Condition Based Monitoring and Protection in Electrical Distribution Systems

by

Uzoma Orji

B.S., Electrical Engineering
Massachusetts Institute of Technology (2006)
M.Eng., Electrical Engineering
Massachusetts Institute of Technology (2007)

Submitted to the Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degree of

Doctor of Philosophy

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

February 2014

Author ....[illegible].......
Department of Electrical Engineering and Computer Science
September 4, 2013

Certified by.......[illegible].......
Steven B. Leeb
Professor of Electrical Engineering and Computer Science
Thesis Supervisor

Accepted by ....[illegible].......
Leslie A. Kolodziejski
Chairman, Departmental Committee on Graduate Theses

Condition Based Monitoring and Protection in Electrical Distribution Systems

by

Uzoma Orji

Submitted to the Department of Electrical Engineering and Computer Science on September 16, 2013, in partial fulfillment of the requirements for the degree of Doctor of Philosophy

Abstract

The U.S. Department of Energy has identified "sensing and measurement" as one of the "five fundamental technologies" essential for driving the creation of a "Smart Grid". Consumers will need "simple, accessible..., rich, useful information" to help manage their electrical consumption without interference in their lives. There is a need for flexible, inexpensive metering technologies that can be deployed in many different monitoring scenarios. Individual loads may be expected to compute information about their power consumption. New utility meters will need to communicate bidirectionally, and may need to compute parameters of power flow not commonly assessed by most current meters. These meters may be called upon to perform not only energy score-keeping, but also assist with condition-based maintenance. They may potentially serve as part of the utility protection gear. And they may be called upon to operate in new environments, e.g., non-radial distribution systems as might be found on microgrids or warships. This thesis makes contributions in three areas of condition-based maintenance and protection in electric distribution systems.

First, this thesis presents a diagnostic tool for tracking non-integer harmonics on the utility. The tool employs a modified algorithm to enhance the capability of the Fast Fourier Transform (FFT) to determine the precise frequency of a newly detected harmonic. The efficacy of the tool is demonstrated with field applications detecting principal slot harmonics for speed estimation and diagnostics. Second, data from nonintrusive monitors have been shown to be valuable for power systems design. This thesis presents a new behavioral modeling framework developed for microgrid-style shipboard power system design using power observations from the ship's electrical distribution service. Metering can be used to inform new designs or update maintenance parameters on existing ship power systems. Finally, nonintrusive metering allows for new possibilities for adaptive fault protection. Adaptive thresholding of voltage magnitude, angle and harmonic content will be demonstrated for improving protection schemes currently used in ship electrical distribution systems.

Thesis Supervisor: Steven B. Leeb
Title: Professor of Electrical Engineering and Computer Science

Acknowledgments

I wish to thank Professor Steven B. Leeb for his guidance and his advising. I am also thankful for the inputs from my thesis readers, Professors James L. Kirtley and Leslie K. Norford.

Thanks are due to the many colllaborators on this research: Warit Wichakool, Chris Schantz, Al-Thaddeus Avestruz, Jim Paris, Bartholomew Sievenpiper, Katherine Gerhard, Chad Tidd, Darrin Barber, Christopher Laughman, John Cooley, Zachary Remscrim, Shahriar Khushrushahi, Andrew Paquette, Ashley Fuller, Greg Elkins, Sabrina Neuman, Jeremy Leghorn, Zachary Clifford, Rachel Chaney, Daniel Vickery, John Donnal, Jin Moon, Mark Gillman, Kawin Surabitbovom, Niaja Farve, Arthur Chang, USCGC Escanaba Crew, Jukkrit Noppakunkajorn, and Andrew Carlson.

Finally, I wish to acknowledge the support I have received from my friends and family. Their words of encouragment are deeply appreciated.

This research was supported by the Office of Naval Research (ONR) Structural Acoustics Program led by program manager Debbie Nalchajian and the Grainger Foundation.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

When the U.S. Department of Energy identified "sensing and measurement" as one of the "five fundamental technologies" essential for driving the creation of a "Smart Grid", one of the goals included a "deployment of 'smart' technologies ... for metering, communications concerning grid operations and status, and distribution automation." Other goals included "integration of 'smart' appliances and consumer devices" and "development and incorporation of demand response, demand-side resources, and energy-efficiency resources" [1]. The Smart Grid would require "smart" metering devices and communication networks to collect and deliver necessary information about the power system for the operation of the power grid to the consumer. Consumers will need quick, easy access to "simple, accessible ..., rich, useful information" to help manage their electrical consumption without interference in their lives.

There is also a need for flexible, inexpensive metering technologies that can be deployed in many different monitoring scenarios. Such smart devices should be able to provide not only just the total power consumed but could also give diagnostic parameters of loads from the electrical signals. Diagnostic parameters such as the rotor speed of an induction motor and high frequency variation of the power consumption can be extracted by smart meters to provide additional information about the health of these loads.

As these demands become apparent for a smarter grid, new technologies will be needed for efficient control and access to power systems. These include the need for diagnostic systems to track pathological energy consumption, new design techniques for smaller or "islanded" power systems and new protection schemes. This thesis looks to make contributions in these areas.

1.1 Contributions

Extensive research has been in condition based monitoring by tracking speed, vibration, temperature, pressure and other metrics to monitor monitor faults such as broken rotors bars, damaged bearings, rotor eccentricity and shaft speed oscillations. These methods are not only limited for single loads but also may require intrusive installation of external sensors. This thesis will develop methods to track harmonics using power measurements from electrical signals in single loads and will also extend these methods for multiple loads.

Second, data from nonintrusive monitors have been shown to be valuable for power systems design. This thesis presents a new behavioral modeling framework developed for microgrid-style shipboard power system that improves upon existing methods. The framework makes use of using power observations from the ship's electrical distribution service to develop stochastic models used in simulating the total load of a ship. Metering can be used to inform new designs or update maintenance parameters on existing ship power systems.

Finally, nonintrusive metering allows for new possibilities for adaptive fault protection in zonal electrical distributions. Current methods use static threshold levels for fault detection which may be insufficient depending on type of fault or operating condition. This thesis presents an adaptive fault detection which uses the data from nonintrusive for improved fault detection. Adaptive thresholding of voltage magnitude, angle and harmonic content will be demonstrated for improving protection schemes currently used in ship electrical distribution systems.

1.2 Organization

A background information on previous work on nonintrusive load monitoring (NILM) is presented in Chapter 2. Hardware improvements were made to NILM to allow for tracking of small harmonics in the electrical signals used for condition-based maintenance (CBM) through electrical monitoring.

Chapters 3 and 4 present methods for estimating speed and vibration by tracking harmonics in the electrical signals for loads. Experiments and results are presented to show the results of the methods. The NILM environment allows for CBM for multiple loads as will be explained. Chapter 5 discusses CBM for vibration using acceleration when tracking harmonics in the electrical signals becomes infeasible.

Chapter 6 describes the framework that can be used for future ship for improved reliability of electric distribution systems. A software graphical user interface (GUI) is shown to implement the design of the framework.

Chapter F presents a testbench shipboard dual-generator setup and Chapter 7 describes the implementation of a multi-function monitor in the protection of a testbench MVAC zonal electrical distribution system. Computer simulation is used to shown the efficacy of the improved algorithm for zonal protection.

Finally, Chapter 8 summarizes and gives recommendations for future work.

Chapter 2

Nonintrusive Load Monitor (NILM)

The nonintrusive load monitor has been demonstrated [2–5] as an effective tool for evaluating and monitoring electro-mechanical systems through analysis of electrical power data. The power distribution network can be pressed into "dual-use" service, providing not only power distribution but also a diagnostic monitoring capability based on observations of the way in which loads draw power from the distribution service. A key advantage of the nonintrusive approach is the ability to reduce sensor count by monitoring a collections of loads. A pictorial representation of the NILM is shown in Fig. 2-1

Nonintrusive electrical monitoring has been described in [7, 8] and in other publications. The systems that are described in these papers can be split into two broad categories: transient and steady-state approaches. The transient approach [8] finds loads by examining the full detail of their transient behavior. Reference [2] describes a platform for transient-based nonintrusive load monitoring appropriate for many applications. The following sections will discuss the data acquisition, preprocessor and event detector modules.

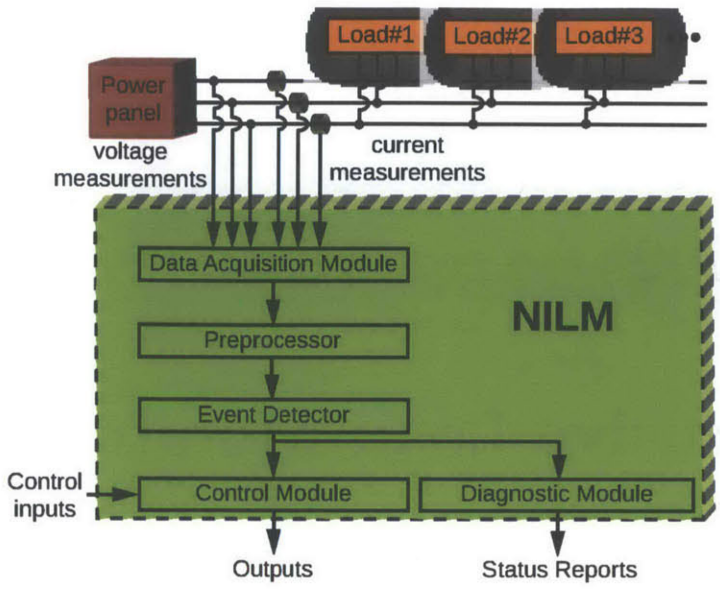


Fig. 2-1: A block diagram of the nonintrusive load monitor (NILM) [6].

2.1 Data Acquisition

NILM experiments from previous research collect data from a current sensor with only analog filtering for anti-aliasing. There is a large 60 Hz line frequency component that dominates the current signal, making detection of the smaller harmonics in the current discussed in chapters 3 and 4 more difficult. To improve the detectability of the harmonics, a 60 Hz notch filter is implemented to remove the large line frequency component before the data acquisition hardware in the NILM samples the current.

In Fig. 2-2, the stator current signal is sent through a 60 Hz notch filter. The output is amplified by a gain stage and later filtered by a passive antialiasing filter. The output buffer drives the input of the NILM data acquisition hardware. The notch filter stage allows for improved signal detectability of the smaller slot harmonics. By removing the large dominant line frequency, the smaller harmonic signals can then be amplified, increasing the overall signal-to-noise ratio by reducing the effect of quantization noise in the analog-to-digital converter (ADC) of the NILM.

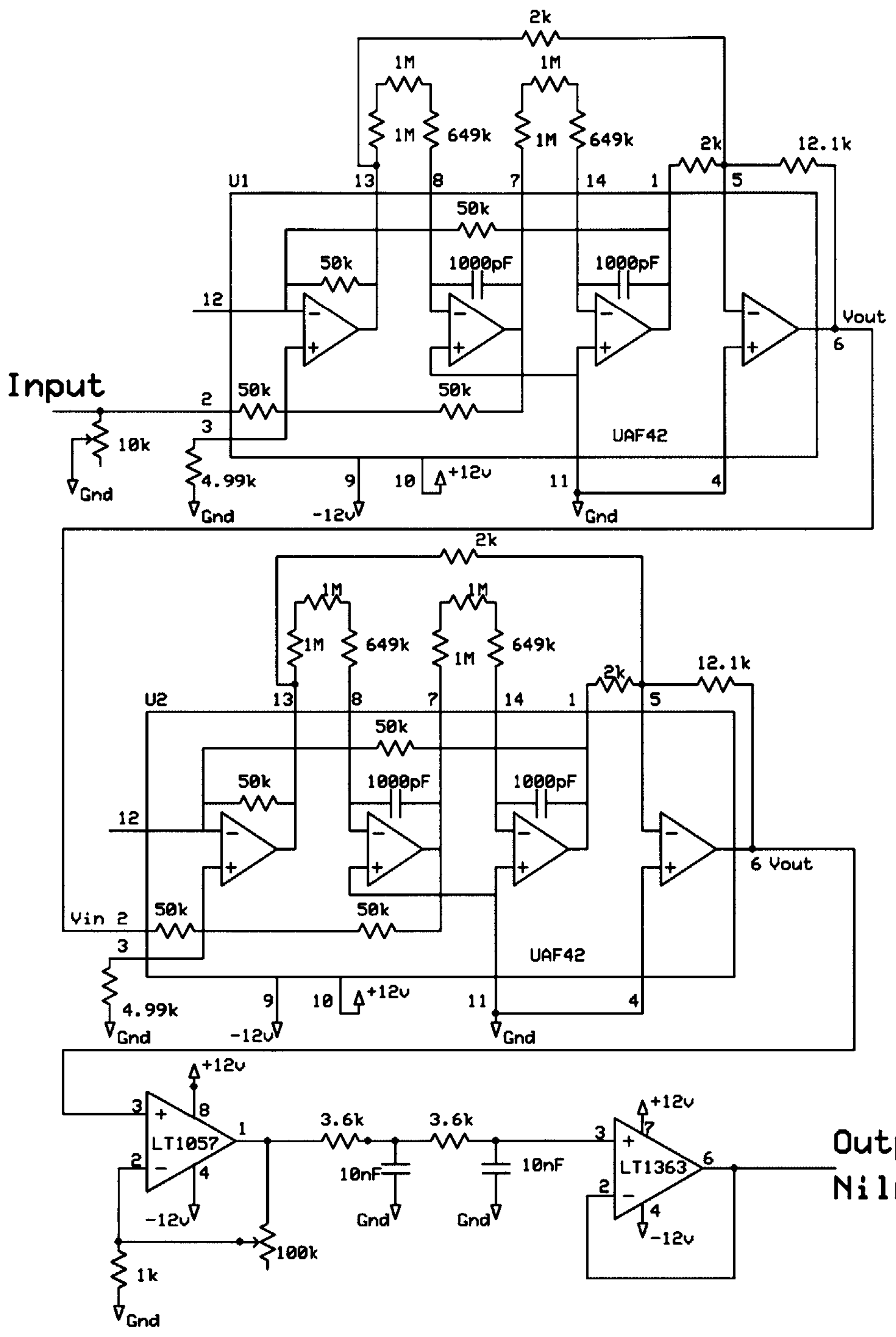


Fig. 2-2: Schematic of the 60 Hz notch filter circuit. The circuit notches the 60 Hz frequency, amplifies the signal and sends the signal through an antialiasing filter.

2.2 Preprocessor Module

The NILM detects the operation of individual loads in an aggregate power measurement by preprocessing measured current and voltage waveforms to compute spectral envelopes [9]. Spectral envelopes are short-time averages of the line-locked harmonic content of a signal. For an input signal $x(t)$ of current, the in-phase spectral envelopes a_k of x are

$$a_k(t) = \frac{2}{T}\int_{t-T}^{t} x(\tau)\sin(k\omega\tau)d\tau, \tag{2.1}$$

where k is the harmonic index. The quadrature spectral envelopes b_k are

$$b_k(t) = \frac{2}{T}\int_{t-T}^{t} x(\tau)\cos(k\omega\tau)d\tau. \tag{2.2}$$

These spectral envelopes may be recognized as the coefficients of a time-varying Fourier series of the input waveform. For transient event detection on the ac utility, the time reference is locked to the line so that fundamental frequency spectral envelopes correspond to real and reactive power in steady state. Higher spectral envelopes correspond to line frequency harmonic content. A high performance transient event detection algorithm [8, 10] is available to disaggregate the fingerprints or spectral envelope signatures of individual loads in the aggregate measurement. This transient event detection is critical for pre-trigger and post-trigger event detection described in §2.3.

2.3 Event Detector Module

By computing line-locked spectral envelopes from input signals, NILM can detect the activation of a motor of interest. Special attention can then be paid to the aggregate current frequency content just before and just after this turn-on transient, as will be shown in Chapters 3 and 4, to identify important frequency content that occurs at frequencies that are not line-locked.

Reference [9] introduces a nonintrusive transient classifier (NITC) program shown in block diagram form in Fig. 2-3. The relevant parts of the NITC routine are slightly modified for the purposes of this thesis.

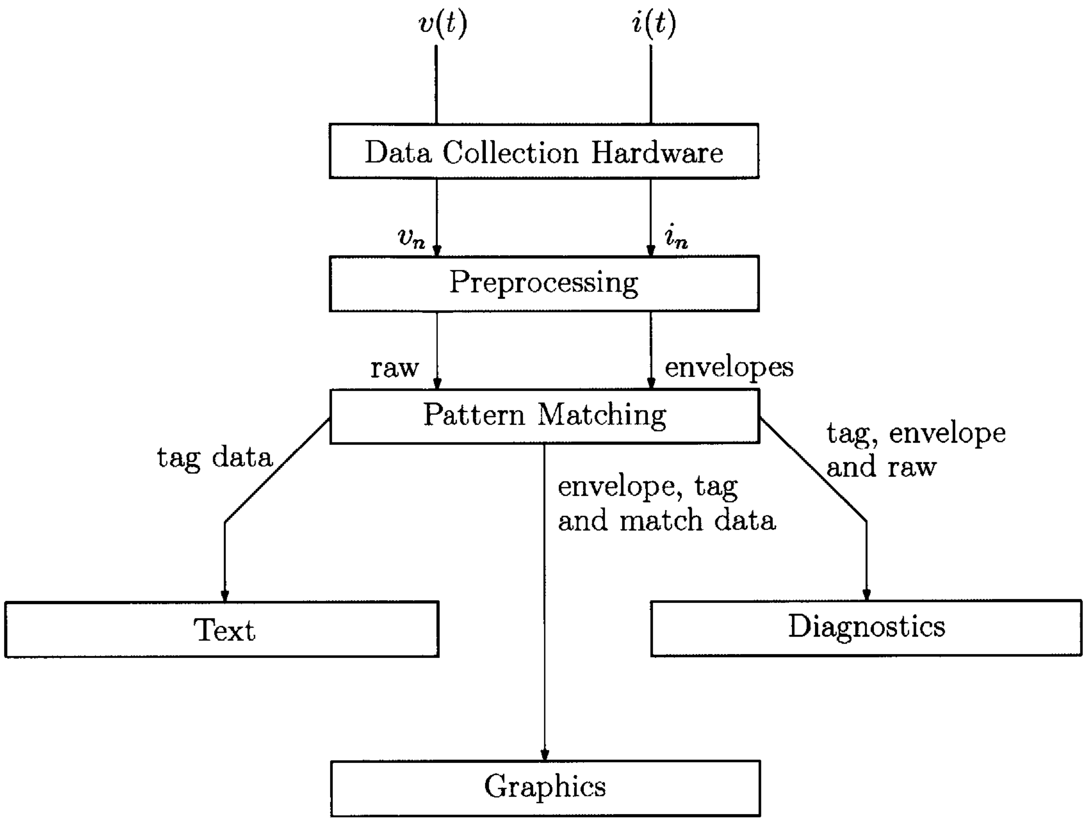


Fig. 2-3: Original nonintrusive transient classifier (NITC) block diagram. The original NITC main program preprocesses, pattern matches and produces output for three data queues [9].

The updated NITC block diagram is shown in Fig. 2-4.

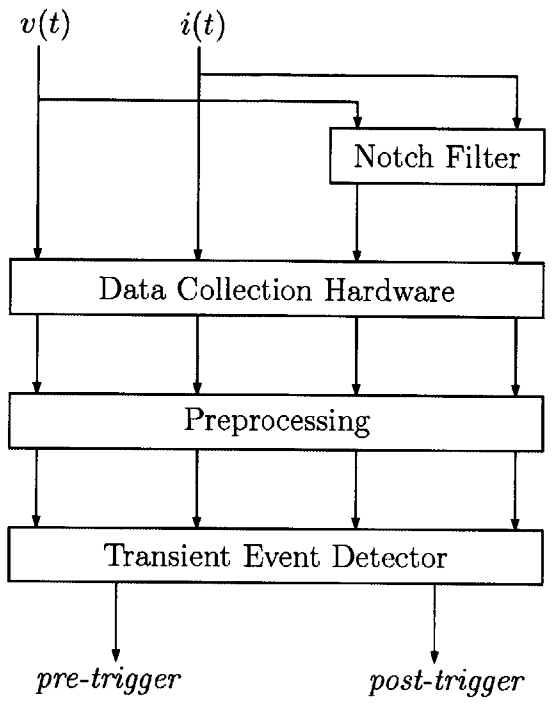


Fig. 2-4: Updated nonintrusive transient classifier (NITC) block diagram. This version includes the filter and outputs the *pre-trigger* and *post-trigger* data.

Once the unfiltered data has been collected by the hardware, the data is preprocessed to produced the spectral envelopes. The pattern matching routine identifies a trigger event once a motor has turned on. The outputs of this pattern matching routine are now the filtered raw data before and after this trigger event. Once the *pre-trigger* and *post-trigger* data are saved, they can be used to identify frequency content used for condition based monitoring in Chapters 3 and 4.

Chapter 3

Speed Detection Using Slot Harmonics

Harmonic analysis of motor current has been used to track the speed of motors for sensorless control. Algorithms exist that track the speed of a motor given a dedicated stator current measurement, for example [11–15]. Harmonic analysis has also been applied for diagnostic detection of electro-mechanical faults such as damaged bearings and rotor eccentricity [16–27].

Rotor slot harmonics are widely used in nonintrusive speed detection algorithms used in many control applications. The ability to estimate the speed of an electric machine from its electrical signals provides a method that does not require the installation and maintenance of sensors or any other hardware. This chapter discusses the limitations of previous research that use these rotor slot harmonics for speed tracking. The improved method developed in this research provides a routine that can estimate the speed with a reduced number of line cycles for faster updates. This chapter concludes with examples that require speed estimation for various applications.

3.1 Summary of Diagnostic Harmonics

Induction motors are vital components of many industrial processes. Extensive research is being done to develop techniques to monitor these motors to minimize the risk of unexpected system failures and reduced motor lifetime. Generally, condition monitoring algorithms have focused on the stator, the rotor or the bearings for sensing certain failures. Most of the recent research is now focused on the electrical monitoring of the stator current spectrum of the motor to sense various faults [16] as discussed below.

3.1.1 Air-Gap Eccentricity

Two methods have been proposed for detecting air-gap eccentricity. Reference [26] monitors the sideband frequencies of slot harmonics located at

$$f_{slot+ecc} = f\left[(kR + n_d)\frac{1-s}{p} + \nu\right], \tag{3.1}$$

where f is the supply frequency; $k = 0, 1, 2, \ldots$; R is number of rotor slots; $n_d = 0, \pm1, \ldots$ is the order of rotor eccentricity; s is the per unit slip, S is the speed in rotations per minute (rpm), p is the number of pole pairs and $\nu = \pm1, \pm3, \ldots$ is the stator MMF harmonic order [15, 28]. For specific values of k, n_d and ν, the corresponding slot harmonics can be used to track speed of the induction motor. A speed detection scheme using these harmonics is discussed later in this chapter.

The second method, used in [18], searches for fundamental sidebands of the supply frequency. These eccentricity harmonics are located at

$$f_{ecc} = f\left[1 \pm m\left(\frac{1-s}{p}\right)\right], \tag{3.2}$$

where $m = 1, 2, 3, \ldots$.

3.1.2 Shaft-Speed Oscillation

Shaft-speed oscillation is a mechanical fault that can be enhanced by certain rotor imbalances. An improperly mounted motor or an unbalanced fan can accentuate shaft-speed oscillation frequencies [18]. The frequencies of interest are predicted by

$$f_{sso} = f\left[k\left(\frac{1-s}{p}\right) \pm \nu\right]. \tag{3.3}$$

Chapter 4 discusses how the shaft-speed oscillation harmonics can be used for vibration analysis.

3.1.3 Bearings Damage

Misalignment of the bearings can produce physical damage of the raceways which house the bearings. The resulting mechanical displacement causes the air gap to vary and the eccentricity harmonics are located at the following frequencies given by

$$f_{bng} = \left|f \pm m f_{i,o}\right|, \tag{3.4}$$

where $m = 1, 2, 3, \ldots$ and $f_{i,o}$ is a characteristic vibration frequency based on the dimensions of the bearings. The location of the vibration frequencies are given by

$$f_{i,o} = \frac{n}{2} f_r \left[1 \pm \frac{bd}{pd}\cos\beta\right], \tag{3.5}$$

where n is the number of bearing balls, f_r is the mechanical rotor speed in rotations per second, bd is the ball diameter, pd is the bearing pitch diameter, and β is the contact angle of the balls in the raceway [22].

This thesis does not explore diagnostics using these bearings frequencies. The following section discusses the slot harmonics derived from (3.1).

3.2 Rotor Slot Harmonics

Rotor slot harmonics present in the stator current of a motor arise from the interaction between the permeance of the machine and the magnetomotive force (MMF) of the current in the stator windings. As the motor turns, the rotor slots alter the effective length of the airgap sinusoidally, thereby affecting the permeance of the machine. This sinusoidal behavior is seen in the flux, which is the product of the MMF and the permeance across the airgap. The odd harmonics present in the stator current introduce additional harmonics. Static and dynamic eccentricity harmonics also appear in the stator current as the rotor turns irregularly in relation to the stator.

The slot harmonics, including the principal slot harmonic (PSH), are located at frequencies

$$f_{sh} = f\left[R\frac{1-s}{p} + \nu\right] = f\left[R\frac{S}{3600} + \nu\right], \tag{3.6}$$

where f is the supply frequency; R is number of rotor slots; s is the per unit slip, S is the speed in rotations per minute (rpm), p is the number of pole pairs and $\nu = \pm1, \pm3, ...$ is the stator MMF harmonic order [15, 28]. These harmonics are located in the family of harmonics given by (3.1) for $k = 1$ and $n_d = 0$. For the data presented in this thesis, there was little rotor imbalance so the most visible slot harmonics are given by (3.6).

When the locations of these speed-dependent harmonics are found, the speed of the electric machine can be calculated rather easily. A full discussion on how these speed-dependent harmonics appear in the stator current can be found in Appendix A.

A substantial amount of literature makes use of (3.6) for speed detection [11–13, 15, 29, 30]. To illustrate, the frequency spectrum of 5 seconds of current from a motor with its slot harmonics (3.6) is shown in Fig. 3-1 for different values of ν. The motor used was a three-phase machine with $R = 48$ rotor slots and $p = 3$ pole pairs loaded by a dynomometer to run at $s = 0.0171$ or 1180 rpm. Typically, the slot harmonics with $k = 1$, referring to the fundamental harmonic of the stator current, and with $n_d = 0$, referring to the case in which the motor has no eccentricity, are the most pronounced in the current spectrum [11, 31]. The PSH corresponds to the harmonic with $\nu = 1$. For a given n_d, the slot harmonics differ exactly by $2f$ in (3.6).

Analog techniques [29, 30] have been implemented to track these harmonics. The

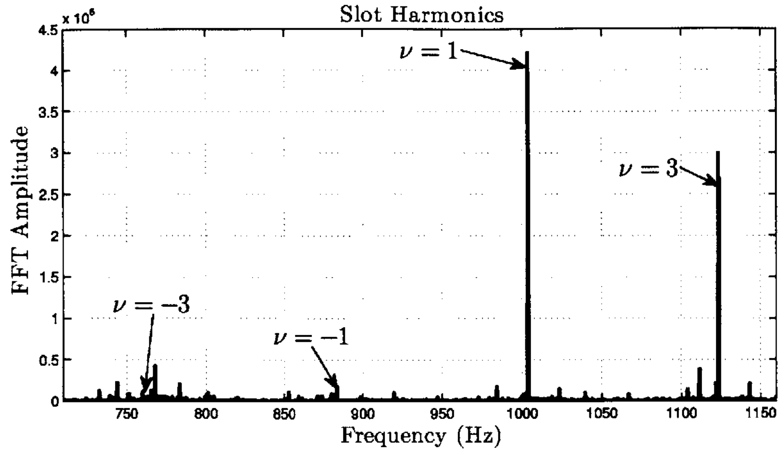


Fig. 3-1: Slot harmonics for a motor with the following parameters: f=60, k=1, R=48, n_d=0, s = 1.71% or 1180 rpm. The harmonics shown in the figure are labeled with the corresponding value of ν.

performance of these approaches can be limited in terms of accuracy, linearity, resolution, speed range, or speed of response [14]. Any analog filtering can require extremely complex circuitry and any output signal can be corrupted by noise. Digital techniques employing the Fast Fourier Transform (FFT) were developed [12] to overcome the flaws in the analog methods. These FFT methods were limited by the uncertainty principle, i.e. the trade-off between high frequency resolution and the response time to changes of speed that deteriorates with long data records. Parametric estimations [11, 13, 15] of the current spectrum were used to overcome the limitations of FFT by attempting to model the process that samples the data using *a priori* knowledge. However, these estimations require digital filters which make these methods less robust than the FFT. With a high stator frequency, the longer computations times can reduce any advantage these parametric methods may have over the FFT.

Some research has been done to combine the FFT and parametric estimation methods [11, 15]. In [11], the techniques do a successful job in tracking the slot harmonics in estimating speeds in a controlled environment. The authors make use of the periodicity of the slot harmonics by aliasing the spectrum such that these harmonics line up to increase detectability.

There are certain trade-offs that must be made when deciding on the proper methods for detecting these speed-dependent rotor slot harmonics. The choices are often dictated by the practical setting. In the experiments conducted in this research, it is observed that the principal slot harmonic (PSH) is the most pronounced slot harmonic in the motors. The methods discussed here, therefore, will only search for the PSH, which simplifies the complexity of the algorithm unless otherwise specified. Furthermore, the method combines the powers of the FFT and a search routine to find the best estimate for the true location of the PSH. Moreover, previous work could only estimate the speed of a single motor [11–15]. This method can estimate speed of multiple motors in a multi-motor environment. A full discussion follows below.

3.3 Practical Limitations on Slip

For high-efficiency induction motors, the slip s usually does not exceed 5% and possibly less. This assumption leads to interesting simplifications when searching for the principal slot harmonic (PSH). The PSH refers to the slot harmonic with $\nu = 1$, $k = 1$, and $n_d = 0$ in (3.6) and is used in speed detection algorithms since it is often the most pronounced [11, 31]. The slot harmonics for different values of ν differ by $2f = 120$ Hz. If the principal slot harmonic was confined to a frequency window of width 120 Hz under these practical limitations of slip, there would be no ambiguity in determining the window in which the PSH is located.

For example, Table 3.1 tabulates the maximum slip for different values of R and p for which the PSH would be confined to a 120 Hz wide frequency window. For a motor with $R = 60$ and $p = 1$, a maximum slip of $s = 0.033$ would need to be assumed in order to constrain the PSH to a 120 Hz wide window. This assumption would be unreasonable because it would be possible for the motor to be running with a slip of 0.04. In this situation, there would be some ambiguity in determining in which window the PSH lies.

On the other hand, a motor with $R = 48$ and $p = 3$ can have a maximum slip of $s = 0.125$ to unambiguously determine the window of the PSH. If the motor were running with no slip ($s = 0$), the principal slot harmonic would be located at 1020 Hz.

Table 3.1: Maximum Slip for Unambiguous Speed Estimation for Several Values of R and p

	P			
R	1	2	3	4
16	0.1247	0.2494	0.3742	0.4989
20	0.0997	0.2000	0.3000	0.4000
24	0.0833	0.1667	0.2492	0.3333
28	0.0714	0.1428	0.2142	0.2856
32	0.0622	0.1244	0.1875	0.2489
34	0.0586	0.1172	0.1758	0.2344
40	0.0500	0.1000	0.1500	0.1989
44	0.0453	0.0906	0.1358	0.1811
48	0.0417	0.0833	0.1250	0.1667
52	0.0383	0.0767	0.1150	0.1533
56	0.0356	0.0711	0.1067	0.1422
60	0.0333	0.0667	0.1000	0.1333

If the motor were running with slip ($s = 0.125$), the principal slot harmonic would be located at 900 Hz. Therefore, it can be assumed that the principal slot harmonic will be located in the 120 Hz window between 900 Hz and 1020 Hz. This slip satisfies any reasonable low-slip assumptions.

3.4 Speed Estimation via Slot Harmonics

Using a "low-slip" assumption in which the PSH is restricted to a single 120 Hz wide frequency window, this section will describe the application of the slot harmonics in determining speed of operation just after startup in a multi-load/multi-motor environment. All Matlab code for this optimized speed estimation method is shown in Appendix B. To demonstrate the effectiveness of the algorithm, the speed estimation method was conducted with the following two motors. The first motor (Motor 1) was a three-phase motor from an HVAC evaporator in an air-handling unit. Motor 1 had $R = 48$ rotor slots and $p = 3$ pole pairs. The second motor (Motor 2) is a single-phase line-to-line machine from a fresh-air ventilation unit with $R = 34$ rotor slots and $p = 1$ pole pairs.

Consider an illustrative example in Fig. 3-2 in which the transient responses of induction motors are shown as they turn on and off. The region labeled A is

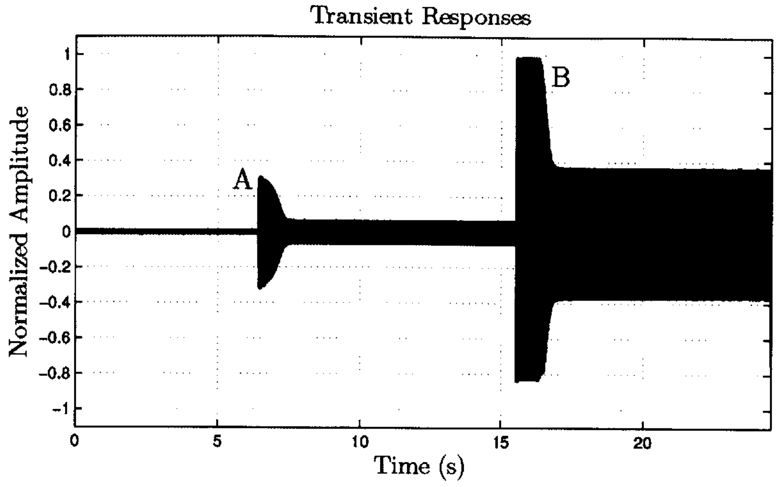


Fig. 3-2: Electrical current data stream with transient responses of induction machines.

when Motor 1 (the evaporator motor) turns on. Zooming in on Region A shows the transient response of Motor 1 as shown in Fig. 3-3. The region labeled B in Fig. 3-2

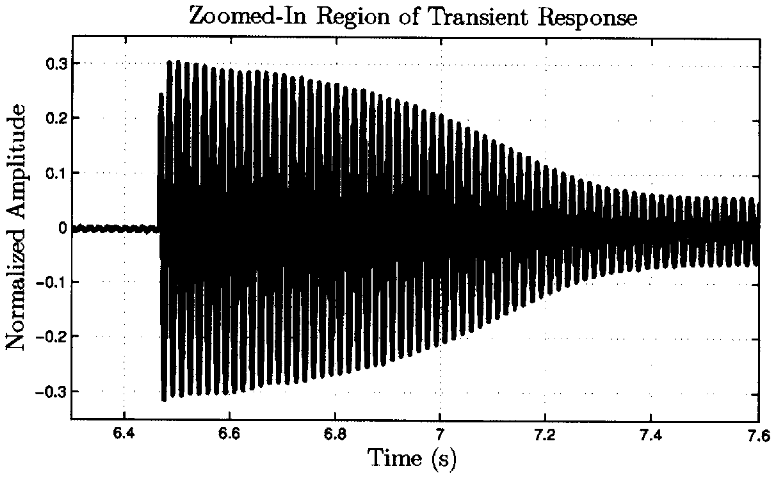


Fig. 3-3: The turn-on transient when Motor 1, labeled Region A in Fig. 3-2.

marks the region in which Motor 2 turns on so that both motors are running. The spectral envelope corresponding to "real" power as calculated by the NILM described in §2.2 is shown in Fig. 3-4.

For the motors used in these experiments, Table 3.1 confirms that they satisfy the low-slip assumption. The PSH for Motor 1 will be between 900 Hz and 1020 Hz while the PSH for Motor 2 will be between 1980 Hz and 2100 Hz using (3.6).

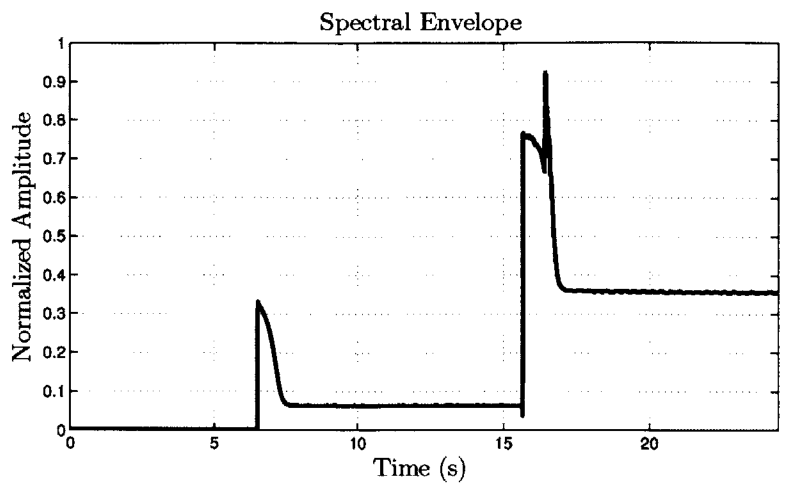


Fig. 3-4: Spectral envelope calculation.

As Motor 1 turns on, the NILM characterizes this new load from its transient turn-on response and spectral envelope as described in Chapter 2. After Motor 1 has turned on, the window of interest to locate the PSH for Motor 1 is shown in Fig. 3-5. Using 5 seconds of current data, sampled at 7800 Hz, the location of the PSH can

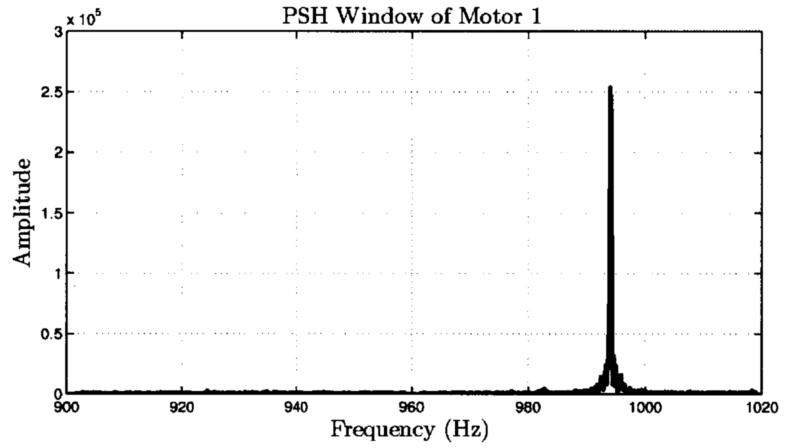


Fig. 3-5: FFT frequency content of the current between 900 Hz and 1020 Hz after Motor 1 has turned on and reached steady state.

be estimated to be 994 Hz by finding the location of the maximum value within the window. Using (3.6), the speed can be estimated to be 1167.500 RPM.

By taking the maximum value within the PSH window, the resolution of the estimate is limited to the size of the FFT frequency bins, which is determined by the sampling time. The estimate of the PSH (and the motor's speed) can be refined even further. As shown in Fig. 3-6, the energy of the PSH is actually spread over several frequency bins implying that there is not enough resolution to determine the PSH precisely [32]. One way to obtain finer resolution would be to sample data over a

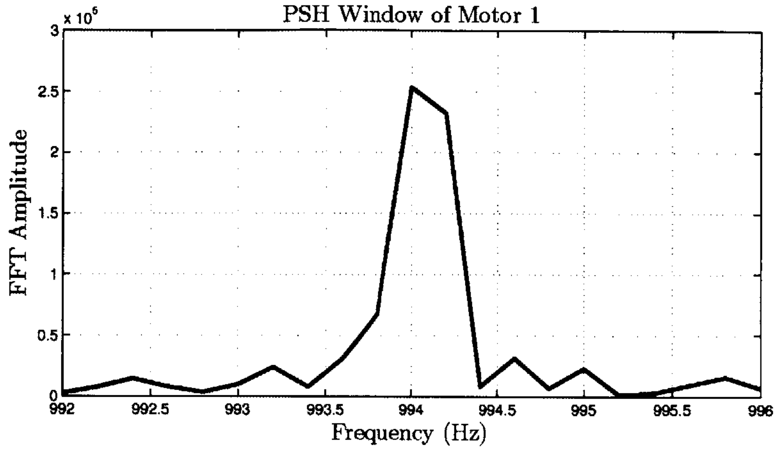


Fig. 3-6: Current spectrum with with Motor 1 running. The frequency content of the PSH is spread over multiple bins.

longer interval, which is unattractive as the speed of the motor may vary during the interval. Furthermore in control systems that require constant speed updates, longer sampling times may negatively impact the overall performance of the controller. A different approach is to use the information in the frequency bins near the peak.

Consider the example in Figure 3-7 of the FFT of 5 seconds of a pure sine wave with a frequency of 994.34 Hz. For longer sampling times, the frequency resolution is

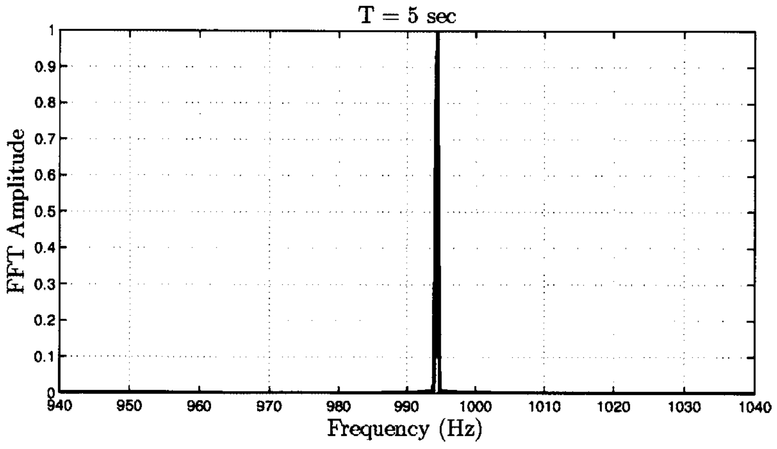


Fig. 3-7: The FFT spectrum is shown for the sinusoid with a sampling time of 5 seconds.

fine enough such that the underlying frequency can be resolved relatively accurately. Compare this result with that of Fig. 3-8 and Fig. 3-9 in which the data length is reduced to 0.5 seconds and 0.05 seconds respectively. The energy of the signal is spread over an increasingly wider frequency band as the data duration decreases. The frequency resolution becomes so coarse that a reliable speed estimate based on maximum value would be poor.

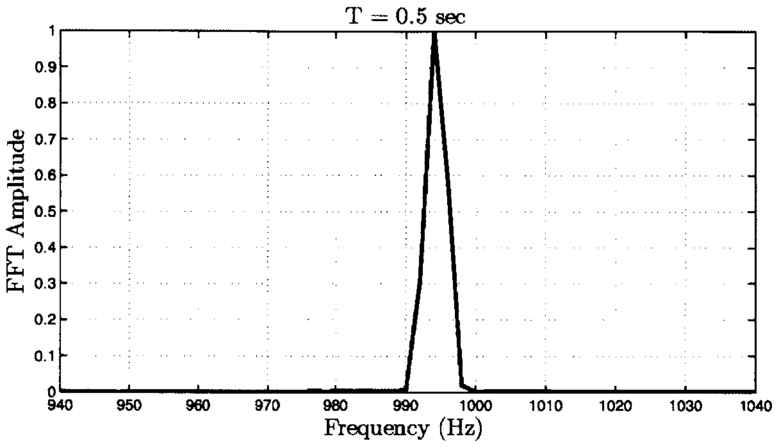


Fig. 3-8: The FFT spectrum is shown for a sinusoid with a frequency of 999.34 Hz with a sampling time of 0.5 seconds. As expected, the energy from the sinusoid is spread over multiple bins.

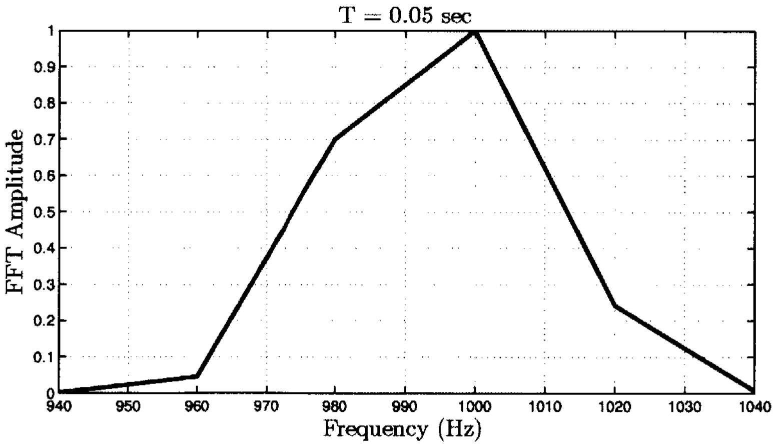


Fig. 3-9: The FFT spectrum is shown for the sinusoid with a frequency of 999.34 Hz with a sampling time of 0.05 seconds. The frequency resolution is very coarse that a reliable speed estimate based on maximum value would be poor.

When dealing with the short, 0.05 second long data, each bin of the resulting FFT is 20 Hz wide. Estimating the frequency (994.34 Hz, in this example) from simply looking at the peak within a window could only hope to provide resolution of 20 Hz. The situation can be improved by producing 0.05 seconds of data from each of a family of sine waves with frequencies in the neighborhood of the observed peak. An optimization routine can pick the frequency and amplitude of a sinusoid whose FFT is the best fit (in the minimum squared error sense) for the observed data over the entire frequency neighborhood around the peak. This routine makes the assumption that there is only a single, pure sinusoid in the window responsible for all observed frequency content. For each frequency it loops through, a sine wave is generated and multiplied by a corresponding scale factor. The scale factor is the quotient of the spectral energy of the actual data and that of the unscaled sine wave. The frequency (and scale factor) that produces the least amount of error is the output of the routine. Appendix B shows the Matlab code used to implement the optimation routine. In this case, the optimization routine selects exactly 994.34 Hz as there is no noise to corrupt the data. Essentially, a finer estimate can be made by using the energy that has spread over several bins as opposed to only making use of the content in just one bin.

This same idea can be applied to the actual data taken from Motor 1. All of the following data is taken at a sampling frequency of 7800 Hz. Using the full 5 seconds length of data, the estimate of the PSH for Motor 1 is 994.100 Hz. A speed estimate of 1167.625 RPM is then calculated from this estimate of the PSH. The results of applying the optimization routine to shorter lengths of data are shown in Fig. 3-10 where the frequency content for 0.5 seconds of data of Motor 1 near the PSH is displayed.

The optimization routine finds the the frequency of a sinusoid whose FFT best matches the observed data, in this case 994.420 Hz. Again, the optimization routine is making the assumption that there is only a single, pure sinusoid responsible for all new observed frequency content in the post-trigger window. If there is no load turned on in the pre-trigger window, the routine can assume that the observed frequency content in the post-trigger window comes from one sinusoid. The dotted line in Fig. 3-10 is the FFT of a 0.5 second sinusoid with a frequency of 994.420 Hz. The FFT of this

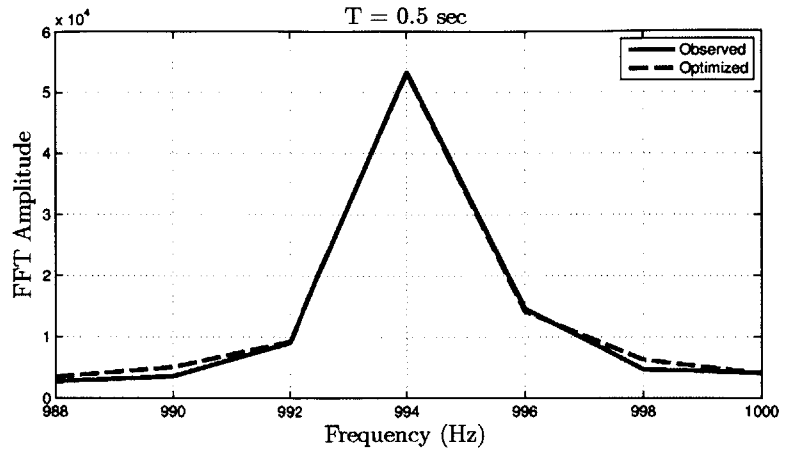


Fig. 3-10: The observed PSH is shown in the solid line. The dotted line is the FFT of the best-fit sinusoid. The sampling time is 0.5 seconds.

sinusoid closely matches that of the observed data (solid line) in the region of interest. A speed estimate of 1168.025 RPM is then calculated from the optimized estimate of the PSH. This is repeated for 0.05 seconds of data in Fig. 3-11. The optimized PSH is 994.980 Hz and the corresponding speed estimate is 1168.725 RPM.

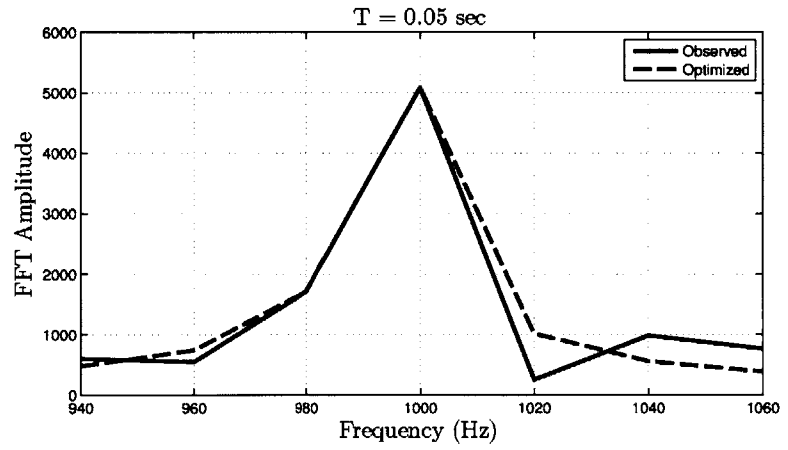


Fig. 3-11: The observed PSH is shown in the solid line. The dotted line is the FFT of the best-fit sinusoid. The sampling time is 0.05 seconds.

By utilizing more data from the nearby frequency bins, and the assumption that only a single sinusoid is responsible for the observed frequency content, the routine can predict speed estimates on a smaller data set. Table 3.2 displays the results of the optimizing routine for different sampling times.

Not only can this speed detection estimate speeds of a single motor with small data

Table 3.2: Optimization Routine PSH and Speed Estimates at Different Sampling Times

T (sec)	Line Cycles	PSH (Hz)	Speed (RPM)
5	300	994.100	1167.625
3	180	994.111	1167.639
1	60	994.240	1167.800
0.5	30	994.420	1168.025
0.1	6	994.500	1168.125
0.05	3	994.980	1168.725
0.0333	2	982.600	1153.250
0.0166	1	995.640	1169.550

records, NILM allows for the capability of detecting many motors in a multi-motor environment described in §2.3. When Motor 2 turns on, there are scenarios that may complicate tracking speeds. If both motors are identical in parameters, the worst case scenario would have both motors running at the same speed. In such a case, the principal slot harmonic (PSH) of each motor would be at the exact same location. To prevent this, if possible, the motors can be selected so that such a scenario could not occur. For example, Motor 1 was an intake ventilation unit taken off a US Coast Guard ship. The exhaust ventilation unit used on the ship was an identical motor. Tracking the speeds of both of these machines would be difficult if the slip of both machines are running at the same speed. However, for new ships, motors can be selected with different number of rotor bars such that their principal slot harmonics would appear in separate windows.

Another potential problem that could complicate speed detection is the utility line having its own set of harmonics which are visible in the current spectrum. The frequency spectrum of the utility line is shown in Fig. 3-12.

The 17th line harmonic at 1020 Hz could be troublesome in trying to identify the PSH of Motor 1. Fortunately, as these undesirable harmonics are located at integer multiples of 60 Hz, they can be filtered out using the digital filter $y[n] = x[n] - x[n-N]$ where N is the number of sample points per 60 Hz line cycle. In this experiment, all data is sampled at 7800 Hz, which corresponds to $N = 130$. This filter will notch out all the integer multiples of the line frequency.

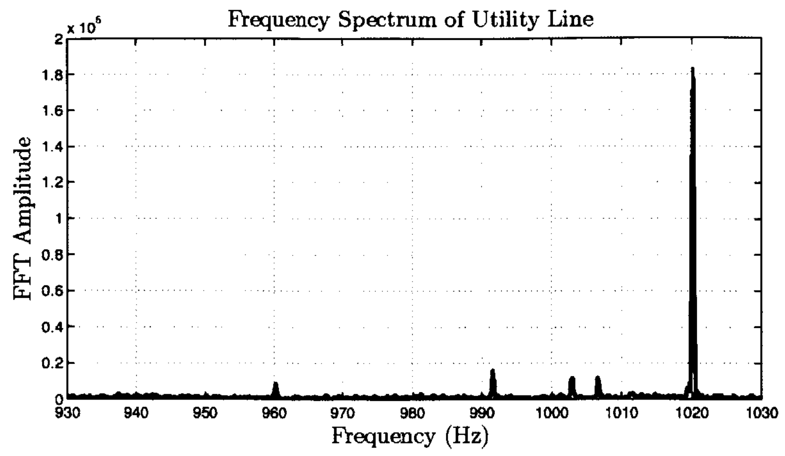


Fig. 3-12: Frequency spectrum of the voltage from the utility line with inherent distortions.

The PSH for Motor 2 was determined to be between 1980 and 2100 Hz. The current spectrum of the pre-trigger window when only Motor 1 was running is shown in Fig. 3-13. There are no harmonics caused by any line distortions that appear

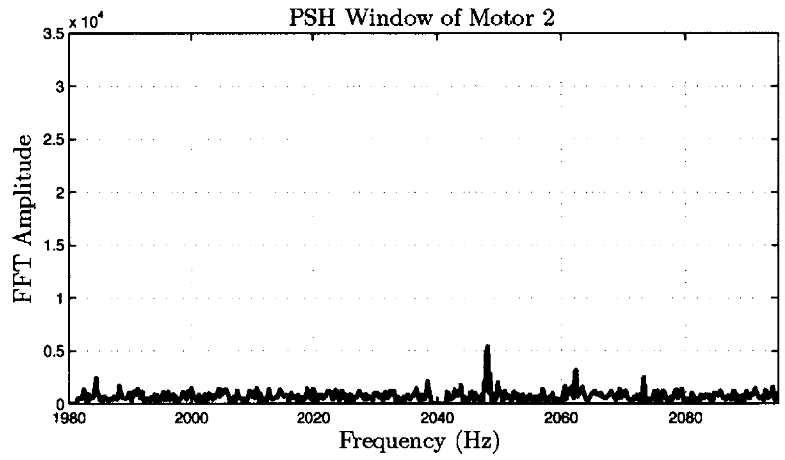


Fig. 3-13: The aggregate current spectrum in the PSH window of Motor 2 when only Motor 1 is running. There are no noticeable features which should make the detection of the PSH of Motor 2 difficult.

in this PSH window. Detecting the PSH of Motor 2 should be rather feasible in this scenario.

In this experiment, the parameters of the motors were chosen such that the PSH of each motor would appear in separate windows. One problem, which shows up in this experiment as Motor 2 turns on, is the addition of eccentricity harmonics [16].

Motor 2 is loaded with a fan which exacerbates these eccentricities. Figure 3-14 shows the window of interest for the PSH of Motor 1 when both motors are running.

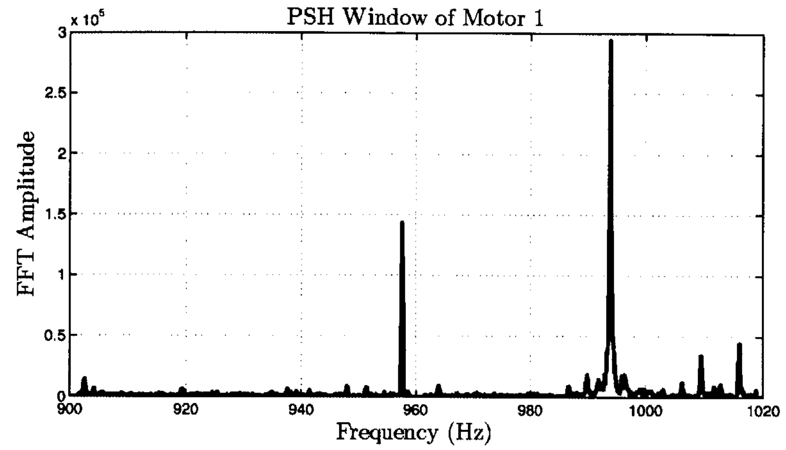


Fig. 3-14: The aggregate current spectrum in the PSH window of Motor 1 when both motors are running.

In this research, the motors examined had slot harmonics that are larger in amplitude than any of the eccentricity harmonics given by (3.1). Tracking these eccentricity harmonics may be possible by first estimating the speed of the eccentric motor. Once the slip is estimated, all possible eccentricity harmonics can be tabulated and tracked. Changes in the observed amplitudes of these eccentricity harmonics can be used to diagnose the health of the motor. Also, the above algorithm can use knowledge of eccentricity harmonics to make better speed predictions by properly including the effects of the eccentricity harmonics in the observed frequency content.

The current spectrum for the post-trigger PSH window for Motor 2 is shown in Fig. 3-15. When compared to the pre-trigger PSH window shown in Fig. 3-13, the content in the post-trigger window can be assumed to have come from a single sinusoid. Under this assumption, the optimized PSH for Motor 2 is estimated at 2046.28 Hz which corresponds to 3505.20 rpm.

3.5 Limitations to Proposed Method

Earlier, it was mentioned that Motor 2 was an intake ventilation fan that was paired with an identical motor used in the exhaust fan. In some cases, it may not be possible

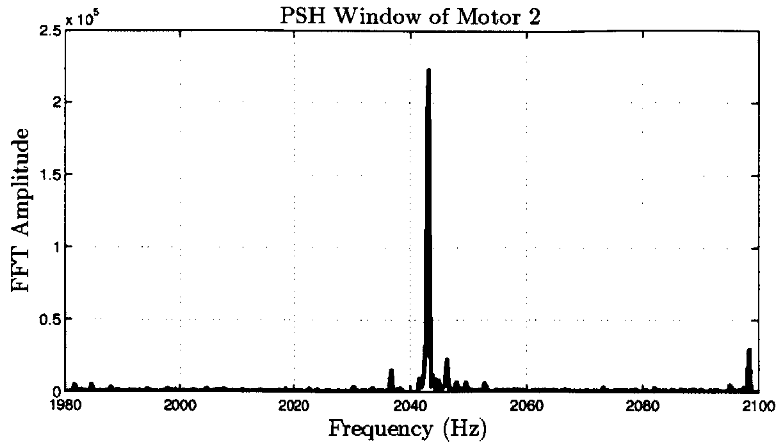


Fig. 3-15: The aggregate current spectrum in the PSH window of Motor 2 when both motors are running.

to replace motors such that their principal slot harmonics are in the same window. However to further motivate the need to take this into consideration, examine the following experiment.

Figure 3-16 shows the laboratory setup of the single housing used to enclose two ventilation fans. As constructed, the two ventilation fans would share the same airspace in that the air being filtered in by the intake fan is then pushed out by the exhaust fan. These two fans are mechanically coupled and their load conditions are different when they are turned on at the same time.

When each fan is on by itself, the location of the PSH is tracked by monitoring the proper PSH window. For each of these fans, the motors are identical and the PSH window is between 1980 Hz and 2100 Hz. The frequency content for 5 seconds of current from the intake fan when on by itself is shown in Fig. 3-17. The optimization routine estimates the principal slot harmonic to be 2042.100 Hz with a corresponding speed of 3497.824 rpm. The frequency content for 5 seconds of current from the exhaust fan when on by itself is shown in Fig. 3-18. Its PSH is estimated to be 2047.680 Hz with a corresponding speed of 3507.671 rpm.

When both motors are turned on, the loading conditions are different and the speed of each motor changes. The PSH window when both motors are on is shown in Fig. 3-19. Table 3.3 summarizes the differences in the location of the principal

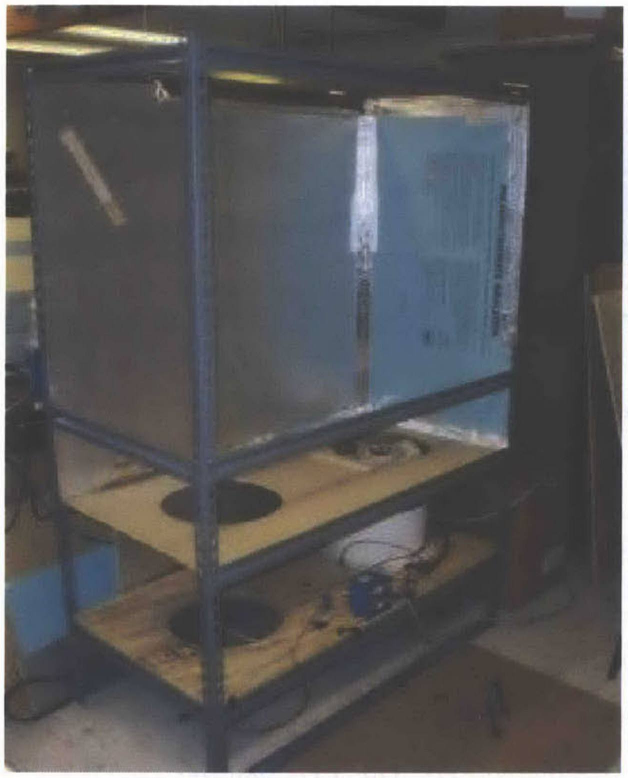


Fig. 3-16: Single housing used to enclose two ventilation fans.

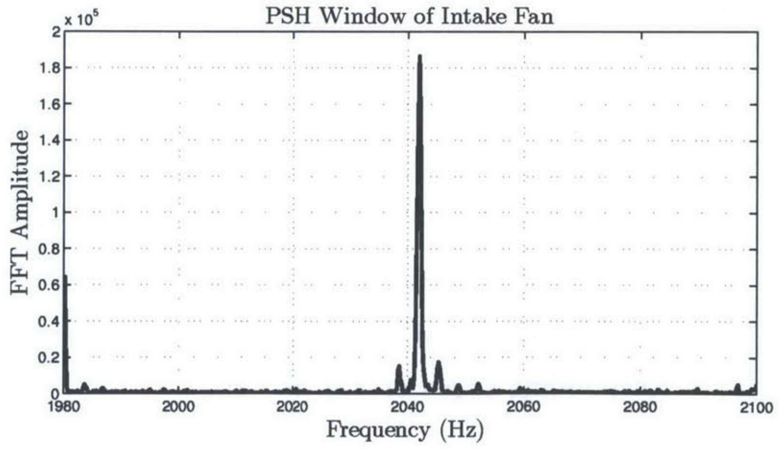


Fig. 3-17: Frequency content of the intake fan current. Its PSH is estimated to be 2042.100 Hz and the speed is then estimated to be 3497.824 rpm.

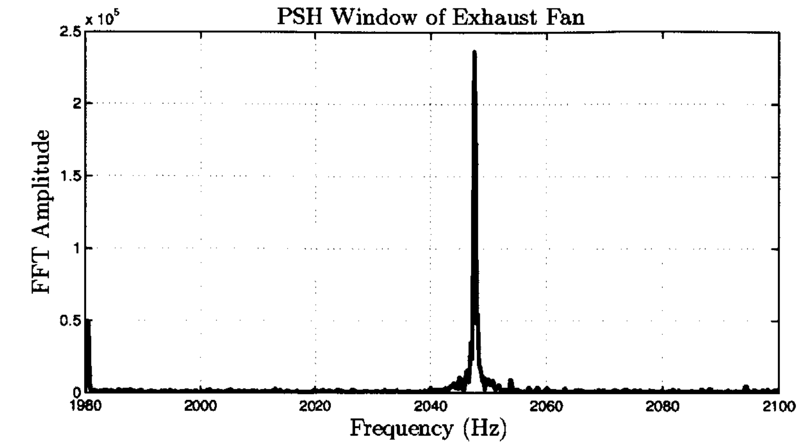


Fig. 3-18: Frequency content of the exhaust fan current. Its PSH is estimated to be 2047.680 Hz and the speed is then estimated to be 3507.671 rpm.

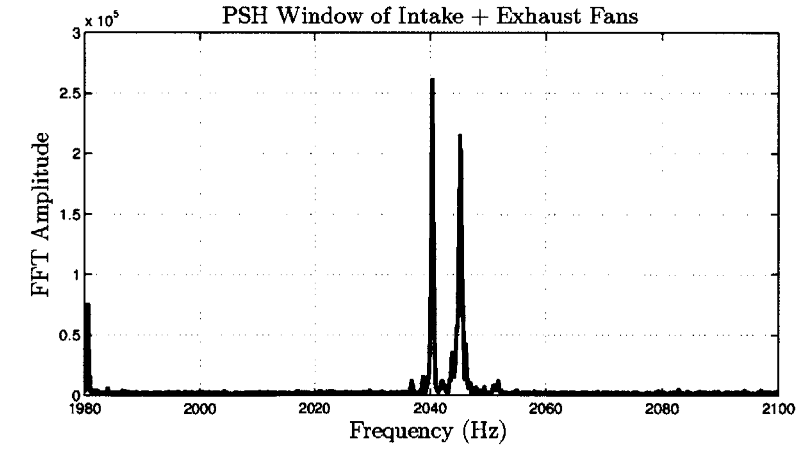


Fig. 3-19: Frequency content of the current when both the intake and exhaust fans are on. The PSHs of both fans have shifted as the loading conditioning have changed. Comparing these PSHs to the PSHs when each motor was turned on alone becomes challenging.

slot harmonics of the both fans when the loading conditions change over time.

Table 3.3: Differences in the location of the principal slot harmonics of the intake and exhaust fans when the loading conditions change over time.

