    axs[1].set_ylabel('Battery Power [kW]', color='m')
    axs[1].set_title('Time vs Battery Power')
    axs[1].grid(True)
    axs[1].legend()

    plt.tight_layout()
    plt.show()


# New function to calculate the energy demand based on the mission profile
def calculate_energy_demand(TimeP, Pw_P):

    # Convert power from kW to W for the calculation
    Pw_P_watts = Pw_P * 1000

    # Calculate the time difference between consecutive time points
    delta_t = np.diff(TimeP, prepend=0)  # Prepend a zero to maintain length consistency

    # Calculate energy for each time interval and sum it up
    energy_joules = np.sum(Pw_P_watts * delta_t)

    # Convert joules to Wh
    energy_Wh = energy_joules / 3600

    return energy_Wh

def calculate_energy_demand_positive(TimeP, Pw_P):
    """
    Calculate the energy demand (in Wh) by integrating the power profile (in kW)
    over time, but only considering the area under the first continuous positive segment.

    Parameters:
        TimeP (np.array): Time stamps in seconds.
        Pw_P (np.array): Power profile in kW.

```

```
    Returns:
        energy_Wh (float): Energy demand in Wh.
    """
    # Ensure inputs are numpy arrays
    TimeP = np.array(TimeP)
    Pw_P = np.array(Pw_P)

    # Find the index of the first negative power value
    neg_indices = np.where(Pw_P < 0)[0]

    if neg_indices.size == 0:
        # No negative values: integrate the whole profile using the trapezoidal rule.
        energy_joules = np.trapz(Pw_P, TimeP) * 1000  # Convert kW-s to J
    else:
        idx = neg_indices[0]
        if idx == 0:
            energy_joules = 0  # The profile starts negative
        else:
            # Integrate the samples before the first negative using trapezoidal rule.
            energy_joules = np.trapz(Pw_P[:idx], TimeP[:idx]) * 1000
            # Now, assume that the positive segment extends until TimeP[idx]
            # using the last measured positive power (Pw_P[idx-1]) over the remaining
    interval.
            extra_interval = TimeP[idx] - TimeP[idx - 1]
            energy_joules += Pw_P[idx - 1] * extra_interval * 1000

    # Convert energy from joules to Wh
    energy_Wh = energy_joules / 3600
    return energy_Wh



def get_flying_speed(flight_phase, df_flight):
    # Filter the dataframe based on the flight phase
    df_filtered = df_flight[df_flight[df_flight.columns[0]] == flight_phase]

    # Get the flying speed v_x
    v_x = df_filtered['v_x [m/s]'].values

    # If v_x is not empty, return the first value as an integer
    if v_x.size > 0:
        return float(v_x[0])
    # If v_x is empty, return None
    else:
        return None


def get_altitude(flight_phase, df_flight):
    # Filter the dataframe based on the flight phase
    df_filtered = df_flight[df_flight[df_flight.columns[0]] == flight_phase]

    Zh = df_filtered['Altitude [m]'].values

    if Zh.size >= 0:
        return float(Zh[0])
    else:
        return None


def get_power(flight_phase, df_flight):
    # Filter the dataframe based on the flight phase
    df_filtered = df_flight[df_flight[df_flight.columns[0]] == flight_phase]

    Pw = df_filtered['Power [kW]'].values

    if Pw.size >= 0:
        return float(Pw[0])
    else:
        return None


```

```python
def get_flying_Mach(Zh, vx):
    atmosphere = Atmosphere(Zh)
    # Calculate the speed of sound
    a0 = atmosphere.speed_of_sound[0]

    # Calculate the Mach number
    Mach = vx / a0

    return Mach

def get_elapsed_time(flight_phase, df_flight):
    # Filter the dataframe based on the flight phase
    df_filtered = df_flight[df_flight[df_flight.columns[0]] == flight_phase]

    # Get the elapsed time (Time [s])
    elapsed_time = df_filtered['Time [s]'].values

    # If elapsed_time is not empty, return the first value as a float
    if elapsed_time.size > 0:
        return float(elapsed_time[0])
    # If elapsed_time is empty, return None
    else:
        return None

def calulate_air_properties(altitude, Mach):
    atmosphere = Atmosphere(altitude)
    kappa_air = 1.4  # ratio of specific heats for air
    # Calculate the speed of sound
    a0 = atmosphere.speed_of_sound[0]

    # Calculate the static temperature
    T0 = atmosphere.temperature[0]

    # Calculate the total temperature
    Tt0 = T0 * (1 + ((kappa_air - 1) * 0.5 * Mach ** 2))

    # Calculate the static pressure
    p0 = atmosphere.pressure[0]

    # Calculate the total pressure

    Pt0 = p0 * (1 + Mach ** 2 * 0.5 * (kappa_air - 1)) ** (kappa_air / (kappa_air - 1))

    # Calculate the density
    rho0 = atmosphere.density[0]

    # Calculate the dynamic pressure
    q0 = 0.5 * rho0 * (Mach * a0) ** 2

    return T0, Tt0, p0, Pt0, rho0, q0

def power_to_frequency_domain(TimeP, Pw_P):
    """
    Converts power demand from the time domain to the frequency domain.

    Parameters:
    - TimeP: Array of time points (s).
    - Pw_P: Array of power demand values (W) corresponding to the time points.

    Returns:
    - frequencies: Array of frequencies (Hz) corresponding to the power spectrum.
    - power_spectrum: Array of power magnitudes in the frequency domain.
    """
    # Time step (assumes evenly spaced time intervals in TimeP)
    delta_t = TimeP[1] - TimeP[0]

    # Perform the Fourier Transform on the power demand
    power_fft = np.fft.fft(Pw_P)

    # Compute the corresponding frequency bins
    n = len(Pw_P)
```

```
399     frequencies = np.fft.fftfreq(n, delta_t)
401     # Only return the positive half of the spectrum
        positive_frequencies = frequencies[:n // 2]
403     power_spectrum = np.abs(power_fft[:n // 2])  # Magnitude of the FFT
405     return positive_frequencies, power_spectrum
407 def divide_power_mean_based(Pw_P):
        """
409     Divides power demand between the fuel cell and battery based on mean power.
411     Parameters:
        - TimeP: Array of time points (s).
413     - Pw_P: Array of total power demand (kW).
415     Returns:
        - fuel_cell_power_profile: Array of power provided by the fuel cell (kW).
417     - battery_power_profile: Array of power provided by the battery (kW).
        - mean_power: Mean power demand calculated (kW).
419     """
        # Calculate the mean power demand
421     mean_power = np.mean(Pw_P)
423     # Initialize power profiles
        fuel_cell_power_profile = np.full_like(Pw_P, mean_power)  # Fuel cell provides constant
        power
425     battery_power_profile = Pw_P - fuel_cell_power_profile  # Battery handles the residual
        power
427     return fuel_cell_power_profile, battery_power_profile, mean_power
429 def plot_thesis_friendly_mission_profile(timeF, Zh, TimeP, Pw_P):
        # Convert time from seconds to minutes
431     timeF_minutes = [t / 60 for t in timeF]
        TimeP_minutes = [t / 60 for t in TimeP]
433
        # Create the figure and axes for stacked plots
435     fig, (ax1, ax2) = plt.subplots(2, 1, figsize=(10, 8), sharex=True)
437     # Plot mission profile (altitude vs. time)
        ax1.plot(timeF_minutes, Zh, 'b-', label='Altitude')
439     ax1.set_ylabel('Altitude [m]', fontsize=12, color='b')
        ax1.tick_params(axis='y', labelcolor='b')
441     ax1.grid(True, linestyle='--', alpha=0.6)
        ax1.set_title('Mission Profile and Power Demand', fontsize=14, fontweight='bold')
443
        # Plot power demand (power vs. time)
445     ax2.plot(TimeP_minutes, Pw_P, 'r-', label='Power Demand')
        ax2.set_xlabel('Time [minutes]', fontsize=12)
447     ax2.set_ylabel('Power [kW]', fontsize=12, color='r')
        ax2.tick_params(axis='y', labelcolor='r')
449     ax2.grid(True, linestyle='--', alpha=0.6)
451     # Adjust layout for better appearance
        plt.tight_layout()
453     plt.show()
455 def read_data_fs(file_name):
        df_flight = pd.read_csv(file_name)
457
        # Original data columns from the CSV
459     flight_condition = list(df_flight[df_flight.columns[0]])
        Zh = list(df_flight['Altitude [m]'])
461     Pw_r = list(df_flight['Power [kW]'])
        Time = list(df_flight['Time [s]'])
463     # Create a cumulative time vector corresponding to each row in the file.
        timeF = list(np.cumsum(Time))
465     v_x = list(df_flight['v_x [m/s]'])
        v_z = list(df_flight['v_z [m/s]'])
467
```

```
    # Create a high-resolution time vector (TimeP) with one sample per second,
    # spanning from 0 to the total mission time.
    TimeP = np.linspace(0, timeF[-1], int(timeF[-1]))

    # Initialize new lists for the high-resolution data.
    Pw_P = []               # Power at each second.
    flight_condition_ts = []  # Flight condition at each time step.
    Zh_ts = []              # Altitude at each time step.
    v_x_ts = []             # Horizontal velocity at each time step.
    v_z_ts = []             # Vertical velocity at each time step.

    index = 0  # This index points to the current row in the original data.
    for t in TimeP:
        # Advance the index if the high-res time has reached (or passed) the next cumulative
    time.
        if index < len(timeF) - 1 and t >= timeF[index]:
            index += 1
        # Append the data corresponding to the current row (index).
        Pw_P.append(Pw_r[index])
        flight_condition_ts.append(flight_condition[index])
        Zh_ts.append(Zh[index])
        v_x_ts.append(v_x[index])
        v_z_ts.append(v_z[index])

    # Convert Pw_P to a NumPy array (if needed for further calculations)
    Pw_P = np.array(Pw_P)

    # Return all the original data (if needed) plus the high-resolution lists.
    return timeF, Zh, Pw_r, TimeP, Pw_P, flight_condition_ts, Zh_ts, v_x_ts, v_z_ts,
    df_flight
```

flightdata_reader.py

C.2. PERFORMANCE CALCULATION TOOL

```
import matplotlib.pyplot as plt
import flightdata_reader as fr
import battery_model as bm
import fc_model as fc
from BOP import *
from scipy.optimize import differential_evolution
import numpy as np

# ----------------------------
# Common Definitions
# ----------------------------
# Battery characteristics for the Panasonic NCR18650B
battery_params = {
    'R0_discharge': 0.0019,   # Ohms
    'R1_discharge': 0.0017,   # Ohms
    'C1_discharge': 5.5984e3, # Farads
    'R2_discharge': 0.0139,   # Ohms
    'C2_discharge': 352.253,  # Farads

    'R0_charge': 0.0019,      # Ohms
    'R1_charge': 0.0017,      # Ohms
    'C1_charge': 5.5984e3,    # Farads
    'R2_charge': 0.0139,      # Ohms
    'C2_charge': 352.253,     # Farads

    'delta_t': 0.4,           # *internal* time step for sub-stepping
    'Q_nominal': 2.7,       # Ah
    'Nominal_V': 3.6,         # V
    'initial_SOC': 1.0,       # fully charged
    'Max_discharge_current': 9,  # A
    'Max_charge_current': 1.625,   # A
    'Cell_weight': 46.5e-3,   # kg
    'Cell_volume': 1.76e-5,   # m^3
```

```
    'soc_table': np.array([0.01, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0]),
    'voc_table': np.array([2.57, 3.13, 3.2575, 3.2825, 3.445, 3.5075, 3.6325, 3.7575,
    3.78875, 3.945, 4.0]),
}

# Operating conditions for the fuel cell (only used when the fuel cell is active)
test_vector = {
    "T": 353.15,    # Kelvin
    "PH2": 2.5,     # atm
    "PO2": 0.518,   # atm
    "i-start": 0,   # Ampere
    "i-stop": 1.0, # Ampere
    "i-step": 0.001, # Ampere
    "A": 1,         # cm^2
    "l": 0.08,      # cm
    "lambda": 14,   # -
    "N": 1,         # -
    "R": 0,         # Ohm
    "JMax": 1,      # A/cm^2
    "Name": "Amphlett_Test"
}

# A helper function to scale normalized parameters back to their physical ranges.
def scale_to_original_range(normalized, min_val, max_val):
    return min_val + normalized * (max_val - min_val)

# ----------------------------
# Merged Objective Function
# ----------------------------
def objective_function(params):
    # The design vector: [normalized_fc_power, normalized_voltage_fc, normalized_voltage_bat]
    norm_fc = params[0]
    norm_voltage_fc = params[1]
    norm_voltage_bat = params[2]

    if norm_fc < 1e-6:
        # ----------------------------
        # Battery-Only Scheme
        # ----------------------------
        desired_voltage_bat = scale_to_original_range(norm_voltage_bat, 100, 2500)
        fuel_cell_max_power_arg = 206.6  # unused in battery-only

        # Read flight data
        timeF, Zh, Pw_r, TimeP, Pw_P, Pw_fuel_cell_P, Pw_battery_P, \
        flight_condition, v_x, v_z, flight_data = \
            fr.read_data_fc('Flight_Data.csv', fuel_cell_max_power=fuel_cell_max_power_arg)
        _, _, _, _, _, flight_condition_ts, Zh_ts, v_x_ts, v_z_ts, _ = \
            fr.read_data_fs('Flight_Data.csv')

        PE_converter_efficiency = 0.98
        PE_converter_pd = 7500  # W/kg

        inverter_power = np.max((Pw_P * 1000) / PE_converter_efficiency)
        battery_power = np.max(Pw_P) * 1000 / (PE_converter_efficiency ** 2)
        bat_power = Pw_P / (PE_converter_efficiency ** 2)
        dc_converter_power = battery_power

        # Battery energy demand (Wh)
        battery_energy_demand = fr.calculate_energy_demand(TimeP, bat_power)/0.90
        print("Battery-only: Battery energy demand =", battery_energy_demand, "Wh")

