AN ABSTRACT OF THE THESIS OF

Derek M. Jackson for the degree of Master of Science in Electrical and Computer Engineering presented on June 11, 2021.

Title: Multi-Objective Power Electronics System Design and Optimization, A Machine Learning Approach

Abstract approved: _____

Yue Cao

The integration of power electronics within the energy and transportation sectors enlists a demand for the rapid development of energy-efficient electrical systems. While model-based design and physics-based simulations are effective ways to handle the multi-disciplinary and multi-objective (MO) design of these complex systems, design exploration remains a time-consuming procedure. A recently developed machine learning (ML) framework that outperforms other optimization algorithms in both accuracy and speed is one promising solution. This thesis presents the integration of the ML framework into the MO design process for power electronic systems. The autonomy and efficiency of the ML approach enables development of low-cost and energy-efficient systems by reducing the required time and resources. A discussion of electrical system design, modeling, and optimization theory will lay the groundwork for demonstrating the proposed ML approach.

Multi-Objective Power Electronics System Design and Optimization, A Machine Learning Approach

by

Derek M. Jackson

A THESIS

submitted to

Oregon State University

in partial fulfillment of
the requirements for the
degree of

Master of Science

Presented June 11, 2021
Commencement June 2021

Master of Science thesis of Derek M. Jackson presented on June 11, 2021.

APPROVED:

Major Professor, representing Electrical and Computer Engineering

---

Head of the School of Electrical Engineering and Computer Science

---

Dean of the Graduate School

I understand that my thesis will become part of the permanent collection of Oregon State University libraries. My signature below authorizes release of my thesis to any reader upon request.

Derek M. Jackson, Author

ACKNOWLEDGEMENTS

I would like to acknowledge all who have helped make this possible. I'd like to thank my advisor Dr. Yue Cao for his guidance throughout these past two years. Much appreciation is given to Alastair Thurlbeck of Oregon State University for his efforts in the development of the UAV power system base models. Finally, I'd like to acknowledge Syrine Belakaria and Dr. Janardhan Doppa of Washington State University who developed the machine learning algorithm discussed in this thesis.

TABLE OF CONTENTS

TABLE OF CONTENTS (Continued)

LIST OF FIGURES

LIST OF FIGURES (Continued)

LIST OF TABLES

LIST OF APPENDIX FIGURES

Chapter 1: Introduction

The integration of power electronics within the energy and transportation sectors enlists a demand for rapid development of energy efficient electrical systems. However, many prohibitive challenges arise in the design of such complex systems. For example, electric vehicles, fast vehicle charging stations, and renewable energy sources include many interacting subsystems that span across multiple disciplines and requires collaboration between various domain experts. Design specifications for individual subsystems also become dependent on the system as a whole. The automotive and aerospace industries, who are both known for their large and complicated systems, take a Multidisciplinary Design Optimization (MDO) approach to development. The MDO process requires exploration of numerous design decisions, which in the past required hardware prototyping – a time-consuming and expensive procedure – and could often lead to a dead end. This iterative process is accelerated when mathematical models representing real-physical systems and behaviors are utilized to simulate a design's effectiveness – a core element of model-based design. Model-based design aims to fast track the development process by efficiently breaking down the design problem in a hierarchical way and employ computer simulations for design exploration [1]. Through efficient engineering comes the realization of optimally low-cost and high-energy efficient power electronic systems.

While simulations are more efficient than hardware prototyping, design exploration remains a time-consuming procedure. To enable faster development times, autonomous and efficient methods to explore optimal designs are needed. This thesis focuses on formulating the design automation problem and increasing optimization efficiency at the system level.

Optimization methods capable of efficiently searching through a large design space reduce the need for human interference. Engineers make decisions that limit the design space size, or otherwise it is computationally unreasonable to exhaustively search a complete design space. By shifting the decision-making process to an optimization algorithm, design automation can be achieved while also enabling the efficient search of large design spaces. Hence, effective design automation is dependent on the performance and efficiency of the optimization algorithm.

Recent theoretical advances in machine learning (ML) for optimization have made it a promising candidate for effective design automation. The ML algorithm explored in this thesis is capable of searching through a constraint-heavy design space to efficiently discover Pareto optimal designs (optimal designs considering multiple conflicting objectives). Without loss of generality, this thesis targets the design of a vertical-takeoff-landing (VTOL) heavy-duty all-electric unmanned aerial vehicle (UAV) power system. However, the proposed ML-based power system design framework is general and can be used for various complex applications, such as more electric aircraft, on/off-road vehicles, ships, grid-connected buildings, renewable energy systems, etc. Using a UAV system, this thesis demonstrates the efficacy of the approach, especially the drastic reduction of the number of simula-

tion iterations towards converging to Pareto optimal designs. Experimental results demonstrate the ML algorithms consistent performance over a Genetic Algorithm (GA) and ML algorithm competitor in both Pareto front quality and convergence rate, where in one trial the optimal Pareto front is discovered after exploring only 4% of the design space.

This thesis begins with a discussion of the design and optimization process for electrical power systems. Following is a review of optimization algorithms used for power electronic system level design. Chapter 2 ends with an in-depth look into the proposed ML algorithm process as well as an overview of the family of GAs. Chapter 3 defines the multi-physics power system models. Simulations using these models are provided in Chapter 4. Finally, Chapter 5 demonstrates system optimization using the novel ML algorithm and two other algorithms for comparison, with a supplementary experiment to emphasize the benefits of the novel ML algorithm.

Chapter 2: Design and Optimization Process

Model-based design begins at the system level, where high-level models of a proposed architecture are developed. The selected optimal architecture and system parameters then serve as the design specifications for each subsystem (e.g., battery pack, DC-AC inverter, electric motor, etc.). An emphasis on system-level functionality is beneficial, as the overall performance of a large power system (e.g., electric vehicles, more-electric aircraft, heavy-duty unmanned aerial vehicles) outweighs the performance of an individual component. For example, selecting a state-of-the-art power electronics converter may not be feasible due to size, weight, or temperature constraints when integrated with the rest of the system.

System-level design relies on simulations to explore feasibility and performance, especially early in the development process. Simulations require the development of models with multi-physics domains (e.g., electrical, mechanical, thermal) that run on multiple timescales (e.g., seconds, milli-seconds, and micro-seconds range) depending on the required fidelity and can be computationally slow. For example, a 300-minute flight in a more electric aircraft thermally integrated power system simulation completes one design candidate in about 15 minutes [2]. Even if simulation times were reduced by an order of magnitude, the high number of design candidates, typically in the order of thousands, can still slow down the total design evaluation. Additionally, modeling uncertainties in design require Monte

Carlo simulations that further increase computation time [3]. Such simulations can take a few hours to several days to explore the entire design space and are usually tailored for a particular drive cycle or mission profile. When the mission profile is replaced, such computations must start over, creating lengthy processes.

2.1 Problem Formulation

In engineering, a design is optimized for a specific set of objectives. Examples of these objectives are efficiency or energy consumption, cost, reliability, and lifetime. These objectives are to either be maximized or minimized. The realization of a designed physical system is also subject to a set of constraints that limits the feasibility of certain designs. Constraints in electrical systems include voltage and current ratings, temperature limits, and energy storage limits. Given these objective functions $F_k(\boldsymbol{x})$, inequality $G_i(\boldsymbol{x})$ and equality $H_j(\boldsymbol{x})$ constraint functions, and constraint limits g_i and h_j, the optimization problem is formally defined by

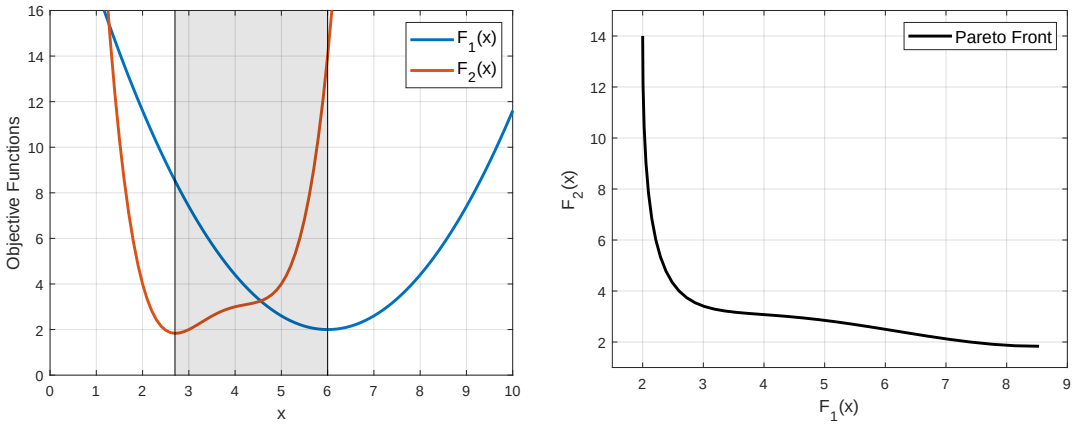
$$\begin{aligned} \min_{\boldsymbol{x}} \quad & F_1(\boldsymbol{x}), ..., F_K(\boldsymbol{x}) \\ \text{s.t.} \quad & G_1(\boldsymbol{x}) \leq g_1 \\ & \vdots \\ & G_I(\boldsymbol{x}) \leq g_I \\ & H_1(\boldsymbol{x}) = h_1 \\ & \vdots \\ & H_J(\boldsymbol{x}) = h_J \end{aligned} \tag{2.1}$$

2.2 Multiple Objectives and Pareto Optimality

When considering the multidisciplinary design of electrical power systems, it is often impossible to optimize all objectives at once due to their conflicting nature, such as minimizing energy consumption, total weight, and cost. This leads to a set of Pareto optimal solutions where an objective cannot be improved without degrading another. This difference between single-objective optimization (SOO) and multi-objective optimization (MOO) is important to distinguish. The goal of SOO is to find the global optimum based on one criterion, whereas, with MOO, a global optimum may no longer exist [4]. Individual objectives that have different global optimizers introduce Pareto optimality, which significantly complicates the optimization process.

An example of this increase in complexity is given in Fig. 2.1, which gives a one-dimensional problem with two polynomial objectives functions to be minimized, $F_1(x)$ and $F_2(x)$. The individual global optimizers for $F_1(x)$ and $F_2(x)$ are located at $x = 6.0$ and $x = 2.7$, respectively. However, when Pareto optimality is introduced, a global optimum no longer exists and instead becomes a range between the two individual global optimizers (shaded region in Fig. 2.1(a)). The resulting Pareto front, where no single solution can be considered better than another, is shown in Fig. 2.1(b).

The two approaches to MOO are to either linearly combine objectives and treat them as a SOO problem or apply metaheuristic methods [5]. The former approach is less favorable as it can lead to aggressive exploitation behavior resulting in sub-



(a) (b)

Figure 2.1: One-dimensional problem with two objective functions (a) and resulting Pareto front (b).

optimal solutions [6]. Reducing a problem into a single objective also requires more human intervention through prioritizing each objective. For the latter approach, algorithms capable of true Pareto front optimization primarily use gradient-free methods, such as genetic algorithms (GAs), particle swarm optimization (PSO), ant colony optimization (ACO), and divided rectangle (DIRECT) [5][7].

2.3 Optimization Algorithms for System-Level Power System Design

The challenges with power electronics system-level design can be attributed to the following:

- Non-linear functionality

- Mixed-integer design parameters
- Non-convex objective space
- Often involves multiple objectives
- Essentially a black-box

Not all optimization algorithms perform well given these traits. Almost all optimization algorithms utilize either deterministic, metaheuristic, or ML methods [5]. Deterministic algorithms often resort to gradient-based methods which fail to generate high-quality solutions with a non-convex objective space that have multiple local optima. Therefore, the gradient-free methods of metaheuristic-based optimization are widely used in the system-level design of electric power systems [7][8]. Traditional optimization methods for electric power systems that consider a single objective or a linear combination of multiple objectives also often use a manual approach [9] or use weighted sums and linear approximations [10][11][12] at the system level. As mentioned in the previous section, these approaches can lead to sub-optimal solutions and can be impractical due to large design spaces. Some popular metaheuristic methods are GA, ACO, and PSO. A Metaheuristic method follows a set of rules based on various concepts (e.g., evolution, ant colonies, etc.), which through trial and error have been found to be effective optimizers. Metaheuristic methods have been used for the design of a solar-powered hybrid airship [13], an all-electric vehicle [14], a solar power conversion system [15], and hybrid electric vehicles [16][17], to name a few.

While these metaheuristic methods have shown to be somewhat effective optimizers, they suffer from the following limitations: 1) requiring a large number of design evaluations, which may not be practical when design simulations are computationally expensive; 2) suffering from convergence related challenges; 3) not always able to uncover the optimal Pareto front [18]. Bayesian optimization (BO) is a ML-based framework that has the potential to overcome the drawbacks of GA, especially in reducing the number of expensive design simulations to discover (approximate) optimal Pareto solutions [19]. While ML models are typically trained with previously generated data, BO instead builds statistical surrogate (representative) models throughout the optimization process, using the knowledge of prior design evaluations to improve these models continuously. These models are employed to intelligently select the sequence of designs for evaluation by maximizing a utility function defined in terms of the learned statistical model's prediction and uncertainty. Optimizing the utility function is a cheaper alternative because evaluating the surrogate models is often less time-consuming than evaluating the physical models.

In the field of electrical engineering, BO algorithms have been used to optimize a converter level design of a multi-output switched-capacitor voltage regulator and achieved a 90% reduction in the number of simulations required to optimize design parameters [20]. However, there has been little research in developing power electronic system-subsystem-oriented design automation tools using ML-based optimization. ML-enabled algorithms aim to further increase the speed of design iterations by efficiently searching through the design space to minimize the number

of computationally-expensive simulations to uncover high-quality Pareto designs.

The following section will delve into the functionality of BO with a focus on a novel BO algorithm called MESMOC. Section 2.5 will provide and overview of the widely popular family of GAs with a focus on the NSGA-II algorithm, as understanding the GA will help to understand the benefits of BO.

2.4 Machine Learning: Bayesian Optimization

Developments in the past twenty years have made BO a strong candidate for efficient global MOO, with notable early implementations being ParEGO [21] and SMS-EGO [22]. The key idea of BO is to build statistical models of objective functions to select the sequence of designs for evaluation based on probabilistic calculations. The information inferred from the statistical models can predict a designs performance before directly evaluating the objective functions. However, this ability comes with a cost. Compared to other algorithms, such as GA, PSO, and SA, the BO algorithm time complexity is large. Thus, for simple optimization problems that have fast function evaluation times there is little benefit to using BO. When function evaluations become time-expensive, BO becomes a strong candidate for the optimization process.

A high-level overview of the BO process will first be given, keeping the explanation generic to all multi-objective BO. The fundamental theories behind BO will then be discussed. Finally, a novel BO algorithm called MESMOC will be explained. MESMOC is later compared to an alternative BO algorithm called

PESMOC and the NSGA-II algorithm in Section 5.4.

2.4.1 High-Level Design Process

This overview is generalized to apply to all multi-objective BO algorithms. However, due to variations in the BO process, it most accurately reflects the MESMOC algorithm which is discussed in-depth in Section 2.4.3.

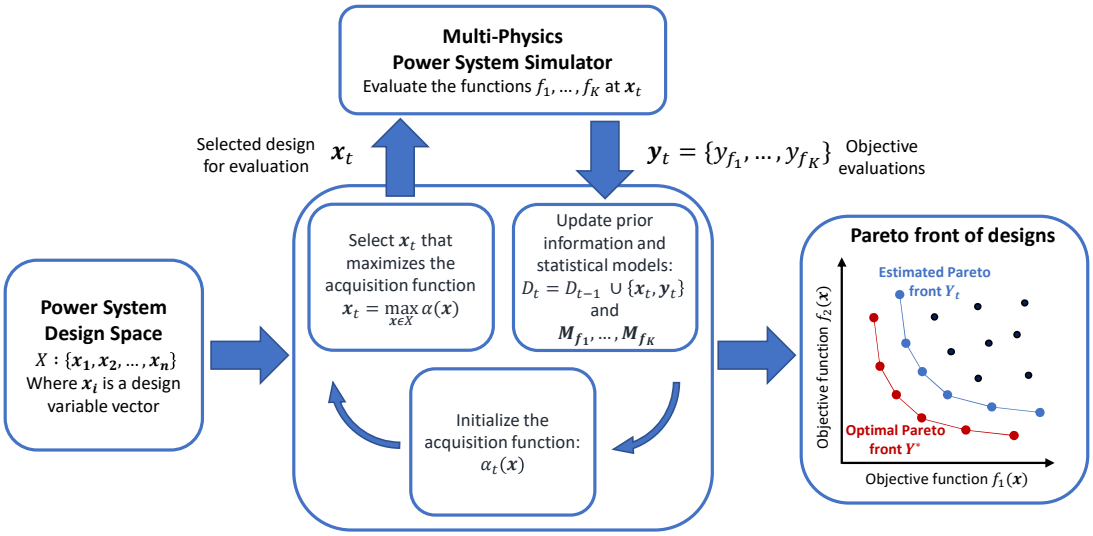


Figure 2.2: A high-level overview of the ML process for power system design and optimization.

A high-level overview of the proposed design process is presented in Fig. 2.2, which will be referred to throughout this section. The design process begins by constructing subsystem models of a electrical power system, which in this case is the UAV to be used in a mission-based simulation. A mixed-fidelity modeling (e.g., static, dynamic, quasi-dynamic) approach may be chosen, depending on the

design objectives and level of details. The ML algorithm discussed suits a variety of modeling methods, as they are treated as black-box functions. This physical modeling part is reflected in the upper box "multi-physics power system simulator" in Fig. 2.2.

Next comes the design variable selection and evaluation process. A design variable vector $\boldsymbol{x}_t$ represents the set of parameters used in each power subsystem model (e.g., battery pack voltage and capacity, motor quantity, etc.), where t denotes the iteration number. The design space X, is the set of all possible design variable vectors $X : \{\boldsymbol{x}_1, \boldsymbol{x}_2, ..., \boldsymbol{x}_n\}$, where n is the design space size, which means all possible combinations of individual physical parameter choices. This design space is defined upfront, as illustrated in the left box of Fig. 2.2. The vital role of BO is to intelligently select the proper $\boldsymbol{x}_t$ for the next iteration without the need to go through all n design vectors, which will be discussed in the next paragraph and in detail in Section 2.4.2. Referring to the upper side of Fig. 2.2, assuming an $\boldsymbol{x}_t$ is chosen, then the power system simulation yields an output vector, $\boldsymbol{y}_t = F(\boldsymbol{x}_t)$, where F is the black-box function defined by the physical models. To be specific, given K design objectives the power system simulation output is $\boldsymbol{y}_t = \{y_{f_1}, ..., y_{f_K}\}$. An objective function f_k can be any design evaluation metric to be optimized such as energy consumption, system weight, cost, or reliability. The latest input $\boldsymbol{x}_t$ and output $\boldsymbol{y}_t$ joins a pool of all previous evaluated inputs and outputs D_t (i.e., $D_t = D_{t-1} \cup \{\boldsymbol{x}_t, \boldsymbol{y}_t\}$). This entire pool, known as prior information, is used by the ML algorithm to intelligently select a new design variable $\boldsymbol{x}_t$ for the next iteration.

Inside the design variable selection box (center of Fig. 2.2), the proposed ML

process utilizes statistical models, represented as $\boldsymbol{M_{f_k}}$, to learn the true mapping function from the input parameters to the design objectives. The statistical models provide (Gaussian) probability distributions, based on the prior information, of the objective and constraint values for each design candidate in X. The statistical models and prior information are then used to construct an acquisition function $\alpha_t(\boldsymbol{x})$. The acquisition function is a critical element of a BO algorithm. The information $\alpha_t(\boldsymbol{x})$ provides is what guides BO to intelligently select the next design candidate $\boldsymbol{x}_t$ to evaluate. The different types of acquisition functions are discussed in Section 2.4.2. The next simulated design $\boldsymbol{x}_t$ is selected by maximizing the acquisition function which ensures the best design is evaluated (based on the $\alpha_t(\boldsymbol{x})$ criteria) in the next iteration.

Then this $\boldsymbol{x}_t$ feeds to the next iteration of the multi-physics power system simulator as discussed previously. After the design $\boldsymbol{x}_t$ is evaluated, the results $\boldsymbol{y}_t$ are used to update statistical models $\boldsymbol{M_{f_k}}$. The process is then repeated. Throughout the optimization process, the estimated (known) Pareto front $\boldsymbol{Y_t}$ is continuously updated according to the new information and will approach the optimal Pareto front $\boldsymbol{Y^*}$. After a specified number of iterations, the final Pareto set is ready to review, as illustrated in the right side of Fig. 2.2.