	Intake		Exhaust	
	PSH	**RPM**	**PSH**	**RPM**
Intake Only	2042.100	3497.824	-	-
Exhaust Only	-	-	2047.680	3507.671
Intake + Exhaust	2040.400	3494.823	2045.200	3503.029

In the previous sections, the optimization routine was shown to work best when the motors are selected in such a way that the principal slot harmonics of each motors appears in different windows. Under these conditions, it can be assumed that the frequency content of the current in a post-trigger window comes from a single additional sinusoid. The routine can then find the frequency of this sinusoid that best fits the data to estimate the location of the PSH. It was shown in this section, that if the machines are mechanically coupled and the PSH appear in the same window, the routine breaks down. Any sinusoid in the pre-trigger window has shifted and a more sophisticated routine will need to be designed that keeps track of how mechanically coupled motors affect one another. If the machines are not mechanically coupled, the routine would work as long as any motors in the pre-trigger window have not changed slip significantly as the new load turns on.

3.6 Airflow Diagnostics Application

For an example of the utility of knowing motor speed in addition to power consumption for load diagnostics, consider ventilation systems in residential or commercial buildings. A number of surveys of airflow faults in buildings hint at the range and extent of these problems. One compendium of fault surveys [33], which examined 503 rooftop air-conditioning units in 181 buildings in five states in the Western U.S. from 2001-2004, found that the airflow was out of the specified range in approximately 42% of the units surveyed. A separate study [34] of 4,168 commercial air-conditioners in California reported that 44% of the surveyed units had airflow that was out of specifications. Studies of 29 new homes in Washington State [35] found that average duct leakage rates to the exterior ranged from 687 to 140 cubic feet per minute (CFM). Extrapolating from such fault surveys, one estimate for the total energy consumed by duct leakage is $5 billion/year [36].

A system that is able to monitor the state of airflow and detect faults in ventilation systems would fulfill a significant need in contemporary buildings, due to the prevalence of airflow faults. One widely used ventilation system employs air-side distribution systems for air-conditioning units typically called air handlers, air handling units, or AHUs. A picture and a visual representation of the AHU used in [37] are shown in Fig. 3-20.

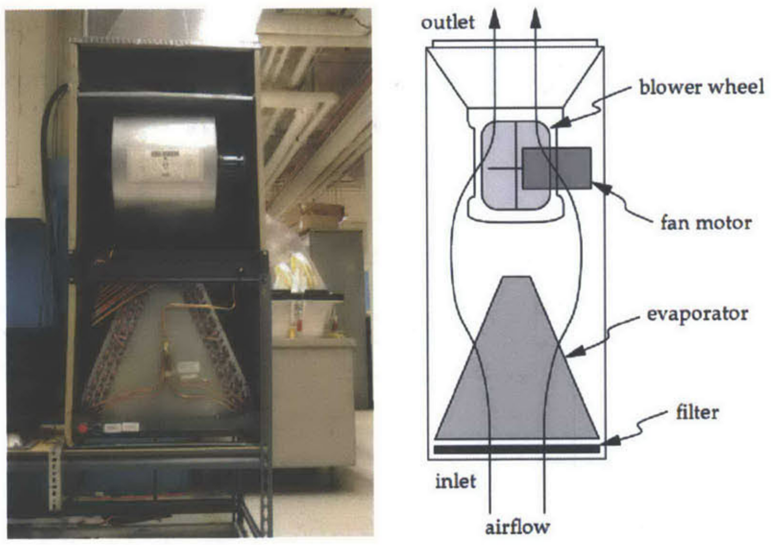


Fig. 3-20: Schematic diagram and picture of air handler unit [37].

A common fault in these systems occurs when the filter to the air handler, or the evaporator itself, is clogged, causing the airflow through the fan to be reduced. While the most notable effect of such a fault will be on the reduction in the air delivered to the building occupants, this fault can also potentially chill the volume of air flowing through the AHU further than is intended. A dramatically reduced flow rate could also affect the system health of the overall air-conditioning system, and of the compressor more specifically; if little air is traveling through the evaporator, the cooling load on the evaporator could be substantially reduced, causing the amount of refrigerant evaporated in the evaporator to be much smaller than required by the design specifications. This could potentially result in liquid refrigerant entering the compressor through the suction line, causing the ingestion of liquid refrigerant by the compressor, permanently damaging it. Also, the accumulation of material on the filter or the evaporator can also effect the health of building occupants. The accumulation of bacteria or mold on these surfaces can affect people breathing the air.

The architecture of the airflow estimation method in [37] is dependent upon the estimation of three related quantities: the mechanical torque applied to the fan τ_f , the speed of the fan blades ω_f, and the fan curve at the operating point of the fan. Since the fan curve is measured empirically by the manufacturer, it is necessary to develop a method to determine ω_f and τ_f from the motor electrical variables V_m and I_m. The block diagram in Fig. 3-21 shows the estimation scheme employed in [37].

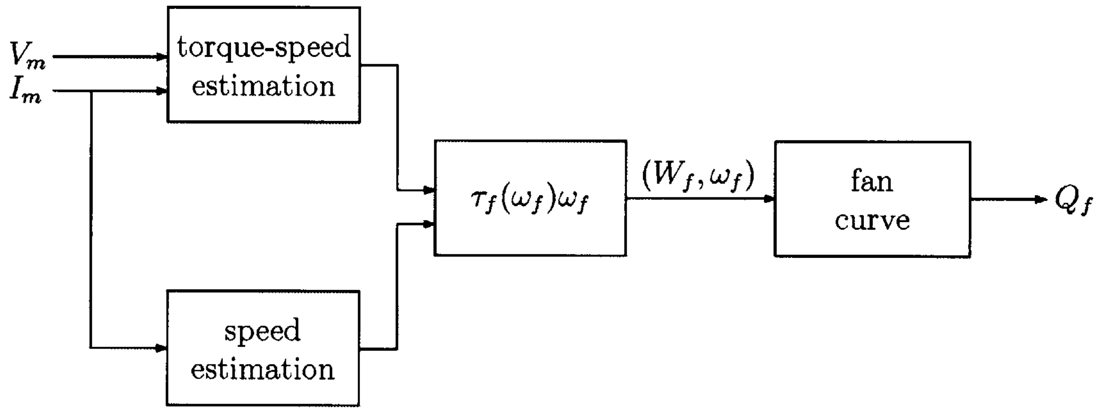


Fig. 3-21: Structure of the airflow estimation method [37].

To identify the airflow Q_f, the speed ω_f and the torque-speed curve $\tau_f(\omega_r)$ are first estimated independently from the electrical variables. The methods introduced in [37] describe a way that the speed ω_f of the fan blades can be nonintrusively estimated. The torque $\tau_f(\omega_r)$ at the motor's present operating point can then be identified by monitoring the voltage and current supplied to the fan. With these estimates of ω_f and τ_f, the operating mechanical shaft power W_f can be identified. The mechanical power and the operating speed are then used to identify the point on the fan curve that describes the fan's current state, thereby generating an estimate of the volumetric airflow Q_f through the fan. An estimate of Q_f (cfm) can be recorded many times after many starts of the fan. These estimates can be collected in a histogram and tracked and trended over time as shown in Fig. 3-22 [37] for diagnostic purposes.

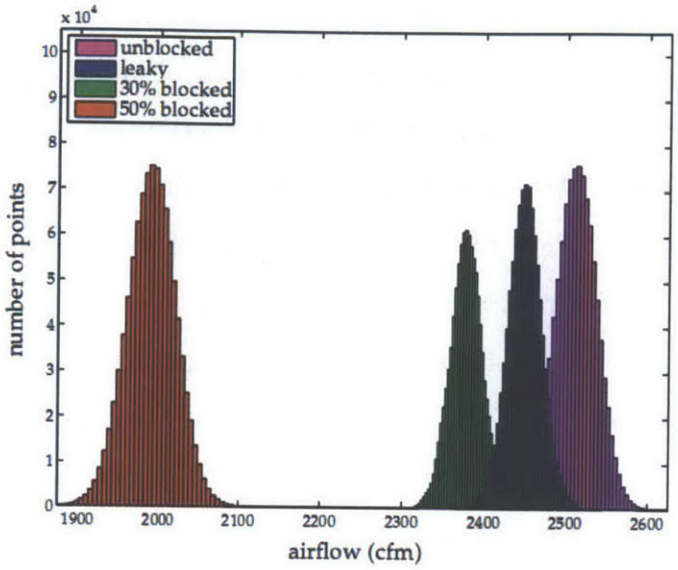


Fig. 3-22: Illustration of airflow detectability using torque-speed curves that are generated from minimization against the motor current and the torque-speed curve, as collected for each blockage condition [37].

Fig. 3-22 shows experimentally derived histograms for the airflow of the AHU when the intake filter is unblocked, 30% blocked, 50% blocked and leaky. As expected, the airflow estimates are indicative of the mechanical condition of the AHU.

3.7 Reciprocating Compressors Application

Another application for nonintrusive speed detection is described in [38] for nonintrusive fault detection in reciprocating compressors. The standard reciprocating compressor is a machine that relies on the electromagnetic domain of the induction motor, the mechanical motion of the crankshaft and reciprocating pistons, and the fluid flow regime of the gas dynamics governed by the compressor's valves. The three-phase induction motor drives the crank shaft of the reciprocating compressor, which causes motion of the pistons. The goal of the nonintrusive fault detection algorithm is to provide a signal of exceptional sensitivity to mechanical faults within the compressor from nonintrusive electrical measurements. The compressor voltage and current signals are used to “invert” the induction motor model equations to calculate the torque and speed of the motor. The motor shaft speed, along with the driving torque, is used to calculate the load torque on the crankshaft. The crankshaft load torque signal is sensitive to valve and other mechanical faults in the compressor. In [38], a longer custom shaft was built into the induction machine so that a custom mount could be added. An encoder with 2500 counts is attached on the mount to provide speed readings. This intrusive modification of the reciprocating compressor circumvents the goal of nonintrusive fault detection.

The utility of the nonintrusive speed detection algorithm is clearly evident in this application. To compare, the compressor has a steady tank at 70 psi. The algorithm in [38] uses a running average of 1 second with a 50% overlap to compute speed from the encoder. The encoder speed estimate is shown as the solid line in Fig. 3-23. The nonintrusive speed detection estimate is shown as the dotted line in Fig 3-23.

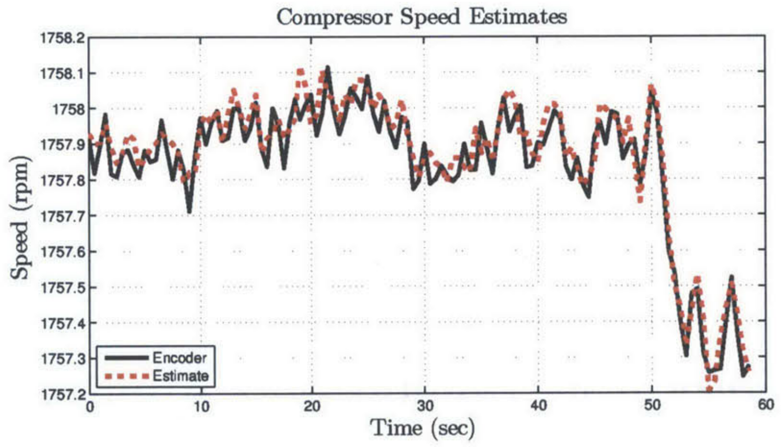


Fig. 3-23: Speed estimate comparison of a compressor with a steady tank at 70 psi. The solid line is the estimate from the custom mounted encoder and the dotted line is the nonintrusive speed detection estimate.

The data between the two methods agree well. The main difference is that the nonintrusive speed detection algorithm did not require any modification of the compressor.

3.8 NilmDB and Cottage Street School

NilmDB is a database system created by Jim Paris [39] that can manage the large quantity of data collected by a NILM device. The algorithms incorporated in NilmDB allow for quick processing of data on different time scales. Load identification can be made based up on startup transients which can be on the order of milliseconds whereas load duty cycles can be on the order of hours or days. NILM Manager is an online web interface created by John Donnal [40] that provides a graphical user interface for viewing the NilmDB data. A screenshot of the NILM Manager is shown in Fig. 3-24.

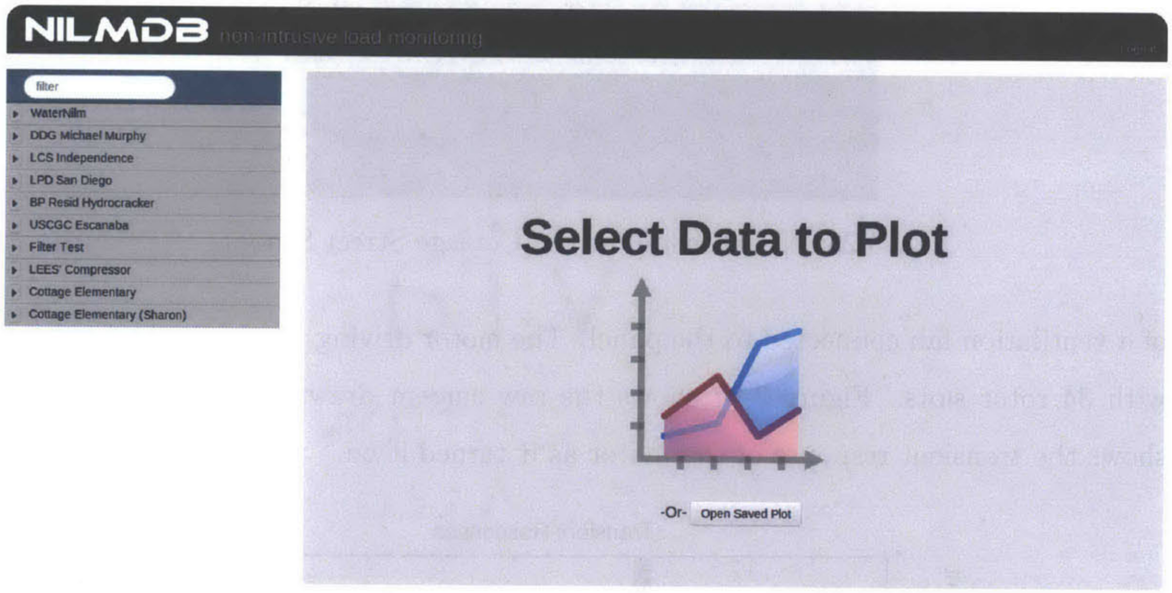


Fig. 3-24: NILM Manager startup screen [40].

The Laboratory for Electromagnetic and Electronic Systems at MIT has partnered with the Cottage Street Elementary School (Fig. 3-25) in Sharon, MA to install a NILM device in their boiler room as shown in Fig. 3-26. The NILM device is connected to the most critical electrical subpanel that serves as the hub for the kitchen, septic, heating, hot water, communications, and other key systems. The ultimate goal is to track consumption with the goal of translating data into cost savings for this facility.

A test was conducted to determine if the NILM installation can monitor the speed

Fig. 3-25: Cottage Street Elementary School in Sharon, MA

Fig. 3-26: NILM install at the Cottage Street School

of a ventilation fan connected to the panel. The motor driving the fan is single-phase with 34 rotor slots. Figure 3-27 shows the raw current drawn by the panel which shows the transient response of the motor as it turned it on.

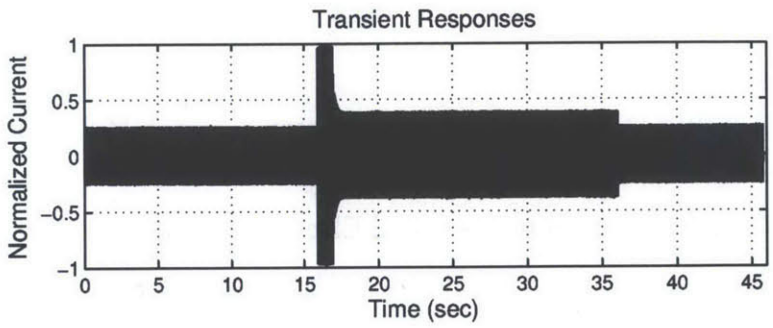


Fig. 3-27: Raw current with ventilation fan turn on transient

The PSH of the ventilation fan can be found by comparing the FFT window of the motor current before and after the motor has turned on. Figure 3-28 shows the FFT of the current during the pre-trigger window before the motor has turned off. Figure 3-29 shows the FFT of the current during the post-trigger window after the

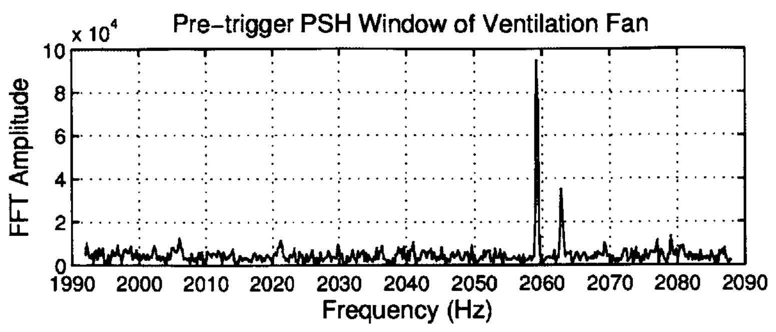


Fig. 3-28: Pre-trigger PSH window of ventilation fan

motor has turned on and reached steady state.

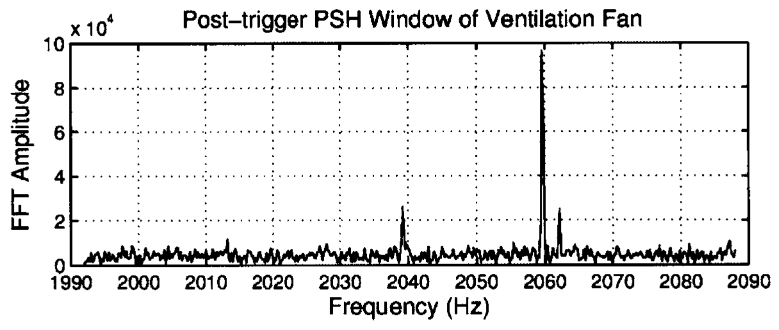


Fig. 3-29: Post-trigger PSH window of ventilation fan

The PSH of the motor is visible near 2040 Hz. The FFT optimization routine is conducted to estimate the underlying location of the harmonic. The results are tabulated in Table 3.4 for various lengths of time.

Table 3.4: PSH estimation for various lengths of times

Time Length	PSH	RPM
.5	2037.651	3489.972
1	2039.689	3493.568
5	2039.237	3492.771
10	2039.239	3492.774

3.9 Summary

This chapter illustrates how the known behavior of motor harmonics could be exploited with reasonable assumptions about operating conditions to estimate speed in a nonintrusive setting. Spectral envelope computations are used to characterize the operating schedule of loads as then turn on. Once recognized as "on", the current before and after the transient can be analyzed to estimate the speed of the new motor joining the collection of operating loads on the monitored service. A fine estimate of speed can be calculated by employing an optimization routine to find the optimal frequency of a sinusoid that closely matches the spectral content of the observed data. This optimization routine allows for smaller sampling windows to obtain desirable frequency resolution. This technique can be extended to estimate the speeds of multiple motors from an aggregate current signal. The efficacy of this technique was tested in various applications and even in the field.

Chapter 4

Vibration Monitoring Using Shaft Speed Oscillation Harmonics

Previous research has been done to study the effects of mechanical faults on the stator current spectrum [16–27, 41]. These faults, including broken rotors bars, damaged bearings, rotor eccentricity, bearings failures, and shaft speed oscillations, produce distinct harmonics in the current spectrum. One mechanical fault of interest is the shaft speed oscillation, which can be enhanced when an imperfectly balanced fan is attached to the motor shaft. The frequencies of interest [18] are predicted by

$$f_{sso} = f\left[k\left(\frac{1-s}{p}\right) \pm \nu\right]. \tag{4.1}$$

These harmonics from the shaft speed oscillations are present in the stator current, complicating the detection of the principal slot harmonics.

The ac utility line is a potentially distorted sinusoid containing only the fundamental frequency f and harmonic multiples of this frequency. Depending on the time of day, the amount of loading on the utility line can vary substantially and can cause large distortions on the utility line. Also, line impedances create voltage distortions at frequencies determined by other loads in the system. These distortions, like the mechanical faults, introduce extra harmonics on the current spectrum.

4.1 Current Method

According to [42], the United States Coast Guard uses vibration analysis to assess a particular electric machine's condition including bearing faults, misalignment, imbalance, electrical faults and several other potentially damaging faults. The current method of vibration monitoring requires mounting a sensor to the outer hub of the motor and taking readings from a hand-held vibration meter. There are some inherent flaws in this method. One concern with this method involves the position of the transducer that is placed on the outer hub. A preliminary study by Thomson et al. [43] showed that stator frame vibration changes as a function of the transducer position around the periphery of the frame. There is no guarantee that the transducer will be placed at the same location each time there is a maintenance check. The accuracy of these readings over time may not be as reliable and trustworthy as needed. Another flaw in the previous method includes the high cost of performing such quarterly vibration monitoring checks by bringing in outside experts to perform the vibration analysis. Finally, these vibration checks are only conducted four times a year which may not frequent enough to catch any serious, potentially fatal, damage in a motor.

Electrical-based vibration monitoring is a proposed solution that overcomes some of the shortcomings in the previous method. By monitoring the current drawn by the machine, the data recorded can be trusted and will be consistent throughout the life of the motor. Whereas the results from the previous method's data can be altered by simply moving the transducer, electrical-based readings are more reliable. Secondly, electrical-based monitoring can be automated so that a computer can track the vibration of the motor more periodically instead of the four times a year as was the case for the previous method. Implementing an electrical-based vibration monitoring would not require contracting outside experts as the analysis can be done "in-house" by a computer. This chapter will propose a method that can track vibration on a motor using only electrical signals.

4.2 Shaft Speed Oscillation Harmonics

To predict the location of shaft speed oscillation harmonics in the current spectrum, a simplified steady-state equivalent circuit is used to model the motor. The following derivation is adapted from [18]. The equivalent circuit only includes a slip dependent rotor resistance as shown in Fig 4-1.

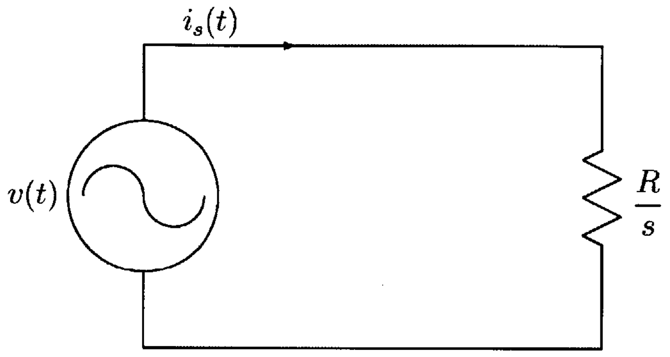


Fig. 4-1: Simplified steady-state equivalent circuit model of motor containing only a voltage source and a slip-dependent rotor resistance. The location of the shaft speed oscillation harmonics can be determined by fully expressing the stator current $i_s(t)$.

The shaft speed of the motor is assumed to be comprised of an average speed and a sinusoidally varying component given by (4.2). The line voltage is modeled as a pure sine wave of the line frequency given by (4.3). The relative phases between the sinusoids are ignored in this derivation. The shaft speed $n(t)$ and line voltage $v(t)$ are:

$$n(t) = N_{avg} + \sum_k N_k \sin(2\pi k f_c t) \tag{4.2}$$

$$v(t) = V \sin(2\pi f t), \tag{4.3}$$

where N_{avg} is the unperturbed synchronous speed (rpm), N_k are the Fourier components of the speed fluctuation, f_c is fundamental frequency of the speed fluctuation, V is the voltage amplitude, and f is voltage line frequency. The line current $i_s(t)$ is given by the following expression:

$$i_s(t) = v(t)/r(s) \tag{4.4}$$

where $r(s)$ is the slip-dependent rotor resistance and is expressed as R/s for some nominal resistance R and slip s. The slip s of the motor is defined as

$$s = \frac{N_{sync} - n(t)}{N_{sync}}, \tag{4.5}$$

where N_{sync} is the synchronous speed of the motor.

Combining (4.4) and (4.5), the line current can be fully expressed as

$$i_s(t) = \frac{V}{RN_{sync}}\left(N_{sync} - N_{avg} - \sum_k N_k \sin(2\pi k f_c t)\right)\sin(2\pi f t) \tag{4.6}$$

Letting $A = \frac{V(N_{sync}-N_{avg})}{RN_{sync}}$ and $A_k = \frac{VN_k}{RN_{sync}}$,

$$i_s(t) = A\sin(2\pi f t) - \sum_k A_k \sin(2\pi k f_c t)\sin(2\pi f t) \tag{4.7}$$

The expression for $i_s(t)$ can be expressed by taking the fundamental frequency of the speed fluctuation, f_c, to be equal to the shaft speed. The underlying assumption is that the shaft speed oscillation is periodic with the position of the shaft. Substituting $f_c = f\frac{1-s}{p}$ where p is the number of pole pairs and using the product of sines trigonometric identity,

$$\begin{aligned} i_s(t) &= A\sin(2\pi f t) + \sum_k \frac{A_k}{2}\Big(\cos\big(2\pi\left(kf_c + f\right)t\big) - \cos\big(2\pi\left(kf_c - f\right)t\big)\Big) \\ &= A\sin(2\pi f t) + \sum_k \frac{A_k}{2}\left[\cos\left(2\pi f\left(k\frac{1-s}{p} + 1\right)t\right)\right. \\ &\qquad \left. - \cos\left(2\pi f\left(k\frac{1-s}{p} - 1\right)t\right)\right] \end{aligned} \tag{4.8}$$

The stator current $i_s(t)$ has a line frequency component which is present from the non-varying speed component of the rotor. The other two terms are symmetric sidebands which arise from the shaft speed variation. Therefore, the shaft speed oscillation harmonic can expressed as

$$f_{sso} = f\left[k\left(\frac{1-s}{p}\right) \pm 1\right], \tag{4.9}$$

where s is per unit slip, $k = 0, 1, 2...$ and p is the number of pole pairs.

4.3 Single Motor Environment

To verify the expression in (4.9), the following experiment was conducted. A 3 phase, $p = 3$ pole pair induction machine is loaded with an evaporator fan in a heating, ventilation and air conditioning (HVAC) unit. Additional weight was placed on the fan to create an imbalance leading to more vibration by adding paper clips to the fan blade. A current transducer was placed around one of the phases and thirty seconds of data were recorded for each incremental weight added.

A fast Fourier transform (FFT) was taken on the data to compute the spectral content. For this motor, with $p = 3$ and for a low-slip assumption ($s < .05$) and a $f = 60$ Hz supply line frequency, the principal shaft speed oscillation ($k = 1$) is between 79 Hz and 80 Hz. Figure 4-2 displays the frequency content of the motor for different amounts of paper clips.

Table 4.1: This table lists the normalized energy content for different amount of weight added. This shaft speed oscillation harmonic energy correlates well with vibration reading from a vibration meter.

Number of clips	Vibration Reading (g)	Normalized Energy
0	0.03	3.1570
4	0.09	6.5454
6	0.10	8.2172
10	0.19	15.9049

Table 4.1 lists the normalized energy content in the frequency window shown in

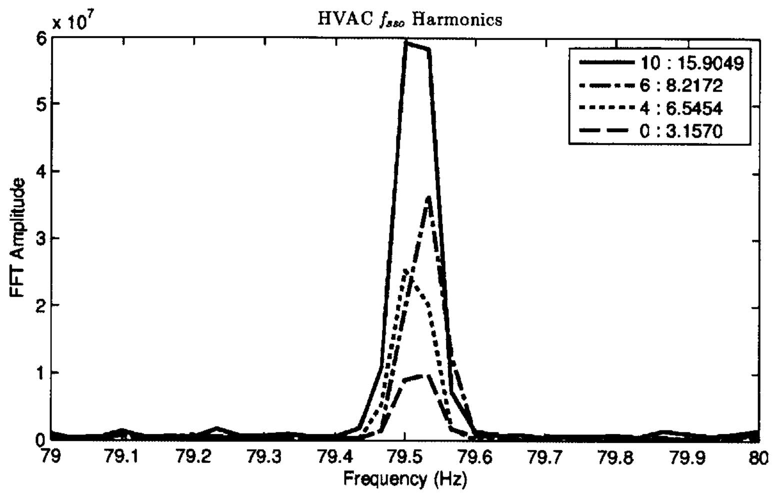


Fig. 4-2: Energy content for an HVAC evaporator fan for different amounts of weights. More weight leads to more vibration and more energy content in the shaft speed oscillation harmonic.

Fig. 4-2. The second column displays the reading from an Extech 407860 Heavy Duty Vibration Meter in units of g. The results show that the energy from the shaft speed oscillation harmonic correlates well with radial vibration.

A second experiment was conducted on a pair of single phase ventilation fans donated by the Industrial Electric Shop (IES) on the 1st District Coast Guard base in Boston, Massachusetts. These machines were previously used for intake and exhaust ventilation in the main engine room aboard one of the Coast Guard's 49 ft buoy boats. For these motors, with $p = 1$ and for low-slip assumption ($s < .05$) and a $f = 60$ Hz supply frequency, the principal ($k = 1$) shaft speed oscillation harmonics are between 117 Hz and 120 Hz. To avoid the 120 Hz supply harmonic, the window of interest only monitored the frequencies between 117 and 119 Hz. Each motor was turned on independently and recorded for 30 seconds.

Figure 4-3 graphs the frequency content of the intake fan for different amounts of weight. Figure 4-4 graphs the content of the exhaust fan. Data were taken under **normal** (black trace) conditions, with a **little screw** (green) and a **big screw** (blue) attached to the hub of the fan blade.

Table 4.2 lists the normalized energy content in the frequency window for the ventilation fans. The vibration readings were recorded from an Extech 407860 Heavy

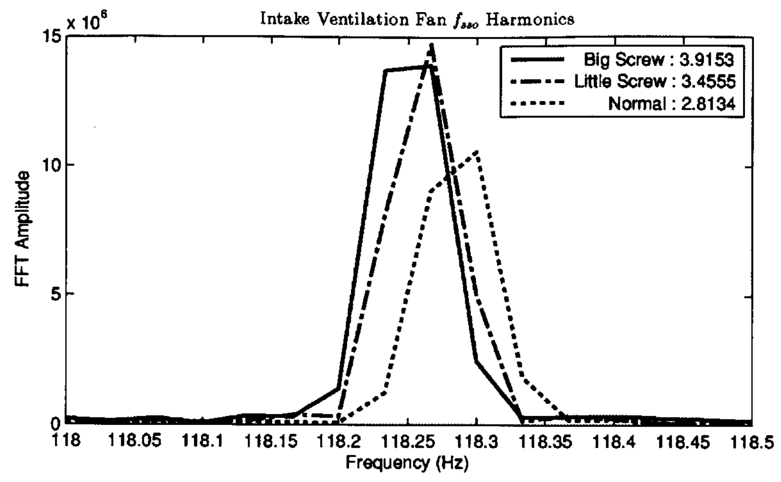


Fig. 4-3: Shaft speed oscillation harmonic energy content for intake ventilation fan. More eccentric weight correlates with more energy content from the shaft speed oscillation harmonic.

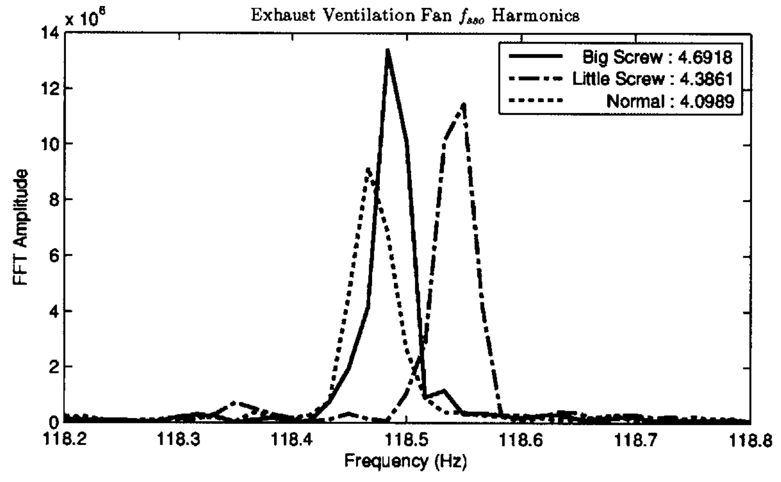


Fig. 4-4: The shaft speed oscillation harmonic energy content for the exhaust ventilation fan is shown.

Table 4.2: Vibration reading and energy content of intake and exhaust ventilation fans under normal conditions, with a little screw and a big screw. More weight correlates with more vibration and more energy from the shaft speed oscillation harmonic.

	Intake		Exhaust	
Eccentric Conditions	Vibration Reading (g)	Normalized Energy	Vibration Reading (g)	Normalized Energy
Normal	0.37	2.8134	0.78	4.0989
Little Screw	0.47	3.4555	1.09	4.3861
Big Screw	0.59	3.9153	1.15	4.6918

Duty Vibration Meter in units of g. The results show that the energy from the shaft speed oscillation harmonic correlates well with radial vibration.

4.4 Multiple Motor Environment

The previous section demonstrated the effectiveness of using the energy of the shaft speed oscillation harmonic as a measure of radial vibration when only one motor is running on the utility line. This section will extend the algorithm in a multiple motor environment.

A nonintrusive load monitor (NILM) can be used to determine the operating schedule of major electrical loads from measurements taken from a building's electric utility [7, 8]. Dynamic changes in the power and harmonic consumption of a load, e.g., during turn-on or turn-off transients, can serve as a fingerprint for identifying load operation [2, 9] as discussed in §2.3. For example, an observed turn-on transient or exemplar from a training observation produced by one of a collection of loads can be used to identify the load in an aggregate current measurement. An analogous procedure can be performed using turn-off transients. All that is needed, in principle, to determine the operating schedule of a collection of loads is to record the aggregate current drawn by those loads and then match each observed transient to the turn-on or turn-off fingerprint of a particular load in the collection.

The ability of the NILM to identify loads based on their turn-on transients was used to detect the speed of a motor in a multiple motor environment using principal slot harmonics [44]. By keeping track of the current and voltage before (pre-trigger) and after (post-trigger) a turn-on transient, the algorithm can estimate the speed of the new load. A similar method is introduced here to estimate the energy content of a shaft speed oscillation harmonic in a multiple motor environment.

The speed estimation method described in [44] had to calculate the location of the principal slot harmonic. For vibration analysis, the method must calculate the amplitude, or the energy, of the shaft speed oscillation harmonic. This difference leads to a slight modification in how the pre-trigger and post-trigger current spectral information are handled.

Section 4.4.1 will discuss how the method will estimate the energy of the shaft speed oscillation harmonic when there is no harmonic from a previous motor in the frequency window of interest. This could mean that there was no motor running prior to a trigger event or it could mean that the speed variation harmonic(s) of previous motor(s) show up in a different window than does the harmonic of the motor causing the trigger event. Section 4.4.2 will discuss how to estimate the energy of the harmonic of a motor in a window when there is another motor with the same number of pole pairs currently running.

4.4.1 No Previous Motor

Consider the single-phase, single pole-pair intake ventilation fan and the three-phase three pole-pair HVAC evaporator fan described earlier. When the intake ventilation fan is running by itself, there is a speed variation harmonic near 118 Hz harmonic described by (4.9) as shown in Fig. 4-5. Furthermore, there is no harmonic content near 79 Hz as shown in Fig. 4-6.

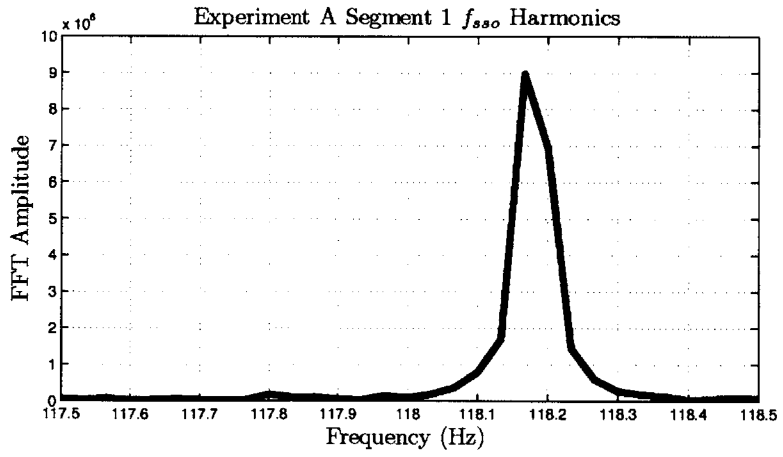


Fig. 4-5: Harmonic content of a single phase, $p = 1$ ventilation fan near 118 Hz is shown.

When the HVAC motor turns on, the harmonic content near 118 Hz from the intake ventilation fan is still present in Fig. 4-7 but now there is a spike in the 79 Hz window in Fig. 4-8. The harmonic content near 79 Hz can be thought of as the

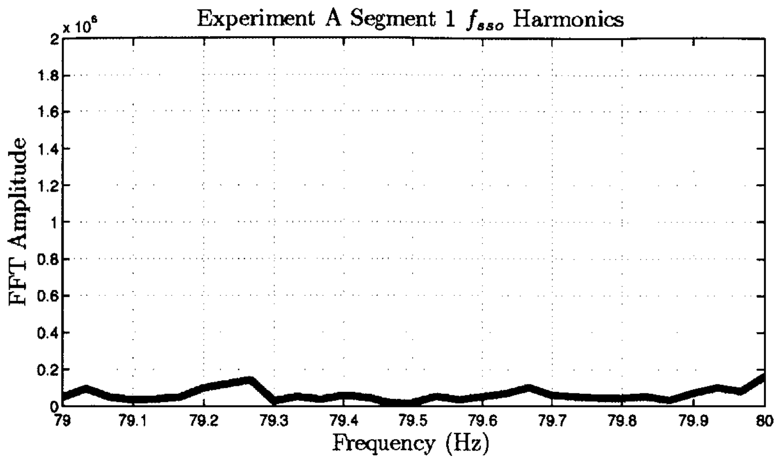


Fig. 4-6: No harmonic content near 79 Hz.

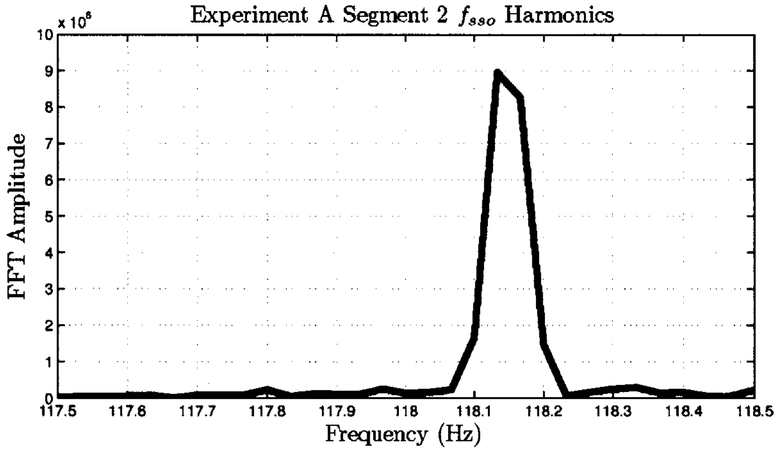


Fig. 4-7: Harmonic content of a single phase, $p = 1$ ventilation fan and a three phase, $p = 3$ HVAC motor. The $p = 1$ ventilation fan has content near 118 Hz.

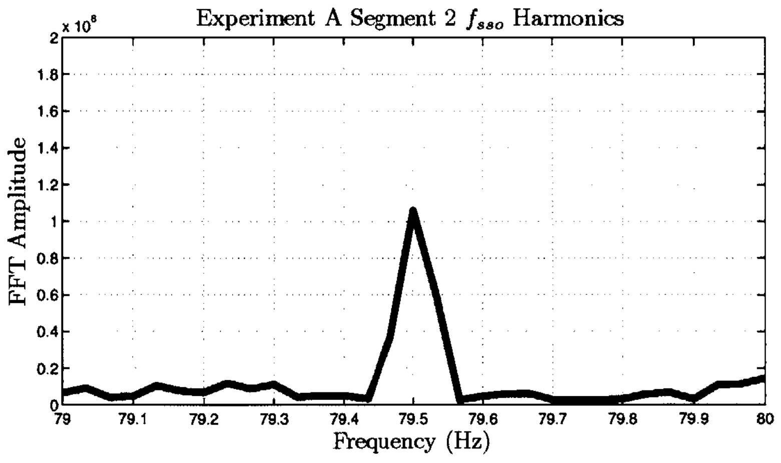


Fig. 4-8: The $p = 3$ HVAC fan has its harmonic near 79 Hz.

sum of the content from the intake ventilation fan and the content from the HVAC motor. Figure 4-9 shows the harmonic content near 118 Hz while Fig. 4-10 shows the content near 79 Hz when only the HVAC motor was on.

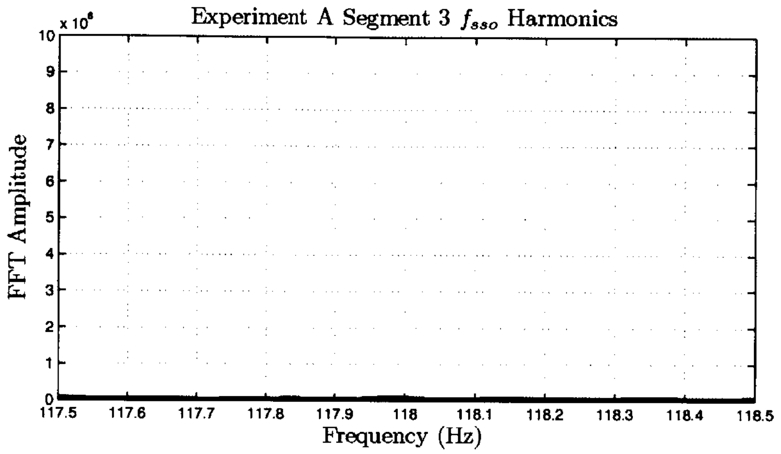


Fig. 4-9: The three phase, $p = 3$ HVAC fan has no harmonic content near 118 Hz as shown.

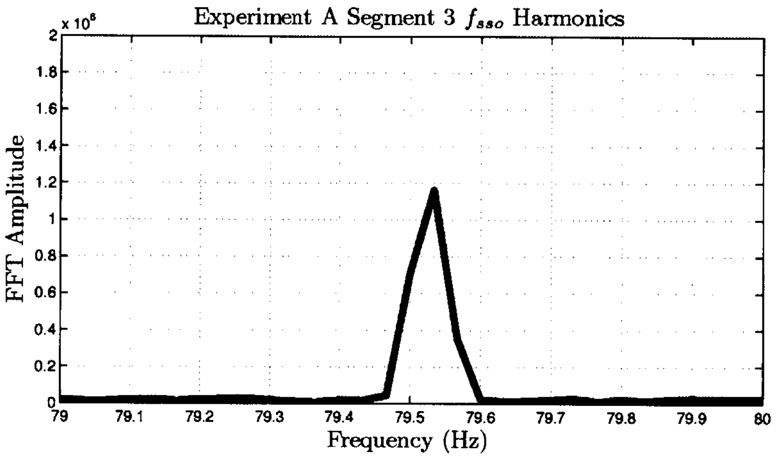


Fig. 4-10: No content near 79 Hz.

4.4.2 Previous Motor

In the case in which there are two speed oscillation harmonics in the same window, the phase between the two harmonics is now a concern since the method wishes to estimate the amplitude of each harmonic. Consider the following example of two fundamental sinusoids with a relative phase angle of ϕ with frequencies 117.5 and

117.6 Hz, respectively. Let $Y(\phi, t) = \sin(2\pi \cdot 117.5 \cdot t) + \sin(2\pi \cdot 117.6 \cdot t - \phi)$ be the sum of the two sine waves. Depending on the value of ϕ, the amount of spectral energy contained in a window centered around 117.5 Hz varies. Table 4.3 shows the amount of a spectral content in a window between 117 and 118 Hz for several values of ϕ using a 10 second long data record.

Table 4.3: Energy content of 2 sinusoids with various phase angle difference. The harmonic content energy is a function of this phase difference.

ϕ	Energy ($\cdot 10^4$)
0°	4.3200
15°	4.4269
30°	4.7176
45°	5.1274
60°	5.5950
75°	6.0749
90°	6.5356
105°	6.9550
120°	7.3170
135°	7.6097
150°	7.8247
165°	7.9559
180°	8.0000

As evident in Table 4.3, the energy contained in the window varies depending on the phase difference between the two sinusoids. This fact could lead to an incorrect estimation if the window is not chosen at the appropriate time. Unfortunately, there is no simple way of knowing beforehand when the two shaft speed oscillation harmonics will interfere constructively. To circumvent this problem, tracking the energy over a long enough period with a sliding window will guarantee that eventually these two harmonics will constructively interfere. The following example illustrates this point.

Consider the following scenario involving the intake and exhaust ventilation fans. Assume that the former is running at a slip of s_I with a corresponding shaft speed oscillation harmonic located at f_{ssoI} given by (4.9). This harmonic can be modeled as $y_I(t) = \sin(2\pi f_{ssoI} t)$. Now assume that the exhaust ventilation fan is running at a slip of s_E with a harmonic located at f_{ssoE}. The exhaust speed oscillation harmonic is modeled as $y_E(t) = \sin(2\pi f_{ssoE} t - \phi)$ where ϕ is the relative phase angle between

$y_I(t)$ and $y_E(t)$.

The algorithm essentially takes advantage of the beat frequency between $y_I(t)$ and $y_E(t)$. If the spectral energy is tracked long enough, the shaft speed oscillation harmonics will be in phase and the energy of each harmonic can be estimated more reliably. As an illustration, assume the intake fan is running at $s_I = 0.0417$ with $f_{ssoI} = 117.5$ Hz. Also, assume the exhaust fan is running at $s_E = 0.04$ with $f_{ssoE} = 117.6$ Hz with a phase angle $\phi = \pi/7$. With a sampling rate of 8000 Hz, a 30 second data record of $y_I(t) + y_E(t)$ is shown in Fig. 4-11. The spectral energy of the shaft speed oscillation harmonics is calculated over a 10 second window. A sliding window with an offset of $T_{offset} = 0.25$ seconds is moved along the entire 30 second window and the spectral energy is calculated in each window.

Figure 4-12 shows the result of the sliding window spectral energy calculation. It is evident that spectral energy is periodic based on the beat frequency between the harmonics. The two peaks in the figure coincide with the alignment of the phases of the two shaft speed oscillation harmonics.

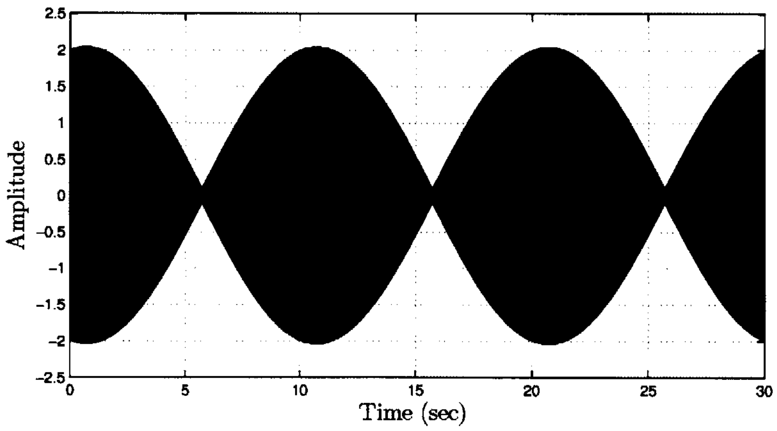


Fig. 4-11: A plot of two sinusoids with frequencies 117.5 and 117.6 Hz with a phase difference of $\phi = \pi/7$ is shown.

The beat period can be calculated by the following expression

$$T_{beat} = \frac{1}{f_{beat}} = \frac{1}{f\,|s_I - s_E|}, \tag{4.10}$$

where f is the frequency of the supply line and s_I and s_E are the slips of the intake

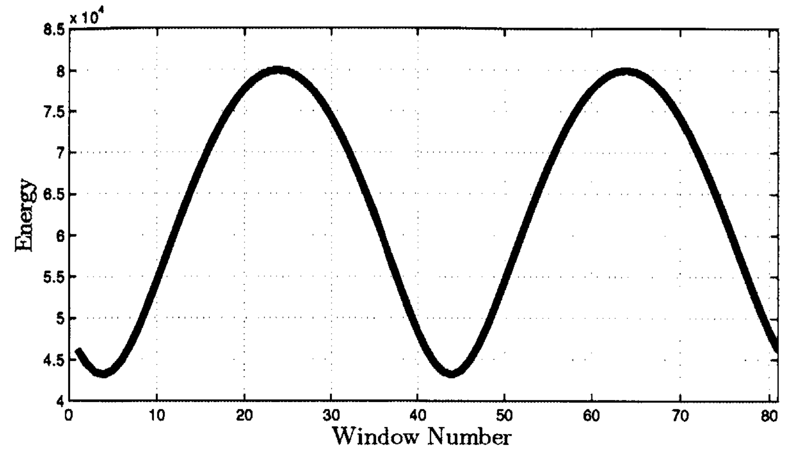


Fig. 4-12: The sliding window algorithm shows the beat period of the energy content.

and exhaust ventilation fans, respectively.

In the previous example with $f = 60$ Hz, $s_I = 0.0417$ and $s_E = 0.04$, the theoretical beat period is $T_{beat-theo} = 9.8$ seconds. Careful examination of Fig. 4-12 shows that the local maxima occur during the 24th and 64th windows, or every $W_{beat} = 40$ windows. With the window offset period of $T_{offset} = 0.25$ seconds, the experimental beat period is $T_{beat-exp} = W_{beat} \cdot T_{offset} = 10$ seconds. If the sampling frequency was faster or if the window offset period was shorter, the error between the theoretical and experimental beat periods would be smaller.

If the slip of the exhaust fan were to decrease to $s_E = 0.0383$, the beat frequency increases and the beat period should decrease. Figure 4-13 shows the results in which the window beat count has decreased to $W_{beat} = 20$ windows which results in an experimental beat period of $T_{beat-exp} = W_{beat} \cdot T_{offset} = 5$ seconds. The experimental result agrees with the theoretical calculation of $T_{beat-theo} = 4.902$ seconds.

In summary, to compute the energy of a shaft speed harmonic contributed by a motor when there is another motor running with the same number of poles requires an additional amount of work. The first step is to calculate the slip of each motor. Reference [44] suggests one possible method of estimating slip for two motors in a multiple motor environment. Once the slips have been estimated, the theoretical beat T_{beat} period is computed using (4.10). This algorithm then requires observing the aggregate current stream from a sliding window for at least T_{beat} seconds to observe

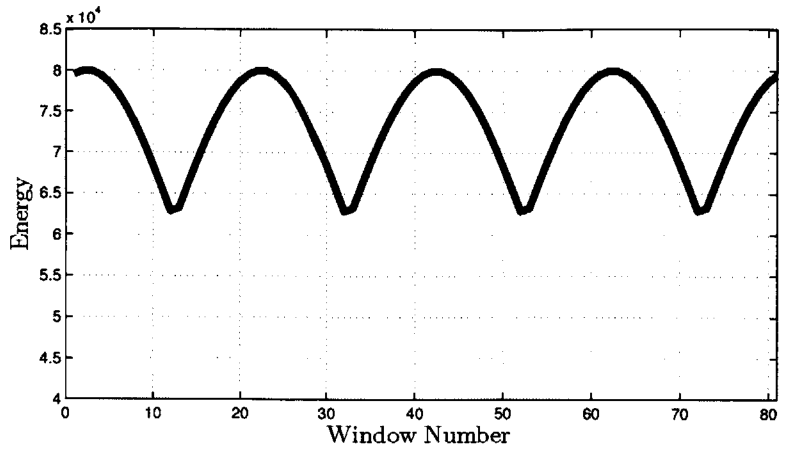


Fig. 4-13: Sliding window algorithm of 2 sinusoids with slips of .0412 and .0383. With a larger absolute difference in slips between the 2 motors, the beat period has decreased.

at least one peak in the beat period. The maximum energy observed over this time is used as the estimate of sum of the energy of the two shaft speed oscillation harmonics from the two motors.

This algorithm breaks down if both motors are running at the exact slip. In this instance, the energy observed is only a function of ϕ_{start} which is the phase angle between the two motors the instance the second motor turns on.

4.4.3 Experiment

The theory described in the previous section is applied in the experiments described below. Both the intake and exhaust ventilation fans were used. In the following two experiments, the exhaust fan has no additional weights attached to the fan hub. The intake fan has no additional weight but in the first experiment, an air filter is used to partially block the air flow. In the second experiment, a piece of cardboard filter is used to fully block the air flow to change the slip of the intake fan.

Figures 4-14 - 4-16 show the shaft speed oscillation harmonics appearing in the aggregate current. The spectral data shown are taken with a sampling time of 30 seconds and sampling frequency of $f_s = 8000$ Hz. When the intake fan is running by itself, its harmonic is shown in Fig. 4-14. The exhaust fan is later turned on and both harmonics are shown in Fig. 4-15. Figure 4-16 contains the shaft speed oscillation when the exhaust fan is on by itself.

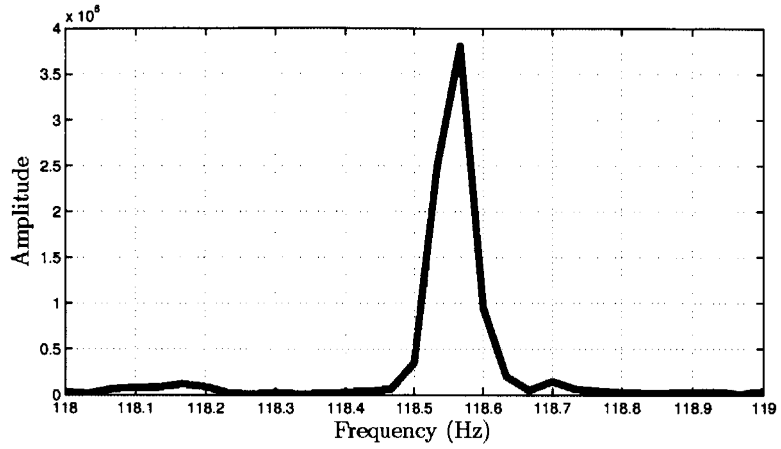


Fig. 4-14: When the intake ventilation fan runs by itself, its shaft speed oscillation harmonic is shown.

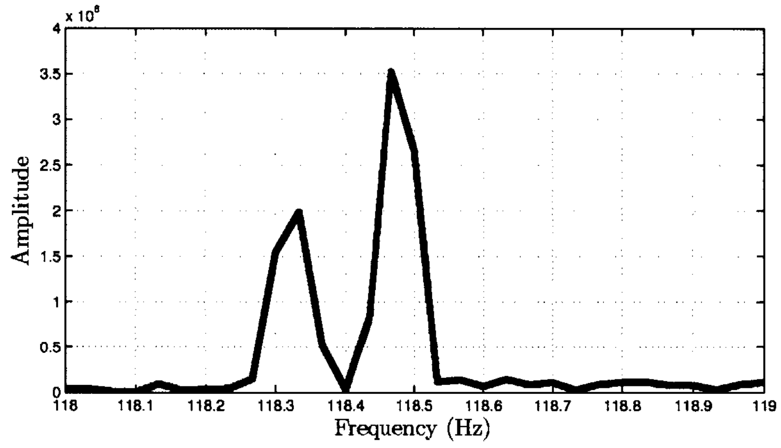


Fig. 4-15: The exhaust ventilation fan is then turned on and both oscillation harmonics are shown.

Analysis of the principal slot harmonics in Fig. 4-17 can be used to estimate the speed of any motor [11–15, 44]. When both motors are running, the intake ventilation fan is running at a slip of $s_I = .0249$ and the exhaust ventilation fan is running at a slip of $s_E = .0276$.

To get an accurate estimate of the energy of the shaft speed oscillation harmonic from the exhaust fan, the sliding window algorithm mentioned before will have to be employed. Equation (4.10) estimates that the beat period of the oscillation harmonics

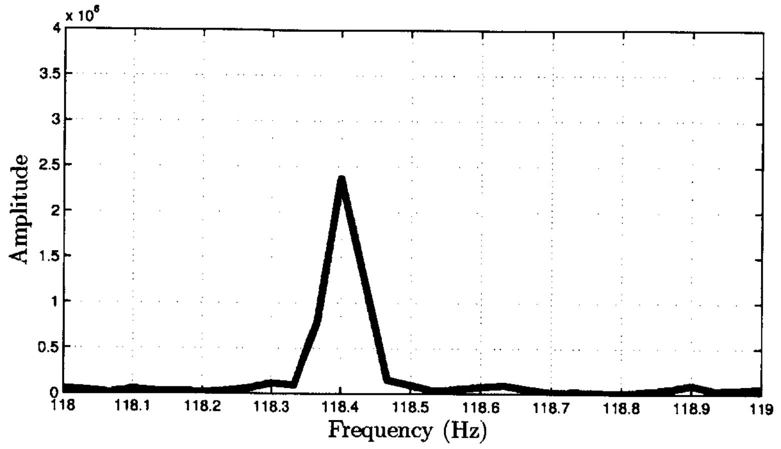


Fig. 4-16: The intake ventilation fan is then turned off and the oscillation harmonic of the exhaust fan when running by itself is shown.

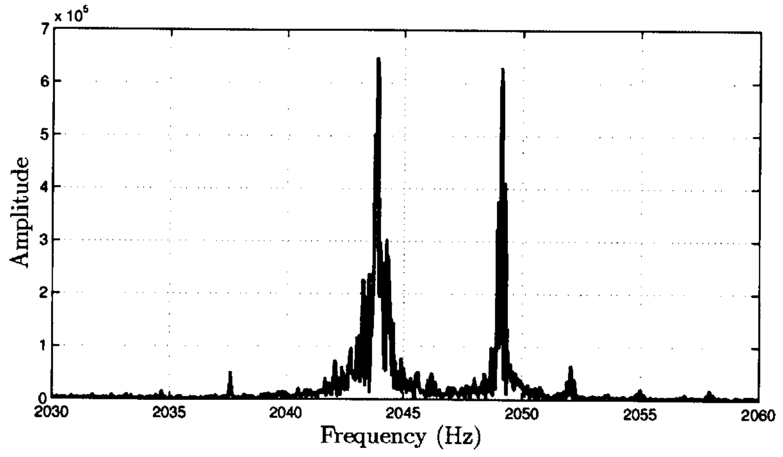


Fig. 4-17: The principal slot harmonics of the intake and exhaust ventilation fans with an air filter placed over the intake fan. These harmonics can be used to estimate the speeds of each motor.

is $T_{beat-theo} = 6.1728$ seconds. The result from the sliding window algorithm is shown in Fig. 4-18 with a data length of 10 seconds and an offset length of $T_{offset} = 0.25$ seconds.

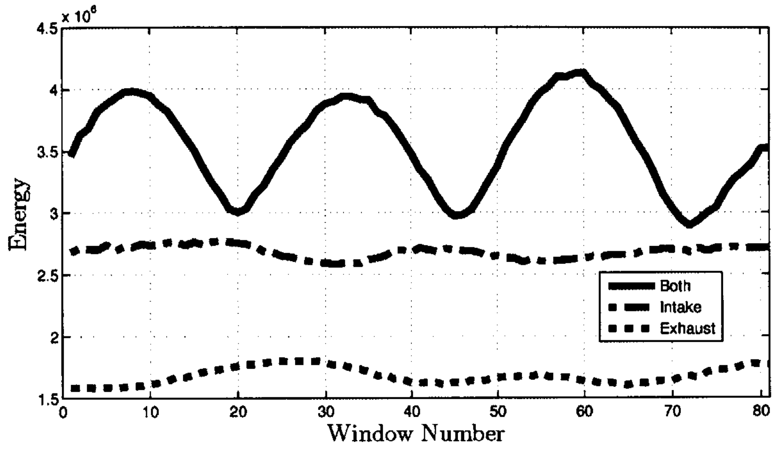


Fig. 4-18: The bottom two traces show the sliding window algorithm results when the intake and exhaust ventilation fans are running by themselves. The top trace shows the sliding window when both fans are on. The harmonic energy content is periodic and the period can be determined from calculating the slips of each fan.

The beat periods in counts of the sliding window between the three peaks are 24 and 28. The experimental beat period in seconds is $T_{beat-exp} = W_{beat} \cdot T_{offset}$. Therefore, the beat period is between 6 and 7 seconds which agrees with the theoretical calculation of 6.1728 seconds. Section 4.4.4 will provide a discussion on the numerical results.

The second experiment is similar to the first but a piece of the cardboard is used to block the air of the intake fan thereby altering the slip. The shaft speed oscillation harmonics are shown when the intake motor is on by itself (Fig. 4-19), when both motors are on (Fig. 4-20) and when the exhaust motor is on by itself (Fig. 4-21).

Analysis of the principal slot harmonics in Fig. 4-22 shows that when both motors are running, the intake ventilation fan is running at a slip of $s_I = .0253$ and the exhaust ventilation fan is running at a slip of $s_E = .0271$. The theoretical beat period of the oscillation harmonics is $T_{beat-theo} = 9.2593$ seconds.

The results from the sliding window algorithm is shown in Fig. 4-23. The beat period in window counts between the two peaks is 37 which results in an experimental

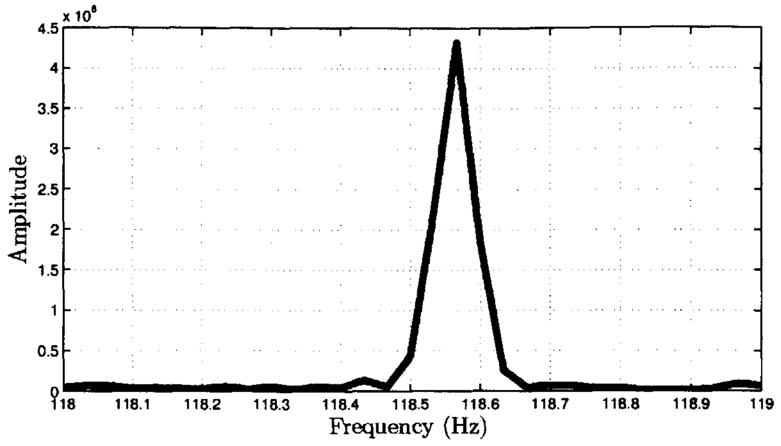


Fig. 4-19: When the intake ventilation fan runs by itself with a higher slip, its shaft speed oscillation harmonic is shown.

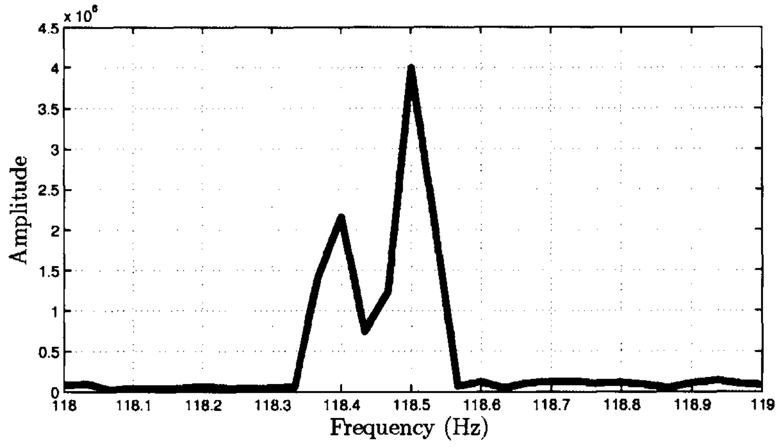


Fig. 4-20: The exhaust ventilation fan is then turned on and both oscillation harmonics are shown.

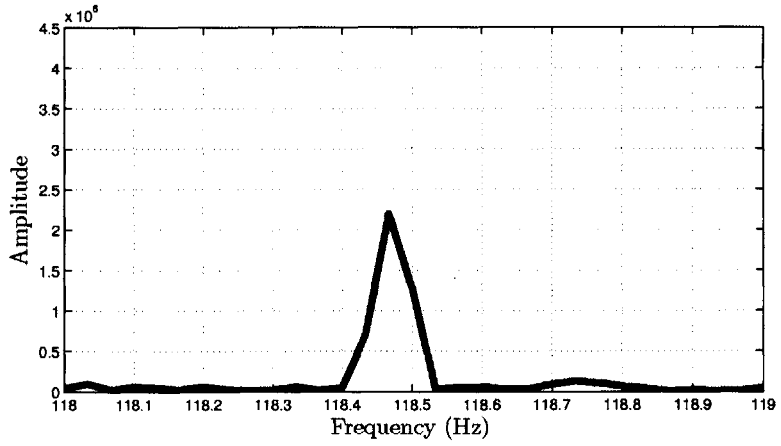


Fig. 4-21: The intake ventilation fan is then turned off and the oscillation harmonic of the exhaust fan when running by itself is shown.

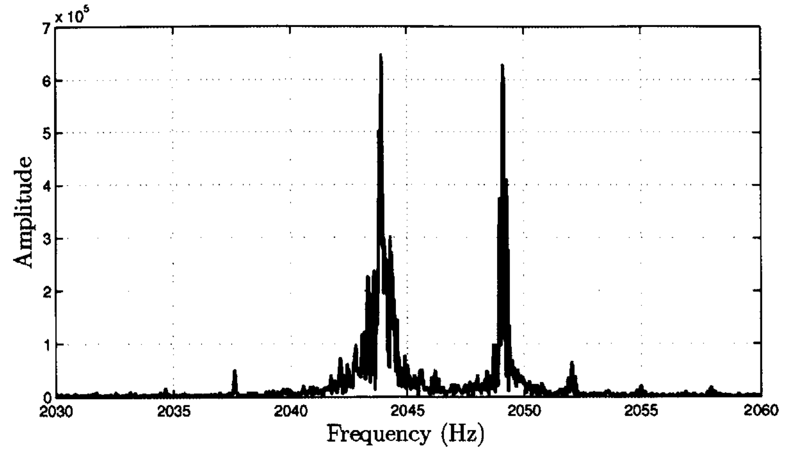


Fig. 4-22: The principal slot harmonics of the intake and exhaust ventilation fans with cardboard placed over the intake fan. These harmonics can be used to estimate the speeds of each motor.

beat period of $T_{beat-exp}$ of 9.25 seconds. Both the theoretical and experimental beat periods agree with one another.