        # Battery sizing
        num_series, num_parallel = bm.calculate_cells_based_on_power_energy_and_current(
            battery_power, battery_energy_demand, desired_voltage_bat, battery_params)
        battery_weight = num_series * num_parallel * battery_params['Cell_weight'] / 0.75

        bat_dc_weight = dc_converter_power / PE_converter_pd
        inverter_weight = inverter_power / PE_converter_pd
        PE_converter_weight = inverter_weight + bat_dc_weight
        structure = 1905  # kg
```

```
        hps_weight = battery_weight + PE_converter_weight + structure
        return hps_weight

    else:
        # ---------------------------
        # Fuel Cell / Battery Scheme
        # ---------------------------
        desired_voltage_fc = scale_to_original_range(norm_voltage_fc, 100, 2500)
        desired_voltage_bat = scale_to_original_range(norm_voltage_bat, 100, 2500)
        fuel_cell_max_power = scale_to_original_range(norm_fc, 0, 400)

        # Read flight data
        timeF, Zh, Pw_r, TimeP, Pw_P, Pw_fuel_cell_P, Pw_battery_P, \
        flight_condition, v_x, v_z, flight_data = \
            fr.read_data_fc('Flight_Data.csv', fuel_cell_max_power=fuel_cell_max_power)
        _, _, _, _, _, flight_condition_ts, Zh_ts, v_x_ts, v_z_ts, _ = \
            fr.read_data_fs('Flight_Data.csv')

        PE_converter_efficiency = 0.98

        # Fuel cell power
        peak_power_fc = np.max(Pw_fuel_cell_P) * 1000 / (PE_converter_efficiency ** 2)
        fc_converter_power = peak_power_fc
        inverter_power = np.max((Pw_P * 1000) / PE_converter_efficiency)

        # Fuel cell sizing
        sizing_results = fc.size_fuel_cell(desired_voltage_fc, peak_power_fc, test_vector,
    flight_data)
        fuel_cell_stacks = sizing_results['The number of cells in series']
        fuel_cell_area = sizing_results['Optimal Area (cm^2)']
        fuel_cell_mass = sizing_results['Lowest Mass (kg)']

        # Off-design simulation
        segments = ["Idle", "Take Off", "Ascend", "Approach", "Cruise",
                    "Take Down", "Idle", "Take Off R", "Cruise R", "Take Down R"]
        off_design_results = {}
        for seg in segments:
            results = fc.simulate_off_design(fuel_cell_area, fuel_cell_stacks, test_vector,
    flight_data, seg)
            off_design_results[seg] = results

        effective_power_detailed = np.array([
            0 if seg == "Idle" else off_design_results.get(seg, {}).get("Best Effective Power
     (W)", 0)
            for seg in flight_condition_ts
        ])
        hydrogen_consumption_detailed = np.array([
            0 if seg == "Idle" else off_design_results.get(seg, {}).get("Best Hydrogen Used (
    kg/s)", 0)
            for seg in flight_condition_ts
        ])

        power_deficiency = (np.array(Pw_P) * 1000 - effective_power_detailed)
        power_deficiency_detailed = np.maximum(power_deficiency, 0)
        fc_fuel_used = np.sum(hydrogen_consumption_detailed)

        # Battery energy demand (Wh)
        battery_energy_demand = fr.calculate_energy_demand_positive(
            TimeP, power_deficiency / (1000 * (PE_converter_efficiency ** 2))/0.95
        )
        print("Fuel cell active: Battery energy demand =", battery_energy_demand, "Wh")

        # Battery sizing
        battery_power_demand = np.max(power_deficiency_detailed / (PE_converter_efficiency **
     2))
        bat_converter_power = battery_power_demand
        num_series, num_parallel = bm.calculate_cells_based_on_power_energy_and_current(
            battery_power_demand, battery_energy_demand, desired_voltage_bat, battery_params)
        battery_weight = num_series * num_parallel * battery_params['Cell_weight'] / 0.75

        # Compressor & cooling power
```

```
        cruise_powers = fc.simulate_off_design(fuel_cell_area, fuel_cell_stacks, test_vector,
    flight_data, 'Cruise')
        compressor_power = cruise_powers['Best Compressor Power (W)']
        hex_power = cruise_powers['Best Cooling Power (W)']

        hps_weight = weight_hps_bat(fuel_cell_mass, fc_fuel_used, battery_weight,
                                    compressor_power, hex_power,
                                    fc_converter_power, inverter_power, bat_converter_power)
        return hps_weight


# ----------------------------
# Optimization via Differential Evolution
# ----------------------------
initial_guess = [0.5, 0.5, 0.5]
bounds = [(0, 1), (0, 1), (0, 1)]
result = differential_evolution(objective_function, bounds, maxiter=10000, popsize=50)

# Recover physical parameters
opt_fc_norm, opt_fc_voltage_norm, opt_bat_voltage_norm = result.x
opt_fuel_cell_max_power = scale_to_original_range(opt_fc_norm, 0, 400)  # kW
opt_fc_voltage = scale_to_original_range(opt_fc_voltage_norm, 100, 2500)  # V
opt_bat_voltage = scale_to_original_range(opt_bat_voltage_norm, 100, 2500)  # V

print("Optimal fuel cell max power:", opt_fuel_cell_max_power, "kW")
print("Optimal fuel cell voltage:", opt_fc_voltage, "V")
print("Optimal battery voltage:", opt_bat_voltage, "V")
print("Minimum eVTOL weight:", result.fun, "kg")

# ----------------------------
# Additional Performance Simulation
# ----------------------------
if opt_fuel_cell_max_power < 1e-6:
    # ----------------------------
    # Battery-Only Mode
    # ----------------------------
    print("\n--- Performance Simulation: Battery-Only Scheme ---")
    desired_voltage_bat = opt_bat_voltage
    fuel_cell_power_arg = 206.6  # dummy for flight data
    timeF, Zh, Pw_r, TimeP, Pw_P, Pw_fuel_cell_P, Pw_battery_P, \
    flight_condition, v_x, v_z, flight_data = \
        fr.read_data_fc('Flight_Data.csv', fuel_cell_max_power=fuel_cell_power_arg)
    _, _, _, _, _, flight_condition_ts, Zh_ts, v_x_ts, v_z_ts, _ = fr.read_data_fs('
    Flight_Data.csv')

    PE_converter_efficiency = 0.98
    PE_converter_pd = 7500  # W/kg

    inverter_power = np.max((Pw_P * 1000) / PE_converter_efficiency)
    battery_power = np.max(Pw_P) * 1000 / (PE_converter_efficiency ** 2)
    bat_power = Pw_P / (PE_converter_efficiency ** 2)
    dc_converter_power = np.max(bat_power)
    battery_energy_demand = fr.calculate_energy_demand(TimeP, bat_power)
    print("Battery-only simulation: Battery energy demand =", battery_energy_demand, "Wh")

    # Battery sizing
    num_series, num_parallel = bm.calculate_cells_based_on_power_energy_and_current(
        battery_power, battery_energy_demand, desired_voltage_bat, battery_params)
    battery_weight = num_series * num_parallel * battery_params['Cell_weight'] / 0.75

    # For plotting & final results
    structure = 1905  # kg
    bat_dc_weight = dc_converter_power / PE_converter_pd
    inverter_weight = inverter_power / PE_converter_pd
    PE_converter_weight = inverter_weight + bat_dc_weight
    hps_weight = battery_weight + PE_converter_weight + structure
    print("Aircraft OEW:", hps_weight, "kg")

    weight_breakdown_data = weight_breakdown_bat(battery_weight, dc_converter_power,
    inverter_power)
```

```
    plot_weight_breakdown(weight_breakdown_data, "The weight breakdown of the BAT
    architecture")

    battery_energy_density = battery_energy_demand / battery_weight
    battery_power_density = battery_power * 1000 / battery_weight
    print("\n--- Performance Metrics ---")
    print("Battery Energy Density (Wh/kg):", battery_energy_density)
    print("Battery Power Density (W/kg):", battery_power_density)

    time = TimeP
    power_demand = (Pw_P * 1000) / (PE_converter_efficiency ** 2)

    # Initial battery states
    SOC = battery_params['initial_SOC']
    V1, V2 = 0.0, 0.0

    SOC_list = [SOC]
    voltage_list = []
    efficiency_list = []
    current_list = []

    for i in range(1, len(time)):
        # Time step from flight data
        dt_flight = time[i] - time[i - 1]

        P_demand = power_demand[i]
        # Estimate pack current
        V_oc = bm.ocv_soc_relationship(SOC, battery_params) * num_series
        Pack_current = P_demand / V_oc if V_oc > 1e-4 else 0
        I = Pack_current / num_parallel

        # --- Sub-stepping ---
        sub_steps = int(np.ceil(dt_flight / battery_params['delta_t']))
        sub_dt = dt_flight / sub_steps
        for _ in range(sub_steps):
            V_terminal, V1, V2, SOC = bm.thevenin_2nd_order_model(
                I, battery_params, sub_dt, V1, V2, SOC, battery_params['Q_nominal']
            )

        # After sub-stepping, compute efficiency at final state
        total_internal_res = (battery_params['R0_discharge'] +
                              battery_params['R1_discharge'] +
                              battery_params['R2_discharge'])
        total_pack_res = total_internal_res * num_series / num_parallel
        final_voltage = V_terminal * num_series
        power_loss = Pack_current ** 2 * total_pack_res
        output_power = final_voltage * Pack_current
        eff = (output_power / (output_power + power_loss)) * 100 if (output_power +
    power_loss) > 0 else 0

        # Store results
        SOC_list.append(SOC)
        voltage_list.append(V_terminal)
        efficiency_list.append(eff)
        current_list.append(Pack_current)

    # Plotting
    plt.figure(figsize=(10, 12))

    # SOC
    plt.subplot(4, 1, 1)
    plt.plot(time, SOC_list, label='State of Charge (-)', color='blue')
    plt.xlabel('Time (s)')
    plt.ylabel('SOC (-)')
    plt.title('Battery State of Charge Over Mission')
    plt.grid(True)

    # Terminal Voltage
    plt.subplot(4, 1, 2)
    plt.plot(time[1:], voltage_list, label='Terminal Voltage (V)', color='green')
    plt.xlabel('Time (s)')
```

```
    plt.ylabel('Voltage (V)')
    plt.title('Battery Terminal Voltage Over Mission')
    plt.grid(True)

    # Efficiency
    plt.subplot(4, 1, 3)
    plt.plot(time[1:], efficiency_list, label='Efficiency (%)', color='orange')
    plt.xlabel('Time (s)')
    plt.ylabel('Efficiency (%)')
    plt.title('Battery Efficiency Over Mission')
    plt.grid(True)

    # Current
    plt.subplot(4, 1, 4)
    plt.plot(time[1:], current_list, label='Pack Current (A)', color='red')
    plt.xlabel('Time (s)')
    plt.ylabel('Current (A)')
    plt.title('Battery Pack Current Over Mission')
    plt.grid(True)

    plt.tight_layout()
    plt.show()

else:
    # ---------------------------
    # Fuel Cell / Battery Mode
    # ---------------------------
    print("\n--- Performance Simulation: Fuel Cell / Battery Scheme ---")

    desired_voltage_fc = opt_fc_voltage
    desired_voltage_bat = opt_bat_voltage
    fuel_cell_max_power = opt_fuel_cell_max_power

    print("Simulating performance for optimized design:")
    print("Fuel Cell Max Power:", fuel_cell_max_power, "kW")
    print("Desired Fuel Cell Voltage:", desired_voltage_fc, "V")
    print("Desired Battery Voltage:", desired_voltage_bat, "V")

    timeF, Zh, Pw_r, TimeP, Pw_P, Pw_fuel_cell_P, Pw_battery_P, \
    flight_condition, v_x, v_z, flight_data = \
        fr.read_data_fc('Flight_Data.csv', fuel_cell_max_power=fuel_cell_max_power)
    _, _, _, _, _, flight_condition_ts, Zh_ts, v_x_ts, v_z_ts, _ = fr.read_data_fs('
    Flight_Data.csv')

    PE_converter_efficiency = 0.98
    peak_power_fc = np.max(Pw_fuel_cell_P) * 1000 / (PE_converter_efficiency ** 2)
    fc_converter_power = peak_power_fc
    inverter_power = np.max((Pw_P * 1000) / PE_converter_efficiency)

    # Fuel cell sizing
    sizing_results = fc.size_fuel_cell(desired_voltage_fc, peak_power_fc, test_vector,
    flight_data)
    fuel_cell_stacks = sizing_results['The number of cells in series']
    fuel_cell_area = sizing_results['Optimal Area (cm^2)']
    fuel_cell_mass = sizing_results['Lowest Mass (kg)']

    print("\nFuel Cell Sizing Results:")
    print(" - Number of cells in series:", fuel_cell_stacks)
    print(" - Optimal Area (cm^2):", fuel_cell_area)
    print(" - Fuel Cell Mass (kg):", fuel_cell_mass)

    segments = ["Idle", "Take Off", "Ascend", "Approach", "Cruise",
                "Take Down", "Idle", "Take Off R", "Cruise R", "Take Down R"]
    off_design_results = {}
    for segment in segments:
        results = fc.simulate_off_design(fuel_cell_area, fuel_cell_stacks, test_vector,
    flight_data, segment)
        off_design_results[segment] = results

    effective_power_detailed = np.array([
```

```
        0 if seg == "Idle" else off_design_results.get(seg, {}).get("Best Effective Power (W)
    ", 0)
375         for seg in flight_condition_ts
    ])
377     hydrogen_consumption_detailed = np.array([
        0 if seg == "Idle" else off_design_results.get(seg, {}).get("Best Hydrogen Used (kg/s
    )", 0)
379         for seg in flight_condition_ts
    ])
381
    power_deficiency = (np.array(Pw_P) * 1000 - effective_power_detailed)
383 power_deficiency_detailed = np.maximum(power_deficiency, 0)
    fc_fuel_used = np.sum(hydrogen_consumption_detailed)
385
    # Battery demand
387 battery_energy_demand = fr.calculate_energy_demand_positive(
        TimeP, power_deficiency / (1000 * PE_converter_efficiency ** 2)
389 )
    print("\nBattery Energy Demand:", battery_energy_demand, "Wh")
391
    battery_power_demand = np.max(power_deficiency_detailed / (PE_converter_efficiency ** 2))
393 bat_converter_power = battery_power_demand
    num_series, num_parallel = bm.calculate_cells_based_on_power_energy_and_current(
395     battery_power_demand, battery_energy_demand, desired_voltage_bat, battery_params)
    battery_weight = num_series * num_parallel * battery_params['Cell_weight'] / 0.75
397 print("Battery Weight (kg):", battery_weight)