2.4.2 Components of Bayesian Optimization

The three main components of BO are the statistical models, acquisition function, and the acquisition optimization procedure. All three components are summarized

below.

Statistical Models: At the core of BO is the well-known Bayes' theorem, defined as

$$p(w|D) = \frac{p(D|w)p(w)}{p(D)} \tag{2.2}$$

and where $p(w)$ is the prior probability distribution of w, $p(D|w)$ is the likelihood of the collected data D given w, and $p(D)$ is the distribution of D [19]. It states that given the *a priori* information of $p(w)$ and likelihood model $p(D|w)$, a *posterior* probability of w can be found. In terms of optimization, given the current knowledge of the objective functions from previously evaluated designs and an assumed function model, a prediction of the objective function values for a new design can be made [23].

The most common statistical model used in BO is the Gaussian Process (GP). This is because special properties of the GP allow for simplified calculations, such as a closed form equation for the marginal and conditional likelihood [24]. By assuming all points of an objective function $f(\cdot)$ are jointly distributed with zero mean, the probability distribution is given as

$$\boldsymbol{f} \sim \mathcal{N}(\mathbf{0}, K) \tag{2.3}$$

where $\boldsymbol{f}$ is a vector of all previously evaluated values of the objective function $\{f_1, ..., f_n\}$ *and* the desired objective function predictions $\{\hat{f}_1, ..., \hat{f}_i\}$, and K is the covariance kernel matrix. Based on the minimum mean square estimator (mmse) for joint Gaussian distributions, the posterior mean $\hat{\mu}(\hat{\boldsymbol{x}})$ and variance $\hat{\sigma}^2(\hat{\boldsymbol{x}})$ for a

point $\hat{\boldsymbol{x}}$ are given in (2.4) and (2.5), respectively [24].

$$\hat{\mu}(\hat{\boldsymbol{x}}) = k(\hat{\boldsymbol{x}}, \boldsymbol{x})k(\boldsymbol{x}, \boldsymbol{x})^{-1}\boldsymbol{f}(\boldsymbol{x}) \tag{2.4}$$

$$\hat{\sigma}^2(\hat{\boldsymbol{x}}) = k(\hat{\boldsymbol{x}}, \hat{\boldsymbol{x}}) - k(\hat{\boldsymbol{x}}, \boldsymbol{x})k(\boldsymbol{x}, \boldsymbol{x})^{-1}k(\boldsymbol{x}, \hat{\boldsymbol{x}}) \tag{2.5}$$

In (2.4) and (2.5) is the covariance kernel function $k(\cdot, \cdot)$. This kernel function is what defines the general shape or smoothness of the objective function model. While a handful of kernel functions exist, the squared exponential covariance kernel function is popular for design optimization because of the smoothness of the objective functions [25]. With this GP-based statistical model, predictions of yet to be evaluated designs can be made.

Acquisition Functions: The acquisition function is what varies the most between BO algorithms. The purpose of an acquisition function is to quantify the usefulness of evaluating a design candidate. A good acquisition function carefully balances the trade-off between exploration and exploitation to efficiently discover optimal points without missing any points [23]. A common approach of an acquisition function is to based it on improvement. This includes the probability of improvement (PI) and expected improvement (EI) functions [25]. Another popular approach utilizes an upper confidence bound (UCB) [19]. More advanced acquisition functions take an information theory approach such as max-value entropy search (MES) [26][27] and predictive entropy search (PES) [28]. While all types mentioned have been effective in various applications, not all are capable of handling multiple objectives. Even more restricting, only two acquisitions functions

(at the time of this writing) have been shown to handle multiple objectives *and* constraints, namely MESMOC [27] and PESMOC [29]. MESMOC is discussed in more detail in Section 2.4.3.

MESMOC and PESMOC both include additional steps to initialize the acquisition function. Sample models of objective functions and constraints are first constructed using Monte Carlo sampling techniques such as Thompson sampling and Fourier features on the statistical models [30]. Sample Pareto fronts are then generated using the sampled models with a traditional optimization algorithm like DIRECT or GA. The sample Pareto fronts are used within the acquisition function to quantify the expected entropy reduction from evaluating the design.

Optimization Procedure: The next candidate design is selected through maximizing the acquisition function and is demonstrated in Fig. 2.3. In Fig. 2.3 the objective function ($f(\cdot)$) and its statistical model are shown on top, with the acquisition function ($u(\cdot)$ in this figure) shown on bottom of each plot. Given the previously observed/evaluated points, the acquisition function quantifies the usefulness of evaluating $f(\cdot)$ at any point. To maximize the acquisition function will thus ensure the most information is gained from the next design simulation. Optimization of the acquisition function can be performed with any global optimization algorithm but a popular approach is to use the deterministic algorithm DIRECT [6].

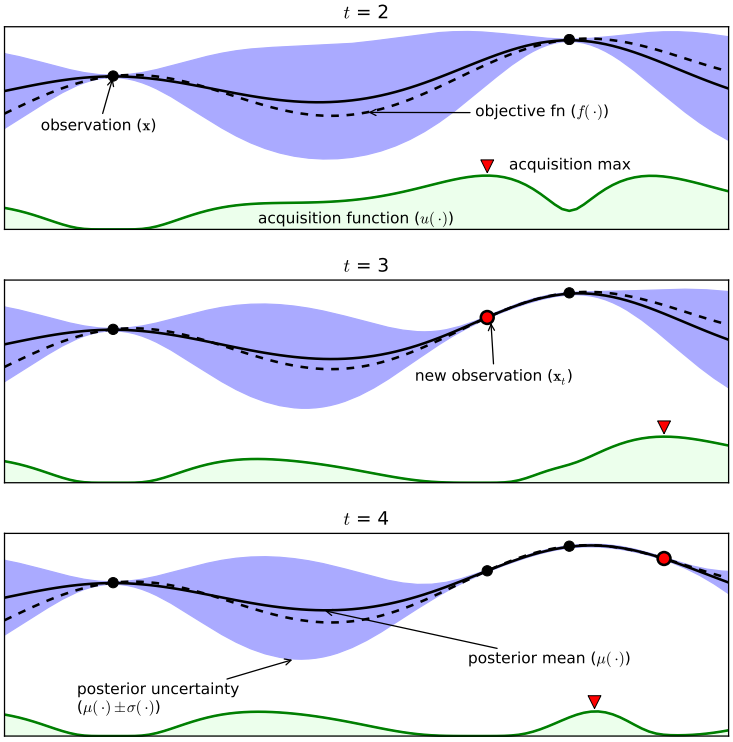


Figure 2.3: Illustration of how the acquisition function quantifies the usefulness of evaluating a design candidate for objective function maximization. In each iteration, the objective function is evaluated at the acquisition function maximum. The process is repeated given the posterior information [23].

2.4.3 MESMOC

The novel ML-based optimization algorithm, namely Max-value Entropy Search for Multi-objective Optimization with Constraints (MESMOC), is capable of searching through a constraint-heavy design space to efficiently discover Pareto optimal designs and first presented in [27]. Experiments using a prior version of the ML algorithm without constraints, i.e., MESMO, consistently outperform state-of-the-art algorithms at providing an accurate, computationally efficient, and robust optimization solver [6]. While the entire optimization procedure is called MESMOC, it specifically refers to the acquisition function used in the more general BO procedure. The key idea is to build statistical models of both design objectives and constraints and use the models to select the sequence of designs for evaluation based on the information-theoretic principle of output space entropy search: maximize the information gain about the optimal Pareto front.

The BO design process overview of Fig. 2.2 is modified in the following way for MESMOC. The power system simulator output vector now becomes $\boldsymbol{y}_t = \{y_{f_1}, ..., y_{f_K}, y_{c_1}, ..., y_{c_K}\}$. In addition to the K design objectives are L constraints that the system must satisfy. A constraint function C_l is a metric that limits the physical realization of a design such as component temperatures, battery state, or spatial limits. Statistical models of the constraint functions, M_{c_l} are created along with the objective function models M_{f_k}.

The acquisition function initialization is now expanded into a cheap MO problem in order to create sample Pareto fronts, as discussed in the previous section.

Approximations of the objectives $(\tilde{f}_1, ..., \tilde{f}_K)$ and constraints $(\tilde{C}_1, ..., \tilde{C}_L)$ are generated by sampling from these distributions via random Fourier features. Each objective and constraint are often approximated multiple times to create a set of sampled functions. With these sampled functions, a set of approximate Pareto fronts Y_s^* are generated by solving multiple cheap (fast) MOO problems with a GA. This procedure is considered cheap because the sampled functions provide quick evaluations compared to the power system simulation. The approximated Pareto fronts Y_s^* and previously evaluated designs D_t are then used to construct the acquisition function $\alpha(\boldsymbol{x})$, which infers the potential entropy reduction about the (real) optimal Pareto front Y^* for a given design $\boldsymbol{x}$.

2.5 Genetic Algorithms

The family of GAs have been widely used for MO power system optimization, as discussed previously in Section 2.3. The reason being that many GAs can handle the mixed-integer design space and is capable of globally optimizing complex black-box functions [31]. All GAs take a metaheuristic approach to optimization that mimics the evolutionary process of natural selection. Instead of iteration-ally progressing to the optimal solutions, they generation-ally converge to the optimal solutions. During each generation a set of designs called a population produce offspring and mutate in a similar fashion to real evolution. Whether a design produces offspring or mutates depends on their fitness value and stochastic processes. Given the proper fitness metric and enough generations, the GA will often converge

to a set of optimal or near-optimal solutions.

GAs differentiate between each other based on the type of genetic operators used. These operators are defined as selection, crossover, and mutation [31]. The selection technique uses a fitness value and determines whether a design will reproduce. The crossover operator determines how two designs are combined to create the offspring, sometimes including randomness as a factor. The third operator, mutation, is what ensures design exploration occurs. Using stochastic processes, some designs within the population will randomly change design parameters.

While the metaheuristics of a GA provides high-probability of converging to well-performing designs, optimal solution (or Pareto front) convergence is not guaranteed. There is no intelligent component that is learning and searches for the optimal solution.

In the following section the genetic operators and functionality of the famous NSGA-II is discussed.

2.5.1 NSGA-II

The Non-dominated Sorting Genetic Algorithm II (NSGA-II) is a fast and elitist multi-objective algorithm [32]. The concept of Pareto non-domination rank partially quantifies the fitness of a certain design within the population. Given two points, A and B, point A is non-dominated by point B if and only if point A outperforms point B in at least one objective. The lower the rank, the more points are dominated by that point. A visualization of Pareto ranking is shown in

Fig. 2.4, where the line closest to the origin has the lowest rank (for a minimization problem). Each line represents a set of points which are non-dominated by each other.

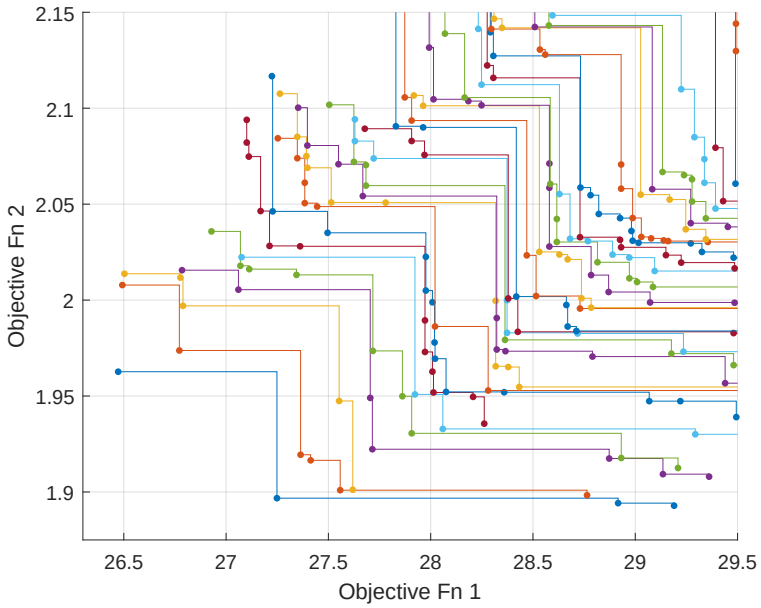


Figure 2.4: Example sets of Pareto non-domination ranks. Moving towards the origin, each staircase line is dominated by the next line.

What makes NSGA-II differ from other non-domination ranking GAs is the crowding distance function [32]. The crowding distance represents the average distance between a single point and all neighboring points in the objective space. Giving a lower rank to points which have a larger crowding distance encourages a more uniformly spread and diverse Pareto front.

NSGA-II uses the binary tournament selection technique to determine which designs will be used to generate the offspring. The crossover and mutation operators used in the original NSGA-II implementation are single-point and bitwise,

respectively [32]. However, their functionality will not be discussed here.

Chapter 3: Power System Modeling

The ML algorithm must be integrated with a physical model, i.e., the domain knowledge. A high-level static (or averaged) model and dynamic model are developed for each component in the power system, both including multiple physics domains. The two models are developed to show how model fidelity can effect the system level optimization process and results. Plenty of research exists on the development and experimental validation of both static and dynamic models of power subsystems. For example, a multi-timescale parametric electrical battery model is described in [33], and [2][34][35] demonstrate the integration of multiple subsystems for electric transportation power systems. Once the physical models are combined to form the desired system architecture, the ML algorithm can treat the simulation as a black-box function where the outputs are optimization objectives and evaluation of constraints, and the inputs are the design parameters. Both models have been implemented in MATLAB/Simulink and the corresponding code and block diagrams can be found in Appendix A and B.

3.1 High-Level Operation

The UAV system architecture consists of a central Li-ion battery pack, hex-bridge DC-AC inverters, permanent magnet synchronous machines (PMSM), and neces-

sary wiring, as shown in Fig. 3.1 for the static model. A set of variable design parameters, such as the battery pack configuration and motor size, are included in the system models. This set, known as the design space, will be searched by the ML algorithm to find the optimal designs. The sweeping ranges of the design parameters will be discussed in Section 5.3.

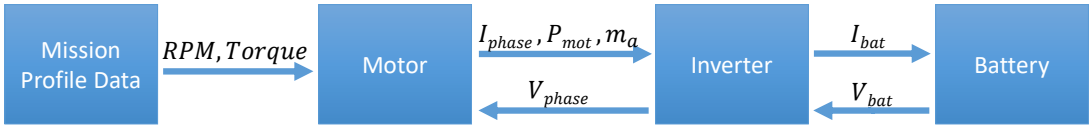


Figure 3.1: Block diagram of the static model UAV power system showing the interfaces between each component model.

The mass of the aircraft frame and its cargo is held constant for all designs. Additional mass is added to the total craft mass depending on the number of cells in the battery pack, motor sizing, and the number of motors. Note that power electronics mass is assumed constant for this study since the semiconductor weight variation is relatively small. Other design details may be included, such as heat sinks or filters. However, this thesis focuses on the development of the ML-physical integrated framework rather than a high-fidelity model.

The system follows a pre-configured mission-profile during a simulation, such as the 30-minute mission shown in Fig. 3.2 that represents a roundtrip flight. The mission-profile is structured as a normalized thrust vs. time array with increments of 1 second in the case of static modeling. Thrust values are scaled by the UAV's total mass and converted into propeller mechanical speed ω_{mech} and torque τ_{mech}, which serve as inputs to the motor model. Thrust to speed and torque conversion

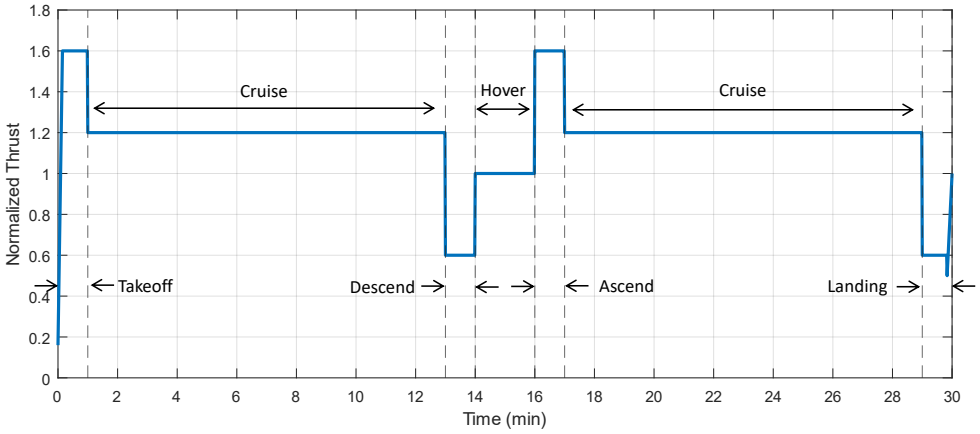


Figure 3.2: Mission-profile used for UAV simulations [27].

is achieved in the same manner as [34], where a propeller-dependent relation is developed. The required motor power characteristics to achieve this mission profile are back-propagated through the motor drive to the battery.

3.2 Static Models

3.2.1 Motor

The motors considered in this design illustration are fixed-phase fixed-pole (e.g., 3ϕ 8-pole) surface-mounted PMSMs. A chosen reference motor provides initial electrical and mechanical parameters. All potential motor designs are scaled from this reference using two design parameters: 1) number of coil winding turns N_{turns}, and 2) height of the stator structure h_{stator}. These two physical parameters are selected because of their influence over the electrical parameters. For example, the

formula for a single coil's inductance,

$$L = \frac{\mu N_{turns}^2}{l} \cdot \pi \left(\frac{h_{stator}}{2} \right)^2 \tag{3.1}$$

is dependent on N_{turns} and h_{stator}, where l is the length of the coil (governing motor diameter) and μ is the magnetic permeability constant. For a given set of N_{turns} and h_{stator}, the thickest wire gauge in AWG for the stator coil is selected while still satisfying the winding fill factor limit. In general, motor design is accomplished by perturbing reference values of stator resistance, synchronous inductance, back EMF constant, and mass using these two design parameters. Fig. 3.3 shows how the motor parameters vary with these two design parameters. The model utilizes the per-phase equivalent circuit of a PMSM motor for all electrical calculations.

The inputs and outputs of the motor static subsystem is given in

$$\begin{aligned} &[I_{phase}, m_a, P_m, P_{m,loss}, T_{m,n+1}] = \\ &\qquad Motor(\omega_{mech}, \tau_{mech}, V_{in}, T_{m,n} | N_{motors}, N_{turns}, h_{stator}) \end{aligned} \tag{3.2}$$

where the inputs are mechanical speed ω_{mech}, mechanical torque τ_{mech}, input voltage from the inverter V_{in}, and motor winding temperature $T_{m,n}$, all under various combinations of number of motors N_{motors}, N_{turns}, h_{stator}. Not all the necessary math will be covered in this thesis; however, an overview is provided to aid comprehension. For a detailed derivation of this model, see [34][35]. For the model outputs, the RMS phase current can be found from the mechanical output power

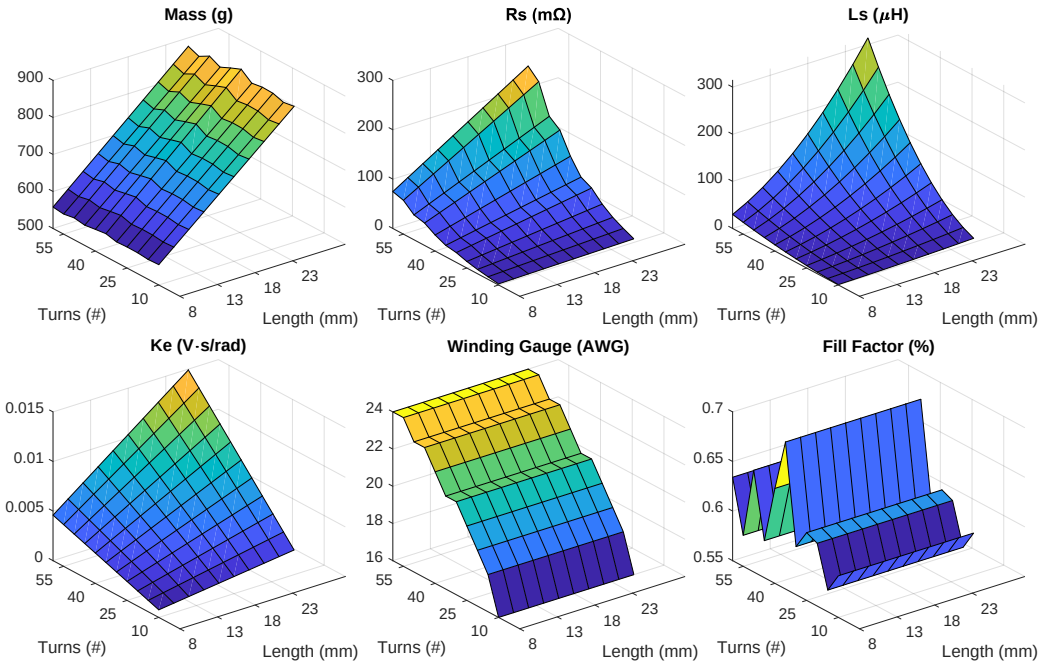


Figure 3.3: Motor parameter perturbations based on the two design parameters h_{stator} and N_{turns}.

and loss using

$$I_{phase} = \frac{\omega_{mech}\tau_{mech} + P_{mech,loss}}{3V_{EMF}} \tag{3.3}$$

The modulation index m_a is given in

$$m_a = \frac{\sqrt{6}V_t}{V_{in}} \tag{3.4}$$

but requires knowledge of the terminal voltage V_t given the back EMF voltage V_{EMF} and voltage drop due to motor impedance. While m_a is calculated in the motor model, it is utilized in the inverter subsystem, to be described in the following subsection. P_m represents the input power to the motor and is found with (3.5), where $P_{m,loss}$, in (3.6), is the combined mechanical and electrical losses.

$$P_m = P_{out} + P_{m,loss} \tag{3.5}$$

$$P_{m,loss} = 3I_{phase}^2 R_s + P_{mech,loss} \tag{3.6}$$

While the static model typically captures steady-state values only, the motor's thermal transients can be accurately captured at a seconds time-scale. Thus, a dynamic thermal model is used. The differential equation of the motor winding's temperature gradient is represented by

$$m_m c_p \frac{dT_m}{dt} = P_{m,loss} + h_{air} A_m (T_a - T_m) \tag{3.7}$$

where $m_m c_p$ is the motor's thermal capacitance, h_{air} is the heat transfer coefficient

based on the Nusselt number, A_m is the surface area of the motor coils, T_a is the ambient temperature, and T_m is the motor temperature.