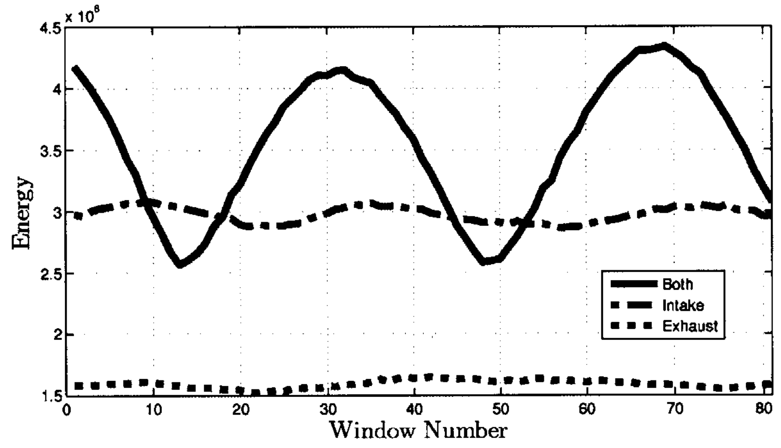


Fig. 4-23: The bottom two traces show the sliding window algorithm results when the intake and exhaust ventilation fans are running by themselves. The intake fan's airflow is blocked by a piece of cardboard. The top trace shows the sliding window when both fans are on. The harmonic energy content is periodic and the period can be determined from calculating the slips of each fan.

4.4.4 Slip Difference

The biggest problem facing the vibration monitoring algorithm deals with the slip changes that occurs when both ventilation fans are running at the same time. When both motors are loading the utility line, the voltage amplitude drops and this slows down the motors. This slip difference however also manifests itself in the shaft speed oscillation harmonic as the total energy does not remain consistent. One possible explanation is when the slip changes, the motor is operating at a new point on its torque-speed curve. Each point on the curve may react differently to eccentric weight depending on the torque turning the shaft. Another explanation for the change in harmonic content is that the circuit model in Fig. 4-1 includes a slip-dependent resistance. If the slip changes, then the amplitude of the current will also change.

Table 4.4 shows how the slip differs for the intake and exhaust ventilation fans when each is running by itself versus when they are running at the same time.

Table 4.4: Slip changes for the intake and exhaust ventilation fans. As shown, the slip increases (speed decreases) for each motor when both are turned on.

	Experiment 1 Filter		Experiment 2 Board	
Segment	Intake	Exhaust	Intake	Exhaust
Intake Only	.0237	–	.0242	–
Intake + Exhaust	.0249	.0276	.0253	.0271
Exhaust Only	–	.0260	–	.0259

Table 4.5 shows the total amount of energy in the window in which the shaft speed oscillation harmonic is located. Recall that Experiment 1 had an air filter placed over the intake fan while Experiment 2 had a cardboard blocking the airflow over the intake fan. The expected sum is the sum of the first 2 columns, the energy content from each fan when they are running by themselves. The observed sum is the actual observed energy content. The results show that there is some error in estimation.

Table 4.5: This table shows the energy content of each fan when they are running by themselves. It also shows the error between the expected energy content when both are running and the observed.

	Intake	Exhaust	Expected Sum	Observed Sum
Experiment 1	2.6796	1.6787	4.3583	4.1344
Experiment 2	2.9756	1.5902	4.5658	4.3411

If the exhaust ventilation fan's vibration had to be estimated assuming the intake's vibration is as enumerated in column 1, then the estimate would be higher than the true value. To circumvent this problem, the algorithm could have a baseline reading for each motor under all the possible permutations. There would be an baseline vibration reading for the intake fan when it runs by itself and when it runs with the exhaust fan running concurrently. Two similar baseline readings would need to be recorded for the exhaust fan.

4.5 Effect of Fan Mounting Stiffness

Ducted fans all contain three parts: a fan, a motor, and the duct pipe section. The fan is rigidly attached to the shaft of the motor forming the fan/rotor assembly. The fan/rotor assembly is secured axially and radially to the motor by a pair of deep grove ball bearings. Ball bearings, primarily through the flexibility of their constituent materials, possess a radial stiffness that can be modeled as a linear spring and damper in parallel. The motor body in a ducted fan is rigidly attached to the duct pipe section, while the duct pipe section is often attached to low stiffness but highly dampening vibration isolation mounts which secure the unit to the surrounding structure.

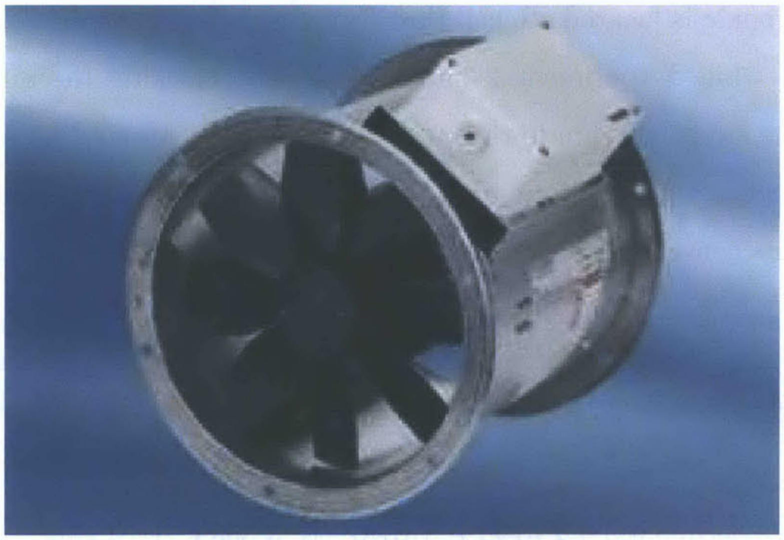


Fig. 4-24: Duct fan picture

Fan imbalance faults occur when the center of mass of a fan rotor assembly is not coincident with the center of rotation of the assembly. These faults can be easily induced in the lab by attaching a small mass to a given fan blade of an otherwise balanced or fault free fan assembly. The presence of an eccentric weight of mass m attached at radius r from the axis of rotation gives rise to an imbalance force F described acting on the rotor fan assembly given by

$$F_i = m_i \omega^2 r \sin(\omega t). \tag{4.11}$$

Assuming that the mass of the rotor assembly is very much larger than the imbalance mass ($m_r >> m_i$), the equations representing the radial motion of the fan/rotor assembly and the motor stator/ duct pipe assembly are given as follows for the x or y direction (perpendicular to the axis of the fan):

$$m_d \ddot{x}_d = -x_d k_m - \dot{x}_d b_m + (x_r - x_d) k_b + (\dot{x}_r - \dot{x}_d) b_b \tag{4.12}$$

$$m_r \ddot{x}_r = -(x_r - x_d) k_b - (\dot{x}_r - \dot{x}_d) b_b + F_i \tag{4.13}$$

The meaning of the variables above may be deduced Fig 4-25.

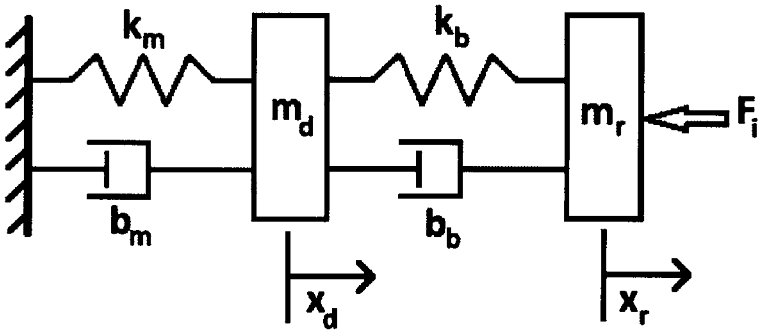


Fig. 4-25: Model of duct fan [45].

Electrical detection methods for imbalance related faults in an induction motor rely on the relative radial motion of the rotor and stator: $(x_r - x_d)$. In fault-free situations a balanced rotor runs true and its radial clearance is kept uniform by the bearing stiffness. However an imbalanced rotor will exert the F_i reaction force on the bearing causing a vibration in the radial direction. The bearing will flex under the imbalance force and a point of minimum radial distance between the inner race and the outer race will form and this point is fixed with respect to the inner race. As the balls of the bearing roll past this point, they will cause a periodic disturbance/variation in the rotational friction coefficient of the bearing. This periodic variation in the rotation friction will occur at the bearing's inner race ball pass frequency. The variation in friction will also cause a periodic rotor speed variation, and this speed variation will show up in the current signature of the motor at the same frequency. This is the basis for the electrical detection method. A metric of electrical sensitivity to imbalance is the amplitude of $(x_r - x_d)$. This metric happens to be sensitive to the properties of the vibration isolation mount, specifically its stiffness k_m. For appropriate scaling of all parameters i.e. $m_d \approx m_r$ and $k_b >> k_m$, one can get the following behavior of the electric detection metric:

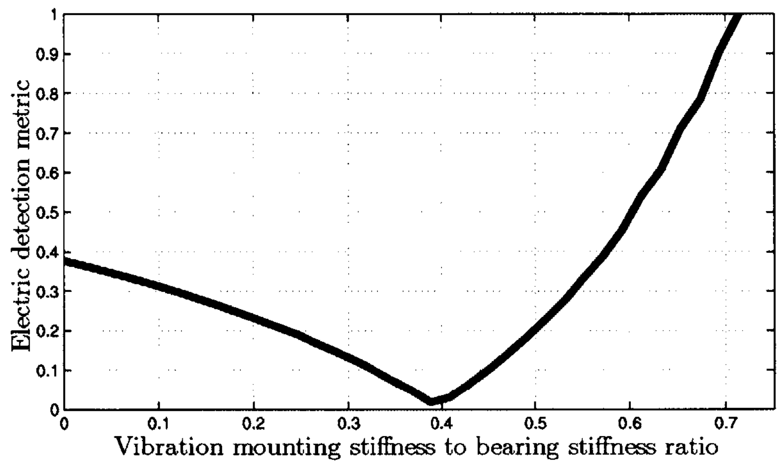


Fig. 4-26: The effect of mounting stiffness on the electric detection metric. It can be seen that the detection metric is sensitive to the vibration mounting stiffness [45].

The rise in the metric toward the right end of the plot is a resonance associated with the specific excitation frequency chosen. It can be seen that the detection metric is sensitive to the vibration mounting stiffness. The MATLAB code for the simulation is contained in Appendix C.

4.6 Summary

This chapter has demonstrated that the shaft speed oscillation harmonics can serve as an electrical-based vibration reading for monitoring the health of motors. Results from experiments in a laboratory environment were consistent with this claim. Furthermore, an algorithm was developed that could allow for diagnostics of two loads from one aggregate current signal. There are some limitations to these methods. It was shown that two loads running off the same utility may cause the slips of each load to decrease. A robust algorithm will need to take this phenomenon into consideration. Finally, tracking the shaft speed harmonics may not be feasible in all conditions. A simulation was done to show how the vibration mounting can affect the the energy content of shaft speed oscillation harmonic.

Chapter 5

Advanced Vibration Monitoring Using The Hilbert Transform

Chapter 4 demonstrated methods for tracking vibration using only electrical signals. However, there are conditions in which these electrical signals may not provide the robust metric needed for accurate condition based monitoring. A simulation demo was provided to show how vibration stiffness mounting can affect the sensitivity of the electric signal metric. This chapter will explore the disadvantages of the current methods of vibration analysis and provide new advances that improve upon these methods.

5.1 Background

Both the United States Navy (USN) and the United States Coast Guard (USCG) employ vibration monitoring for determining system health for a variety of rotating electric machine systems. The USCG currently employs outside contractors to handle their vibration monitoring needs on a periodic basis [42], although USCG crews have conducted vibration monitoring in the past. There are interesting trade-offs in "permanent" versus "occasional" monitoring. Some of the disadvantages with current methods are explained in full in §4.1. In summary, the data taken from handheld vibration meters are unreliable and measurements are taken infrequently. Equally important in both USCG and USN approaches, where information may be gathered by contractors or by an *in situ* or permanently installed vibration sensor, respectively, is the measuring instrument itself and the nature of the test. Handheld vibration sensors condense information from a broad frequency range into a single "average" number or reading. This reading mixes signal energy and information from all vibration bands and eliminates valuable indicators. Underway, e.g., as part of USN ICAS [46], it may not be possible to cleanly distinguish vibration from a component versus coupled vibration from a neighboring machine, confounding diagnosis. In-port, as the USCG conducts vibration monitoring, it is possible to isolate the operation of a single or small number of machines. However, test conditions may not reflect realistic use conditions.

This chapter will provide a flexible approach for processing sensor data that scales required communication bandwidth, storage requirements, and computation with the needs and desire of the engineering duty team. A "boiled" number or figure-of-merit, like an average vibration reading, may be all that is desired for a quick look under time pressure. It will be shown in §5.3 that steady-state vibration can be viewed as a narrowband signal. The raw acceleration signal can be filtered by a passband filter and the steady-state vibration levels can be tracked over time if an average vibration reading is all that is required.

More extensive information, e.g., energy in frequency bands, may be useful for understanding more complex pathologies. It will be shown that using this extensive information is essential in diagnosing underlying problems that cause abnormal levels of vibration. The ability to employ short-time frequency estimates to trade resolution and detail against computational complexity and data storage and transmission requirements has proven of great value in power system monitoring [8].

5.2 Extracting Spectral Envelopes With The Hilbert Transform

Most data are susceptible to the possible requirement for interpretation without full context. That is, the watchstander or shore-side maintainer will likely never have all of the information necessary to interpret sensor data with complete confidence. Part of the pressure that moves the design of sensing systems for maintenance in one direction or another, depending on the maintenance philosophy, stems from opinions surrounding the process of wading through a river of data to reach a conclusion. Conclusions drawn from data collected with "hidden" correlations to unmeasured or ignored processes are potentially little better than numerology. Reduced numbers derived from sensors, e.g., averaged over time or frequency or both, essentially become "go/no-go" indicators.

The Hilbert transform is widely used in amplitude modulation systems for its ability to extract a narrowband signal that has been modulated by a carrier frequency. It is this ability that motivates the use of the Hilbert transform in measuring the acceleration levels.. The efficacy of using the Hilbert transform for such applications, e.g., vibration monitoring, is shown in §5.3.

The underlying assumption used throughout this chapter is that there exists a narrowband of frequency content in the raw acceleration signal that can be used for diagnostic purposes. An example is shown in §5.3. The Hilbert transforms makes it possible to compute these spectral envelopes and extract useful information from the rest of the acceleration signal.

An approach for computing the Hilbert transform, $\mathcal{H}_t\{\cdot\}$, can be seen in the frequency domain. Let the Fourier transform of $x(t)$ be $X(j\omega)$ and let the Fourier transform of the Hilbert transform, $\mathcal{H}_t\{x\}$ be $\mathcal{H}_\omega\{x\}$. The relationship between $x(t)$ and its Hilbert transform in the frequency domain is

$$\mathcal{H}_\omega\{x\} = (-j \cdot \operatorname{sgn}(\omega)) \cdot X(j\omega), \tag{5.1}$$

where sgn$(\cdot)$ is the sign function. Therefore,

$$\mathcal{H}_\omega\{\cdot\} = \begin{cases} j, & \omega < 0 \\ 0, & \omega = 0 \\ -j, & \omega > 0. \end{cases} \tag{5.2}$$

In effect, the Hilbert transform applies a 90° phase shift to the negative frequency content and a −90° phase shift to the positive frequency content.

A simple example, shown in [47], gives a concrete example of how the Hilbert transform can be used to compute a spectral envelope. While it is assumed that useful content appears in a narrowband of frequency, for this example, consider the useful content exists at one frequency: ω_c. The amplitude (or vibration level) is the useful information and as it changes over time will be called $A(t)$ in the example.

Therefore, let $x(t) = A(t)\cos(\omega_c t)$ where the sinusoid of a carrier frequency ω_c is modulated by an amplitude envelope $A(t)$. Furthermore, assume $A(t)$ varies slowly compared to the "carrier" frequency and can therefore be approximated as some constant for simplicity. The Hilbert transform for $x(t)$ as shown in [47] is

$$y(t) \approx A(t)\sin(\omega_c t). \tag{5.3}$$

The next step to obtaining the envelope in amplitude modulation scenarios involves computing a signal $z(t)$ as such:

$$z(t) = x(t) + j\mathcal{H}_t\{x\}. \tag{5.4}$$

The signal $z(t)$ is called an analytic signal [47]. In the example,

$$z(t) \approx A(t)(\cos(\omega_c t) + j\sin(\omega_c t)) \tag{5.5}$$

$$\approx A(t)e^{j\omega_c t}, \tag{5.6}$$

where the last equality holds from Euler's relation. Once $z(t)$ is computed, obtaining the envelope $A(t)$ is found by taking the absolute value as

$$\left|z(t)\right| \approx \left|A(t)e^{j\omega_c t}\right| \tag{5.7}$$

$$\approx \left|A(t)\right|. \tag{5.8}$$

The signal $\left|A(t)\right|$ is the envelope of the underlying narrowband signal which is used to track vibration levels in this chapter. An implementation of computing the Hilbert transform spectral envelopes in multiple frequency bands is shown in Appendix D.

5.3 Fan Vibration

To demonstrate the use of Hilbert transforms for vibration monitoring, two ventilation fans from a USCG cutter were run in a laboratory setting conducted by Donnal et al. [48]. These fans illustrate the potential value of a more detailed look at vibration data correlated with operating condition (turn-on, turn-off, steady running, etc.).

Axial-flow fans can develop radial vibration due to imbalance. To better understand the conditions that lead to radial vibration, two ventilation fans were set up in the laboratory as shown in Fig. 5-1.

The two motors are mechanically coupled through the steel frame. The motor on the left is the intake while the motor on the right is the exhaust. The following subsections detail the experiments conducted to test the effect of certain faults on the radial vibration.

5.3.1 Rotor Imbalance

A cause of radial vibration on the outer frame of the motor is rotor imbalance. This situation occurs in the field when fan blades are damaged or fouled or when bearings wear. An imbalance can be modeled as a weight attached to the rotor. With each rotation, this extra weight causes the body of the motor to displace which leads to vibration. In the lab, weights of different sizes were added to the hub of the fan and the radial acceleration was measured for each weight. Figure 5-2 shows the top of the

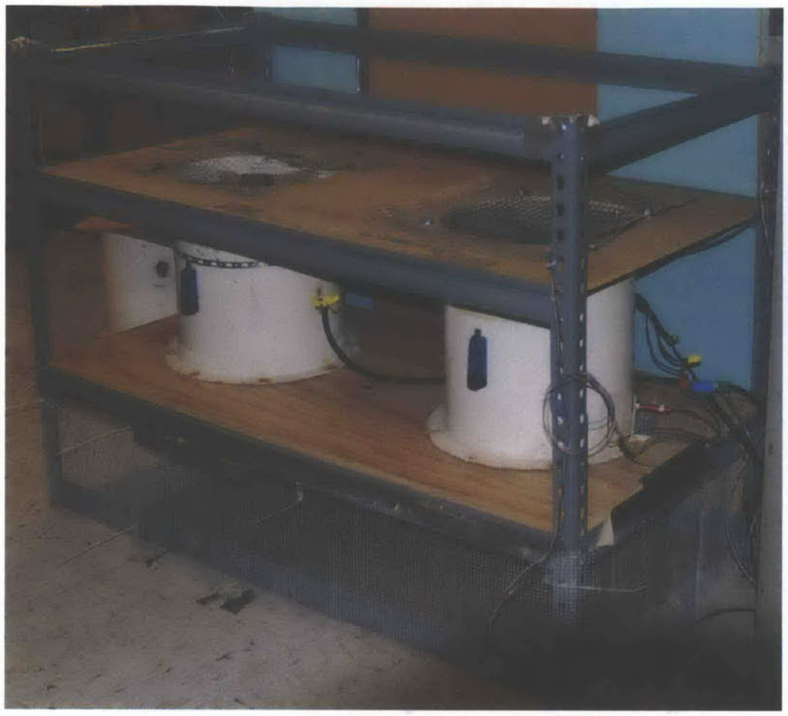


Fig. 5-1: Laboratory setup with two ventilation fans mounted to a steel frame.

fan in which no weight is added and Fig. 5-3 shows the fan that has been imbalanced.

On the top of the hub, a screw, and a screw with a nut were added on different runs. Figure 5-4 shows the spectral content for the acceleration of the intake ventilation fan during five seconds of steady-state operation between 50 Hz and 60 Hz under baseline and faulty conditions.

Almost all of the spectral content is located at the speed of the fan in rotations per second (rps). In this frequency band, the underlying signal can be approximated as

$$x(t) \approx A(t)\cos(\omega_c t), \tag{5.9}$$

where ω_c is the rotational speed of the fan and $A(t)$ is the time-varying amplitude of the sinusoid. During steady-state operation, $A(t)$ will remain approximately constant and will change only in the presence of a fault.

With high-resolution data and an appropriate windowing transform, e.g., the Hilbert transform in this case, we are not restricted to examining only steady-state information. The Hilbert transform can be applied to extract time-evolving content

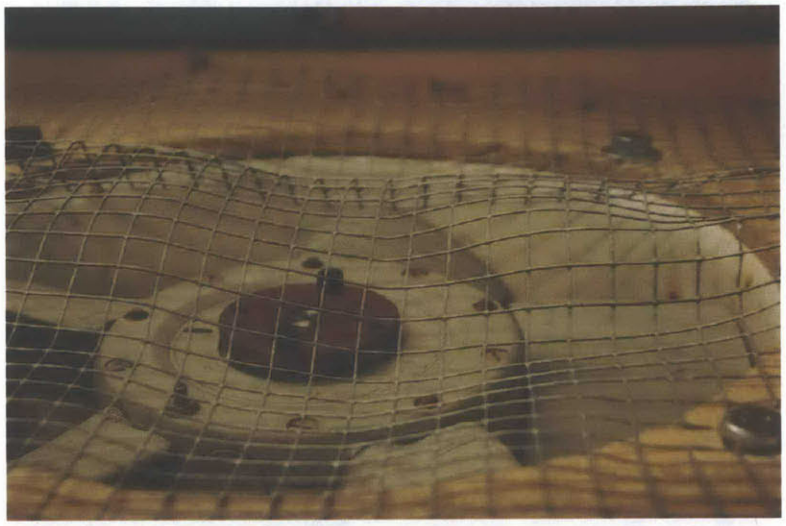


Fig. 5-2: A top view of the intake ventilation fan with no imbalance.

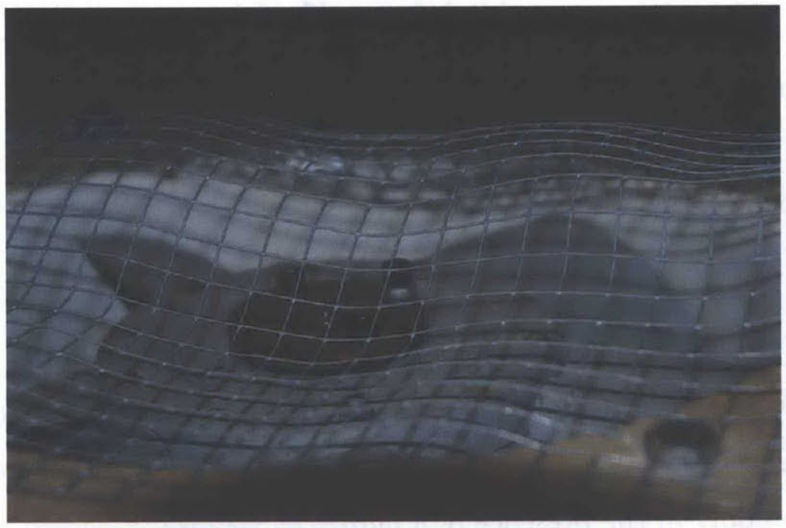


Fig. 5-3: A screw attached to the fan hub imbalances the motor causing radial vibration.

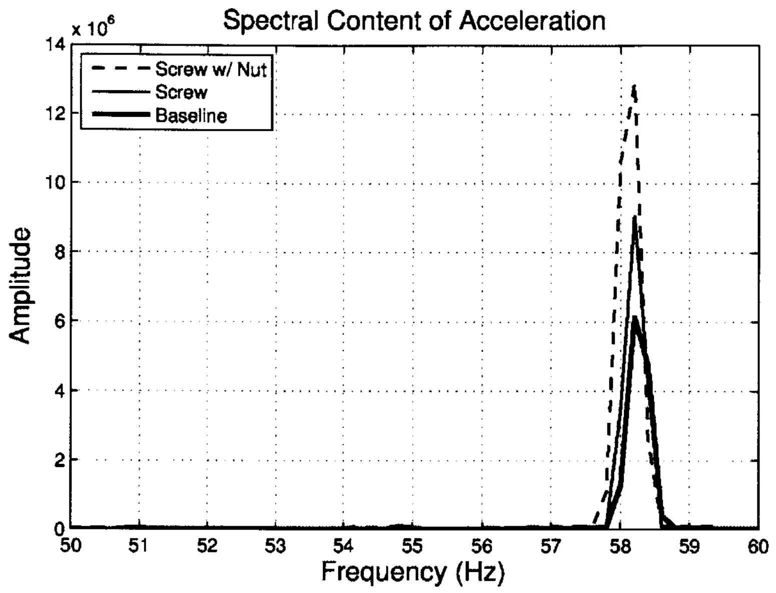


Fig. 5-4: Spectral content of the ventilation fan for five seconds of steady-state operation under baseline and faulty conditions between 50 and 60 Hz. All of the energy is centered around the frequency corresponding to the rotational speed of the fan. Approximating the energy in this frequency as a single sinusoid allows for the use of the Hilbert transform to easily track the envelope of the underlying sinusoid over time [48].

in small frequency bands of acceleration. For example, the acceleration levels in the 50 – 60 Hz frequency band for each run are shown in Fig. 5-5. The units of "acceleration" in this plot are "counts" from the analog-digital converter reading from the accelerometer, and are of course limited to a specific frequency band. They do not correspond directly to the "g" reading on a hand-held accelerometer, but they are correlated with the vibration meter readings. With each increase in weight, the fan is more unbalanced and the amount of radial vibration increases as expected and can be tracked over time.

It should be noted again that the passband energy used in this chapter produce arbitrary units that cannot be converted to "g" or any other standard units of acceleration. However, as shown in Table 5.1, the passband energy levels monotonically increase as the amount of eccentric weight on the fan hub is increased. The values from the handheld meter follows the same pattern.

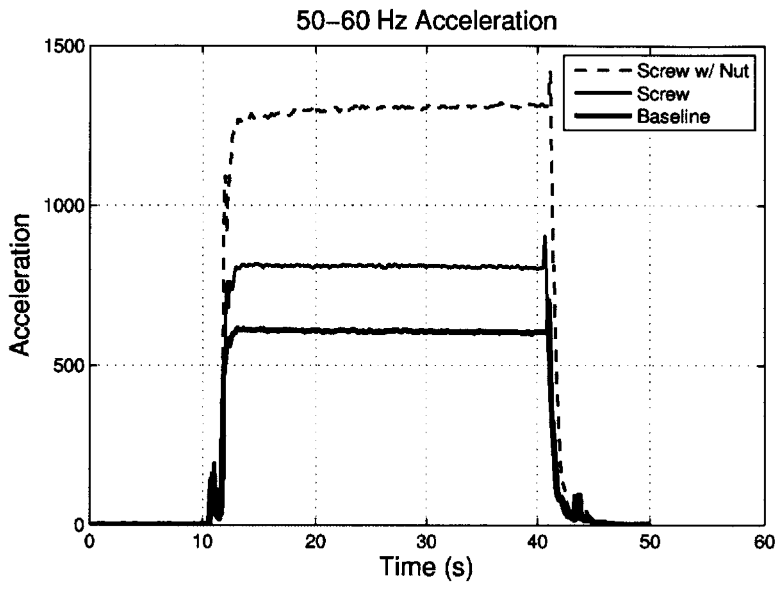


Fig. 5-5: Acceleration levels in the 50 – 60 Hz frequency bands for three different runs. The baseline test in which the fan has no imbalance gives the lowest amount of vibration. A second test was conducted with a screw added to the fan hub. The last test added a screw and a nut. The vibration levels increase accordingly [48].

Table 5.1: Numerical energy levels in the second column for each of the tests conducted with the intake ventilation fan. These readings are consistent with those from a handheld vibration meter shown in the last column.

	Passband Energy Level	Handheld Vibration Meter (g RMS)
Baseline	610	0.10
Screw	810	0.55
Screw & nut	1300	0.70

5.3.2 Loose Mounting

A second cause of radial vibration is from loose mounting. If a mounting or frame loosens over time, the mount can no longer dampen any vibration. This results in an increase in the radial vibration of the body of the machine. An improperly mounted machine can add noise which can be a source of distraction for the crew and a source of acoustic signature in the water. Mechanically, loose mounting also causes fatigue in the structure of the motor which can reduce the lifetime of the machine.

To emulate a loose mount in the laboratory, the mounting screws attached to the frame of the motor were loosened. Figure 5-6 shows a properly tightened screw which mounts the intake motor to the steel cage. The screw is loosened as shown in Fig. 5-7.

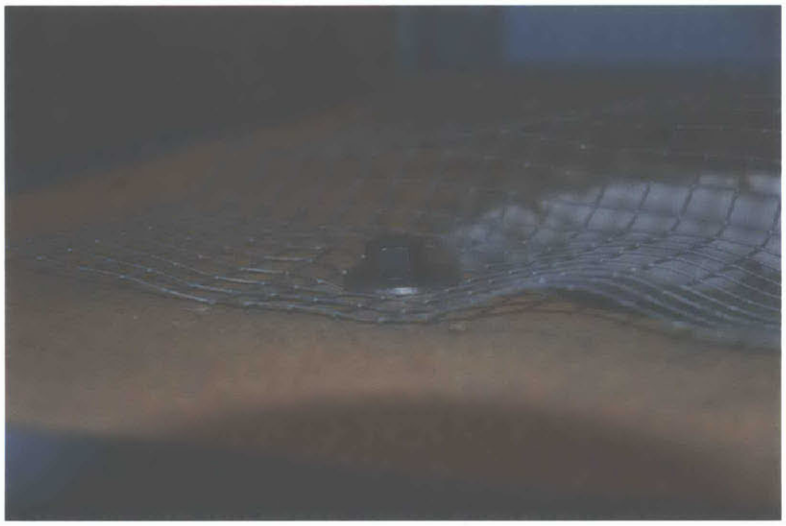


Fig. 5-6: An intake ventilation fan which is properly mounted to the steel frame.

As the mounting screw was loosened, the amount of radial vibration increased as expected as shown in Fig. 5-8. These values are consistent from the values obtained from a handheld vibration meter. A summary of the values is shown in Table 5.2.

5.4 Differentiating Source of Vibration

Simple vibration readings can only indicate if there is a significant change in the amount of vibration. They cannot, however, diagnose the potential cause of the

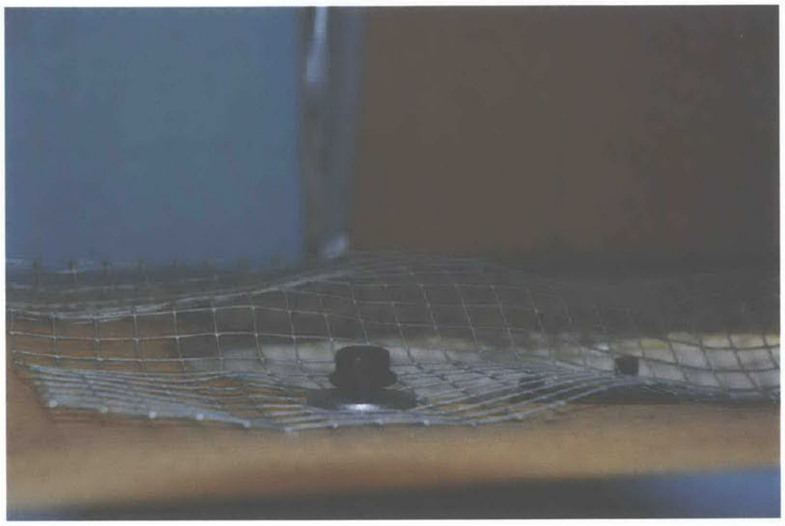


Fig. 5-7: An intake ventilation fan with loosened mounting screws.

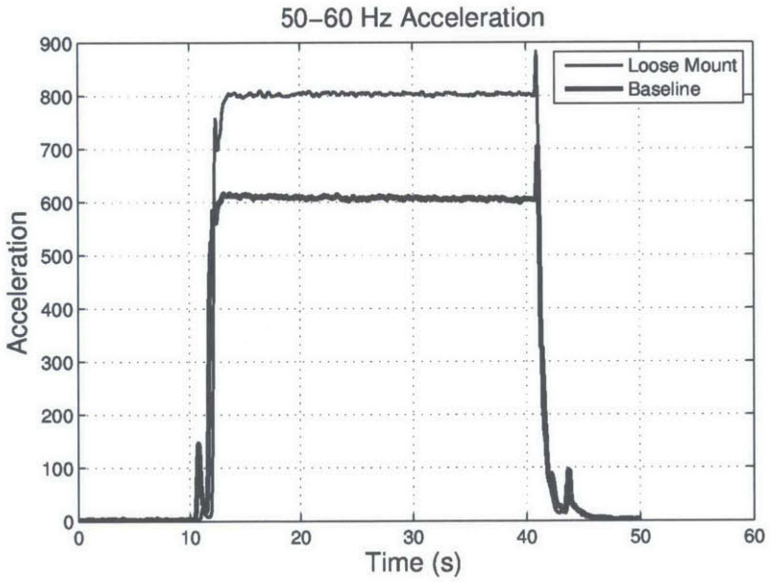


Fig. 5-8: The acceleration content in the 50–60 Hz frequency band for a baseline test is shown. A second run was conducted in which the mounting screws are loosened. The acceleration content is also shown. As expected, the readings show an increase in radial vibration [48].

Table 5.2: This table shows the numerical energy levels in the second column for each of the tests conducted with the intake ventilation fan. These readings are consistent with those from a handheld vibration meter shown in the last column.

	Passband Energy Level	Handheld Vibration Meter (g RMS)
Baseline	610	0.10
Loose	805	0.23

problem. In Fig. 5-9, the baseline for the intake motor is shown as the bottom traces. The top two traces depict the two faulty runs of the ventilation fans. In one run, the motor has a rotor imbalance by adding an eccentric weight to the fan hub. In the other run, the motor is loosely mounted to the steel frame. The vibration readings for the faulty runs are higher than that from the baseline but are indistinguishable from one another in this example. Diagnosing the problem between a rotor imbalance or a loose mount cannot be inferred from a vibration reading alone.

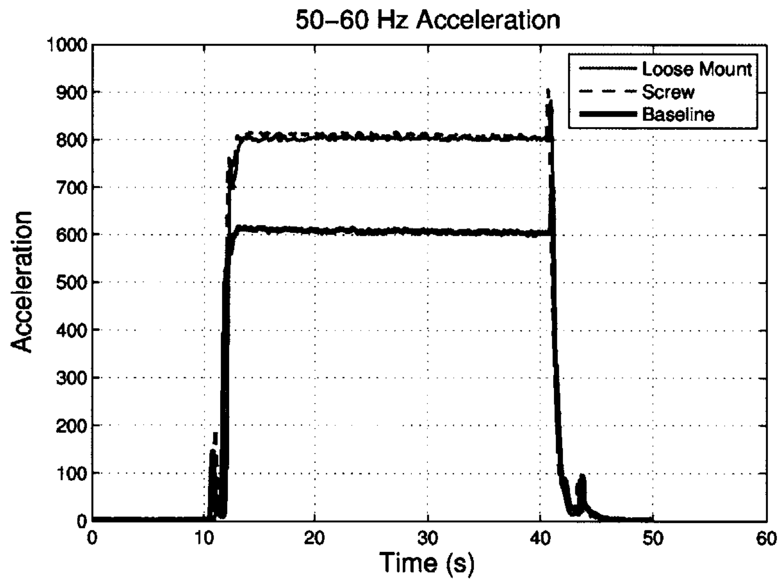


Fig. 5-9: Acceleration levels for the intake fan under normal ("baseline") conditions, with a rotor imbalance ("screw"), and with a loose mount. The faulty runs both show an increase in vibration. However, this information alone is insufficient to differentiate the two flaws. More analysis is needed to diagnose the underlying fault [48].

During turn-off transients of the ventilation fan, the spectral content of the fan's acceleration passes through the lower frequency bands as the fan slows down. When the fan is loosely mounted, the vibrational energy is dampened such that the envelope is indistinguishable from the baseline as show in Fig. 5-10.

There is, however, no dampening effect with a rotor imbalance. The spectral content is higher than those from the baseline and the loose mounting runs. While vibration readings in the 50 – 60 Hz bandwidth give a simple indication as to the deviation from the baseline reading, vibration levels in the other frequency bands allow for a richer assessment. These methods for vibrational analysis were tested

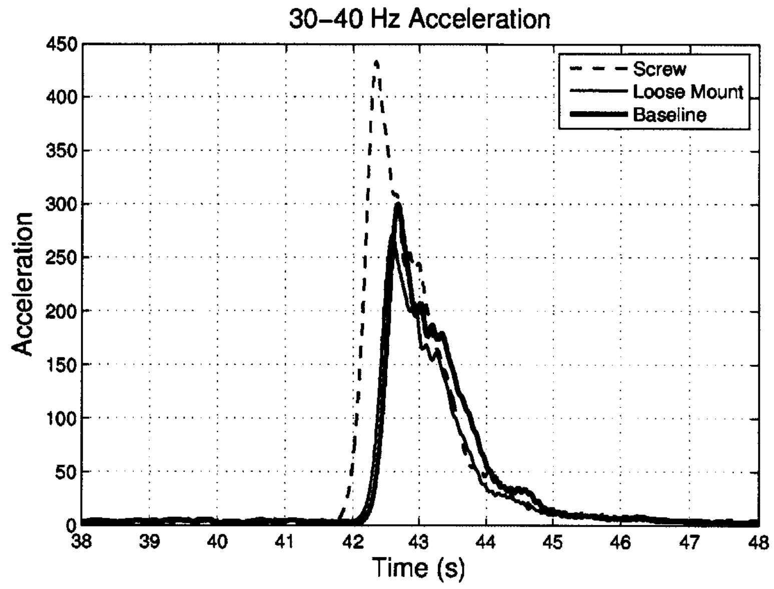


Fig. 5-10: Analysis of the 30–40 Hz frequency band is needed to differentiate between a rotor imbalance of a loose mount. During turn-off transients, there is more energy in the sub-bands when there is a rotor imbalance [48].

onboard the USCG *ESCANABA*.

5.5 *ESCANABA* Data

Measurements discussed in this section were taken on board USCG *ESCANABA* (WMEC-907) shown in Fig. 5-11.

Measurements were taken from two paired vacuum pumps used to maintain pressure in the waste water tank while the ship was at port. The motors are located in the top compartment of the engine room. An image of these motors is shown in Fig. 5-12. The motor in the foreground will be referred to as Motor A. The motor in the background will be referred to as Motor B.

Observations were made during which an accelerometer was physically attached to each motor and the corresponding motor is turned on while the other motor remains off. The sample rate of the accelerometer is 160 Hz. All of the analysis is run against bands less than 60 Hz so the sample rate is higher than the Nyquist rate of 120 Hz.

Figure 5-13 shows the raw data of the acceleration for Motor A in a trial run.

Fig. 5-11: USCG *ESCANABA* (WMEC-907). Measurements were taken from two paired vacuum pumps located above the ship's engine room.

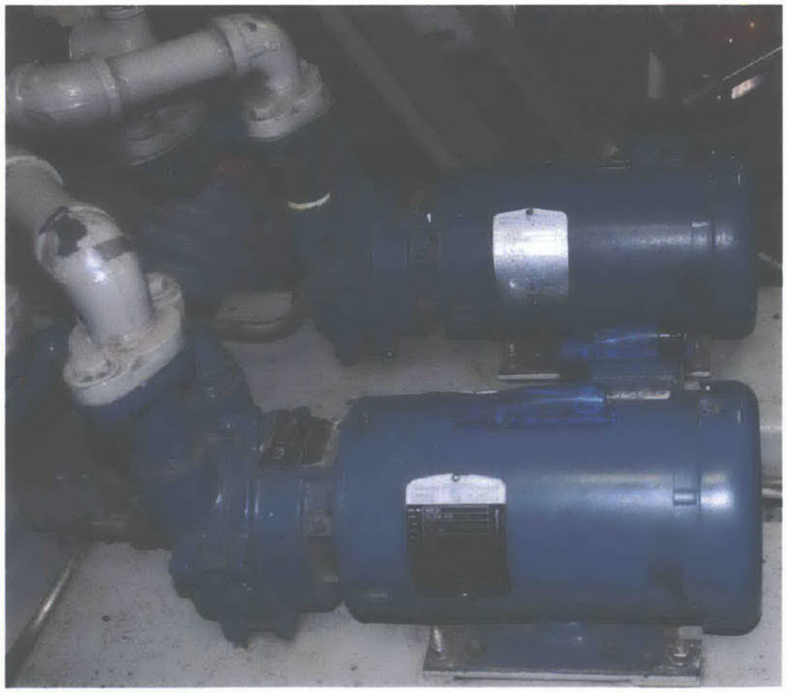


Fig. 5-12: Two vacuum pumps in the engine room on the *ESCANABA*. The motor in the foreground will be referred to as Motor A. The motor in the background will be referred to as Motor B.

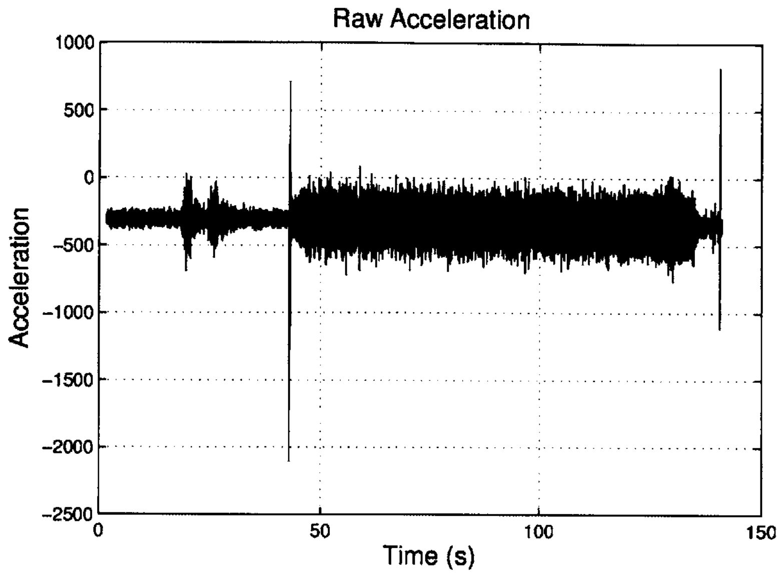


Fig. 5-13: Raw acceleration of motor A when turned on [48].

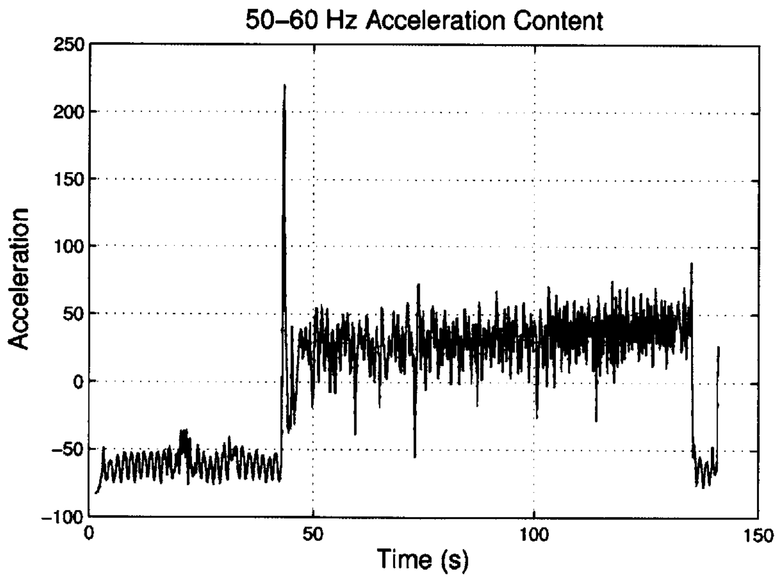


Fig. 5-14: The acceleration content takes the raw data and computes the envelope of the Hilbert transform of the data between 50 and 60 Hz. The acceleration content metric can be computed by taking a region when the motor is in steady-state operation [48].

The motor is turned on at around 43 seconds and switched off at 141 seconds. The envelope in the 50-60 Hz frequency band of motor A is shown in Fig. 5-14. The raw data was filtered with a bandpass filter center around these frequencies. The amplitude of the Hilbert transform of this output signal is the envelope. This band was chosen because the nominal speed of the pump is 3420 rotation per minutes (rpm) or 57 rotations per second.

Ideally, the Hilbert transform is applied to narrowband signals in which the amplitude changes slowly compared to the carrier frequency. In this example, this condition does not hold and there are oscillations. A moving average is computed on the signal to smooth out these oscillations During steady-state operation of the motor, the Hilbert transform can compute a metric that correlates directly with vibration. This value provides a baseline to which comparisons from future tests will be made. As with a handheld meter, this test condenses the information into a single number and serves as a reference point.

A second test was performed in which the the accelerometer is placed on motor A while motor B is running. The raw acceleration data is shown in Fig. 5-15.

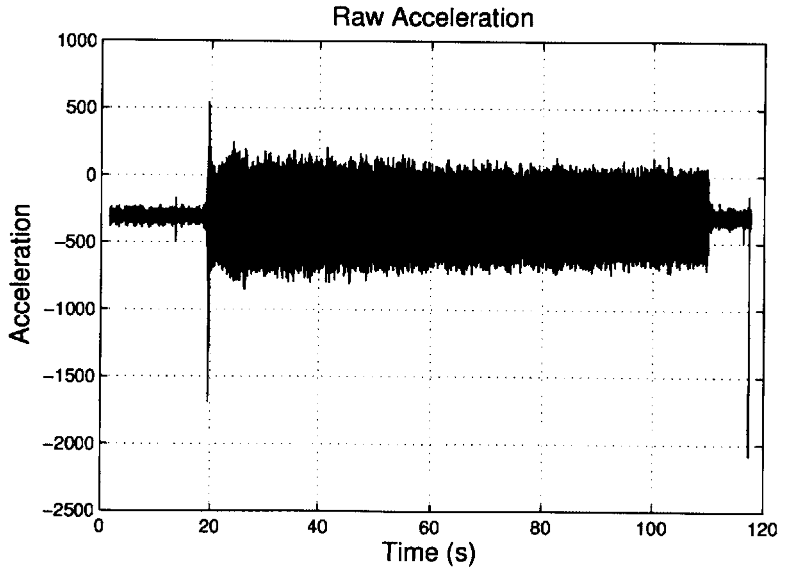


Fig. 5-15: Raw acceleration of motor A when motor B is turned on. The amplitude of the signals is greater than that in Fig. 5-13 [48].

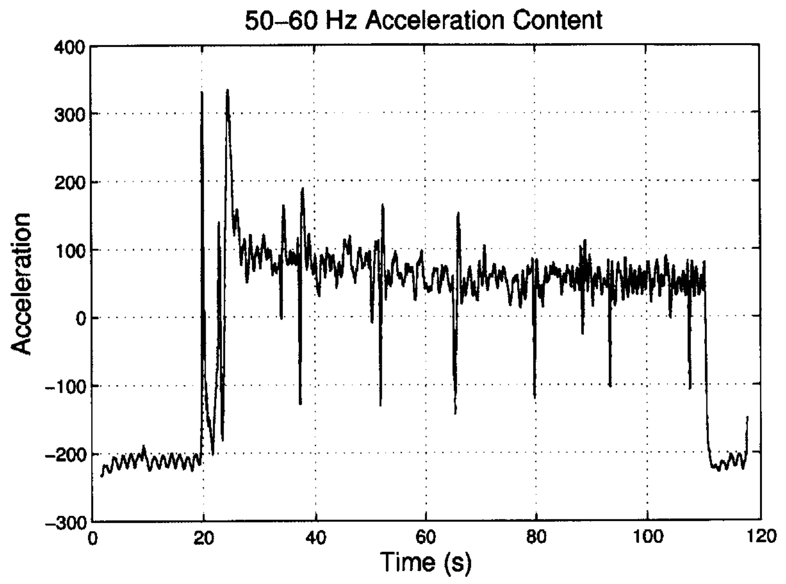


Fig. 5-16: The acceleration content is similar to that in Fig. 5-14. These metrics can be misleading if no further analysis is done [48].

Once again, the Hilbert transform is used to compute a numerical metric. It should be noted via visual inspection that this metric will be larger than that recorded when motor A was running. In other words, the accelerometer on motor A detects a higher vibration when another motor is on.

A purely automated analysis of the vibration readings (ignorant of operating schedule) from this experiment or from a vibration meter would inaccurately ascribe vibration readings to the wrong motor. In this instance, the machines are mechanically coupled. A small vibration from Motor B can be enhanced through the coupling leading to a large vibration reading from an accelerometer placed on Motor A. These cross-interference effects cannot be inferred from boiled numbers.

However, analysis from the time-series data can show if and how two or more motors are mechanically coupled. Figure 5-17 shows the raw data of the acceleration on Motor A when both motors A and B are on. The interaction between the two motors is apparent as there is a beat frequency in the data.

If prior knowledge is known as to whether or not two motors are mechanically coupled, this information could be included in the analysis of vibration measurements. Furthermore, the fidelity of the data can be maintained depending on which motors

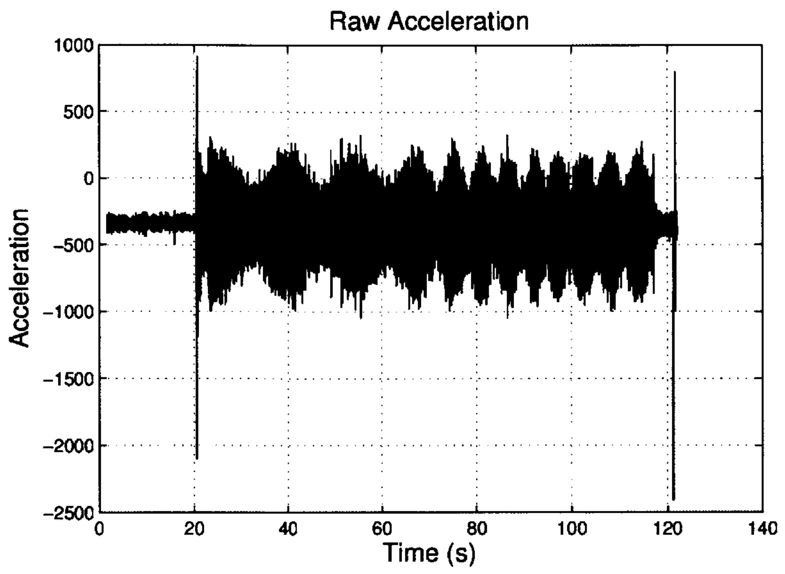


Fig. 5-17: The raw acceleration of motor A when both motors A and B are turned on. Visual inspection of the time series shows that there is coupling between the motors as there is a beat frequency present. This observation can be necessary in improving vibration diagnostics [48].

are turned on. For instance, the data for the vibration of Motor A can be rejected if it is known that Motor B is also turned on. A nonintrusive load monitoring (NILM) environment provides a platform in which loads can be monitored by their turn-on and turn-off transients to keep track of which loads are on and off. A brief discussion of the NILM environment is described in Chapter 2.

Not only can mechanically coupled motors provide false readings, but the actual sea state can also influence the data. Data of these motors were also taken when the *ESCANABA* ship was underway. Low-frequency acceleration caused by the rocking of the ship also add to the vibration readings. This phenomenon is readily seen in the raw acceleration in Fig. 5-18.

Before the motor turns on at the 13 second mark, a low-frequency oscillation caused by the ship's rocking is visible. A low-pass filter removes the sea state from the rest of the data and its envelope is shown in of Fig. 5-19. All other frequencies add background noise which makes the handheld readings less reliable as shown in the second graph. A band-pass filter around the rotational speed of the motor isolates the actual vibration reading from the motor shown in the Fig. 5-20. Processed time

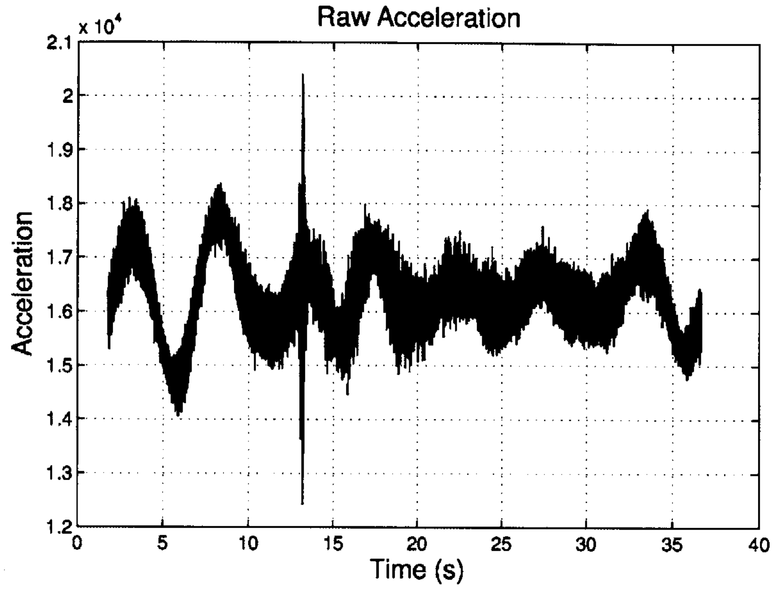


Fig. 5-18: Raw acceleration of motor A while the ship was at sea with visible acceleration from the rocking of the ship [48].

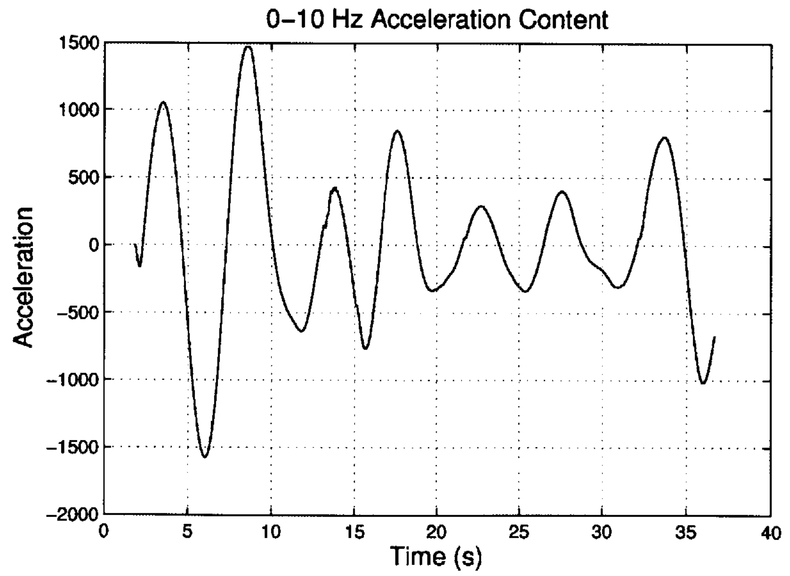


Fig. 5-19: The visible decoupled low-frequency rocking [48].

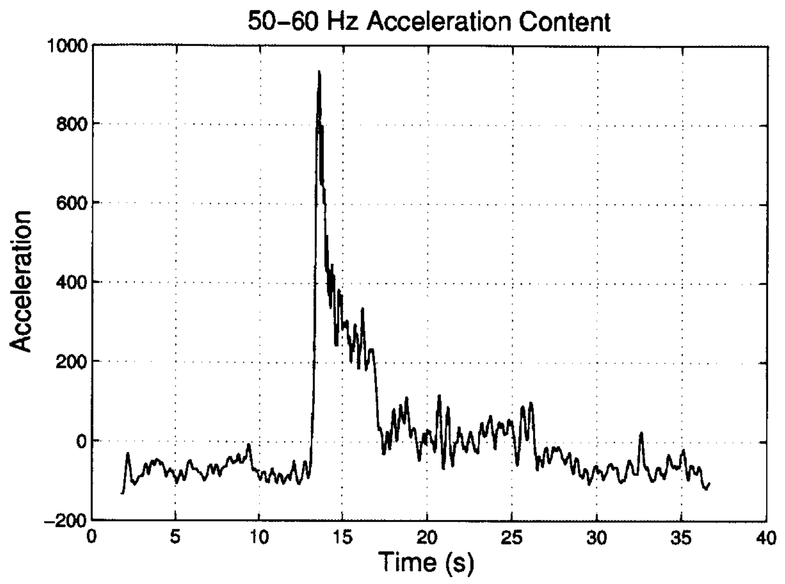


Fig. 5-20: The actual vibration from motor A [48].

series data permits background information like sea state to be removed, revealing desired information.

5.6 Summary

Common methods for collecting data for system diagnostics are susceptible to error from a variety of sources. Hand sensors may be deployed incorrectly. Attempts to reduce data to figures-of-merit can hide important details. And individual "flavors" of sensors, e.g., vibration or temperature, may provide information that is best understood in the context of its correlation with other variables such as power consumption. Empowering crews to to examine sensor data at scalable or adjustable levels of detail, as appropriate, to investigate a problem or casualty will result in improved fleet readiness with reduced operating costs.

Chapter 6

Behavioral Modeling Framework

Electrical power systems for naval vessels face spiraling demands in absolute and transient power delivery requirements. Naval vessels, particularly warships with relatively large and increasing power requirements, offer a unique laboratory for understanding "islanded" or isolated power grids. Reflecting a potentially wider concern for small power distribution systems, existing approaches for modeling and designing Naval power systems have not necessarily kept pace with increasingly complex operational requirements. For the US Navy today, the DDG-51 is the largest ship class at sea and its power system represents a canonical example of a shipboard power grid. This chapter examines the DDG-51 power distribution system as an example for motivating new opportunities for power system analysis and design based on a rich but computationally reasonable approach for behavioral load modeling.

6.1 Motivation and Background

The design process for Navy ships is complex as it requires making decisions in the present for ships that will be operating several decades into the future. On-board electrical generation plants have seen tremendous growth over the course of the 20th and 21st centuries and this trend shows no signs of abating. In a recent study [49] performed by the US Navy on alternative propulsion methods, the authors examined the maximum margined electrical load for ships. The resulting historic and projected electrical load growth, driven primarily by combat systems load growth, is shown in Fig. 6-1.

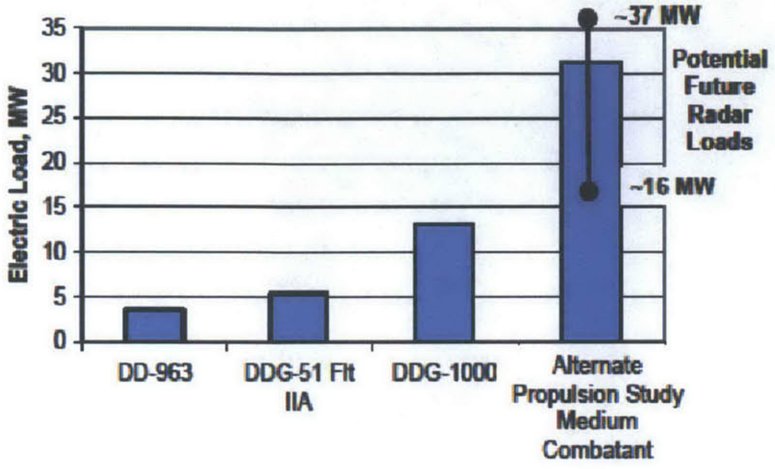


Fig. 6-1: Electrical Load Growth on Surface Combatants [49].

At the same time, a study on alternative propulsion methods conducted by the Navy, through the Naval Sea Systems Command (NAVSEA), produced a technology roadmap for the next generation integrated power system (NGIPS) [50]. An integrated power system (IPS) in the Navy refers to a system in which the ship's service electrical distribution and ship's propulsion power are provided by a single distribution system. The purpose of the NGIPS development is to understand the technical challenges and the enabling technologies required to move to an all-electric warship. Demonstrated graphically, the roadmap is shown in Fig. 6-2.

A limitation called out in the NGIPS process, however, is that the current methods used to size and characterize shipboard loads may be insufficient for these future electrical distribution methods [50]. Defining the loads is critical in designing and sizing major electrical generation components.

The design guidance for determining electrical plant sizing calls for first developing an electric power load analysis (EPLA). This guidance is codified in the data design sheet (DDS) 310-1 [51], published by NAVSEA. The EPLA is a crucial portion of the design of a ship as it is used to determine component sizing in the electrical distribution system for everything from generating capacity to breaker and cable sizing. One limitation presented by this method is that the EPLA calculation has

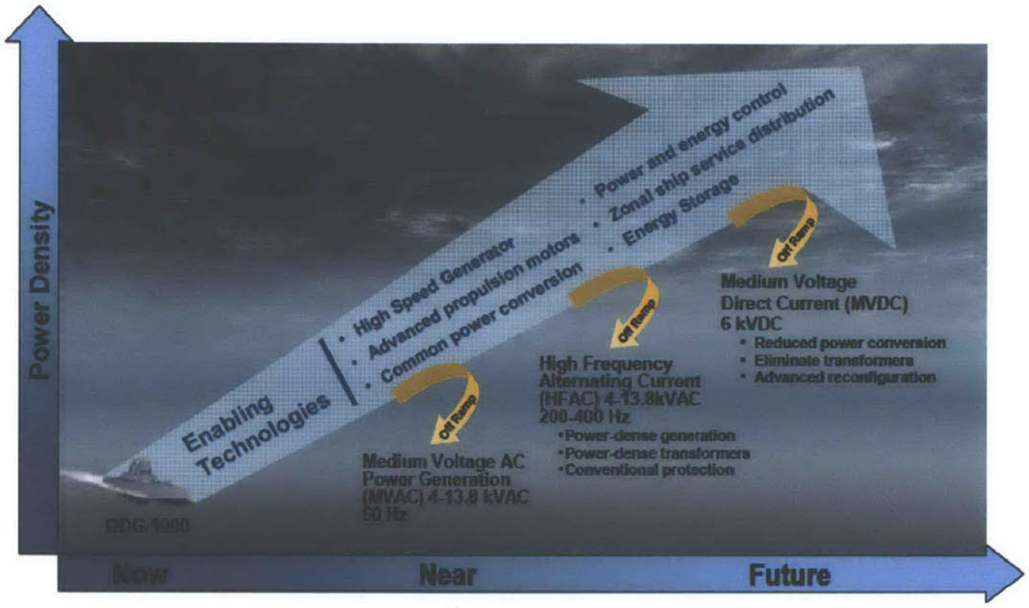

Power Density
Enabling Technologies
• High Speed Generator
• Advanced propulsion motors
• Common power conversion
• Power and energy control
• Zonal ship service distribution
• Energy Storage
Off Ramp
Medium Voltage AC Power Generation (MVAC) 4-13.8 kVAC 60 Hz
Off Ramp
High Frequency Alternating Current (HFAC) 4-13.8kVAC 200-400 Hz
•Power-dense generation
•Power-dense transformers
•Conventional protection
Off Ramp
Medium Voltage Direct Current (MVDC) 6 kVDC
• Reduced power conversion
• Eliminate transformers
• Advanced reconfiguration
DDG 1000
Now
Near
Future

Fig. 6-2: NGIPS Technology Development Roadmap

been performed historically with a standardized process [52]. Additionally, the EPLA is not presented in a manner that supports answering dynamic questions, such as quality of service [53].

Developing a model for a ship's power system that is flexible enough to remain viable throughout the different stages of the ship's life cycle would address both current and future needs. Such a model would be capable of not only producing a single value for the expected shipboard load but could be queried to show changes output based on varying input parameters. Instead of functioning merely as a tool to size electrical components, it could be used to develop dynamic fuel consumption estimates or perform sensitivity analyses based on changing fleet operational needs. A program developed to be extensible on both the input and output side would provide the entire design community a tool useful for answering problems not yet envisioned. This chapter will develop the framework for such a model by using the DDG-51 class to examine current methods and define one possible process to build a next-generation model.

6.1.1 Why Focus on the DDG-51 Class?

The DDG-51 class (or ARLEIGH BURKE class) is simultaneously the largest current class of ships in the Navy's inventory and a large portion of planned acquisition in the coming years. To date, there have been 62 DDG-51 class ships commissioned, with three major class variants. These variants are known as the Flight I, II, and IIA ships. The Navy's 30-year shipbuilding plan expects a fourth variant, the Flight III, to enter service in the early 2020's [54]. These naval assets will therefore be constructed and utilized well into the future. The expected inventories of surface combatants over the coming decades will be dominated by the DDG-51 class, as can be seen in Fig. 6-3.