399 # Compressor/cooling
    cruise_powers = fc.simulate_off_design(fuel_cell_area, fuel_cell_stacks, test_vector,
    flight_data, 'Cruise')
401 compressor_power = cruise_powers['Best Compressor Power (W)']
    hex_power = cruise_powers['Best Cooling Power (W)']
403
    hps_weight = weight_hps_bat(fuel_cell_mass, fc_fuel_used, battery_weight,
405                             compressor_power, hex_power,
                                fc_converter_power, inverter_power, bat_converter_power)
407 print("Aircraft OEW:", hps_weight, "kg")

409 fuel_cell_system_mass = weight_fc_system(fuel_cell_mass, compressor_power, hex_power,
    fc_fuel_used)
    fuel_cell_power_density = peak_power_fc / fuel_cell_system_mass
411
    weight_breakdown_data = weight_hps_bat_breakdown(
413     fuel_cell_mass, fc_fuel_used, battery_weight,
        compressor_power, hex_power,
415     fc_converter_power, inverter_power, bat_converter_power
    )
417
    battery_energy_density = battery_energy_demand / battery_weight
419 battery_power_density = battery_power_demand / battery_weight

421 print("\n--- Performance Metrics ---")
    print("Fuel Cell System Mass (kg):", fuel_cell_system_mass)
423 print("Fuel Cell Power Density (W/kg):", fuel_cell_power_density)
    print("Battery Energy Density (Wh/kg):", battery_energy_density)
425 print("Battery Power Density (W/kg):", battery_power_density)

427 time = TimeP
    power_demand = power_deficiency
429
    # Initial battery states
431 SOC = battery_params['initial_SOC']
    V1, V2 = 0.0, 0.0
433
    SOC_list = [SOC]
435 voltage_list = []
    efficiency_list = []
437 current_list = []

439 for i in range(1, len(time)):
        dt_flight = time[i] - time[i - 1]
```

```
441
        P_demand = power_demand[i]
443     V_oc = bm.ocv_soc_relationship(SOC, battery_params) * num_series
        Pack_current = P_demand / V_oc if V_oc > 1e-4 else 0
445     I = Pack_current / num_parallel

447     # --- Sub-stepping ---
        sub_steps = int(np.ceil(dt_flight / battery_params['delta_t']))
449     sub_dt = dt_flight / sub_steps
        for _ in range(sub_steps):
451         V_terminal, V1, V2, SOC = bm.thevenin_2nd_order_model(
                I, battery_params, sub_dt, V1, V2, SOC, battery_params['Q_nominal']
453         )

455     # Efficiency at final state
        total_internal_res = (battery_params['R0_discharge'] +
457                           battery_params['R1_discharge'] +
                           battery_params['R2_discharge'])
459     total_pack_res = total_internal_res * num_series / num_parallel
        final_voltage = V_terminal * num_series
461     power_loss = Pack_current ** 2 * total_pack_res
        output_power = final_voltage * Pack_current
463     eff = (output_power / (output_power + power_loss)) * 100 if (output_power +
    power_loss) > 0 else 0

465     SOC_list.append(SOC)
        voltage_list.append(V_terminal)
467     efficiency_list.append(eff)
        current_list.append(Pack_current)
469
    # Final plots
471 plt.figure(figsize=(10, 12))

473 # SOC
    plt.subplot(4, 1, 1)
475 plt.plot(time, SOC_list, label='State of Charge (-)', color='blue')
    plt.xlabel('Time (s)')
477 plt.ylabel('SOC (-)')
    plt.title('Battery State of Charge Over Mission')
479 plt.grid(True)

481 # Terminal Voltage
    plt.subplot(4, 1, 2)
483 plt.plot(time[1:], voltage_list, label='Terminal Voltage (V)', color='green')
    plt.xlabel('Time (s)')
485 plt.ylabel('Voltage (V)')
    plt.title('Battery Terminal Voltage Over Mission')
487 plt.grid(True)

489 # Efficiency
    plt.subplot(4, 1, 3)
491 plt.plot(time[1:], efficiency_list, label='Efficiency (%)', color='orange')
    plt.xlabel('Time (s)')
493 plt.ylabel('Efficiency (%)')
    plt.title('Battery Efficiency Over Mission')
495 plt.grid(True)

497 # Current
    plt.subplot(4, 1, 4)
499 plt.plot(time[1:], current_list, label='Pack Current (A)', color='red')
    plt.xlabel('Time (s)')
501 plt.ylabel('Current (A)')
    plt.title('Battery Pack Current Over Mission')
503 plt.grid(True)

505 plt.tight_layout()
    plt.show()
507
    plot_weight_breakdown(weight_breakdown_data, 'The weight breakdown of the FC-BAT
    architecture')
```

Perf_calc_tool.py

C.3. BATTERY MODEL

```
import math
import numpy as np
from scipy.interpolate import CubicSpline

# Define SOC and VOC from the table you provided
soc_table = np.array([0.01, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0])
voc_table = np.array([2.5, 3.06, 3.1875, 3.2125, 3.375, 3.4375, 3.5625, 3.6875, 3.71875,
    3.875, 4.0626])

soc_table_secondary = np.array([0.01, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0])
voc_table_secondary = np.array([2.5, 3.34, 3.55, 3.60, 3.62, 3.66, 3.75, 3.84, 3.94, 4.04,
    4.17])

# Create a cubic spline interpolator
ocv_spline = CubicSpline(soc_table, voc_table)


def ocv_soc_relationship(SOC, battery_params):
    """
    Returns the Open Circuit Voltage (OCV) for a given State of Charge (SOC),
    using the SOC-OCV relationship stored in the battery dictionary.

    Parameters:
    SOC (float): State of Charge (between 0 and 1).
    battery (dict): Battery dictionary containing SOC-OCV lookup tables.

    Returns:
    float: Open Circuit Voltage (OCV).
    """
    if SOC < 0 or SOC > 1:
        raise ValueError("SOC must be between 0 and 1")

    # Extract SOC-OCV tables from the battery dictionary
    soc_table = battery_params['soc_table']
    voc_table = battery_params['voc_table']

    # Create a cubic spline interpolator
    ocv_spline = CubicSpline(soc_table, voc_table)
    return ocv_spline(SOC)


def thevenin_2nd_order_model(I, params, delta_t, V1_prev, V2_prev, SOC_prev, Q_nominal):
    """
    Computes the terminal voltage and updated internal states for a given current.

    Parameters:
    I (float): Current in Amperes.
    params (dict): Dictionary containing battery parameters.
    delta_t (float): Time step in seconds.
    V1_prev (float): Previous value of V1.
    V2_prev (float): Previous value of V2.
    SOC_prev (float): Previous State of Charge.
    Q_nominal (float): Nominal capacity in Ah.

    Returns:
    tuple: Terminal voltage, next V1, next V2, next SOC.
    """
    if I >= 0:  # Discharging
        R0 = params['R0_discharge']
        R1 = params['R1_discharge']
        C1 = params['C1_discharge']
        R2 = params['R2_discharge']
        C2 = params['C2_discharge']
    else:  # Charging
```

```
        R0 = params['R0_charge']
        R1 = params['R1_charge']
        C1 = params['C1_charge']
        R2 = params['R2_charge']
        C2 = params['C2_charge']
        charging_efficiency = 0.95
        I = I * charging_efficiency  # Apply charging efficiency (less effective current for SOC increase)

        # Limit charging current to max charge current
        max_charge_current = params['Max_charge_current']
        if abs(I) > max_charge_current:
            I = -max_charge_current

    V_oc = ocv_soc_relationship(SOC_prev, params)
    V1_next = V1_prev + delta_t * (-V1_prev / (R1 * C1) + I / C1)
    V2_next = V2_prev + delta_t * (-V2_prev / (R2 * C2) + I / C2)
    V_terminal = V_oc - R0 * I - V1_next - V2_next

    SOC_next = SOC_prev - (I * delta_t) / (Q_nominal * 3600)  # Convert Q_nominal to Coulombs
    SOC_next = max(0, min(1, SOC_next))

    return V_terminal, V1_next, V2_next, SOC_next

# def thevenin_2nd_order_model(I, params, delta_t, V1_prev, V2_prev, SOC_prev, Q_nominal):
#     """
#     Computes the terminal voltage and updated internal states for a given current,
#     using backward Euler for the RC state updates.
#
#     Parameters:
#     I (float): Current in Amperes (positive for discharge, negative for charge).
#     params (dict): Dictionary containing battery parameters:
#                    {
#                        'R0_discharge': ...,
#                        'R1_discharge': ...,
#                        'C1_discharge': ...,
#                        'R2_discharge': ...,
#                        'C2_discharge': ...,
#                        'R0_charge': ...,
#                        'R1_charge': ...,
#                        'C1_charge': ...,
#                        'R2_charge': ...,
#                        'C2_charge': ...,
#                        'Max_charge_current': ...,
#                        ... (other parameters like OCV mapping) ...
#                    }
#     delta_t (float): Time step in seconds.
#     V1_prev (float): Previous voltage on the first RC branch (V).
#     V2_prev (float): Previous voltage on the second RC branch (V).
#     SOC_prev (float): Previous State of Charge (0.0 to 1.0).
#     Q_nominal (float): Nominal capacity in Ah (ampere-hours).
#
#     Returns:
#     tuple: (V_terminal, V1_next, V2_next, SOC_next)
#            V_terminal: Terminal voltage (float)
#            V1_next: Updated voltage on the first RC branch (float)
#            V2_next: Updated voltage on the second RC branch (float)
#            SOC_next: Updated State of Charge (float, clipped to [0,1])
#     """
#     # Select the parameter set depending on charging or discharging
#     if I >= 0:  # Discharging
#         R0 = params['R0_discharge']
#         R1 = params['R1_discharge']
#         C1 = params['C1_discharge']
#         R2 = params['R2_discharge']
#         C2 = params['C2_discharge']
#     else:  # Charging
#         R0 = params['R0_charge']
#         R1 = params['R1_charge']
#         C1 = params['C1_charge']
#         R2 = params['R2_charge']
```

```
#         C2 = params['C2_charge']
#
#         # Apply charging efficiency
#         charging_efficiency = 0.95
#         I = I * charging_efficiency
#
#         # Enforce maximum charging current
#         max_charge_current = params['Max_charge_current']
#         if abs(I) > max_charge_current:
#             I = -max_charge_current  # negative sign for charging
#
#     # Get open-circuit voltage (depends on SOC)
#     V_oc = ocv_soc_relationship(SOC_prev, params)
#
#     # --- Backward Euler Updates ---
#     #
#     # For dV1/dt = -V1/(R1*C1) + I/C1,
#     # the backward Euler formula gives:
#     #   V1_next = [ V1_prev + delta_t*(I/C1) ] / [ 1 + (delta_t)/(R1*C1) ]
#     #
#     # Similarly for V2.
#
#     denom1 = 1.0 + (delta_t / (R1 * C1))
#     V1_next = (V1_prev + delta_t * (I / C1)) / denom1
#
#     denom2 = 1.0 + (delta_t / (R2 * C2))
#     V2_next = (V2_prev + delta_t * (I / C2)) / denom2
#
#     # Compute terminal voltage
#     V_terminal = V_oc - R0 * I - V1_next - V2_next
#
#     # Update SOC (basic coulomb counting)
#     # Note: Q_nominal is in [Ah], so multiply by 3600 to convert to [Coulombs].
#     SOC_next = SOC_prev - (I * delta_t) / (Q_nominal * 3600.0)
#     # Ensure SOC stays in [0, 1]
#     SOC_next = max(0.0, min(1.0, SOC_next))
#
#     return V_terminal, V1_next, V2_next, SOC_next


def calculate_number_of_cells(desired_voltage, battery_params, load_current, num_iterations
    =100):
    """
    Calculates the number of cells needed in series and parallel to achieve the desired
    voltage and current.

    Parameters:
    desired_voltage (float): Desired voltage for the battery pack.
    battery_params (dict): Dictionary containing battery parameters.
    load_current (float): Desired load current.
    num_iterations (int): Number of iterations for convergence.

    Returns:
    tuple: Number of cells in series, number of cells in parallel.
    """
    max_cell_current = battery_params['Max_discharge_current']
    num_parallel = math.ceil(load_current / max_cell_current)

    cell_current = load_current / num_parallel
    nominal_voltage = estimate_nominal_voltage(battery_params, cell_current, num_iterations)
    num_series = math.ceil(desired_voltage / nominal_voltage)

    return num_series, num_parallel


def estimate_nominal_voltage(battery_params, cell_current):
    """
    Estimates the nominal voltage of one battery cell using the Thevenin model.

    Parameters:
    battery_params (dict): Dictionary containing battery parameters.
```

```
    cell_current (float): Current per cell.
    num_iterations (int): Number of iterations for convergence.

    Returns:
    float: Estimated nominal voltage.
    """
    V1, V2 = 0.0, 0.0
    SOC = battery_params['initial_SOC']
    V_terminal, V1, V2, SOC = thevenin_2nd_order_model(
        cell_current, battery_params, battery_params['delta_t'], V1, V2, SOC, battery_params['Q_nominal']
    )
    return V_terminal


def simulate_efficiency_curve(battery_params, current_range, num_iterations):
    """
    Estimates the terminal voltage and efficiency for various currents.

    Parameters:
    battery_params (dict): Dictionary containing battery parameters.
    current_range (array-like): Range of currents to simulate.
    num_iterations (int): Number of iterations for convergence.