3.2.2 Power Electronics

The power electronics motor drive is a 3ϕ hex-bridge DC-AC inverter consisting of MOSFET/diode pairs (Fig. 3.4). For this given topology and use of static modeling, two practical design parameters are the MOSFET/diode selection M_{sw}, and the switching frequency f_{sw}. Inverter modeling is based on [34], which uses an averaged switching approach to calculate inverter losses and basic control requirements.

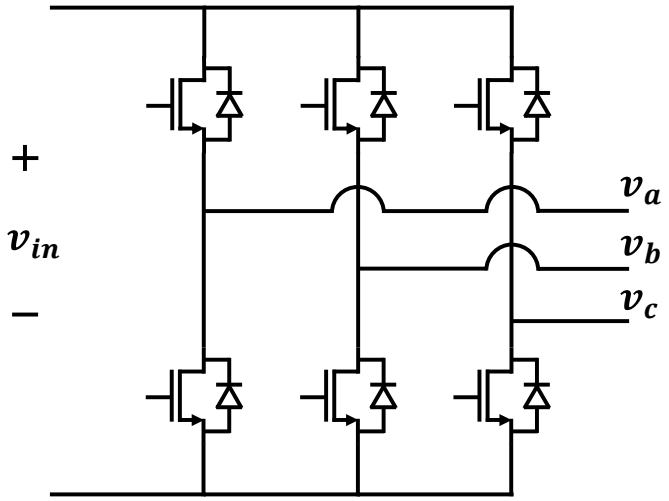


Figure 3.4: Hex inverter circuit.

Similar to the motor subsystem, (3.8) shows the inputs and outputs of the

inverter subsystem, where V_{DC} is the DC bus voltage, I_{phase} is a phase RMS current of the motor stator, m_a is the amplitude modulation index of the SPWM switching scheme, P_m is the motor input power from (3.2), and $T_{i,n}$ is the temperature of the MOSFET/diode junction.

$$[I_{bat}, P_{i,loss}, T_{i,n+1}] = \\ Inverter(V_{DC}, I_{phase}, m_a, P_m, T_{i,n} | f_{sw}, M_{sw}) \tag{3.8}$$

In order to determine the left-hand-side outputs in (3.8), some intermediate loss calculations are required. The conduction loss of a single MOSFET is found with

$$P_{c,m} = 2I_{phase}^2 R_{on} \left[\frac{1}{8} + \frac{m_a \cos\phi}{3\pi}\right] \tag{3.9}$$

which averages the time-varying duty cycle to approximate the losses with a drain-source on-resistance R_{on} and I_{phase} [36]. R_{on} comes from the selected MOSFET datasheet where curve-fitting is used to adjust the on-resistance given at $T_{i,n}$. The MOSFET switching loss is given in

$$P_{sw,m} = f_{sw}(E_{on} + E_{off}) \tag{3.10}$$

which is broken up into turn-on energy (3.11) and turn-off energy (3.12) [37]. Approximating E_{on} and E_{off} requires the reverse recovery charge Q_{rr} and rising/-

falling current and voltage times obtained from the datasheet and [38].

$$E_{on} = V_{DC} \cdot \frac{\sqrt{2} I_{phase}}{\pi} \cdot \frac{t_{ri} + t_{fu}}{2} + Q_{rr} V_{DC} \tag{3.11}$$

$$E_{off} = V_{DC} \cdot \frac{\sqrt{2} I_{phase}}{\pi} \cdot \frac{t_{ru} + t_{fi}}{2} \tag{3.12}$$

Similar to the MOSFET conduction loss, diode conduction loss is given in (3.13) but uses the forward voltage drop rather than the on-resistance. Q_{rr} of the diode is required to solve for the switching power loss using (3.14) and can be found on the datasheet. It is worth noting these approximate loss calculations result in worst case scenario values.

$$P_{c,d} = V_{on} \cdot \sqrt{2} I_{phase} \cdot \left[\frac{1}{2\pi} - \frac{m_a \cos \phi}{8} \right] \tag{3.13}$$

$$P_{sw,d} = \frac{Q_{rr} V_{DC} f_{sw}}{4} \tag{3.14}$$

Total power loss $P_{i,loss}$ of the inverter is found by summing up (3.9)-(3.14) and scaled with the number of switches. The required input current from the battery I_{bat} is derived from the total power into the inverter and V_{DC}.

An aluminum heatsink is assumed to attach all switching devices. The differential equation for the temperature gradient in the heatsink is

$$m_s c_p \frac{dT_s}{dt} = \frac{T_i - T_s}{R_{th,jc} + R_{th,cs}} + \frac{T_a - T_s}{R_{th,sa}} \tag{3.15}$$

were $m_s c_p$ is the heatsink's thermal capacitance, $R_{th,(\cdot)}$ are the thermal resistances,

and T_s, T_i, and T_a are the heatsink, MOSFET/Diode junction, and ambient temperatures, respectively. Assuming zero thermal capacitance of the MOSFET/-Diode, the instantaneous junction temperature T_i is found with

$$T_i = T_s + P_{loss} R_{th,jc} \tag{3.16}$$

3.2.3 Battery

The Li-ion battery pack configuration consists of three variables: 1) number of cells in series N_{series}, 2) number of cells in parallel $N_{parallel}$, and 3) the battery cell model dataset, M_{bat}. The number of cells in series determines the battery pack voltage, whereas the number of cells in parallel indicates the pack's Ah capacity. As individual battery cell current can be determined using $N_{parallel}$, it is only necessary to model a single battery cell. It is assumed that each cell is identical and discharges at the same rate.

Battery cell modeling follows the work [33], where a Randle's equivalent circuit is used (shown in Fig. 3.5). Battery internal voltage V_{int} is parametrically computed as a function of state of charge (SOC) using (3.17) and experimentally gathered coefficients (denoted by a_k). Battery internal impedance (R_{int} and $X_{c,int}$) are calculated in a similar manner as V_{int} using (3.17) but with different coefficients a_k.

$$V_{int} = \exp\left[\sum_{k=0}^{6} a_k \ln^k SOC\right] \tag{3.17}$$

. In [33], the a_k parameters are found over different time scales of seconds, min-

utes, and hours. However, capacitive elements can be ignored for static modeling, allowing simplification of internal resistances into a single constant R_{int}.

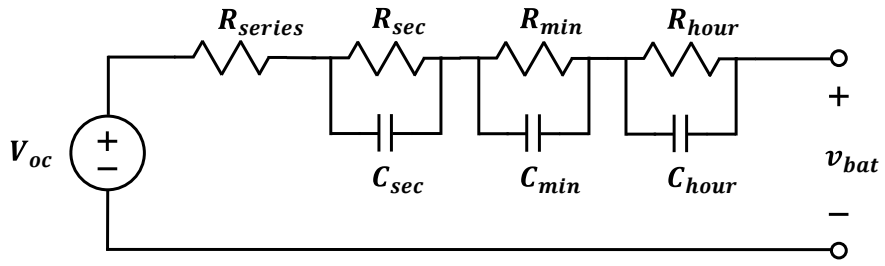


Figure 3.5: Randle's equivalent circuit for a three timescale battery model. All variables are functions of SOC.

A function notation of the battery pack model is given in

$$[V_{bat}, SOC_{n+1}, P_{b,loss}] = Battery(SOC_n, I_{bat}|N_{series}, N_{parallel}, M_{bat}) \tag{3.18}$$

showing the subsystem interface. With known V_{int} and R_{int} and single-cell current demand of the system I_{cell}, battery terminal voltage V_{bat} and power loss $P_{b,loss}$ are found using Ohm's law and I^2R losses, respectively. After every time step of the simulation, the SOC is adjusted given the energy consumption during the last interval. Battery temperature change is not considered here.

3.3 Dynamic Models

The dynamic UAV model shares the same architecture as the static model. An overview of the dynamic model is provided in Fig. 3.6. Considering subsystem

interfaces, the only difference between the static and dynamic models is the motor model output and inverter input, which will be discussed in the following subsection.

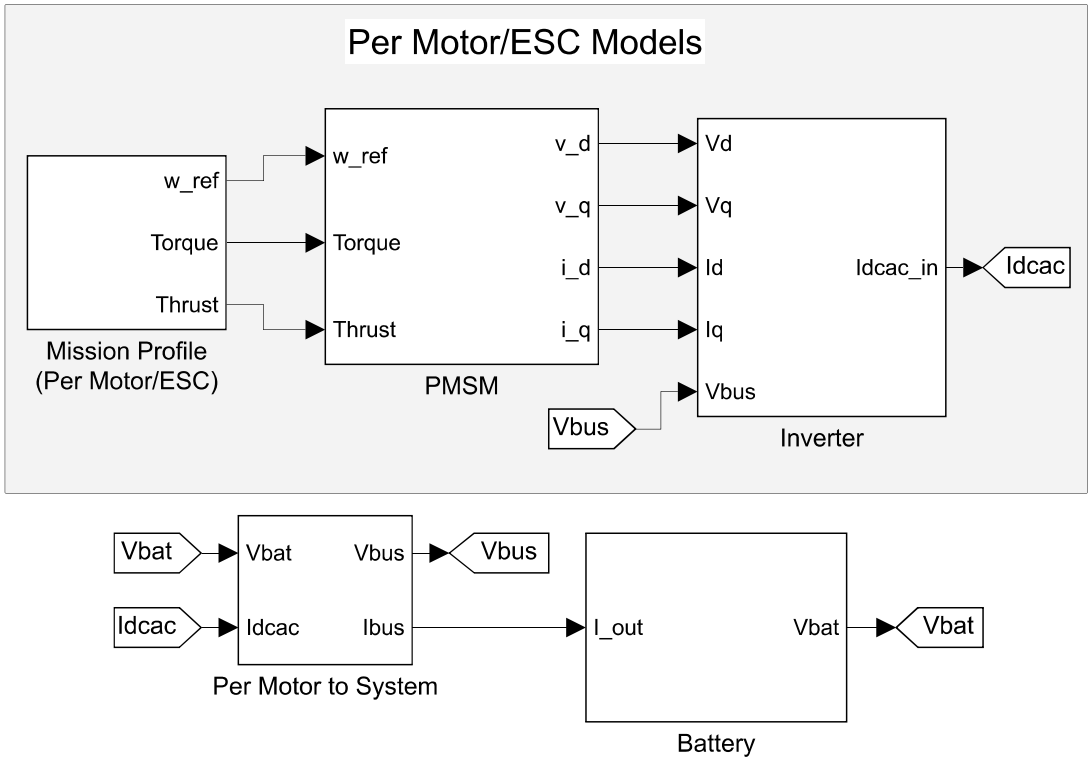


Figure 3.6: Block diagram of the dynamic model UAV power system showing the interfaces between each component model.

3.3.1 Motor

The dynamic motor model is capable of capturing the more realistic behavior of the motor. While the static motor model assumed a zero error between the mission profile and the actual speed and torque, the dynamic model introduces error and

motor controls.

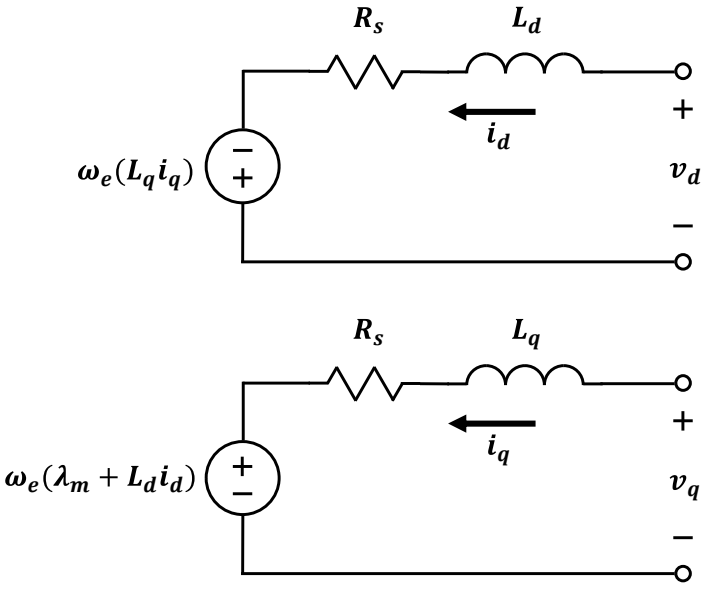


Figure 3.7: PMSM equivalent circuits for the d-axis (top) and q-axis (bottom).

The dynamic motor is modeled in the amplitude-invariant dq-reference frame (the 0-axis is omitted because balanced operation is assumed). The equivalent circuits are given in Fig. 3.7. The accompanying dq-phase terminal voltage equations are

$$v_d = R_s i_d + L_d \frac{di_d}{dt} - \omega_e L_q i_q \tag{3.19}$$

$$v_q = R_s i_q + L_q \frac{di_q}{dt} + \omega_e(\lambda_m + L_d i_d) \tag{3.20}$$

where R_s is the equivalent per-phase resistance, L_d and L_q are the dq phase inductances, ω_e is the electrical frequency (in rad/s), and λ_m is the permanent magnet flux. As this model is for a surface-mounted PMSM, $L_d = L_q$.

The rotational mechanics is modeled with (3.21) where the electromagnetic torque T_{em} is found with (3.22). M_1 and M_2 are mechanical friction losses and were experimentally derived in [35]. T_L represents the load torque from the mission profile, J is the motors rotational inertia, ω_m is the mechanical speed, and p represents the number of poles of the motor.

$$T_{em} - T_L - M_1 - M_2\omega_m = J\frac{d\omega_m}{dt} \tag{3.21}$$

$$T_{em} = \frac{3}{2}\frac{p}{2}\lambda_m i_q \tag{3.22}$$

The motor is controlled using Field-Oriented Control (FOC) technique [35]. FOC utilizes two PI controllers for independent dq current control and one PI controller for speed control. A Simulink implementation of the dynamic motor model can be found in Appendix B.2.

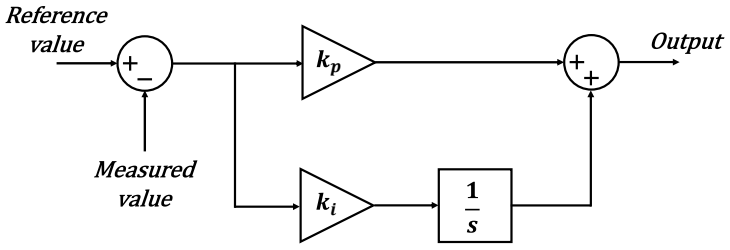


Figure 3.8: PI control loop.

3.3.2 Power Electronics

Inverter input current I_{dcac} along with power losses are calculated the same as the static inverter model. As switching behavior is not of interest or importantance for system level design, the presented average inverter model in Section 3.2.2 is sufficient for the dynamic model as well. The I_{phase} RMS current input given in (3.8) is calculated from the dynamic motor i_d and i_q currents using

$$I_{phase} = \frac{1}{\sqrt{2}}\sqrt{i_d^2 + i_q^2} \tag{3.23}$$

A Simulink implementation of the dynamic inverter model can be found in Appendix B.3.

3.3.3 Battery

When considering the dynamics of the UAV power system, the battery capacitive elements (of the Randles equivalent circuit of Fig. 3.5) must be included in the model. The terminal voltage is now calculated with (3.24), where each $V_{Z(\cdot)}$ is the voltage drop from the corresponding time-scale RC parallel network given in (3.25) and modeled in the frequency domain as Fig. 3.9. The open circuit voltage V_{int}, resistances and capacitances are all derived from (3.17). SOC is calculated the same as the static model. A Simulink implementation of the dynamic battery

model can be found in Appendix B.4.

$$V_{term} = V_{int} - V_{R(ser)} - V_{Z(sec)} - V_{Z(min)} - V_{Z(hour)} \tag{3.24}$$

$$V_{Z(\cdot)} = I_{cell}\left(\frac{R_{(\cdot)}}{sR_{(\cdot)}C_{(\cdot)} + 1}\right) \tag{3.25}$$

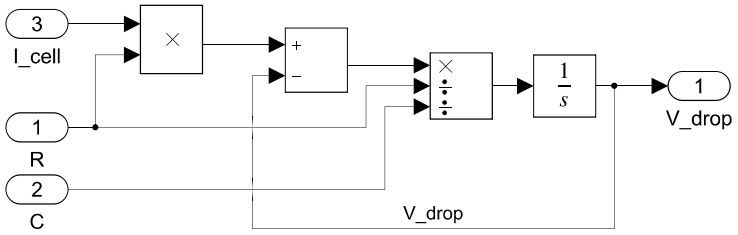


Figure 3.9: Simulink model of the battery voltage drop from an RC parallel network.

3.4 Quasi-Dynamic Models

A quasi-dynamic model of the UAV power system is also developed. It combines the computational efficiency of a static model with the transient accuracy of the dynamic model. A purely dynamic model simulates at a smaller time-scale and often includes more calculations than a static model, which results in more accurate results but also longer execution times. However, the benefits of dynamic modeling only become apparent during acceleration and deceleration stages of flight. In fact, there is little difference between the static and dynamic models during the periods of constant thrust as the dynamic model will quickly settle to a steady-state. The similar behavior of static and dynamic models during periods of constant

thrust can be leveraged in a quasi-dynamic modeling approach. In a quasi-dynamic model, only the transitions between thrust commands are dynamically modeled with the majority of the flight mission statically modeled. Unfortunately, there are implementation challenges involved when transitioning between the two models.

Within the dynamic model exists many state variables that are impertinent to the system behavior. Some examples of state variables include integral blocks within the motor's electrical/mechanical dynamics and controls (see Fig. 3.8), and within the battery impedance calculations (see Fig. 3.9). If a simulation begins in the middle of a mission flight, inaccurate current, voltage, and temperature measurements will be recorded. A simple workaround is to provide an initialization period for the dynamic UAV system by beginning the simulation a few seconds prior to when system information is captured. However, this does not solve all state issues. In the case of the motor, very large error measurements within the speed and current PI controls produce a large current transient if no initial condition is defined for the state variables. This is undesirable when maximum current is a constraint variable. On dynamic simulation start-up, PI control errors can be minimized by configuring the motors initial speed to the commanded motor speed. The combination of initializing the motors speed and an initialization period is sufficient to mitigate any motor measurement inaccuracies.

In the battery model, the voltage drop inaccuracy in each time-scale RC parallel network cannot be resolved with an initialization period of a couple seconds. This is because it takes more than 3 time-constants for a RC network to reach its steady-state value since the time constants are on the seconds, minutes and hours scale.

However, the RC network states can be approximated using

$$V_{Z(\cdot)} = V_{R(\cdot)} \left[1 - \exp\left(\frac{-t_{sim}}{R_{(\cdot)} C_{(\cdot)}} \right) \right] \tag{3.26}$$

where $V_{R(\cdot)}$ is the corresponding static model voltage drop and t_{sim} is the current simulation time. Battery input current and SOC is held constant during the initialization period to mitigate deviation from the true system state.