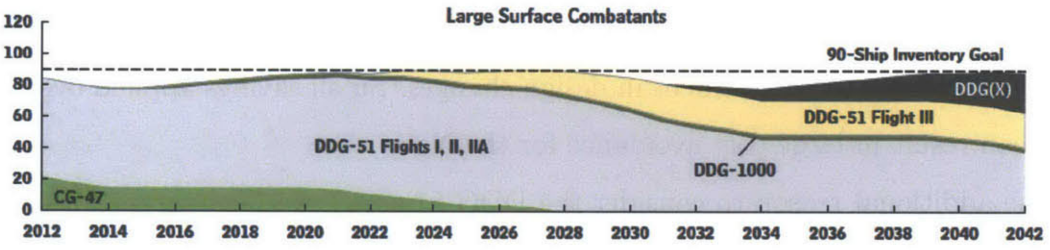


Fig. 6-3: Expected Inventory of Large Surface Combatants for Next 30 Years

The quantity and length of production for these ships make it a unique platform to study for several reasons. The construction of the Flight III version will require design work in the coming years for which a large body of data already exists. The ability to fully understand the manner in which the current DDG-51s are operated gives a potential advantage to the ship design community, as fewer assumptions are required in the design.

Another potential benefit of studying this class of ships is that the large number in operation makes them a fertile research area for technology changes or enhancements. Systems such as hybrid electric drive (HED) are currently being developed and tested for potential inclusion into the propulsion train of future or existing destroyers. With the HED system, the ship would utilize a motor powered by the ship's electrical distribution system at slow speeds as a means of saving fuel [55]. Other recent studies have examined hydrodynamic changes such as larger or contra-rotating propellers, a bow bulb, a stern bulb, or an updated stern flap [56]. Each of these hydrodynamic

Table 6.1: DDG-51 Class Basic Characteristics

DDG Characteristics			
	Flight I	**Flight II**	**Flight IIA**
Length (ft)	505	505	509
Beam (ft)	59	59	59
Draft (ft)	30.5	30.5	30.5
Displacement (LT)	8320	8673	9496
Manning	276	276	276
Speed (kts)	30+	30+	30+
Shafts	2	2	2
Gas Turbines (Per Shaft)	2	2	2

changes would be performed as a means of lowering fuel consumption over the operational lifetime of the ship. These studies also demonstrate that the large size of the ship class justifies the investment in design changes. Small savings applied over many ships can result in large cost avoidance for the Navy.

One additional reason to consider the DDG-51 as an excellent target for study is that it has a large number of onboard sensors monitoring equipment. The growth of remote sensing and control onboard ships has increased in recent designs, as sensors have become more prevalent. The ability to collect data for future analysis using onboard functionality is a capability that was exploited to gather data in this thesis.

6.1.2 DDG-51 Class Ship Introduction

The ARLEIGH BURKE class destroyers are modern warships capable of fulfilling missions ranging from anti-air warfare (AAW) to anti-submarine warfare (ASW) to anti-surface warfare (ASUW). To accomplish this variety of missions, the ships are constructed with a wide variety of weapons and detection systems. Weapons systems include a 5-inch gun, a variety of air, land, and anti-missile missiles, torpedoes, and small arms. Detection systems range from SPY-1D 3-D phased array radar to surface search radar to the SQQ-89 sonar [57]. This wide variety of equipment onboard ensures that the DDG-51 possesses tremendous operational flexibility. A summary of the basic characteristics of the DDG-51 class is shown in Table 6.1.

The DDG-51 has two main machinery rooms, each of which independently possesses the necessary equipment to power one shaft. Each shaft has two gas turbine modules (GTM), each of which contains a GE LM 2500-30 gas turbine engine capable of generating approximately 27000 shaft horsepower (SHP). Each pair of GTMs is mated to its associated shaft through a main reduction gear, taking the high speed turbine output and producing a lower speed high torque shaft rotation. Each shaft is equipped with a controllable pitch propeller (CPP), and is operated with programmed logic to select an optimum pitch and shaft rotational speed for a given ordered speed through the water.

The engineering modes of the ship will be referred to as the selection of the propulsion train configuration for the vessel. The trail shaft engineering mode is a configuration where one shaft is being powered by a single GTM while the remaining shaft is allowed to spin free. The trail shaft configuration is the most efficient engineering mode in terms of fuel consumption, at the cost of a restricted top speed. Additionally, the trail shaft mode uses only a single GTM, which presents a potential operational risk in the event of an engine casualty. The reliability concern and speed restriction can be partially addressed using the split plant engineering mode. The split plant mode consists of two operating GTMs, with one powering each shaft. Split plant allows greater top speed (though still limited), and increased propulsion reliability when desired by ship operators. The full power operating mode for the engineering plant is the condition where all 4 GTMs are operated, with 2 GTMs powering each shaft. While the full power mode allows the ship to travel at its maximum speed of 30+ knots, it is the least fuel efficient method of operating the ship. The full power mode also represents the maximum amount of propulsion operating waveform. Alternatively, a master component is on when its subsystem is active, but has its own operational profile that also modifies or defines the top trace in Fig. 6-31, described by a stochastic model that is defined by the user. redundancy for the ship, and is typically utilized when the potential impact of a propulsion casualty could lead to catastrophic effects for the ship.

The electric plant for the DDG-51 class has some variation depending on the flight of the vessel in question. All DDG-51 class ships generate electrical power using 3 Allison 9140 gas turbine generators (GTG). The earlier GTG sets were rated to 2500 kW, with later ships being built with 3000 kW generator sets. The typical operating configuration for the DDG-51 is to have 2 of 3 GTG sets online. The original Flight I design of the DDG-51 electrical plant utilized a radial distribution system, which was fully replaced by a zonal electrical distribution system (ZEDS) for ship hull numbers 78 and higher. The ZEDS architecture is designed to distribute electrical power within specific zones to ensure the ship is better prepared to retain fighting capability following potential damage. The ZEDS breaks the ship down into 6 primary zones, and with buses serving the upper and lower portions of the ship [58]. The electrical generation system also utilizes multi-function monitors (MFM) to provide fault isolation for the distribution system. Improvements to fault isolation algorithms used by the MFMs are presented in Chapter 7. A pictorial representation of all major switchboards, and the MFM monitoring them, is shown in Fig. 6-4.

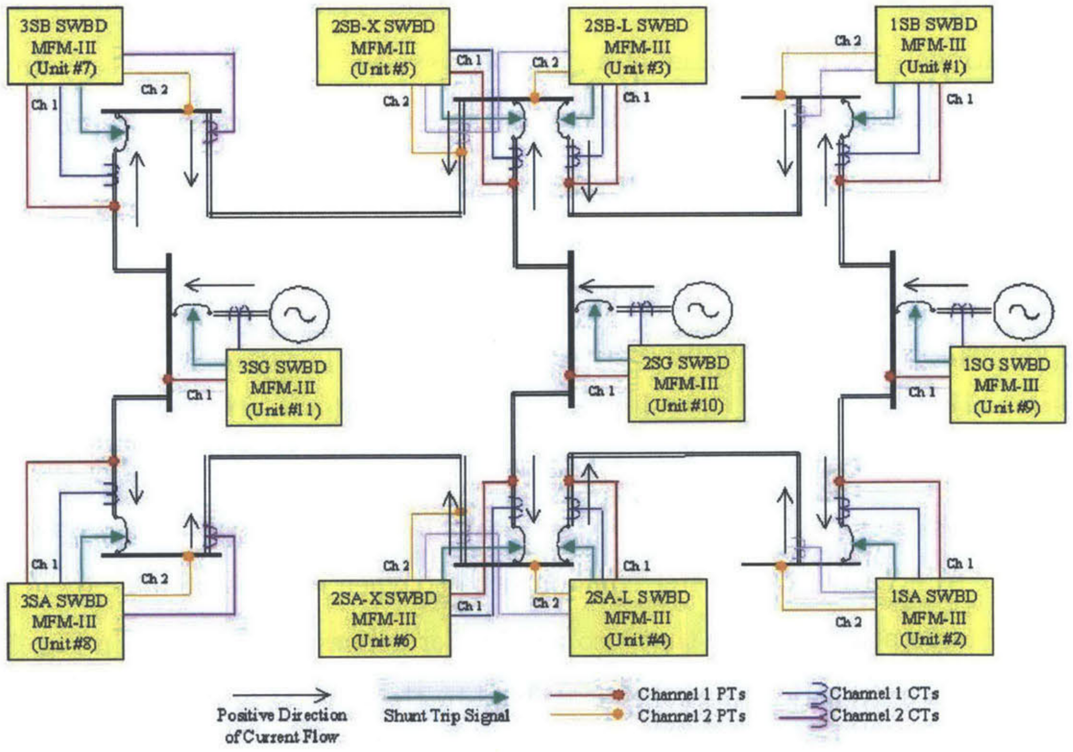

3SB SWBD MFM-III (Unit #7)
2SB-X SWBD MFM-III (Unit #5)
2SB-L SWBD MFM-III (Unit #3)
1SB SWBD MFM-III (Unit #1)
3SG SWBD MFM-III (Unit #11)
2SG SWBD MFM-III (Unit #10)
1SG SWBD MFM-III (Unit #9)
3SA SWBD MFM-III (Unit #8)
2SA-X SWBD MFM-III (Unit #6)
2SA-L SWBD MFM-III (Unit #4)
1SA SWBD MFM-III (Unit #2)
Ch 1
Ch 2
Positive Direction of Current Flow
Shunt Trip Signal
Channel 1 PTs
Channel 2 PTs
Channel 1 CTs
Channel 2 CTs

Fig. 6-4: ZEDS Electrical Distribution (Switchboards and MFMs Shown) [59].

6.2 Electric Plant Load Analysis (EPLA)

The design guidance for determining the electrical plant sizing calls for developing an electric power load analysis (EPLA). This guidance is codified in the data design sheet (DDS) 310-1 [51], published by NAVSEA. The EPLA is a crucial portion of the design of a ship, such as the DDG-51 class. It is utilized to determine component sizing in the electrical distribution system for everything from generating capacity to breaker and cable sizing.

One limitation presented by this method exists, however, in that these have been historically performed without a standardized process [52]. Additionally, the EPLA is not presented in a manner that supports answering dynamic questions, such as quality of service [53].

The EPLA uses the list of all components installed on-board the ship to calculate demand power. This analysis of demand load is then used to perform a variety of other tasks in the ship design process, ranging from estimating fuel consumption to sizing electrical components. An example of these different tasks, and the sections of DDS 310-1 they are defined in, is presented in Fig. 6-5.

The highlighted section of Fig. 6-5 contains the the three methods currently used for calculating demand power. Each of these methods has benefits and drawbacks, and will be further described in the following sections.

6.2.1 Load Factor Analysis

Load factor analysis has historically been used to calculate the ship's demand power. This method assumes that each component onboard a ship is small relative to both the prime mover's rating and the operating load of the ship. In this method of performing an EPLA, each onboard component is assigned a load factor for a given set of conditions. The load factor represents the long-term average operating power level as a fraction of the component's rated load. To calculate a load factor, one must estimate the fraction of the operating time (FOP) that a system will be in operation and the average power the component will draw when in operation (P_{avg}). The load

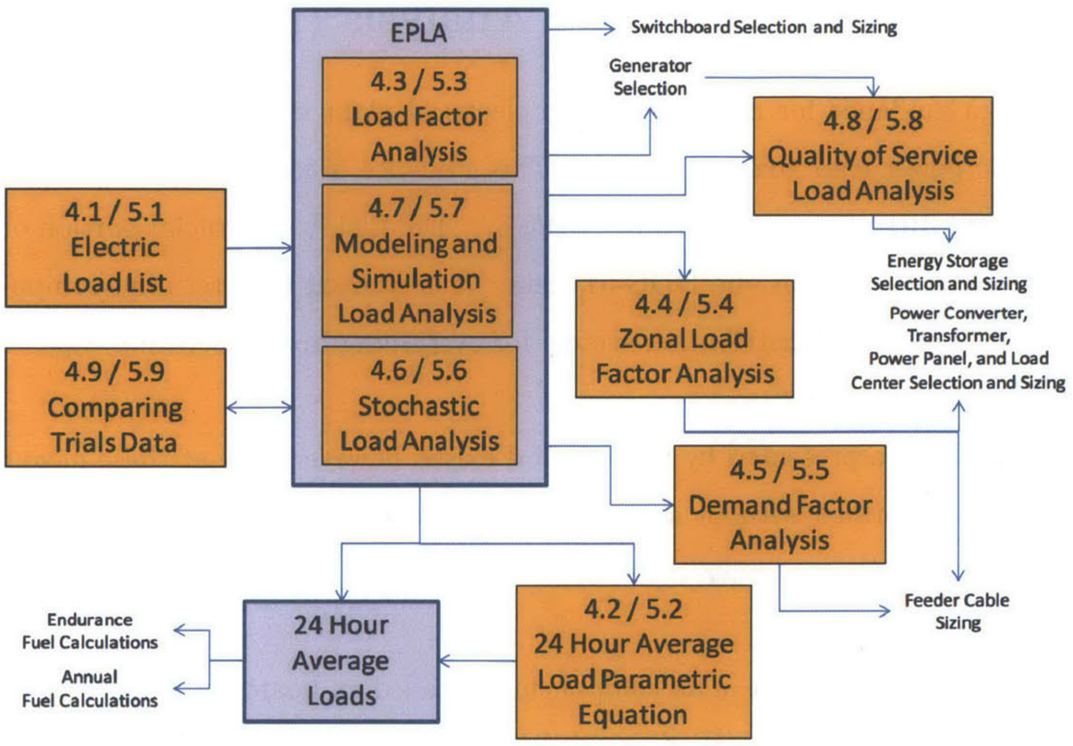

EPLA
4.3 / 5.3 Load Factor Analysis
4.7 / 5.7 Modeling and Simulation Load Analysis
4.6 / 5.6 Stochastic Load Analysis
4.1 / 5.1 Electric Load List
4.9 / 5.9 Comparing Trials Data
Switchboard Selection and Sizing
Generator Selection
4.8 / 5.8 Quality of Service Load Analysis
Energy Storage Selection and Sizing
Power Converter, Transformer, Power Panel, and Load Center Selection and Sizing
4.4 / 5.4 Zonal Load Factor Analysis
4.5 / 5.5 Demand Factor Analysis
Feeder Cable Sizing
24 Hour Average Loads
4.2 / 5.2 24 Hour Average Load Parametric Equation
Endurance Fuel Calculations
Annual Fuel Calculations

Fig. 6-5: Inter-relationship of DDS 310-1 Tasks [51].

factor (LF) will be the product of these values, as shown in (6.1).

$$LF = FOP \cdot P_{avg}. \tag{6.1}$$

When the load factor for a given component is being calculated, it is important to note that most components will not operate at rated load. Since most pumps and motors are purchased in standard sizes and not custom designed for a given purpose, these items are often oversized for a specific use. As a result, components will typically not operate at full power even when a system is operating at maximum design loading conditions. Determining the expected operational fraction or average power consumption might be performed using historical data from similar types of ships or systems or by using manufacturer design information such as pump curves. When this information is not available, standard estimated values for load factors are provided in DDS 310-1.

The above description for a load factor must be applied to a variety of operating states for the ship. The four possible conditions that the ship to consider when performing an EPLA are shore, anchor, cruising, and functional. The shore condition is the period of time when the ship is in port and shut down and electrical power is supplied by a tender or shore power. Anchor refers to the condition where the ship is moored or anchored, but supplying its own electrical power. Cruising is the condition in which the ship is sailing with defense capability, but is not engaged in its functional mission. The functional condition refers to the period of time when the ship is performing its primary mission. For surface combatants (including the DDG-51), this would be a period of battle though for other ship classes it varies based on the functional mission of the ship. The additional condition of emergency may be included for a ship possessing separate emergency and ship's service electrical generation (i.e. a steam-powered ship), and would describe the state where the ship is underway operating only on the emergency generator. This does not apply to the DDG-51 class, and will not be discussed further in this thesis.

For each of these operating conditions, the ambient external conditions must also be evaluated. This is performed by assigning a load factor for the summer and winter seasons under each of the four conditions discussed above. For some systems, there may be little variation based on the environmental state. System components involved with heating and cooling are most impacted by these external effects.

For each component on the electric load list, the load factors are tabulated and then used to compute a calculated load for each condition. The calculated load P_{calc} is the product of the load factor LF and the rated load P_{rated} as shown in (6.2).

$$P_{calc} = LF \cdot P_{Rated}. \tag{6.2}$$

An example of the tabulated results for two nominal components is given in Table 6.2. In this example, a pump is envisioned that operates infrequently while moored, but at increasing rates when in the cruising and functional condition. A ventilation heater is also represented, but is assumed dependent only on the ambient season, not on the operating condition.

To complete the load factor analysis method of the EPLA, all components on-

Table 6.2: Load Factor Analysis Calculation

Component	Rating		Shore				Anchor			
			Summer		Winter		Summer		Winter	
	kW	Hp	LF	Load	LF	Load	LF	Load	LF	Load
Pump	65	50	0.1	6.5	0.1	6.5	0.1	6.5	0.1	6.5
Ventilation Heater	50	-	0	0	0.5	25	0	0	0.5	25

Component	Rating		Cruising				Functional			
			Summer		Winter		Summer		Winter	
	kW	Hp	LF	Load	LF	Load	LF	Load	LF	Load
Pump	65	50	0.6	39	0.6	39	0.9	58.5	0.9	58.5
Ventilation Heater	50	-	0	0	0.5	25	0	0	0.5	25

board the ship would be included in tables such as this. The columns are then summed to generate the total expected load on-board the ship in each operating condition and ambient state for the ship, resulting in eight different EPLA load estimates for the ship. Tabulated estimates for load in specific load centers or switchboards could be performed in a similar manner to aid in sizing these portions of the electrical distribution system.

While there are several benefits of using load factor analysis, there are also limits to its utility. It is a straightforward way to estimate the electrical demand on the ship, and by using historically defined load factors, this estimate can be quickly achieved. For many applications, such as defining the 24-hour average electrical demand to determine annual fuel consumption, this method may provide a reasonable estimate. The load factor analysis does not give indications of maximum expected peak power demand, however, as it creates long-term averages instead of loading at specific points in time. To compensate for these effects, the use of electrical margins must be incorporated into a design to ensure the generation capacity is not undersized.

6.2.2 Stochastic Load Analysis

The stochastic load analysis is an alternative method provided in DDS 310-1 for estimating the demand power on-board a ship in the design process. This method assumes that for each component a probability distribution function (PDF) and an associated cumulative distribution (CDF) for electrical loading can be determined or estimated. Examples of the three most typical distributions expected for shipboard modeling are the uniform, triangular, and discrete distributions, shown in Figs. 6-6, 6-7, and 6-8, respectively. Care must be chosen when implementing a distribution to ensure it reflects the actual loading conditions and real-world possibilities. Using a distribution with infinite tail sections, such as a normal distribution, could create negative loading conditions, and must be judiciously applied.

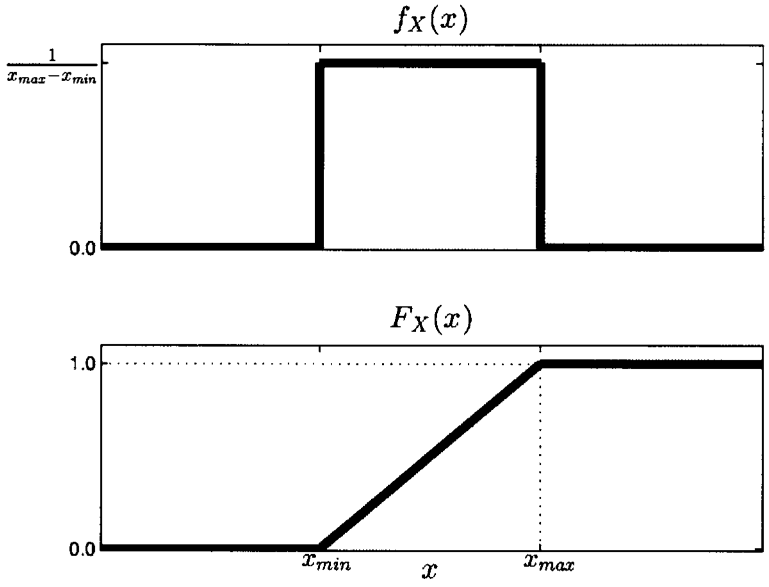


Fig. 6-6: Uniform Distribution PDF and CDF [51].

From these distributions, estimations can take place to match a PDF and CDF to each individual system or component. With knowledge about how a system is expected to operate, one could estimate the mean in the triangular distribution, for example, and include a region of distribution around this point. Proper selection of the distribution allows the ship designer to incorporate variation directly into the model, instead of relying on margins to deal with uncertainty.

Once each load has been assigned a distribution for each condition, a simulation process must be completed to create a total overall loading profile. The method prescribed in DDS 310-1 as the most common method is the Monte Carlo simulation. In this method, random variables represent an input to each component, assigning a

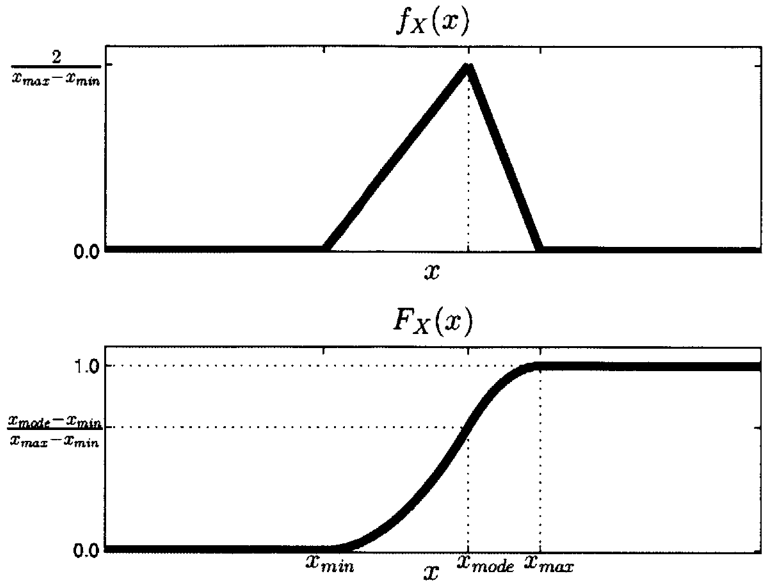


Fig. 6-7: Triangular Distribution PDF and CDF [51].

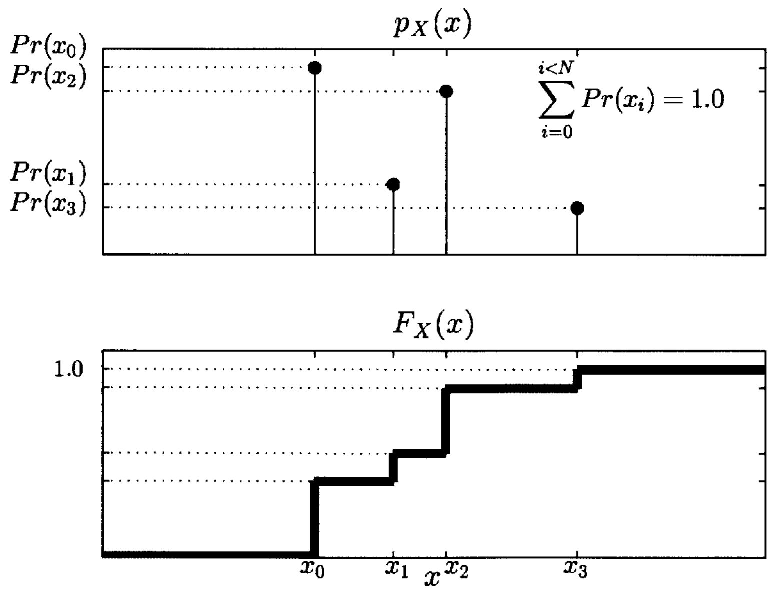
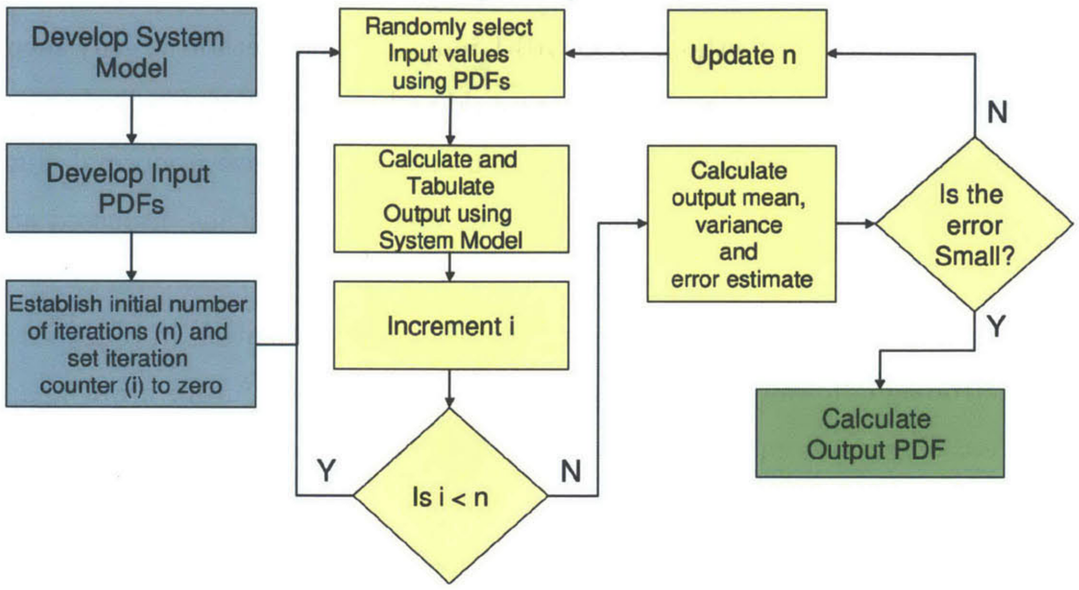


Fig. 6-8: Discrete Distribution PDF and CDF [51].

load based on the distribution. The total load is a summation of the loading of each component. This simulation is then run for a large sample group to determine relevant output statistics. An example of the Monte Carlo simulation method is shown in Fig. 6-9.



Develop System Model
Develop Input PDFs
Establish initial number of iterations (n) and set iteration counter (i) to zero
Randomly select Input values using PDFs
Calculate and Tabulate Output using System Model
Increment i
Is i < n
Y
N
Update n
Calculate output mean, variance and error estimate
Is the error Small?
N
Y
Calculate Output PDF

Fig. 6-9: Monte Carlo Simulation Algorithm [51].

Using this method to quantify electrical loading conditions is appealing as once the distributions are estimated, it is relatively straightforward to develop output statistics. The relationships between load and output are not based on first principles, therefore the method is computationally efficient and does not require solving equations of state.

Stochastic analysis provides more robust output results than the simpler load factor analysis. For each loading condition of the ship, there will not only be an estimated mean value for the load, but a standard deviation. Understanding the range of expected loading provides improved information for sizing the electrical distribution. The EPLA produced through stochastic analysis still does not predict how the actual loading will occur over time, but provides better insight into the characteristics of the electrical plant.

6.2.3 Modeling and Simulation

Modeling and simulation is the most complex of the three methods for performing an EPLA, as it requires defining how components behave over time and interact with other components. As a result of the significant investment required to perform the modeling and simulation method, DDS 310-1 indicates this method is only invoked when specifically required. Such examples would include modeling components that are large relative to generation capacity, when components have abnormal characteristics, or when the components cannot be modeled by the means discussed above [51].

One example of a component requiring modeling and simulation in today's Navy is the electromagnetic rail gun currently being developed as a future naval weapon. When fired, these systems have large pulse loads that could have large transient effects on the electrical distribution system. In studies of future rail gun technology, the design has been focused on achieving a projectile muzzle velocity of 2500 m/s and kinetic energy of 64 MJ [60]. In studies surrounding this design, the thermal and electrical demands of such a system have been analyzed to determine the specific design issues presented to ship integration. An example electrical transient for a railgun from one of these studies is shown in Fig. 6-10. In this example, the rail gun has four track segments that fire sequentially to launch a projectile.

It is important to note the time scale associated with the model, which is on the order of milliseconds. Solving computationally intensive differential equations presents a means of accurately modeling the transient behavior seen as a result of firing the rail gun. This may be required to ensure proper powering and heat rejection for the weapon, but would not be practical as a means of determining the long-term behavior of total ship electrical load.

While modeling and simulation clearly presents the most accurate method of creating a representation of the loading of the electrical plant, it does not lend itself well to being used as an early-stage design tool. In the example of the rail gun, the component is fully defined and allows for a model based on first principles to

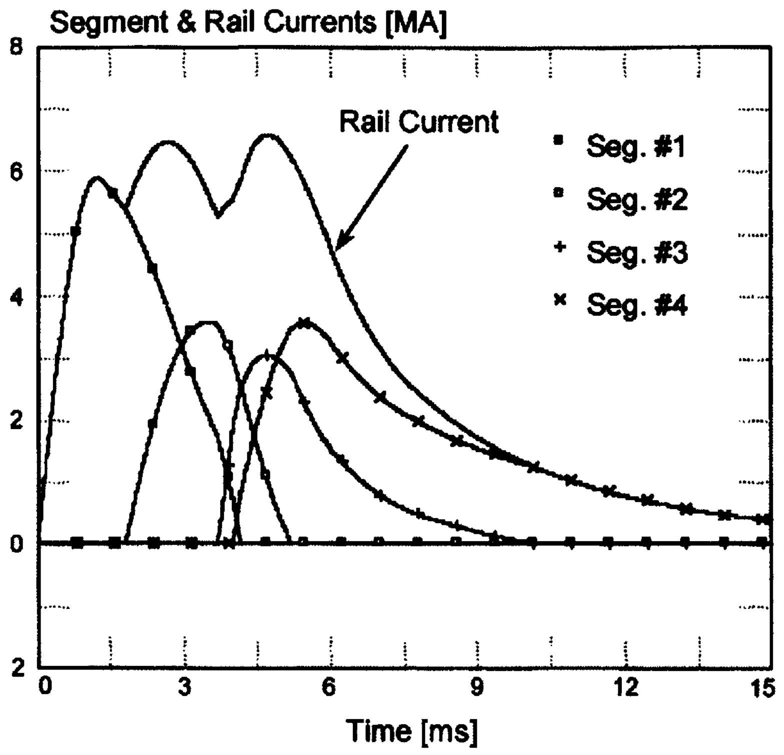


Fig. 6-10: Railgun Electrical Transient [60].

be developed. In many cases the specific loads may change as the design progresses, requiring flexibility on the part of the design tool.

6.2.4 Potential Improvements

As shipboard electrical power demands continue to rise, the need to accurately estimate future consumption will become increasingly important. Understanding how the fleet of today is operating and using this information to develop improved tools for future design work is critical. As an exemplar of a well-documented Navy shipboard power system, the DDG-51 will be used to evaluate current means of determining an EPLA. The following sections of the thesis will begin by discussing the sources of information available for the DDG-51 class and their applicability to this effort. Using these available resources, existing load factors will be updated to reflect current fleet operations. Finally, the framework for a behavioral model as an alternative to current modeling and simulation practices will be developed. The behavioral model combines the computational efficiency of stochastic methods with an understanding of ship behaviors to simulate electrical plant responses based to variations in input parameters. This modeling method would provide increased capability to predict ship behaviors over time and would be an important step towards meeting the power demand challenges presented in future years.

6.3 Sources of Information

Several disparate sources were brought together in this thesis, originating in an effort to update the speed-time profile for the DDG-51 class. Each of these sources will be discussed in greater depth in the following sections of this thesis.

Development of an updated DDG-51 class operating profile required raw operational data, which was gathered from visits to the fleet concentration areas of San Diego, CA, Norfolk, VA, and Pearl Harbor, HI. Individual DDG-51 class ships were visited on these trips, and engineering and deck logs were collected and manually collated to create a database of operational data. Using this database, the amount of time spent at each speed was determined and an updated speed-time profile was created. Additional mission or engineering mode profiles were also developed to assist in understanding the operational behaviors of the ship class.

In an attempt to gather operational data in an electronic form, data from the machinery control message acquisition system (MCMAS) were acquired from the Navy's Ships Systems Engineering Station (SSES). The MCMAS program contains a record of the configuration of the ship's systems over time, which provided a basis for understanding and profiling equipment behavior.

The DDG-51 program office also provided several crucial documents that aided in the development of the concepts seen later in this chapter. These documents included the current references for the DDG-51 class, such as the EPLA and electric plant schematics, but also included a baseline report produced as part of the new construction process for the DDG-111 (USS SPRUANCE) conducted by Alaris Companies, LLC [61, 62]. The baseline report contains the raw data and plots of the operating load for many components onboard the ship, which formed the basis for modeling the electrical profiles in the simulation portion.

6.3.1 Data Collection and Database Development

The primary source for the development of a new operating profile was the ship's logs. The ship's logs are the legal record of the events occurring onboard the vessel, providing an accurate means of garnering the relevant information. The deck log and engineering log were collected from each DDG visited. The deck log provides an accurate reflection of the operations, speed, course, and rudder angle the ship took during a given day. The engineering log provides an account of the status of major engineering plant machinery status over the course of a day. Each of these logs then provided a different element required for the study. The deck logs provided the speed and mission of the ship, while the engineering logs provided the configuration of the propulsion and electrical generation systems onboard.

To gather the required information, visits were performed by Bartholomew Sievenpiper and Katherine Gerhard [63] to the fleet home ports of San Diego, CA, Norfolk, VA, and Pearl Harbor, HI. A summary of the ships visited is shown in Table 6.3.

Table 6.3: DDGs Visited for Data Analysis

Hull	Name	Flight	Home port	Hours of Data
52	USS Barry	I	Norfolk, VA	2184
53	USS John Paul Jones	I	San Diego, CA	2208
59	USS Russell	I	Pearl Harbor, HI	2177
61	USS Ramage	I	Norfolk, VA	2206
70	USS Hopper	I	Pearl Harbor, HI	783
84	USS Bulkeley	IIA	Norfolk, VA	1968
87	USS Mason	IIA	Norfolk, VA	1488
88	USS Preble	IIA	San Diego, CA	2185
90	USS Chafee	IIA	Pearl Harbor, HI	192
91	USS Pinckney	IIA	San Diego, CA	2193
95	USS James E. Williams	IIA	Norfolk, VA	1438
97	USS Halsey	IIA	San Diego, CA	2159
100	USS Kidd	IIA	San Diego, CA	3418
102	USS Sampson	IIA	San Diego, CA	744
103	USS Truxtun	IIA	Norfolk, VA	1407
104	USS Sterett	IIA	San Diego, CA	1464

To ensure that the data were representative of actual ship operations, a three month period of each vessel's data was collected when practical. A longer time period evaluating fewer ships was considered, but this approach was rejected since it could allow an outlying data set to have an undo impact on the final result. These logs represent the widest possible data available, including units with home ports in both the Atlantic and Pacific, units operating both near home port and deployed, and units from different flights of the DDG-51 ship class. The time period for the data collected ranged from September 2011 to August 2012.

Each set of logs was compiled together and included the times of relevant ship status changes. A database of operational data was generated and contained the times of each speed, mission, GTM, and GTG change across all ships included in the study.

6.3.2 Machinery Control Message Acquisition System

The DDG-51 class was designed so that each vessel had a machinery control message acquisition system (MCMAS) onboard at the time of delivery [64]. This system was designed to provide a record of the state of equipment in the engineering plant over time, and provide a means of troubleshooting equipment in the event of plant casualties. In 1996, the integrated condition assessment system (ICAS) was officially developed, and integrated into the existing MCMAS system [64]. The ICAS system provides the foundation of the condition-based assessment program in the US Navy, and considerable resources have been devoted to increasing the number of engineering components monitored.

The data collected onboard by the MCMAS system are stored in a centralized database, located on a dedicated computer terminal in the ship's central control station. The program includes a graphical interface that provides the option to analyze components monitored by the system as either a text file or as a graphical time series. Data are available in a variety of data rates, ranging from 1 to 300 seconds. Most engineering systems are monitored in the MCMAS system, including propulsion, electrical distribution, and auxiliary systems. The indications available depend on the system and can range from the simple on/off status of a pump to very detailed analyses such as discharge pressures or vibrations. The data of particular interest in this thesis are those recorded from the electrical distribution system that monitors current, voltage, frequency and the positions of major breakers.

In an effort to develop the DDG-51 operational profile using MCMAS data, several months of actual data from the USS HALSEY (DDG-97) were obtained from SSES. The months of data obtained correlate with the same time period for which the ship's engineering and deck logs were previously obtained. This correlation was vital for the interpretation of the data within the MCMAS system. Without the ability to understand the functional configuration of the ship at a given time, it is difficult to interpret a series of outputs from the engineering plant.

6.3.3 USS SPRUANCE (DDG-111) Baseline Report

As part of the construction process for the USS SPRUANCE (DDG-111), Alaris Companies, LLC, conducted a study and performed a baseline power analysis for the ship during the performance of builder's trials [61, 62]. The builder's trials are a period of time in which the construction yard takes the ship to sea, testing systems to ensure that thee vessel meets all required specifications. During this time period, a power reading was logged for hundreds of components on the ship, showing the steady state and/or transient behavior depending on the load.

The report and original data used to generate the report were provided by the DDG-51 program office to assist with the performance of the analysis performed in this thesis. The data in this report provide the electrical fingerprint for a given component, exhibiting its electrical operating characteristics. This will be used later in this thesis when evaluating individual component behaviors independently of system behaviors.

In addition to the individual component power traces, the baseline report determined an average operating load, and average annual power consumption for the ship. This average load is useful in understanding how the power consumption actually seen for the DDG-111 compares to the data predicted in the EPLA for the DDG-51 class.

6.3.4 Nonintrusive Load Monitoring

Extensive research has been done to explore the use of nonintrusive load monitoring (NILM) power monitoring for potential uses ranging from supervisory control [65] to diagnostic monitoring [4, 44, 45, 48] to monitoring aggregate shipboard load [66]. By monitoring the current in each phase and the system voltage, the NILM device uses computational methods to determine the power signature for the monitored system. For more complex systems, such as power panels with multiple loads, the NILM uses frequency analysis to disaggregate composite loads into individual components [65].

To facilitate the testing and operational implementation of future technologies the Navy maintains a land-based engineering site (LBES) in Philadelphia, PA. This location has an operational model of the Main Engine Room #2 of a DDG-51 class engineering plant. The LBES includes the propulsion and electrical generating equipment, including GTM, GTG, a main reduction gear (MRG), and shafting to mimic the operational vessel [64]. In several previous academic efforts, a NILM device has been used at the LBES to demonstrate the ability to monitor loads and perform supervisory functions for the engineering plant [5, 65, 67]. Plant components were monitored with a NILM, and potential shipboard applications were developed. For the purposes of this thesis, the data collected by the NILM devices provide a series of dynamic power traces with potential applicability in a modeling approach to system design. For future generations of ships, locations such as LBES could provide a means of recording power traces for developing power traces for predictive models before the ships are constructed. An example of the work performed in [65] is given in Fig. 6-11. In this example, NILM power monitoring is being performed on 2A fuel oil service pump (FOSP) during the transition in turning on 2B FOSP and securing the 2A FOSP.

By utilizing power traces from actual plant components at the LBES, another data set for the DDG-51 class for power transients over time becomes available for comparison. As shown in the above figure, many of these data sets include high fidelity images of plant transients, which could be used to develop system interrelations when

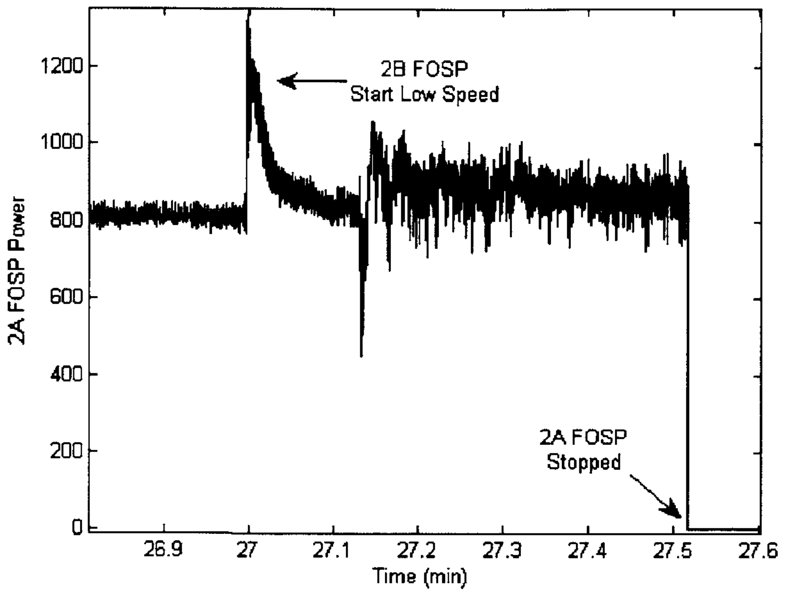


Fig. 6-11: NILM Power Trace of LBES 2A FOSP [65].

developing system models.

6.4 Updating Existing Methods

Bringing together the diverse data shown in §6.3 of this thesis inspired investigation into its applicability to current design methods. The data have relevance in the design community as the standard load factors in DDS 310-1 were not updated in its most recent revision [50]. Since the load factor analysis has historically been the means of producing an EPLA, it is important that these values be as accurate as possible and reflecting the actual behavior in the fleet. Profiles for systems and their related components will be developed using both MCMAS data and information collected from the fleet operating profile.

Additionally, it is desirable to understand how easily power probability distributions could be constructed using the data developed above. Developing relevant power distributions is critical for the implementation of the stochastic load analysis methods of performing an EPLA.

6.4.1 Updating Load Factors

As introduced in §6.2.1, the load factor analysis method of performing the EPLA is the most straightforward method available. Using load factors for components directly from DDS 310-1 [51] or developed based on expected system behavior, the power consumption in a given ship condition can be readily predicted. There is room for improvement within this process, however, as available fleet data are currently not being used to improve understanding of system behaviors. Through the use of MCMAS, an improved operating profile, and actual ship power traces, improvements could be made to the assumed load factors for next-generation ship design.

It is important to recall that there are two pieces to a load factor: the load utilization and the power trace for the load. The load utilization of a component refers to the length of time a component operates, on average, in a given period of time. The power trace for a component is the actual power demand on the electric plant due to the operations for a given load over a given time. The manufacturer label plate data gives a rated load for a given motor or component, but many systems are constructed with motors larger than that which is required to fully operate the system. The load factor is then the product of the utilization and the power consumption in operations, and understanding each side of this equation can yield process improvements.

Load utilization would be expected to be a more consistent parameter than the power trace of a component. Since system behaviors are often defined by standard operating procedures or operational demand, these values would be expected to differ less between different classes of ships. If the load utilizations were known, the most conservative assumption when a power trace is not available would be to establish the load factor as equal to the load utilization.

Load Utilization Determination

Determining a load utilization profile for a component, it decouples system operations from specific component selection. While components in a system might change in a future ship design, understanding system behavior allows predicting load factors for future designs. By applying the expected power consumption to a known load utilization, a future load factor could be readily extrapolated for a next-generation system.

MCMAS Load Utilization

A starting point for examining how systems and components operate is through MCMAS. This program allows individual ship systems and component states to be tracked over a time series, which can determine how components are actually being used. As an example, a time series plot for the #2 lube oil purifier over the course of 23 days in February 2012 onboard a deployed DDG is shown in Fig. 6-12. The lube oil purification system uses differential pressure from the reduction gear lube oil system to recirculate the operating fluid through a purifier. When running, the purifier is electrical powered to remove sediment through a centrifugal separator.

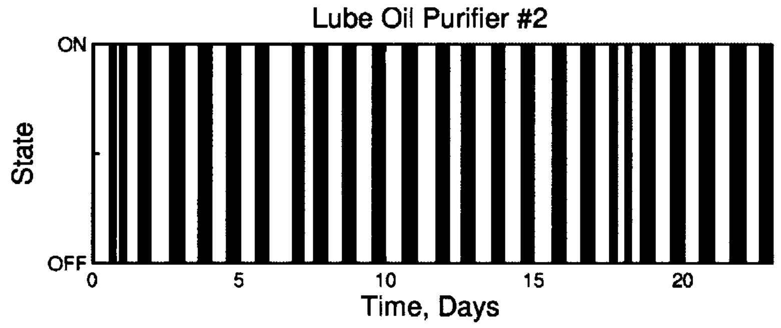


Fig. 6-12: Operational Time Series for Lube Oil Purifier #2

At the simplest level, the operations of the lube oil purifier utilization rate can be expressed simply as the fraction of time the purifier operates to the total time period. In the above demonstration, this would relate to a utilization rate of 0.43 for the lube oil purifier. Direct analysis of long periods of operating data also have the convenience that statistical periodicity of the operating cycle can be established and could be of further use for methods of stochastic load analysis or modeling and simulation. It is also important to note that the purifier has a subordinate component, the purifier lube oil heater, which operates when the purifier is in operation. This device operates thermostatically to control a desired temperature in the incoming oil.

The lube oil heater would therefore also have the load utilization rate of 0.43, since it is in operation when the purifier is running.

The next system considered was the fuel oil transfer and purification system. This system allows for transfer of fuel between storage tanks and service tanks or the recirculation of a tank for purification. Plots for the transfer pump and purifier are shown in Fig. 6-13.

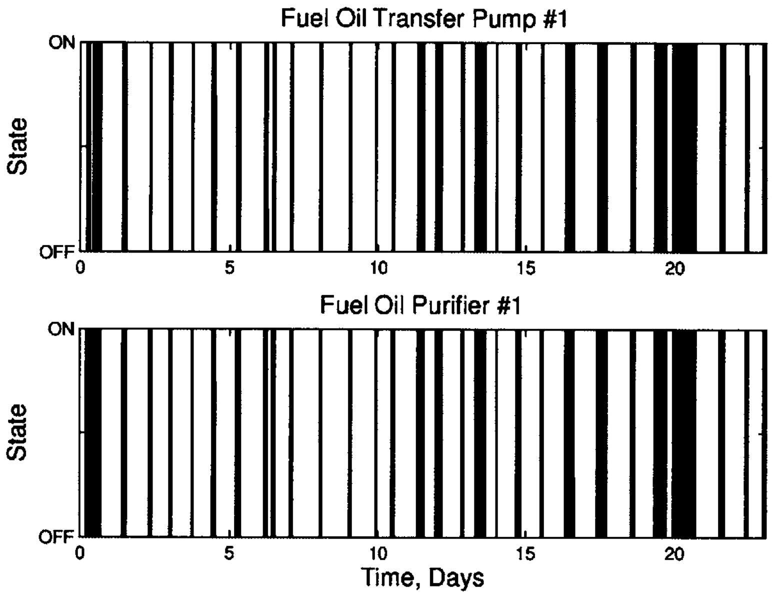


Fig. 6-13: Fuel Oil Transfer Pump and Purifier Operation

It becomes immediately evident analyzing the operation of the system that these components are operated simultaneously; for each operation of the fuel transfer pump, a corresponding operation of the purifier occurs. In this case, these systems operate with a utilization rate of 0.23. Similar to the lube oil purifier, the fuel oil purifier heater is utilized in conjunction with the purifier and is assigned a utilization rate of 0.23.

Another system displaying a different type of cyclic operation is the potable water pumps. A potable water pump is typically in operation underway, providing sufficient pressure to provide supply water throughout the ship. The system consists of two 100 gpm (gallon per minute) pumps, with normal operation consisting of one pump in operation and the other in standby. If demand causes system pressure to drop to a lower setpoint level, the standby pump will turn on to increase supply. Once system pressure reaches a higher setpoint level, the standby pump will secure. The recorded operation over time for these pumps is demonstrated in Fig. 6-14.

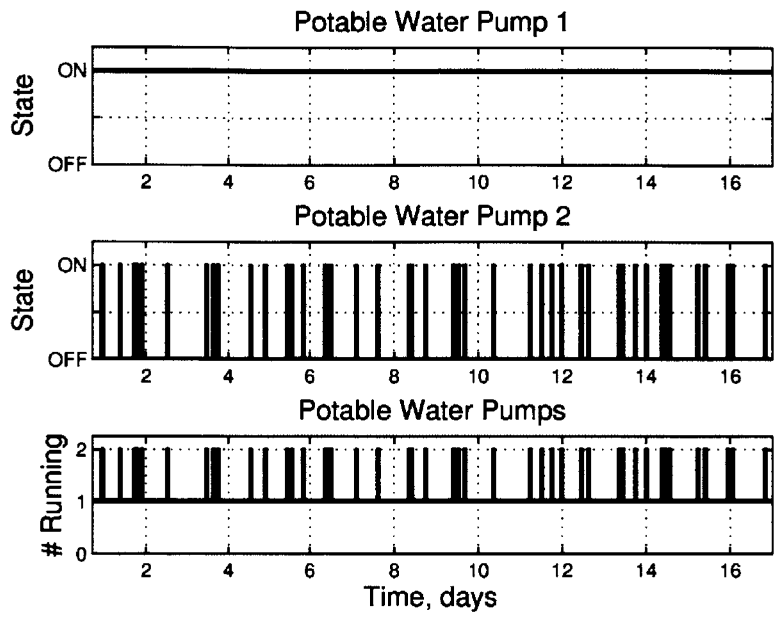


Fig. 6-14: Operation of Potable Water System

In this system, the behavior exists as desired. One pump runs continuously while the other pump runs for short periods of time. Evaluating the running time, however, shows that the running period is minimal for the standby pump. In a 16 day period, only 62 minutes of running operations for the standby pump was observed. This results in a utilization of 0.0027, which is essentially zero. The system overall then has a utilization of 1 for the first potable water pump and 0 for the standby pump.

The MCMAS system can also be used to validate the operating history of equipment systems. With many systems, such as the fire main, there are multiple redundant pumps onboard the ship, with only a subset running at any given time. These systems tend to run for long courses of time without configuration change. A plot of the behavior they exhibit can be seen in Fig. 6-15.

In this profile, it can be seen that the expected condition onboard the ship at any given time would be two of six fire pumps and three of five sea water service pumps. Understanding the periodicity with which these systems reconfigure is relatively unimportant for the development of load factors, as the vast majority of the

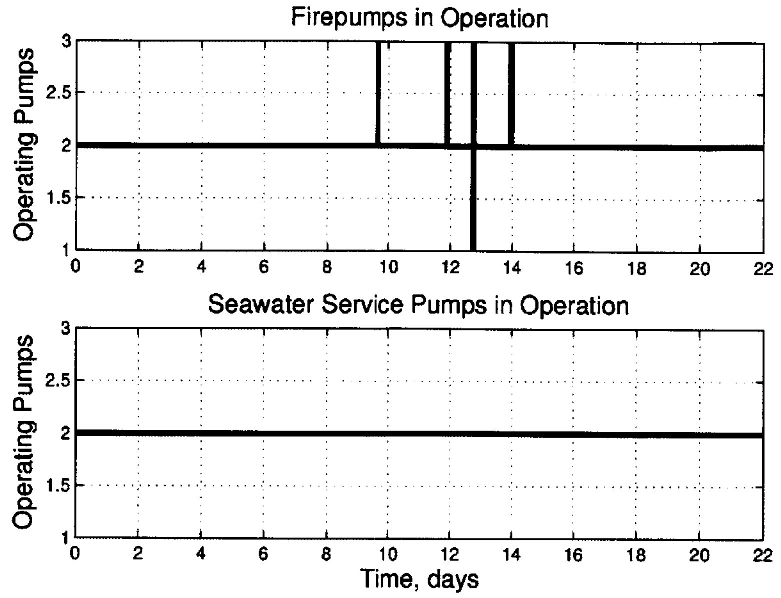


Fig. 6-15: Fire and Seawater Service Pump Operating Profile

time a set number of pumps is in operation. For later development of behavioral modeling, however, understanding the manner by which the ship is operated becomes important. Temporary spikes can be seen in the operating number of running fire pumps, as a result of switching transients. A close up of the transient behavior exhibited by the fire pumps during day 12 of Fig. 6-15 is shown in Fig. 6-16.

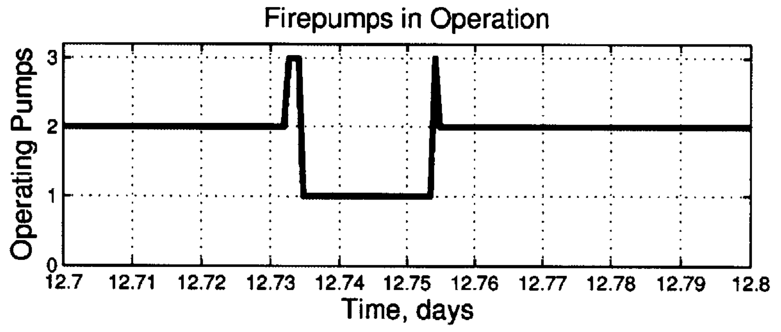


Fig. 6-16: Detailed View of Fire Pump Switching Transient

The transient shown in Fig. 6-16 demonstrates several characteristics that are expected and others that are not. For most fluid systems, a pump switching evolution would bring an additional pump online prior to securing a running pump. It is unknown, however, what caused the drop to a single online pump in this figure. Infrequent conditions, such as equipment casualties or ship drill periods, could cause temporary configuration changes out of the ordinary. These minor deviations have little relevance when determining the long-term operating profile for a system.

The air-conditioning plants, which are not pictured but are discussed in depth later in this thesis, operate three of five plants during summer cruise conditions. In some cases, this fact can be used as validation for the operating state assumed in the EPLA. For the DDG-51 class, the predicted value for fire pumps in normal cruising condition is one of six in operation. Validation is important not just for improving current load analysis, but also to provide feedback to ship designers about how the fleet is utilizing the systems.

A summary of the load utilizations determined using the MCMAS program is shown in Table 6.4.

The data in this table show load utilizations using the data collected from a single vessel. It should be noted that a more diverse data set would provide more accurate results. This data does, however, demonstrate the relative ease with which simple data mining tools could turn unused fleet information into results that would be pertinent to the design community.

Fleet Profile Load Utilization

A second approach to determining the load utilization rate of a component is by considering its contribution to a system with a defined ship profile. Through knowledge of the engineering plant, load utilization factors could be determined through the DDG-51 class operating profile level. This method has the added benefit of using a large and proven data taken across a large subset of the fleet, though a somewhat more limited applicability.

Guidance for the development of load factors directs that standby or redundant equipment is zero unless the equipment is actually running concurrently [51]. As a result, the loading will be considered as distributed evenly over the 1A and 2A engines with a contribution from the 1B and 2B engines only when the ship is in a full power operational mode. While this is not reflective of the actual plant operations, it ensures that the long-term average loading for that portion (forward or aft main machinery

Table 6.4: MCMAS Derived Load Utilization Rates

Component	Load Utilization Rate
Lube Oil Purifier (#1 & #2)	0.43
Lube Oil Purifier Heater (#1 & #2)	0.43
Fuel Oil Purifier (#1 & #2)	0.23
Fuel Oil Purifier Heater (#1 & #2)	0.23
Potable Water Pump 1	1.00
Potable Water Pump 2	0.00
Fire Pump 1	1.00
Fire Pump 2	0.00
Fire Pump 3	0.00
Fire Pump 4	1.00
Fire Pump 5	0.00
Fire Pump 6	0.00
Sea Water Service Pump 1	1.00
Sea Water Service Pump 2	0.00
Sea Water Service Pump 3	1.00
Sea Water Service Pump 4	0.00
Sea Water Service Pump 5	1.00
AC Compressor 1	1.00
AC Chilled Water Pump 1	1.00
AC Compressor 2	0.00
AC Chilled Water Pump 2	0.00
AC Compressor 3	1.00
AC Chilled Water Pump 3	1.00
AC Compressor 4	0.00
AC Chilled Water Pump 4	0.00
AC Compressor 5	1.00
AC Chilled Water Pump 5	1.00

room) is correct. These values can be seen as calculated in Table 6.5.

Table 6.5: Operating Mode Determination of GTM Utilization

Operating Mode	% of Time in Mode	1A	1B	2A	2B
Full Power	6	0.06	0.06	0.06	0.06
Split Plant	24	0.24	0	0.24	0
Trail Shaft	70	0.35	0	0.35	0
TOTAL		0.65	0.06	0.65	0.06

This indicates that in each plant over a long time history there will be approximately 65 hours where at least one GTM is running, and 6 hours when both GTM are running. Both of these values will become important when one studies the interrelation of the operations of the GTM and the equipment that operates in conjunction with it. One could simply roll the operating time of the 1B and 2B engines into the values for 1A and 2A (making each a 0.71 rate), however this would ignore certain dependencies that exist within the data.

GTM Services Each GTM has two major direct support components that it depends on to operate properly, the cooling fan and enclosure heater. The first is the GTM cooling fan, which is a relatively large (130 HP) fan that promotes exhaust gas removal and provides enclosure cooling. This fan will, therefore, be running whenever the GTM is in operation. Each GTM enclosure also has a small heating unit (8 kW) that operates to maintain temperature in the enclosure high enough to ensure suitable fuel viscosity for GTM starting. This thermostatically controlled heater is in operation when the unit is shutdown [68].

Fuel Service Pumps Ensuring that all operating turbines (GTM and GTG) have sufficient fuel supply, the fuel oil service pumps are also required for GTM operation. Each plant has two fuel service pumps, one of which supplies fuel pressure and the other acting as a redundant pump. These are 2-speed positive displacement pumps, with a single pump capable of supplying two fully loaded GTG and two fully loaded GTM [69]. Using fuel consumption curves at each speed in each propulsion mode, the speed of the service pump at each ship speed can be calculated. To err conservatively, it is assumed that the pump is also supplying a fully loaded GTG. The fuel consumption curve for a single plant operating is seen in Fig. 6-17. It is important to note that the trail shaft condition assumes one fuel service pump is monitoring the operating plant. At a given ship speed the operating shaft utilizes more fuel, but only one shaft is in operation.

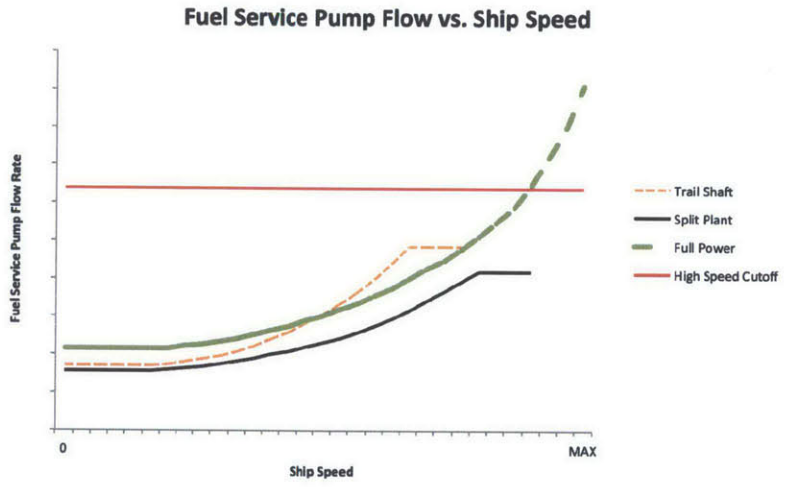


Fig. 6-17: Fuel Service Pump Flow vs. Ship Speed

From these curves, it can be easily seen that very rarely is the ship operating at a speed and propulsion mode that would require a service pump in high speed. The ship spends approximately 6 percent of its operational hours in full power. Furthermore, only 5 percent of operational hours in full power is spent above the speed needed to reach the high-speed fuel service pump. As a result, the approximately 0.3% of the time the ship should reach this level becomes insignificant, and the EPLA should consider the pump only in the slow speed. Since it can be assumed that a GTG will typically be in operation in each plant and a positive displacement pump in half speed operates at approximately half power, the utilization rate of the 1A and 2A fuel oil service pumps should be 0.5. Since they will not run in this scenario, 1B and 2B fuel oil service pumps have a utilization rate of 0.

Main Lube Oil Pumps To transfer the high-speed output of one or both operating GTM in a propulsion plant to the relatively low speed motion of the shaft, gas turbine powered ships utilize main reduction gears (MRG). The MRG require lubricating oil to prevent damage when rotating, which is supplied from the main lube oil pumps to the main lube oil system. There are two two-speed electric pumps, one standby and one automatic backup, for each MRG. There is also a gear-driven pump that is mechanically coupled to the MRG that supplies lube oil flow to the system. In normal operation, one pump would operate in slow speed. If system pressure drops below specification, the pump would ship into high speed. If shaft rotational speed is great enough, the gear-driven pump can provide sufficient pressure, and the electric pump will secure until speed drops.

Using fleet data for the shaft rpm compared to ship speed a plot can be developed to demonstrate the regions in which the electric main lubricating oil (MLO) pump will secure. This cutout speed is shown in Fig. 6-18.

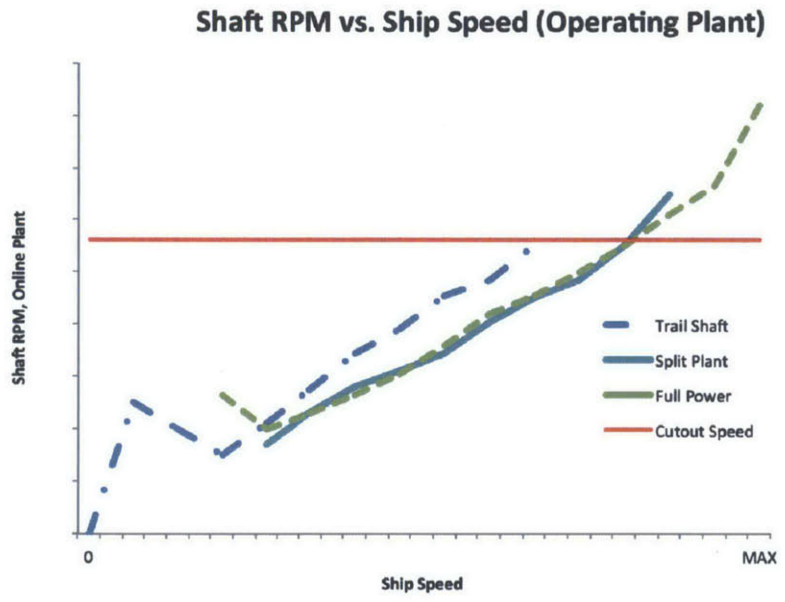


Fig. 6-18: Shaft RPM vs. Ship Operating Speed

From this graph, it can be seen that the electric pump will secure only in the highest ranges of ship speed in a given engineering operating condition. For the trail shaft condition, the operating speed never reaches the required shaft RPM in the operating plant, and the propeller spinning free on the other shaft will obviously not reach this limit. As a result, a MLO pump in each plant will be operating. In the full power condition, the speed will be high enough for the electric pump to cutout 11.2% of operating time, and for the full power condition this will occur 11.9% of operating time. Using the known values for the operating time in each engine configuration, the electric MLO pump would be expected to be operating 96% of the time.

The final piece to defining this load factor is that the electric MLO pump is a 2-speed screw-type pump. The high-speed state is twice that of the low-speed state, and for this style of pump the electrical power consumed for the low-speed state is half that of the high-speed state. Since the pump in normal operating conditions would only operate in low speed, this means it would be operating at half power. Combining this with the amount of time the pump is expected to run, the load utilization for the electric MLO pump is 0.48.

Controllable Pitch Hydraulic Pump There is one controllable pitch propeller (CPP) attached to each shaft, and each propeller has an independent hydraulic system to adjust the pitch of propeller blades. Similar to the main lube oil system, there is one pump driven mechanically by the MRG, and one electric pump. The electric pump maintains hydraulic system pressure when ship speed is low, and will automatically secure when the speed increases above a threshold value. The speeds associated with this transition are the same as those for the MLO pumps, and therefore the same operating rates can be assumed. Since the CPP electric pump is a single speed pump, this indicates an operational utilization rate of 0.96 for the pump in each engineering plant.

GTG Services Typical ship operations for the DDG-51 include the use of 2 GTG to provide electrical power to the ship. Analyzing the DDG-51 class data set for at-sea operations, it is evident that this is a nearly exclusive operating condition for the ship class. This can be seen graphically in Fig. 6-19. The region of single generator operations primarily occurred when maintenance on a running generator was required while another generator was down due to casualty. The region of 3 generator operations, or full power, was a transient condition during generator switching operations.

For the generator services required by a GTG, the fuel service pump was con-

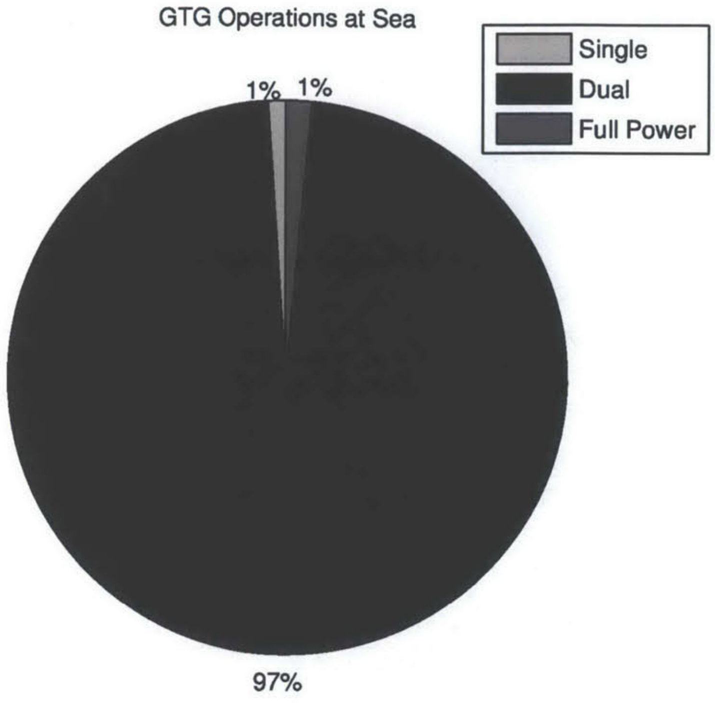


Fig. 6-19: GTG At-Sea Operations

sidered in conjunction with the GTM and will not be considered here. The other required services for GTG operation are the cooling fan, cooling water pump, and enclosure heater. For the GTG cooling fan and cooling water pump, the utilization rate for operating generators should be 1.0, while the heater will have a utilization rate of 1.0 for the secured generator.