    Returns:
    tuple: List of terminal voltages, list of efficiencies.
    """
    voltages = []
    efficiencies = []
    for I in current_range:
        V1, V2 = 0.0, 0.0
        SOC = battery_params['initial_SOC']
        for _ in range(num_iterations):
            V_terminal, V1, V2, SOC = thevenin_2nd_order_model(
                I, battery_params, battery_params['delta_t'], V1, V2, SOC, battery_params['Q_nominal']
            )
        voltages.append(V_terminal)

        # Calculate power loss and efficiency
        total_internal_resistance = battery_params['R0_discharge'] + battery_params['R1_discharge'] + battery_params[
            'R2_discharge']
        power_loss = I ** 2 * total_internal_resistance
        output_power = V_terminal * I
        efficiency = (output_power / (output_power + power_loss)) * 100 if output_power + power_loss > 0 else 0
        efficiencies.append(efficiency)

    return voltages, efficiencies


def simulate_pack_efficiency_curve(battery_params, pack_current_range, num_iterations, num_series, num_parallel):
    """
    Simulates the efficiency for a battery pack.

    Parameters:
    battery_params (dict): Dictionary containing battery parameters.
    pack_current_range (array-like): Range of currents for the battery pack.
    num_iterations (int): Number of iterations for convergence.
    num_series (int): Number of cells in series.
    num_parallel (int): Number of cells in parallel.

    Returns:
    tuple: List of terminal voltages, list of efficiencies.
    """
    voltages = []
    efficiencies = []
    for I_pack in pack_current_range:
        I_cell = I_pack / num_parallel  # Current per cell
```

```
        V1, V2 = 0.0, 0.0
        SOC = battery_params['initial_SOC']
        for _ in range(num_iterations):
            V_terminal_cell, V1, V2, SOC = thevenin_2nd_order_model(
                I_cell, battery_params, battery_params['delta_t'], V1, V2, SOC,
    battery_params['Q_nominal']
            )
        V_terminal_pack = V_terminal_cell * num_series  # Pack voltage
        voltages.append(V_terminal_pack)

        # Calculate total internal resistance for the whole pack
        total_internal_resistance_per_cell = (
                battery_params['R0_discharge'] +
                battery_params['R1_discharge'] +
                battery_params['R2_discharge']
        )
        total_internal_resistance_series = num_series * total_internal_resistance_per_cell
        total_internal_resistance_pack = total_internal_resistance_series / num_parallel

        # Calculate power loss and efficiency for the whole pack
        power_loss = I_pack ** 2 * total_internal_resistance_pack
        output_power = V_terminal_pack * I_pack
        efficiency = (output_power / (output_power + power_loss)) * 100 if output_power +
    power_loss > 0 else 0
        efficiencies.append(efficiency)

    return voltages, efficiencies


def simulate_polarization_curve(battery_params, current_range, num_iterations):
    """
    Simulates the polarization curve for a single cell.

    Parameters:
    battery_params (dict): Dictionary containing battery parameters.
    current_range (array-like): Range of currents to simulate.
    num_iterations (int): Number of iterations for convergence.

    Returns:
    list: List of terminal voltages.
    """
    voltages = []
    for I in current_range:
        V1, V2 = 0.0, 0.0
        SOC = battery_params['initial_SOC']
        for _ in range(num_iterations):
            V_terminal, V1, V2, SOC = thevenin_2nd_order_model(
                I, battery_params, battery_params['delta_t'], V1, V2, SOC, battery_params['
    Q_nominal']
            )
        voltages.append(V_terminal)
    return voltages


def simulate_pack_polarization_curve(battery_params, pack_current_range, num_iterations,
    num_series, num_parallel):
    """
    Simulates the polarization curve for a battery pack.

    Parameters:
    battery_params (dict): Dictionary containing battery parameters.
    pack_current_range (array-like): Range of currents for the battery pack.
    num_iterations (int): Number of iterations for convergence.
    num_series (int): Number of cells in series.
    num_parallel (int): Number of cells in parallel.

    Returns:
    list: List of terminal voltages for the pack.
    """
    voltages = []
    for I_pack in pack_current_range:
```

```
        I_cell = I_pack / num_parallel  # Current per cell
        V1, V2 = 0.0, 0.0
        SOC = battery_params['initial_SOC']
        for _ in range(num_iterations):
            V_terminal_cell, V1, V2, SOC = thevenin_2nd_order_model(
                I_cell, battery_params, battery_params['delta_t'], V1, V2, SOC,
    battery_params['Q_nominal']
            )
        V_terminal_pack = V_terminal_cell * num_series  # Pack voltage
        voltages.append(V_terminal_pack)
    return voltages


def simulate_power_density_curve(battery_params, current_range, num_iterations,
    battery_volume):
    """
    Calculates the power density curve for a single cell.

    Parameters:
    battery_params (dict): Dictionary containing battery parameters.
    current_range (array-like): Range of currents to simulate.
    num_iterations (int): Number of iterations for convergence.
    battery_volume (float): Volume of a single battery cell.

    Returns:
    list: List of power densities.
    """
    voltages = simulate_polarization_curve(battery_params, current_range, num_iterations)
    power_density = [(V * I) / battery_volume for V, I in zip(voltages, current_range)]
    return power_density


def simulate_pack_power_density_curve(battery_params, pack_current_range, num_iterations,
    num_series, num_parallel,
                                      cell_volume):
    """
    Calculates the power density curve for a battery pack.

    Parameters:
    battery_params (dict): Dictionary containing battery parameters.
    pack_current_range (array-like): Range of currents for the battery pack.
    num_iterations (int): Number of iterations for convergence.
    num_series (int): Number of cells in series.
    num_parallel (int): Number of cells in parallel.
    cell_volume (float): Volume of a single battery cell.

    Returns:
    list: List of power densities for the pack.
    """
    voltages = simulate_pack_polarization_curve(battery_params, pack_current_range,
    num_iterations, num_series,
                                                num_parallel)
    pack_volume = cell_volume * num_series * num_parallel  # Total pack volume
    power_density = [(V * I_pack) / pack_volume for V, I_pack in zip(voltages,
    pack_current_range)]
    return power_density


def calculate_cells_based_on_power_energy_and_current(peak_power, total_energy,
    desired_voltage, battery_params):
    """
    Calculates the number of cells required based on peak power, energy requirements, and max
     discharge current.

    Parameters:
    peak_power (float): Peak power requirement in Watts.
    total_energy (float): Total energy requirement in Wh.
    battery_params (dict): Dictionary containing battery parameters.
    num_iterations (int): Number of iterations for convergence.

    Returns:
```

```
    tuple: Number of cells in series, number of cells in parallel.
    """
    # Step 1: Calculate the number of cells in series to meet voltage requirements for peak power
    nominal_voltage = estimate_nominal_voltage(battery_params, battery_params['Max_discharge_current'])
    # print('the nominal voltage of 1 cell is ', nominal_voltage, ' V')

    # Calculate the number of cells required in series to meet the voltage requirements
    num_series = math.ceil(desired_voltage / nominal_voltage)

    # Step 2: Calculate the number of cells in parallel to meet energy requirements
    # Convert nominal capacity from Ah to Wh by multiplying by nominal voltage
    cell_energy_capacity = battery_params['Q_nominal'] * nominal_voltage  # in Wh
    # print('The cell energy capacity', cell_energy_capacity, ' Wh')

    # Calculate the number of cells required in parallel to meet the total energy requirement
    num_parallel_energy = math.ceil(total_energy / (cell_energy_capacity * num_series))
    print('the number parallel  based on energy is', num_parallel_energy)

    # Step 3: Calculate the number of cells in parallel to meet the maximum discharge current requirement
    # Ensure that the current per cell does not exceed the maximum allowable discharge current
    num_parallel_current = math.ceil(peak_power / (nominal_voltage * num_series * battery_params['Max_discharge_current']))
    print('the number parallel  based on current is', num_parallel_current)

    # Take the maximum of both parallel requirements to ensure both energy and current demands are met
    num_parallel = max(num_parallel_energy, num_parallel_current)

    return num_series, num_parallel



def hybrid_configurable_battery_pack(
    peak_power, voltage_required, total_energy,
    num_series_primary, num_parallel_primary,
    battery_primary, battery_secondary
):
    """
    Calculates the configuration of a hybrid configurable battery pack to meet power, voltage, and energy requirements.
    """

    # Step 1: Calculate the primary pack contributions
    # Step 1 calculate the nominal voltages

    nominal_voltage_primary = estimate_nominal_voltage(battery_primary, battery_primary['Max_discharge_current'])
    nominal_voltage_secondary = estimate_nominal_voltage(battery_secondary, battery_secondary['Max_discharge_current'])


    total_voltage_primary = num_series_primary * nominal_voltage_primary
    total_energy_primary = (
        num_series_primary * num_parallel_primary *
        battery_primary['Q_nominal'] * nominal_voltage_primary
    )
    max_power_primary = (
        num_series_primary * num_parallel_primary *
        battery_primary['Max_discharge_current'] * nominal_voltage_primary
    )

    # Step 2: Determine the deficits
    energy_deficit = total_energy - total_energy_primary
    power_deficit = peak_power - max_power_primary
    voltage_deficit = voltage_required - total_voltage_primary

    # Penalize infeasible designs with negative voltage deficit
```

```
    if voltage_deficit <= 0:
        return float('inf'), None, None  # Return penalty and skip further calculations

    # Step 3: Calculate the series cells needed for the secondary battery
    num_series_secondary = math.ceil(voltage_deficit / nominal_voltage_secondary)

    # Step 4: Calculate the parallel cells needed to meet energy and power deficits
    secondary_cell_energy = num_series_secondary * nominal_voltage_secondary *
    battery_secondary['Q_nominal']
    if secondary_cell_energy == 0:
        return float('inf'), None, None  # Return penalty for infeasible configurations

    num_parallel_energy = math.ceil(max(0, energy_deficit) / secondary_cell_energy)

    secondary_cell_power = num_series_secondary * nominal_voltage_secondary *
    battery_secondary['Max_discharge_current']
    num_parallel_power = math.ceil(max(0, power_deficit) / secondary_cell_power)

    # Final number of parallel cells
    num_parallel_secondary = max(num_parallel_energy, num_parallel_power)

    # Step 5: Calculate weights
    primary_pack_weight = (
        num_series_primary * num_parallel_primary * battery_primary['Cell_weight']
    )
    secondary_pack_weight = (
        num_series_secondary * num_parallel_secondary * battery_secondary['Cell_weight']
    )
    total_weight = primary_pack_weight + secondary_pack_weight

    return total_weight, num_series_secondary, num_parallel_secondary


# Function to plot curves including efficiency
def plot_curves(current_range, pack_current_range, polarization_curve, power_density_curve,
    power_pack_density_curve, efficiency_curve, pack_efficiency_curve):
    plt.figure(figsize=(18, 12))

    # Plot Polarization Curve (Single Cell)
    plt.subplot(2, 3, 1)
    plt.plot(current_range, polarization_curve, marker='o', linestyle='-')
    plt.title("Polarization Curve (Single Cell)")
    plt.xlabel("Current (A)")
    plt.ylabel("Terminal Voltage (V)")
    plt.grid(True)

    # Plot Power Density Curve (Single Cell)
    plt.subplot(2, 3, 2)
    plt.plot(current_range, power_density_curve, marker='s', linestyle='-', color='orange')
    plt.title("Power Density Curve (Single Cell)")
    plt.xlabel("Current (A)")
    plt.ylabel("Power Density (W/m )")
    plt.grid(True)

    # Plot Power Density Curve (Battery Pack)
    plt.subplot(2, 3, 3)
    plt.plot(pack_current_range, power_pack_density_curve, marker='^', linestyle='-', color='
    purple')
    plt.title("Power Density Curve (Battery Pack)")
    plt.xlabel("Current (A)")
    plt.ylabel("Power Density (W/m )")
    plt.grid(True)

    # Plot Efficiency Curve (Single Cell)
    plt.subplot(2, 3, 4)
    plt.plot(current_range[1:], efficiency_curve[1:], marker='s', linestyle='-', color='green
    ')  # Skip first point to avoid jump
    plt.title("Efficiency Curve (Single Cell)")
    plt.xlabel("Current (A)")
    plt.ylabel("Efficiency (%)")
    plt.grid(True)
```

```
    # Plot Efficiency Curve (Battery Pack)
    plt.subplot(2, 3, 5)
    plt.plot(pack_current_range[1:], pack_efficiency_curve[1:], marker='^', linestyle='-',
    color='blue')  # Skip first point to avoid jump
    plt.title("Efficiency Curve (Battery Pack)")
    plt.xlabel("Current (A)")
    plt.ylabel("Efficiency (%)")
    plt.grid(True)

    plt.tight_layout()
    plt.show()
```

battery_model.py

C.4. FUEL CELL MODEL

```
from opem.Static.Amphlett import Static_Analysis
import numpy as np
from BOP import mdots, compressor_power, HEX_Power, fc_mass
from scipy.optimize import minimize
import flightdata_reader as fr
import matplotlib.pyplot as plt



def size_fuel_cell(desired_voltage, power_required, test_vector, flight_data):
    """
    Function to size a fuel cell for a given desired voltage, power requirement, test vector,
    and mission profile.

    Parameters:
        desired_voltage (float): The desired stack voltage (V).
        power_required (float): The required power output (W).
        test_vector (dict): The test vector parameters for the fuel cell model.
        flight_data (dict): Mission profile data.