Chapter 4: UAV Simulations and Results

All power system models were developed in MATLAB/Simulink using the default library packages. The static models solely used MATLAB while the dynamic models were built in Simulink. The quasi-dynamic model ran with MATLAB code with calls to the Simulink dynamic models. The code and block diagrams can be found in Appendix A and B.

4.1 Quasi-Dynamic Simulation

This section demonstrates a UAV power system quasi-dynamic simulation, which captures the steady-state performance of the static model along with the transient performance of the dynamic model. Therefore, a complete demonstration of the standalone static and dynamic simulations will not be given. A comparison between all three fidelity models is discussed in the following section. Evaluation of this design in terms of performance and constraint violations will not be discussed here, and instead is included in Section 5.

The simulation follows the mission profile given in Fig. 3.2 with the following design parameters: $N_{series} = 12$, $N_{parallel} = 26$, $N_{motors} = 8$, $h_{stator} = 10mm$, $N_{turns} = 60$, and $f_{sw} = 30kHz$. Simulation results that characterize UAV performance is given in Fig 4.1. Almost all of these measurements are used as objectives

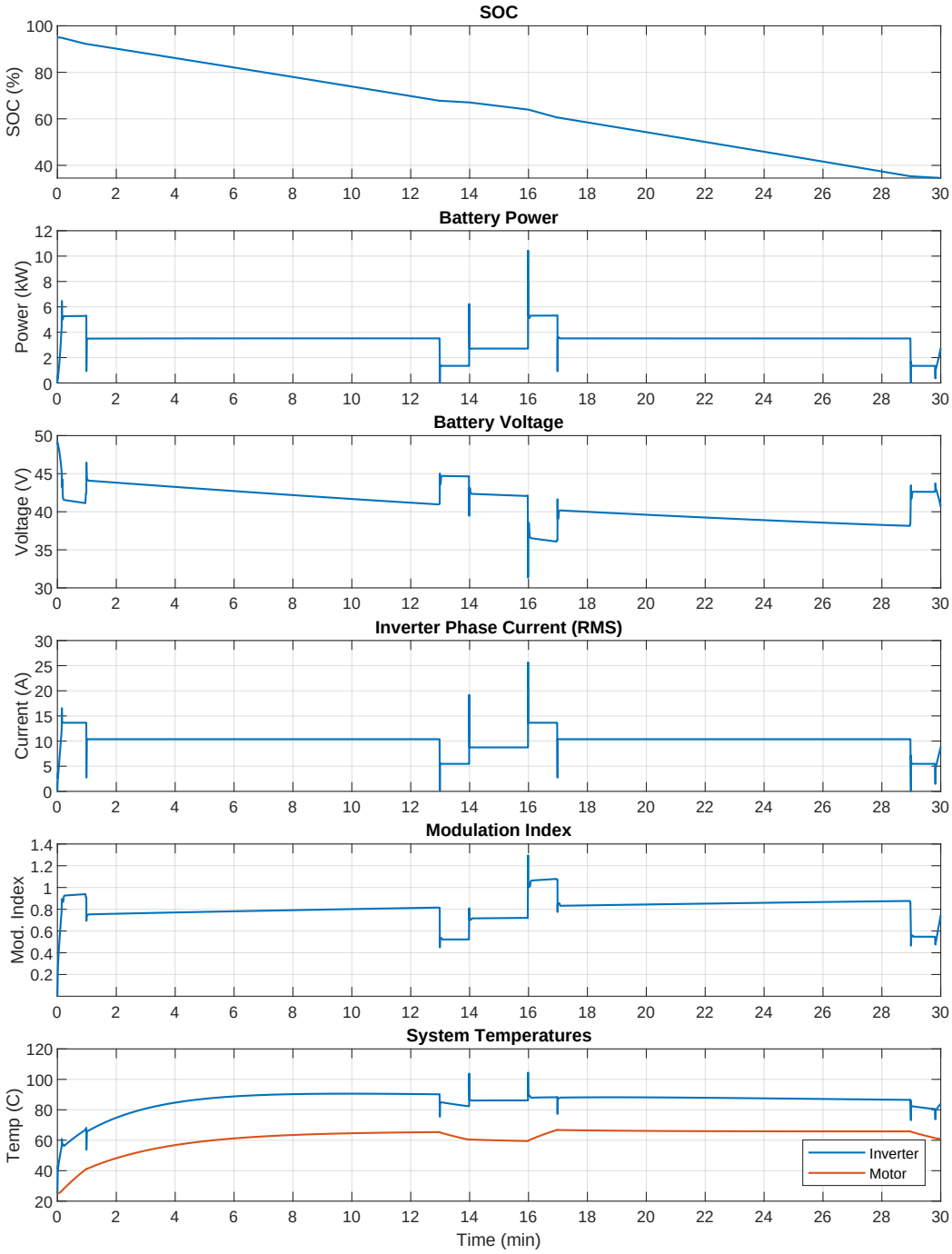


Figure 4.1: Quasi-dynamic simulation.

and constraints for system optimization in Chapter 5. The general increase in modulation index throughout the mission is correlated with the decrease in battery voltage, which itself is a function of SOC. Between each stage of flight (e.g., takeoff, cruising, hovering, etc.) exists an acceleration/deceleration period that cause a sudden increase (or decrease) in power and can be observed as impulse-like spikes in Fig 4.1. The high current needed for acceleration causes a drop in battery voltage due to the internal impedance. There is also a sudden change in inverter temperature during the acceleration periods. As the inverter's thermal model assumes zero thermal capacitance of each MOSFET junction, a sudden increase in power results in a sudden increase in temperature, as seen in (3.16). This behavior is not observed with motor temperature because the entire motor is modeled as a lumped thermal capacitor.

4.2 Model-Fidelity Variance

When considering what level of fidelity is necessary/sufficient for system design optimization, it helps to understand how the results vary between them. A comparison between the static, dynamic, and quasi-dynamic models is provided herein. This comparison uses a single UAV design ($N_{series} = 12$, $N_{parallel} = 26$, $N_{motors} = 8$, $h_{stator} = 10mm$, $N_{turns} = 60$, and $f_{sw} = 30kHz$); but note that similar differences are observed for any set of design parameters. Table 4.1 summarizes the simulation results that characterize UAV performance and are also used as objectives and constraints for optimization in Chapter 5. Two values for minimum V_{cell} and

maximum I_{cell} are provided in Table 4.1 for the quasi-dynamic model since it uses both static and dynamic models.

Table 4.1: Comparison of Static, Dynamic and Quasi-dynamic Simulation Results and Execution Time.

Measurement	**Static**	**Dynamic**	**Quasi-dynamic**
E_{total} (kWh)	1.9627	1.9546	1.9618
Total DOD (%)	60.72	60.47	60.47
min V_{cell} (V)	3.0047	2.6170	3.0071/2.6148
max I_{cell} (A)	5.6645	12.7830	5.6612/12.7915
max m_a	1.0799	1.2951	1.0790/1.2960
max T_i (°C)	91.70	104.52	105.46
max T_m (°C)	66.68	67.43	66.89
Approx. Execution Time (s):	0.2	1620	90

In summary, minimal differences are observed between all models for total energy E_{total}, total DOD, and maximum motor temperature T_m. The discrepancy between static and dynamic models is apparent with the battery cell voltage V_{cell}, battery cell current I_{cell}, modulation index m_a and inverter temperature T_i. This is primarily due to the spike in power when the UAV is accelerating. This transient is not captured in the static model, which instead transitions to the steady-state load power after acceleration. Fig. 4.2 provides an example of this difference between the static, dynamic and quasi-dynamic models. During acceleration, a large inrush of current into the motors cause a drastic voltage drop of the battery due to its internal impedance. To supply the acceleration current, the inverter's m_a must increase to compensate. This current spike occurs for less than a second and the models quickly stabilize to the same level.

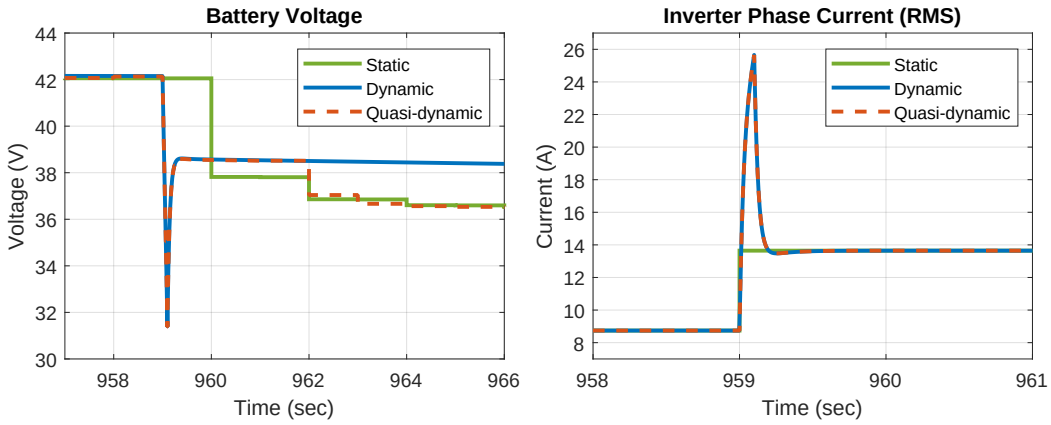


Figure 4.2: Simulation comparison between the static, dynamic, and quasi-dynamic fidelity models.

A larger variation in battery voltage after acceleration can be observed in Fig. 4.2. The capacitive elements within the dynamic battery model requires time before the voltage reaches its steady-state value, which the static model immediately calculates. Due to the slow voltage response time present in a real battery and captured by the dynamic battery model, the static model actually represents the worst case scenario battery voltage during periods of no acceleration. However, the static model also does not capture the voltage drop caused from acceleration. This is one of the benefits of a quasi-dynamic simulation, where both voltages can be measured. The lack of voltage drop with the static model isn't necessarily a disadvantage, either. While batteries have a minimum cut-off voltage, they typically can operate lower than the cut-off point for short periods of time. The short duration of the voltage drop in Fig. 4.2 thus should be considered separate to the steady-state minimum battery voltage captured by the static model.

The approximate execution time of each fidelity simulation is also provided in Table 4.1. Based on these results, the static simulation runs 8100 times faster than the dynamic simulation and 450 times faster than the quasi-dynamic simulation. The quasi-dynamic simulation is 18 times faster than the dynamic simulation. The minimal difference in UAV performance between the three models clearly highlights the advantage of using the static or quasi-dynamic simulation – especially for system level design and optimization where there are many designs to be explored. However, system dynamics are necessary when controls are considered, as apparent with m_a.

4.3 Pareto Front Comparison

The higher fidelity of the dynamic models allow for more design constraints and performance measurement accuracy than the static models, as discussed in Section 4.2. This will actually result in an alternate Pareto front. To demonstrate the differences, an exhaustive search of the design space of Table 4.2 was performed using the static model simulation. Since the dynamic model only increases the constraints on a particular design, it's only necessary to simulate designs that passed all constraints with the static model. Note that these results use a slightly different design space and power system models than the results in Section 5.4.

Fig. 4.3 shows the valid design and Pareto fronts for both the static and dynamic modeling simulations. Of the 5,443 designs that passed all static constraints, only 2,234 designs passed the dynamic constraints. Additionally, all designs considered

Table 4.2: UAV Design Space for Static and Dynamic Model Pareto Front Comparison.

Design Parameter	Range
Battery cells in series, N_{series} (#)	[8:18]
Battery cells in parallel, $N_{parallel}$ (#)	[10:42]
Quantity of motors, N_{motors} (#)	[6:14]
Height of stator structure, h_{stator} (mm)	[8:26]
Motor stator winding turns, N_{turns} (#)	[10:60]

Pareto optimal using the static model did not pass the constraints of the dynamic model. The two primary contributing factors are the increased modulation index and inverter temperature caused by the acceleration stages of flight.

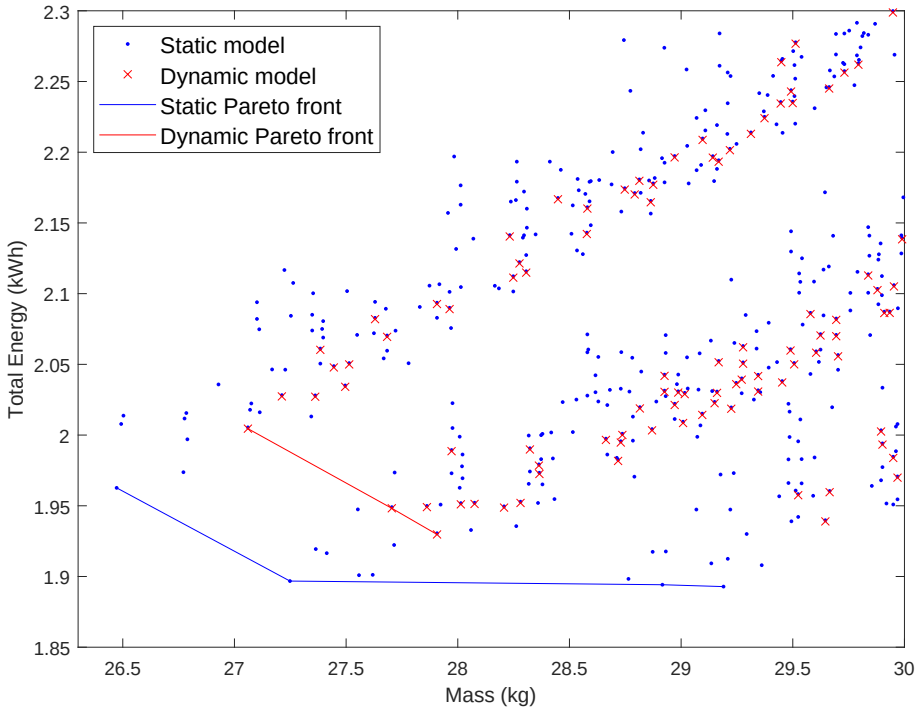


Figure 4.3: Pareto front comparison between the static and dynamic models.

Chapter 5: Power System Optimization

5.1 Model-ML Integration Issues

The physical modeling approach described in Chapter 3 requires a proper interface with the ML algorithm. A major challenge of model-ML integration is how to handle failed design cases, as the failed designs still provide useful learning information although the simulations face non-preferred conditions. In order to maintain consistent results, simulations must continue running regardless of whether the design succeeds the mission under constraints or not. Consistency in results is essential, since MESMOC (and BO in general) learns from every simulation. However, simulations are subject to instability when the vehicle is operating at extreme limits. For example, the motor winding temperature is especially susceptible to positive feedback and eventual simulation instability.

Simulation instability can be prevented by suppressing model critical variables, which are often constraint variables (temperature, voltage, current, etc.), when a specified limit is exceeded. A soft limit approach still allows differentiation between healthy and ill designs by preserving information about the extremity of constraint violations. A hyperbolic function, such as the *tanh* trigonometric function, realizes this soft limit by asymptotically approaching a value. When a variable's constraint is to be violated, a hyperbolic function suppresses the output through variable

saturation. For the example of motor winding temperature, applying a soft limit when the temperature exceeds its physical limit reduces further positive feedback by saturating the calculated temperature. An example of temperature suppression is shown in Fig. 5.1. This provides more information to the ML algorithm than if a hard ceiling limit is used, where various failed system designs would report the same motor temperature that would be indistinguishable by the ML program.

In the case of battery energy depletion, the SOC is reset to the simulation's initial SOC. This is preferred over terminating a simulation at the minimum allowed SOC, as the reported total energy consumed will likely be much lower than a design that completes the mission. A depth of discharge (DOD) variable keeps track of the total amount of battery discharged throughout multiple SOC resets. By resetting the SOC, the total energy consumed will more accurately represent the necessary energy to complete the mission for a specific design.

The modifications mentioned above are only important for integrating with ML optimization algorithms. Other algorithms like NSGA-II have no intelligent component involved in selecting a design for evaluation (where it's actually randomized). Additionally, *how much* a constraint is violated is irrelevent as NSGA-II takes a binary approach to handling constraints (a design does or does not violate a constraint).

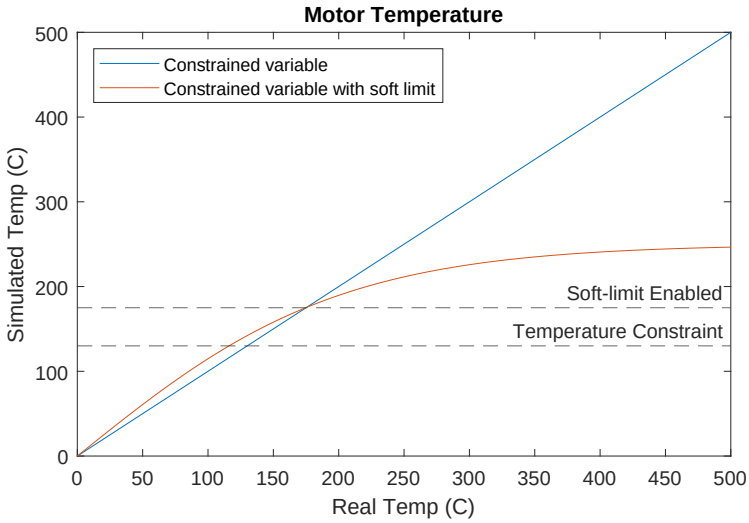


Figure 5.1: Example of the soft limit imposed on the motor temperature, a model critical constraint variable.

5.2 Problem Statement

With the power system models constructed and the design variables, objectives and constraints identified, the optimization process is ready to begin. In this chapter, the proposed ML MOO framework is demonstrated using the VTOL heavy-duty all-electric UAV architecture outlined in Chapter 3. The optimization problem for

this UAV design is formally defined as

$$\begin{aligned} \min_{\boldsymbol{x}} \quad & E_{total}, m_{UAV} \\ \text{s.t.} \quad & DOD \leq 0.75 \\ & V_{bat} \geq 3.0\,V \\ & T_m \leq 130°C \\ & T_i \leq 125°C \\ & m_a \leq 1.155 \end{aligned} \tag{5.1}$$

This case study chooses two optimization objectives: minimizing the energy consumption throughout the mission, E_{total}, and minimizing the UAV's mass, m_{UAV}. These two objectives are common practice in industrial designs; however, users may choose others such as reliability and cost. The two selected objectives contain mutual tradeoffs, since heavier components (e.g., motors) tend to be more efficient, however, the extra weight must consume additional energy. On the other hand, many constraints apply to the realistic design candidates; and five representative constraints serve for demonstration purpose in this thesis. In particular, Li-ion batteries have a typical operating range between 20% and 95% SOC, thus a maximum DOD of 75%. A minimum battery cell terminal voltage, V_{bat}, is imposed to prevent cell damages [36]. The maximum motor winding temperature, T_m, limits thermal degradation of wire lamination and is based on the NEMA insulation class B rating [39]. Semiconductor device failure is avoided with a maximum inverter temperature, T_i, that is set marginally lower than the datasheet informa-

tion [36]. A maximum modulation index, m_a, is set to avoid excessive unwanted distortion that can occur under a SPWM switching scheme [40]. At the end of a simulation, the constraint and objective variables are returned to MESMOC, as described in Fig. 2.2 in Section 2.4.

5.3 Design Space Selection and Analysis

Selecting the range of design parameters (together called the design space) to be considered during optimization can have a strong impact on the optimization process effectiveness. A large design space immediately requires more simulation executions to find the Pareto optimal designs. Additionally, when constraints are considered, selecting too large of a design space often results in more constraint-violating designs rather than Pareto optimal candidates. Some optimization algorithms do not handle constraint-heavy spaces well. Inversely, too small of a design space results in the risk of excluding a Pareto optimal design. Some design parameters are easier to select bounds for than others. For example, an upper bound for the battery or system voltage can be set based on device ratings such as the switches in a power converter. Prior experience of the design engineers may also be used to select the bounds; however, it may include risks of excluding good design candidates. Another method to determine the design space is to perform an exhaustive search with a large parameter step size (i.e, test 20V, 40V, 60V instead of 20V, 25V, 30V,..., 60V) or a lower-fidelity model with low computational time (such as static vs. dynamic) as discussed in [8].

The design space used for UAV power system optimization in the following section is provided in Table 5.1 and includes battery sizing (N_{series}, $N_{parallel}$), motor count (N_{motors}), motor sizing (h_{stator}, N_{turns}), and inverter switching frequency (f_{sw}). These bounds were determined from prior simulations and maximum voltage ratings of the MOSFET/Diode of the inverter. This method of boundary selection is sufficient for demonstration purpose in this thesis, while real-life selections also consider the specific application, the physical model's input/output details, and the designer's confidence interval. The computational overhead of using the brute force method to determine the design space range is ignored in this thesis because the same space is used for each algorithm tested.

Table 5.1: UAV Design Space for Trials 1 and 2 in Section 5.4.