Load Utilization Summary The overall load utilization determined using the fleet data discussed above is presented in Table 6.6. It is important to note that when determining the load factor, which is the product of utilization and the fraction of rated power consumed when operating, that the state of the equipment assumed below is understood. For example, it was assumed the MLO electric pump is operating in slow speed, so the fraction of rated power used to get the load factor should be that observed in slow speed operation.

Table 6.6: Summary of Load Utilization for Selected Propulsion Components

Component	Load Utilization Rate
GTM 1A Cooling Fan	0.65
GTM 1B Cooling Fan	0.06
GTM 2A Cooling Fan	0.65
GTM 2B Cooling Fan	0.06
GTM 1A Enclosure Heater	0.35
GTM 1B Enclosure Heater	0.94
GTM 2A Enclosure Heater	0.35
GTM 2B Enclosure Heater	0.94
GTG 1 Cooling Fan	1.00
GTG 2 Cooling Fan	0.00
GTG 3 Cooling Fan	1.00
GTG 1 SW Cooling Pump	1.00
GTG 2 SW Cooling Pump	0.00
GTG 3 SW Cooling Pump	1.00
GTG 1 Enclosure Heater	0.00
GTG 2 Enclosure Heater	1.00
GTG 3 Enclosure Heater	0.00
1A MLO Pump	0.48
1B MLO Pump	0.00
2A MLO Pump	0.48
2B MLO Pump	0.00
CPP Hydraulic Pump 1	0.96
CPP Hydraulic Pump 2	0.96
1A Fuel Oil Service Pump	0.50
1B Fuel Oil Service Pump	0.00
2A Fuel Oil Service Pump	0.50
2B Fuel Oil Service Pump	0.00

Power Requirements

The second half of determining the load factor for a system is defining the power consumption of the component within a system. One important reason to separate the power consumption from the utilization is that the design of the system and selection of components will affect the percent of rated load that a component operates at.

To help illustrate the differences that can arise, Fig. 6-20 depicts the starting transients for the cooling fans from a GTG and GTM. These plots were created using the transient data from the USS SPRUANCE baseline report, then normalizing the data with the EPLA rated load for each fan.

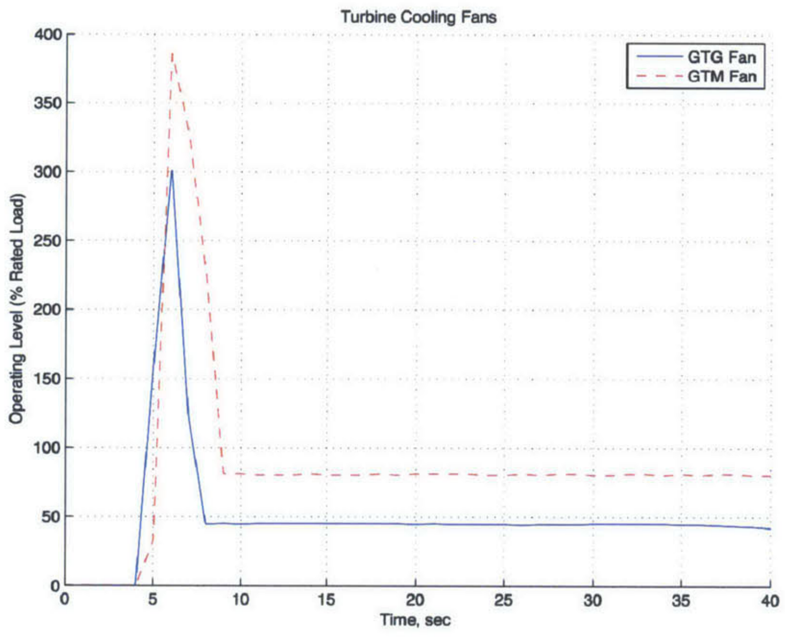


Fig. 6-20: Comparison of Turbine Cooling Fans

It is readily apparent from this plot that there are strong operational differences between the GTG and GTM cooling fan. Following the starting transient, the GTM cooling fan runs at approximately 80% of its rated load, while the GTG cooling fan runs at just under 50% of the rated load.

Many reasons may exist for why there is a discrepancy in operating level between the different types of fans, but this issue presents a clear problem in determining the load factor. Since a component load factor is the product of its utilization rate and its average operating power, the final load factor should change depending on the system design and component selection. If a fan is oversized relative to the requirements of the system, it may operate at a lower percent of rated load but have other selection properties desirable to the ship designer. Additionally, since each GTG has a different intake and exhaust arrangement, it is possible that each GTG cooling fan could have a different steady state operating point.

Load Factor Calculation

As previously described, the overall load factor is a product of the load utilization and power trace. To demonstrate the calculation of a load factor, the lube oil purification system will be examined, which provides a system with two components that exhibit vastly different operating profiles; the lube oil purifier and the purifier heater. The lube oil purifier itself is powered by an induction motor, which spins the purifier. The operation of the purifier is demonstrated in Fig. 6-21.

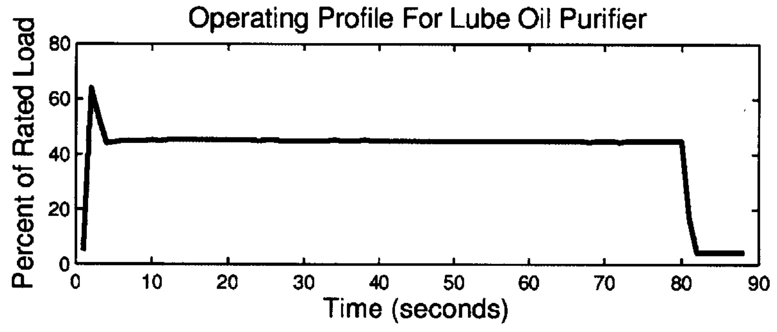


Fig. 6-21: Operation of Lube Oil Purifier

The purifier demonstrates similar operation as the turbine cooling fans seen in Fig. 6-20. The average steady-state operation of the purifier is 44.9% during the analyzed period. The purifier heater, however, demonstrates a much different profile, seen in Fig. 6-22.

The purifier heater operates using a thermostatic controller to heat the incoming oil to the purifier to a certain level. The above profile shows the average operating level, recorded as averaged 30-second increments over several days of operations. The periods of inactivity seen in the graph correlate to periods when the purifier is not in operation. Taking the average operating level of the non-zero elements of the purifier

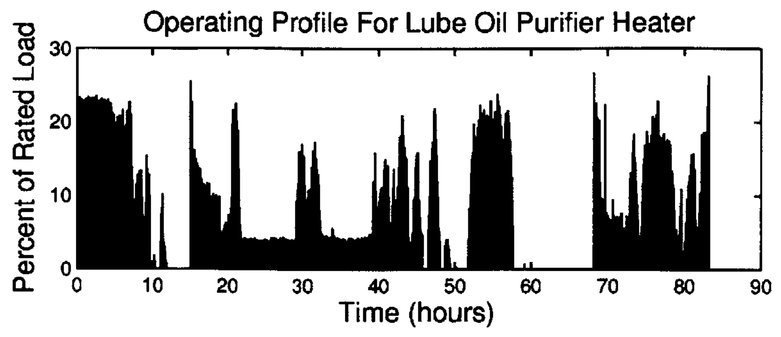


Fig. 6-22: Operation of Purifier Heater

heater's operations, the heater operates at 6.45% operating power.

Combining the information from the power profiles and combining it with the previously determined load utilization for the system, a load factor can be determined. These results are shown in Table 6.7.

Table 6.7: Lube Oil Purification Load Factor

Load	Load Utilization	Average Operational Rate	Calculated Load Factor
Lube Oil Purifier	0.43	44.90%	0.19
Lube Oil Purifier Heater	0.43	6.45%	0.03

The result of 0.19 for the lube oil purifier seems relatively low, but the standard (not ship specific) value given in DDS 310-1 for the purifier is 0.3 [51]. This indicates that while the value has changed, it is not radically different. The load factor for the purifier heater is not specified in the DDS 310-1, but the value of 0.03 seems lower than may be expected. Understanding these differences could be an invaluable portion of post-construction validation of the EPLA in future ship classes. By validating the EPLA the sources of deviation from predicted values could be determined, whether it is from the operating rate or in the load utilization.

6.4.2 Updating Stochastic Load Analysis

The inclusion of stochastic methods into DDS 310-1 [51] occurred in the most recent revision of the design guide. Since it has not been a method used historically to conduct the EPLA, distribution functions are not readily available for shipboard components. Examples of methods to create a PDF for a given load using available data are presented in the following sections.

Developing a Distribution: AC Compressor Motor

Examining the data available within the MCMAS database the air conditioning (AC) compressor stood out, as a result of the system interface between the MCMAS system and the AC plant. A detailed description of the AC plants in the DDG-51 is found in §6.7.2. There are 5 AC plants onboard the DDG-51 class, labeled 1, 1A, 2, 3, and 4. MCMAS receives a direct reading of motor current, tonnage, and other system parameters at each time step, providing a direct means of determining the distribution of motor current over time. Making an assumption for the approximate power factor for the AC plant, this motor current could be transformed into a kW loading for comparison to the EPLA. Examining several weeks of data from MCMAS, the data were sorted using MATLAB for each AC plant. A distribution over this time period is shown in Fig. 6-23 for each AC unit.

By taking an average of the plants over the time period, a PDF for the operation of a single AC plant was developed, as shown in Fig. 6-24.

This PDF inherently contains two separate pieces of design information required by DDS 310-1, the plant configuration and operating distribution. The plant configuration can be inferred by the amount of time spent with no power, which occurs approximately 40% of the time. In the normal operating configuration, 3 of 5 AC plants are operating onboard the ship. The other piece of information available from this PDF is the running distribution of power. With this known distribution, a triangular or normal distribution could be fit to the data set for the purpose of stochastic modeling.

The PDF demonstrated in Fig. 6-24 shows a relatively straightforward means of developing a distribution. It would be expected that using this PDF to perform a stochastic analysis, the average total power consumption among all AC plants would appear similar to the actual distribution seen in Fig. 6-25.

It is important to note in this figure the long tail to the right hand side of the

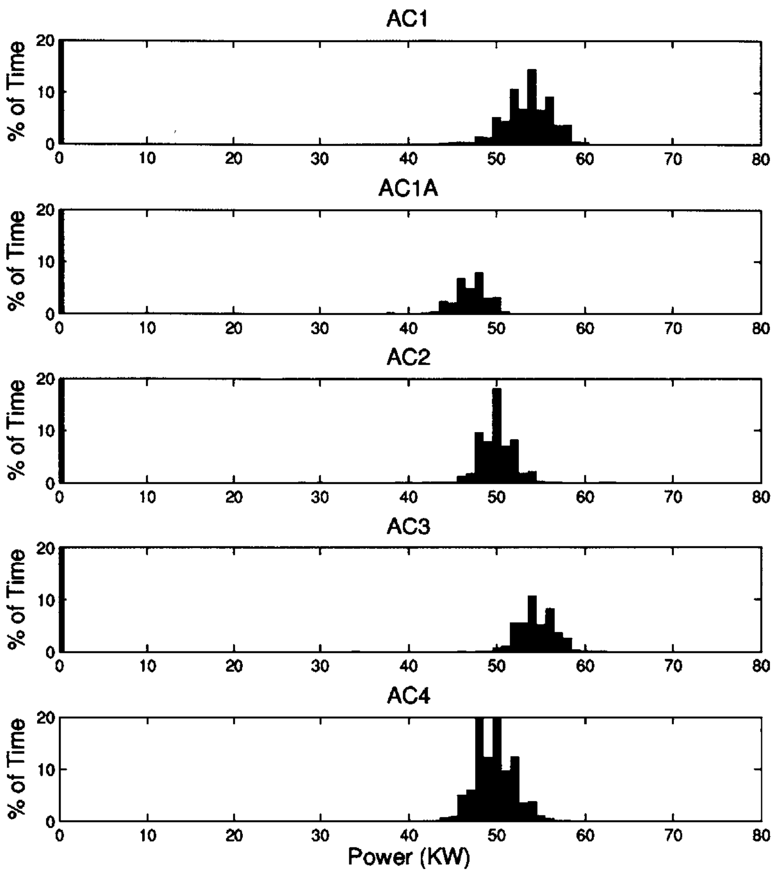


Fig. 6-23: Individual AC Compressor Power Distributions

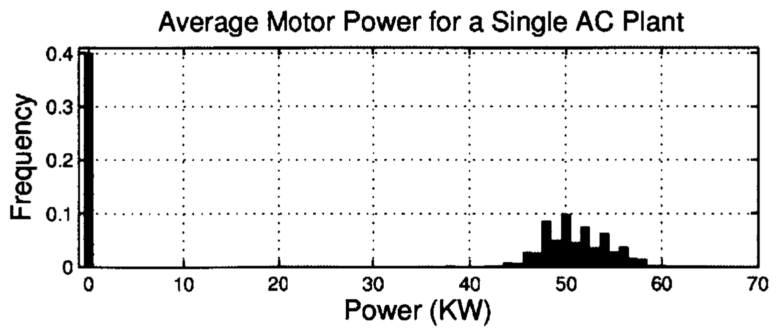


Fig. 6-24: AC Compressor PDF

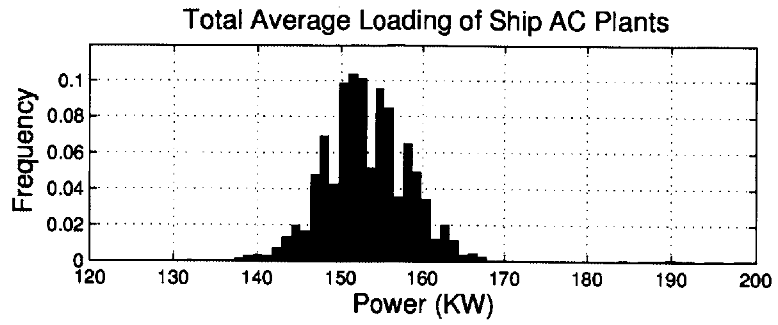


Fig. 6-25: Average Cumulative Loading of all AC Plants

graph (loading seen at approximately 190 kW) that does not exist in Fig. 6-24 for the average AC plant. This tail is representative of the overlap time that exists when switching between AC plants, yielding a temporary condition where 4 AC plants are in operation. For a stochastic analysis of the entire ship to capture the peak loading conditions correctly, seemingly minor details, like infrequent periods operating 4 AC plants, must be included in the stochastic model. A means of correcting this is type of problem is suggested in DDS 310-1, using a two-level model. In this method, the first level would be a variable that would define how many AC plants were running (3 or 4) with specific probabilities associated for each. In the second level, a random variable generated for each operating compressor would generate a simulated load.

Implementing the stochastic method should be able to reproduce distributions that are similar to Fig. 6-25, but an examination of the underlying AC plant data shows the method does not fully capture all effects. A plot of the total load consumed by all AC compressors is shown as a time series in Fig. 6-26.

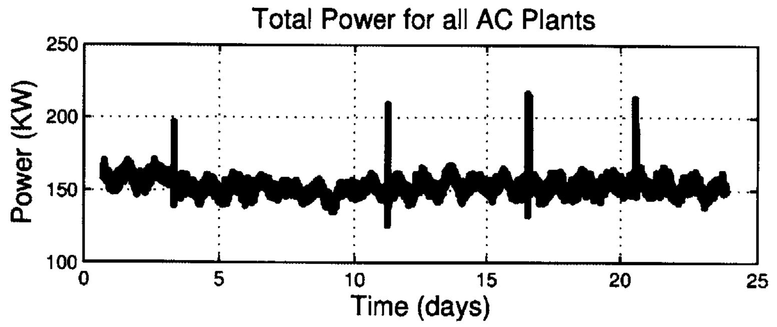


Fig. 6-26: Time Series of AC Plant Cumulative Load

From this time series, it is evident that the loading profile has a couple of notable features. The first feature is that the transient periods of switching AC plants yield power spikes periodically, as discussed previously. The second is that the AC plant loading is heavily diurnal. Over each one-day period, there exists a minimum that occurs in the early portion of the morning and a maximum that occurs in the afternoon.

The diurnal behavior presents an additional difficulty for a stochastic model. Since the AC plants appear to have a time-variant loading profile, assigning random variables to each AC plant would not reflect the true behavior. Rather, it would be expected that when load is high for one running AC plant it would be high for all AC plants. These effects would need to be addressed for the stochastic model to provide realistic simulation results.

Trials Data Stochastic Analysis

A new feature included in the most recent revision of DDS 310-1 is guidance in adjusting an EPLA based on the sea trials data of a newly constructed ship. In addition to providing an update, the component monitoring performed during these trials provides a chance to develop or improve stochastic models. The trials data for the USS SPRUANCE used in this thesis provide an example of how simply this could be performed. Revisiting the operation of the lube oil purifier heater, shown previously as a time series in Fig. 6-22, the heater exhibited a varying level of power consumption over time. This behavior, controlled thermostatically, did not seem to follow any regular pattern. By analyzing the data as a distribution, however, the data can be examined as a PDF, shown in Fig. 6-27.

This PDF demonstrates how readily available data from the construction sea trials for a ship could quickly be used to develop a database of stochastic profiles. Proper planning would be required prior to conducting this analysis to ensure that the data series taken for different components captures the fully variability seen in the load to ensure proper PDF development.

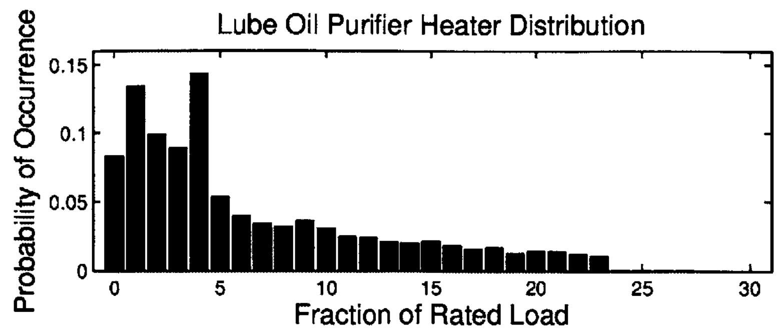


Fig. 6-27: PDF of Lube Oil Purifier Heater Operation

Stochastic Analysis Summary

Stochastic analysis provides a more robust method of determining ship's electrical demand load than load factor analysis when computing an EPLA. Unlike load factor analysis, this method is capable of producing output statistics and developing relevant estimates for parameters such as peak loading. There are potential pitfalls that could occur using this method, however, as shown in the AC compressor example. The following section of this thesis outlines a method of performing behavioral analysis, which is in essence an extension of the stochastic models shown here. The framework of this analysis would utilize system behaviors instead of the random Monte Carlo method to perform the simulation. The implementation for a behavioral model may be different than the stochastic simulation, but the capability to define the relevant statistical information for a system or component remains a vital consideration.

6.5 Behavioral Modeling of DDG-51 Class

The behavioral modeling approach outlined in this section incorporates many of the elements derived in previous sections to define a new method of performing an EPLA. Fundamentally, this approach allows a user to define system responses to global inputs and uses statistical models to predict component electrical demand within the system. This method of performing an EPLA could provide an adaptable model capable of increasing in fidelity as the ship design process progresses.

6.5.1 Introduction and Motivation

Design efforts for US Navy ships typically begin many years before the first ship will be commissioned. The original DDG-51 design efforts, for example, commenced in the late 1970's, the first ship was contracted in fiscal year 1985, and the DDG-51 was commissioned in 1991 [70]. This long timeline creates a problem for the ship designer. Equipment central to a ship's mission may not even exist when aspects of the ship, such as electrical generation capacity, must be determined.

An alternative to the detailed modeling and simulation method of performing an EPLA would be to create models that use simpler methods to provide a realistic time series approximation of data. Unlike the modeling and simulation approach shown previously, this method of modeling would not be based on developing component models from the underlying equations governing their electrical properties. Instead, this method would use statistical quantification of expected loading properties, not unlike those shown in the stochastic simulation example to develop realistic load profiles over time.

To demonstrate the behavioral approach, a model of the 1SA switchboard was developed. This switchboard provides the primary source of power for the forward lower portion of a DDG-51, and its location in the ZEDS can be seen in Fig. 6-4. The 1SA model was jointly constructed with Bartholomew Sievenpiper and Katherine Gerhard.

The goal of this more limited model was to demonstrate the functionality envisioned in a ship-wide behavioral model, examine limitations presented by the method, and compare the accuracy of the model to actual fleet data. Systems were modeled across the entire ship, as interdependencies cannot be truncated to only those components on this switchboard. The resulting load outputs were modeled, however, only for loads connected to the 1SA switchboard. Due to the total number of loads onboard a ship, using the 1SA switchboard limited the data input required to a manageable size while retaining the demonstration capability.

The 1SA switchboard is the largest switchboard on the ship, with more than 20%

of the ship's connected load and more than 25% of the expected operating load in a cruising condition [71]. The 1SA switchboard also represents most of the forward main engine room's source of power, and the engineering plant contains many of the loads most readily modeled. For these reasons, the 1SA switchboard was selected as the focal point for model development.

For real-world comparison, the selection of the 1SA switchboard also has the benefit of being monitored in MCMAS. Referring to the electric plant diagram in Fig. 6-4, it is seen that all power consumed by the 1SA switchboard must pass through either the 1S-1SA or 1SA-2SA breakers. Since readings for the current, voltage, and frequency can be determined from the MCMAS program at these locations the power consumption can be estimated using the average power factor seen at the generators (which have the above indications, as well as power). Graphically, a representation of approximately three weeks of load for these breakers and the 1SA switchboard can be seen in Fig. 6-28.

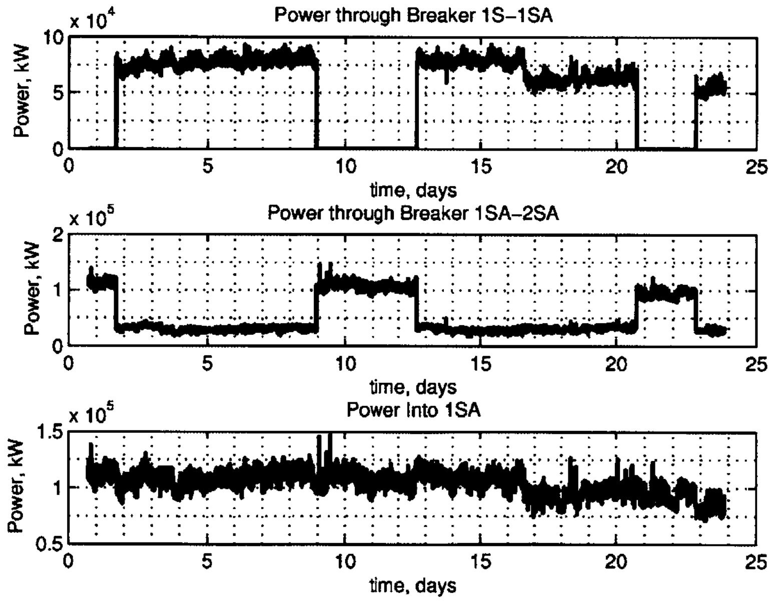


Fig. 6-28: 1SA Loading Profile

The power seen through the 1S-1SA breaker drops to zero during periods when GTG #1 is secured, and based on the changes in power levels when this occurs it can be seen that the power in the switchboard is the summation of the two inputs sensed at the breaker.

6.5.2 Ship Operational Concept

The purpose of a naval vessel is to provide a platform capable of meeting required mission objectives. The ship's mission at a given time will dictate a certain set of operational equipment to be in use, which will directly impact the electrical generation capacity for the vessel. The creation of a fully functional model, therefore, must be responsive to the input signal variability caused by changing operating conditions.

The operating conditions selected for the behavioral model are based on those defined in the DDS 310-1: anchor, shore, cruising, and functional. Based on available data, the cruising operating condition will be evaluated in this model.

It is important to note that engine configuration and ship's speed are coupled. There are certain speeds that cannot be achieved without a specific engine configuration. The overall profile of engine configuration and operating speeds should ultimately reflect the actual operating profile of the ship's if the model is to reflect current fleet performance. For the purpose of the model development, actual data sets from DDG-51 class ships were used for the engine configuration and speed profile to ensure accurate inputs.

For design studies involving future ships the engine configuration, ship speed, and mission should all be reflected through the concept of operations (CONOPS) for the vessel. The CONOPS reflect the expected operational utility the vessel will have within a battle force, and the expected utility it should provide. These documents are developed from the point of view of the individual or organization that will be operating and utilizing the asset.

An example of how the concept of operations is used to develop can be found in examining the concept of operations released in September 2012 by the US Coast Guard for the Offshore Patrol Cutter (OPC). This document was released as part of the request for proposals (RFP) for the design of the next-generation mid-sized cutter in the Coast Guard fleet. This document handles everything from the required operating areas the ship is expected to utilize to the missions it will undertake. The expected mission profile for OPC is shown in Fig. 6-29.

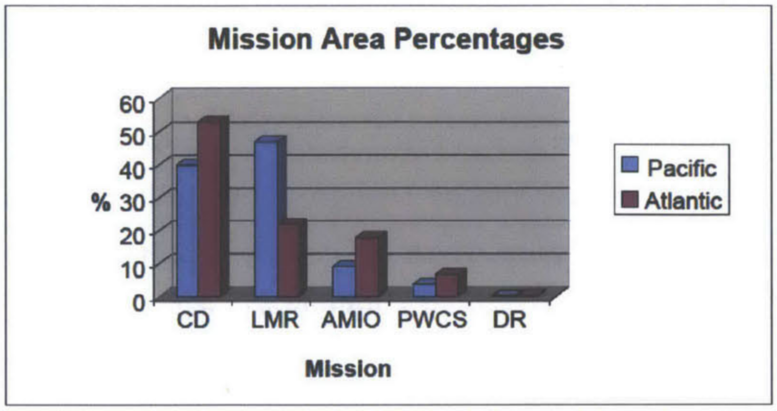


Fig. 6-29: CONOPS Mission Percentages for OPC [72].

In this graphic, the first mission depicted is counter-drug (CD), which is the pursuit of narcotics traffickers in maritime environments. Living marine resource (LMR) is the mission of protecting coastal waters from foreign encroachment and enforcing fisheries laws. Migrant interdiction (AMIO) is the prevention of unauthorized entry of foreign personnel into the country via boat. The ports/ waterways/ coastal security (PWCS) and defense readiness (DR) missions include the potential use of force to protect American assets domestically and abroad. Each of the missions would require a different subset of shipboard equipment to perform the ship's functional duties. When output results are properly tied to input parameters, the ship designer should be able to look at Fig. 6-29 and answer a question such as: "How will predicted electrical loading vary based on the different mix of Atlantic and Pacific mission assignments?"

Beyond missions, the speed-time distribution is documented in the CONOPS for the OPC. As discussed previously, these speed-profiles will directly affect electrical consumption based on dependencies established in the engineering plant. The expected OPC profile is shown in Fig. 6-30.

If one wanted to evaluate the space for propulsion to select between IPS and conventional diesel or gas turbine drive, having an adaptable design tool to model fuel consumption would be needed. Current EPLA methods could have difficulty accounting for electrical propulsion loading effects.

These graphs from the OPC RFP demonstrate a typical series of early-stage defi-

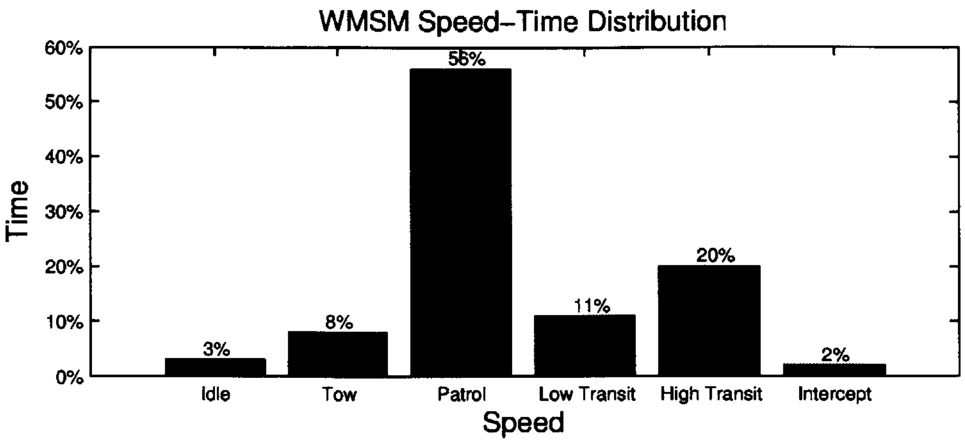


Fig. 6-30: CONOPS Speed-Time Profile for OPC [72].

nitions. They also highlight the types of questions that current EPLA methods do not address. Model development must ensure that future processes can interface cleanly with external input parameters to provide useful results.

6.5.3 Model Framework

To develop a model for electric plant design, the first challenge was to develop an interface that would be robust enough to demonstrate the potential utility of the proposed model. The system was first required to accept representations of the external inputs, which represent the operational profile for the ship. This includes both the direct drivers onboard the ship, such as speed and engine profile, and the external inputs such as ambient temperature. The system characteristics must then be defined for each component in a manner that reflects their dependency on these external inputs. The component responses, and associated electrical signature, must then be integrated into the model to ultimately create a realistic simulation.

In this process, system behaviors are decoupled from electrical responses. An example of how the power simulation is created is shown in Fig. 6-31, which depicts a notional system component. In this example the top image is the square wave operating state for a component that is modeled based on system behaviors. This is the "ones and zeros" representation for the component, while the center image depicts the transient electrical response seen when energizing the component. By superimposing the electrical behavior of the on the square wave profile, an estimated power trace for the component is developed.

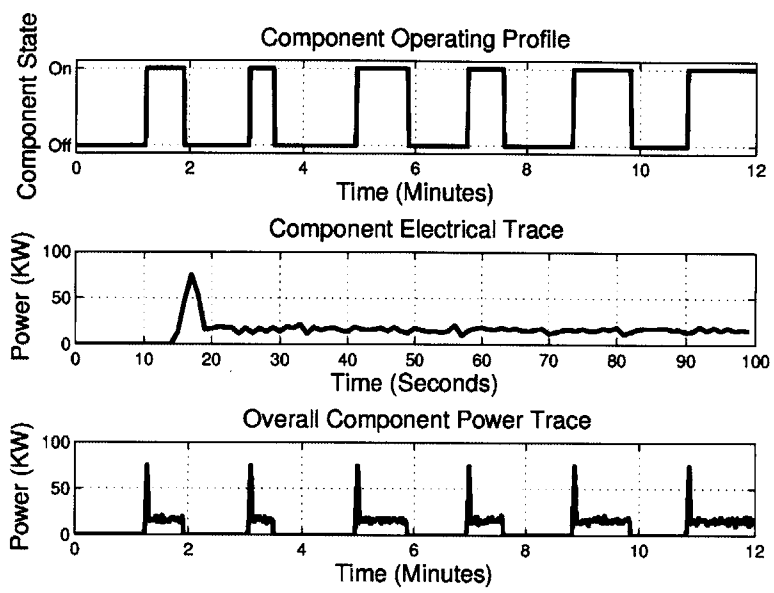


Fig. 6-31: Method of Creating Power Trace

This creates model that is adaptable over time, as information is gained on how either the system behaves or how the component responds improvements can be made independently of other variables.

Overall, the process of developing a model that takes global inputs and produces an electrical model is shown graphically in Fig. 6-32. Each of the steps involved in developing this model will be shown in the following sections.

It is important to note that the model developed ignores an introductory screen in which the basic ship variables required for a model are defined. This screen would allow the ship designer to define the propulsion plant architecture, electrical architecture, and other ship parameters of interest. By using a current ship class, this information is well defined. The major parameters representing this ship information are shown in Table 6.8.

The first variable presented is the ship classification, which in the case of a DDG-51 is a surface combatant. This distinction is important because it defines the functional condition for each ship. For example, the battle condition is functional for a surface combatant but would be air operations for aircraft carriers. It would be expected

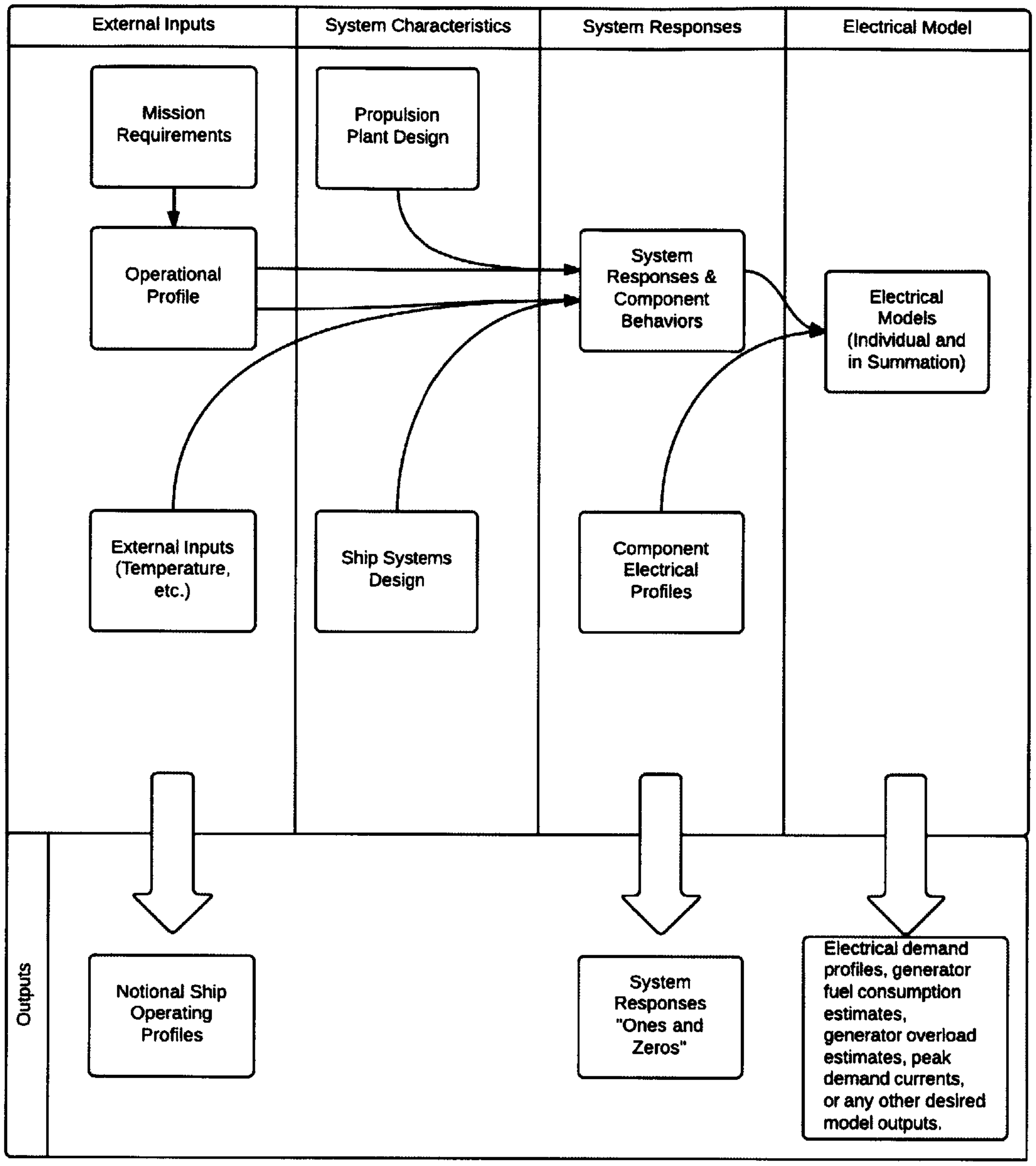

External Inputs
System Characteristics
System Responses
Electrical Model
Mission Requirements
Propulsion Plant Design
Operational Profile
System Responses & Component Behaviors
Electrical Models (Individual and in Summation)
External Inputs (Temperature, etc.)
Ship Systems Design
Component Electrical Profiles
Outputs
Notional Ship Operating Profiles
System Responses "Ones and Zeros"
Electrical demand profiles, generator fuel consumption estimates, generator overload estimates, peak demand currents, or any other desired model outputs.

Fig. 6-32: DDG-51 Model Design Process

Table 6.8: Ship-Type Definitions of a DDG-51

Ship-Type Definitions	
Variable	DDG-51
Ship Classification	Surface Combatant
Propulsion Type	Gas Turbine
# of Shafts	2
Electrical Generation Type	Gas Turbine
Electrical Distribution Type	Zonal
Combat Systems	Aegis
Auxillaries	Defined

that the fraction of time spent in a functional condition for a aircraft carrier would be higher than that of a surface combatant.

The propulsion type selection defines the required systems onboard for the propulsion of the ship. For the DDG-51, the propulsion is achieved through the use of gas turbine engines although consideration has been given to retrofitting these ships with a hybrid drive system. In the case of a hybrid drive, an electric motor is used for lower speeds and the gas turbine engines are brought online to achieve higher speeds. For hybrid drives, the electrical demand on the ship's generators would depend on the speed of the ship. The propulsion type also defines the support systems required to operate the plant such as the number of shafts, or propulsion units, onboard the ship.

Electrical generation type and distribution define the required support systems for electrical generation. It also defines the architecture for the distribution, which determines how power is distributed to individual components. In evaluating the EPLA for a ship, the electrical distribution is key in determining the sizes of the breakers and cables onboard the ship.

6.5.4 Global Inputs

Global inputs, as defined in this thesis, are those parameters external to the ship that drive the behaviors of systems within the ship. The global parameters will not generally change for different ship types, as they impact the ship regardless of the definition of systems onboard. The global inputs evaluated in this model were propulsion plant operating mode, ship speed, time, season, and ambient temperature. For demonstration purposes in this model, the data used was actual operating data for a deployed DDG Flight IIA ship. As a result, no model was created to generate the global values, since the inputs were well defined.

The model developed to demonstrate this concept required a series of inputs prior to running a simulation. The initial screen operated by the user is shown in Fig. 6-33. In this model, three selection buttons are provided on the main screen; one for modifying the environment-type inputs, one for the ship-type inputs, and one for model parameters. Once all selections have been made, provisions are made for the running the simulation and acquiring output results.

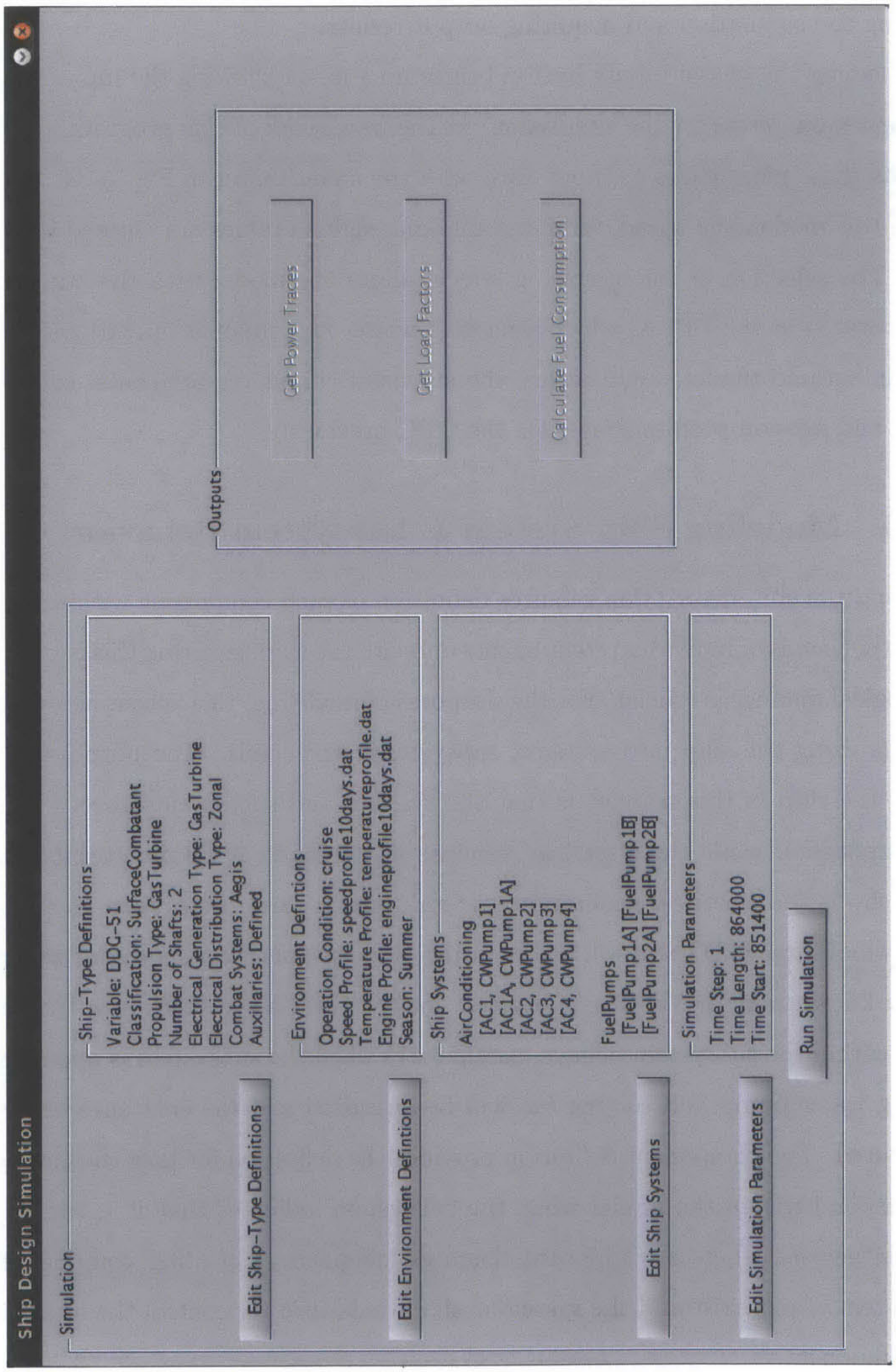
Selecting the global inputs button brings up a menu allowing the input of several different parameters for the simulation. In the framework of this program, this menu accepts these parameters as input files, with the menu shown in Fig. 6-34.

In this model, the speed, temperature, and engine profiles are entered in as data files. The selection of the operating season aligns the model with the current data incorporated in the EPLA, which separates winter and summer operations. A more fully integrated model would utilize the stochastic input requirements, such as the speed and mission profiles shown for the OPC previously.

6.5.5 Modeling Ship System & Subsystem Behaviors

Entering the ship description requires definition of each component within the ship. With thousands of individual components onboard the ship, ensuring this is performed in a logical manner is crucial. For the purposes of modeling, this is best accomplished by organizing the ship into systems, subsystems, and loads. The purpose of separating the ship in this manner is that the systems and subsystems govern behavior of components, while the electrical response of the load. A simple example of this hierarchy is the electrical generation system. In this case, the system level governs that two of three GTG sets will be operating, and the operating generator will switch with a known periodicity. The subsystem for the GTG set governs how the components within the subsystem behave, particularly when the subsystem is operating the cooling water pump and cooling fan will be energized and the enclosure heater will be secured. The component definition provides the definition for how the component will respond within the model when the subsystem indicates that it is running. In some situations where a component operates independent of other components, no subsystem would exist and the system design would directly control the operation of the component.

The relationship between the system, subsystem, and component can also be seen graphically, as demonstrated in Fig. 6-35. In this case, the air conditioning system


Ship Design Simulation
Simulation
Edit Ship-Type Definitions
Edit Environment Defintions
Edit Ship Systems
Edit Simulation Parameters
Run Simulation
Outputs
Get Power Traces
Get Load Factors
Calculate Fuel Consumption

Ship-Type Definitions
Variable: DDG-51
Classification: SurfaceCombatant
Propulsion Type: GasTurbine
Number of Shafts: 2
Electrical Generation Type: GasTurbine
Electrical Distribution Type: Zonal
Combat Systems: Aegis
Auxillaries: Defined

Environment Defintions
Operation Condition: cruise
Speed Profile: speedprofile10days.dat
Temperature Profile: temperatureprofile.dat
Engine Profile: engineprofile10days.dat
Season: Summer

Ship Systems
AirConditioning
[AC1, CWPump1]
[AC1A, CWPump1A]
[AC2, CWPump2]
[AC3, CWPump3]
[AC4, CWPump4]

FuelPumps
[FuelPump1A] [FuelPump1B]
[FuelPump2A] [FuelPump2B]

Simulation Parameters
Time Step: 1
Time Length: 864000
Time Start: 851400

Fig. 6-33: Graphical Main Interface Screen

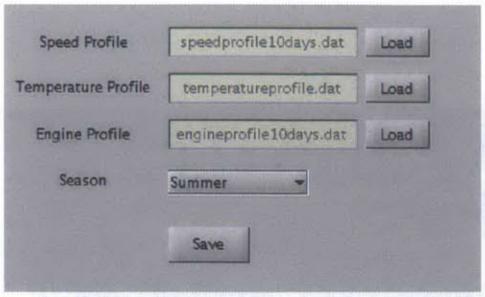

Speed Profile speedprofile10days.dat
Temperature Profile temperatureprofile.dat
Engine Profile engineprofile10days.dat
Season Summer

Fig. 6-34: Global Inputs Menu

is broken into five subsystems with each subsystem operating two components.

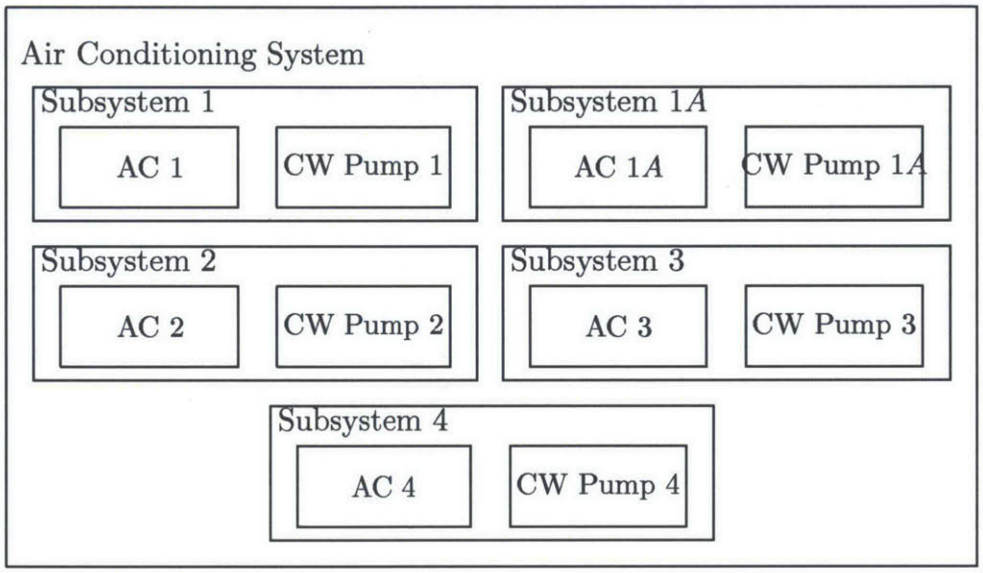
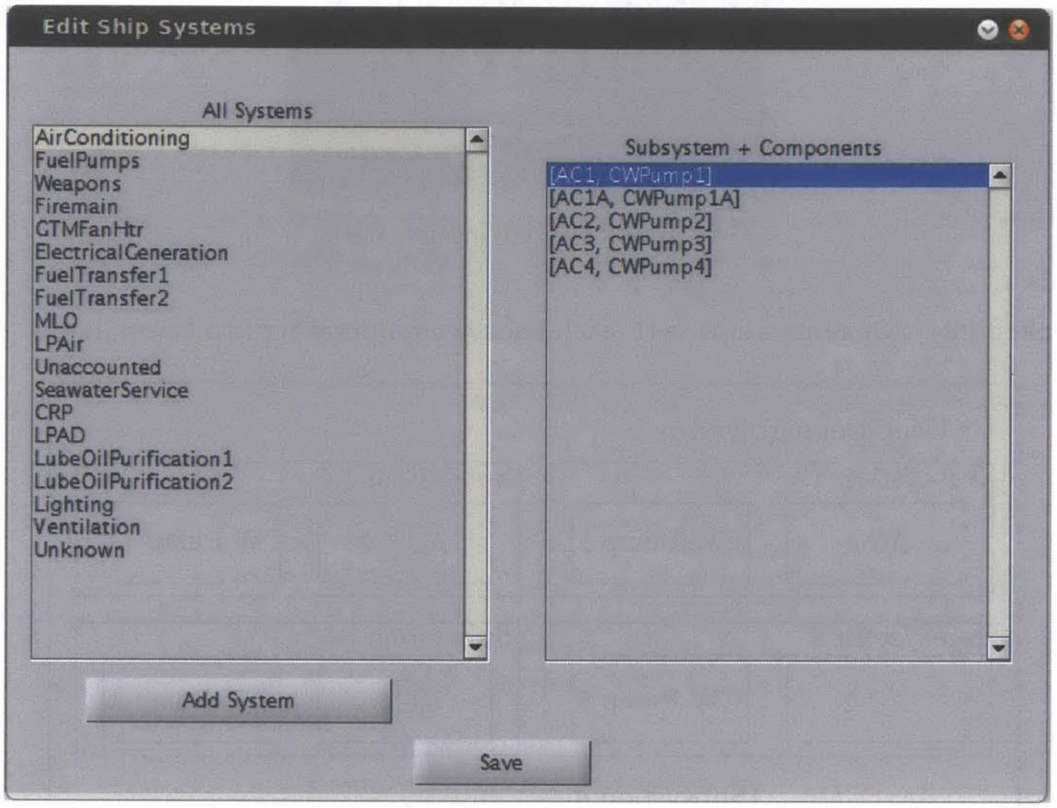


Fig. 6-35: AC System Relationships

The first step in the component definition therefore is the screen that provides a means of entering ship systems. This entry page is demonstrated in Fig. 6-36, and shows a sample list of systems. The system components, grouped by specific subsystems, are shown in a secondary window when the ship system is selected.

When the "add system" button is selected, it brings up the option to build a new system into the program, as shown in Fig. 6-37.

When a new system is entered, it will require definition first based on the type of system that it represents. In developing the model for the DDG-51, six types of models were identified and each will be discussed in the following sections. These model types are general enough that they describe both the system and subsystem


Edit Ship Systems

All Systems

AirConditioning
FuelPumps
Weapons
Firemain
GTMFanHtr
ElectricalGeneration
FuelTransfer1
FuelTransfer2
MLO
LPAir
Unaccounted
SeawaterService
CRP
LPAD
LubeOilPurification1
LubeOilPurification2
Lighting
Ventilation
Unknown

Subsystem + Components

[AC1, CWPump1]
[AC1A, CWPump1A]
[AC2, CWPump2]
[AC3, CWPump3]
[AC4, CWPump4]

Fig. 6-36: Model Ship System Entry

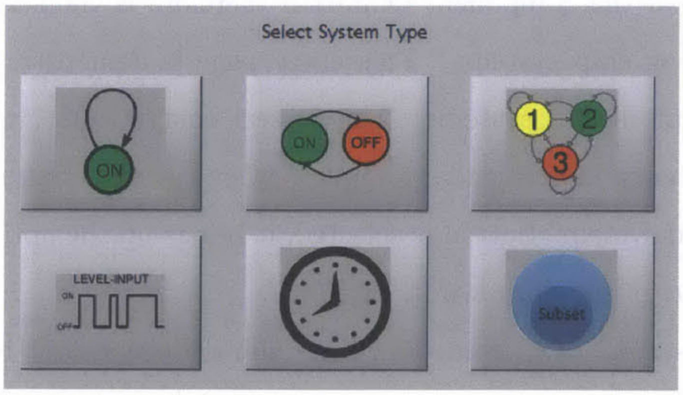


Fig. 6-37: System Entry Selection Menu

behaviors as the selection of an operating subsystem may behave differently than the method in which the subsystem operates individual loads.

The framework was implemented in the Matlab programming environment using an object-oriented programming data structure. The MATLAB code for the system object is shown in **ShipSystem.m**, which is located in Appendix E. The ShipSystem object is responsible for generating an operational profile for all of its subsystems by calculating the intervals of times in which each subsystem is on or activated. The six models identified to calculate these time intervals are discussed below.

Single-State

The single-state condition refers to a system or subsystem that maintains a single configuration in a given ship state, that is, a "base" load that is always "on". Examples of such systems include ventilation fans, some radars, or communication equipment that runs continuously during ship operation. It could also apply to equipment, such as a main reduction gear turning gear, that are secured during all underway operations. This equipment would use the input state of the ship to dictate the configuration of the system.

Cycle Type

A system with cycle-type characteristics is a "two-state" load that behaves periodically, but independent of the time of day. These systems have stochastic on/off cycling behavior. An example of this is the cycling behavior is the lube oil purifier onboard a ship. As shown in Fig. 6-12, the purifier runs with a regular periodicity that can be readily defined. By understanding the distribution of the time the purifier is running and the length of time it is stopped, this piece of equipment can be categorized. The stochastic models that govern the length of time the system remains in the on or off state require user inputs. These behaviors can be determined using the methods developed for stochastic modeling, defined in §6.4.2.

Finite State Machine

A finite state machine (FSM) allows the selection of operational conditions based on probability of transitioning between various states. A simple example of a FSM is shown in Fig. 6-38. This example represents a system with three states: off, low, or high. At a given time step, the system undergoes a transition and the possibility of the transition is defined with a given probability. For any system that uses an FSM definition, it would be imperative to properly define all possible transitions and the probability associated with each.

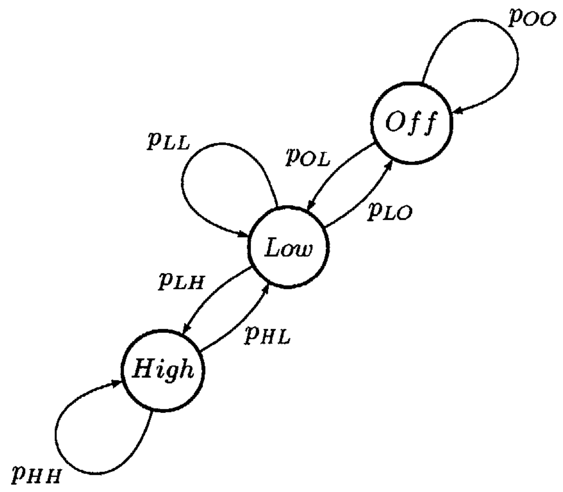


Fig. 6-38: Finite State Machine Example

The user defines the stochastic models of each state and the overall transition probability matrix. For the FSM in Fig. 6-38, the corresponding probability matrix P is shown in (6.3). Through the use of the transition probability matrix, the model will randomly select the next state at each step.

$$P = \begin{pmatrix} p_{OO} & p_{OL} & 0 \\ p_{LO} & p_{LL} & p_{LH} \\ 0 & p_{HL} & p_{HH} \end{pmatrix} \tag{6.3}$$

Level-Type

The level-type system is directly dependent on the state of a specific input. In this case, the system may be on whenever a specified condition is met and secured during all other conditions. An example of this rule is shown in the fuel service system in which a pump for a specific plant will be energized whenever one of the two GTMs in the operating plant is on. In this case, the subsystem is dependent on the "level" state of the corresponding GTMs shown graphically in Fig. 6-39. For a DDG-51, the GTM configuration is dependent on the ship speed as certain speeds require certain plant configurations as specified in the current NAVSEA design standards.

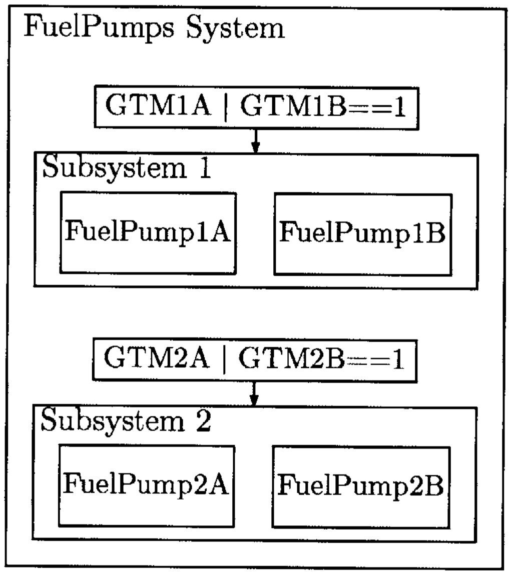


Fig. 6-39: Fuel Service System Model

Time Dependent

The time-dependent system depends on the time variable of the model to drive the cycling performance of the system. Some systems operate in a predictable manner over the course of a day, or periodically over the course of several days. Food service equipment in the galley is operated during meal hours, and sparsely during other times of day. Systems with these time-dependencies are modeled to energize with varying frequency based on the time of day. The user interface would provide a flexible design environment where the user can define the temporal interdependencies required to properly model the system.

Random Subset

The random subset method is necessary as Navy ships are constructed with a large amount of redundancy. This redundancy drives the addition of multiple subsystems that can all perform the same function equally. The random subset method will select the required number of running subsystems, and at a defined periodicity switch the running subsystems by selecting a new random subset. The AC plants, as shown in Fig. 6-35 previously, were modeled in this manner at the system level.

Ensuring these systems are design correctly is vital to ensuring a realistic output is achieved. An example of this is the ship's auxiliary systems. The auxiliary systems are the largest group of electrical consumers onboard a ship. Of the electrical energy used on a DDG-51 class ship, approximately 45% alone comes from the heating, ventilation, and air conditioning system [62]. The auxiliary systems incorporate everything from firemain to sewage processing, and encompass many independent systems that each have a unique design. In the case of the DDG-51, the systems are well defined. For a new ship design effort, however, each system would have to be designed in the model.

6.5.6 Load Electrical Modeling

Once the behaviors of components have been developed through the system and subsystem definition, a means of modeling the electrical response must be implemented. Within each subsystem, all subordinate components must be defined by the user. The MATLAB code for the component object is shown in `ShipComponent.m`, which is located in Appendix E. The user first defines each component as a master or slave. A slave component has the operational profile of its subsystem. When the subsystem is active, the component is always on. A master component is on when its subsystem is active, but has its own operational profile described by a stochastic model that is defined by the user. The various types of stochastic models are described in §6.4.2.

Depending on the operating procedure of the system, the user may define offset parameters. These values can be used to prevent components from turning on at the identical time, which could cause erroneously high transient behavior. Onboard a ship, operators would perform steps sequentially instead of the same time. For example, in the Electrical Generation system, when the corresponding GTG turns on, the module cooling fan turns on first. Within a couple of minutes, the cooling pump turns on and several minutes later, the heater turns off. These steps all occur prior to starting the GTG. The turn-on offset parameter for each component can be set accordingly to follow this operating procedure. It should be noted that the heater turns off when the subsystem is active and turns on when the subsystem is inactive. The user can define a flag parameter to invert the operational profile of the component in comparison to its subsystem.

The ShipComponent object is also responsible for generating the electrical response based on the operation of its parent subsystem. For this model, four separate methods of implementing a component response were used: constant, finger-print, finite state machine and user-defined as shown in Fig. 6-40.

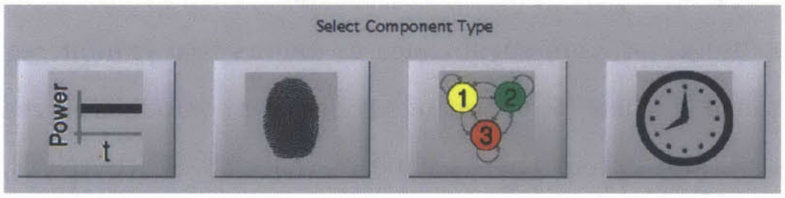


Fig. 6-40: Load Modeling Selection Screen

Each of these methods provides a unique means of developing a power trace, and is expanded upon in the following sections.

Constant

In a load factor analysis derived EPLA, the electrical utilization is accounted for using the rated (or nameplate) load of the component and multiplying it by the load factor for a given condition or season, as shown in (6.4).

$$P_{avg} = LF \cdot P_{rated}. \tag{6.4}$$

This method was described in significant detail in §6.2.1. Many loads do not have defined electrical behavior, and using the load factor assumptions may remain the best approach. Once the electrical (or potentially system) behaviors are determined, the load could be modeled using a higher-fidelity method.

Incorporating load factors from previous EPLA for every load onboard the ship, the user can compare the results of a simulation to those that would be obtained from the load factor analysis. In cases where the model deviates significantly in its long-term statistics from the load factor results, this can provide a means of determining the sources of error.

Fingerprint

The fingerprint method assumes that the load follows a unique and regular behavior that consists of three phases: a transient turn-on phase, a steady-state phase, and a transient turn-off phase. Each phase is individually defined within the model. This method is most useful for components such as motors that exhibit regular behavior, and is most applicable when the operating profile for a component is readily available. With the data obtained from the baseline of the USS SPRUANCE, many of the components on the ships have available profiles. An example of how this data looks graphically is shown in Fig. 6-41.

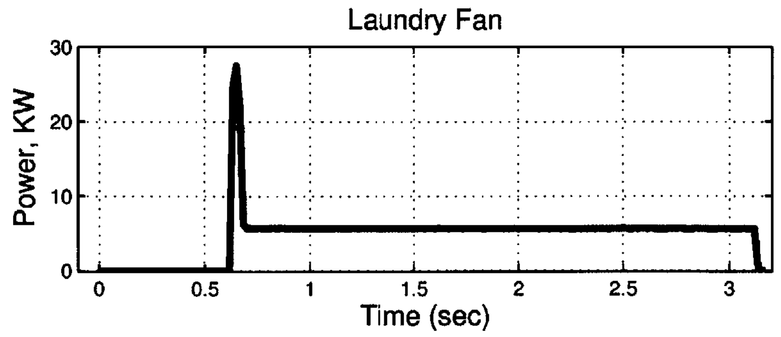


Fig. 6-41: Example Electrical Operating Data [62].

From this figure, it is straightforward to determine the three ranges of the operating cycle. The start-up transient is a short, well-defined inrush spike lasting only a few seconds. The steady-state region is clearly a well-behaved one for this fan, with little variation in the operating load for the component. The turn-off phase when the component is secured is again a short (order of seconds) transient that can be easily defined.

Each region of the load's operation is defined and stored within the model, and can be applied based on the condition of the load. When a load starts, the transient behavior will run through the allotted number of time steps. Following this, the steady-state behavior takes over until the component is secured. Steady state behavior can be modeled either as a looped series that recreates the actual variation seen in the equipment, or modeled as a stochastic distribution.

A large fraction of the HVAC, auxiliary, and engineering loads onboard ships utilize induction motors to perform their function. The profile for these motors is very similar, regardless of the function they serve. The profile looks like that shown in Fig. 6-41. There is an initial starting current of approximately 5-7 times the running current followed by steady operation. If the load demand changes for the motor, some variation will be seen in the steady region. Incorporating product models for common equipment, like induction motors or resistive heaters, would simplify the process of developing a model for a future ship design.

FSM

The finite state model is similar to that described in the fingerprint model, but requires a more complex series of inputs. For subsystems with different operating levels or conditions, the FSM model was used as described above. When the FSM model is utilized, however, it is necessary to define an electrical profile for each operating state of the equipment.

For most types of equipment, the electrical profile will be similar but at a different power level, though this does not have to be the case. In each operating state, it would be possible to define the electrical consumption using any of the other methods described here.

The NILM data from collected by Bennett at LBES demonstrates the effects for fuel oil service pumps, shown in Fig. 6-42. These pumps are positive displacement two speed pumps, and it is evident that when the pump shifts from low to high speed, doubling pump flow rate, the running current approximately doubles. This is the expected behavior for a positive displacement pump, but demonstrates the need for unique fingerprint models in each state.

For components described by an FSM, the user must define the stochastic model for each state and a transition probability matrix like those discussed in §6.5.5.

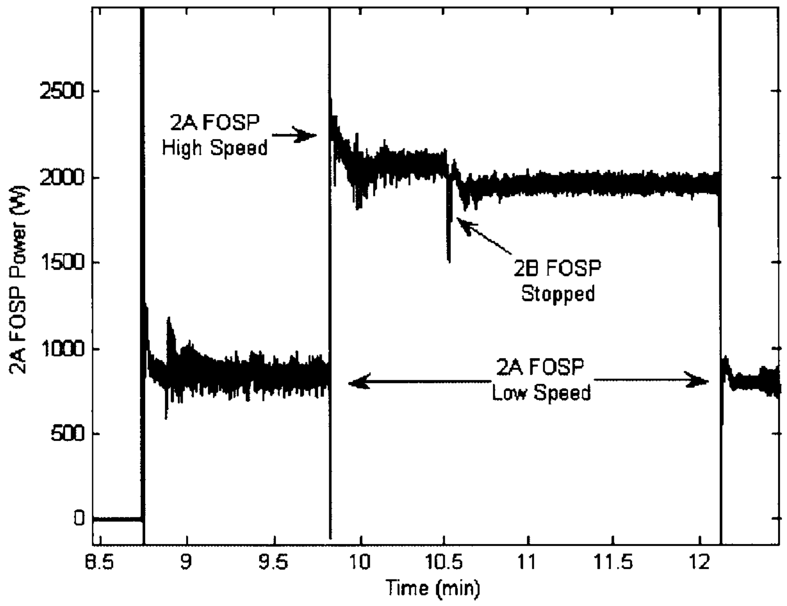


Fig. 6-42: LBES Power Drawn by 2A FOSP During a FOSP Pump Shift Evolution [65].

User Defined

The final profile is one required for input-dependent systems. In this case, one of the input variables for the model drives the electrical profile of the load. When using this method, the user defines the relationship between global inputs to the model and component electrical response. This user defintion allows for the creation of dynamic component profiles that simpler methods cannot replicate.