    Returns:
        dict: A dictionary containing the sized fuel cell's characteristics and performance
    metrics.
    """

    # ------------------------------------------------------------------------
    #  Extract Mission Profile Data
    # ------------------------------------------------------------------------
    Altitude = fr.get_altitude('Cruise', flight_data)
    flight_speed = fr.get_flying_speed('Cruise', flight_data)
    flight_power = fr.get_power('Cruise', flight_data)
    flight_mach = fr.get_flying_Mach(Altitude, flight_speed)
    tankIV_pd = 5.7  # wt%

    # Retrieve flight/mission time for this segment (adjust if needed)
    flight_time = fr.get_elapsed_time('Cruise', flight_data)

    # Air properties
    T0, Tt0, p0, pt0, _, _ = fr.calulate_air_properties(Altitude, flight_mach)

    # Compressor pressure ratio
    beta = 0.25e6 / pt0

    # ------------------------------------------------------------------------
    #  Run Static Analysis
    # ------------------------------------------------------------------------
    data = Static_Analysis(
        InputMethod=test_vector,
        TestMode=True,
        PrintMode=False,
        ReportMode=False
    )
```

```
    # Store static analysis values as arrays
    fc_power = np.array(data["P"])
    fc_current = np.array(data["I"])
    fc_voltage = np.array(data["V"])
    fc_eff = np.array(data["EFF"])
    fc_VE = np.array(data["VE"])
    fc_ph = np.array(data["Ph"])
    fc_eta_active = np.array(data["Eta_Active"])
    fc_eta_conc = np.array(data["Eta_Conc"])
    fc_eta_ohmic = np.array(data["Eta_Ohmic"])

    # ---------------------------------------------------------------------
    #  Filter valid points
    # ---------------------------------------------------------------------
    valid_indices = (fc_voltage >= 0) & (fc_current > 0)

    fc_power      = fc_power[valid_indices]
    fc_current    = fc_current[valid_indices]
    fc_voltage    = fc_voltage[valid_indices]
    fc_eff        = fc_eff[valid_indices]
    fc_VE         = fc_VE[valid_indices]
    fc_ph         = fc_ph[valid_indices]
    fc_eta_active = fc_eta_active[valid_indices]
    fc_eta_conc   = fc_eta_conc[valid_indices]
    fc_eta_ohmic  = fc_eta_ohmic[valid_indices]

    # ---------------------------------------------------------------------
    #  Compute basic sizing parameters
    # ---------------------------------------------------------------------
    n_stacks = np.ceil(desired_voltage / fc_voltage)
    current_required = power_required / desired_voltage
    initial_area = current_required / fc_current  # for the minimize() starting guess

    # ---------------------------------------------------------------------
    #  Find the optimal area for each valid operating point
    # ---------------------------------------------------------------------
    optimal_areas = []
    for i in range(len(fc_voltage)):

        if fc_voltage[i] <= 0 or fc_current[i] <= 0:
            optimal_areas.append(np.nan)
            continue

        # We only return area in objective, since we pick final design by total system mass
        def objective(area):
            return area

        # Constraint ensures net power (FC minus parasitics) >= required
        def constraint(area):
            P_FC = fc_voltage[i] * fc_current[i] * area * n_stacks[i]
            mdot_air_in, _, _, _, _ = mdots(P_FC, fc_voltage[i])
            P_comp = compressor_power(beta, mdot_air_in, Tt0, 0.9, 0.8)
            P_cs, _ = HEX_Power(test_vector["T"], T0, fc_voltage[i], P_FC)
            return P_FC - P_comp - P_cs - power_required

        result = minimize(
            fun=objective,
            x0=[initial_area[i]],
            bounds=[(0, None)],
            constraints={"type": "ineq", "fun": constraint},
            method="SLSQP"
        )

        if result.success:
            optimal_areas.append(result.x[0])
        else:
            optimal_areas.append(np.nan)

    optimal_areas = np.array(optimal_areas)
```

```
    # ------------------------------------------------------------------------
    #  Hardware Mass (fuel cell itself)
    # ------------------------------------------------------------------------
    cell_mass = fc_mass(n_stacks, optimal_areas)

    # ------------------------------------------------------------------------
    #  Determine total system mass (FC hardware + H2 used)
    # ------------------------------------------------------------------------
    total_mass_array = []
    for i in range(len(fc_voltage)):
        if np.isnan(optimal_areas[i]):
            total_mass_array.append(np.nan)
            continue

        # Power from fuel cell at this design
        P_FC_i = fc_voltage[i] * fc_current[i] * optimal_areas[i] * n_stacks[i]

        # Hydrogen consumption rate
        mdot_air_in_i, o2_used_i, mdot_air_out_i, h2o_produced_i, h2_used_i = mdots(P_FC_i,
    fc_voltage[i])

        # total H2 mass for the flight_time
        total_h2_mass = h2_used_i * flight_time


        total_tank_mass = (total_h2_mass/(tankIV_pd/100)) - total_h2_mass
        # cell hardware mass + H2 mass
        total_system_mass = cell_mass[i] + total_h2_mass + total_tank_mass
        total_mass_array.append(total_system_mass)

    total_mass_array = np.array(total_mass_array)

    # ------------------------------------------------------------------------
    #  Filter valid results
    # ------------------------------------------------------------------------
    valid_mask = ~np.isnan(total_mass_array)
    fc_voltage      = fc_voltage[valid_mask]
    fc_current      = fc_current[valid_mask]
    fc_power        = fc_power[valid_mask]
    fc_eff          = fc_eff[valid_mask]
    fc_VE           = fc_VE[valid_mask]
    fc_ph           = fc_ph[valid_mask]
    fc_eta_active   = fc_eta_active[valid_mask]
    fc_eta_conc     = fc_eta_conc[valid_mask]
    fc_eta_ohmic    = fc_eta_ohmic[valid_mask]
    fc_stacks       = n_stacks[valid_mask]
    optimal_areas   = optimal_areas[valid_mask]
    cell_mass       = cell_mass[valid_mask]
    total_mass_array = total_mass_array[valid_mask]

    # ------------------------------------------------------------------------
    #  Find the design with the minimum total system mass
    # ------------------------------------------------------------------------
    min_mass_index = np.argmin(total_mass_array)
    lowest_mass_h2 = total_mass_array[min_mass_index]

    # Retrieve the associated design variables
    lowest_mass = cell_mass[min_mass_index]
    corresponding_voltage = fc_voltage[min_mass_index]
    corresponding_current = fc_current[min_mass_index]
    corresponding_power_density = fc_power[min_mass_index]
    corresponding_area = optimal_areas[min_mass_index]
    corresponding_stacks = fc_stacks[min_mass_index]
    corresponding_VE = fc_VE[min_mass_index]

    # Final power at the chosen design point
    corresponding_fc_power = (
        corresponding_voltage
        * corresponding_current
        * corresponding_area
        * corresponding_stacks
```

```
    )

    # Final flows and parasitics for the chosen design
    mdot_air_in, o2_used, mdot_air_out, h2o_produced, h2_used = mdots(
        corresponding_fc_power, corresponding_voltage
    )
    power_comp = compressor_power(beta, mdot_air_in, Tt0, 0.9, 0.8)
    power_hex, _ = HEX_Power(
        test_vector["T"], T0, corresponding_voltage, corresponding_fc_power
    )

    # ------------------------------------------------------------------
    #  Return the results in the same dictionary format as before
    # ------------------------------------------------------------------
    return {
        "Lowest Mass (kg)": lowest_mass,  # now includes both FC hardware + H2
        "Optimal Voltage (V)": corresponding_voltage,
        "Optimal Current Density (A/cm^2)": corresponding_current,
        "Optimal FC Power (W)": corresponding_fc_power,
        "Efficiency": fc_eff[min_mass_index],
        "The Nernst Voltage": corresponding_VE,
        "Optimal Area (cm^2)": corresponding_area,
        "The number of cells in series": corresponding_stacks,
        "PH": fc_ph[min_mass_index],
        "Active Overpotential (V)": fc_eta_active[min_mass_index],
        "Concentration Overpotential (V)": fc_eta_conc[min_mass_index],
        "Ohmic Overpotential (V)": fc_eta_ohmic[min_mass_index],
        "Airflow in (kg/s)": mdot_air_in,
        "Airflow out (kg/s)": mdot_air_out,
        "Hydrogen consumption rate (kg/s)": h2_used,
        "Oxygen consumption rate (kg/s)": o2_used,
        "Water production rate (kg/s)": h2o_produced,
        "Compressor Power (W)": power_comp,
        "HEX Power (W)": power_hex
    }


def simulate_off_design(off_area, off_stacks, Test_Vector, flight_data, flight_operation='
    Take Off'):
    """
    Simulates the off-design performance of the fuel cell for a given flight operation.

    Parameters:
        off_area (float): Active area of the fuel cell (cm ).
        off_stacks (int): Number of cells in series (stacks).
        Test_Vector (dict): Fuel cell operating conditions (temperature, pressure, etc.).
        flight_operation (str): The flight phase for the simulation (e.g., 'Take Off').

    Returns:
        dict: A dictionary containing the best performance metrics during the given flight
    operation.
    """

    # Run the static Amphlett model for off-design conditions
    off_data = Static_Analysis(InputMethod=Test_Vector, TestMode=True, PrintMode=False,
    ReportMode=False)

    # Store static analysis values as arrays
    off_power = np.array(off_data["P"])
    off_current = np.array(off_data["I"])
    off_voltage = np.array(off_data["V"])
    off_eff = np.array(off_data["EFF"])
    off_ph = np.array(off_data["Ph"])
    off_eta_active = np.array(off_data["Eta_Active"])
    off_eta_conc = np.array(off_data["Eta_Conc"])
    off_eta_ohmic = np.array(off_data["Eta_Ohmic"])

    # Filter valid indices where voltage >= 0 and current > 0
    off_valid_indices = (off_voltage >= 0) & (off_current > 0)
```

```
    # Apply mask to all relevant arrays
    off_power = off_power[off_valid_indices]
    off_current = off_current[off_valid_indices]
    off_voltage = off_voltage[off_valid_indices]
    off_eff = off_eff[off_valid_indices]
    off_ph = off_ph[off_valid_indices]
    off_eta_active = off_eta_active[off_valid_indices]
    off_eta_conc = off_eta_conc[off_valid_indices]
    off_eta_ohmic = off_eta_ohmic[off_valid_indices]

    # Get the flight power requirement for the selected operation
    off_flight_power = fr.get_power(flight_operation, flight_data)

    # Get the air properties at the altitude corresponding to the selected operation
    off_Altitude = fr.get_altitude(flight_operation, flight_data)
    off_flight_speed = fr.get_flying_speed(flight_operation, flight_data)
    off_flight_mach = fr.get_flying_Mach(off_Altitude, off_flight_speed)
    T0_off, Tt0_off, p0_off, pt0_off, _, _ = fr.calulate_air_properties(off_Altitude,
    off_flight_mach)
    # off_beta = 0.25e6 / pt0_off
    off_beta = 3.5

    # Initialize variables to track the best performance
    max_effective_power = None
    best_current_density = None
    best_voltage = None
    best_efficiency = None
    best_compressor_power = None
    best_cooling_power = None
    best_mdot_air_in = None
    best_o2_used = None
    best_mdot_air_out = None
    best_h2o_produced = None
    best_h2_used = None

    # Iterate over all valid current densities
    for i in range(len(off_current)):
        # Calculate fuel cell power
        P_FC = off_voltage[i] * off_current[i] * off_area * off_stacks

        # Calculate BoP losses
        mdot_air_in, o2_used, mdot_air_out, h2o_produced, h2_used = mdots(P_FC, off_voltage[i
    ])
        P_comp = compressor_power(off_beta, mdot_air_in, Tt0_off, 0.9, 0.8)
        P_cs, qheat = HEX_Power(Test_Vector["T"], T0_off, off_voltage[i], P_FC)

        # Calculate effective power
        P_effective = P_FC - P_comp - P_cs

        # Check if this is the best effective power
        if max_effective_power is None or P_effective > max_effective_power:
            max_effective_power = P_effective
            best_current_density = off_current[i]
            best_voltage = off_voltage[i]
            best_efficiency = off_eff[i]
            best_compressor_power = P_comp
            best_cooling_power = P_cs
            best_mdot_air_in = mdot_air_in
            best_o2_used = o2_used
            best_mdot_air_out = mdot_air_out
            best_h2o_produced = h2o_produced
            best_h2_used = h2_used

    # Return the best performance metrics as a dictionary
    return {
        "Best Effective Power (W)": max_effective_power,
        "Best Current Density (A/cm )": best_current_density,
        "Best Voltage (V)": best_voltage,
        "Best Efficiency": best_efficiency,
        "Best Compressor Power (W)": best_compressor_power,
        "Best Cooling Power (W)": best_cooling_power,
```

```
        "Best Airflow Rate (kg/s)": best_mdot_air_in,
        "Best Oxygen Used (kg/s)": best_o2_used,
        "Best Air Outflow Rate (kg/s)": best_mdot_air_out,
        "Best Water Produced (kg/s)": best_h2o_produced,
        "Best Hydrogen Used (kg/s)": best_h2_used,
    }


def calculate_hydrogen_fuel(fuel_cell_area, fuel_cell_stacks, test_vector, flight_data,
    flight_operation=None):
    """
    Calculates the hydrogen fuel consumption for a specific flight operation or the entire
    mission.

    Parameters:
        fuel_cell_area (float): Active area of the fuel cell (cm ).
        fuel_cell_stacks (int): Number of cells in series in the fuel cell stack.
        test_vector (dict): Fuel cell operating parameters (temperature, pressure, etc.).
        flight_data (pd.DataFrame): Mission profile data containing flight phases, altitude,
    and power.
        flight_operation (str, optional): Specific flight phase (e.g., 'Take Off'). If None,
    calculates for the entire mission.