Design Parameter	Range
Battery cells in series, N_{series} (#)	[10:18]
Battery cells in parallel, $N_{parallel}$ (#)	[16:70]
Quantity of motors, N_{motors} (#)	[6:10]
Height of stator structure, h_{stator} (mm)	[8:26]
Motor stator winding turns, N_{turns} (#)	[10:55]
Inverter switching frequency, f_{sw} (kHz)	[10:40]

The given 30-minute mission profile of Fig. 3.2 and the design space defined in Table 5.1 results in a highly constrained design space. To be precise, only 20% of designs passed all constraints listed in (5.1) (with f_{sw} omitted from the design space). A visualization of the constraint-heavy design space is given in Fig. 5.2 where the percent of valid designs given a motor size or battery size are shown. In the best case for a single motor size, only 45% of designs passed all constraints

due to the other selected parameters. Similarly, the best case battery size had 55% valid designs.

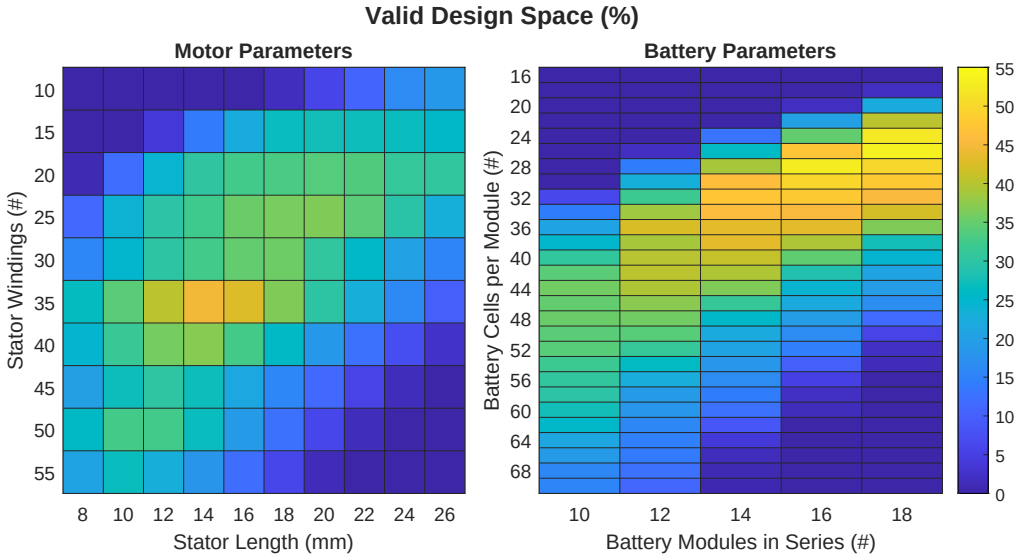


Figure 5.2: Percent of valid designs for each design parameter combination. Motor size parameters and battery size parameters are considered independently.

Fig. 5.2 also shows that 13 (out of 100) motor sizes and 33 (out of 140) battery sizes never resulted in a valid design. This demonstrates how the valid design space is non-convex. For most (if not all) optimization algorithms the input design space must be a hyperrectangle and convex, which means the bounds for a single design parameter cannot vary based on other design parameters. For example, to omit from the design space the battery size of $N_{series} = 18$ and $N_{parallel} = 68$, *all* designs with $N_{series} = 18$ *or* $N_{parallel} = 68$ must be removed from the design space. This would result in the removal of many potentially optimal designs. It is possible to define a (non-)linear constraint that limits the battery size to certain

combinations, equivalent to a limit of total stored energy. However, in black-box optimization such as with this UAV power system, prior knowledge of required energy is not available.

5.4 Optimization With MESMOC, PESMOC, and NSGA-II

This section contains results from a paper [41] to appear in the IEEE Transaction of Transportation Electrification journal and was written, in part, by Syrine Belakaria of Washington State University.

MESMOC is compared to the known constrained GA, namely NSGA-II, and to the BO method PESMOC. Each algorithm's performance is evaluated using the Pareto hypervolume (PHV) metric, which is a commonly employed metric to measure the quality of a given Pareto front [42]. Given the randomness in the NSGA-II process, the algorithm is ran several times and only the Pareto front of the best performing run and the hypervolume curves of three different runs are reported. MESMOC and PESMOC utilize an open-source BO library called *Spearmint*, and the *PyGMO* library is used for the NSGA-II algorithm in the cheap MO solver. The *Platypus* library was used for the NSGA-II trials and configured for the same total number of design evaluations as MESMOC and PESMOC. The *matlab.engine* library enables MATLAB function and script calls in the Python environment.

Each optimization algorithm is applied to the design process of a UAV power system in two trials to evaluate performance. The first trial approaches the power

system design with five parameters, while the second trial adds an additional parameter to explore the scalability of the ML algorithm *under the same computing assumption*. The design space for both trials, comprised of each parameter and the range, is summarized in Table 5.1.

5.4.1 Trial 1: 5-Dimensional Design Space

The design space for this trial consists of 42,000 design combinations using five parameters: N_{series}, $N_{parallel}$, N_{motors}, N_{turns}, and h_{stator}, indicated in Table 5.1. Based on the constrained optimization problem given in (5.1), brute force, MESMOC, PESMOC and NSGA-II methods were run. Fig. 5.3 shows all the points evaluated by brute force, MESMOC, PESMOC and NSGA-II, and the Pareto fronts of all algorithms. The figure indicates each point as a potential design with respect to the two objective functions (energy minimization and mass minimization), omitting points that do not pass all constraints. Note that the sporadic empty spaces and discrete boundaries are due to a few factors: 1) the battery parameters, N_{series} and $N_{parallel}$, cause discrete changes to the objectives; 2) the inherent difference in the energy requirement and vehicle mass between n-motored UAVs creates clusters of points within the objective space; and 3) many designs in the design space are not valid due to the defined constraints and are thus not shown.

MESMOC uncovered all six points in the optimal Pareto front. This was achieved with 1,736 design evaluations, equating to 4% of the entire design space

searched. In comparison, PESMOC and NSGA-II did not find any of the optimal Pareto optimal points under the same number of design evaluations. Additionally, given that the constraints are defined as black-box, it is important to evaluate the ability of each algorithm to select inputs that satisfy the constraints. PESMOC and NSGA-II experiments show poor performance with percentages of valid points selection of 4% and 39%, respectively. For MESMOC, 95% of the selected inputs are valid.

While Fig. 5.3 provides a visual aid to compare the Pareto fronts to the entire design space, it does not show the convergence behavior and progress of the algorithms. Hence the PHV metric, measuring the hypervolume of the best Pareto front throughout the search, is shown in Fig. 5.4. The graph depicts how each algorithm's hypervolume approaches the volume of the optimal PHV (constant line shown in brown), which is calculated from the brute force search results. Note that an iteration for NSGA-II is equivalent to a design evaluation. A single generation for NSGA-II will consist of many iterations (based on population size). From Fig. 5.4, not only does MESMOC successfully discover the optimal Pareto front, it also converges faster than PESMOC and NSGA-II do to their best front. This experiment highlights MESMOC's ability to reduce design evaluations while also maintaining search accuracy.

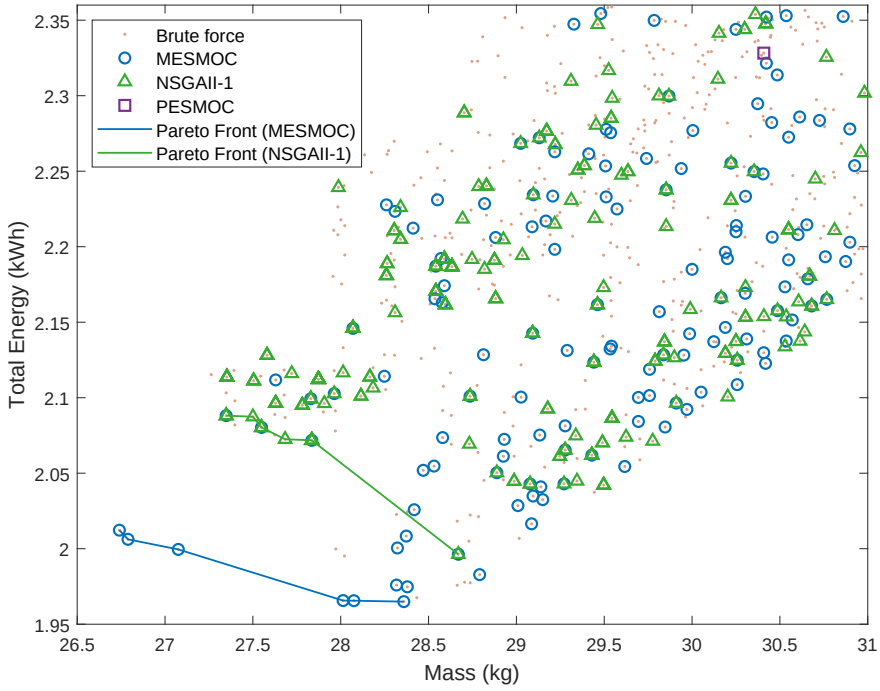


Figure 5.3: Trial 1: Pareto fronts and design space evaluated by brute force, MESMOC, PESMOC and NSGA-II.

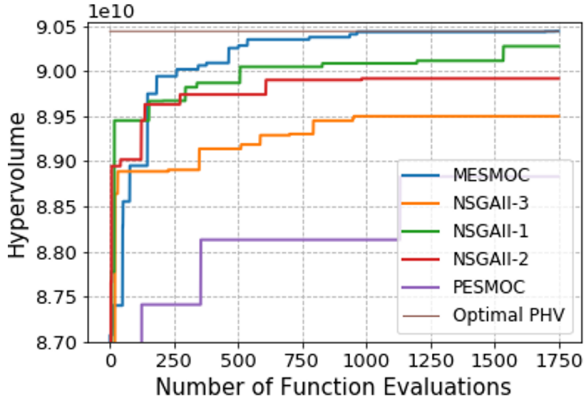


Figure 5.4: Trial 1: PHV through the design search for MESMOC, PESMOC and three runs of NSGA-II.

5.4.2 Trial 2: 6-Dimensional Design Space

The terminating point for MOO algorithms such as GA or BO is a continuing challenge. When the optimal Pareto front is unknown, there is no deterministic method to generate a termination point. As such, it is common to interpret the given Pareto front after a set number of iterations as the most optimal. Algorithms that provide an accurate and fast convergence rate are then highly favorable, as the quality of optimization is dependent on the Pareto front at the end of a search. By quadrupling the design space size while retaining the same number of design evaluations, we can compare the non-optimal Pareto front quality of the proposed MESMOC to that of PESMOC and NSGA-II. To achieve this increase in design space size, the inverter switching frequency, f_{sw}, becomes a design variable. All other parameters and their ranges remain the same as in Trial 1. Thus, the total number of design combinations increases to 168,000.

With the same constrained optimization problem as Trial 1, MESMOC uncovered two of the five points on the Pareto front while PESMOC and NSGA-II did not discover any Pareto optimal points. Fig. 5.5 shows the Pareto fronts of all algorithms, along with the optimal Pareto front. It can be noted that while MESMOC does not find the entire optimal Pareto front, it offers a strong final Pareto front. Moreover, MESMOC maintains its high performance at selecting 80% of valid designs while the performance of PESMOC and NSGA-II degrades even further to selecting only 1% and 15% of valid points, respectively.

A PHV plot throughout the search with MESMOC, PESMOC and NSGA-II is

provided in Fig. 5.6, showing that the Pareto front found by MESMOC has a PHV quantitatively close to the optimal PHV. Compared to the PHV in Fig. 5.4 from Trial 1, MESMOC's PHV curve in Trial 2 grows much slower and contains more extended periods of zero improvement. During the perceived periods of no improvement, MESMOC is still selecting designs to simulate that will maximize the information gain with respect to the optimal Pareto front. The intelligent component of MESMOC to continually improve the statistical model of the design space with respect to the objective functions guarantees continual Pareto front improvement. However, the increased design space size requires more design evaluations to build a robust statistical model of the objective functions. Before MESMOC was terminated at 1,750 designs, the PHV was still increasing. On the other hand, NSGA-II depends on random 'mutations' of its current set of designs for PHV improvement. Because of the GA's inherent randomness, there is no guarantee it will converge on the optimal Pareto front. The enhanced performance of the output-space entropy search used in MESMOC is made clear when compared to PESMOC, another BO algorithm.

While MESMOC can drastically reduce the number of design evaluations compared to brute force, PESMOC and NSGA-II, it comes at a cost. The MESMOC acquisition function relies on the previously evaluated design parameters and objective values. Thus, the computational cost of selecting the next design to simulate increases with respect to the iteration number. To put it into context, MESMOC consumes an average (throughout the optimization process) of 60 seconds per iteration (excluding physical model's simulation time) for Trial 1 and an average

of 68 seconds per iteration for Trial 2. In terms of scalability, the time per iteration should increase linearly with the increase of number of constraints and objectives, experimentally shown in [6] with MESMO. This computational cost increase is part of the motivation behind the parallel implementation of MESMOC. Regardless, the performance of MESMOC is highly favorable, as there is a trade-off between computation time and optimization accuracy for systems with a large design space. In the end, the drastic reduction in design evaluations outweighs the computation gain in parameter selection per iteration. The benefit is especially obvious when simulating complex physical models (inside the black-box) where each physical simulation iteration takes much time.

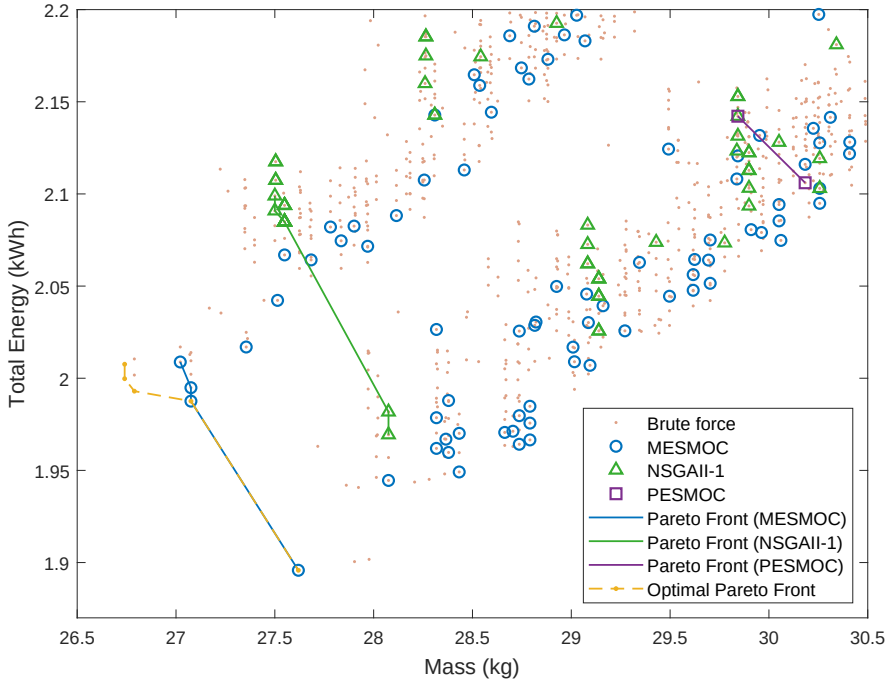


Figure 5.5: Trial 2: Pareto fronts and design space evaluated by brute force, MESMOC, and NSGA-II.

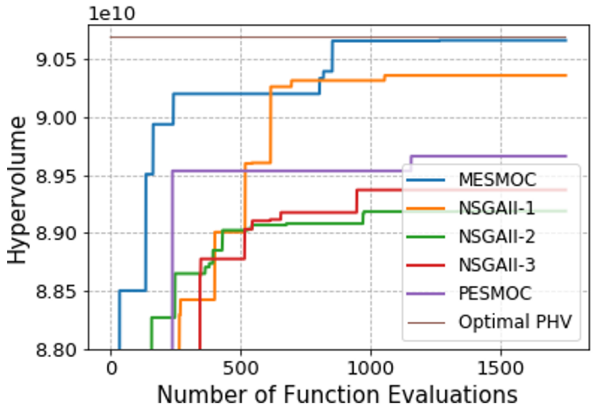


Figure 5.6: Trial 2: PHV through the design search for both MESMOC, PESMOC and three runs of NSGA-II.

5.5 Randomness of Genetic Algorithms

This section further demonstrates the randomness and variability of the GA, specifically NSGA-II. In Section 5.4 NSGA-II used a fixed population size for each run. As discussed in Section 2.5, there are multiple options that can be configured even for a specific GA (like NSGA-II). The population size, crossover rate, and mutation rate are varied and 5 trials are run for each combination, totalling 560 runs. The same design space is used as Trial 2 of Section 5.4. For this demonstration a maximum of 1,800 design simulations are run, to closely match the results of Trial 2.

All NSGA-II runs were implemented using the MATLAB multi-objective GA library within the *Global Optimization Toolbox*. Custom creation and mutation functions were written because the default MATLAB functions do not allow multi-

objective optimization of a discrete design space. The custom creation function implemented latin hypercube sampling for the initial population. The bitwise mutation operator of the original NSGA-II implementation was adapted to work with an integer vector-based genome.

The Pareto fronts of all 560 NSGA-II runs and subset plots are given in Fig. 5.7. The top right plot used population sizes of $\{25, 50, 100, 200\}$. The bottom left plot used crossover rates of $\{0.2, 0.4, 0.6, 0.8\}$. The bottom right plot used mutation rates of $\{0.2, 0.4, 0.6, 0.8\}$. Each option combination ran 5 times to show the randomness of the stochastic process.

While the true Pareto front was almost or completely discovered during some NSGA-II runs, the majority of runs did not uncover the true front. Specific GA options and the stochastic nature of the algorithm all contributed to the variability in results. There isn't the option to test a range of options and see which give the best results when evaluating a design incurs a large time expense. Therefore, an algorithm that is consistent in it's performance regardless of the variable options can greatly reduce the guess work of the optimization process.

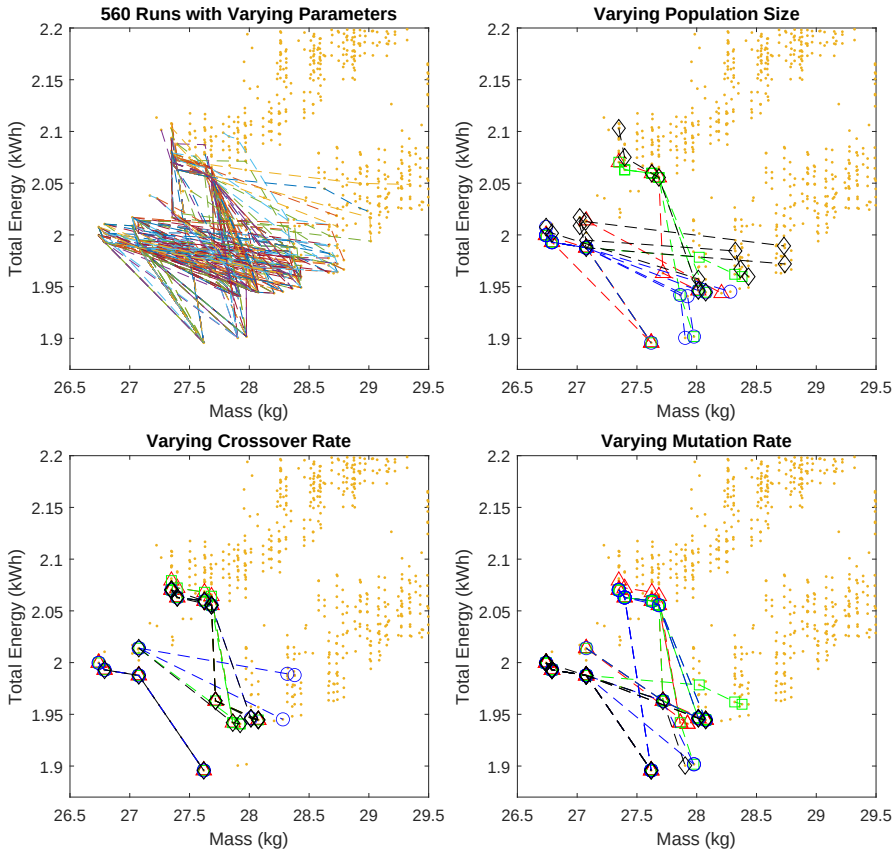


Figure 5.7: Top left: the Pareto fronts of all 560 NSGA-II runs. Top right: a subset of 20 runs of varying population size and all other options fixed. Bottom left: a subset of 20 runs of varying crossover rate and all other options fixed. Bottom right: a subset of 20 runs of varying mutation rate and all other options fixed.