One example of a component using a user-defined model is the AC motor compressor. When developing a PDF for the AC compressor in the stochastic simulation section (Fig. 6-24), it was noted that the overall electrical consumption profile could be obtained but would lack the characteristics of the actual loading profile. To examine these temporal effects, the data was sorted such that individual profiles were obtained for each 2-hour block of time over the course of a day: 0000-0200, 0200-0400, etc. By taking the mean and standard deviation of all individual AC plants during each time segment using several weeks of data, Fig. 6-43 was created.

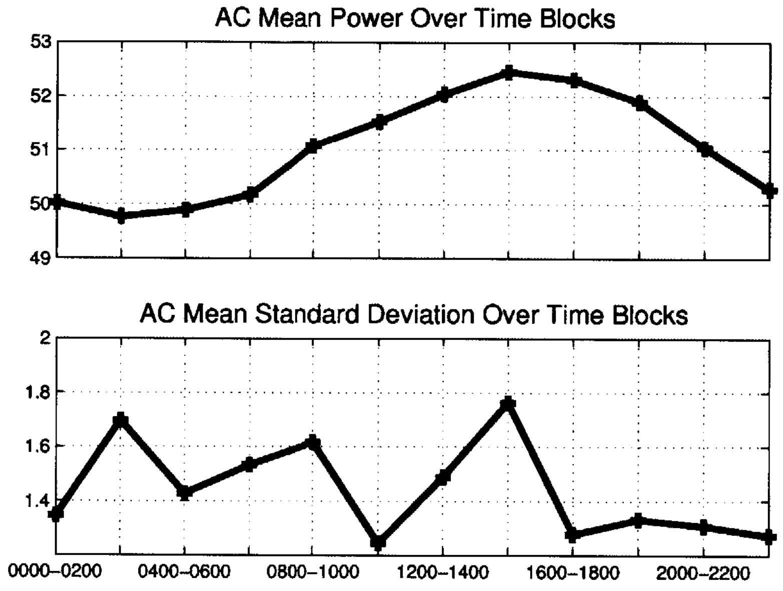


Fig. 6-43: AC Compressor Power Mean and Standard Deviation

Additionally, the AC system loading would be expected to change with the day-to-day variations in ambient temperatures. During the time period of analysis, the vessel is operating in a single location performing a continuing presence in anti-piracy operations. This singular mission profile allowed the investigation the effects of ambient temperature. By plotting the daily high, low, and mean temperatures seen in cities near the ship's operating location in the Gulf of Aden the weak effects of the temperature can be seen in Fig. 6-44.

Examining this data, it was noted that there was an additional correlation between the mean value for temperature and the total load placed on the AC compressors. By allowing the mean shown in Fig. 6-43 to drift slightly with the variation in

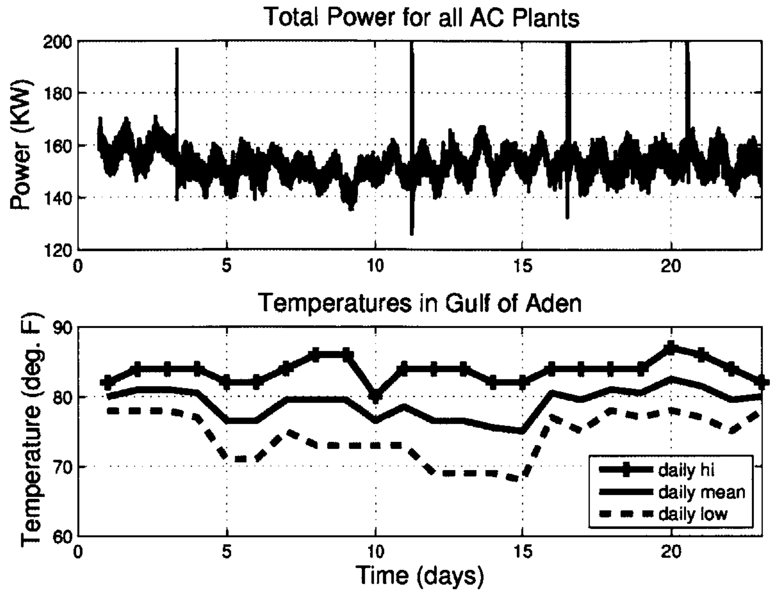


Fig. 6-44: Total Ship Compressor Loading and Temperature Variation

temperature a more accurate behavioral model could be developed.

For this model, the user can input parameters (A, B, C, D, E) to model the effect of the global input. For the AC compressor, the relationship between the temperature, T, and the temperature dependent mean, $\mu(T)$, is modeled as shown in (6.5), where the remaining variables are fit to the data set.

$$\mu_T = \frac{A(T-B)^C + D}{E}. \tag{6.5}$$

In a preliminary design of a ship many of the interrelations needed to develop these loads may not be available, and fidelity would be expected to improve as the design progresses. This electrical definition process does, however, allow a means of incorporating future capabilities, such as radar systems, in a unique way.

6.6 Stochastic Models

Stochastic models are used to describe processes with distribution functions that are known or can be estimated. Previous research has been done to model the stochastic behavior of components. Reference [73] uses normal distributions to model the uncertainty of daily power peaks loads in a system. Reference [74] proposes the use of normal, log-normal and beta distributions to model high voltage loads and reference [75] advocates that load distribution is a combination of normal, log-normal and Poisson distributions. For the purposes of the framework and the components surveyed from the Alaris study discussed in §6.3, the stochastic models used to describe operational behavior and load distribution are constant (deterministic), uniform, normal and exponential. While these distributions were sufficient for modeling the loads surveyed in the development of this framework, including other distributions such as log-normal, Poisson, Beta can be readily done. Estimating their parameters can be done similarly to those measured in the distributions discussed in the following sections.

6.6.1 Constant

The constant method represents the simplest possible distribution used in the stochastic models as it is the only discrete random variable. In this model the user defines a value of a, which is related to the random variable X_c by

$$\mathrm{P}(X_c = a) = 1. \tag{6.6}$$

6.6.2 Uniform

A random variable X_U has a uniform distribution, $f_{X_U}(x)$, if the PDF is constant within the interval a and b. In this case, the user defines the values of a and b, and the distribution is shown in (6.7).

$$f_{X_U}(x) = \frac{1}{b-a}, \quad \text{for } a \leq x \leq b. \tag{6.7}$$

6.6.3 Normal

When a random variable X_N has a normal distribution, the user must input the mean, μ, and standard deviation, σ. The system will then cycle with a frequency dictated by these values according to the distribution shown in (6.8). Care must be used in implementing this distribution, to ensure the probability does not return negative values.

$$f_{X_N}(x) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(x-\mu)^2}{2\sigma^2}}. \tag{6.8}$$

As mentioned in §6.5.6, the power traces for certain components can be created using the fingerprint method. The steady-state portion of the electric loading for the fingerprint method for many components are modeled with a normal distribution. Consider the transient electrical response of fuel oil purifier in Fig. 6-45.

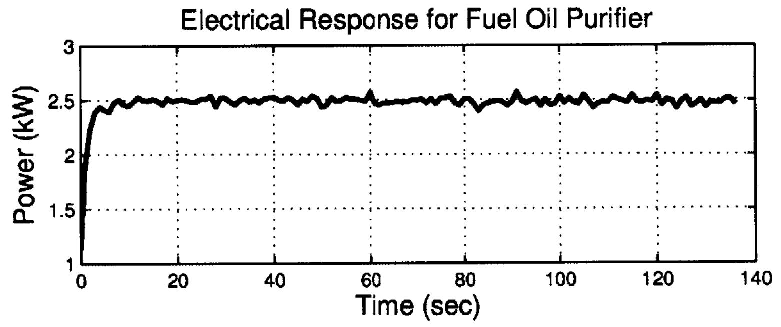


Fig. 6-45: Electrical response of a fuel oil purifier

A normal distribution describes the behavior of the steady-state region as there is an underlying mean with some variation around the mean. The histogram of the steady-state region is shown in Fig. 6-46 and the underlying normal distribution is prevalent.

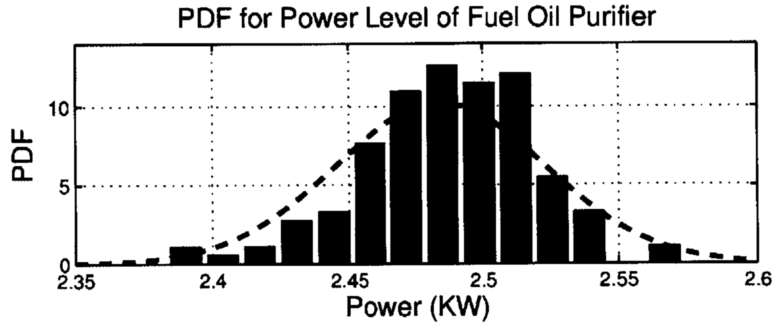


Fig. 6-46: Histogram of the steady-state region in the electrical response of a fuel oil purifier

If needed, a goodness-of-fit test can be conducted to perform a hypothesis test as to whether the data in the steady-state region can be modeled as normal distribution. Section 6.6.5 discusses the goodness-of-fit test.

To estimate the mean μ and standard deviation σ, N points of the steady-state region are extracted. With N observations $(x_1, x_2, \ldots, x_N)$ and the maximum likelihood method to estimate parameters [76], the estimates of the mean and variance of the underlying normal distribution are

$$\hat{\mu}_N = \frac{1}{N}\sum_{i=1}^{N} x_i, \tag{6.9}$$

$$\hat{\sigma}_N = \frac{1}{N}\sum_{i=1}^{N} (x_i - \hat{\mu})^2. \tag{6.10}$$

For a normal distribution, if the mean value is more than three standard deviations greater than zero, then the probability of randomly picking a negative value is less than one percent. To ensure that there are no negative values returned, the random number generator in the software implementation repeatedly selects a random value from a normal distribution until a positive value is returned.

6.6.4 Exponential

A random variable X_E has an exponential distribution $f_{X_E}(x)$ when the PDF is defined as shown in (6.11).

$$f_{X_E}(x) = \lambda e^{-\lambda x}, \quad \text{for } x \geq 0, \tag{6.11}$$

for $\lambda > 0$. When the exponential distribution is used, the user must input the variable λ, which is the rate parameter for the function. The choice of this value will determine how quickly the exponential function decays away, and will therefore influence the width of the distribution of cycle time.

The exponential random variable is widely used in to describe the interarrival times in a stochastic process. In this framework, the times in which the subsystems are turned on are modeled as a queue. In queuing theory, the arrival process (the length of times between arrivals) can be described as Markovian (M) and the service time distribution can be modeled as deterministic (D) or Markovian (M). For the model, each subsystem is assumed to be a single server in that the subsystem is either off (zero items in the queue) or on (one item in the queue). Using Kendall's notation [77], an $M/D/1$ queue is used for a Poisson process in which the service time is constant. However, an $M/M/1$ queue has a Poisson arrival process and a exponential distribution for a service time. Estimating λ is essential in describing the overall Poisson arrival process. With N observations of the interarrival times, a maximum likelihood estimate $\hat{\lambda}_N$ can be calculated as

$$\hat{\lambda}_N = \frac{N}{\sum\limits_{i=1}^{N} x_i}. \tag{6.12}$$

6.6.5 Goodness-of-Fit and Consistency

In the formal framework of hypothesis testing, the null hypothesis H_o states that a given random variable X has a given probability distribution $f_X(x)$. Several goodness-of-fit techniques exist to measure discrepancy of the actual data to a hypothesized set of data from the distribution of $f_X(x)$. If the discrepancy is statistically significant, then the null hypothesis is rejected and the conclusion is that the actual data could not have come from the hypothesized distribution $f_X(x)$.

The Pearson's chi-squared test is a commonly used technique which uses a measure of goodness-of-fit that is the sum of differences between observed and expected outcome frequencies. For each observation O_i, there is an expected observation E_i asserted by the null hypothesis and the test statistic computed as

$$\chi^2 = \sum_{i=1}^{N} \frac{(O_i - E_i)^2}{E_i^2}, \tag{6.13}$$

where N is the number of observations. The resulting value can be compared to the chi-squared distribution to determine the goodness-of-fit for a user-defined level of significance.

While hypothesis testing and goodness-of-fit techniques provide a way to measure the accuracy of an assumed distribution, measuring consistency of the maximum likelihood estimate is essential in determining the accuracy of the parameter. In statistics, a consistent estimator is one whose sequence of estimates becomes more concentrated around the true value of the parameter as the number of data points uses increases.

The estimators in (6.9), (6.10) and (6.12) are consistent estimators. The $\hat{\mu}_N$ estimator of μ, the mean of the normal distribution in (6.9), is itself a normal distribution with a mean of μ and a standard deviation of σ^2/N. As more data points are collected, N increases and the standard deviation of $\hat{\mu}_N$ tends to 0. Figure 6-47 illustrates how the standard deviation decreases as N increases and most of the distribution is centered around the mean.

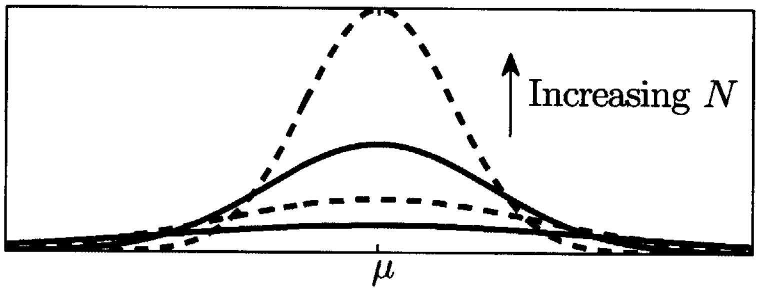


Fig. 6-47: Example normal distribution for several values of N. Standard deviation decreases as N increases.

Mathematically, the estimator $\hat{\mu}_N$ is consistent if

$$\lim_{N\to\infty} \mathbf{P}\left(|\hat{\mu}_N - \mu| \geq \epsilon\right) = 0, \tag{6.14}$$

for any fixed $\epsilon > 0$. As more data are collected to update the value of the estimator, the higher the probability that it is close to the true value of the parameter. The estimators in (6.10) and (6.12) can be proven consistent through a similar exercise [76].

Even with a collection of data, such as those gathered from the Alaris study in §6.3.3, there may not be enough sample points to reliably trust the estimated value of the parameters. The Alaris provided a foundation in which to develop the framework and estimate parameters to fit the underlying distribution. However, these values should be updated as more observations are made.

6.7 Behavioral Model Output and Results

Once the ship has been defined and the time parameters are set, the simulation can run using the user-defined models. After the simulation is complete, the GUI allows the user to plot power traces of individual components and of the entire 1SA switchboard. Three examples follow below to show the results of simulating components with the framework.

6.7.1 Lighting Center

A lighting center on the DDG-51 is a collection of service loads such as lighting, convenient outlets and non-vital variable loading. Based on observed data of the power draw, the lighting center loading was modeled as as a FSM in which there were two states. Both states had power level derived a normal distribution with different means and standard deviations estimated from the collected field data. Figure 6-48 shows a comparison between the actual and simulated data sets for a particular lighting center.

The time-dependent characteristics were included in the component definitions during setup as the mean and standard deviation of the power draw of the load center varied throughout the day. The mean and standard deviation gathered from the DDG-51 study are shown in Fig. 6-49. There are some spikes that occur in the dataset in Fig. 6-48. Such spikes might come from high-power, low-usage loads turning on for a short period of time. These smaller loads add to the distinct characteristics when viewing the lighting centers alone. When viewing a larger zone of the ship, these characteristics are insignificant and are ignored in the model framework.

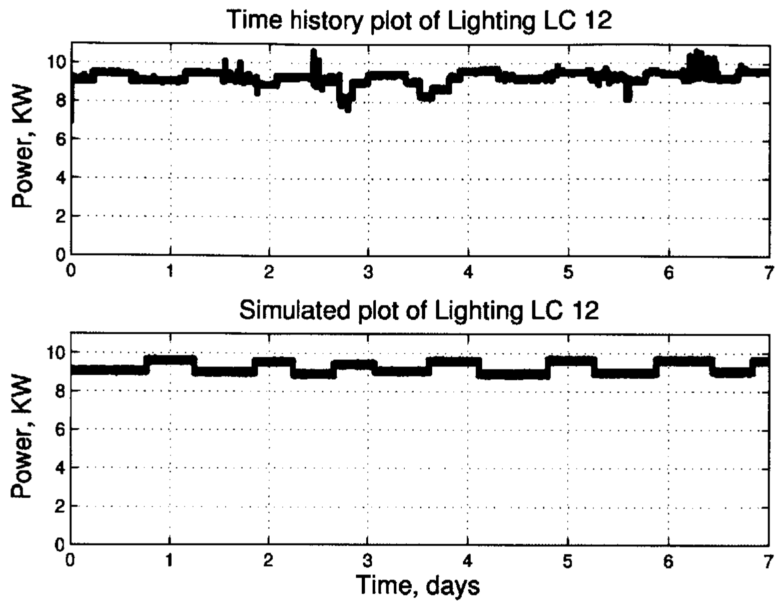


Fig. 6-48: Lighting center comparison

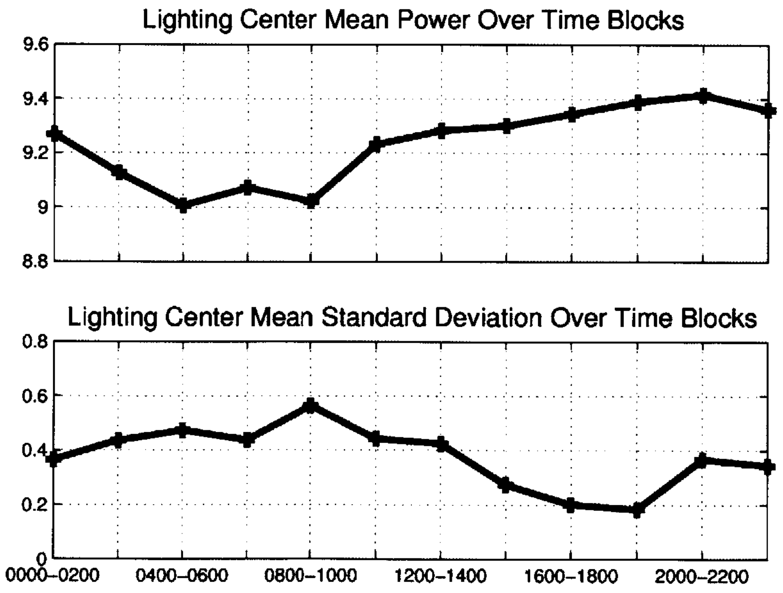


Fig. 6-49: Lighting center statistics

6.7.2 AC Plants

The AC plants on-board the DDG-51 operate by refrigerating a chilled water loop and rejecting heat to seawater. The chilled water is then piped throughout the ship to provide air conditioning and electronic equipment cooling. There are five AC plants on-board the DDG-51 class, labeled 1, 1A, 2, 3, and 4. A brief introduction to the AC plants is included in the Section 6.4.2 By examining several weeks of data from MCMAS, a distribution over this time period was shown in Fig. 6-23 for each AC unit.

It is evident that there are some differences in the mean between AC plants, but each plant demonstrates a similar pattern of loading. Differences in mean are not unexpected since there may be small differences in the chilled water utilization between regions of the ship. An average distribution for a single AC plant was shown in Fig. 6-24.

As previously discussed, three of the five AC plants operate during normal operating configuration. There are intervals of times during switching in which four AC plants are in operation as was shown in the time series which is included here in Fig. 6-50 for convenience.

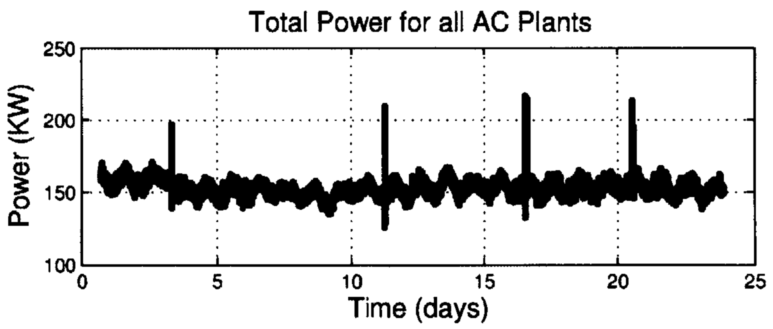


Fig. 6-50: Time Series of AC Plant Cumulative Load

From this time series, it is evident that the loading profile has a couple of notable features. The first is that the transient periods of switching AC plants yield power spikes periodically, as discussed previously. The second is that the AC plant loading is heavily diurnal; over each one-day period there exists a minimum that occurs in the early portion of the morning and a maximum that occurs in the afternoon.

The diurnal behavior presents an additional difficulty for a stochastic model. Section 6.5.6 describes how the model for the AC compressor can include the temporal effects by creating a stochastic profile of the power drawn over the course of a day. Figure 6-43 contained the mean and standard deviation of all individual AC plants during each two-hour interval over in a day and using several weeks of data.

Additionally, the AC system loading was shown to be dependent on temperature. Figure 6-44 showed the temperatures in the Gulf of Aden while the surveyed ship was conducting anti-piracy operations and (6.5) illustrated a model for including the effects of temperature on the electrical profile. Once the diurnal and temperature effects were included in the model for the AC plants, the simulated results match very well with the actual load profiles. These behaviors are shown in Fig. 6-51.

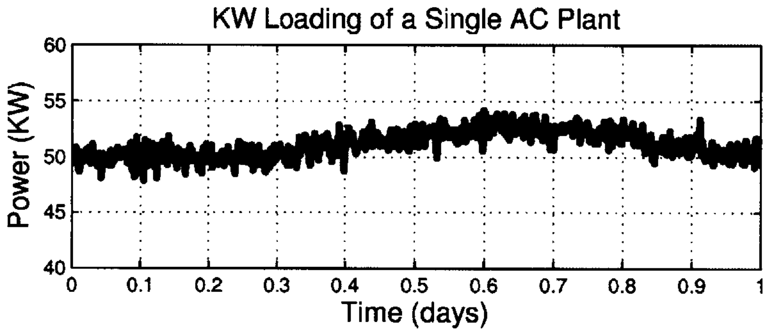


Fig. 6-51: Model Results for single AC Compressor

The user-defined profile with diurnal variation is clearly visible in the model for the AC compressor over time. The unit is secured and restarted within the time period modeled based on defined system behaviors. No two simulations would have the same profile, since the configuration changes are controlled by a random distribution of time with a specified mean. The chilled water pump exhibits the expected power peaking expected using the fingerprint method of load definition. It is also of value to compare the total loading seen across all AC compressors to the loading profile seen in actual fleet data over the time frame. The results for this comparison are presented in Fig. 6-52. These results show the strong correlation level between simulation and fleet data that can be created using the behavioral modeling.

The large spikes in the profile represent the operating condition where four com-

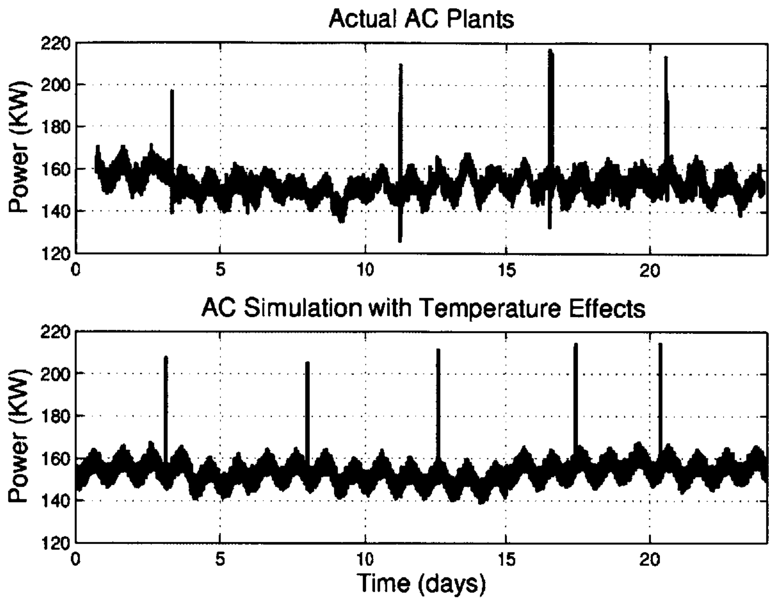


Fig. 6-52: Comparison of AC Load Simulation and Actual Profile

pressors are online in the intermediate state of switching AC plants. The simulation performs this switching randomly with similar periodicity to that seen in the fleet, but it would not be expected to line up at corresponding times. The relative frequency and peak loading that occurs in this condition is also important when validating the results of a final model.

6.7.3 1SA

Once all individual systems, subsystems, and loads applicable to the 1SA switchboard were defined the simulation was run. The most important aspect of the simulation for the calculation of an EPLA is the overall loading, since this will be the primary result used for the sizing of electrical distribution equipment.

The output profile for the 1SA switchboard is shown for a ten-day simulation is shown below in Fig. 6-53, while the actual loading on the 1SA switchboard during this time period is shown in Fig. 6-54.

While the simulation does not perfectly recreate the 1SA switchboard, it captures many features that exist within the system. Power transients associated with starting loads are captured, and much of the behavior over time is included. Since power

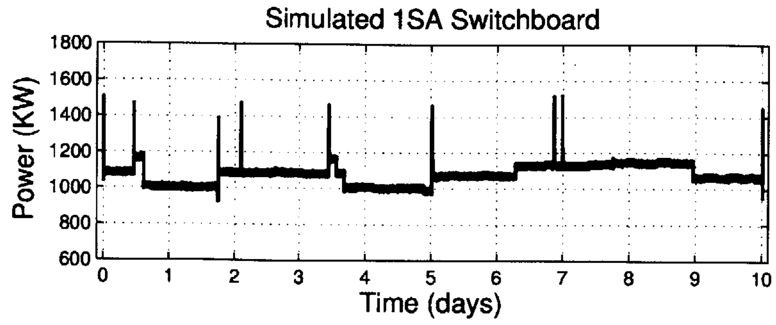


Fig. 6-53: 1SA Simulation (10 days)

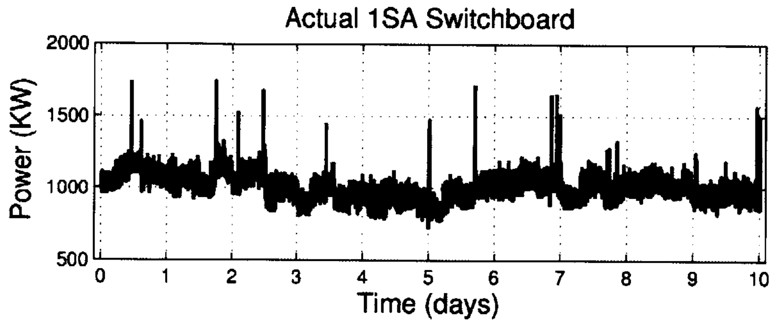


Fig. 6-54: 1SA Actual Load (10 Days)

transients have an impact on the sizing of generators, breakers, and cabling this behavioral model could allow designers to rely less on large margins and instead optimize the plant for expected load conditions. These results, using a relatively small subset of fleet data, demonstrate that the method can deliver high fidelity results and could enhance the ship design process. Overall, the simulation for the 1SA switchboard is bound within the same general region (800-1200 kW), indicating a good data fit for this simulation. Some components (primarily weapons and combat systems) did not have profiles that could be modeled from either MCMAS or the USS SPRUANCE data. For these components the previous EPLA values, calculated with load factors, was used, which led to producing a more level output than what is seen in the fleet.

The randomness of the system models dictates that no two simulations will be the same, and that the different power levels seen in Fig. 6-53 will change for each run. Behaviors linked to inputs (such as GTM stops and starts) would be the same for every simulation. By running the simulation many times a long term statistical description of ship behavior could be created, similar to the process for a Monte Carlo method.

Within the program, results from this simulation are available to the user through an interface that allows examination of the results on both a component and overall basis. An additional benefit of using the program is that the individual results for a selected system could be analyzed if desired. This benefit would provide the ability to use model results to inform selection of components, or could be used for the purposes of model validation.

6.8 Conclusions and Future Work

The framework described in this chapter for the development of power system behavioral models implements a flexible solution to the increasingly complex problem of conducting "what-if" studies for a proposed or existing power system design. Here, the focus was on an "islanded" power system, specifically, the distribution network of a DDG-51 destroyer. The emulation described in this chapter can be used to reproduce the behavior of the ship and power system under a variety of different operating scenarios. The emulation can be used to provide base data for other studies, including fuel consumption surveys, damage assessments, and sensitivity analyses to determine the reliability of metrics like EPLA load factors. By structuring a model such that the inputs and outputs are linked questions not yet formulated could be answered rapidly. This approach can be extended to other "small" power systems like microgrids, or regions of a power system that can be considered from a local perspective where renewables and distributed generation may be heavily present.

Utilizing available electronic data to update the information in DDS 310-1 is a topic with many avenues to be pursued. In the near-term, the data potentially available through the MCMAS system onboard ships presents a method of accurately updating the load factors used today. In addition to the load factors, they can provide the statistics necessary for accurately developing stochastic analyses or modeling systems. Working with fleet sponsors to create a statistically relevant data set across the fleet would be required to perform these tasks.

In the longer-term, developing a program that expanded the capability to model the ship in a simpler manner would be beneficial. Developing this program as a user-friendly program would move the modeling of the electric plant to a more realistic representation of equipment operation. As shown in the model developed here, this program would allow the user to to build models are they are known. The designer could model certain components with load factors when information is limited and increase fidelity as the design becomes clearer. This tool would be useful not only in sizing the electrical generating system, but could be useful throughout a ship's lifetime.

In developing the behavioral model, a significant amount of future work exists. Comparing the actual 1SA data to the simulated 1SA data (previously shown in Fig. 6-53 and Fig. 6-54), the degree of fluctuation about a mean operating state is not captured. One reason variation in the simulated data may be underrepresented is that there are weak dependencies in the electrical responses to system variables. By referring to the data collected by Bennett at the LBES facility, the response of a main lube oil pump under maneuvering transients is shown in Fig. 6-55.

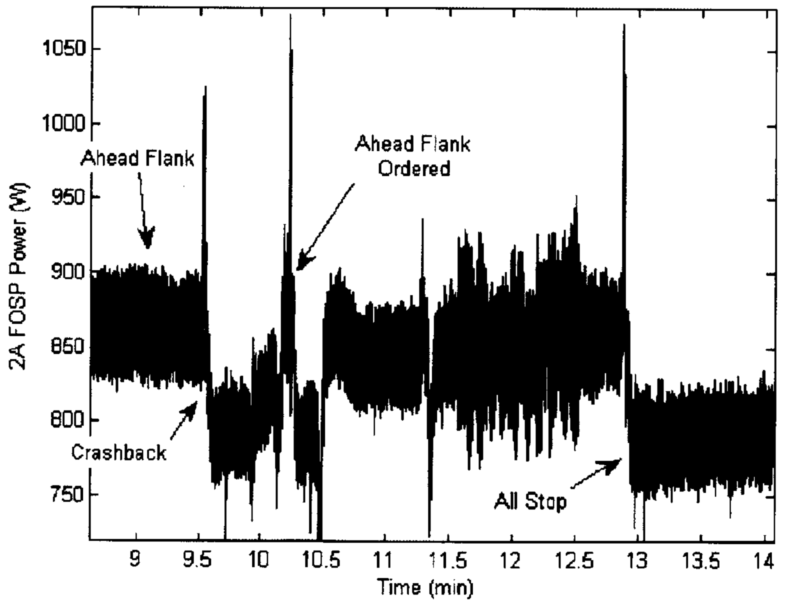


Fig. 6-55: LBES Variation in MLO Pump With Maneuvering Transients [65].

In this case, changes in the speed of the propulsion shafting causes change in lube oil system pressure, which causes slight changes in the operating point on the pump curve. Given the available data used for model development, these small changes could not be modeled. Across many components, this lack of fidelity could cause these errors. In future iterations of ship models, increasing the uncertainty in the individual loads by adjusting their PDF and CDF could improve the simulation characteristics.

Another potential cause for error in this model is some uncertainty in the data set used to develop the model. The data is a fairly comprehensive set of loads on the ship, but each load is individually only monitored for a few minutes in most cases. Monitoring for longer periods, and under different operating conditions, could give an improved data series from which to draw from.

Chapter 7

Multi-Function Monitor

The US Navy uses AC zonal electrical distribution system (ACZEDS) to provide reliable and survivable power in their DDG-51 ships. Multi-function monitors (MFM) III units have also been used for enhanced zonal protection. These units use algorithms to detect faults, to communicate with one another and to isolate faults by tripping selective breakers. Currently, these MFM units are configured by user-selected static thresholds for fault detection. This chapter will propose methods to enhance the performance of the MFM III units by incorporating nonintrusive load monitoring (NILM) methods to protect against high impedance faults. Experiments conducted in a laboratory setting will demonstrate the utility and effectiveness of a NILM-enhanced MFM unit. While the discussion in this chapter is based on the zonal protection for a ship's power grid, the methods here can be applied to a variety of microgrids.

7.1 Introduction

Before the use of zonal electrical distribution systems (ZEDS), fault isolation was performed by selecting the overcurrent settings on circuit breakers such that those closest to the fault will trip first. This process of selective tripping was designed such that a fault will be isolated while minimizing the impact on the rest of the system [78]. With the implementation of ZEDS, the task of isolating faults became more involved as the direction of current flow through the components in the system depended on the system configuration. This problem required a protective system that could sense the direction of current flow and provide intelligent fault detection and isolation actions [79] as discussed in §7.2.

As previously shown in §6.1.2, the ZEDS system for a DDG-51 Flight IIA Arleigh Burke class destroyer is shown in Fig. 7-1. The DDG-51 is equipped with eleven

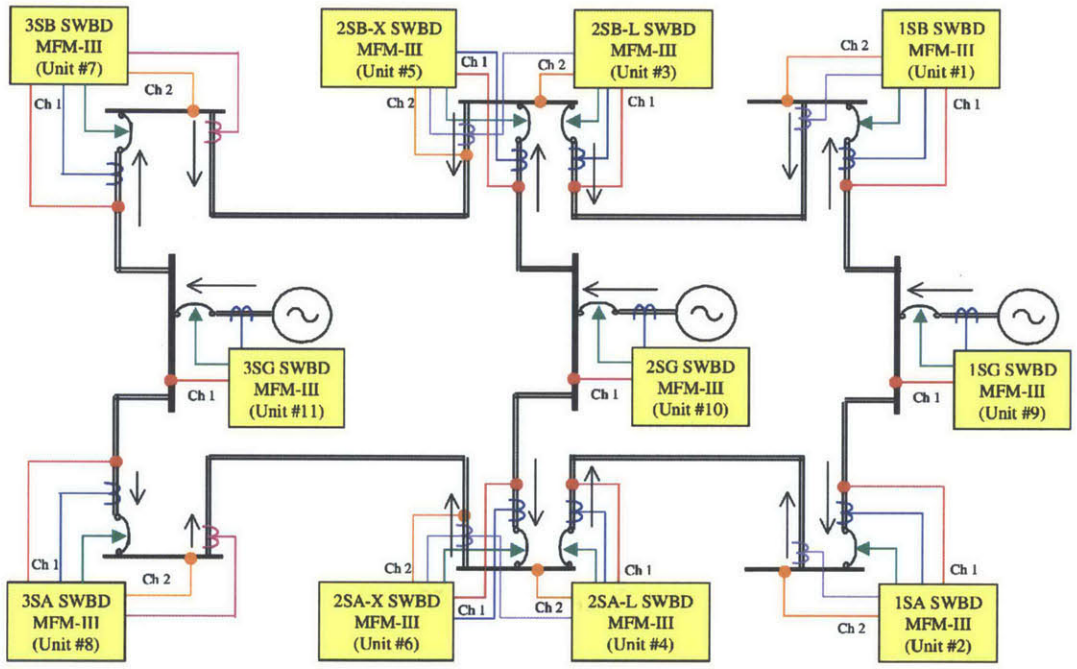

3SB SWBD MFM-III (Unit #7)
2SB-X SWBD MFM-III (Unit #5)
2SB-L SWBD MFM-III (Unit #3)
1SB SWBD MFM-III (Unit #1)
3SG SWBD MFM-III (Unit #11)
2SG SWBD MFM-III (Unit #10)
1SG SWBD MFM-III (Unit #9)
3SA SWBD MFM-III (Unit #8)
2SA-X SWBD MFM-III (Unit #6)
2SA-L SWBD MFM-III (Unit #4)
1SA SWBD MFM-III (Unit #2)
Ch 1
Ch 2

Fig. 7-1: DDG-51 Flight IIA ZEDS. This diagram includes the addressing, locations and signal inputs of the MFMs. The defined positive direction of current flow is also shown for each MFM. [59].

multi-function monitors (MFMs). In the event of a fault, the MFM III units attempt to minimize the area of fault isolation to maintain power to the maximum number of undamaged loads in the power system [59].

The simplified electrical diagram of the MFM III is shown in Fig. 7-2.

Each MFM III unit measures two line-to-line voltages from two sets of potential transformers and the three line currents from two sets of current transformers. The voltage and current sets are separated into channel 1 and channel 2 signal inputs. The defined positive direction of current flow for each MFM is shown in 7-1. Each MFM III unit also has three ethernet ports to send and receive a system information matrix to adjacent MFM III units. Fig. 7-1 includes the addressing, locations and

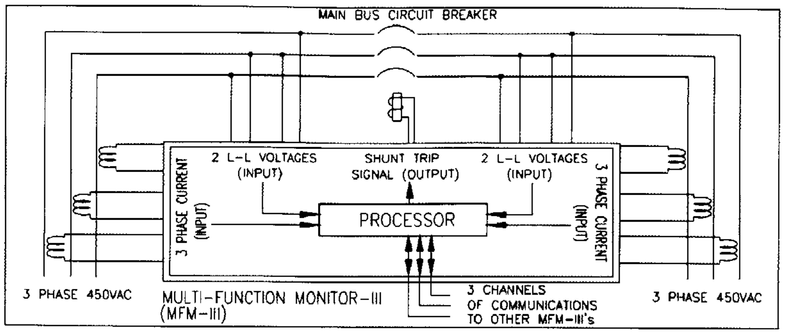


Fig. 7-2: Functional Diagram of MFM. The diagram shows the sensor layout for the MFM. The three phase current and two voltage measurements per channel, the three ethernet communications channels and the shunt trip output to the associated circuit breaker. The shunt trip status input of the circuit breaker is not shown in the diagram.

signal inputs of the MFMs. This system information matrix is discussed further in §7.2.1. Finally, each unit inputs the open/close status of its adjacent circuit breaker. The MFM III units run the integrated protective coordination system, as discussed in §7.2, to output a shunt trip signal to open its adjacent circuit breaker for fault isolation.

7.2 Integrated Protective Coordination System

In order for the integrated protective coordination system to work effectively, the MFM III units need to be addressed and numbered properly. By knowing its own address and type, each MFM III can determine its direct neighbors and the remote information necessary to make a shunt trip decision during a switchboard fault detection. All MFM III units contain the same software regardless of location, but the algorithms or routines accessed by each MFM III unit ultimately depends on its address or type shown in Table 7.1.

The MFM units numbered 9, 10 and 11 are a generator-type unit, type 4, and only execute MFM-I software. The MFM I was the first generation of the MFM specifically designed in radial distribution systems. Their functions were to isolate port and starboard buses in the event of significant down-line faults when measuring current level over a specified threshold [80]. The MFM III units placed on the "outside

Table 7.1: MFM III address/number/location/type assignment

Address	Number	Location	Type
1SB-X	1	1	1
1SA-X	2	1	1
2SB-L	3	2	2
2SA-L	4	2	2
2SB-X	5	2	3
2SA-X	6	2	3
3SB-X	7	3	1
3SA-X	8	3	1
1SG	9	1	4
2SG	10	2	4
3SG	11	3	4

corners" of the plant, 1SA, 1SB, 3SA and 3SB switchboards, are type 1. The units placed on the "inside corners", 2SA and 2SB are type 3 units. The remaining type 2 units are responsible for shunt tripping the longitudinal air circuit breakers (ACB) and do not run MFM-I software. The location of the MFM is based on its physical location in one of the three zones of the ship.

7.2.1 System Information Matrix

As mentioned earlier, each MFM III unit is equipped with three ethernet ports used to transmit and receive system information. Once each unit knows the status of every other unit, a coordinated response can be made. The system information matrix passed to each unit is an 11 x 15 matrix. The eleven rows of the matrix correspond to information pertaining to each of the eleven MFM III units. The fifteen columns consist of IPCS logic information necessary to achieve coordinated protection. Each of the fifteen columns is listed below with a brief description.

Number See Table 7.1.

Location See Table 7.1.

Type See Table 7.1.

CT1 Direction This value is the current flow direction in channel 1. This value is set to 0 when no fault is detected. Figure 7-1 indicates positive power/current directions (1). Flow in the reverse direction returns a -1. This value is fixed to 0 for the Type 4 MFM III units.

CT2 Direction This value is the current flow direction in channel 2. This value is set to 0 when no fault is detected. Figure 7-1 indicates positive power/current directions (1). Flow in the reverse direction returns a -1. This value is fixed to 0 for the Type 4 MFM III units.

Circuit Breaker Status This value is the local circuit breaker status. It is set to 0 for open and 1 for closed. The value is initialized to 0 at startup.

Fault Status This value is the fault detection status. It is set to 0 when no fault is detected and 1 when a fault is detected. The value is initialized to 0 at startup. This value is fixed to 0 for the Type 4 MFM III units.

Switchboard Fault Status This value is the internal/downstream switchboard fault detection status. It is set to 0 when a local switchboard fault is not detected and 1 if detected. The value is initialized to 0 at startup. This value is fixed to 0 for the Type 4 MFM III units.

MFM1 Action Status This value is the MFM I shunt trip flag. It is set to 0 if there are no local shunt trips based on MFM I algorithms. It is set to 1 if there is one. The value is initialized to 0 at startup. This value is fixed to 0 for the Type 4 MFM III units.

Fault Counter This value is the fault counter. It is set to 0 when no fault is detected. While there is a fault, the value is incremented by 1. The value is initialized to 0 at startup.

Fault Reset Counter This value is the fault reset flag. It is set to 1 when a fault is cleared. It will remain 1 for at least 5 ms until a fault is no longer detected and then reset to 0. It is initialized to 0 at startup and set to 0 while a fault is detected. This value is fixed to 0 for the Type 4 MFM III units.

ACB Shunt Trip Status This value is the air circuit breaker (ACB) shunt trip flag. It is set to 0 when there is no local shunt trip and it is set to 1 if there is one. This value will reset to 0 when the (ACB) is reclosed. The value is initialized to 0 at startup. This value is fixed to 0 for the Type 4 MFM III units.

I'm OK Status This value is set to 1 if local MFM III has communicated with all other MFM III units in the last second. It is set to 0 if any of the other MFM III units has not communicated with local unit in the last second.

CT1 Current Magnitude This value is the current magnitude flag for channel 1. It is initialized to 0 at startup and set to 0 when no fault is detected. During a fault, the value may be 1, 2 or 3. See §7.2.4 for appropriate settings. This value is fixed to 0 for the Type 4 MFM III units.

CT2 Current Magnitude This value is the current magnitude flag for channel 2. It is initialized to 0 at startup and set to 0 when no fault is detected. During a fault, the value may be 1, 2 or 3. See §7.2.4 for appropriate settings. This value is fixed to 0 for the Type 4 MFM III units.

The Fault Status column is determined by the High Speed Relay (HSR) algorithm that is explained in §7.2.2. Once a fault has been detected and has persisted long enough according to the Fault Counter, the IPCS algorithm seeks to determine the type of fault. More information of fault type determination is found in §7.2.4. Using the information found in the system information matrix, a coordinated response can be made by each unit to send a shunt trip signal to isolate the fault area.

7.2.2 High Speed Relay Algorithm

The High Speed Relay (HSR) algorithm uses the current and voltage measurements to assess fault status. The outputs of the HSR algorithm used by the IPCS algorithm are the magnitude of the voltage using Park's transformation, instantaneous three-phase power, sum of steady-state powers, and the fault status. The details of these routines and outputs are discussed below.

Park's Transformation

A Park's transformation is used to change the reference frame from that of the physical stator to that of a theoretical rotating 3-phase electrical signal. The general Park's Transformation matrix that is used in the HSR algorithm is

$$\begin{bmatrix} V_d \\ V_q \\ V_0 \end{bmatrix} = \frac{2}{3} \begin{bmatrix} \cos(\theta) & \cos\left(\theta - \frac{2\pi}{3}\right) & \cos\left(\theta + \frac{2\pi}{3}\right) \\ \sin(\theta) & \sin\left(\theta - \frac{2\pi}{3}\right) & \sin\left(\theta + \frac{2\pi}{3}\right) \\ \frac{1}{2} & \frac{1}{2} & \frac{1}{2} \end{bmatrix} \begin{bmatrix} V_a \\ V_b \\ V_c \end{bmatrix}, \tag{7.1}$$

where

$$\theta = \omega_o t \tag{7.2}$$

$$V_a = V_{ab} \cos(\omega_o t + \phi) \tag{7.3}$$

$$V_b = V_{bc} \cos(\omega_o t - \frac{2\pi}{3} + \phi) \tag{7.4}$$

$$V_c = V_{ca} \cos(\omega_o t + \frac{2\pi}{3} + \phi). \tag{7.5}$$

For the purposes of the MFM calculations, it is assumed that the three line-to-line voltages have the same frequency $\omega_o = 2\pi \cdot 60$ as the rotating reference frame but with an arbitrary phase ϕ. Generally, the Park's transformation can be used for currents as well [81].

The MFM III only measures the two line-to-line voltages, V_{ab} and V_{bc}. Therefore, slight modifications must be made to (7.1) to use two line-to-line voltages as opposed to the three line voltages. For a three phase electrical circuit, the sum of the line-to-line voltages must be zero. That is,

$$V_{ab} + V_{bc} + V_{ca} = 0. \tag{7.6}$$

Looking at the first row of the Park's Transformation to compute V_d using the line voltages yields

$$V_d = \frac{2}{3}\left[V_{ab}\cos\theta + V_{bc}\cos\left(\theta - \frac{2\pi}{3}\right) + V_{ca}\cos\left(\theta + \frac{2\pi}{3}\right)\right]. \tag{7.7}$$

Using the relationship of (7.6) and substituting for V_{ca} gives

$$V_d = \frac{2}{3}\left[V_{ab}\cos\theta + V_{bc}\cos\left(\theta - \frac{2\pi}{3}\right) - (V_{ab} + V_{bc})\cos\left(\theta + \frac{2\pi}{3}\right)\right]. \tag{7.8}$$

Multiplying through and gathering like terms give

$$V_d = \frac{2}{3}\left[V_{ab}\left(\cos\theta - \cos\left(\theta + \frac{2\pi}{3}\right)\right) + V_{bc}\left(\cos\left(\theta - \frac{2\pi}{3}\right) - \cos\left(\theta + \frac{2\pi}{3}\right)\right)\right]. \tag{7.9}$$

Using the trigonometric identity

$$\cos u - \cos v = -2\sin\left(\frac{u+v}{2}\right)\sin\left(\frac{u-v}{2}\right) \tag{7.10}$$

and noting that sine is an odd function, (7.9) becomes

$$V_d = \frac{4}{3}V_{ab}\left[\sin\left(\theta + \frac{\pi}{3}\right)\sin\left(\frac{\pi}{3}\right)\right] + \frac{4}{3}V_{bc}\left[\sin\theta\sin\left(\frac{2\pi}{3}\right)\right]. \tag{7.11}$$

Similarly, V_q can be computed using the second row of the Park's Transformation and it produces

$$V_q = \frac{2}{3}V_{ab}\sin\theta + \frac{2}{3}V_{bc}\sin\left(\theta - \frac{2\pi}{3}\right) + \frac{2}{3}V_{ca}\sin\left(\theta + \frac{2\pi}{3}\right). \tag{7.12}$$

Using the relationship of (7.6) and substituting for V_{ca} gives

$$V_q = \frac{2}{3}V_{ab}\sin\theta + \frac{2}{3}V_{bc}\sin\left(\theta - \frac{2\pi}{3}\right) - \frac{2}{3}\left(V_{ab} + V_{bc}\right)\sin\left(\theta + \frac{2\pi}{3}\right). \qquad (7.13)$$

Multiplying through and gathering like terms give

$$V_q = \frac{2}{3}V_{ab}\left[\sin\theta - \sin\left(\theta + \frac{2\pi}{3}\right)\right] + \frac{2}{3}V_{bc}\left[\sin\left(\theta - \frac{2\pi}{3}\right) - \sin\left(\theta + \frac{2\pi}{3}\right)\right]. \qquad (7.14)$$

Using the trigonometric identity

$$\sin u - \sin v = 2\cos\left(\frac{u+v}{2}\right)\sin\left(\frac{u-v}{2}\right) \qquad (7.15)$$

and noting that sine is an odd function, (7.14) becomes

$$V_q = -\frac{4}{3}V_{ab}\left[\cos\left(\theta + \frac{\pi}{3}\right)\sin\left(\frac{\pi}{3}\right)\right] - \frac{4}{3}V_{bc}\left[\cos\theta\sin\left(\frac{2\pi}{3}\right)\right]. \qquad (7.16)$$

Noting that $\sin\left(\frac{\pi}{3}\right) = \sin\left(\frac{2\pi}{3}\right)$ (7.11) and (7.16) can be further simplified as follows

$$V_d = \frac{4}{3}\sin\left(\frac{\pi}{3}\right)\left[V_{ab}\sin\left(\theta + \frac{\pi}{3}\right) + V_{bc}\sin\theta\right] \qquad (7.17)$$

and

$$V_q = -\frac{4}{3}\sin\left(\frac{\pi}{3}\right)\left[V_{ab}\cos\left(\theta + \frac{\pi}{3}\right) + V_{bc}\cos\theta\right]. \qquad (7.18)$$

The V_0 value is not calculated in the HSR routine. This value checks the three voltage inputs to verify that they sum to zero. Since only two line voltages are measured, the system is assumed to be a balanced 3-phase system and it is assumed that the sum of the voltages equals zero. Once the V_q and V_d values are calculated, the magnitude and angle of the voltage are expressed as

$$\begin{aligned} \mathit{Magnitude} &= \sqrt{V_d^2 + V_q^2} & (7.19) \\ \mathit{Angle} &= \arctan\left(\frac{V_q}{V_d}\right). & (7.20) \end{aligned}$$

Fault Detection

The fault detection routine uses two different tests to determine if a fault condition exists. The first test compares the magnitude of the voltage in (7.19) to user-defined thresholds. The second test compares the angle in (7.20) to the previous eight angle measurements under normal operations.

Under normal operation, the newly calculated magnitude value should lie within an interval bounded by the low and high thresholds (0.7 per unit and 1.3 per unit, respectively [59]). If the values lies outside the interval, then a corresponding fault flag is set to indicate a "Magnitude Fault".

The value Δ_{ang} is calculated as the change in the angle computed in the Park's Transformation from the average of the past eight valid samples. If the absolute value of the Δ_{ang} is larger than the specified threshold value (15 degrees [59]), the fault flag is set to "Angle Fault."

The fault status output of the high speed relay takes on the values 0, 1, 2, 3 and 4 as shown in Table 7.2.

Table 7.2: Possible values of the fault status output of the HSR routine

Fault value	**Explanation**
0	No fault conditions
1	Magnitude fault
2	Angle fault
3	Both magnitude and angle faults
4	Fault clearing

If a fault is detected on either channel of the MFM III unit, then the Fault Status column for the corresponding MFM II unit in Table 7.1 is set to 1. Also, the Fault Counter column in Table 7.1 increments until it reaches 512 counts. This counter is used as a timestamp to synchronize the MFMs as the system information matrix is sent.

If a fault clears, then the HSR routine sets the output to a fault clearing state and the Fault Reset column is set to 1. This column will remain at 1 for at least 5 ms until a fault is no longer detected then reset to 0. Once it does, the HSR routine sets its output to a no fault condition. and the Fault Status column is set to 0.

Fault Direction

If the fault detection algorithm indicates that a fault exists, the power is calculated through both input channels to determine if the fault is upstream or downstream of the MFM. The current transducers are oriented specifically during installation such that positive power values correspond to the arrows shown in Fig. 7-1.

By definition, the instantaneous power is

$$P = v_a i_a + v_b i_b + v_c i_c. \tag{7.21}$$

The MFM does not monitor the line voltages v_a, v_b, or v_c but rather v_{ab} and v_{bc} and so the expression for power must be computed in terms of the actual measured voltages. This can be accomplished by noting that the relationship between phase voltages v_a and v_b and line voltage v_{ab} is

$$v_{ab} = v_a - v_b. \tag{7.22}$$

Similarly for v_{bc} and v_{ca}

$$v_{bc} = v_b - v_c \tag{7.23}$$

and

$$v_{ca} = v_c - v_a \tag{7.24}$$

Substituting (7.22) and (7.23) into (7.21) yields

$$P = (v_{ab} + v_b)\, i_a + (v_{bc} + v_c)\, i_b + v_c i_c. \tag{7.25}$$

Noting that the current for a 3-phase electrical system must satisfy the relation

$$i_a + i_b + i_c = 0 \tag{7.26}$$

and solving for i_b, this can be substituted into (7.25) resulting in

$$P = (v_{ab} + v_{bc} + v_c)\, i_a + (v_{bc} + v_c)\,(-i_a - i_c) + v_c i_c. \tag{7.27}$$

Gathering like terms

$$P = v_{ab} i_a + v_{bc} i_a + v_c i_a - v_{bc} i_a - v_{bc} i_c - v_c i_a - v_c i_c + v_c i_c. \tag{7.28}$$

Simplifying this equation gives

$$P = v_{ab} i_a - v_{bc} i_c. \tag{7.29}$$

This gives an accurate calculation of power in the circuit assuming that the loads are equally balanced. If there is a fault on phase B, the power would not be accurate since i_b is not included in the equation. To eliminate uncertainties and provide an indication of a single phase fault a method must be used such that all measurements receive equal weight. Noting that (7.29) was arrived at by substituting for i_b in (7.25), two more equations can be found by substituting for i_a and i_c. Solving for i_a in (7.26) and substituting into (7.25) results in

$$P = (v_{ab} + v_{bc} + v_c)\,(-i_b - i_c) + (v_{bc} + v_c)\, i_b + v_c i_c. \tag{7.30}$$

Simplifying the equation results in

$$P = -v_{ab} i_b - v_{ab} i_c - v_{bc} i_c. \tag{7.31}$$

Gathering like terms results in

$$P = -v_{ab}\,(i_b + i_c) - v_{bc} i_c. \tag{7.32}$$

Similarly, solving (7.26) for i_c and substituting in to (7.25) results in

$$P = v_{ab}i_a + v_{bc}i_a + v_ci_a + v_{bc}i_b + v_ci_b - v_ci_a - v_ci_b. \tag{7.33}$$

Simplifying the equation results in

$$P = v_{ab}i_a + v_{bc}i_a + v_{bc}i_b. \tag{7.34}$$

Gathering like terms results in

$$P = v_{ab}i_a + v_{bc}\left(i_a + i_b\right). \tag{7.35}$$

Each of these three equations computes the power for the entire circuit. To arrive at an accurate power computation where all three phase currents are accounted for equally, the average of (7.29), (7.32), and (7.35) is calculated. Therefore, the instantaneous power in the circuit including all three phase currents is

$$P = \frac{v_{ab}(2i_a - i_b - i_c) + v_{bc}(i_a + i_b - 2i_c)}{3}. \tag{7.36}$$

Determine Fault Direction If no fault is detected by the fault detection routine, the power value is added to a buffer of the last sixteen power values with no faults detected. This buffer will be used to calculate the average steady state power ($\bar{P}_{SS}$) which will be used for fault direction determination when a fault is detected.

If a fault is detected, the calculated power from the HSR algorithm is used to calculate average fault power levels, $\bar{P}_{fault}$ and is subtracted from the average of the buffer to determine a Δ_P as shown in (7.37). The equation used calculate Δ_P is

$$\Delta P = \bar{P}_{fault} - \bar{P}_{SS}. \tag{7.37}$$

A power change is determined to be significant if

$$|\Delta P| \geq 1.0 \cdot \bar{P}_{SS}. \tag{7.38}$$

If the power change is significant, then the direction is considered downline if $\bar{P}_{fault}$ is greater than a defined low positive power threshold. The direction is considered upline if $\bar{P}_{fault}$ is less than a defined low negative power threshold. The direction is considered undetermined otherwise. The direction for each channel is saved in the corresponding columns in the system information matrix.

Depending on the plant configuration, some of the current transformers may measure negligible fault current resulting in no fault direction assignment. For example, if the 1SG generator is supplying all the power to the starboard load centers, a bus-tie fault between the 1SA and 2SA switchboards would result in substantial current flowing through channel 2 of Unit #2 but negligible fault current through channel 1 of Unit #4. Low fault power levels detected by the 2SA-L MFM III unit #4 may result in no fault current direction assignment. The IPCS fault isolation routines must rely on current magnitude comparisons for proper fault isolation in this example by examining current magnitudes flowing in the longitudinal 1SA and 2SA-L CTs.

The modified HSR outputs an average fault current during a fault event. If the average current magnitude falls below a threshold of 0.125 per unit, then the current magnitude flag is set to 1. If the average current magnitude exceeds a a threshold of 1.25 per unit, the current magnitude flag is set to 2. Lastly, if the average current magnitude falls between .125 and 1.25 per unit or below 0.021 per unit, the flag is set to 3. If no fault is detected, the flag is set to 0. The current magnitude flags are updated in their columns in the system information matrix.

7.2.3 Topology and Generator Line-up Assessment

The topology of the ship's electric plant is determined locally by each MFM III unit during "no fault" conditions. Each MFM III unit utilizes the circuit breaker status information received directly in the system information matrix. If the corresponding circuit breaker is closed, the MFM III unit will set the Circuit Breaker Status column in the system information matrix to 1. Otherwise, the column is set to 0.

According to the NAVSEA drawing no. 303-6567496, REV B, a "standard operating configuration for the Zonal Electrical Distribution System (ZEDS) is two generators on-line, the cross-tie breakers of the on-line generators closed, and the cross-tie breakers of the off-line generators open." Such a topology is considered a single ring. All other configurations are considered non-standard. However, for the switchboard fault isolation routines, a double ring with all cross-tie circuit breakers are closed is also considered a standard configuration. The topology assessment routine sets a standard configuration flag to 1 if true and 0 if false.

The generator line-up of the electric plant is used to determine if an MFM III unit is responsible for isolating an internal or downstream switchboard fault event. Table 7.3 shows the generator line-up for each of the eight possible states.

Table 7.3: Generator line-up for each possible state

Value	1SG	2SG	3SG
0			
1			✓
2		✓	
3		✓	✓
4	✓		
5	✓		✓
6	✓	✓	
7	✓	✓	✓

7.2.4 Fault Type Determination

The MFM III unit sends a shunt trip signal only if certain conditions are met. The first condition is that the local unit must detect a fault event as specified by the HSR routine discussed in §7.2.2. A discussion on switchboard faults and bus-tie faults are included below.

Internal/Downstream Switchboard Fault

Presently, the MFM III is unable to discriminate between a fault at a main switchboard (internal) and a fault below (downstream of) of a main switchboard. Figure 7-3 provides an illustration of an internal and downstream switchboard fault with arrows indicating power directions.

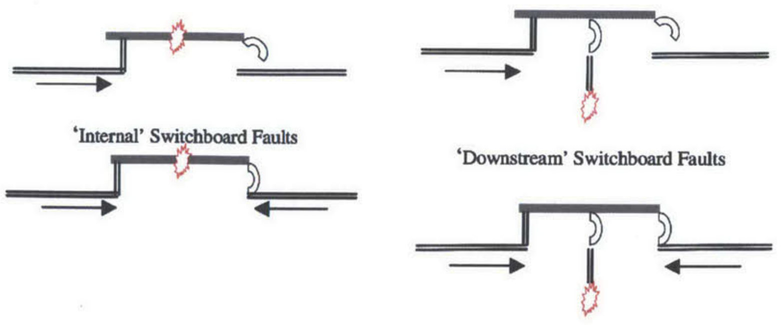

'Internal' Switchboard Faults
'Downstream' Switchboard Faults

Fig. 7-3: Examples of Internal/Downstream Switchboard Faults [59].

MFM III units 1, 2, 5, 6, 7, and 8 are responsible for generating switchboard detection flags for the 1SB, 1SA, 2SB, 2SA, 3SB, and 3SA switchboards, respectively. Switchboard fault detection for units 1, 2, 7 and 8 are straightforward. If there is a fault event, the corresponding breaker is closed, and the directions of power from the two channels indicate that power is flowing into the switchboard, then these units declare a switchboard fault. Furthermore, if the breaker is open and channel two indicates power flowing into the switchboard, the units declare a switchboard fault as well.

Switchboard fault detection for units 5 and 6 are more involved as they require additional remote information from units 3 and 4 respectively to properly declare a fault. If these units determine that the breaker is closed and the power on each local channel is flowing into the switchboard and the power in the remote unit's channel 1 is NOT flowing out of the switchboard, then a switchboard fault is declared. If these units determine that the breaker is open and the power on local channel 2 is flowing into the switchboard and the power in the remote unit's channel 1 is NOT flowing out of the switchboard, then a switchboard fault is also declared. Lastly, if the breaker of the remote unit is open, the power on the remote unit's channel is flowing into the switchboard, the power on the local unit channel 1 is flowing into in the switchboard and the magnitude of the current in local unit channel 2 is below 6000 A, a switchboard fault is declared.

For proper isolation of an internal or downstream switchboard fault event, each MFM III must know the generator line-up and plant topology prior to the fault event, the total number of switchboard faults detected by all MFM III units, and MFM I shunt trip actions anywhere in the electric plant. Internal or downstream switchboard fault isolation will only occur if the electric plant is in a standard single ring or double ring configuration, no MFM I shunt trip actions are detected, no catastrophic switchboard faults are detected and only one internal/downstream switchboard fault event is detected. The MFM I shunt trip action takes place if the MFM III unit senses currents in excess of 6000 A and/or 8000 A and must be able to establish the direction of the fault power.

Bus-Tie Faults

The IPCS algorithm allows the MFM III unit to provide proper isolation of bus-tie faults for both standard and non-standard electric plant configurations. Bus-tie fault detection can be made using both the fault power direction and circuit breaker status or solely on fault current magnitude flags for longitudinal bus-tie faults.

In order to use fault power direction for bus-tie fault detection, local and remote fault power directions must show that power is flowing out of the switchboard from both ends of the bus-tie. In order for a bus-tie fault to be detected based on comparisons of fault current magnitude flags, the bus-tie fault detection routine compares local current magnitude flags with appropriate remote current magnitude flags to determine if a longitudinal bus-tie fault exist. The current magnitude flags are set as follows:

(a) ‘1’ for current magnitudes below 600 A but above 100 A

(b) ‘2’ for current magnitudes above 6000 A

(c) ‘3’ for current magnitudes between 600 A and 6000 A or below 100 A.

If the local current magnitude flag is “1” and the remote current magnitude flag is “2” for a longitudinal bus, or if the local current magnitude flag is “2” and the remote current magnitude flag is “1” for a longitudinal bus, a bus-tie fault flag is set to “1”.

If there is a large amount of current (greater than 6000 A) entering a bus-tie but a small amount of current (less than 600 A) exiting the same bus-tie, then a fault exists somewhere on that bus-tie. Cross-tie fault detection cannot use comparisons of the current magnitude flags due to generator contributions to the fault current flowing into the cross-tie.

For longitudinal bus-tie fault events, shunt trip actions will occur if (1) currents on both end are flowing into the bus-tie, or (2) one current direction flowing into a bus-tie and an open circuit breaker at the other end of the bus-tie, or (3) the current magnitude flags are set as mentioned before. For cross-tie fault events, IPCS shunt trip actions will occur based on either (1) or (2), as shown in the figure below.

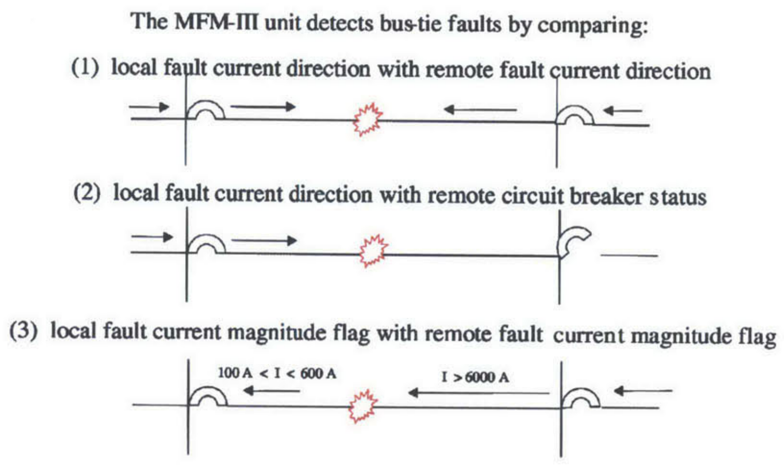

The MFM-III unit detects bus-tie faults by comparing:
(1) local fault current direction with remote fault current direction
(2) local fault current direction with remote circuit breaker status
(3) local fault current magnitude flag with remote fault current magnitude flag
100 A < I < 600 A
I >6000 A

Fig. 7-4: Possible Combinations For Bus-Tie Fault Scenarios [59].

7.3 Static Thresholds

The IPCS algorithm makes use of several user-defined thresholds. Initial thresholds used for the HSR routine for fault detection were set based on live tests conducted at both the Annapolis Detachment of the Carderock Division of the Naval Surface Warfare Center and at SPD Technologies in Philadelphia, PA [80]. The IPCS algorithm also requires that certain time thresholds are met for proper fault detection. For example, a switchboard fault must exist for at least 2 ms before an MFM III reports the fault condition. Such thresholds were set using DDG-51 Flight IIA computer simulations as stated in [59]. Table 7.4 lists several threshold parameters used by the IPCS routine. It should be noted that under normal conditions, the MFM III runs the IPCS routine every 1ms. Therefore, the NSWBDSAMPLE is set to 2 meaning that 2 samples of a switchboard fault condition are required before it is reported by the MFM III unit.