    Returns:
        float: Total hydrogen fuel used (kg).
    """

    if flight_operation:
        # Simulate off-design performance for the given operation
        results = simulate_off_design(fuel_cell_area, fuel_cell_stacks, test_vector,
    flight_data, flight_operation)

        # Extract hydrogen consumption rate (kg/s) and elapsed time for the operation
        h2_rate = results['Best Hydrogen Used (kg/s)']
        elapsed_time = fr.get_elapsed_time(flight_operation, flight_data)

        # Calculate hydrogen used for this flight phase
        hydrogen_used = h2_rate * elapsed_time
        return hydrogen_used
    else:
        # Calculate hydrogen consumption for the entire mission profile
        total_hydrogen_used = 0

        # List of all flight phases
        flight_phases = [
            'Take Off', 'Ascend', 'Cruise', 'Approach',
            'Take Down', 'Take Off R', 'Cruise R', 'Take Down R'
        ]

        for phase in flight_phases:
            # Simulate off-design performance for each phase
            results = simulate_off_design(fuel_cell_area, fuel_cell_stacks, test_vector,
    flight_data, phase)

            # Extract hydrogen consumption rate (kg/s) and elapsed time for the phase
            h2_rate = results['Best Hydrogen Used (kg/s)']
            elapsed_time = fr.get_elapsed_time(phase, flight_data)

            # Add hydrogen consumption for this phase to the total
            total_hydrogen_used += h2_rate * elapsed_time

        return total_hydrogen_used

def new_method_fc(desired_voltage, power_required, test_vector, flight_data):
    """
      Function to size a fuel cell for a given desired voltage, power requirement, test
    vector, and mission profile.

      Parameters:
          desired_voltage (float): The desired stack voltage (V).
          power_required (float): The required power output (W).
```

```
        test_vector (dict): The test vector parameters for the fuel cell model.
        beta (float): Compressor pressure ratio.
        flight_data (dict): Mission profile data.

    Returns:
        dict: A dictionary containing the sized fuel cell's characteristics and performance metrics.
    """

    # Extract mission profile information
    Altitude = fr.get_altitude('Cruise', flight_data)
    flight_speed = fr.get_flying_speed('Cruise', flight_data)
    flight_power = fr.get_power('Cruise', flight_data)
    flight_mach = fr.get_flying_Mach(Altitude, flight_speed)

    # Air properties for the mission profile
    T0, Tt0, p0, pt0, _, _ = fr.calulate_air_properties(Altitude, flight_mach)

    # The compressor pressure ratio
    beta = 0.25e6 / pt0

    # Run Static Analysis
    data = Static_Analysis(InputMethod=test_vector, TestMode=True, PrintMode=False, ReportMode=False)

    # Store static analysis values as arrays
    fc_power = np.array(data["P"])
    fc_current = np.array(data["I"])
    fc_voltage = np.array(data["V"])
    fc_eff = np.array(data["EFF"])
    fc_ph = np.array(data["Ph"])
    fc_eta_active = np.array(data["Eta_Active"])
    fc_eta_conc = np.array(data["Eta_Conc"])
    fc_eta_ohmic = np.array(data["Eta_Ohmic"])

    # Filter valid indices where voltage >= 0 and current > 0
    valid_indices = (fc_voltage >= 0) & (fc_current > 0)

    # Apply mask to all relevant arrays
    fc_power = fc_power[valid_indices]
    fc_current = fc_current[valid_indices]
    fc_voltage = fc_voltage[valid_indices]
    fc_eff = fc_eff[valid_indices]
    fc_ph = fc_ph[valid_indices]
    fc_eta_active = fc_eta_active[valid_indices]
    fc_eta_conc = fc_eta_conc[valid_indices]
    fc_eta_ohmic = fc_eta_ohmic[valid_indices]

    # Calculate stacks and initial area
    n_stacks = np.ceil(desired_voltage / fc_voltage)
    current_required = power_required / desired_voltage
    initial_area = current_required / fc_current

    # Calculate fuel cell mass
    mass = fc_mass(n_stacks, initial_area)

    # Find the index of the configuration with the lowest fuel cell mass
    min_mass_index = np.argmin(mass)

    # Retrieve the corresponding performance characteristics
    lowest_mass = mass[min_mass_index]
    corresponding_voltage = fc_voltage[min_mass_index]
    corresponding_power_density = fc_power[min_mass_index]
    corresponding_area = initial_area[min_mass_index]
    corresponding_stacks = n_stacks[min_mass_index]
    corresponding_fc_power = corresponding_power_density * corresponding_area * corresponding_stacks

    mdot_air_in, o2_used, mdot_air_out, h2o_produced, h2_used = mdots(corresponding_fc_power, corresponding_voltage)
    print("The hydrogen fuel cell is properly sized ...")
```

```
461
    # Return the results
463     return {
            "Lowest Mass (kg)": lowest_mass,
465         "Optimal Voltage (V)": fc_voltage[min_mass_index],
            "Optimal Current Density (A/cm^2)": fc_current[min_mass_index],
467         "Optimal Power Density (W/cm^2)": fc_power[min_mass_index],
            "Efficiency": fc_eff[min_mass_index],
469         "Optimal Area (cm^2)": initial_area[min_mass_index],
            "The number of cells in series": corresponding_stacks,
471         "PH": fc_ph[min_mass_index],
            "Active Overpotential (V)": fc_eta_active[min_mass_index],
473         "Concentration Overpotential (V)": fc_eta_conc[min_mass_index],
            "Ohmic Overpotential (V)": fc_eta_ohmic[min_mass_index],
475         "Airflow in (kg/s)": mdot_air_in,
            "Airflow out (kg/s)": mdot_air_out,
477         "Hydrogen consumption rate (kg/s)": h2_used,
            "Oxygen consumption rate (kg/s)": o2_used,
479         "Water production rate (kg/s)": h2o_produced
    }
```

fc_model.py

C.5. GSP DATA

```
import numpy as np
2 import pandas as pd
from scipy.interpolate import griddata
4
def read_GSP(filename, rows_to_skip):
6     df = pd.read_csv(filename, encoding='unicode_escape', header=[1], skiprows=rows_to_skip,
      decimal=',')
      return df
8
def get_data(df):
10    Altitude = df['Zp\r\n[m]'].astype(float)
      Mach = df['Macha\r\n[-]'].astype(float)
12    Tt4 = df['Tt4\r\n[K]'].astype(float)
      Tt3 = df['Tt3\r\n[K]'].astype(float)
14    PWshaft = df['PWshaft\r\n[kW]'].astype(float)
      WF = df['WF\r\n[kg/s]'].astype(float)
16    W = df['W\r\n[kg/s]'].astype(float)
      return Altitude, Mach, Tt4, PWshaft, WF, W, Tt3
18
rows_to_skip = list(range(2, 989, 17)) + [989, 990, 991, 992] # NaN rows excluding the design
     point
20
df = read_GSP('DATA_GT3.csv', rows_to_skip)
22
# print(df)
24
Altitude, Mach, Tt4, PWshaft, WF, W, Tt3 = get_data(df)
26
# Prepare data for interpolation
28 points = df[['Macha\r\n[-]', 'Tt4\r\n[K]', 'Zp\r\n[m]']].values
valuesWf = df['WF\r\n[kg/s]'].values
30 valuesPWshaft = df['PWshaft\r\n[kW]'].values
valuesTt3 = df['Tt3\r\n[K]'].values
32
def interpolate_wf(Mach, Tt4, Altitude):
34    return griddata(points, valuesWf, (Mach, Tt4, Altitude), method='linear')

36 def interpolate_Tt3(Mach, Tt4, Altitude):
      return griddata(points, valuesTt3, (Mach, Tt4, Altitude), method='linear')
38
def interpolate_PWshaft(Mach, Tt4, Altitude):
40    return griddata(points, valuesPWshaft, (Mach, Tt4, Altitude), method='linear')

42 def calc_gt_efficiency(flight_mach, Tt4, altitude, PW_GT):
```

```
    fuelgtflow = interpolate_wf(flight_mach, Tt4, altitude)
    gt_eff = PW_GT / (fuelgtflow * 120*10**6)
    return gt_eff
```

GSP_data.py

C.6. BALANCE OF PLANT MODEL

```
import matplotlib.pyplot as plt

def mdots(power, cell_voltage, lamda=None):
    if lamda is None:
        lamda = 2  #Typical stoichiometric ratio [Enough air to supply oxygen to the FC]
    F = 96485  # Faraday constant [C/mol]
    o2MolarMass = 31.9988e-3  # [kg/mol]
    h2MolarMass = 2.01588e-3  # [kg/mol]
    h2oMolarMass = 18.01528e-3  # [kg/mol]
    airmolarMass = 28.9647e-3  # [kg/mol]
    mdot_air_in = (airmolarMass / (0.21*4*F)) * (power/cell_voltage) * lamda  #kg/s
    o2_used = (o2MolarMass / (4*F)) * (power/cell_voltage) #kg/s
    mdot_air_out = mdot_air_in - o2_used #kg/s
    h2o_produced = (h2oMolarMass / (2 * F)) * (power/cell_voltage) #kg/s
    h2_used = (h2MolarMass / (2 * F)) * (power/cell_voltage) #kg/s
    return mdot_air_in, o2_used, mdot_air_out, h2o_produced, h2_used

def compressor_power(beta, mdot, Tt0, eta_m, eta_c): # realistically eta_m is 0.9 and eta_c
    is 0.8
    gamma = 1.4  # [-]
    cp = 1004  # J/kgK
    Power = mdot * cp * (Tt0/(eta_m * eta_c)) * (((beta)**((gamma -1)/gamma)) - 1)
    return Power

def HEX_Power(T_op, T_static, V_cell, P_FC):
    st = 1 # [-] # This is the anode stoichiometric ratio, and it is 100% fuel utilization
    # I had 0.95 multiplied by V_cell for some reason but I took it out becasue it does not
    make sense to me
    E0 = 241.83e3 / (2 * 96485)  # J/mol
    q_dot_heat = ((E0*st/(V_cell)) - 1) * P_FC # From the fuel cell system explained by
    larmine book considering water as vapour
    # q_dot_heat = ((1.48*st/(V_cell)) - 1) * P_needed # The equation is from the fuel cell
    fundamentals book considering water as liquid Conservative
    f_dt = 0.0038 * (T_static/(T_op - T_static))**2 + 0.0352 * (T_static / (T_op - T_static)
    ) + 0.1817
    P_cs = (0.371 * q_dot_heat + 1.33) * f_dt
    return P_cs , q_dot_heat  #I am switching these two values around for now !!! [I brought
    it back to normal]

## Thesis by daniel is used here to estimate the fuel cell mass
def fc_mass(n_cells, area_cell):
    """
    Calculate mass of stack(s) in the FC system.

    :param n_stacks_series: Number of stacks in series in FC system
    :param volt_req: Voltage to be delivered by FC system in V
    :param volt_cell: Nominal cell voltage in V
    :param power_req: Electrical power to be delivered by stacks (bigger than propulsive
    output power of FC system)
    :param power_dens_cell: Nominal cell power density in W/m^2
    :return: mass of stack(s) in kg
    """

    # Convert cell area from cm^2 to m^2
    area_cell = area_cell*0.0001

    # # constants
    # bipolar plate
    t_bp = 2e-4  # m
    rho_bp = 8e3  # kg/m3 - SS304L
```

```
    # endplate
    t_ep = 2.5e-2  # m
    rho_ep = 8e3  # kg/m3 - same as bp

    # bolts - remove those?
    n_bolt = 10  # see Dey 2019
    rho_bolt = 8e3

    # MEA
    rho_mea = 0.2  # kg/m2 - Kadyk 2018

    m_bp = n_cells*t_bp*rho_bp*area_cell  # mass of bipolar plates of one stack
    m_ep = 2*t_ep*rho_ep*area_cell  # mass of endplates of one stack
    m_mea = rho_mea*area_cell*n_cells  # mass of MEA of one stack

    # d_bolt = sqrt(area_cell/600)  # diameter of bolts - see Dhttps://www.
    volkshuisvestingnederland.nl/wat-betekent-de-wet-betaalbare-huur-voor-mij/situaties/ik-
    huur/ik-huur-een-woningey 2019
    # l_bolt = 2*t_ep + n_cells*t_bp  # length of bolts
    # m_bolts = n_bolt*pi*(d_bolt/2)**2*l_bolt*rho_bolt  # mass of bolts of one stack

    m_tot = m_bp+m_ep+m_mea  # mass of a single stack


    return m_tot

def weight_hps(fcmass, fuelused, battery_weight, compressor_power, cooling_power):
    tg_pd = 0.00435e6 #W/kg
    comp_pd = 11/11250 #kg/W
    hex_pd = 11.1 / (1.84e3 + 3.72e3) #kg/W
    tankIV_pd = 5.7 #wt%
    # tankIV_pd = 60  #wt%
    fuel_tank = (fuelused/(tankIV_pd/100)) - fuelused
    # bop = 51.1 #kg
    structure = 1905 #kg
    print('The weight of the compressor is',compressor_power*comp_pd , ' kg')
    print('The weight of the HEX is', cooling_power*hex_pd, ' kg')
    print('The weight of the fuel tank is', fuel_tank, ' kg')
    hps_weight = fcmass + battery_weight + compressor_power*comp_pd + cooling_power*hex_pd +
    structure + fuel_tank - fuelused
    return hps_weight

def weight_hps_tank(fcmass, fuelused, battery_weight, compressor_power, cooling_power,
    fc_converter_power, inverter_power, bat_converter_power):
    comp_pd = 11 / 11250  # kg/W
    hex_pd = 11.1 / (1.84e3 + 3.72e3)  # kg/W
    tankIV_pd = 1.5  # kgh2/kgtank
    PE_converter_pd = 7500  # W/kg
    fc_dc_weight = fc_converter_power / PE_converter_pd
    bat_dc_weight = bat_converter_power / PE_converter_pd
    inverter_weight = inverter_power / PE_converter_pd
    PE_converter_weight = fc_dc_weight + inverter_weight + bat_dc_weight
    structure = 1905  # kg
    print('The weight of the compressor is', compressor_power * comp_pd, ' kg')
    print('The weight of the HEX is', cooling_power * hex_pd, ' kg')
    print('The weight of the fuel tank is', fuelused / tankIV_pd , ' kg')
    hps_weight = fcmass + battery_weight + PE_converter_weight + compressor_power * comp_pd +
     cooling_power * hex_pd + structure + (fuelused / tankIV_pd) - fuelused
    return hps_weight

def weight_hps_bat(fcmass, fuelused, battery_weight, compressor_power, cooling_power,
    fc_converter_power, inverter_power, bat_converter_power):
    comp_pd = 11/11250 #kg/W
    hex_pd = 11.1 / (1.84e3 + 3.72e3) #kg/W
    tankIV_pd = 5.7 #wt% gravimetric capacity of tank
    PE_converter_pd = 7500 #W/kg
    fuel_tank = (fuelused / (tankIV_pd / 100)) - fuelused
    fc_dc_weight = fc_converter_power/PE_converter_pd
    bat_dc_weight = bat_converter_power/PE_converter_pd
    inverter_weight = inverter_power/PE_converter_pd
    PE_converter_weight = fc_dc_weight + inverter_weight + bat_dc_weight
```