Chapter 6: Conclusion

While the modern day methods of model-based design and physics-based simulations make the engineering task more manageable, the developmental process remains a time-consuming venture. To keep up with the growing demand for power electronic integrated systems found within the energy and transportation sectors, a machine learning approach to accelerate the design and optimization process is presented. The approach aims to automate the multi-objective design exploration process by utilizing an efficient and high-performance optimization algorithm and multi-physics power system modeling. Design of a VTOL heavy-duty all-electric UAV serves as a demonstration.

A review of the design and optimization process emphasizes the challenges with multi-objective power electronic system design. The concept of Pareto optimality and a non-convex, constraint-heavy design space being two contributing factors. A large component to design and optimization involves developing the multi-physics power system models that are used to evaluate a designs performance. The modeling theory and approach accompanied by simulation demonstrations lays out the required electrical domain knowledge. This includes model-algorithm integration issues that must be addressed for effective optimization. Design parameter and constraint selection requires the electrical domain expertise as well.

The nature of electrical system optimization greatly limits algorithm options.

Metaheuristics are often employed when exhaustive or deterministic methods are not an option due to limitations in computational resources. However, these metaheuristic methods involve stochastic processes that reduce their reliability and can result in slow convergence rates. Bayesian optimization (BO), a machine learning framework, takes an intelligent approach to design exploration and removes the stochastic component. By utilizing statistical models which map the design space to the objective space, the design space is efficiently searched to discover the Pareto optimal points. A novel BO algorithm, called MESMOC, is shown to outperform two optimization algorithm competitors in both Pareto front quality and convergence rates.

In this thesis, the machine learning optimization approach utilizes static modeling techniques for the UAV power system simulation. Future work will investigate the use of higher-fidelity models for design and optimization. Additional design constraints and more accurate performance metrics become available with increased fidelity, such as the dynamic and quasi-dynamic models discussed, but increases simulation times. A bi-level optimization approach, where multiple fidelity models are used, could integrate well within the machine learning framework. How the model fidelity and ML influence each other is an interesting topic.

Bibliography

[1] R. Aarenstrup, *Managing Model-Based Design*. Natick, MA: The MathWorks, Inc., 2015.

[2] Y. Cao, M. A. Williams, B. J. Kearbey, A. T. Smith, P. T. Krein, and A. G. Alleyne, "20x-real time modeling and simulation of more electric aircraft thermally integrated electrical power systems," in *Proc. IEEE International Conf. Electrical Systems for Aircraft, Railway, Ship Propulsion and Road Vehicles (ESARS-ITEC)*, pp. 1–6, 2016.

[3] A. Papageorgiou, M. Tarkian, K. Amadori, and J. Ölvander, "Multidisciplinary design optimization of aerial vehicles: A review of recent advancements," *International Journal of Aerospace Engineering*, vol. 2018, pp. 1–21, 2018.

[4] A. Ghosh and S. Dehuri, "Evolutionary algorithms for multi-criterion optimization: A survey," *International Journal of Computing & Information Sciences*, vol. 2, pp. 38–57, 2004.

[5] S. Zhao, F. Blaabjerg, and H. Wang, "An overview of artificial intelligence applications for power electronics," *IEEE Transactions on Power Electronics*, vol. 36, pp. 4633–4658, 2021.

[6] S. Belakaria, A. Deshwal, and J. R. Doppa, "Max-value entropy search for multi-objective bayesian optimization," in *Proc. Advances in Neural Information Processing Systems (NeurIPS) Conf.*, pp. 7823–7833, 2019.

[7] X. Hu., J. Han, X. Tang, and X. Lin, "Powertrain design and control in electrified vehicles: A critical review," *IEEE Transactions on Transportation Electrification*, pp. 1–19, 2021.

[8] E. Silvas, T. Hofman, N. Murgovski, L. F. P. Etman, and M. Steinbuch, "Review of optimization strategies for system-level design in hybrid electric vehicles," *IEEE Transactions on Vehicular Technology*, vol. 66, no. 1, pp. 57–70, 2017.

[9] Z. Jiang, H. Huang, X. Jia, and J. Zhang, "Model-based design and real-time testing of commercial more-electric aircraft power systems," in *Proc. IEEE National Aerospace and Electronics Conf.*, pp. 28–35, 2018.

[10] N. Rashidi, Q. Wang, R. Burgos, C. J. Roy, and D. Boroyevich, "Multi-objective design and optimization of power electronics converters with uncertainty quantification – part i: Parametric uncertainty," *IEEE Transactions on Power Electronics*, vol. 36, no. 2, pp. 1463–1474, 2020.

[11] N. Rashidi, Q. Wang, R. Burgos, C. J. Roy, and D. Boroyevich, "Multi-objective design and optimization of power electronics converters with uncertainty quantification – part ii: Model-form uncertainty," *IEEE Transactions on Power Electronics*, vol. 36, no. 2, pp. 1441–1450, 2020.

[12] E. Vinot, R. Trigui, B. Jeanneret, J. Scordia, and F. Badin, "Hevs comparison and components sizing using dynamic programming," in *Proc. IEEE Vehicle Power and Propulsion Conf.*, pp. 314–321, 2007.

[13] L. Zhang, M. Lv, W. Zhu, H. Du, J. Meng, and J. Li, "Mission-based multi-disciplinary optimization of solar-powered hybrid airship," *Journal of Energy Conversion and Management*, vol. 185, pp. 44–54, 2019.

[14] N. B. Hadj, J. K. Kammoun, and R. Neji, "Application of evolutionary algorithm for triobjective optimization: Electric vehicle," *Int. Journal of Energy Optimization and Engineering (IJEOE), IGI Global*, vol. 3, no. 3, pp. 1–19, 2014.

[15] M. D'Antonio, C. Shi, B. Wu, and A. Khaligh, "Design and optimization of a solar power conversion system for space applications," *IEEE Transactions on Industry Applications*, vol. 55, no. 3, pp. 2310–2319, 2019.

[16] C. Desai and S. S. Williamson, "Optimal design of a parallel hybrid electric vehicle using multi-objective genetic algorithms," in *Proc. IEEE Vehicle Power and Propulsion Conf.*, pp. 871–876, 2009.

[17] X. Wu, B. Cao, J. Wen, and Z. Wang, "Application of particle swarm optimization for component sizes in parallel hybrid electric vehicles," in *Proc. IEEE Congress on Evolutionary Computation (IEEE World Congress on Computational Intelligence)*, pp. 2874–2878, 2008.

[18] M. P. Mark Hauschild, "An introduction and survey of estimation of distribution algorithms," *Journal of Swarm and Evolutionary Computation*, vol. 1, no. 3, pp. 111–128, 2011.

[19] B. Shahriari, K. Swersky, Z. Wang, R. P. Adams, and N. de Freitas, "Taking the human out of the loop: A review of bayesian optimization," *Proceedings of the IEEE*, vol. 104, no. 1, pp. 148–175, 2016.

[20] Z. Zhou, S. Belakaria, A. Deshwal, W. Honga, J. R. Doppa, P. Pande, and D. Heo, "Design of multi-output switched-capacitor voltage regulator via machine learning," in *Proc. IEEE/ACM Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pp. 502–507, 2020.

[21] J. Knowles, "Parego: a hybrid algorithm with on-line landscape approximation for expensive multiobjective optimization problems," *IEEE Transactions on Evolutionary Computation*, vol. 10, no. 1, pp. 50–66, 2006.

[22] W. Ponweiser, T. Wagner, D. Biermann, and M. Vincze, "Multiobjective optimization on a limited budget of evaluations using model-assisted s-metric selection," in *Proc. Parallel Problem Solving from Nature*, pp. 784–794, Springer Berlin Heidelberg, 2008.

[23] E. Brochu, V. M. Cora, and N. de Freitas, "A tutorial on bayesian optimization of expensive cost functions, with application to active user modeling and hierarchical reinforcement learning," Tech. Rep. UBC TR-2009-23, Dept. of Computer Science, University of British Columbia, 2009.

[24] C. Williams and C. Rasmussen, *Gaussian processes for machine learning*. Cambridge, MA, USA: MIT Press, 2006.

[25] J. R. Gardner, M. J. Kusner, Z. Xu, K. Q. Weinberger, and J. P. Cunningham, "Bayesian optimization with inequality constraints," in *Proc. of the 31st International Conference on International Conf. on Machine Learning*, pp. 937–945, 2014.

[26] Z. Wang and S. Jegelka, "Max-value entropy search for efficient bayesian optimization," in *Proc. International Conf. on Machine Learning (ICML)*, pp. 1–12, 2017.

[27] D. Jackson, S. Belakaria, Y. Cao, J. Doppa, and X. Lu, "Machine learning enabled fast multi-objective optimization for electrified aviation power system design," in *Proc. IEEE Energy Conversion Congress & Expo (ECCE)*, pp. 6385–6390, 2020.

[28] J. M. Hernández-Lobato, M. A. Gelbert, and et al, "Predictive entropy search for bayesian optimization with unknown constraints," in *Proc. International Conf. on Machine Learning*, vol. 37, pp. 1699–1707, 2015.

[29] E. C. Garrido-Merchán and D. Hernández-Lobato, "Predictive entropy search for multi-objective bayesian optimization with constraints," *Neurocomputing*, vol. 361, pp. 50–68, 2019.

[30] A. Rahimi and B. Recht, "Random features for large-scale kernel machines," in *Proc. Advances in Neural Information Processing Systems (NeurIPS) Conf.*, pp. 1177–1184, 2008.

[31] S. Katoch, S. S. Chauhan, and V. Kumar, "A review on genetic algorithm: past, present, and future," *Multimedia Tools and Applications*, vol. 80, no. 5, pp. 8091–8126, 2021.

[32] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan, "A fast and elitist multiobjective genetic algorithm: Nsga-ii," *IEEE Transactions on Evolutionary Computation*, vol. 6, no. 2, pp. 182–197, 2002.

[33] Y. Cao, R. C. Kroeze, and P. T. Krein, "Multi-timescale parametric electrical battery model for use in dynamic electric vehicle simulations," *IEEE Trans. Transportation Electrification*, vol. 2, no. 4, pp. 432–442, 2016.

[34] A. Thurlbeck and Y. Cao, "Analysis and modeling of uav power system architectures," in *Proc. IEEE Transportation Electrification Conf. (ITEC)*, pp. 1–8, 2019.

[35] Y. Cao and A. Thurlbeck, "Heavy-duty uav electric propulsion architectures and multi-timescale multi-physics modeling," in *Proc. AIAA/IEEE Electric Aircraft Technologies Symposium (EATS)*, pp. 1–13, 2019.

[36] J. Hayes and G. Goodarzi, *Electric Powertrain: energy systems, power electronics and drives for hybrid, electric and fuel cell vehicles.* Hoboken, NJ: John Wiley & Sons, 2018.

[37] D. Graovac and M. Purschel, "Mosfet power losses calculation using the datasheet parameters," 2006. Infineon Application Note.

[38] T. A. Lipo, *Introduction to AC Machine Design.* Hoboken, NJ: John Wiley & Sons, 2017.

[39] Engineering ToolBox, "Nema insulation classes," 2004. www.engineeringtoolbox.com/nema-insulation-classes-d_734.html.

[40] P. T. Krein, *Elements of Power Electronics, 2nd ed.* New York: Oxford University Press, 2015.

[41] D. Jackson, S. Belakaria, Y. Cao, J. Doppa, and X. Lu, "Machine learning enabled design automation and multi-objective optimization for electric transportation power systems." Submitted to *IEEE Transactions on Transportation Electrification.*

[42] E. Zitzler, *Evolutionary algorithms for multiobjective optimization: Methods and applications.* PhD thesis, ETH Zurich, Switzerland, 1999.

APPENDICES

Appendix A: MATLAB Code for Static Modeling

A.1 Simulation Main

```
function [Total_Energy,Final_DOD,Mass,Vmin_cell,Imax_cell,...
    Max_Temp_Mot,Max_Temp_ESC,Max_Mod_Index,simdata]...
    = run_sim_static(bs, bp, mn, msl, msw, fsw, mssn)
% This function runs the UAV flight mission simulation given the design
% parameters.

%% Setup
%get params
[bat, esc, motor, sys] = sys_params(bs, bp, mn, msl, msw, fsw);

%init loss collection
bat_wire = 0;
bat_cell = 0;
esc_wire = 0;
esc_fet = 0;
mot_wire = 0;
mot_mech = 0;

I_cell = 0;

%initialize results
Total_Energy = 0;   %total energy consumed, in J
Final_DOD = 0;      %final DOD of battery
Mass = sys.mass;
Max_Temp_Mot = 0;
Max_Temp_ESC = 0;
Max_Mod_Index = 0;
Vmin_cell = 4.2;    %init max cell voltage
Imax_cell = 0;      %init max cell current

%Calculate RPM and Torque
time = mssn.time;

%removes non-integer values for the static sim
toRemove = fix(time)~=time;
```

```
time(toRemove) = [];
timeStep = mssn.timeStep;
Thrust = mssn.Thrust*sys.mass; %total thrust, kgf
Thrust(toRemove) = [];

Thrust_m = (Thrust/sys.motor_num)*9.805;            %thrust per motor (N)
Torque = motor.prop_a*(Thrust/sys.motor_num);       %torque per motor (N-m)
RPM = motor.prop_b*(Thrust/sys.motor_num).^(motor.prop_c); %per motor (RPM)



%% Run Simulation
SOC_init = 0.95;
SOC = SOC_init;   %initial State of Charge
temp_mot(1) = 25; %initial temp of motor, starting at ambient temp
temp_esc(1) = 25; %initial temp of ESC, starting at ambient temp
temp_s(1) = 25;   %initial temp of heat sinks in the ESC

%compute initial state of battery
[V_bat(1), eff_bat(1), Ploss_bat(1), Ploss_bat2esc(1),...
    Pout_bat(1), SOC(1)] = Battery(SOC,.001,timeStep,bat,sys.motor_num);
V_bus(1) = V_bat(1);

for i = time/timeStep
    %reset SOC if below 20%
    if SOC(i+1) < .2
        Final_DOD = Final_DOD + SOC_init - 0.2; %update DOD
        SOC(i+1) = SOC_init;
    end

    [Iphase(i+1), ma(i+1), Pin_mot(i+1), Ploss_mot(i+1), eff_mot(i+1),...
     temp_mot(i+2), Pout_mot(i+1), mot_loss]...
        = Motor(RPM(i+1), Torque(i+1),Thrust_m(i+1), V_bus(i+1),...
          temp_mot(i+1),timeStep, motor);

    V_bus(i+2) = V_bat(i+1) - Iphase(i+1)*bat.R_w*2; %wire voltage drop

    [eff_esc(i+1), Iin_esc(i+1), Ploss_esc(i+1), Ploss_esc2mot(i+1),...
     temp_esc(i+2), temp_s(i+2), esc_loss] = ESC(V_bus(i+1),Iphase(i+1),...
        ma(i+1),Pin_mot(i+1),temp_esc(i+1),temp_s(i+1),timeStep,esc);

    [V_bat(i+2), eff_bat(i+1), Ploss_bat(i+1),Ploss_bat2esc(i+1),...
     Pout_bat(i+1), SOC(i+2), bat_data] = Battery(SOC(i+1),Iin_esc(i+1),...
        timeStep,bat,sys.motor_num);
    %battery() will find the next iterations Vbat and SOC using the
```

```
    %current iteration's SOC and I_out.


    %% Update constraint vars
    Vmin_cell = min(Vmin_cell, bat_data.Vcell);
    Imax_cell = max(Imax_cell,bat_data.I_cell);
    Icell(i+1) = bat_data.I_cell;

end

%% Gather up results
Psystem = (Ploss_bat + Pout_bat); %total power of system
Ploss_system = Ploss_bat + (Ploss_esc + Ploss_mot +...
    Ploss_bat2esc + Ploss_esc2mot)*sys.motor_num; %total power loss

Total_Energy = sum(Psystem*timeStep);        %total energy consumed, in J
Final_DOD = Final_DOD + SOC_init - SOC(end); %final SOC of battery
Max_Temp_Mot = max(temp_mot);                %max temp of motor
Max_Temp_ESC = max(temp_esc);                %max temp of ESC
Max_Mod_Index = max(ma);                     %max modulation index

simdata.Loss_Energy = sum(Ploss_system*timeStep); %Total energy lost, in J
simdata.Efficiency = (Total_Energy - simdata.Loss_Energy)/Total_Energy;

end
```

A.2 Motor

```
function [Iphase, ma, P_in, Ploss, Eff, temp_new, P_out, mot_loss]...
    = Motor(RPM, Torque, Thrust, Vin, temp, time_step, motor)

w_m = RPM * pi/30;                 %angular mechanical speed (rad/s)
w_e = w_m * motor.polePair;        %electrical frequency (rad/s)
EMF = motor.Ke*w_m*motor.polePair; %back emf
if EMF == 0
    EMF = 0.001;                   %avoid div 0 when w_m = 0
end

%% Intermediate Power calculations
P_out = w_m*Torque;                %power output of motor
Ploss_mech = motor.Mech2*(w_m^2) + motor.Mech1*w_m; %mechanical power loss
P_motor = P_out + Ploss_mech;      %intermediate total power into motor

%% Electrical calculations
```

```
17
18 %temperature adjusted resistance, 0.00393 is constant for copper in ohm/C
19 Rs = motor.Rs*(1 + 0.00393*(temp-25));
20
21 Iphase = P_motor/(3*EMF);                          %single phase current
22 Vterm = ((EMF^4 + 2*(EMF^2)*(P_motor/3)*Rs + ((P_motor/3)^2)*Rs^2 ...
23     + ((P_motor/3)^2)*(w_e*motor.Ls)^2)^(1/2))/EMF; %terminal voltage
24 ma = (Vterm*sqrt(2)*sqrt(3))/Vin;                  %modulation index
25
26 %make sure mod index does not go to zero
27 if (ma < 0.001)
28     ma = 0.001;
29 end
30
31 %% Final Power and Efficiency calculations
32 Ploss_elec = Rs*(Iphase^2)*3;   %electrical loss for all 3 phases
33 P_in = P_motor + Ploss_elec;    %total power in
34 Eff = P_out/P_in;               %efficiency of motor
35 Ploss = Ploss_elec + Ploss_mech;
36
37 %% Temperature calculations
38 rho = motor.rho;     %air density, kg/m^3
39 D_m = motor.D_m;     %diameter of motor, m
40 L = motor.Length;    %height of motor, m
41 u_air = motor.u_air; %absolute air viscosity, N*s/m^2
42 k = motor.k;         %thermal conductivity of air, W/mk
43
44 A_m = motor.A_m;                          %area of heat transfer of motor
45 A_a = motor.A_a;                          %area of air moving around motor
46 v_a = sqrt(Thrust/(rho*A_a))*motor.fracoef;%vel of air relative to motor.
47
48 Re = rho*v_a*D_m/u_air;         %Reynolds number for air
49 Pr = motor.Pr;                  %Prandtl number for air
50 Nu = 0.3 + ((0.62*sqrt(Re)*Pr^.333)/((1 + (0.4/Pr)^.666)^.25))*...
51     (1 + (Re/282000)^.625)^.8;  %Nusselt number
52 h_air = Nu*k/L;                 %heat transfer coefficient
53 T_amb = 25;                     %ambient temperature
54
55 if (time_step == 0)
56     %steady-state temp calc
57     temp_new = T_amb + Ploss/(A_m*h_air); %temp of motor for next iteration
58 else
59     %dynamic temp calc
60     t_delta = (Ploss + A_m*h_air*(T_amb - temp))*...
61         time_step/(motor.mass*motor.c_p);
```

```
    temp_new = temp + t_delta;
end

%start suppressing temp values when getting too hot
%max temp will be 250C, well above constraint.
%a scaled tanh curve is used.
var_a = 201.5;  %variable to tune tanh curve
if temp_new > 175
    %temp of motor for next iteration
    temp_new = 250*(exp(temp_new/var_a)-exp(-temp_new/var_a))/...
        (exp(temp_new/var_a)+exp(-temp_new/var_a));
end

end
```