Table 7.4: Thresholds for the IPCS routine

Threshold Name	**Parameter**	**Description**
low_mag	0.7 p.u.	low cutoff threshold for voltage magnitude
high_mag	1.3 p.u.	high cutoff threshold for voltage magnitude
ang_thresh	15°	maximum deviation for voltage angle compared to reference frame in Park's transformation
NSAMPLE	2	number of sampled required by the HSR routine to detect a fault before setting direction
NBTSAMPLE	2	number of samples from the remote current transformer required to declare a bus-tie fault
NCBSAMPLE	4	number of samples from the remote circuit breaker required to declare a bus-tie fault
NSWBDSAMPLE	2	number of samples required to declare a local switchboard fault
MAX_FLTRST	5	number of voltage samples within the thresholds required to clear a fault

Reference [59] states that "future hardware studies may dictate more appropriate delays than those determined through computer simulation." Moreover, these static thresholds do not adjust based on the current operating condition of the ship. This thesis looks to incorporate the techniques used in nonintrusive load monitoring applications to dynamically adjust these thresholds for better zonal protection.

7.4 Nonintrusive Monitoring in Ring Power Systems

Chapter 2 introduced the nonintrusive load monitoring concept developed in previous research and its many applications for diagnostics. It was shown that the NILM uses the spectral envelope of the real power draw to characterize loads by their transient behavior. In other words, NILM can be implemented to determine when loads turn and off. However, the NILM devices used in these applications were meant for radial electrical distribution. By monitoring the voltages and currents upstream from all the loads, the NILM can provide diagnostic information from the aggregate electrical measurements.

Unlike in radial distributions, zonal electrical distribution systems with a ring architecture have no central monitoring point for load monitoring. Measuring the power draw of monitored loads requires additional care. In Fig. 7-1, MFM III units 1, 2, 7 and 8 are labeled as corner units as they are situated in the four corners of the ship. These devices can be readily enhanced to incorporate the NILM functionality as they monitor the currents on either side of the their respective switchboards. Let I_1 be the current measured by channel 1 of the corner MFM III units, let I_2 be the current measured by channel 2, and let I_S be the current drawn by all the loads connected to the switchboard. Kirchoff's current law (KCL) states that

$$I_1 = I_S + I_2. \tag{7.39}$$

Therefore, these corner MFM III units can easily subtract the currents from both channels to compute I_S. Doing so, units 1, 2, 7 and 8 can be used as NILM devices monitoring the loads connected to the 1SB, 1SA, 3SB, and 3SA switchboards, respectively.

The MFM-III units 3, 4, 5 and 6 cannot be immediately implemented as NILM devices as readily as the corner units. The bus-tie connected to the 2SG generator provides an additional path for current and one MFM III unit is not equipped to measure the currents in three separate paths. One possible permanent solution would be to merge the functionality of two neighboring MFM units (i.e. Unit 3 and Unit 5) into one unit that can control the cross and longitudinal circuit breakers feeding the switchboard and monitor the currents in all three paths. This way, this larger unit can use KCL to compute the current feeding all loads connected to the switchboard and be used as a NILM device.

Using the existing hardware and configuration, there are topologies in which these

MFM III units can be used as a NILM device. If the circuit breaker connected to Unit 5 is open, then Unit 3 can be used as a NILM device as Equation 7.39 holds. The reverse situation also holds. If the circuit breaker connected to Unit 3 is open, Unit 5 can be used a NILM device. Units 4 and 6 have the same relationship as do Units 3 and 5. Currently, the MFM units already share their circuit breaker status with one another as shown in §7.2.1 and no additional functionality is required. This chapter aims to demonstrate how a NILM-enhanced MFM III unit can provide better zonal protection than the current methods. Not only can NILM be used for detection of common switchboard and bus-tie faults, NILM can help with more insidious high impedance faults.

7.5 High Impedance Faults

The MFM III units currently used in zonal protection employ static thresholds for fault detection. These thresholds in concert with the over-current devices are set such that if the currents and voltages exceed certain values, then a fault is detected. Unfortunately, high impedance faults (HIF) have fault current magnitudes similar to those of normal loads. As a result, normal overcurrent protection devices such as the MFM I cannot detect and clear these faults.

Much research has been done to detect these high impedance faults by modeling their waveform characteristics [82, 83]. Other research has attempted to characterize the nonlinearity, time-varied resistance, randomness and arc features of these high impedance faults [84–89].

Figure 7-5 illustrates the waveform distortions and randomness exhibited by high impedance faults.

A specific type of high impedance fault is known as arcing, which forms in an air gap resulting from poor contact with a grounded object. There exists a high potential over the small air gap and arcing is produced when the air gap breaks down. Previous works in [90–93] have shown that the current magnitude of the arcing fault is limited by the impedance of the return ground path. Like other high impedance faults, the

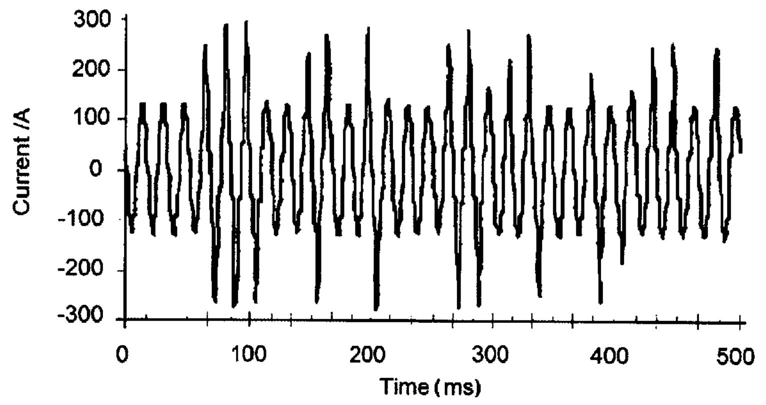


Fig. 7-5: Example high impedance fault current from Texas A&M University [85].

level of the fault is not large enough to trip over-current devices. As a result, the arcing fault escalates into further system damage, release of energy and a threat to human life [94].

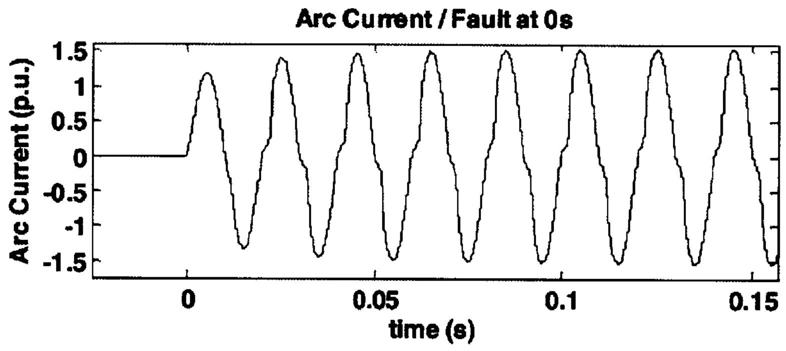


Fig. 7-6: Example arcing current [89].

An arcing waveform is shown in Fig. 7-6 and the the waveform appears to be distorted by harmonics due to the grounding resistance non-linearity. The current HSR algorithm does not analyze harmonic content for improved fault detection. Reference [95] includes a detailed discussion in modeling the higher harmonics in high impedance faults.

This chapter will look into improved performance of fault detection by a NILM-enhanced MFM III units against standard faults but also against high impedance faults by incorporating dynamic thresholding and analyzing harmonic content. A number of laboratory experiments were performed to test the improved algorithm and a summary of the setup and results are provided below.

7.6 Dynamic Thresholding

By comparing the transient responses in the spectral envelope of the power draw, the NILM-enhanced MFM III unit can keep track of the on/off state of the monitored loads. Such a characteristic can prove useful for improved zonal protection. The parameters listed in Table 7.4 were set and altered based on computer simulations of a DDG-51 [59, 80]. These static thresholds were set as such for a desired tradeoff between detection speed and false alarms [80] based on the characteristics of the system being protected.

As loads turn off and on, the characteristics of the system being protected change. Voltage thresholds used for primary fault detection may be set to take into account the largest load on the ship. If this load is already on, the previous static threshold should be lowered for faster detection speed.

The NILM-enhanced MFM III unit can keep track of loads as they turn off and on and can adjust thresholds dynamically for improved performance. To illustrate the need for such a capability, the following sections will show the impact of loads on the electric distribution system and how different thresholds can be adjusted dynamically.

As the loads turn off and on, the HSR algorithm can declare fault conditions depending on the voltage magnitude thresholds (low_mag and high_mag). It should also be noted that the voltage distortion is greater for loads that draw more power. A similar story can be said about angle distortion as will be shown in §7.7.1.

7.7 Laboratory Experiments

To demonstrate the utility for a NILM enhanced MFM unit, a test bench with two 5000 Watt synchronous generators was constructed to emulate a U.S. Navy DDG 51 FLT IIA class ship electric plant. A full discussion of the setup is included in Appendix F. As stated before, the standard operating configuration for the zonal electrical distribution system is two generators on-line [59]. Therefore, only two generators were used for the test platform even though a DDG 51 FLT IIA class ship is equipped with three generators. A picture of the test platform was shown in Fig. F-1.

Furthermore, a hardware model of a shipboard zonal electrical distribution system was built. More information on the specifications can be found in [96]. A picture of the model is shown in Fig. 7-7 and a simplified one-line diagram is shown in Fig. 7-8.

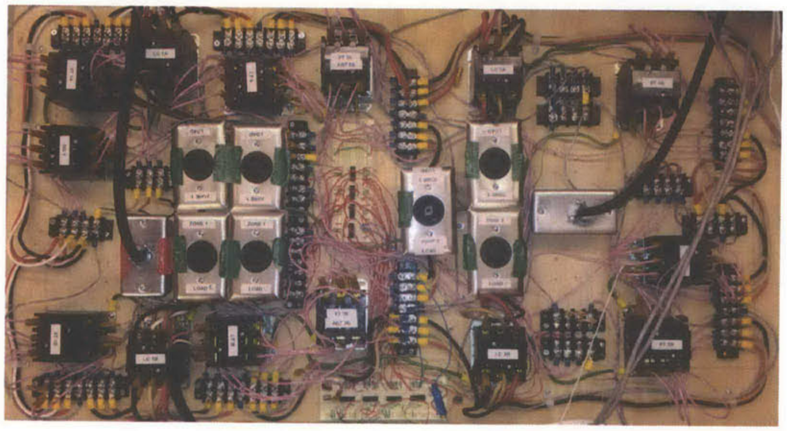


Fig. 7-7: Picture of the hardware model of a ship's ACZEDS [96].

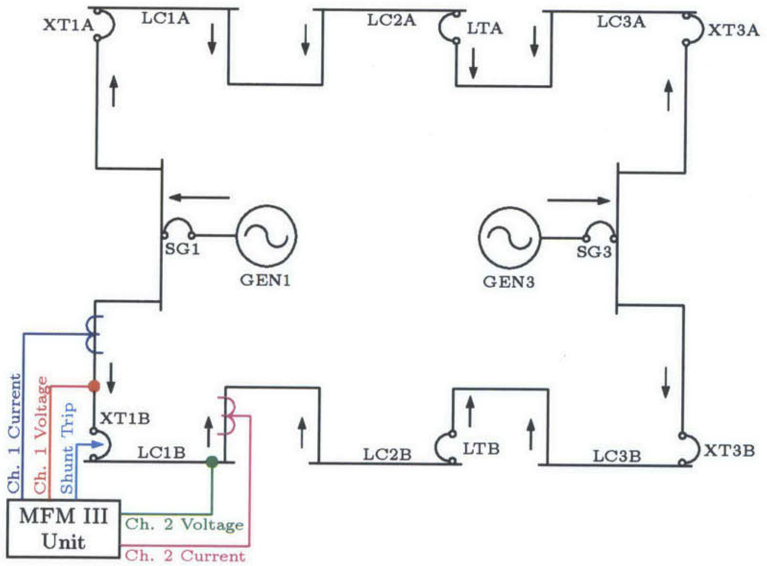


Fig. 7-8: One-line diagram of ACZEDS hardware model

A custom MFM unit was constructed as shown in Fig. 7-9. The MFM unit is equipped with two separate voltage transducer boards to measure two channels of three-phase AC voltage. There are also two sets of current transducers to measure the current in each channel. This unit is designed to be easily removed and installed anywhere in the ring bus.

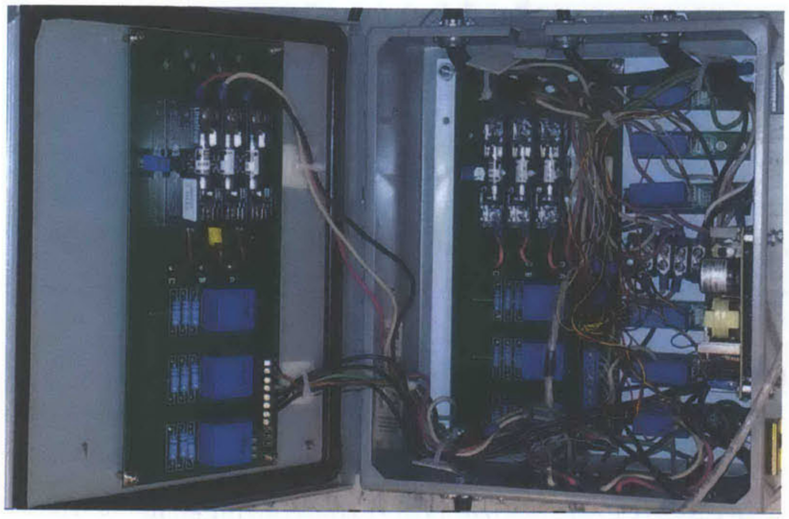


Fig. 7-9: Picture of the test MFM unit

To demonstrate the effectiveness of a NILM-enhanced MFM III unit, several experiments were conducted with various loads under certain fault conditions. Table 7.5 and Fig. 7-10 show the topology, the loads and the locations of the loads used in the laboratory experiments. Moreover, Figure 7-10 shows the MFM III unit monitoring the LC1B switchboard and the orientation used for positive power flow throughout the ACZEDS.

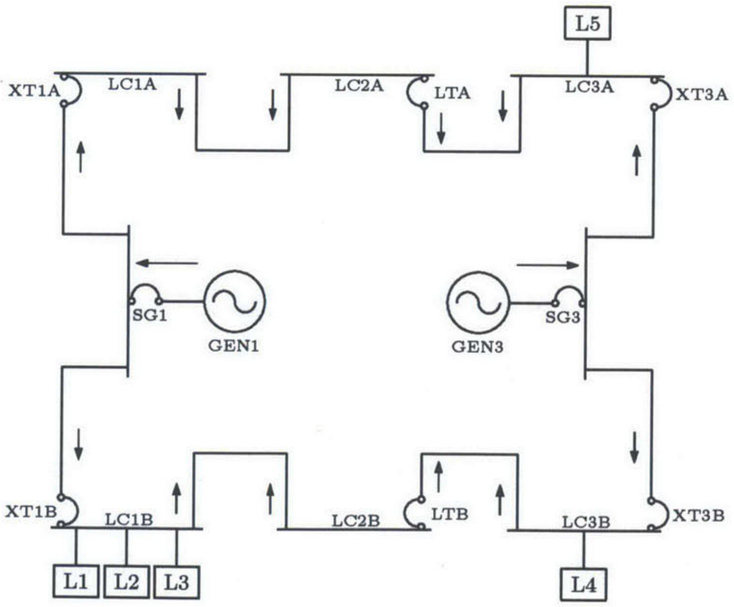


Fig. 7-10: Default topology and load configuration for laboratory experiments

Table 7.5: Loads used in laboratory experiments

Load	Switchboard	Description
L1	LC1B	225 W resistive load
L2	LC1B	1/2 HP unloaded motor
L3	LC1B	1 HP loaded motor
L4	LC1A	1/3 HP unloaded motor
L5	LC3A	120 Watt resistive load

7.7.1 Experiment I

As mentioned earlier, by monitoring the currents entering and leaving a given switchboard, a corner MFM III unit can use (7.39) to determine the aggregate current of all the loads connected to the switchboard. To demonstrate this fact, Loads L4 and L5 are connected to their assigned switchboards as provided in Table 7.5. Meanwhile, loads L1, L2, and L3 are cycled on and off in succession on the LC1B switchboard.

Figure 7-11 summarizes the results for this experiment. The top plot shows the

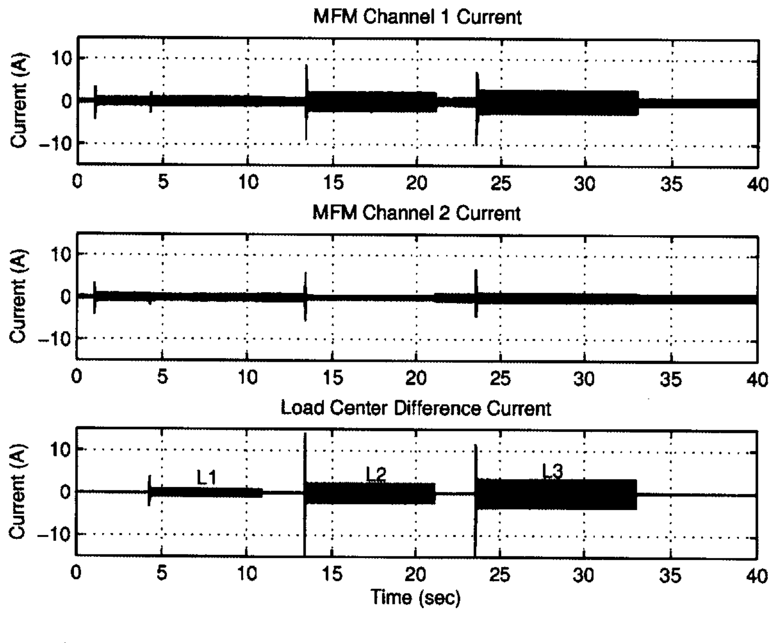


Fig. 7-11: Experiment I MFM channel currents

current in Phase A of channel 1 of the MFM unit while the middle plot shows the current in Channel 2. The difference of the two channels is shown in the bottom graph and is the aggregate current drawn by the loads connected to the switchboard. As expected, the corner MFM III can use (7.39) to monitor the aggregate current of the loads connected to the LC1B switchboard and consequently can be used as a NILM device to identify loads as they turn on and off as a NILM device. Even in a ring topology and in the presence of connected loads in other switchboards, the MFM III unit can monitor the aggregate current of loads connected to the LC1B switchboard. Furthermore, all of the algorithms and methods used by NILM devices, as explained in Ch. 2, can be applied here for this MFM unit.

The real power spectral envelope can be readily calculated using (2.1) and is plotted in Fig. 7-12. As expected, L1 is purely resistive and all of its power is real.

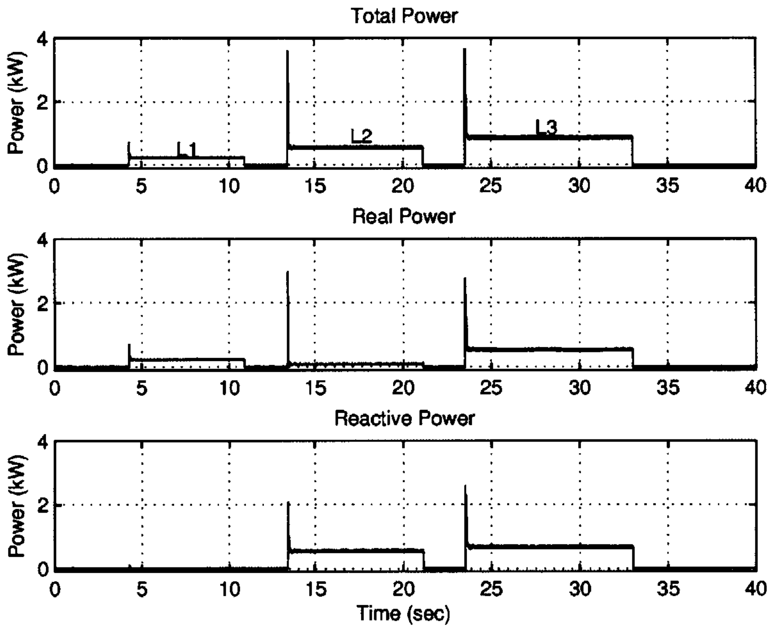


Fig. 7-12: Total, real and reactive power spectral envelopes

L2 is an unloaded motor and most of its power is reactive. L3 is a loaded motor and has a combination of real and reactive power. By analyzing the transient responses and the change in real-power, the NILM-enhanced MFM III unit can track loads as they turn on and off.

Figure 7-13 shows the voltage distortion caused by each of the three loads connected to the LC1B switchboard. Load 3, L3 is the largest load and consequently has the largest distortion. When L3 turns on, a NILM-enhanced MFM-III unit could lower the voltage magnitude thresholds accordingly based on the remaining possible loads on the monitored switchboard. If L3 goes offline, then the MFM unit can in-

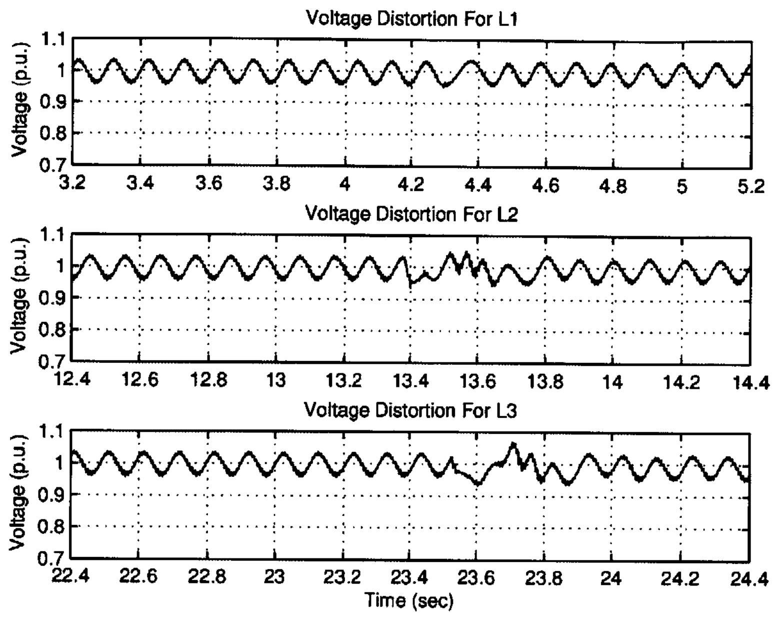


Fig. 7-13: Voltage distortion for the loads on the LC1B switchboard

crease the thresholds to the starting or default levels. As stated in [59], computer simulations can determine the appropriate threshold based on a given set of online loads.

The HSR routine also declares fault conditions based on angle measurements of the voltage. The degree of angle distortion varies according to the load. Figure 7-14 shows the angle waveform when each of the three loads turns on. Using a NILM-

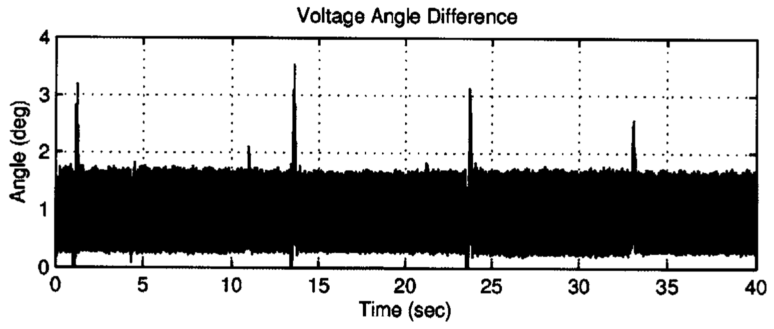


Fig. 7-14: Angle difference for the loads on the LC1B switchboard

enhanced MFM III unit, the angle threshold ang_thresh can be adjusted accordingly as loads come online.

7.7.2 Experiment II

The second experiment conducted tested the MFM's ability to detect a switchboard fault on the LC1B switchboard. In the laboratory setting, an actual fault was not used for testing purposes. Instead, faults were simulated using 50 Ω resistors to disrupt the bus voltage as was done in [80].

Figure 7-15 shows the setup used for the experiment.

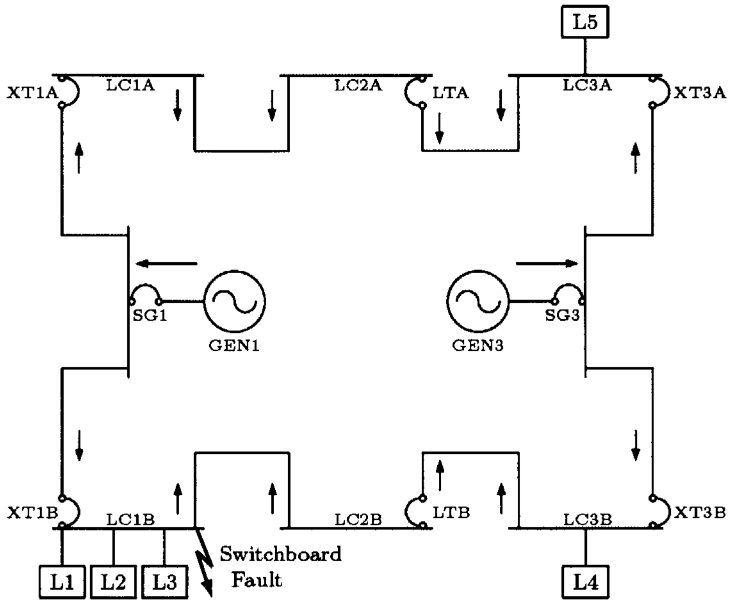


Fig. 7-15: Experiment II setup

Once again, loads L4 and L5 remained online in their respective load centers as specified by Table 7.5. Loads L1, L2, and L3 were turned on in succession. Once all loads were all on, a fault was introduced on the LC1B switchboard. Figure 7-16 shows the current drawn by all the loads on the LC1B switchboard during the entire run.

The HSR algorithm monitors the voltage for initial fault detection. In the current version of the HSR relay, a fault is detected if the voltage magnitude or angle exceeds predefined static thresholds. Figure 7-17 plots the per unit voltage supplied to the LC1B load center and the static thresholds. The voltage undergoes a minor disruption when each of three connected loads turns on but the magnitude remains within the thresholds. When the fault is simulated at $t = 20.4$, the voltage drops below the

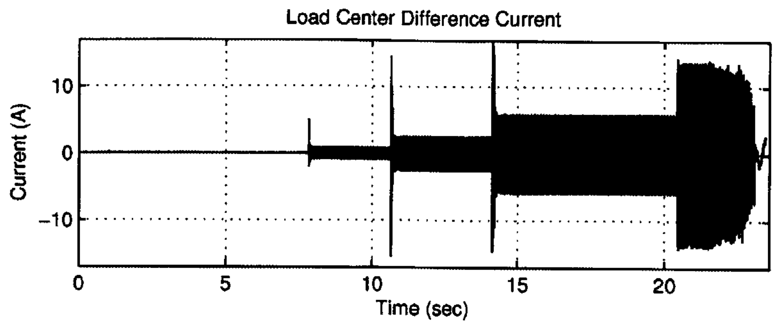


Fig. 7-16: Current drawn by the loads connected to the LC1B switchboard.

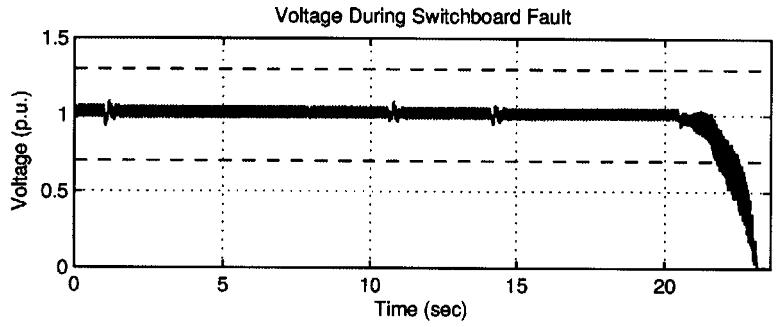


Fig. 7-17: Per unit voltage magnitude

lower static threshold and the magnitude fault detection flag is set by the MFM.

Figure 7-18 plots the Park's transformation angle of the voltage measurements during this experiment.

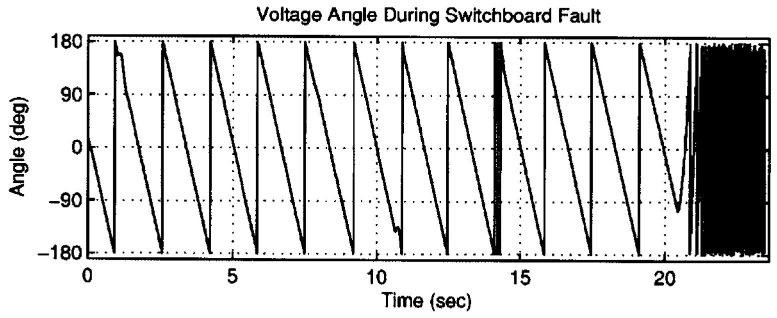


Fig. 7-18: Angle difference between voltage and Park's transformation rotating reference when L_1 comes online

Under normal conditions, the angle difference ϕ_d can be represented as

$$\phi_d = -\omega_d t, \tag{7.40}$$

where ω_d is the difference in frequency between the voltage and the rotating reference frame.

As stated in §7.2.2, the HSR routine compares the angle with the average of the past eight valid samples for angle fault detection. In Fig. 7-18, each load causes an abnormality in the angle waveform. The degree of angle distortion varies according to the load.

Figure 7-19 plots the absolute difference between each angle measurement and the mean of the previous eight valid samples. This metric is used to determine if the

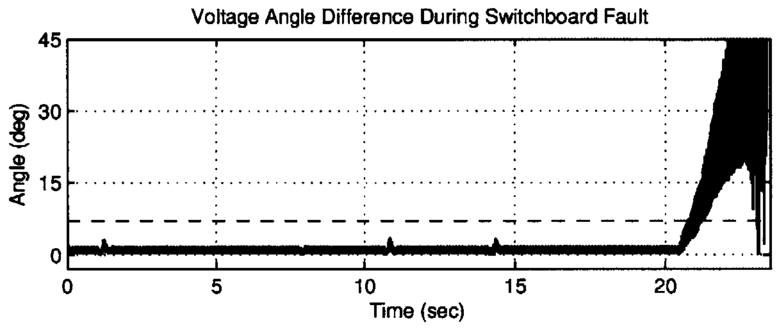


Fig. 7-19: Absolute angle difference

angle fault detection flag should be set. The lower and upper angle thresholds are also shown in the figure. It should be noted that depending on the load, the angle distortion may be large enough to set a fault detection flag when the load comes on. Such faults should clear once the load has reached steady state.

Once a fault has been detected, the IPCS routine begins and each MFM unit will try to locate the fault. Figure 7-20 plots the fundamental real power in each channel of the MFM unit.

After time $t = 20.4$ when the fault is simulated, channel 1 on the MFM unit indicates a downstream power flow, while channel 2 indicates an upstream power flow. This combination for a corner MFM unit suggests that power is flowing into the switchboard and consequently a local switchboard fault is detected by the MFM unit.

For this experiment, the simulated fault is sufficiently severe that the voltage magnitude and angle exceed the predefined thresholds and the IPCS routine can

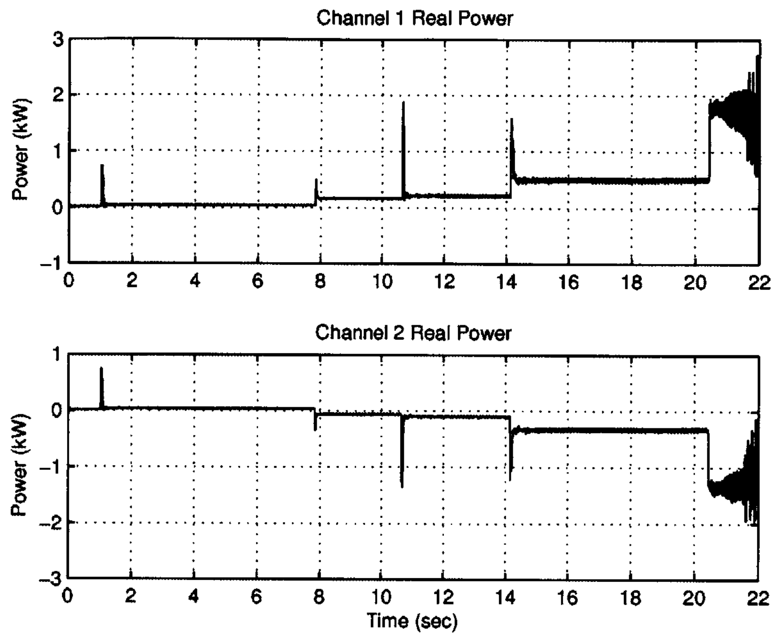


Fig. 7-20: Fundamental real power in both MFM channel for experiment II

then begin to isolate the fault area. The experiments discussed in §7.7.4 show how high impedance faults have fault current magnitudes similar to normal loads and will not be recognized as faults.

7.7.3 Experiment III

The third experiment conducted tested the MFM's ability to detect a bus-tie fault between the LC1A and the LC1B switchboards.

Figure 7-21 shows the setup used for the experiment.

With loads L4 and L5 on, loads L1, L2 and L3 were turned on in succession. Once all loads were on, the 50 Ω fault was introduced on the bus-tie as indicated in Fig. 7-21. Figure 7-22 shows the current drawn by all the loads on the LC1B switchboard during the entire run.

Figure 7-23 plots the per unit voltage supplied to the LC1B load center during the bus-tie fault run. As in §7.7.2, the voltage undergoes a minor disruption when each of three connected loads turns on but the magnitude remains within the thresholds. When the fault is simulated at t = 19.2, the voltage drops below the lower static

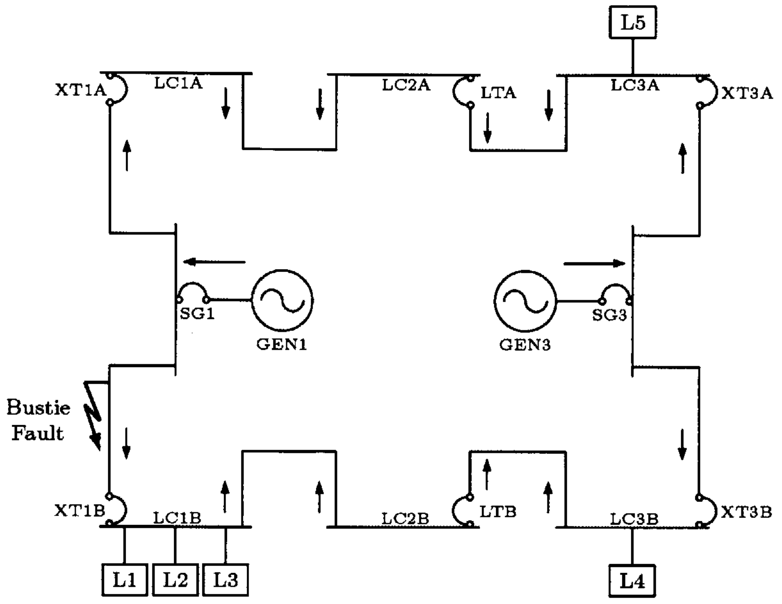


Fig. 7-21: Experiment III setup

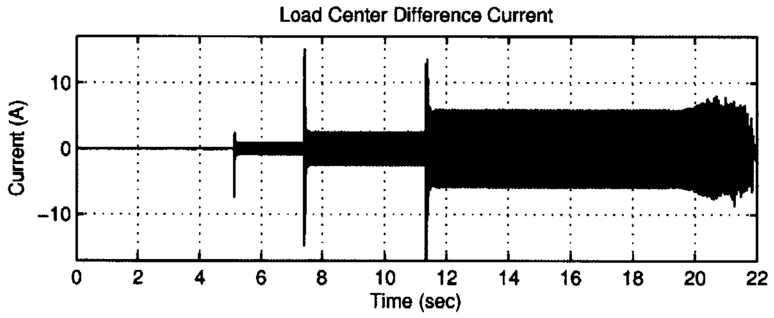


Fig. 7-22: Current drawn by the loads connected to the LC1B switchboard.

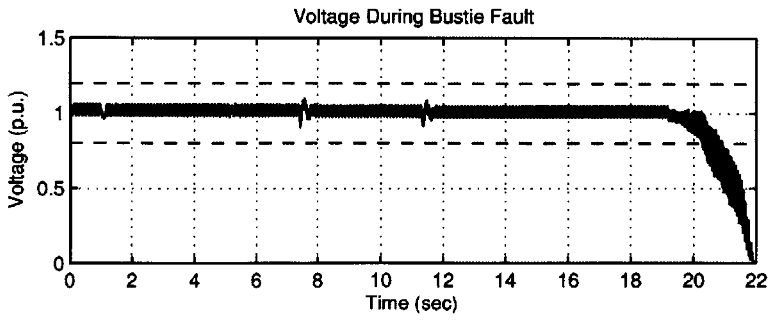


Fig. 7-23: Per unit voltage magnitude

threshold and the magnitude fault detection flag is set by the MFM.

Figure 7-24 plots the Park's transformation angle of the voltage measurements during this experiment. Moreover, the absolute difference between each angle mea-

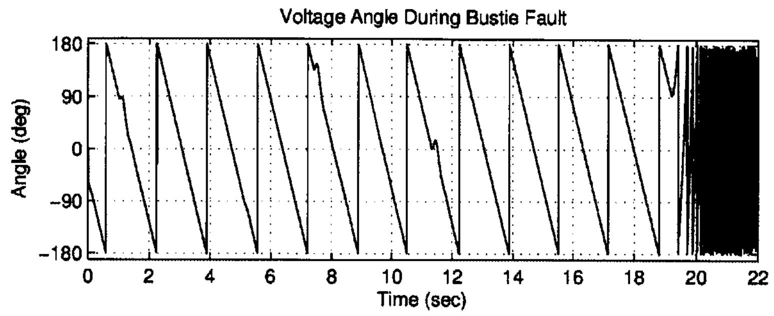


Fig. 7-24: Angle difference between voltage and Park's transformation rotating reference when L_1 comes online

surement and the mean of the previous eight valid samples is shown in Fig. 7-25. The lower and upper angle thresholds are also shown in the figure.

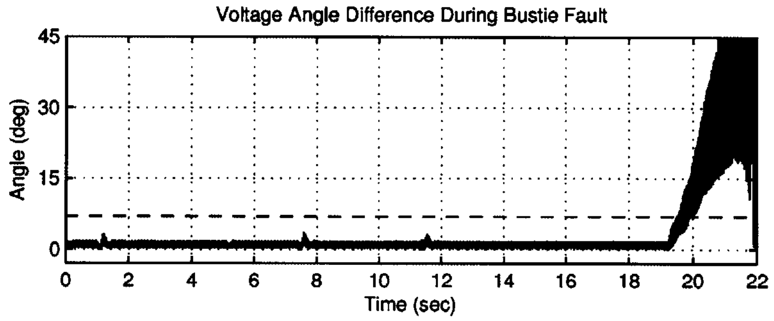


Fig. 7-25: Absolute angle difference

Once the fault has been detected, the IPCS routine tries to locate the fault. Figure 7-26 plots the fundamental real power in each channel of the MFM unit After time $t = 19.2$ when the fault is simulated, both channel 1 and channel 2 on the MFM unit indicate an upstream power flow. This combination for the MFM unit suggests that the fault is not a local switchboard fault. The fault is either on another switchboard or on a bus-tie.

For this set of laboratory experiments, a system information matrix was simulated for test purposes. In this run, an test MFM on the LC1A switchboard was simulated

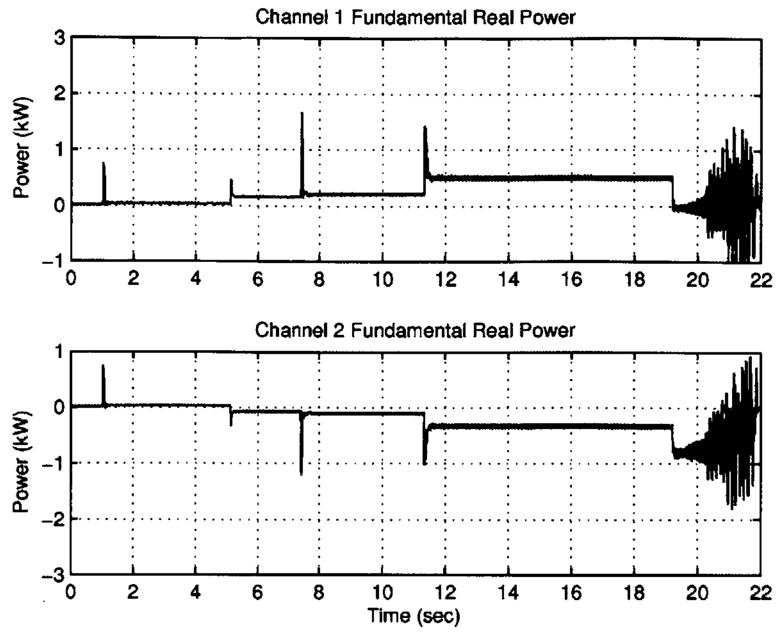


Fig. 7-26: Fundamental real power in both MFM channel for experiment III

to indicate fault power upstream from both of its channels. Using this remote information, the MFM unit on the LC1B can correctly identify the fault as a bus-tie fault.

7.7.4 Experiment IV

As stated earlier, high impedance faults have fault current magnitudes similar to those of normal loads. Such faults are undetected by the normal overcurrent protection limits and can lead to damaging effects.

The high impedance fault shown in Fig. 7-5 was simulated in the laboratory setting using the schematic shown in Fig. 7-27.

When the fault is energized, the first switch remains closed. The second switch closes to provide a pattern similar to that shown in Fig. 7-5. Figure 7-28 shows the resulting current of the high impedance fault load.

In this experiment, the high impedance fault load was attached to the LC1B switchboard as shown in Fig. 7-29.

With loads L4 and L5 on, loads L3 and L2 were turned on in succession. Once

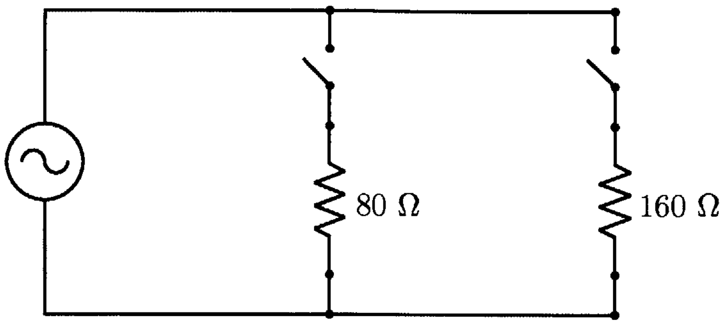


Fig. 7-27: High impedance fault schematic

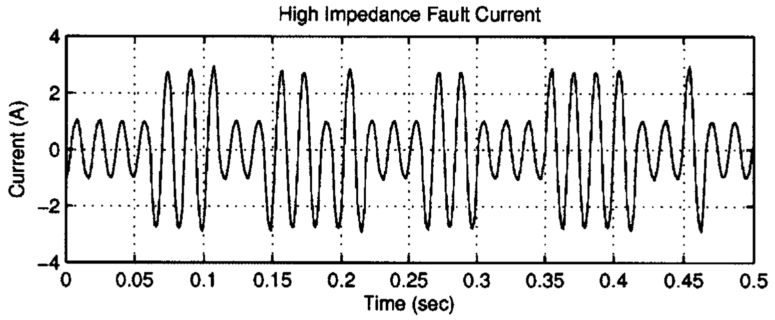


Fig. 7-28: High impedance fault current

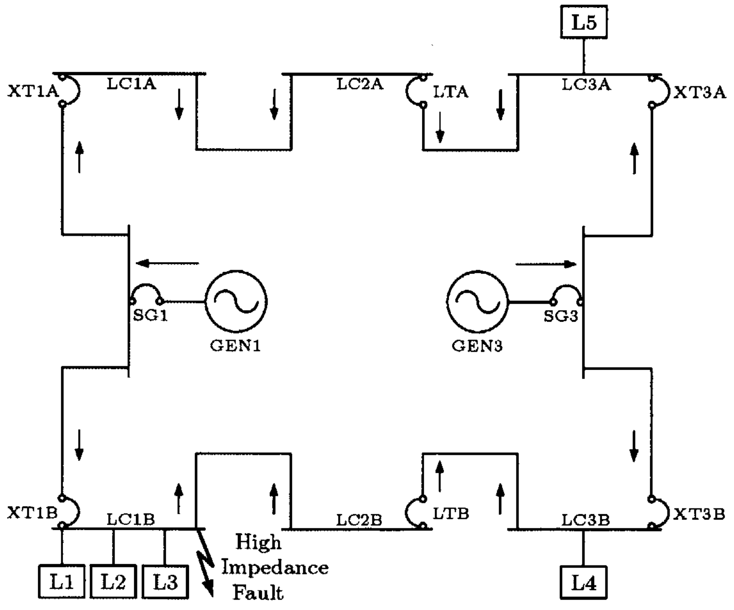


Fig. 7-29: Experiment IV setup

these loads were on, the high impedance fault load was energized. Figure 7-30 shows the current drawn by all the loads on the LC1B switchboard during the entire run.

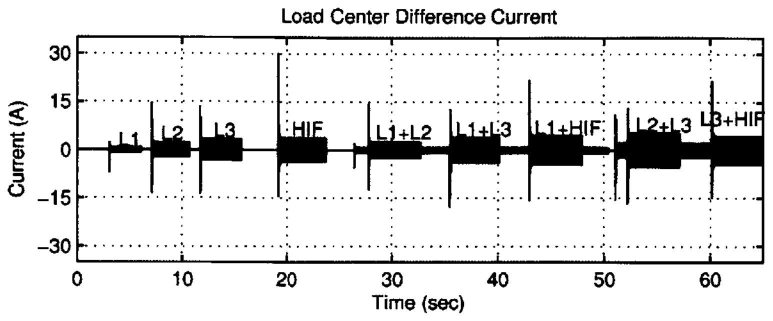


Fig. 7-30: Current drawn by the loads connected to the LC1B switchboard.

Figure 7-31 shows the total power drawn by the loads in the LC1B switchboard. The erratic high impedance fault current will not exceed the threshold set by the

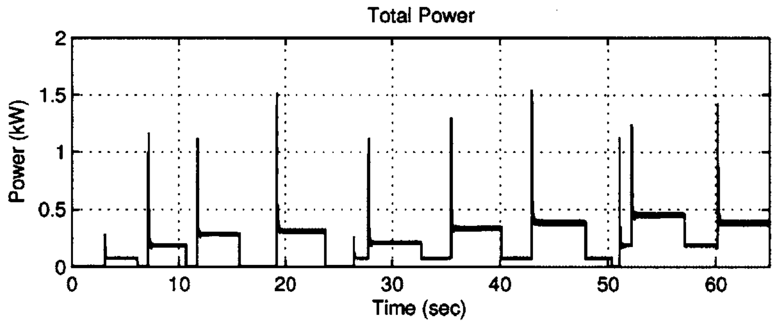


Fig. 7-31: Total power draw on LC1B switchboard

overcurrent relay. As the MFM recognizes loads by steady-state power, the threshold for transient voltage angle difference adapts so that it is above the largest transient difference of any load not turned on. Figure 7-32 shows the voltage angle difference during the trial run.

The resulting shunt trip signal is shown in Fig. 7-33. When the transient angle difference crosses the threshold, the shunt trip is set until the high impedance fault is turned off.

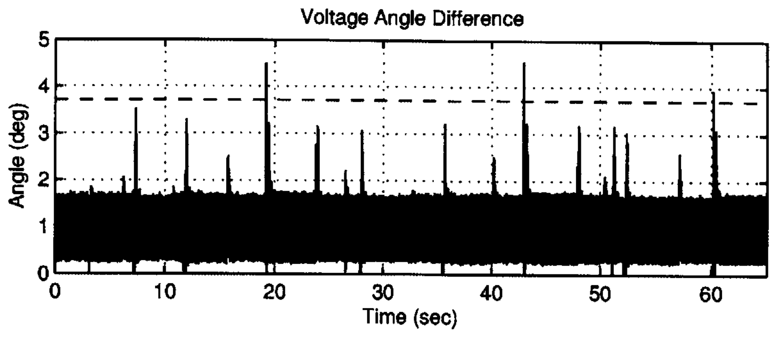


Fig. 7-32: Angle difference for the loads on the LC1B switchboard for experiment IV

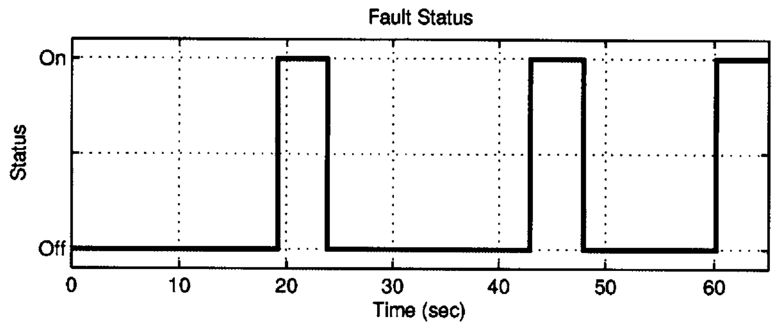


Fig. 7-33: Shunt trip signal for MFM unit for experiment IV

7.7.5 Experiment V

The NILM-enhanced MFM III unit can also protect against arcing faults shown in Fig. 7-6. Figure 7-34 shows the schematic used to simulate an arcing fault and the current waveform is shown in Fig. 7-35.

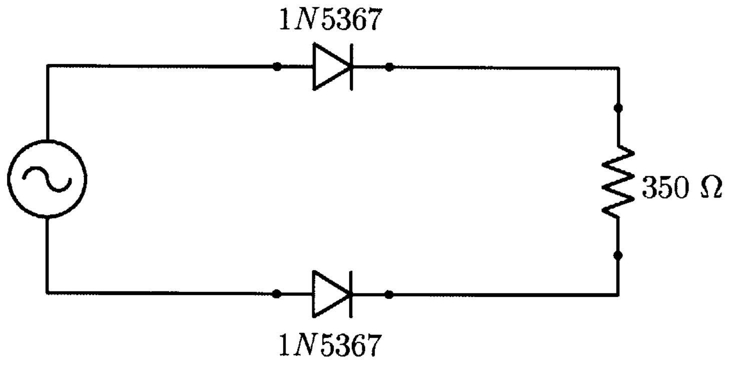


Fig. 7-34: Arcing fault schematic [92]

Even though arcing faults have current levels similar to that of normal loads, the distinct harmonic content can provide a tool for fault protection. In this experiment,

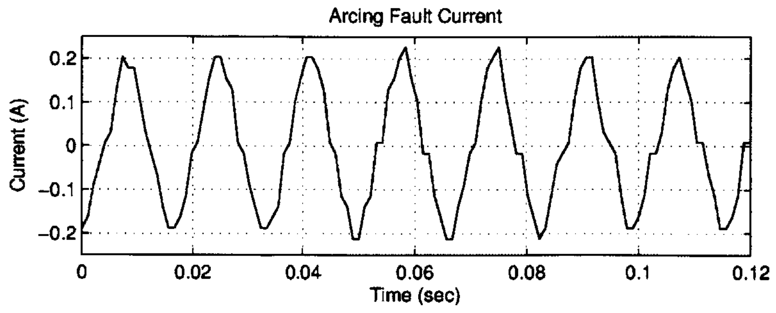


Fig. 7-35: Arcing fault current

each load on the LC1B switchboard is a balanced three phase load so there is no third harmonic content. Therefore, the MFM III unit can monitor third harmonic content in the voltage and detect faults based on a fixed threshold.

With loads L4 and L5 on, loads L3 and L2 and the arcing fault were turned on in sequence on the LC1B switchboard. Figure 7-36 shows the current drawn by all the loads on the LC1B switchboard during the entire run.

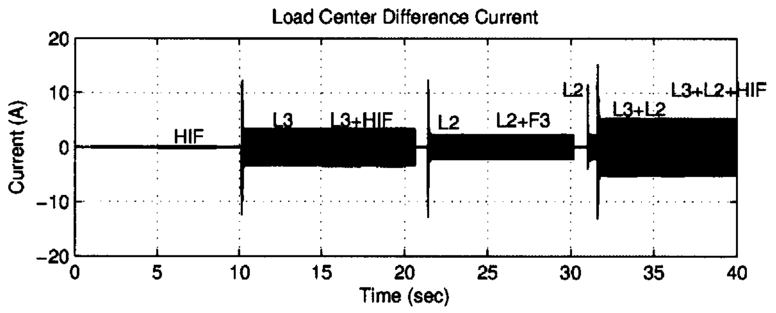


Fig. 7-36: Current drawn by the loads connected to the LC1B switchboard.

The graph in Fig. 7-37 shows the third harmonic content in the voltage during the run. When the arcing fault is on, there is substantial harmonic content. There

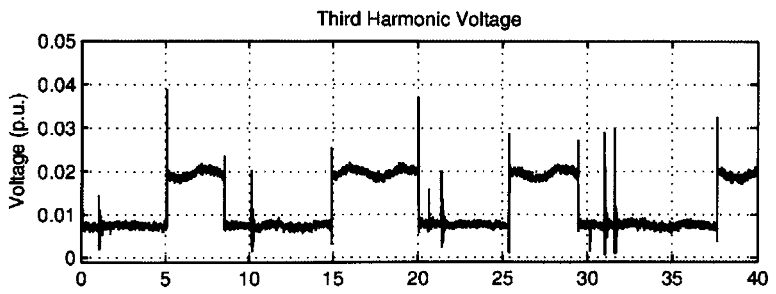


Fig. 7-37: Third harmonic content in the voltage for experiment V

is also harmonic content during the transients when loads turn on and off. A time threshold is incorporated to ensure that the harmonic content persists long enough to detect a fault. The resulting shunt trip signal is shown in Fig. 7-38.

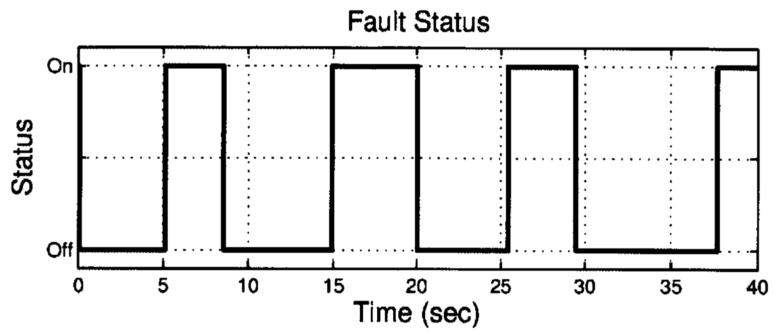


Fig. 7-38: Shunt trip signal for MFM unit for experiment V

7.8 Conclusion

This chapter presented a detailed description in the fault detection algorithms employed by the MFM III units in the DDG-51 Flight IIA class ships. A testbench simulation was built in the laboratory to simulate the behavior of the electric generation plant and the zonal electrical distribution system of a DDG-51. Laboratory results show that an MFM III unit can be enhanced with NILM capabilities to improve performance of the fault detection algorithm by dynamically setting thresholds and analyzing harmonic content as identifiable loads come online and offline.

Chapter 8

Conclusion and Future Work

Smart Grid technology is one of many potential instruments used to keep the power system ready for higher electricity demand in the future. A "smart grid" relies on the extensive communication network that can direct necessary and useful information to utilities and consumers. The smart metering device should be not only provide the required information such as power consumption but may be extended to provide diagnostic information about the loads at the metered site. Nonintrusive load monitoring (NILM) has been demonstrated as a tool that can be implemented to extract diagnostic parameters from electrical signals.

This thesis proposed several techniques and methods for increasing the maximizing the utility of nonintrusive load monitors. The specific contributions include developing an FFT-based algorithm to locate and track small harmonics in power measurements. These harmonics provide useful diagnostic information such as speed for sensorless control application and airflow estimation and vibration monitoring.

Nonintrusive load monitors also provided the raw data from which behavioral models can be developed for improved power system design. As electrical distribution systems become more complex in the future, design tools should evolve to meet the increased demands needed by designers. A framework was developed to show how simple models can be created to simulate the power demands of a microgrid such as the zonal electrical distribution systems in the DDG-51 class ships.

Finally, these nonintrusive load monitors have been used primarily for radial distribution systems. The zonal electrical distribution systems such as those found in the DDG-51 require special care in order to incorporate the methods developed in the past using NILM data. Moreover, these NILM data can be used in conjunction with the multi-function monitors presently found in the distribution systems to provide enhanced protection from various faults.

The results obtained from this thesis can be extended into different lines of research. In the longer-term, developing a framework tool for the design of an electric grid that expanded the capability to model the ship in a simpler manner would be beneficial. Developing this program as a user-friendly program would move the modeling of the electric plant from using spreadsheets to be a more interactive graphical approach. In developing the behavioral model, a significant amount of future work exists. More extensive field study research is needed to model the behavior of classified loads. As these characteristics are included in future iterations of the design tool, the higher order aggregate results of a switchboard or an entire ship will become more accurate.

In the protection of the zonal electrical distribution system, only two types of high impedance faults were considered to show the efficacy of a NILM-enhanced multi-function unit. As more research is done in analyzing and modeling faults in electrical distribution systems, new algorithms may be needed in isolating these faults.

Appendix A

Derivation of Rotor Slot Harmonics

The literature on principal slot harmonics mostly focuses on how these harmonics show up in three phase motors [26, 28, 29, 97–99]. In the first two sections, the frequency locations of the slot harmonics for single phase motors will be derived. The first derivation will correspond to capacitor-start motors. For these types of motors, the transient start-up motor current will be ignored and only the steady state slot harmonics will be considered. The second derivation will correspond to capacitor-run motors. Last, a derivation of the slot harmonics in three-phase motors is reproduced.

A.1 Capacitor-Start Motors

For single-phase motors, an auxiliary winding is added that is orthogonal to the main winding, shown in Fig. A-1. The combination of these two phases provides the torque required to start the motor. Once the motor is running, the auxiliary winding is later disconnected, usually by means of a centrifugal switch.

This section will derive the location of the slot harmonics once the auxiliary winding has been disconnected and the motor has reached steady state. To develop the model needed to compute the slot harmonics, consider the cartoon sketch of a cross section of an induction motor with concentrated windings in Fig. A-2(a).

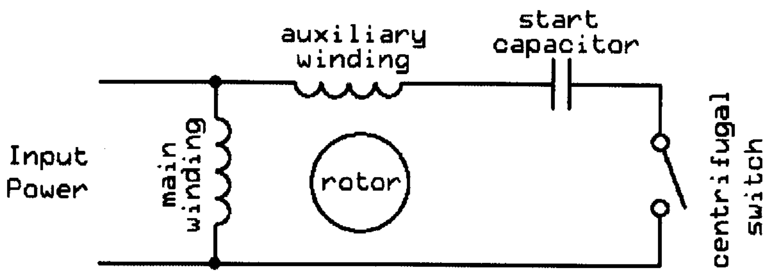
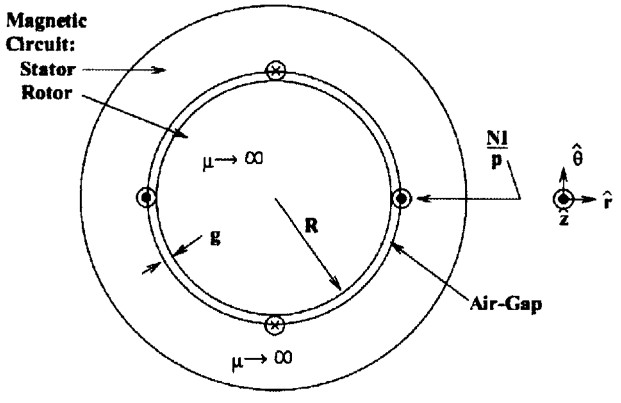
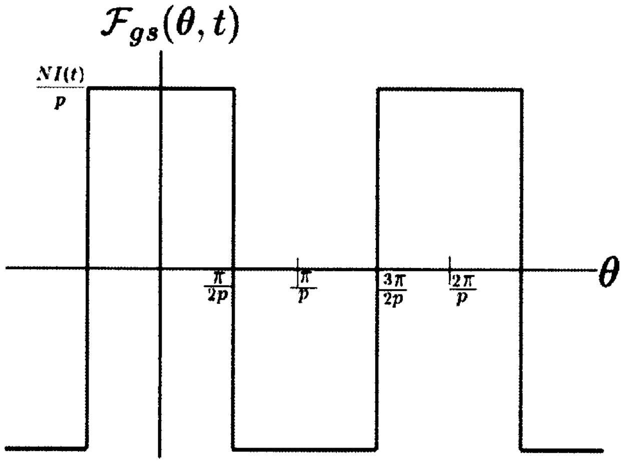


Fig. A-1: Circuit diagram of a capacitor start motor.



(a)



(b)

Fig. A-2: Cartoon sketch of an induction motor with concentrated winding in (a). The corresponding MMF wave is shown in (b). Figures adapted from [100].

The magnetomotive force (MMF) wave in the airgap in the stator reference $\mathcal{F}_{gs}(\theta,t)$ in Fig. A-2(b) can be expanded using Fourier series.

$$\mathcal{F}_{gs}(\theta,t) = \sum_{\substack{n=1 \\ n\ odd}}^{\infty} \frac{4}{n\pi}\frac{NI(t)}{p} cos(np\theta), \tag{A.1}$$

where N is the number of turns in the winding, $I(t)$ is the stator current and p is the number of pole pairs. This MMF wave Fourier series expansion assumes a concentrated winding. A corresponding series expansion for a distributed winding pattern can be computed by introducing a winding factor $k_{wn} = k_{pn}k_{bn}$ which is a product of the pitch factor k_{pn} and the breadth factor k_{bn}.

As mentioned, this derivation for a capacitor-start single-phase motor assumes that the auxiliary winding has been disconnected and the motor has reached steady state. The current in the main winding can then be expressed as

$$I(t) = I_o cos(\omega t), \tag{A.2}$$

which is substituted in (A.1). The MMF wave $\mathcal{F}_{gs}(\theta,t)$ can now be expressed as

$$\mathcal{F}_{gs}(\theta,t) = \sum_{\substack{n=1 \\ n\ odd}}^{\infty} f_{n+}\cos(np\theta - \omega t) + f_{n-}\cos(np\theta + \omega t), \tag{A.3}$$

where f_{n+} is the amplitude of the forward traveling MMF wave and f_{n-} is the amplitude of the backward traveling MMF wave. The main and auxiliary windings create a torque which specifies the motor's direction of rotation before the auxiliary winding is disconnected. Therefore, the forward and backward traveling waves have different amplitudes where the wave traveling in the same direction in which the motor rotates has the higher amplitude.

To express the MMF wave in the same reference frame of the rotor, the following expression is made:

$$p\theta = p\theta_r + \omega_m t, \tag{A.4}$$

where θ is the physical position of the stator, θ_r is the position of the rotor, and ω_m is the electrical frequency of the rotation of the rotor. Lastly, $\omega_m = p\Omega_m$ where Ω_m is the physical rotational frequency of the rotor. For simplicity, the rest of this analysis will be done for a given space harmonic number n. Substituting $p\theta$ in (A.3) to reflect the rotor's reference frame, the MMF wave for a given space harmonic n, $\mathcal{F}_{grn}$, is as follows:

$$\mathcal{F}_{grn} = f_{n+} \cos\left(np\theta_r + n\omega_m t - \omega t\right) + f_{n-} \cos\left(np\theta_r + n\omega_m t + \omega t\right). \tag{A.5}$$

$$\begin{aligned} &= f_{n+} \cos\Big(np\theta_r + (n-1)\omega_m t - (\omega - \omega_m)t\Big) \\ &\quad + f_{n-} \cos\Big(np\theta_r + (n-1)\omega_m t + (\omega + \omega_m)t\Big). \end{aligned} \tag{A.6}$$

The relationship between the stator current frequency ω and the electrical frequency of the rotation of the rotor ω_m is $\omega_m = (1-s)\omega$, where s is the slip. This relationship implies that $\omega - \omega_m = s\omega$ and $\omega + \omega_m = (2-s)\omega$. Substitutions in (A.6) yield

$$\begin{aligned} \mathcal{F}_{grn} &= f_{n+} \cos\Big(np\theta_r + (n-1)\omega_m t - s\omega t\Big) \\ &\quad + f_{n-} \cos\Big(np\theta_r + (n-1)\omega_m t + (2-s)\omega t\Big). \end{aligned} \tag{A.7}$$

So far, the magnetic flux wave in the air gap has been assumed to be sinusoidal and the induction machine is perfectly symmetrical. However, rotor slot harmonics and static and dynamic eccentricity modulations are generated by a non-uniform magnetic flux wave caused by the presence of rotor slots or imbedded conductor bars on the rotors.