```
    structure = 1905 #kg
    print('The weight of the compressor is',compressor_power*comp_pd , ' kg')
    print('The weight of the HEX is', cooling_power*hex_pd, ' kg')
    print('The weight of the fuel tank is', fuel_tank, ' kg')
    hps_weight = fcmass + battery_weight + PE_converter_weight + compressor_power*comp_pd +
    cooling_power*hex_pd + structure + fuel_tank
    return hps_weight

def weight_fc_system(fcmass, compressor_power, cooling_power, fuel_used):
    comp_pd = 11 / 11250  # kg/W
    hex_pd = 11.1 / (1.84e3 + 3.72e3)  # kg/W
    tankIV_pd = 5.7  # wt% gravimetric capacity of tank
    compressor_weight = compressor_power * comp_pd
    hex_weight = cooling_power * hex_pd
    tank_weight = fuel_used/(tankIV_pd/100)
    fc_system_weight = fcmass + compressor_weight + hex_weight + tank_weight
    return fc_system_weight


def weight_hps_tg(fcmass, fuelused_fc, compressor_power, cooling_power, tg_power, fuelused_tg
    ):
    tg_pd = 0.00435e6 #W/kg
    comp_pd = 11/11250 #kg/W
    hex_pd = 11.1 / (1.84e3 + 3.72e3) #kg/W
    tankIV_pd = 5.7 #wt% gravimetric capacity of tank
    weight_tg = tg_power / tg_pd
    total_fuel = fuelused_fc + fuelused_tg
    structure = 1905 #kg
    print('The weight of the fuel cell stack is ', fcmass, ' kg')
    print('The weight of the turbogenerator is',weight_tg, ' kg' )
    print('The weight of the compressor is',compressor_power*comp_pd , ' kg')
    print('The weight of the HEX is', cooling_power*hex_pd, ' kg')
    print('The weight of the tank is', total_fuel/(tankIV_pd*100), ' kg')

    hps_weight = fcmass + compressor_power*comp_pd + cooling_power*hex_pd + structure +
    weight_tg + total_fuel/(tankIV_pd*100)
    return hps_weight

def weight_hps_bat_breakdown(fcmass, fuelused, battery_weight,
                             compressor_power, cooling_power,
                             fc_converter_power, inverter_power, bat_converter_power):
    comp_pd = 11/11250      # kg/W
    hex_pd = 11.1/(1.84e3 + 3.72e3)  # kg/W
    tankIV_pd = 5.7         # wt% gravimetric capacity of tank
    # tankIV_pd = 60
    PE_converter_pd = 7500  # W/kg

    # Power electronics masses
    fc_dc_weight = fc_converter_power / PE_converter_pd
    bat_dc_weight = bat_converter_power / PE_converter_pd
    inverter_weight = inverter_power / PE_converter_pd
    PE_converter_weight = fc_dc_weight + bat_dc_weight + inverter_weight

    # Other subcomponent masses
    structure = 1905  # kg (fixed structure mass)
    comp_weight = compressor_power * comp_pd
    hex_weight = cooling_power * hex_pd
    tank_weight =  (fuelused / (tankIV_pd / 100)) - fuelused  # hydrogen tank mass

    # The total is the sum minus the actual hydrogen mass (fuelused)
    # because the function originally subtracts it.
    # (You can adjust if you prefer total mass *with* the fuel included.)
    total_hps_weight = (
        fcmass
        + battery_weight
        + PE_converter_weight
        + comp_weight
        + hex_weight
        + structure
        + tank_weight
    )
```

```
    # Build a dictionary for clarity
    breakdown = {
        'Fuel Cell Stack': fcmass,
        'Battery Pack': battery_weight,
        'PE Converters': PE_converter_weight,
        'Compressor': comp_weight,
        'Heat Exchanger': hex_weight,
        'Tank (empty)': tank_weight,
        'Fuel Mass': fuelused,
        'Structure': structure,
        'OEW': total_hps_weight
    }
    return breakdown


def plot_weight_breakdown(mass_breakdown, title='Weight Breakdown for FC-BAT Architecture'):
    """
    Plots a simple bar chart of weight breakdown from a mass breakdown dictionary.

    Parameters
    ----------
    mass_breakdown : dict
        A dictionary with subcomponent names (str) as keys
        and their masses in kg (float) as values.
        Example keys:
            'Fuel Cell Stack', 'Battery Pack', 'PE Converters',
            'Compressor', 'Heat Exchanger', 'Tank (empty)',
            'Structure', 'Minus Fuel Mass', 'Total'.
        The function will skip the 'Total' key when plotting.

    title : str, optional
        Chart title, by default 'Weight Breakdown for FC-BAT Architecture'.
    """

    # Separate keys and values for plotting
    labels = []
    values = []

    for key, val in mass_breakdown.items():
        # Skip the 'Total' key (or any other key you don't want to plot).
        if key.lower() == 'total':
            continue
        labels.append(key)
        values.append(val)

    # Create the bar chart
    plt.figure(figsize=(8, 5))
    bars = plt.bar(labels, values)
    plt.xticks(rotation=45, ha='right')
    plt.ylabel('Mass (kg)')
    plt.title(title)
    plt.grid(axis='y')

    # Add numeric labels above each bar
    for rect, val in zip(bars, values):
        height = rect.get_height()
        plt.text(
            rect.get_x() + rect.get_width() / 2,  # X position: center of bar
            height,                               # Y position: top of bar
            f"{val:.2f} kg",                      # Label text (two decimals)
            ha='center',                          # Horizontal alignment
            va='bottom'                           # Vertical alignment
        )

    plt.tight_layout()
    plt.show()


def weight_breakdown_bat(battery_weight, dc_converter_power, inverter_power):
    """
```

```
    Computes the weight breakdown for a battery-only architecture.

    Parameters
    ----------
    battery_weight : float
        The total battery mass in kg.
    dc_converter_power : float
        The DC converter power requirement in W (e.g., from battery_power).
    inverter_power : float
        The inverter power requirement in W (e.g., from the peak motor power).

    Returns
    -------
    dict
        A dictionary mapping each subcomponent (and the total) to its mass in kg.
    """

    # Constants
    PE_converter_pd = 7500.0  # W/kg (same as in your code)
    structure = 1905.0  # kg (fixed structural mass from your script)

    # Power electronics mass
    bat_dc_weight = dc_converter_power / PE_converter_pd
    inverter_weight = inverter_power / PE_converter_pd
    PE_converter_weight = bat_dc_weight + inverter_weight

    # Sum up total
    total_mass = battery_weight + PE_converter_weight + structure

    # Return a dictionary of subcomponent masses
    breakdown = {
        'Battery Pack': battery_weight,
        'DC Converter': bat_dc_weight,
        'Inverter': inverter_weight,
        'PE Converters (Total)': PE_converter_weight,
        'Structure': structure,
        'OEW': total_mass
    }
    return breakdown

def weight_hps_tg_breakdown(
    fcmass,                 # Fuel cell stack mass (kg)
    fuelused_fc,            # Hydrogen consumed (kg) by the fuel cell
    compressor_power,       # FC compressor power (W)
    cooling_power,          # FC heat exchanger/cooling power (W)
    tg_power,               # Turbogenerator shaft power (W)
    fuelused_tg,            # Turbogenerator fuel consumed (kg)
    dc_converter_power,     # DC/DC converter power (W)
    rectifier_power,        # Rectifier power (W)
    inverter_power,         # Inverter power (W)
    structure=1905.0        # Base eVTOL structural mass (kg), default
):

    # Power densities / scaling (from your prior scripts):
    tg_pd       = 0.00435e6           # W/kg for turbogenerator
    comp_pd     = 11 / 11250.0        # kg/W
    hex_pd      = 11.1 / (1.84e3 + 3.72e3)  # kg/W
    tankIV_pd   = 5.7                 # 5.7 wt% => 0.057 ratio
    PE_converter_pd = 7500.0          # W/kg for all power-electronics components

    # 1) Calculate turbogenerator mass
    tg_mass = tg_power / tg_pd

    # 2) Calculate total fuel (FC hydrogen + TG fuel)
    total_fuel = fuelused_fc + fuelused_tg

    # 3) Compressor & HEX mass
    compressor_mass = compressor_power * comp_pd
    hex_mass        = cooling_power   * hex_pd

    # 4) Tank mass from total_fuel
```

```
    tank_mass = (total_fuel / (tankIV_pd / 100.0)) - total_fuel

    # 5) Compute the mass for each power electronics component
    dc_converter_mass = dc_converter_power / PE_converter_pd
    rectifier_mass    = rectifier_power    / PE_converter_pd
    inverter_mass     = inverter_power     / PE_converter_pd

    # Sum of all power electronics
    pe_converter_mass = dc_converter_mass + rectifier_mass + inverter_mass

    # 6) Final total
    total_mass = (
        fcmass
        + tg_mass
        + compressor_mass
        + hex_mass
        + pe_converter_mass
        + tank_mass
        + structure
    )

    # Build the dictionary for the breakdown
    breakdown = {
        'Fuel Cell Mass': fcmass,
        'Turbogenerator Mass': tg_mass,
        'Compressor Mass': compressor_mass,
        'Heat Exchanger Mass': hex_mass,
        'DC/DC Converter Mass': dc_converter_mass,
        'Rectifier Mass': rectifier_mass,
        'Inverter Mass': inverter_mass,
        'PE Converter Mass': pe_converter_mass,
        'Tank Mass': tank_mass,
        'Total Fuel': total_fuel,
        'Structure': structure,
        'EOW': total_mass
    }

    return breakdown
```

BOP.py

C.7. FUEL CELL - TURBOGENERATOR SIMULATION MODEL

```
import matplotlib.pyplot as plt
import flightdata_reader as fr
import fc_model as fc
import numpy as np
from GSP_data import read_GSP, get_data, interpolate_wf, interpolate_Tt3, interpolate_PWshaft
    , calc_gt_efficiency
from BOP import *

# The operating conditions of the fuel cell
test_vector = {
    "T": 353.15,         # Kelvin
    "PH2": 2.5,          # atm
    "PO2": 0.518,        # atm
    "i-start": 0,        # Ampere
    "i-stop": 1.0,       # Ampere
    "i-step": 0.01,     # Ampere
    "A": 1,              # cm^2
    "l": 0.08,           # cm^2
    "lambda": 14,        # -
    "N": 1,              # -
    "R": 0,              # Ohm
    "JMax": 1,           # A/cm^2
    "Name": "Amphlett_Test"
}

desired_voltage = 800  # Volts
```

```
26
28  # ---------------------------------------------------------------------
    # Read Coarse Flight Data (from read_data_fc)
30  # ---------------------------------------------------------------------
    # These are the coarse data (one row per flight data point)
    timeF, Zh, Pw_r, TimeP, Pw_P, Pw_fuel_cell_P, Pw_battery_P, flight_condition, v_x, v_z,
        flight_data = \
32      fr.read_data_fc('Flight_Data.csv', fuel_cell_max_power=206.5937151)

34  # ---------------------------------------------------------------------
    # Read Detailed (High-Resolution) Flight Condition Data (from read_data_fs)
36  # ---------------------------------------------------------------------
    # This returns high-res flight_condition_ts that has the same dimensions as Pw_P.
38  _, _, _, _, _, flight_condition_ts, Zh_ts, v_x_ts, v_z_ts, _ = fr.read_data_fs('Flight_Data.
        csv')

40  TimeD = np.array(timeF)
    # ---------------------------------------------------------------------
42  # Fuel Cell Sizing and Off-Design Performance Mapping
    # ---------------------------------------------------------------------
44  PE_converter_efficiency = 0.98
    # power_required = Pw_fuel_cell_P.max() * 1000 / (PE_converter_efficiency**2) # in Watts  #
        DC/DC and rectifier
46  power_required = Pw_fuel_cell_P.max()*1000
    # DC_converter_weight = Pw_fuel_cell_P.max() / (PE_converter_efficiency**2) / 7.5           #
         Specific power of PE converter
48
    sizing_results_fc = fc.size_fuel_cell(desired_voltage, power_required, test_vector,
        flight_data)
50  fc_cells      = sizing_results_fc['The number of cells in series']
    fc_voltage    = sizing_results_fc['Optimal Voltage (V)']
52  fc_power      = sizing_results_fc['Optimal FC Power (W)']
    fc_current    = sizing_results_fc['Optimal Current Density (A/cm^2)']
54  fc_area       = sizing_results_fc['Optimal Area (cm^2)']
    fc_compressor = sizing_results_fc['Compressor Power (W)']
56  fc_hex        = sizing_results_fc["HEX Power (W)"]
    fc_hydrogen   = sizing_results_fc["Hydrogen consumption rate (kg/s)"]
58  fc_eff        = sizing_results_fc["Efficiency"]
    voltage_efficiency = sizing_results_fc["The Nernst Voltage"]
60  fuel_cell_mass = fc_mass(fc_cells, fc_area)


62
    print("Fuel Cell Nominal Performance:")
64  print("  FC Power:", fc_power, "W")
    print("  Compressor Power:", fc_compressor, "W")
66  print("  HEX Power:", fc_hex, "W")