A.3 Power Electronics

```
function [Efficiency, Iin, Ploss, Ploss_w, Tj_new ,Ts_new, esc_data]...
    = ESC(Vin,Iphase,ma,Pmotor,Tj,Ts,time_step,esc)
%calculates efficiency and required input current of an individual ESC

if (Tj > 140)
    Tj = 140; %max out at 140C
end
Ron_ds = interp1(esc.Ta_lu, esc.Rdson_lu,Tj); %MOSFET on resistance
Von_d = esc.Von_d;                            %diode forward voltage drop

Paux = 6;             %measured auxillary power losses at no loads
I_o = sqrt(2)*Iphase; %peak phase current
I_dc = I_o/pi;        %DC equivalent current


%% Switching times
tfu1 = (Vin - Ron_ds*I_dc)*esc.Ron_g*esc.Cgd1/(esc.V_g-esc.V_miller);
tfu2 = (Vin - Ron_ds*I_dc)*esc.Ron_g*esc.Cgd2/(esc.V_g-esc.V_miller);
tfu = (tfu1 + tfu2)/2;
tru1 = (Vin - Ron_ds*I_dc)*esc.Roff_g*esc.Cgd1/(esc.V_miller);
tru2 = (Vin - Ron_ds*I_dc)*esc.Roff_g*esc.Cgd2/(esc.V_miller);
tru = (tru1 + tru2)/2;

%% Mosfet Losses
Pcond_m = Ron_ds*(I_o^2)*(1/8 + (ma*esc.DisFac/(3*pi))); %conduction loss
%Using Qrr of diode because it will dominate over mosfets internal diode
Eon_m = Vin*I_dc*((esc.tri+tfu)/2) + esc.Qrr*Vin; %turn on energy cost
```

```
Eoff_m = Vin*I_dc*((tru+esc.tfi)/2);              %turn off energy cost
Pswitch_m = (Eon_m + Eoff_m) * esc.f_switch;      %average switching power

%% Diode Losses
Pcond_d = Von_d*I_o*(1/(2*pi) - (ma*esc.DisFac)/8); %diode conduction loss
if Pcond_d < 0
    Pcond_d = 0;
end
Eon_d = (esc.Qrr*Vin)/4;    %turn on energy of diode
Pon_d = Eon_d*esc.f_switch; %average diode turn on power

%% ESC Power and Efficiency calculations
%power loss from mosfet/diode
Ploss_m = (Pcond_m + Pswitch_m + Pcond_d + Pon_d)*6;
%power loss from 3 phase wires from esc to motors
Ploss_w = (Iphase^2)*esc.R_w * 3;
Ploss = Ploss_m + Paux;

P_in = Pmotor + Ploss + Ploss_w; %total power going into ESC
Efficiency = Pmotor/P_in;

%% Input current
Iin = P_in/Vin;

%% Temperature calculations

T_amb = 25; %ambient temperature

if (time_step == 0)
    %steady-state temp calc
    Ts_delta = (Tj-Ts)/esc.Rth_jc + (T_amb-Ts)/esc.Rth_sa;
    Ts_new = Ts + Ts_delta;
    Tj_new = Ts_new + Ploss*esc.Rth_jc; %new temp
else
    %dynamic temp calc
    Ts_delta = ((Tj-Ts)/esc.Rth_jc + (T_amb-Ts)/esc.Rth_sa)*...
        time_step/(esc.hs_mass*esc.cp_al);
    Ts_new = Ts + Ts_delta;
    Tj_new = Ts_new + Ploss*esc.Rth_jc*time_step;

end

end
```

A.4 Battery

```
function [V_bat, Eff, P_loss, P_lossW, P_out, SOC_new, bat_data]...
    = Battery(SOC,Iout,timeStep,bat,mnum)

%% Initialize Results
bat_data.Vlow = 4.2;

%% Battery Equivalent Circuit Parameters

%calculate V and R using the cell parameters and curve fitting
V_int = exp(bat.Voc(1) + bat.Voc(2)*log(SOC)...
    + bat.Voc(3)*(log(SOC))^2 ...
    + bat.Voc(4)*(log(SOC))^3 ...
    + bat.Voc(5)*(log(SOC))^4 ...
    + bat.Voc(6)*(log(SOC))^5 ...
    + bat.Voc(7)*(log(SOC))^6);
R_seriesD = exp(bat.R_serD(1) + bat.R_serD(2)*log(SOC)...
    + bat.R_serD(3)*(log(SOC))^2 ...
    + bat.R_serD(4)*(log(SOC))^3 ...
    + bat.R_serD(5)*(log(SOC))^4 ...
    + bat.R_serD(6)*(log(SOC))^5 ...
    + bat.R_serD(7)*(log(SOC))^6);
R_secD = exp(bat.R_secD(1) + bat.R_secD(2)*log(SOC)...
    + bat.R_secD(3)*(log(SOC))^2 ...
    + bat.R_secD(4)*(log(SOC))^3 ...
    + bat.R_secD(5)*(log(SOC))^4 ...
    + bat.R_secD(6)*(log(SOC))^5 ...
    + bat.R_secD(7)*(log(SOC))^6);
R_minD = exp(bat.R_minD(1) + bat.R_minD(2)*log(SOC)...
    + bat.R_minD(3)*(log(SOC))^2 ...
    + bat.R_minD(4)*(log(SOC))^3 ...
    + bat.R_minD(5)*(log(SOC))^4 ...
    + bat.R_minD(6)*(log(SOC))^5 ...
    + bat.R_minD(7)*(log(SOC))^6);
R_hourD = exp(bat.R_hourD(1) + bat.R_hourD(2)*log(SOC)...
    + bat.R_hourD(3)*(log(SOC))^2 ...
    + bat.R_hourD(4)*(log(SOC))^3 ...
    + bat.R_hourD(5)*(log(SOC))^4 ...
    + bat.R_hourD(6)*(log(SOC))^5 ...
    + bat.R_hourD(7)*(log(SOC))^6);
%add up all R
R_int = R_seriesD + R_secD + R_minD + R_hourD;


```

```
%% I,V,P Calculations

%Find single cell P,I,V values
Ibat = Iout*mnum;               %scale to number of ESC/motors
I_cell = Ibat / bat.parallel;   %I of each cell series
V_cell = V_int - I_cell*R_int;  %V of each cell
P_lossC = I_cell^2 * R_int;     %P loss of each cell
bat_data.Vcell = V_cell;        %store cell voltage

%calculate individual time scale voltage drops for dyn sim
bat_data.Vsec = R_secD*I_cell;
bat_data.Vmin = R_minD*I_cell;
bat_data.Vhour = R_hourD*I_cell;


if V_cell < bat.Vmin
    V_cell = bat.Vmin;
end

%Calculate battery pack values
V_bat = V_cell*bat.series;  %battery voltage seen at the end of wire

P_lossW = Iout^2 *bat.R_w*mnum*2; %Ploss from battery terminals to each esc
P_loss = P_lossC*bat.series*bat.parallel; %total power loss
P_out = V_bat*Ibat;
P_total = P_out + P_loss; %total power delivered by battery

%store cell current
bat_data.I_cell = I_cell;

Eff = P_out/P_total; %calculate efficiency

%% Update SOC
E_cur = bat.E_max*SOC - P_total*timeStep; %get updated Energy stored
SOC_new = E_cur/bat.E_max; %calculate new SOC

end
```

Appendix B: Simulink Diagrams and MATLAB Code for Dynamic Model Subsystems

B.1 Simulation Main

```
function [Total_Energy, Final_DOD, Mass, Vmin_static, Vmin_dyn,...
    Imax_static, Imax_dyn, Max_Temp_Mot, Max_Temp_ESC,...
    Max_Mod_Index_static, Max_Mod_Index_dyn, simdata]...
    = run_sim_dynamic(bs, bp, mn, msl, msw, fsw, mssn)

%% Setup
%get params
[bat, esc, mot, sys] = sys_params(bs, bp, mn, msl, msw, fsw);

%initialize results
SOC_init = 0.95;
Total_Energy = 0; %Total energy consumed, in J
Final_DOD = 0; %final DOD of battery
Mass = sys.mass;
Max_Temp_Mot = 0;
Max_Temp_ESC = 0;
Max_Mod_Index_static = 0;
Max_Mod_Index_dyn = 0;
Vmin_dyn = 4;
Vmin_static = 4;
Imax_dyn = 0;
Imax_static = 0;

%Calculate RPM and Torque
time = mssn.time;
timeStep = mssn.timeStep;
Thrust = mssn.Thrust*sys.mass;              %total thrust, kgf
Thrust_m = (Thrust/sys.motor_num)*9.805;    %thrust per motor, in N
Torque = mot.prop_a*(Thrust/sys.motor_num); %torque per motor (N-m)
RPM = mot.prop_b*(Thrust/sys.motor_num).^(mot.prop_c); %RPM per motor (RPM)
%create timeseries of profile
Torque_ts = timeseries(Torque(1,:).', time(1,:));
```

```
RPM_ts = timeseries(RPM, time);
Thrust_ts = timeseries(Thrust_m, time);

%% Init sim variables
Iphase          = timeseries();
ma              = timeseries();
P_mot_in        = timeseries();
P_mot_loss      = timeseries();
eff_mot         = timeseries();
temp_mot        = timeseries(25,0);
P_mot_out       = timeseries();
eff_esc         = timeseries();
Iin_esc         = timeseries();
P_esc_loss      = timeseries();
temp_esc        = timeseries(25,0);
temp_esc_s      = timeseries(25,0);
V_bat           = timeseries();
V_bus           = timeseries();
eff_bat         = timeseries();
P_bat_loss      = timeseries();
P_bat_out       = timeseries();
SOC             = timeseries(SOC_init,0);
I_cell          = timeseries();
RPM_out         = timeseries();
P_bat2esc_loss  = timeseries();
P_esc2mot_loss  = timeseries();

%% Run Simulation

%compute initial state of battery
[s_V_bat, s_eff_bat, s_Ploss_bat, s_Ploss_bat2esc, s_Pout_bat, s_SOC]...
    = Battery(SOC.Data(end),.001,timeStep,bat,sys.motor_num);

V_bat   = addsample(V_bat,   'Data',s_V_bat,'Time',0,'OverwriteFlag',true);
V_bus   = addsample(V_bus,   'Data',s_V_bat,'Time',0,'OverwriteFlag',true);
SOC     = addsample(SOC,     'Data',s_SOC,  'Time',0,'OverwriteFlag',true);
Iin_esc = addsample(Iin_esc, 'Data',0,      'Time',0,'OverwriteFlag',true);
Iphase  = addsample(Iphase,  'Data',0,      'Time',0,'OverwriteFlag',true);
temp_mot= addsample(temp_mot,'Data',25,     'Time',0,'OverwriteFlag',true);

s_temp_s = 25;
s_V_bus = V_bus.Data(end);

%initialize these vars for dyn sim
bat_data.Vsec = 0; %voltage drop sec timescale
```

```
bat_data.Vmin = 0; %voltage drop min timescale
bat_data.Vhour = 0;%voltage drop hour timescale

i = 0;
while i < time(end)
    %stop simulation if SOC is below 20%
    if SOC.Data(end) < .2
        Final_DOD = Final_DOD + SOC_init - 0.2; %update DOD
        SOC.Data(end) = SOC_init;
        break
    end

    %check whether dynamic sim needs to happen
if any(time(i+1) == mssn.dyn_start)
    %% dynamic sim
    dyn_start = mssn.dyn_start(mssn.dyn_start == time(i+1));
    dyn_rec = dyn_start;
    dyn_dur = mssn.dyn_dur(mssn.dyn_start == dyn_start);
    dyn_end = dyn_start + dyn_dur;
    dyn_init = mssn.dyn_init(mssn.dyn_start == dyn_start);
    %set back dyn sim start time to allow for system to become stable
    if (dyn_start - dyn_init) > 0
        dyn_start = dyn_start - dyn_init;
    end

    if dyn_start == 0
        Pesc_loss = 0;
    else
        Pesc_loss = P_esc_loss.Data(end);
    end

    RPM_dyn_init = RPM_ts.Data(i+1)*pi/30; %initialize motor temp

    dyn_result = sim('Models_Dynamic\UAV_model.slx',...
        'SrcWorkspace','current', 'StartTime',...
        num2str(dyn_start), 'StopTime', num2str(dyn_end));

    %share updated values with static sim
    Iphase     =merge_ts(Iphase,    dyn_result.Iphase,    dyn_rec,dyn_end);
    P_mot_loss =merge_ts(P_mot_loss,dyn_result.P_mot_loss,dyn_rec,dyn_end);
    P_mot_out  =merge_ts(P_mot_out, dyn_result.P_mot_out, dyn_rec,dyn_end);
    P_mot_in   =merge_ts(P_mot_in,  dyn_result.P_mot_in,  dyn_rec,dyn_end);
    Iin_esc    =merge_ts(Iin_esc,   dyn_result.Iin_esc,   dyn_rec,dyn_end);
    P_esc_loss =merge_ts(P_esc_loss,dyn_result.P_esc_loss,dyn_rec,dyn_end);
    V_bat      =merge_ts(V_bat,     dyn_result.Vbat,      dyn_rec,dyn_end);
```

```
123     V_bus      =merge_ts(V_bus,     dyn_result.Vbus,      dyn_rec,dyn_end);
124     P_bat_loss =merge_ts(P_bat_loss,dyn_result.P_bat_loss,dyn_rec,dyn_end);
125     P_bat_out  =merge_ts(P_bat_out, dyn_result.P_bat_out, dyn_rec,dyn_end);
126     SOC        =merge_ts(SOC,       dyn_result.SOC,       dyn_rec,dyn_end);
127     I_cell     =merge_ts(I_cell,    dyn_result.I_cell,    dyn_rec,dyn_end);
128     RPM_out    =merge_ts(RPM_out,   dyn_result.RPM_out,   dyn_rec,dyn_end);
129     ma         =merge_ts(ma,        dyn_result.ma,        dyn_rec,dyn_end);
130     temp_mot   =merge_ts(temp_mot,  dyn_result.Temp_mot,  dyn_rec,dyn_end);
131     temp_esc   =merge_ts(temp_esc,  dyn_result.T_esc,     dyn_rec,dyn_end);
132     temp_esc_s =merge_ts(temp_esc_s,dyn_result.T_esc_s,   dyn_rec,dyn_end);
133     P_bat2esc_loss = merge_ts(P_bat2esc_loss,...
134         dyn_result.P_bat2esc_loss, dyn_rec, dyn_end);
135     P_esc2mot_loss = merge_ts(P_esc2mot_loss,...
136         dyn_result.P_esc2mot_loss, dyn_rec, dyn_end);
137     i = i + dyn_dur +2; %inc counter, +2 to account for sub-second thrusts
138
139     %% Update constraint vars
140     Vmin_dyn_temp = min(dyn_result.Vcell.Data);
141     Vmin_dyn = min(Vmin_dyn, Vmin_dyn_temp);
142
143     Imax_dyn_temp = max(dyn_result.I_cell.Data);
144     Imax_dyn = max(Imax_dyn, Imax_dyn_temp);
145
146     Max_Mod_Index_dyn_temp = max(dyn_result.ma.Data);
147     Max_Mod_Index_dyn = max(Max_Mod_Index_dyn, Max_Mod_Index_dyn_temp);
148
149 else
150     %% static sim
151     [s_Iphase, s_ma, s_Pin_mot, s_Ploss_mot, s_eff_mot, s_temp_mot,...
152         s_Pout_mot, mot_loss] = Motor(RPM(i+1), Torque(i+1),...
153         Thrust_m(i+1), V_bus.Data(end), temp_mot.Data(end), 1, mot);
154
155     s_V_bus = V_bat.Data(end) - s_Iphase*bat.R_w*2; %wire voltage drop
156
157     [s_eff_esc, s_Iin_esc, s_Ploss_esc, s_Ploss_esc2mot, s_temp_esc,...
158         s_temp_s, esc_loss] = ESC(s_V_bus,s_Iphase,s_ma,s_Pin_mot,...
159         temp_esc.Data(end),temp_esc_s.Data(end),1,esc);
160
161     [s_V_bat, s_eff_bat, s_Ploss_bat,s_Ploss_bat2esc, s_Pout_bat, s_SOC,...
162      bat_data]=Battery(SOC.Data(end),s_Iin_esc,timeStep,bat,sys.motor_num);
163     %battery() will find the next iterations Vbat and SOC using current
164     %iteration's SOC and I out.
165
166     %save new vars
167     Iphase      = addsample(Iphase,    'Data',s_Iphase,...
```

```
        'Time',time(i+1),'OverwriteFlag',true);
    ma          = addsample(ma,         'Data',s_ma,...
        'Time',time(i+1),'OverwriteFlag',true);
    P_mot_in    = addsample(P_mot_in,   'Data',s_Pin_mot,...
        'Time',time(i+1),'OverwriteFlag',true);
    P_mot_loss  = addsample(P_mot_loss,'Data',s_Ploss_mot,...
        'Time',time(i+1),'OverwriteFlag',true);
    temp_mot    = addsample(temp_mot,   'Data',s_temp_mot,...
        'Time',time(i+1),'OverwriteFlag',true);
    P_mot_out   = addsample(P_mot_out,  'Data',s_Pout_mot,...
        'Time',time(i+1),'OverwriteFlag',true);
    Iin_esc     = addsample(Iin_esc,    'Data',s_Iin_esc,...
        'Time',time(i+1),'OverwriteFlag',true);
    P_esc_loss  = addsample(P_esc_loss,'Data',s_Ploss_esc,...
        'Time',time(i+1),'OverwriteFlag',true);
    temp_esc    = addsample(temp_esc,   'Data',s_temp_esc,...
        'Time',time(i+1),'OverwriteFlag',true);
    temp_esc_s  = addsample(temp_esc_s,  'Data',s_temp_s,...
        'Time',time(i+1),'OverwriteFlag',true);
    V_bat       = addsample(V_bat,      'Data',s_V_bat,...
        'Time',time(i+1),'OverwriteFlag',true);
    V_bus       = addsample(V_bus,      'Data',s_V_bus,...
        'Time',time(i+1),'OverwriteFlag',true);
    P_bat_loss  = addsample(P_bat_loss,'Data',s_Ploss_bat,...
        'Time',time(i+1),'OverwriteFlag',true);
    P_bat_out   = addsample(P_bat_out,  'Data',s_Pout_bat,...
        'Time',time(i+1),'OverwriteFlag',true);
    SOC         = addsample(SOC,        'Data',s_SOC,...
        'Time',time(i+1),'OverwriteFlag',true);
    P_bat2esc_loss = addsample(P_bat2esc_loss,'Data',s_Ploss_bat2esc,...
        'Time',time(i+1),'OverwriteFlag',true);
    P_bat2esc_loss = addsample(P_bat2esc_loss,'Data',s_Ploss_bat2esc,...
        'Time',time(i+1),'OverwriteFlag',true);

    %additional sim info
    I_cell      = addsample(I_cell,   'Data',bat_data.I_cell,...
        'Time',time(i+1),'OverwriteFlag',true);

    i = i + 1; %increment counter

    %% Update constraint vars
    Vmin_static = min(Vmin_static, bat_data.Vcell);
    Imax_static = max(Imax_static, bat_data.I_cell);
    Max_Mod_Index_static = max(Max_Mod_Index_static, s_ma);
end
```

```
213
214 end
215
216 %% Gather up results
217 Psystem = P_bat_loss + P_bat_out; %Total power of system
218 Ploss_system = P_bat_loss + P_esc_loss*sys.motor_num +...
219     P_mot_loss*sys.motor_num; %Total power loss
220
221 Total_Energy = trapz(Psystem.Time, Psystem.Data);    %total energy, in J
222 Final_DOD = Final_DOD + SOC.Data(1) - SOC.Data(end); %final DOD of battery
223 Max_Temp_Mot = max(temp_mot.Data);                   %max temp of motor
224 Max_Temp_ESC = max(temp_esc.Data);                   %max temp of ESC
225
226 end
```

B.2 Motor

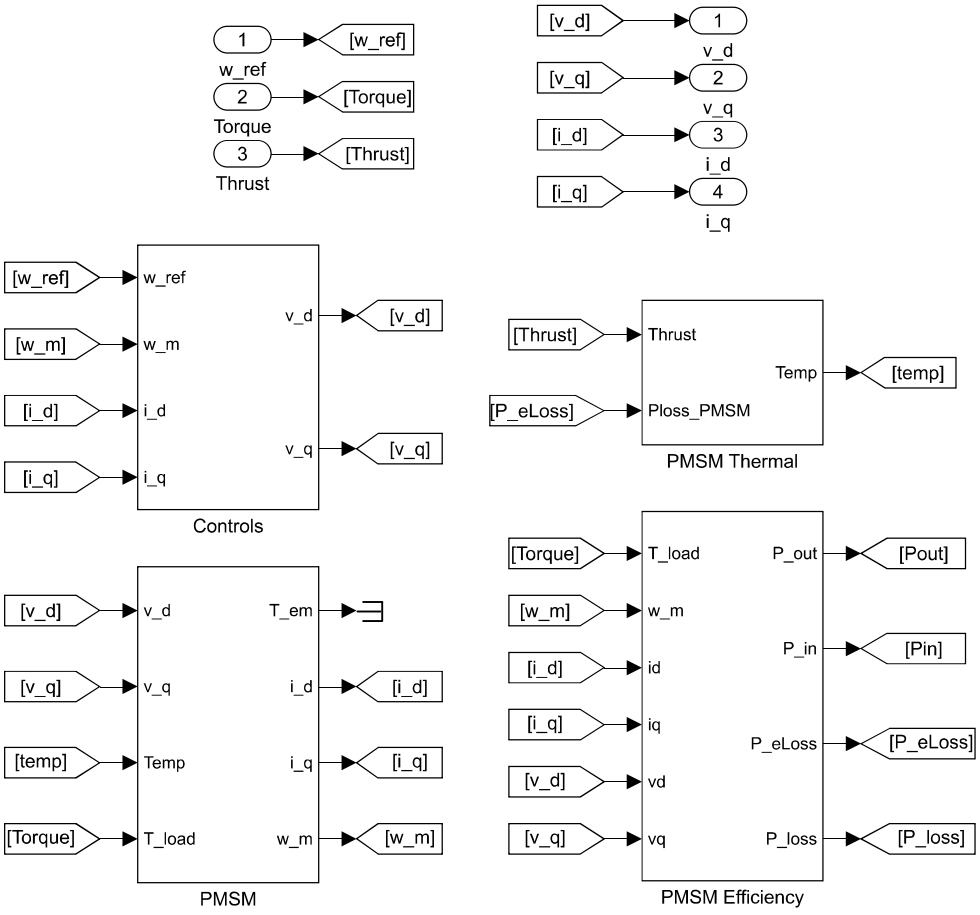


Figure B.1: PMSM Top Level.