68  # Define flight segments (these labels must exactly match those used in off_design)
    segments = ["Idle","Take Off", "Ascend", "Approach", "Cruise", "Take Down", "Idle", "Take Off
         R", "Cruise R", "Take Down R"]
70
    off_design_results = {}
72  for segment in segments:
        results = fc.simulate_off_design(fc_area, fc_cells, test_vector, flight_data, segment)
74      off_design_results[segment] = results
        print(f"--- Segment: {segment} ---")
76      print("  Effective Power (W):", results.get('Best Effective Power (W)', 'N/A'))
        print("  Hydrogen Consumption (kg/s):", results.get('Best Hydrogen Used (kg/s)', 'N/A'),
        "\n")
78
    # Build mappings for effective power and hydrogen consumption rate.
80  effective_power_map = {seg: off_design_results.get(seg, {}).get("Best Effective Power (W)",
        0)
                           for seg in segments}
82  hydrogen_rate_map   = {seg: off_design_results.get(seg, {}).get("Best Hydrogen Used (kg/s)",
        0)
                           for seg in segments}
84
    print("Effective powers by segment:")
86  for seg in segments:
        print(f"  {seg}: {effective_power_map[seg]} W")
```

```
print("Hydrogen consumption rates by segment:")
for seg in segments:
    print(f"  {seg}: {hydrogen_rate_map[seg]} kg/s")



effective_power_detailed = np.array([
    0 if seg == "Idle" else off_design_results.get(seg, {}).get("Best Effective Power (W)",
    0)
    for seg in flight_condition_ts
])
hydrogen_consumption_detailed = np.array([
    0 if seg == "Idle" else off_design_results.get(seg, {}).get("Best Hydrogen Used (kg/s)",
    0)
    for seg in flight_condition_ts
])


# Pw_P is in kW; convert to Watts.
power_deficiency_detailed = np.array(Pw_P) * 1000 - effective_power_detailed
power_deficiency_detailed = np.maximum(power_deficiency_detailed, 0)
# power_deficiency_detailed = power_deficiency_detailed / (PE_converter_efficiency**2)


#  Plot Power Deficiency
plt.figure(figsize=(10, 4))
plt.plot(TimeP, power_deficiency_detailed / 1000, '-', label='Detailed Power Deficiency (kW)'
    )
plt.xlabel("Detailed Time (s)")
plt.ylabel("Power Deficiency (kW)")
plt.title("Detailed Power Deficiency vs Time")
plt.legend()
plt.grid(True)
plt.show()

# For detailed, assume each time step is 1 second.
total_hydrogen_detailed = np.sum(hydrogen_consumption_detailed)
print("Total hydrogen consumed (detailed): {:.4f} kg".format(total_hydrogen_detailed))

# Obtain the flight mach number and air properties at off-design conditions
flight_mach = fr.get_flying_Mach(Zh_ts, v_x_ts)
_, Tt0, _, _, _, _  = fr.calulate_air_properties(Zh_ts, flight_mach)



# # # Read the GSP data

rows_to_skip = list(range(2, 989, 17)) + [989, 990, 991, 992] # NaN rows excluding the design
     point

df = read_GSP('DATA_GT3.csv', rows_to_skip)

Altitude, Mach, Tt4, PWshaft, WF, W, tt3 = get_data(df)

# Prepare data for interpolation
points = df[['Macha\r\n[-]', 'Tt4\r\n[K]', 'Zp\r\n[m]']].values
valuesWf = df['WF\r\n[kg/s]'].values
valuesPWshaft = df['PWshaft\r\n[kW]'].values
valuesTt3 = df['Tt3\r\n[K]'].values

gt_perf_data_detailed = []  # This list will store one dictionary per detailed time step.
# A simple cache for turbine shaft power results.
pwshaft_cache = {}

def get_cached_pwshaft(flight_mach, Tt4, altitude):
    key = (round(flight_mach, 3), Tt4, round(altitude, 1))
    if key in pwshaft_cache:
        return pwshaft_cache[key]
    else:
        result = interpolate_PWshaft(flight_mach, Tt4, altitude)
        pwshaft_cache[key] = result
```

```
        return result

for i in range(len(TimeP)):
    deficiency = power_deficiency_detailed[i]  # in Watts
    altitude = Zh_ts[i]          # detailed altitude
    v_x_value = v_x_ts[i]        # detailed horizontal speed
    flight_mach = fr.get_flying_Mach(altitude, v_x_value)
    t = 1.0  # assume each detailed time step is 1 second

    # Create a dictionary for GT performance at this detailed time step.
    gt_data = {
        'time': TimeP[i],
        'altitude': altitude,
        'flight_mach': flight_mach,
        'power_required': Pw_P[i] * 1000,  # in Watts (detailed power requirement)
        'power_deficiency': deficiency,      # in Watts
        'GT_PW': 0,       # Turbine shaft power (W)
        'Tt4': 0,         # Turbine inlet temperature (K); use 0 if no deficiency
        'Tt3': 0,         # Compressor outlet temperature (K)
        'GTfuels': 0,     # Turbine fuel flow rate (kg/s)
        # 'GTfuel': 0,      # Total turbine fuel consumed over this time step (kg)
        'gt_eff': 0,      # Turbine efficiency (dimensionless)
        'battery_power': 0  # Battery supplement power (W)
    }

    if deficiency > 0:
        for Tt4 in range(1000, 1600):  # Candidate turbine inlet temperatures from 1000K to
    1599K
            PW_kW = get_cached_pwshaft(flight_mach, Tt4, altitude)
            PW_GT = PW_kW * 1000 * 0.9
            # PW_GT = interpolate_PWshaft(flight_mach, Tt4, altitude) * 1000 * 0.9
            if PW_GT >= deficiency:
                gt_data['GT_PW'] = PW_GT
                gt_data['Tt4'] = Tt4
                Tt3 = interpolate_Tt3(flight_mach, Tt4, altitude)
                gt_data['Tt3'] = Tt3
                GTfuels = interpolate_wf(flight_mach, Tt4, altitude)
                gt_data['GTfuels'] = GTfuels
                # gt_data['GTfuel'] = GTfuels * t
                gt_eff = calc_gt_efficiency(flight_mach, Tt4, altitude,  PW_GT)
                gt_data['gt_eff'] = gt_eff
                gt_data['battery_power'] = 0
                break
            elif Tt4 == 1550 and PW_GT < deficiency:
                battery_power = deficiency - PW_GT
                gt_data['GT_PW'] = PW_GT
                gt_data['Tt4'] = Tt4
                Tt3 = interpolate_Tt3(flight_mach, Tt4, altitude)
                gt_data['Tt3'] = Tt3
                GTfuels = interpolate_wf(flight_mach, Tt4, altitude)
                gt_data['GTfuels'] = GTfuels
                # gt_data['GTfuel'] = GTfuels * t
                gt_eff = calc_gt_efficiency(flight_mach, Tt4, altitude,  PW_GT)
                gt_data['gt_eff'] = gt_eff
                gt_data['battery_power'] = battery_power
                break
            else:
                print(f"Detailed time {TimeP[i]} s: Candidate Tt4 = {Tt4}K yields PW_GT = {
    PW_GT:.1f} W, insufficient for deficiency {deficiency:.1f} W")
    else:
        gt_data['GT_PW'] = 0
        gt_data['Tt4'] = 0
        gt_data['Tt3'] = 0
        gt_data['GTfuels'] = 0
        # gt_data['GTfuel'] = 0
        gt_data['gt_eff'] = 0
        gt_data['battery_power'] = 0

    gt_perf_data_detailed.append(gt_data)
    print(f"Detailed time {TimeP[i]} s: Altitude {altitude}, Power Required {Pw_P[i]*1000} W,
     Deficiency {deficiency} W")
```

```
print("\nCollected Detailed GT Performance Data:")
for entry in gt_perf_data_detailed:
    print(entry)

# ------------------------------------------------------------------------------
# (Optional) Plot Detailed GT Performance Time Series Using Step Plots
# ------------------------------------------------------------------------------
# Extract arrays for plotting:
detailed_times = np.array([entry['time'] for entry in gt_perf_data_detailed])
detailed_GT_PW = np.array([entry['GT_PW'] for entry in gt_perf_data_detailed])
detailed_Tt4 = np.array([entry['Tt4'] for entry in gt_perf_data_detailed])
detailed_Tt3 = np.array([entry['Tt3'] for entry in gt_perf_data_detailed])
detailed_GTfuel = np.array([entry['GTfuels'] for entry in gt_perf_data_detailed])
detailed_gt_eff = np.array([entry['gt_eff'] for entry in gt_perf_data_detailed])
detailed_battery_power = np.array([entry['battery_power'] for entry in gt_perf_data_detailed
    ])

# plt.figure(figsize=(10, 4))
# plt.plot(detailed_times, detailed_GT_PW / 1000, '-', label="GT Shaft Power (kW)")
# plt.xlabel("Time (s)")
# plt.ylabel("GT Shaft Power (kW)")
# plt.title("Detailed GT Shaft Power vs. Time")
# plt.legend()
# plt.grid(True)
#
#
# plt.figure(figsize=(10, 4))
# plt.plot(detailed_times, detailed_Tt4, '-', color='red', label="Turbine Inlet Temp (Tt4, K)
    ")
# plt.xlabel("Time (s)")
# plt.ylabel("Turbine Inlet Temperature (K)")
# plt.title("Detailed Turbine Inlet Temperature vs. Time")
# plt.legend()
# plt.grid(True)
#
#
# plt.figure(figsize=(10, 4))
# plt.plot(detailed_times, detailed_Tt3, '-', color='green', label="Compressor Outlet Temp (
    Tt3, K)")
# plt.xlabel("Time (s)")
# plt.ylabel("Compressor Outlet Temperature (K)")
# plt.title("Detailed Compressor Outlet Temperature vs. Time")
# plt.legend()
# plt.grid(True)
#
#
# plt.figure(figsize=(10, 4))
# plt.plot(detailed_times, detailed_GTfuel, '-', color='purple', label="GT Fuel Consumption
    rate (kg/s)")
# plt.xlabel("Time (s)")
# plt.ylabel("GT Fuel Consumption (kg)")
# plt.title("Detailed GT Fuel Consumption vs. Time")
# plt.legend()
# plt.grid(True)
#
#
# plt.figure(figsize=(10, 4))
# plt.plot(detailed_times, detailed_gt_eff, '-', color='orange', label="GT Efficiency")
# plt.xlabel("Time (s)")
# plt.ylabel("GT Efficiency")
# plt.title("Detailed GT Efficiency vs. Time")
# plt.legend()
# plt.grid(True)

# --- Figure 1: Two stacked subplots ---
fig1, (ax1, ax2) = plt.subplots(2, 1, figsize=(8, 6))

# Subplot 1 (top): GT Shaft Power vs. Time
ax1.plot(detailed_times, detailed_GT_PW / 1000, '-', label="GT Shaft Power (kW)")
ax1.set_xlabel("Time (s)")
```

```
ax1.set_ylabel("GT Shaft Power (kW)")
ax1.set_title("Detailed GT Shaft Power vs. Time")
ax1.legend()
ax1.grid(True)

# Subplot 2 (bottom): Turbine Inlet Temperature vs. Time
ax2.plot(detailed_times, detailed_Tt4, '-', label="Turbine Inlet Temp (Tt4, K)")
ax2.set_xlabel("Time (s)")
ax2.set_ylabel("Turbine Inlet Temperature (K)")
ax2.set_title("Detailed Turbine Inlet Temperature vs. Time")
ax2.legend()
ax2.grid(True)

# Adjust spacing to avoid overlap
fig1.tight_layout()

# --- Figure 2: Two stacked subplots ---
fig2, (ax3, ax4) = plt.subplots(2, 1, figsize=(8, 6))

# Subplot 1 (top): Compressor Outlet Temperature vs. Time
ax3.plot(detailed_times, detailed_Tt3, '-', label="Compressor Outlet Temp (Tt3, K)", color='
    green')
ax3.set_xlabel("Time (s)")
ax3.set_ylabel("Compressor Outlet Temperature (K)")
ax3.set_title("Detailed Compressor Outlet Temp vs. Time")
ax3.legend()
ax3.grid(True)

# Subplot 2 (bottom): GT Fuel Consumption vs. Time
ax4.plot(detailed_times, detailed_GTfuel, '-', label="GT Fuel Consumption (kg/s)", color='
    purple')
ax4.set_xlabel("Time (s)")
ax4.set_ylabel("GT Fuel Consumption (kg/s)")
ax4.set_title("Detailed GT Fuel Consumption vs. Time")
ax4.legend()
ax4.grid(True)

# Adjust spacing
fig2.tight_layout()

# --- Figure 3: Single subplot ---
fig3, ax5 = plt.subplots(figsize=(8, 4))

# Plot: GT Efficiency vs. Time
ax5.plot(detailed_times, detailed_gt_eff, '-', label="GT Efficiency", color='orange')
ax5.set_xlabel("Time (s)")
ax5.set_ylabel("GT Efficiency")
ax5.set_title("Detailed GT Efficiency vs. Time")
ax5.legend()
ax5.grid(True)

plt.show()

# The weight calculations for the entire aircraft.
tg_shaft_power = detailed_GT_PW.max()
tg_fuel = np.sum(detailed_GTfuel)
total_fuel = tg_fuel + total_hydrogen_detailed
gt_dc_converter = power_deficiency_detailed.max()/ (PE_converter_efficiency**2)
inverter_power =  power_deficiency_detailed.max()/ (PE_converter_efficiency)
rectifier_power = power_deficiency_detailed.max()/ (PE_converter_efficiency**2)


H2_LHV = 33.3       # kWh/kg

print("\n--- Performance Metrics ---")
print("Fuel cell Hydrogen Consumption (kg):", total_hydrogen_detailed)
print("Turbogenerator Hydrogen Consumption (kg):", tg_fuel)
print("Total Hydrogen Consumption (kg):", total_fuel)
print("Fuel Cell Hydrogen Energy Equivalent (kWh):", total_fuel* H2_LHV)
```

```
358 weight_breakdown = weight_hps_tg_breakdown(fuel_cell_mass, total_hydrogen_detailed,
        fc_compressor, fc_hex, tg_shaft_power, tg_fuel, gt_dc_converter,rectifier_power,
        inverter_power)

360 plot_weight_breakdown(weight_breakdown)
```

main_fc_gt.py