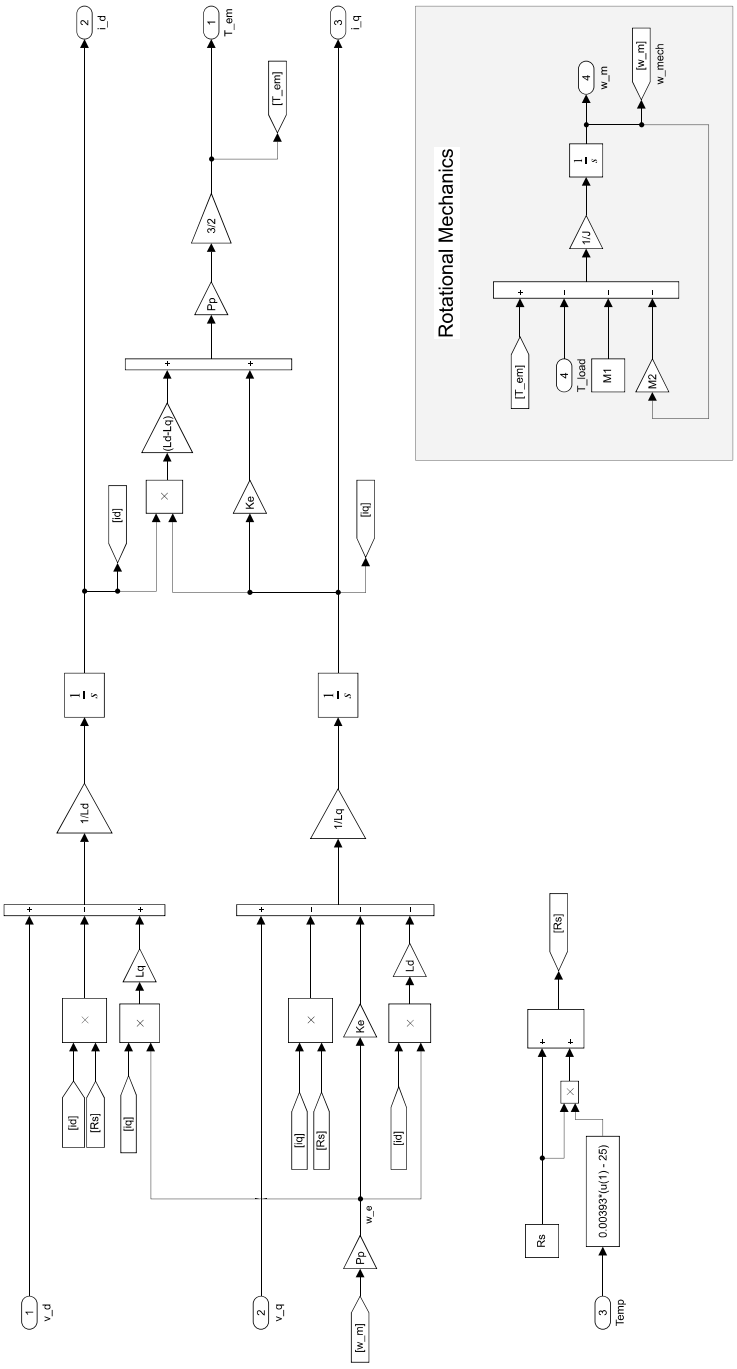


Figure B.2: PMSM Electrical and Rotational Mechanics.

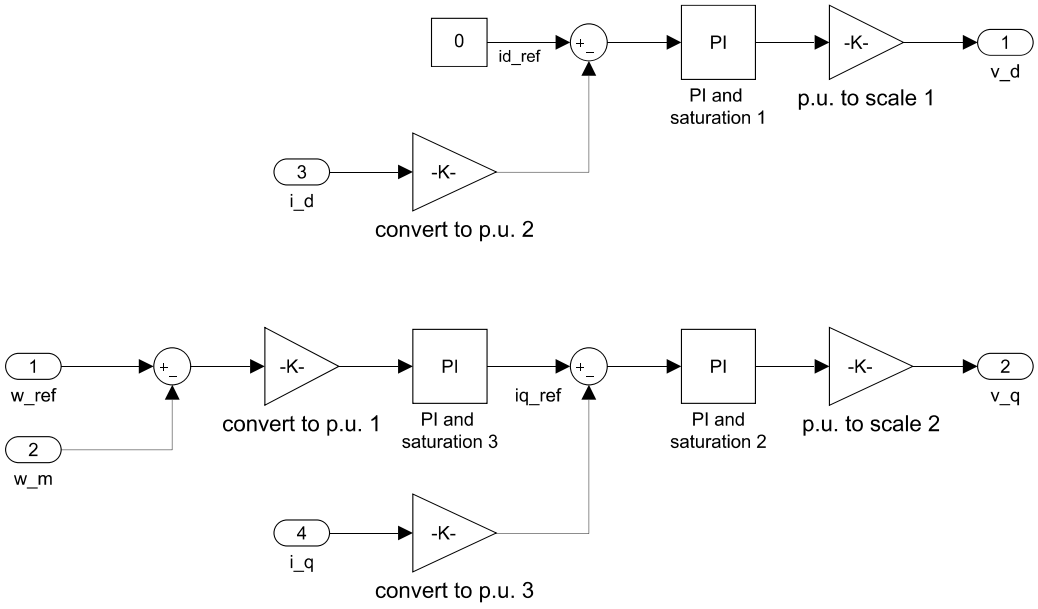


Figure B.3: PMSM Controls.

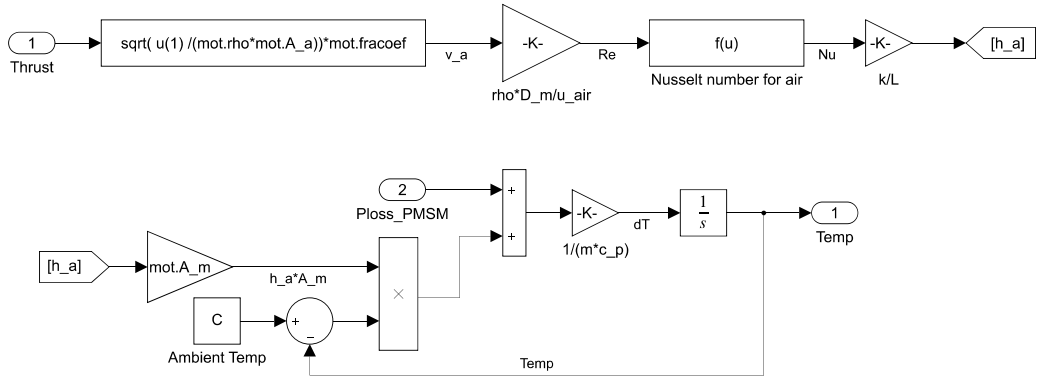


Figure B.4: PMSM Thermal.

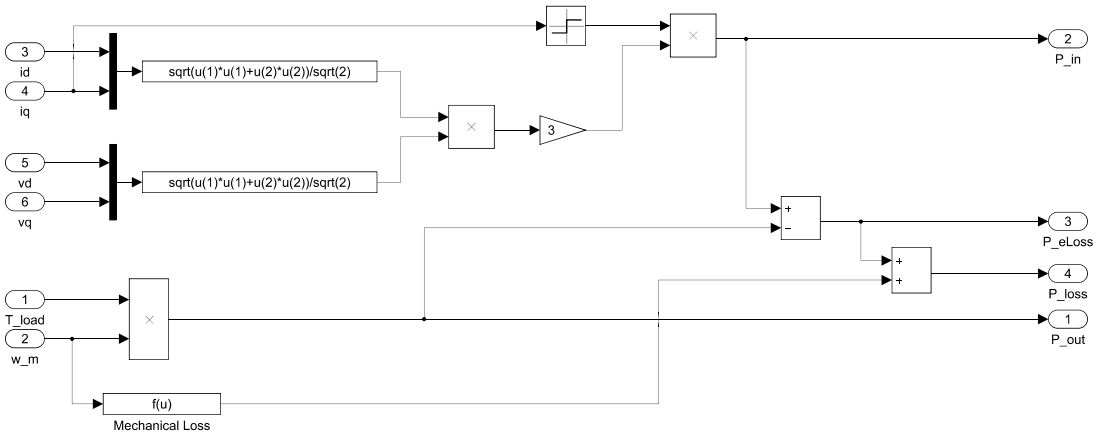


Figure B.5: PMSM Efficiency.

B.3 Power Electronics

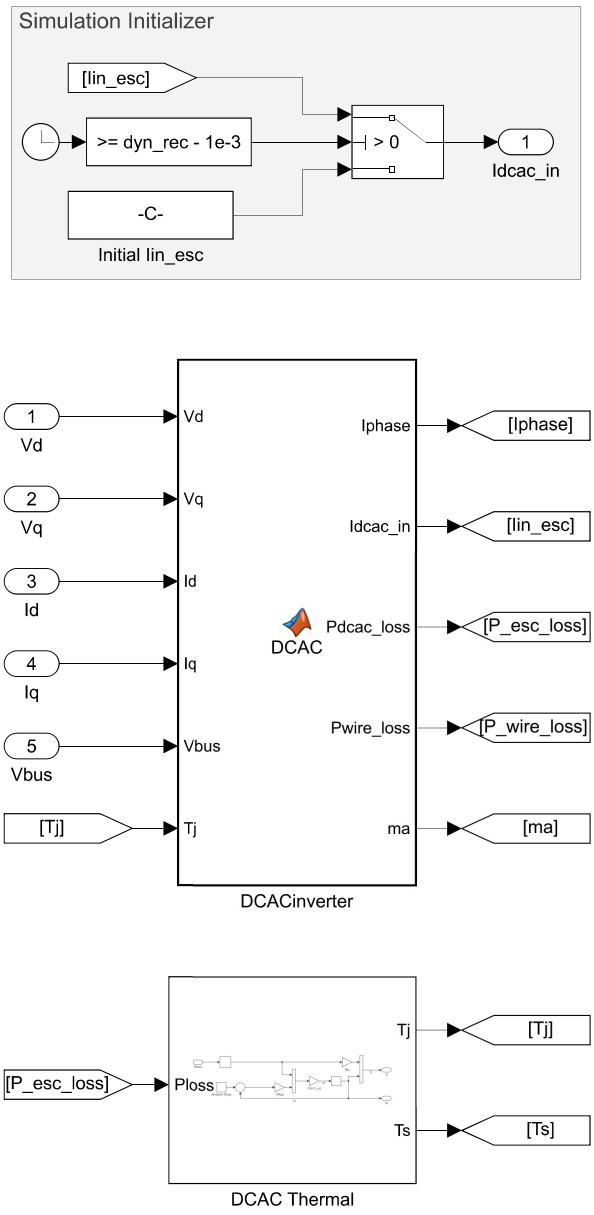


Figure B.6: DCAC Inverter Top Level.

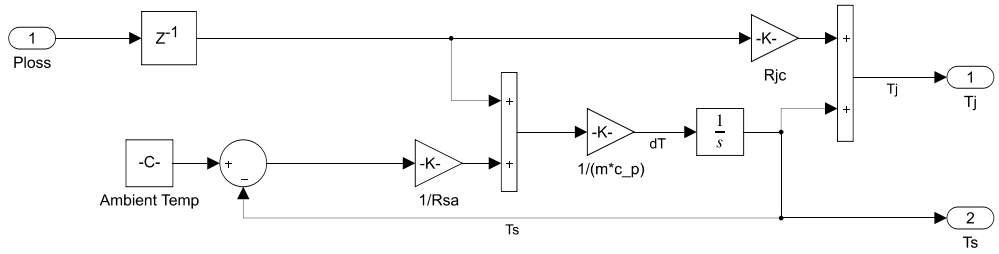


Figure B.7: DCAC Inverter Thermal.

```
function [Iphase, Idcac_in, Pdcac_loss, Pwire_loss, ma]...
    = DCAC(Vd, Vq, Id, Iq, Vbus, Tj, esc)

%reduce temp if pass interpolation range
if Tj > 140
    Tj = 140;
end
Rdson = interp1(esc.Ta_lu, esc.Rdson_lu,Tj); %MOSFET on resistance
Von_d = esc.Von_d;                           %diode forward voltage drop

% dq to abc transformation
Iphase = sqrt(Id^2+Iq^2) / sqrt(2);%per phase RMS current into the motor
Vterm = sqrt(Vd^2+Vq^2) / sqrt(2); %line-neutral RMS voltage into the motor

ma = Vterm * sqrt(2) * sqrt(3) / Vbus; %modulation index
%PMSM input power, Iq tells current direction, negative means regen mode
Ppmsm_in = 3*Iphase*Vterm * sign(Iq);
Pdcac_out = Ppmsm_in; %same as DC-AC output power
Paux_dcac = 6; %measured auxiliary power losses at no loads

%% MOSFET/Diode Parameters
Cgd1 = esc.Cgd1; % Crss for Vbus/2 to Vbus
Cgd2 = esc.Cgd2; % Coss for less than Vbus/2
Vmiller = esc.V_miller; % Miller plateau voltage
Rgon = esc.Ron_g;  % gate resistance for on
Rgoff = esc.Roff_g;  % gate resistance for off
Vgate = esc.V_g; % peak gate voltage

Qrr = esc.Qrr; % diode reverse recovery charge

```

```
%---------Calculated Currents----------
I_o = sqrt(2)*Iphase; %peak phase current
I_dc = I_o/pi;        %DC equivalent current

%% Switching times
tfu1 = (Vin - Ron_ds*I_dc)*esc.Ron_g*esc.Cgd1/(esc.V_g-esc.V_miller);
tfu2 = (Vin - Ron_ds*I_dc)*esc.Ron_g*esc.Cgd2/(esc.V_g-esc.V_miller);
tfu = (tfu1 + tfu2)/2;
tru1 = (Vin - Ron_ds*I_dc)*esc.Roff_g*esc.Cgd1/(esc.V_miller);
tru2 = (Vin - Ron_ds*I_dc)*esc.Roff_g*esc.Cgd2/(esc.V_miller);
tru = (tru1 + tru2)/2;

%% Mosfet Losses
EonM = Vbus*Idc*(tri+tfu)/2 + Qrr*Vbus; %MOSFET turn on energy
EoffM = Vbus * Idc * (tru+tfi)/2;       %MOSFET turn off energy

%average conduction loss per switch
Pon_fet = Rdson * Io^2 * (1/8 + ma*DisFac/(3*pi));
%average switching loss per switch
Pswitch_fet = (EonM + EoffM) * fswitch;

%average conduction loss per diode
Pon_d = Von_d * Io *(1/(2*pi)-ma*DisFac/8);
EonD = Qrr * Vbus/4;        %diode turn on energy
EoffD = 0;                  %absorbed in EonM
Pswitch_d = EonD * fswitch; %average switching loss per diode

Ploss_dcac = 6*(Pon_fet+Pon_d+Pswitch_fet+Pswitch_d); %total losses for 6

%% ESC Power and Efficiency calculations
Rwire = esc.R_w;                    %wire resistance assuming AWG10
Pwire_loss = Iphase^2 * Rwire * 3; %three phase wiring parasitics losses
%total power into DC-AC from DC bus
Pdcac_in = Pdcac_out + Ploss_dcac + Pwire_loss + Paux_dcac;

Pdcac_loss = Ploss_dcac + Paux_dcac;

%DC current into DC-AC, including the current from motor mechanical loss
Idcac_in = Pdcac_in/ Vbus;

end
```

B.4 Battery

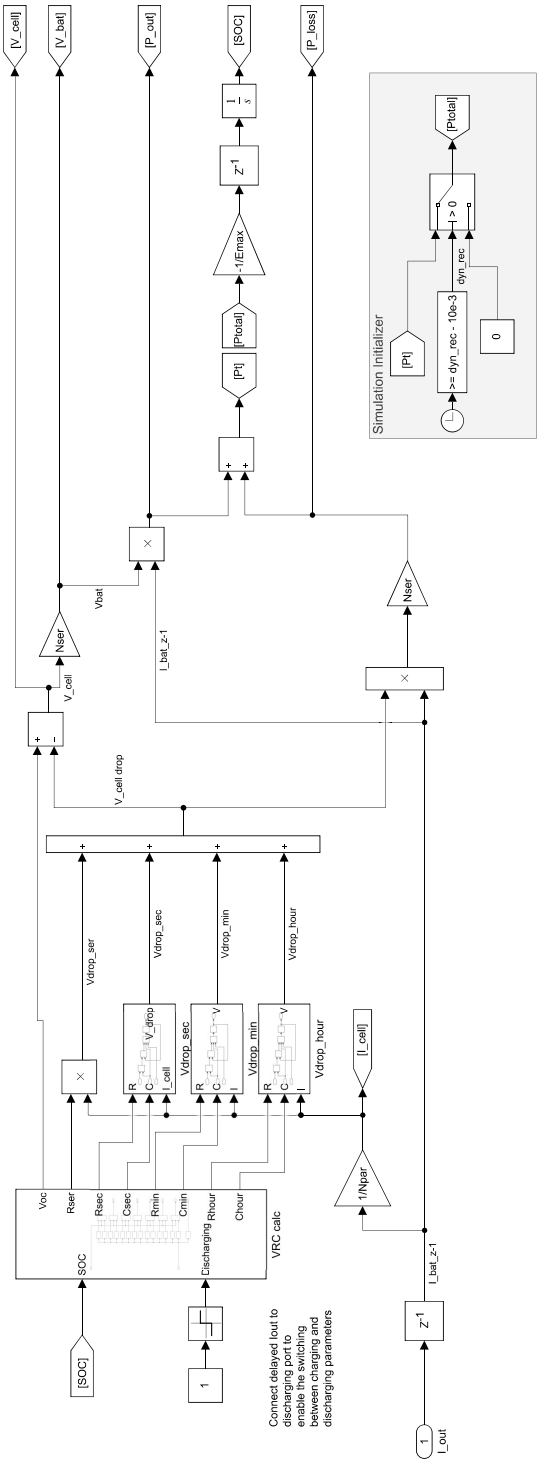


Figure B.8: Battery.

B.5 Per Motor to System Conversion

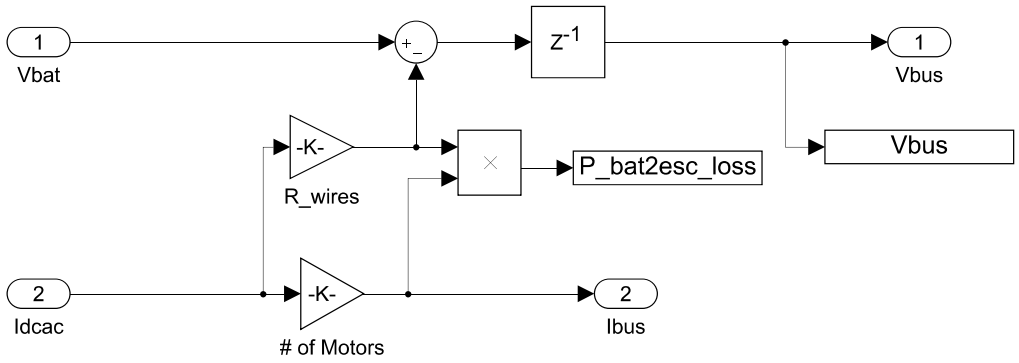


Figure B.9: Per Motor/Inverter to System Conversion.