From a fixed point on the stator, the effective distance in the airgap varies as the rotor rotates which disturbs the average permeance across the airgap. If there are R slots, the permeance will have R cycles of variation. If this variation is modeled to be sinusoidal, the effect of the rotor slots multiplies the average permeance $\mathcal{P}_o$ by $1 + \alpha\cos(R\theta_r)$, where $\alpha < 1$. The stator slots also introduce a permeance wave but this wave is stationary [97, Ch. 9]. The derivation ignores the effect of the stator slots since it is negligible when viewed from the stator. The permeance, including only the effect of the rotor slots, is described by the following expression:

$$\mathcal{P} = \mathcal{P}_o \left(1 + \alpha \cos\left(R\theta_r\right)\right). \tag{A.8}$$

Multiplying the permeance $\mathcal{P}$ by the MMF $\mathcal{F}_{grn}$ gives an expression for the flux Φ_{grn} as seen by the rotor.

$$\begin{aligned}
\Phi_{grn} = {} & A_{n+} \cos\left(np\theta_r + (n-1)\omega_m t - s\omega t\right) \\
& + A_{n-} \cos\left(np\theta_r + (n-1)\omega_m t + (2-s)\omega t\right) \\
& + B_{n+} \cos\left((R+np)\theta_r - s\omega t + (n-1)\omega_m t\right) \\
& + B_{n+} \cos\left((R-np)\theta_r + s\omega t - (n-1)\omega_m t\right) \\
& + B_{n-} \cos\left((R+np)\theta_r + (2-s)\omega t + (n-1)\omega_m t\right) \\
& + B_{n-} \cos\left((R-np)\theta_r - (2-s)\omega t - (n-1)\omega_m t\right),
\end{aligned} \tag{A.9}$$

where $A_{n+} = \mathcal{P}_o f_{n+}$, $A_{n-} = \mathcal{P}_o f_{n-}$, $B_{n+} = \frac{1}{2}\alpha\mathcal{P}_o f_{n+}$ and $B_{n-} = \frac{1}{2}\alpha\mathcal{P}_o f_{n-}$. The previous expression makes use of trigonometric identity $2\cos A \cos B = \cos(A+B) + \cos(A-B)$. To compute the flux Φ_{gsn} as seen from the stator, the following substitution is made for θ_r by rearranging (A.4):

$$\theta_r = \frac{p\theta - \omega_m t}{p}. \tag{A.10}$$

$$\begin{aligned}\Phi_{gsn} = {} & A_{n+}\cos\left(np\frac{p\theta-\omega_m t}{p}+(n-1)\omega_m t - s\omega t\right)\\ & + A_{n-}\cos\left(np\frac{p\theta-\omega_m t}{p}+(n-1)\omega_m t + (2-s)\omega t\right)\\ & + B_{n+}\cos\left((R+np)\frac{p\theta-\omega_m t}{p} - s\omega t + (n-1)\omega_m t\right)\\ & + B_{n+}\cos\left((R-np)\frac{p\theta-\omega_m t}{p} + s\omega t - (n-1)\omega_m t\right)\\ & + B_{n-}\cos\left((R+np)\frac{p\theta-\omega_m t}{p} + (2-s)\omega t + (n-1)\omega_m t\right)\\ & + B_{n-}\cos\left((R-np)\frac{p\theta-\omega_m t}{p} - (2-s)\omega t - (n-1)\omega_m t\right). \end{aligned} \tag{A.11}$$

Simplifying the expression yields the following,

$$\begin{aligned}\Phi_{gsn} = {} & A_{n+}\cos\left(np\theta-\omega_m t - s\omega t\right)\\ & + A_{n-}\cos\left(np\theta-\omega_m t + (2-s)\omega t\right)\\ & + B_{n+}\cos\left((R+np)\theta - \frac{R}{p}\omega_m t - s\omega t - \omega_m t\right)\\ & + B_{n+}\cos\left((R-np)\theta - \frac{R}{p}\omega_m t + s\omega t + \omega_m t\right)\\ & + B_{n-}\cos\left((R+np)\theta - \frac{R}{p}\omega_m t + (2-s)\omega t - \omega_m t\right)\\ & + B_{n-}\cos\left((R-np)\theta - \frac{R}{p}\omega_m t - (2-s)\omega t + \omega_m t\right). \end{aligned} \tag{A.12}$$

Making the substitution $\omega_m = (1 - s)\omega$ and rearranging terms,

$$\begin{aligned}\Phi_{gsn} &= A_{n+} \cos(np\theta - \omega t) \\ &+ A_{n-} \cos(np\theta + \omega t) \\ &+ B_{n+} \cos\left((R + np)\theta - \left(R\frac{1-s}{p} + 1\right)\omega t\right) \\ &+ B_{n+} \cos\left((R - np)\theta - \left(R\frac{1-s}{p} - 1\right)\omega t\right) \\ &+ B_{n-} \cos\left((R + np)\theta - \left(R\frac{1-s}{p} - 1\right)\omega t\right) \\ &+ B_{n-} \cos\left((R - np)\theta - \left(R\frac{1-s}{p} + 1\right)\omega t\right). \end{aligned} \tag{A.13}$$

The slot harmonics, when only considering the fundamental harmonic in the stator current and noting that $\omega = 2\pi f$, are therefore located at

$$f_{sh} = f\left(R\frac{1-s}{p} \pm 1\right), \tag{A.14}$$

where f is the line frequency of the stator current. The derivation, thus far, only included the fundamental component of the rotor slot permeance variation in Eq. (A.8). However, it may be necessary to consider the second and higher harmonics [97, Ch. 9]. The complete expression for the permeance would be

$$\mathcal{P} = \mathcal{P}_o\left(1 + \sum_{k=1}^{\infty} \alpha_k \cos(kR\theta_r)\right). \tag{A.15}$$

Including this k harmonic number yields the following expression for the slot harmonics.

$$f_{sh} = f\left(kR\frac{1-s}{p} \pm 1\right). \tag{A.16}$$

Furthermore, this derivation only included the first time harmonic in the stator current in (A.2). In general, the stator current will include the higher odd harmonics so that

$$I(t) = I_o \sum_{\substack{\nu=1 \\ \nu \ odd}}^{\infty} \cos(\nu\omega t). \tag{A.17}$$

Adding the time harmonics in the stator current will lead to the following expression for the slot harmonics.

$$f_{sh} = f\left(kR\frac{1-s}{p} \pm \nu\right). \tag{A.18}$$

Lastly, the dynamic eccentrictity effects must be taken into account to fully capture all of the slot harmonics.

With each rotation of the rotor, the eccentricity changes the airgap distance between the stator and rotor. This results in modulating the permeance described in (A.15) as

$$\begin{aligned}\mathcal{P} &= \mathcal{P}_o\left(1 + \sum_{k=1}^{\infty} \alpha_k \cos(kR\theta_r)\cos(n_d\theta_r)\right) \\ &= \mathcal{P}_o\left(1 + \sum_{k=1}^{\infty} \frac{\alpha_k}{2} \cos\left((kR \pm n_d)\theta_r\right)\right)\end{aligned} \tag{A.19}$$

where n_d is the harmonic order of eccentricity.

Following the algebraic manipulations shown earlier, the new expression for the slot harmonics is

$$f_{sh} = f\left[(kR + n_d)\frac{1-s}{p} + \nu\right] \tag{A.20}$$

where f is the supply frequency; $k = 0, 1, 2...$; R is number of rotor slots; $n_d = 0, \pm1, ...$ is the order of rotor eccentricity; s is the per unit slip, p is the number of pole pairs and $\nu = \pm1, \pm3, ...$ is the stator MMF harmonic order.

A.2 Capacitor-Run Motors

In a capacitor-run motor, the auxiliary winding remains connected. In essence, the capacitor-run motor is actually a two phase machine when running. The main winding, like the capacitor-start, will have the current $I_m(t) = I_o\cos(\omega t)$. The series capacitor will add a phase angle in the auxiliary winding, shown in Fig. A-3. The current will be expressed as $I_a(t) = I_o \cos(\omega t - \phi)$, where ϕ is some phase angle.

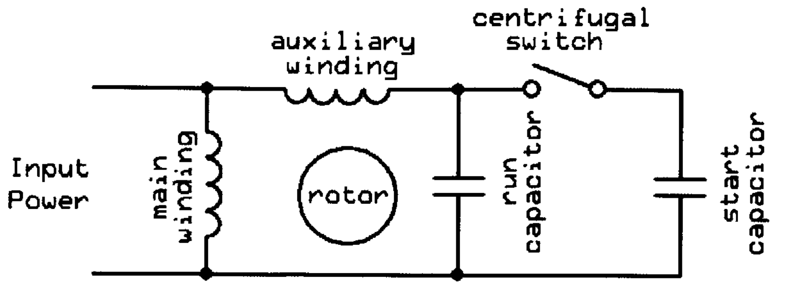


Fig. A-3: Circuit diagram of a capacitor run motor.

The MMF wave expression in (A.1) still holds in a capacitor-run motor. The difference lies in $I(t)$. The stator current is now the sum of the 2 windings.

$$I(t) = I_m(t) + I_a(t) = I_o\left(\cos(\omega t) + \cos(\omega t - \phi)\right). \tag{A.21}$$

Substituting (A.21) into (A.1) yields an expression for the MMF wave of a capacitor-run motor $\mathcal{F}_{gs2}(\theta, t)$.

$$\begin{aligned}\mathcal{F}_{gs2}(\theta, t) = \sum_{\substack{n=1\\ n\ odd}}^{\infty} & f_{nm+}\cos(np\theta - \omega t) + f_{na+}\cos(np\theta - \omega t + \phi) \\ & + f_{nm-}\cos(np\theta + \omega t) + f_{na-}\cos(np\theta + \omega t - \phi),\end{aligned} \tag{A.22}$$

where f_{nm+} is the amplitude of the forward traveling MMF wave from the main winding, f_{na+} is the amplitude of the forward traveling MMF wave from the auxiliary winding, f_{nm-} is the amplitude of the backward traveling MMF wave from the main winding, and f_{na-} is the amplitude of the backward traveling MMF wave from the auxiliary winding.

The MMF wave can be expressed in the rotor's frame of reference by making the substitution in (A.4) and noting that $\omega - \omega_m = s\omega$ and $\omega + \omega_m = (2 - s)\omega$. The nth harmonic of the MMF wave in the rotor frame is

$$\begin{aligned}\mathcal{F}_{grn2}(\theta,t) = f_{nm+}\cos\big(np\theta_r + (n-1)\omega_m t - s\omega t\big)\\ + f_{na+}\cos\big(np\theta_r + (n-1)\omega_m t - s\omega t + \phi\big)\\ + f_{nm-}\cos\big(np\theta_r + (n-1)\omega_m t + (2-s)\omega t\big)\\ + f_{na-}\cos\big(np\theta_r + (n-1)\omega_m t + (2-s)\omega t - \phi\big)\,. \qquad \text{(A.23)}\end{aligned}$$

The flux wave is found by multiplying the MMF wave by the permeance. Note that the permeance is affected by the rotor slots so that the average permeance $\mathcal{P}_o$ is multiplied by $1 + \alpha\cos(R\theta_r)$, where $\alpha < 1$. The nth harmonic of the flux wave seen from the rotor, Φ_{grn2}, is

$$\begin{aligned}\Phi_{grn2} = A_{nm+}\cos\big(np\theta_r + (n-1)\omega_m t - s\omega t\big)\\ + B_{nm+}\cos\big((R+np)\theta_r + (n-1)\omega_m t - s\omega t\big)\\ + B_{nm+}\cos\big((R-np)\theta_r - (n-1)\omega_m t + s\omega t\big)\\ + A_{na+}\cos\big(np\theta_r + (n-1)\omega_m t - s\omega t + \phi\big)\\ + B_{na+}\cos\big((R+np)\theta_r + (n-1)\omega_m t - s\omega t + \phi\big)\\ + B_{na+}\cos\big((R-np)\theta_r - (n-1)\omega_m t + s\omega t - \phi\big)\\ + A_{nm-}\cos\big(np\theta_r + (n-1)\omega_m t + (2-s)\omega t\big)\\ + B_{nm-}\cos\big((R+np)\theta_r + (n-1)\omega_m t + (2-s)\omega t\big)\\ + B_{nm-}\cos\big((R-np)\theta_r - (n-1)\omega_m t - (2-s)\omega t\big)\\ + A_{na-}\cos\big(np\theta_r + (n-1)\omega_m t + (2-s)\omega t - \phi\big)\\ + B_{na-}\cos\big((R+np)\theta_r + (n-1)\omega_m t + (2-s)\omega t - \phi\big)\\ + B_{na-}\cos\big((R-np)\theta_r - (n-1)\omega_m t - (2-s)\omega t + \phi\big)\,, \qquad \text{(A.24)}\end{aligned}$$

where $A_{nm+} = \mathcal{P}_o f_{nm+}$, $B_{nm+} = \frac{1}{2}\alpha\mathcal{P}_o f_{nm+}$, $A_{na+} = \mathcal{P}_o f_{na+}$, $B_{na+} = \frac{1}{2}\alpha\mathcal{P}_o f_{na+}$, $A_{nm-} = \mathcal{P}_o f_{nm-}$, $B_{nm-} = \frac{1}{2}\alpha\mathcal{P}_o f_{nm-}$, $A_{na-} = \mathcal{P}_o f_{na-}$ and $B_{na-} = \frac{1}{2}\alpha\mathcal{P}_o f_{na-}$.

After making the substitutions in (A.10) and $\omega_m = (1-s)\omega$,

$$\begin{aligned}
\Phi_{grn2} = {} & A_{nm+}\cos\left(np\theta - \omega t\right) \\
& + A_{na+}\cos\left(np\theta - \omega t + \phi\right) \\
& + A_{nm-}\cos\left(np\theta + \omega t\right) \\
& + A_{na-}\cos\left(np\theta + \omega t - \phi\right) \\
& + B_{nm+}\cos\left((R+np)\theta - \left(R\frac{1-s}{p}+1\right)\omega t\right) \\
& + B_{nm+}\cos\left((R-np)\theta - \left(R\frac{1-s}{p}-1\right)\omega t\right) \\
& + B_{na+}\cos\left((R+np)\theta - \left(R\frac{1-s}{p}+1\right)\omega t + \phi\right) \\
& + B_{na+}\cos\left((R-np)\theta - \left(R\frac{1-s}{p}-1\right)\omega t - \phi\right) \\
& + B_{nm-}\cos\left((R+np)\theta - \left(R\frac{1-s}{p}-1\right)\omega t\right) \\
& + B_{nm-}\cos\left((R-np)\theta - \left(R\frac{1-s}{p}+1\right)\omega t\right) \\
& + B_{na-}\cos\left((R+np)\theta - \left(R\frac{1-s}{p}-1\right)\omega t + \phi\right) \\
& + B_{na-}\cos\left((R-np)\theta - \left(R\frac{1-s}{p}+1\right)\omega t - \phi\right).
\end{aligned} \tag{A.25}$$

As with capacitor-start motors, capacitor-run motors have slot harmonics, when only considering the fundamental harmonic in the stator current and noting that $\omega = 2\pi f$, are therefore located at

$$f_{sh2} = f\left(R\frac{1-s}{p} \pm 1\right). \tag{A.26}$$

If the derivation includes the odd harmonics in the stator current in (A.2), higher order harmonics in the permeance as described in (A.15) and the eccentricity harmonics in (A.19), the slot harmonics will be described by (A.20).

A.3 Three-Phase Motors

Three-phase induction motors, unlike single-phase motors, require certain relationships between R, the number of rotor slots, and p, the number of pole pairs, in order to observe the slot harmonics. To start, only the fundamental component of the MMF wave will be considered. Thus, the total MMF is

$$\mathcal{F}_{gs3}(\theta,t) = F_1\cos(\theta)i_a + F_1\cos\left(\theta - \frac{2\pi}{3}\right)i_b + F_1\cos\left(\theta + \frac{2\pi}{3}\right)i_c, \tag{A.27}$$

where F_1 is the amplitude of the fundamental component of the MMF wave and i_k is the current in phase k. It is assumed that the three phases are balanced and that

$$i_a + i_b + i_c = 0. \tag{A.28}$$

The three currents can be expanded using Fourier series as

$$i_a = \sum_{\substack{n=1\\ n\ odd}}^{\infty} I_n \cos(n\omega t), \tag{A.29}$$

$$i_b = \sum_{\substack{n=1\\ n\ odd}}^{\infty} I_n \cos\left(n\omega t - \frac{2\pi}{3}\right), \tag{A.30}$$

$$i_c = \sum_{\substack{n=1\\ n\ odd}}^{\infty} I_n \cos\left(n\omega t + \frac{2\pi}{3}\right), \tag{A.31}$$

where I_n is the amplitude of the nth harmonic of the current.
Including (A.29) - (A.31) into (A.27) and arranging terms yields

$$\begin{aligned}\mathcal{F}_{gs3}(\theta,t) = {} & F_1I_1\cos\theta\cos\omega t + F_1I_1\cos\theta\cos(\omega t - \frac{2\pi}{3}) \\ & + F_1I_1\cos\theta\cos(\omega t + \frac{2\pi}{3}) + 3F_1I_3\cos\theta\cos 3\omega t \\ & + 5F_1I_5\cos\theta\cos 5\omega t + F_1I_5\cos\theta\cos 5(\omega t - \frac{2\pi}{3}) \\ & + F_1I_5\cos\theta\cos 5(\omega t + \frac{2\pi}{3}) + \ldots.\end{aligned} \tag{A.32}$$

It can be shown that triplen (multiple of 3) harmonics take the form

$$\mathcal{F}_{gs3k} = kF_1 I_k \cos\theta \cos k\omega t. \tag{A.33}$$

In order to satisfy (A.28) at all points in time, $I_k = 0$. Therefore, there are no triplen harmonics in the MMF. Generally, the odd non-triplen harmonics can be nonzero to satisfy any positive and negative sequence constraints [97]. The total MMF wave can now be expressed as

$$\mathcal{F}_{gs3}(\theta, t) = \sum_{n=6k\pm1} \left[F_1 I_n \cos\theta \cos(n\omega t) + F_1 I_n \cos\theta \cos n\left(\omega t - \frac{2\pi}{3}\right) \right. \\ \left. + F_1 I_n \cos\theta \cos n\left(\omega t + \frac{2\pi}{3}\right) \right]. \tag{A.34}$$

Multiplying the MMF by the permeance creates the flux expression in the stator such as (A.9). The rest of the derivation follows similarly in the single-phase motor case. Reference [28] shows the relationship in (A.35) that must exist between R and p to observe the PSH in three-phase induction motors since the triplen harmonics are zero in the stator MMF.

$$R = 2p[3k \pm r], \tag{A.35}$$

where $k = 0, 1, 2, 3, \ldots$ and $r = 0$ or 1. The location of these slot harmonics are described by (A.20).

Appendix B

Optimized Speed Estimation Method Code

B.1 getOptimalPSH.m

Listing B.1: getOptimalPSH.m

```
function fpsh = getOptimalPSH(data,fs,R,p,fL,fR,fresTest)
% fpsh = getOptimalPSH(data,fs,R,p,fL,fR,fresTest)
% Inputs:
%   data - electrical current data vector
%   fs - sample frequency
%   R - number of rotor slots
%   p - number of pole pairs
%   fL - left frequency /120 (900 -> 7.5)
%   fR - right frequency /120 (1020 -> 8.5)
%   fresTest - frequency resolution of search routine
%
%   Output:
%   fpsh - frequency of estimated principal slot harmonic

L = length(data);
t = 0:1/fs:(L-1)/fs; t = t';

tlen = L/fs;
fres = 1/tlen;
[xfft,ft] = getfft(data,fs,fL,fR);
xfft = abs(xfft);
[max_val, max_ind] = max(xfft);
freqs = ft(max_ind)-fres : fresTest : ft(max_ind)+fres;

%indices that should cover entire peak of psh
```

```
inds_peak = max_ind-3:max_ind+3;

ft = ft(inds_peak);
actual = xfft(inds_peak);
err = zeros(size(freqs));

for k=1:length(freqs)
    yTest = sin(2*pi*freqs(k)*t);
    [fftTest, dummy] = getfft(yTest,fs,fL,fR);

    observed = abs(fftTest(inds_peak));

    scale = sqrt(sum(actual.^2)/sum(observed.^2));
    err(k) = sum(abs((actual-scale*observed)));
end

[dummy,minErrInd] = min(err);

fpsh = freqs(minErrInd);
yTest = sin(2*pi*fpsh*t);
[fftTest, dummy] = getfft(yTest,fs,fL,fR);
observed = abs(fftTest(inds_peak));
scale = sqrt(sum(actual.^2)/sum(observed.^2));
observed = scale*observed;
```

B.2 getfft.m

Listing B.2: getfft.m

```
function [xfft,ft] = getfft(x,fs,fL,fR)
% [xfft,ft] = getfft(x,fs,fL,fR)
% Inputs:
%   x - electrical current data vector
%   fL - left frequency / 120 (900 -> 7.5)
%   fR - right frequency /120 (1020 -> 8.5)
%
%   Output:
%   xfft - FFT of data located in frequency window
%   ft   - FFT frequency vector

f = 60;
N = length(x);
ft = (0:floor(N/2)-1)/N*fs;
xfft = fft(x.*hamming(N));
df = ft(2)-ft(1);

fL = (max(fL*2*f,0))/(2*f);
fR = (min(fR*2*f,ft(end)))/(2*f);
xL = find(ft>fL*2*f-df/2,1);
xR = find(ft>fR*2*f-df/2,1);
xfft = xfft(xL:xR);
ft = ft(xL:xR);
```

Appendix C

Vibration Monitoring Code

C.1 stiffnesseffect.m

Listing C.1: stiffnesseffect.m

```
function [stiffnessratio,electricmetric] = stiffnesseffect
% Matlab Code to test The effect of mounting
%    stiffness on the electric detection metric.
%    It can be seen that the detection metric is sensitive to the
%    vibration mounting stiffness.
%
% Written by: Chris Schantz

   function dx = stiff_mount(t,x)
        z = x(1);
        zd = x(2);

        Force = Amplitude*sin(2*pi*t);
        % Force = 0;

        dx = [zd;...
             (-z*k_bearing + Force -zd*b_bearing)/M_rotor];
    end

    function dx = flexible_mount(t,x)
        w = x(1);
        wd = x(2);
        z = x(3);
        zd = x(4);

        Force = Amplitude*sin(2*pi*t);

        dx = [wd; (-w*k_soft -wd*b_soft +(z - w)*k_bearing + ...
             (zd-wd)*b_bearing)/M_case; zd;...
             (-(z-w)*k_bearing -(zd-wd)*b_bearing + Force)/M_rotor];
    end

k_bearing = 1000;
```

```
Amplitude = 10;
M_rotor = 10;
M_case = 10;
k_soft = 10;
b_bearing = 1 ;
b_soft = 1;

opt = odeset('AbsTol',1e-10,'RelTol',1e-10);
time = [0:.01:200];

[T_stiff, Y_stiff] = ode45(@stiff_mount,time,[0; 0]);
stiff = Y_stiff(:,1);
stiff_max = max(stiff(end-500:end));

N = 50;
k_soft_spread = linspace(1,1000,N);
float_max = zeros(N,1);

for k = 1:N
    k_soft = k_soft_spread(k);

    [T_flex, Y_flex] = ode45(@flexible_mount,time,[0; 0; 0; 0]);

    float = Y_flex(:,1)-Y_flex(:,3);
    float_max(k) = max(float(end-500:end));

end

stiffnessratio = k_soft_spread./k_bearing;
electricmetric = float_max./stiff_max;
plot(stiffnessratio,electricmetric,'k','LineWidth',2);

xlabel('Vibration mounting stiffness to bearing stiffness ratio');
ylabel('Electric detection metric');
grid on;
xlim([0 0.75]);
ylim([0 1]);

end
```

Appendix D

Hilbert Transform Spectral Envelope Code

D.1 getHilbertBands.m

Listing D.1: getHilbertBands.m

```
function [data_bands]=getHilbertBands(data)
    % Written by John Donnal

    % take off the DC offset
    data=data-mean(data);
    % decimate to 1.6 kHz sample rate
    data=decimate(data,5);
    % decimate again to 200 Hz
    data=decimate(data,8);

    block_freq=10; % 10Hz bands
    NUM_BLOCKS=6; % up to 60Hz
    data_bands=zeros(length(data),NUM_BLOCKS);
    % set up the global filter parameters
    Apass = 1;        % Passband Ripple (dB)
    Astop1= 60;
    Astop2= 80;
    match = 'both';  % Band to match exactly

    % CALCULATE PLATFORM STABILITY (SEA STATE)
    % lp filter the data
    Fpass = 8;        % Passband Frequency
    Fstop = 10;        % Stopband Frequency
    Astop = 80;
    h  = fdesign.lowpass(Fpass, Fstop, Apass, Astop, 200);
    Hd = design(h, 'ellip', 'MatchExactly', match);

    % remove the 10Hz variation by using 0.25 second average
    band=filter(Hd,data);
    a=50; b=ones(1,50);
    data_bands(:,1)=filter(b,a,band);

    % CALCULATE MOUNTING PROBLEMS
    % by running successive band pass filters
    for i=2:NUM_BLOCKS
        Fstop1 = (i-1)*block_freq-2;     % First Stopband Frequency
        Fpass1 = (i-1)*block_freq;       % First Passband Frequency
        Fpass2 = i*block_freq;           % Second Passband Frequency
        Fstop2 = i*block_freq+2;         % Second Stopband Frequency

        h  = fdesign.bandpass(Fstop1, Fpass1, Fpass2, Fstop2, ...
            Astop1, Apass, Astop2, 200);
        Hd = design(h, 'ellip', 'MatchExactly', match);

        % generate the bode plot for debugging freqz(Hd);
        band=abs(hilbert(filter(Hd,data)));

        % remove the 10Hz variation by using 0.25 second average
        a=50; b=ones(1,50);
        data_bands(:,i)=filter(b,a,band);
    end
end
```

Appendix E

Behavioral Modeling Framework Code

E.1 mainGUI.m

Listing E.1: mainGUI.m

```
function varargout = mainGUI

hf = findall(0,'Tag',mfilename);
if ~isempty(hf)
    close(hf);
end

hf = localCreateUI;

ad = guidata(hf);

% populate the output if required
if nargout > 0
    varargout{1} = ad;
end


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Function to create the user interface
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function hf = localCreateUI
%try
    % Create the figure, setting appropriate properties
    hf = figure('Tag',mfilename,...
        'Toolbar','none',...
        'MenuBar','none',...
        'IntegerHandle','off',...
        'Position', [624 196 1080 665],...
        'Units','normalized',...
        'Resize','off',...
        'NumberTitle','off',...
        'HandleVisibility','callback',...
        'Name','Ship Design Simulation',...
        'CloseRequestFcn',@localCloseRequestFcn,...
        'Visible','off');

    hsp = uipanel('Parent',hf,...
        'Units','pixels',...
        'Position', [12 34 595 620],...
        'Title','Simulation',...
        'BackgroundColor',get(hf,'Color'),...
        'HandleVisibility','callback',...
        'Tag','simPanel');

    panelTexts = {'Ship-Type Definitions','Environment Defintions'
      ,...
        'Ship Systems', 'Simulation Parameters'};
    btnY = [450 330 135 55];
    panelH = [154 106 190 73];
    textH = [126 78 162 45];
    for idx = 1:4
        uicontrol('Parent',hsp,...
            'Style','pushbutton',...
            'Units','pixels',...
            'Position',[7 btnY(idx) 210 28],...
            'BackgroundColor',get(hf,'Color'),...
            'String',sprintf('Edit %s',panelTexts{idx}),...
            'Callback',@localSimButtonPressed,...
            'HandleVisibility','callback',...
            'Tag',sprintf('SimBtn%d',idx));

        hp1 = uipanel('Parent',hsp,...
            'Units','pixels',...
            'Position', [233 btnY(idx) 351 panelH(idx)],...
            'Title',panelTexts{idx},...
            'BackgroundColor',get(hf,'Color'),...
            'HandleVisibility','callback',...
            'Tag',sprintf('simPanel%d',idx));

        ht1 = uicontrol('Parent',hp1,...
            'Style','text',...
            'Units', 'pixels',...
            'Position',[7 5 334 textH(idx)],...
            'BackgroundColor',get(hf,'Color'),...
            'HorizontalAlignment','left',...
            'ForegroundColor',[0 0 0],...
            'HandleVisibility','callback',...
            'Tag',sprintf('text%d',idx));
    end

    hsb = uicontrol('Parent',hsp,...
        'Style','pushbutton',...
        'Units','pixels',...
        'Position', [212 12 139 33],...
        'BackgroundColor',get(hf,'Color'),...
        'String','Run Simulation',...
        'Callback',@localRunPressed,...
        'HandleVisibility','callback',...
        'Tag','buttonrun');

    hop = uipanel('Parent',hf,...
        'Units','pixels',...
        'Position', [649 165 351 372],...
        'Title','Outputs',...
        'BackgroundColor',get(hf,'Color'),...
        'HandleVisibility','callback',...
        'Tag','outPanel');

    btnTexts = {'Get Power Traces', 'Get Load Factors',...
        'Calculate Fuel Consumption', 'Other Stuff'};
    btnY = [0.8 0.5 0.2 0.2];
    for idx = 1:3
        uicontrol('Parent',hop,...
            'Style','pushbutton',...
            'Units','normalized',...
            'Position',[0.2 btnY(idx) 0.6 0.10],...
            'BackgroundColor',get(hf,'Color'),...
            'ForegroundColor',[0 0 0],...
            'String',btnTexts{idx},...
            'Callback',@localOutputButtonPressed,...
            'HandleVisibility','callback',...
            'Enable','off',...
            'Tag',sprintf('OutBtn%d',idx));
    end


    % Simulation Parameters
    ad.output = hf;
    ad.handles = guihandles(hf);

    % Load Preset Data
    ad = loadPreset(ad);

    str = sprintf('Variable: %s\nClassification: %s\nPropulsion Type
       : %s\nNumber of Shafts: %s\nElectrical Generation Type: %s\
       nElectrical Distribution Type: %s\nCombat Systems: %s\
       nAuxillaries: %s',...
        ad.Variable,ad.Classification,ad.Propulsion,ad.Numshafts,...
        ad.Elecgentype,ad.Elecdisttype,ad.Combat,ad.Auxillaries);
    set(ad.handles.text1,'String',str);

    str = sprintf('Operation Condition: %s\nSpeed Profile: %s\
       nTemperature Profile: %s\nEngine Profile: %s\nSeason: %s',...
        ad.OperatingCondition,ad.Speed,ad.Temperature,ad.Engine,ad.
            Season);
    set(ad.handles.text2,'String',str);

    str = sprintf('Time Step: %d\nTime Length: %d\nTime Start: %d'
        ,...
        ad.TimeStep,ad.TimeLength,ad.TimeStart);
    set(ad.handles.text4,'String',str);

    str = '';

    for idw = 1:numel(ad.AllSystems)
        str = [str sprintf('%s\n  ',ad.AllSystems{idw}(4:end))];
        tempSystem = eval(ad.AllSystems{idw});
        for idx = 1:numel(tempSystem.Subsystems)
            tempLevel1 = tempSystem.Subsystems{idx};
            for idy = 1:numel(tempLevel1)
                str = [str '['];
                tempLevel2 = tempLevel1{idy};
                for idz = 1:numel(tempLevel2)-1
                    str = [str sprintf('%s, ',tempLevel2{idz}(4:end)
                        )];
                end
                str = [str sprintf('%s] ',tempLevel2{end}(4:end))];
            end
            str = [str '\n  '];
        end
       str = [str '\n'];
    end
    str = sprintf(str);
    set(ad.handles.text3,'String',str);

    guidata(hf, ad);
    % Position the UI in the centre of the screen
    movegui(hf,'center')
    % Make the UI visible
    set(hf,'Visible','on');

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Callback Function for Simulation Button
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function localSimButtonPressed(hObject,eventdata)

ad = guidata(hObject);

switch get(hObject,'Tag')
    case 'SimBtn2'
        loadGlobalInputs(ad.output);
    case 'SimBtn3'
        loadShipSystems(ad.output);
    case 'SimBtn4'
        loadSimulationParams(ad.output);
    otherwise % shouldn't be able to get in here
        errordlg('Selection Error', 'modal');
end
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Callback Function for Run button
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function localRunPressed(hObject,eventdata)

% get the application data
ad = guidata(hObject);

TimeStep = ad.TimeStep;
TimeLength = ad.TimeLength-TimeStep;
TimeStart = ad.TimeStart;
AllSystems = ad.AllSystems;

powertrace = zeros(round(TimeLength/TimeStep)+1,1);
numloads = 0;
loadsim = 0;
for k = 1:numel(AllSystems)
    SS = eval(AllSystems{k});
    for l = 1:numel(SS.Subsystems)
        for m = 1:numel(SS.Subsystems{l})
            numloads = numloads + numel(SS.Subsystems{l}{m});
        end
    end
end

H = waitbar(0,'Starting simulation...');

for k = 1:numel(AllSystems)
    SS = eval(AllSystems{k});
    waitbar(0,H,sprintf('Simulating System: %s',SS.Name));
    SS.runSubsystems(TimeLength,TimeStep,TimeStart);
    for l = 1:numel(SS.Subsystems)
        for m = 1:numel(SS.Subsystems{l})
            OnOffVector = SS.SubsystemOnOffVectors{l}{m};
            for n = 1:numel(SS.Subsystems{l}{m})
                tempload = eval(SS.Subsystems{l}{m}{n});
                loadsim = loadsim + 1;
                waitbar(loadsim/numloads,H,...
                    sprintf('In System: %s\n Simulating Load %s',...
                    SS.Name,tempload.Name));
                tempload.run(TimeLength,TimeStep,TimeStart,
                   OnOffVector);
                if ~isempty(tempload.LoadCenter)
                    if (tempload.LoadCenter==11) || (tempload.
                       LoadCenter==23) || (tempload.LoadCenter==61)
                        tempload.getPowerTrace(TimeLength, TimeStep,
                            TimeStart);
                        powertrace = powertrace + tempload.
                           PowerTrace;
                        tempload.getSimLoadFactor(TimeLength,
                           TimeStep);
                    end
                end
            end
            end
        end
    end

    delete(H);
    set(findobj('Tag','OutBtn1'),'Enable','On');
    set(findobj('Tag','OutBtn2'),'Enable','On');

    ad.powertrace = powertrace;
    guidata(hObject,ad);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Callback Function for Output Button
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function localOutputButtonPressed(hObject,eventdata)

ad = guidata(hObject);

switch get(hObject,'Tag')
    case 'OutBtn1'
        loadPowerTraces(ad.output);
    case 'OutBtn2'
        loadLoadFactors(ad.output);
    case 'OutBtn3'
    case 'OutBtn4'
    otherwise % shouldn't be able to get in here
        errordlg('Selection Error', 'modal');
end


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Callback Function for deleting the UI
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function localCloseRequestFcn(hObject,eventdata)

% destroy the window
delete(gcbo);


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Load Engine Profile
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function ad = loadEngineProfile(old_ad)
ad = old_ad;

fid = fopen(ad.Engine);
while 1
    tline = fgetl(fid);

    if ~ischar(tline)
        break;
    end

    words = mystrip(tline);
```

```
        if (numel(words)>1)
            val = words{2};
        end

        VarName = words{1};

        if strcmp(VarName, '};')
            break;
        elseif strcmp(val, '{')
        else
            eval(sprintf('ad.%s = load(''%s'');',VarName,val));
        end
    end
    fclose(fid);


    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    % Load Preset
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    function ad = loadPreset(old_ad)
    fid = fopen('guipreset.abc');

    inShipSystem = false;
    inSubsystems = false;
    inShipComponent = false;
    inGroup = false;
    inLevelRulesText = false;
    inLevel1 = false;
    inLevel2 = false;
    inLevelAnd = false;
    inLevelOr = false;
    inLevelRule = false;
    inLevelRulesText = false;
    tempSystem = ShipSystem;
    tempSubsystems = {};
    tempComponent = ShipComponent;

    myQueue = {};

    tempLevel1 = {};
    tempLevel2 = {};
    tempLevelAnd = {};
    tempLevelOr = {};
    tempLevelRule = {};
    tempLevelRulesText = {};

    ad = old_ad;
    ad.AllSystems = {};

    while 1
        tline = fgetl(fid);

        if ~ischar(tline)
```

```
            break;
        end

        words = mystrip(tline);

        if (numel(words)>1)
            if strcmp(words{2},'{')
                val = words{2};
            else
                val = eval(words{2});
            end
        end

        PropName = words{1};
        switch PropName
            case 'Simulation'
            case 'Name'
                if (inShipComponent)
                    tempComponent.Name = val;
                elseif (inShipSystem)
                    tempSystem.Name = val;
                end
            case 'Variable'
                ad.Variable = val;
            case 'Classification'
                ad.Classification = val;
            case 'Propulsion'
                ad.Propulsion = val;
            case 'Numshafts'
                ad.Numshafts = val;
            case 'Elecgentype'
                ad.Elecgentype = val;
            case 'Elecdisttype'
                ad.Elecdisttype = val;
            case 'Combat'
                ad.Combat = val;
            case 'Auxillaries'
                ad.Auxillaries = val;
            case 'OperatingCondition'
                ad.OperatingCondition = val;
            case 'Speed'
                ad.Speed = val;
                eval(sprintf('ad.SpeedData = load(''%s'');',val));
            case 'Temperature'
                ad.Temperature = val;
                eval(sprintf('ad.Temperatures = load(''%s'');',val));
            case 'Engine'
                ad.Engine = val;
                ad = loadEngineProfile(ad);
            case 'Season'
                ad.Season = val;
            case 'TimeStep'
                ad.TimeStep = val;
            case 'TimeLength'
            ad.TimeLength = val;
        case 'TimeStart'
            ad.TimeStart = val;
        case 'ShipSystem'
            inShipSystem = true;
            tempSystem = ShipSystem;
            myQueue = { 'ShipSystem', myQueue{1:end} };
        case 'Subsystems'
            inSubsystems = true;
            myQueue = { 'Subsystems', myQueue{1:end} };
        case 'ShipComponent'
            inShipComponent = true;
            tempComponent = ShipComponent;
            myQueue = { 'ShipComponent', myQueue{1:end} };
        case 'Level1'
            inLevel1 = true;
            myQueue = { 'Level1', myQueue{1:end} };
        case 'Level2'
            inLevel2 = true;
            myQueue = { 'Level2', myQueue{1:end} };
        case 'Cell'
            tline2 = fgetl(fid);
            cellwords = mystrip(tline2);
            dim = eval(cellwords{2});
            tempCell = cell(1,dim);
            for k = 1:dim
                tline2 = fgetl(fid);
                cellwords = mystrip(tline2);
                element = eval(cellwords{2});
                tempCell{k} = element;
            end
            myQueue = { 'Cell', myQueue{1:end} };
        case 'TemperatureOffset'
            tempComponent.TemperatureOffset = val;
        case 'TemperaturesVarName'
            tempComponent.TemperaturesVarName = val;
            eval(sprintf('tempComponent.Temperatures = %s;',val));
        case 'LevelAnd'
            inLevelAnd = true;
            myQueue = { 'LevelAnd', myQueue{1:end} };
        case 'LevelOr'
            inLevelOr = true;
            myQueue = { 'LevelOr', myQueue{1:end} };
        case 'LevelRule'
            inLevelRule = true;
            myQueue = { 'LevelRule', myQueue{1:end} };
        case '}'
            celltype = myQueue{1};
            switch celltype
                case 'ShipSystem'
                    eval(sprintf('ad.%s = carboncopy(tempSystem);',
                       tempSystem.Name));
                    eval(sprintf('ad.AllSystems = { ad.AllSystems{1:
                       end}, ''ad.%s'' };',tempSystem.Name));
                inShipSystem = false;
            case 'Subsystems'
                tempSystem.Subsystems = tempSubsystems;
                inSubsystems = false;
                tempSubsystems = {};
            case 'ShipComponent'
                eval(sprintf('ad.%s = carboncopy(tempComponent);
                    ',tempComponent.Name));
                eval(sprintf('tempLevel2 = { tempLevel2{1:end},
                    ''ad.%s'' };',tempComponent.Name));
                inShipComponent = false;
            case 'Level2'
                tempLevel1 = { tempLevel1{1:end}, tempLevel2 };
                inLevel2 = false;
                tempLevel2 = {};
            case 'Level1'
                tempSubsystems = { tempSubsystems{1:end},
                    tempLevel1 };
                inLevel1 = false;
                tempLevel1 = {};
            case 'LevelAnd'
                tempLevelOr = { tempLevelOr{1:end}, tempLevelAnd
                     };
                inLevelAnd = false;
                tempLevelAnd = {};
            case 'LevelOr'
                tempLevelRulesText = { tempLevelRulesText{1:end
                    }, tempLevelOr };
                inLevelOr = false;
                tempLevelOr = {};
            case 'LevelRule'
                tempLevelRulesText = { tempLevelRulesText{1:end
                    }, tempLevelRule };
                inLevelRule = false;
                tempLevelRule = {};
            case 'LevelRulesText'
                tempSystem.LevelRulesText = tempLevelRulesText;
                inLevelRulesText = false;
                tempSystem.LevelRules = parseCell(
                    tempLevelRulesText,ad);
                tempLevelRulesText = {};
            case 'Cell'
                if (inLevelAnd)
                    tempLevelAnd = { tempLevelAnd{1:end},
                        tempCell };
                end
            otherwise
                if (inShipComponent)
                    eval(sprintf('tempComponent.%s = tempCell;',
                        myQueue{1}));
                elseif (inShipSystem)
                    eval(sprintf('tempSystem.%s = tempCell;',
                        myQueue{1}));
                end
```

```
                    inGroup = false;
                end
                if (numel(myQueue)>1)
                    myQueue = { myQueue{2:end} };
                else
                    myQueue = {};
                end
            case '};'
                break;
            otherwise
                if strcmp(val,'{')
                    inGroup = true;
                    myQueue = { PropName, myQueue{1:end} };
                else % Property of ShipSystem or ShipComponent
                    if (inShipComponent)
                        eval(sprintf('tempComponent.%s = val;',PropName)
                            );
                    elseif (inShipSystem)
                        eval(sprintf('tempSystem.%s = val;',PropName));
                    end
                end
        end
    end
fclose(fid);
```

E.2 ShipSystem.m

Listing E.2: ShipSystem.m

```
classdef ShipSystem < handle
    properties
        Name
        Subsystems = {};
        SelectionMethod = 'random-subset';
        RandomSubsetParam
        SequentialList
        LevelRulesText
        LevelRules
        StateModel
        OnModel
        OffModel
        TimeDependencies
        FSMCurrentState = 1;
        FSMTransitions
        FSMModels
        SubsystemOnOffVectors
    end % properties

    methods
        function on_times = initializeOnOffVectors(self)
            OnOffVectors = cell(1,numel(self.Subsystems));
            on_times = cell(1,numel(self.Subsystems));
            for k = 1:numel(self.Subsystems)
                OnOffVectors{k} = cell(1,numel(self.Subsystems{k}));
                on_times{k} = cell(1,numel(self.Subsystems{k}));
            end
            self.SubsystemOnOffVectors = OnOffVectors;
        end % initializeOnOffVectors

        function runSubsystems(self,TimeLength, TimeStep, TimeStart)
            on_times = self.initializeOnOffVectors();

            t = TimeStart;
            TimeEnd = TimeStart + TimeLength;

            while ( t < TimeEnd )
                if strcmpi(self.SelectionMethod,'single-state')
                    for k = 1:numel(self.Subsystems)
                        on_times{k}{1} = [TimeStart TimeEnd];
                    end
                    t = TimeEnd + TimeStep;
                elseif strcmpi(self.SelectionMethod,'cycle')
                    t_rand = getRandomModelValue(self.OffModel);
                    t = t + t_rand;
                    t_rand = getRandomModelValue(self.OnModel);
                    on_times{1}{1} = [on_times{1}{1}; [t, t+t_rand
                        ]];
                    t = t + t_rand;
                elseif strcmpi(self.SelectionMethod,'fsm')
                    tempfsmmodel = self.FSMModels{self.
                        FSMCurrentState};
                    t_rand = getRandomModelValue(tempfsmmodel);
                    tempstate = self.FSMTransitions(self.
                        FSMCurrentState,:);
                    rnum = rand(1);
                    probs = cumsum(tempstate{1})>rnum;
                    nextstate = find(probs==1,1);
                    k = self.FSMCurrentState;
                    n = randsample(numel(self.Subsystems{k}), 1);
                    on_times{k}{n} = [on_times{k}{n}; [t, t+t_rand
                        ]];
                    self.FSMCurrentState = nextstate;
                    t = t + t_rand;
                elseif strcmpi(self.SelectionMethod,'level')
                    for k = 1:numel(self.Subsystems)
                        data = evaluateRule(self.LevelRules{k},
                            TimeLength,TimeStep,1);
                        on_times{k}{1} = getOnTimes(data,TimeStart:
                            TimeStep:TimeEnd);
                    end
                    t = TimeEnd + TimeStep;
                elseif strcmpi(self.SelectionMethod,'time-dependency
                    ')
                    for k = 1:numel(self.Subsystems)
                        on_times{k}{1} = getTimeDependency(self.
                                TimeDependencies{k}, TimeLength, TimeStep
                                );
                        end
                        t = TimeEnd + TimeStep;
                    elseif strcmpi(self.SelectionMethod,'random-subset')
                        t_rand = getRandomModelValue(self.StateModel);
                        arr = randsample(numel(self.Subsystems), self.
                            RandomSubsetParam);
                        arr = sort(arr);
                        for ind = 1:numel(arr)
                            k = arr(ind);
                            n = randsample(numel(self.Subsystems{k}), 1)
                                ;
                            on_times{k}{n} = [on_times{k}{n}; [t, t+
                                t_rand]];
                        end
                        t = t + t_rand;
                    else
                        error('SelectionMethod has an unknown value');
                    end
                end % while

                for k = 1:numel(on_times)
                    for l = 1:numel(on_times{k})
                        onoffvec = zeros(round(TimeLength/TimeStep)+1,1)
                            ;
                        cur_on_times = on_times{k}{l};
                        N = size(cur_on_times, 1);
                        for n = 1:N
                            ind_start = round((cur_on_times(n,1)-
                                TimeStart)/TimeStep)+1;
                            ind_end = cur_on_times(n,2);
                            if (ind_end > TimeEnd)
                                ind_end = TimeEnd;
                            end
                            ind_end = round((ind_end-TimeStart)/TimeStep
                                )+1;
                            onoffvec(ind_start:ind_end) = 1;
                        end
                        self.SubsystemOnOffVectors{k}{l} = onoffvec;
                    end % for
                end % for
            end % runSubsystems
        end % methods
end %classdef
```

E.3 ShipComponent.m

Listing E.3: ShipComponent.m

```
classdef ShipComponent < handle
    properties
```

```
        Name
        Type = 'slave';
        OnModel
        OffModel
        OnOffset = 0;
        OffOffset = 0;
        LoadCenter
        SystemNegate = false;
        on_times
        OnOffVector
        PowerTraceType = 'fingerprint'
        LoadPower
        LoadFactor
        SimLoadFactor
        TransientTurnOn
        SteadyState
        TransientTurnOff
        PowerTimeVars
        TemperatureOffset
        TemperaturesVarName
        Temperatures
        PowerFSM
        PowerTrace
    end % properties

    methods
        function run(self,TimeLength,TimeStep,TimeStart,
            SystemOnOffVector)
            if ( self.SystemNegate )
                SystemOnOffVector = ~SystemOnOffVector;
            end

            shiftedleft = SystemOnOffVector;
            shiftedright = SystemOnOffVector;

            if ( self.OnOffset ~= 0 )
                Noffset = round(self.OnOffset/TimeStep);
                shiftedleft = [SystemOnOffVector(Noffset+1:end);
                    zeros(Noffset,1)];
            end

            if ( self.OffOffset ~= 0 )
                Noffset = round(self.OffOffset/TimeStep);
                shiftedright = [zeros(Noffset,1); SystemOnOffVector
                    (1:end-Noffset)];
            end

            SystemOnOffVector = shiftedleft | shiftedright;

            TimeEnd = TimeStart+TimeLength;

            switch lower(self.Type)
                case 'slave'
                    self.OnOffVector = SystemOnOffVector;
                self.on_times = getOnTimes(SystemOnOffVector,
                   TimeStart:TimeStep:TimeEnd);
            case 'master'
                on_times = getOnTimes(SystemOnOffVector,
                   TimeStart:TimeStep:TimeEnd);
                begin_times = on_times(:,1)';
                end_times = on_times(:,2)';

                Nruns = numel(begin_times);
                on_times = [];

                for n = 1:Nruns
                    t = begin_times(n);
                    temp_end_time = end_times(n);
                    while ( t < temp_end_time )
                        % compute the duration of the current
                           off time
                        cur_Model = getCurrentModel(self.
                           OffModel);
                        t_rand = getRandomModelValue(cur_Model);
                        seg_start = t + t_rand;

                        % compute the duration of the current on
                            time
                        cur_Model = getCurrentModel(self.OnModel
                           );
                        t_rand = getRandomModelValue(cur_Model);
                        seg_end = min(seg_start + t_rand,
                           temp_end_time);

                        on_times = [on_times; [seg_start seg_end
                           ]];
                        t = seg_end;
                    end % while
                end % for

                self.on_times = on_times;

                N = size(on_times, 1);

                onoffvec = zeros(round(TimeLength/TimeStep)+1,1)
                   ;

                for n = 1:N
                    ind_start = round((on_times(n,1)-TimeStart)/
                       TimeStep)+1;
                    ind_end = on_times(n,2);
                    if (ind_end > TimeEnd)
                        ind_end = TimeEnd;
                    end
                    ind_end = round((ind_end-TimeStart)/TimeStep
                       )+1;
                    onoffvec(ind_start:ind_end) = 1;
                end
                self.OnOffVector = onoffvec;
            otherwise
                error('Type has an unknown value');
        end % switch
    end % run

    function getPowerTrace(self, TimeLength, TimeStep, TimeStart
       )
        self.PowerTrace = zeros(round(TimeLength/TimeStep)+1,1);
        N = size(self.on_times, 1);
        for n = 1:N
            ind_start = round((self.on_times(n,1)-TimeStart)/
               TimeStep)+1;
            ind_end = round((self.on_times(n,2)-TimeStart)/
               TimeStep)+1;
            int_len = ind_end-ind_start+1;

            if (strcmpi(self.PowerTraceType,'constant'))
                self.PowerTrace(ind_start:ind_end) = self.
                   LoadPower*self.LoadFactor;
            elseif (strcmpi(self.PowerTraceType,'time-dependency
               '))
                self.PowerTrace(ind_start:ind_end) =
                   getTimeDependentPowerTrace(self.PowerTimeVars
                   ,int_len,self.on_times(n,1));
            elseif (strcmpi(self.PowerTraceType,'fsm'))
                self.PowerTrace(ind_start:ind_end) =
                   getFSMPowerTrace(self.PowerFSM,int_len);
            elseif (strcmpi(self.PowerTraceType,'fingerprint'))
                if ~isempty(self.TransientTurnOn)
                    if (strcmpi(self.TransientTurnOn{1},'fixed-
                       model'))
                        TempTurnOn = zeros(self.TransientTurnOn
                           {2},1);
                        for k = 1:self.TransientTurnOn{2}
                            TempTurnOn(k) = getRandomModelValue(
                               self.TransientTurnOn{3});
                        end
                    else
                        TempTurnOn = self.TransientTurnOn{2};
                    end
                    lenOn = numel(TempTurnOn);
                    lenOff = numel(self.TransientTurnOff{2});

                    if (int_len < lenOff)
                        self.PowerTrace(ind_start:ind_end) =
                           self.TransientTurnOff{2}(end-int_len
                           +1:end);
                    elseif (int_len < lenOn + lenOff)
                        self.PowerTrace(ind_start:ind_end) = [
                           getSteadyStateVector(self.SteadyState
                           , int_len-lenOff); self.
                           TransientTurnOff{2}];
                    else
```

```
                                self.PowerTrace(ind_start:ind_start+
                                    lenOn-1) = TempTurnOn;
                                self.PowerTrace(ind_start+lenOn:ind_end-
                                    lenOff) = getSteadyStateVector(self.
                                    SteadyState, int_len-lenOn-lenOff);
                                self.PowerTrace(ind_end-lenOff+1:ind_end
                                    ) = self.TransientTurnOff{2};
                            end % if
                        end % if
                    end % if
                    if ~isempty(self.TemperatureOffset)
                        self.PowerTrace(ind_start:ind_end) =
                            addTemperatureOffset(self.PowerTrace(
                            ind_start:ind_end),int_len,self.on_times(n,1)
                            ,self.Temperatures,self.TemperatureOffset);
                    end
                end % for
            end % getPowerTrace

            function getSimLoadFactor(self, TimeLength, TimeStep)
                self.SimLoadFactor = sum(self.PowerTrace)/TimeLength/
                    self.LoadPower;
            end % getSimLoadFactor
        end % methods
end %classdef
```

E.4 loadGlobalInputs.m

Listing E.4: loadGlobalInputs.m

```
function varargout = loadGlobalInputs(hMainGui)

% Create the UI
hf = localCreateUI(hMainGui);
uiwait(hMainGui);

% populate the output if required
if nargout > 0
    varargout{1} = hf;
end


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Function to create the user interface
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function hf = localCreateUI(hMainGui)
try
    ad = guidata(hMainGui);

    % Create the figure, setting appropriate properties
    hf = figure('Tag',mfilename,...
        'Toolbar','none',...
        'MenuBar','none',...
        'IntegerHandle','off',...
        'Position', [127 133 430 262],...
        'Units','Pixels',...
        'Resize','off',...
        'NumberTitle','off',...
        'HandleVisibility','callback',...
        'Name','Edit Global Inputs',...
        'CloseRequestFcn',@localCloseRequestFcn,...
        'Visible','off',...
        'WindowStyle','modal');

    hts = uicontrol('Parent',hf,...
        'Style','text',...
        'Units', 'pixels',...
        'Position',[4 219 137 17],...
        'BackgroundColor',get(hf,'Color'),...
        'String','Speed Profile',...
        'HandleVisibility','callback',...
        'Tag','textts');

    htt = uicontrol('Parent',hf,...
        'Style','text',...
        'Units', 'pixels',...
        'Position',[4 178 137 17],...
        'BackgroundColor',get(hf,'Color'),...
        'String','Temperature Profile',...
        'HandleVisibility','callback',...
        'Tag','texttt');

    hte = uicontrol('Parent',hf,...
        'Style','text',...
        'Units', 'pixels',...
        'Position',[4 135 137 17],...
        'BackgroundColor',get(hf,'Color'),...
        'String','Engine Profile',...
        'HandleVisibility','callback',...
        'Tag','textte');

    hts2 = uicontrol('Parent',hf,...
        'Style','text',...
        'Units', 'pixels',...
        'Position',[4 93 137 17],...
        'BackgroundColor',get(hf,'Color'),...
        'String','Season',...
        'HandleVisibility','callback',...
        'Tag','textts2');


    hes = uicontrol('Parent',hf,...
        'Enable','Off',...
        'Style','edit',...
        'Units', 'pixels',...
        'Position',[151 215 170 28],...
        'String',ad.Speed,...
        'HandleVisibility','callback',...
        'Tag','edites');

    het = uicontrol('Parent',hf,...
        'Enable','Off',...
        'Style','edit',...
        'Units', 'pixels',...
        'Position',[151 173 170 28],...
        'String',ad.Temperature,...
        'HandleVisibility','callback',...
        'Tag','editet');

    hee = uicontrol('Parent',hf,...
        'Enable','Off',...
        'Style','edit',...
        'Units', 'pixels',...
        'Position',[151 131 170 28],...
        'String',ad.Engine,...
        'HandleVisibility','callback',...
        'Tag','editee');

    hps = uicontrol('Parent',hf,...
        'Style','popupmenu',...
        'Units','pixels',...
        'Position',[151 84 126 28],...
        'BackgroundColor',get(hf,'Color'),...
        'HandleVisibility','callback',...
        'Callback',@localSelectionChanged,...
        'Tag','popups',...
        'String',...
        {'Summer',...
        '<html><font color="gray">Fall</font></html>',...
        '<html><font color="gray">Winter</font></html>',...
        '<html><font color="gray">Spring</font></html>'});

    hbs = uicontrol('Parent',hf,...
        'Style','pushbutton',...
        'Units','pixels',...
        'Position',[326 215 60 28],...
        'BackgroundColor',get(hf,'Color'),...
        'String','Load',...
        'Callback',@localButtonPressed,...
        'HandleVisibility','callback',...
        'Tag','buttonspeed');

    hbt = uicontrol('Parent',hf,...
        'Style','pushbutton',...
        'Units','pixels',...
        'Position',[326 173 60 28],...
        'BackgroundColor',get(hf,'Color'),...
        'String','Load',...
        'Callback',@localButtonPressed,...
        'HandleVisibility','callback',...
```

```
            'Tag','buttontemp');

        hbe = uicontrol('Parent',hf,...
            'Style','pushbutton',...
            'Units','pixels',...
            'Position',[326 131 60 28],...
            'BackgroundColor',get(hf,'Color'),...
            'String','Load',...
            'Callback',@localButtonPressed,...
            'HandleVisibility','callback',...
            'Tag','buttonengine');

        hsb = uicontrol('Parent',hf,...
            'Style','pushbutton',...
            'Units','pixels',...
            'Position',[150 27 75 35],...
            'BackgroundColor',get(hf,'Color'),...
            'String','Save',...
            'Callback',@localSavePressed,...
            'HandleVisibility','callback',...
            'Tag','buttonsave');

        % Create the handles structure
        ad.handles = guihandles(hf);
        ad.parent = hMainGui;
        ad.hes = hes;
        ad.het = het;
        ad.hee = hee;

        % Save the application data
        guidata(hf,ad);

        % Position the UI in the centre of the screen
        movegui(hf,'center')
        % Make the UI visible
        set(hf,'Visible','on');
    catch ME
        % Get rid of the figure if it was created
        if exist('hf','var') && ~isempty(hf) && ishandle(hf)
            delete(hf);
        end
        % throw up an error dialog
        estr = sprintf('%s\n%s\n\n',...
            'The UI could not be created.',...
            'The specific error was:',...
            ME.message);
        errordlg(estr,'UI creation error','modal');
    end


    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    % Callback Function for Save button
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    function localSavePressed(hObject,eventdata)
```

```

% get the application data
ad = guidata(hObject);
hMainGui = ad.parent;

speed = get(ad.hes,'String');
temperature = get(ad.het,'String');
engine = get(ad.hee,'String');

ad = guidata(hMainGui);
ad.Speed = speed;
ad.Temperature = temperature;
ad.Engine = engine;
guidata(hMainGui, ad);

delete(get(gcbo,'Parent'));

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Callback Function for Popup menus
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function localSelectionChanged(hObject,eventdata)

val = get(hObject, 'Value');

switch get(hObject,'Tag')
    case 'popups'
        if val > 1
            set(hObject,'Value',1);
        end
    otherwise
    % shouldn't be able to get in here
        errordlg('Selection Error', 'modal');
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Callback Function for Buttons
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function localButtonPressed(hObject,eventdata)

ad = guidata(hObject);
fileName = uigetfile('*.mat');

switch get(hObject,'Tag')
    case 'buttonspeed'
        set(ad.hes,'String',fileName);
    case 'buttontemp'
        set(ad.het,'String',fileName);
    case 'buttonengine'
        set(ad.hee,'String',fileName);
    otherwise
    % shouldn't be able to get in here
        errordlg('Selection Error', 'modal');
end
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Callback Function for deleting the UI
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function localCloseRequestFcn(hObject,eventdata)

% get the application data
ad = guidata(hObject);
hMainGui = ad.parent;

% destroy the window
delete(gcbo);
```

E.5 loadShipSystems.m

Listing E.5: loadShipSystems.m

```
function varargout = loadShipSystems(hMainGui)

% Create the UI
hf = localCreateUI(hMainGui);
uiwait(hMainGui);

% populate the output if required
if nargout > 0
    varargout{1} = hf;
end


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Function to create the user interface
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function hf = localCreateUI(hMainGui)
try
    ad = guidata(hMainGui);

    % Create the figure, setting appropriate properties
    hf = figure('Tag',mfilename,...
        'Toolbar','none',...
        'MenuBar','none',...
        'IntegerHandle','off',...
        'Position', [127 133 672 488],...
        'Units','Pixels',...
        'Resize','off',...
        'NumberTitle','off',...
        'HandleVisibility','callback',...
        'Name','Edit Ship Systems',...
        'CloseRequestFcn',@localCloseRequestFcn,...
        'Visible','off',...
        'WindowStyle','modal');

```

```
textTags = {'systemTitle','loadTitle'};
textStrings = {'All Systems', 'Subsystem + Components'};
textPos = {[72 438 178 17], [383 415 200 17]};

lbTags = {'lbSys','lbLoads'};
lbPos = {[16 87 298 351], [350 87 301 328]};
lbhs = [ 0 0 ];

for idx = 1:2
    hst = uicontrol('Parent',hf,...
        'Style','text',...
        'Units','pixels',...
        'Position', textPos{idx},...
        'String',textStrings{idx},...
        'BackgroundColor',get(hf,'Color'),...
        'HandleVisibility','callback',...
        'Tag',textTags{idx});

    hsl = uicontrol('Parent',hf,...
        'Style','listbox',...
        'Units', 'pixels',...
        'Position',lbPos{idx},...
        'BackgroundColor',get(hf,'Color'),...
        'String','',...
        'Callback',@localListboxSelected,...
        'HandleVisibility','callback',...
        'Tag',lbTags{idx});

    lbhs(idx) = hsl;
end

hba = uicontrol('Parent',hf,...
    'Style','pushbutton',...
    'Units','pixels',...
    'Position',[50 46 201 32],...
    'BackgroundColor',get(hf,'Color'),...
    'String','Add System',...
    'Callback',@localButtonPressed,...
    'HandleVisibility','callback',...
    'Tag','buttonaddsys');

hbs = uicontrol('Parent',hf,...
    'Style','pushbutton',...
    'Units','pixels',...
    'Position',[263 6 118 32],...
    'BackgroundColor',get(hf,'Color'),...
    'String','Save',...
    'Callback',@localSavePressed,...
    'HandleVisibility','callback',...
    'Tag','buttonsave');

systemlbtext = cell(numel(ad.AllSystems),1);
for k = 1:numel(ad.AllSystems)
    systemlbtext{k} = ad.AllSystems{k}(4:end);
    end

    set(lbhs(1),'String',systemlbtext,'Value',1);

    % Create the handles structure
    ad.handles = guihandles(hf);
    ad.parent = hMainGui;

    % Save the application data
    guidata(hf,ad);

    % Position the UI in the centre of the screen
    movegui(hf,'center');
    % Make the UI visible
    set(hf,'Visible','on');
catch ME
    % Get rid of the figure if it was created
    if exist('hf','var') && ~isempty(hf) && ishandle(hf)
        delete(hf);
    end
    % throw up an error dialog
    estr = sprintf('%s\n%s\n\n',...
        'The UI could not be created.',...
        'The specific error was:',...
        ME.message);
    errordlg(estr,'UI creation error','modal');
end


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Callback Function for Save button
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function localSavePressed(hObject,eventdata)

% get the application data
ad = guidata(hObject);
hMainGui = ad.parent;

ad = guidata(hMainGui);
guidata(hMainGui, ad);

delete(get(gcbo,'Parent'));

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Callback Function for Listbox
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function localListboxSelected(hObject,eventdata)
ad = guidata(hObject);

switch get(hObject,'Tag')
    case 'lbSys'
        val = get(gcbo,'Value');
        str = get(gcbo,'String');
        str = sprintf('ad.%s',str{val});
```

```
            tempsys = eval(str);
            subsys = tempsys.Subsystems;
            newstr = cell(numel(subsys),1);
            for k = 1:numel(subsys)
                tempstr = '';
                for l = 1:numel(subsys{k})
                    temp2str = '[';
                    for m = 1:numel(subsys{k}{l})
                        temp2str = [temp2str sprintf('%s, ',subsys{k}{l
                            }{m}(4:end))];
                    end
                    temp2str = [temp2str(1:end-2), '] '];
                    tempstr = [tempstr, temp2str];
                end
                newstr{k} = tempstr(1:end-1);
            end
            set(findobj('Tag','lbLoads'),'String',newstr,'Value',1);
        case 'lbLoads'
        otherwise
        % shouldn't be able to get in here
            errordlg('Selection Error', 'modal');
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Callback Function for Buttons
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function localButtonPressed(hObject,eventdata)

ad = guidata(hObject);

switch get(hObject,'Tag')
        case 'buttonaddsys'
        otherwise
        % shouldn't be able to get in here
            errordlg('Selection Error', 'modal');
end


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Callback Function for deleting the UI
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function localCloseRequestFcn(hObject,eventdata)

% destroy the window
delete(gcbo);
```

E.6 loadSimulationParams.m

Listing E.6: loadSimulationParams.m

```
function varargout = loadSimulationParams(hMainGui)
```

```
% Create the UI
hf = localCreateUI(hMainGui);

% populate the output if required
if nargout > 0
    varargout{1} = hf;
end


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Function to create the user interface
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function hf = localCreateUI(hMainGui)
try
    % Create the figure, setting appropriate properties
    hf = figure('Tag',mfilename,...
        'Toolbar','none',...
        'MenuBar','none',...
        'IntegerHandle','off',...
        'Position', [624 728 440 135],...
        'Units','normalized',...
        'Resize','off',...
        'NumberTitle','off',...
        'HandleVisibility','callback',...
        'Name','Edit Simulation Parameters',...
        'CloseRequestFcn',@localCloseRequestFcn,...
        'Visible','off',...
        'WindowStyle','modal');

    htl = uicontrol('Parent',hf,...
        'Style','text',...
        'Units', 'pixels',...
        'Position',[34 103 168 17],...
        'BackgroundColor',get(hf,'Color'),...
        'String','Simulation Time Length',...
        'HandleVisibility','callback',...
        'Tag','texttl'); %#ok

    hts = uicontrol('Parent',hf,...
        'Style','text',...
        'Units','pixels',...
        'Position',[34 61 168 17],...
        'BackgroundColor',get(hf,'Color'),...
        'String','Simulation Time Step',...
        'HandleVisibility','callback',...
        'Tag','textts'); %#ok

    hlp = uicontrol('Parent',hf,...
        'Style','popupmenu',...
        'Units','pixels',...
        'Position',[220 102 180 22],...
        'BackgroundColor',get(hf,'Color'),...
        'HandleVisibility','callback',...
        'Callback',@localSelectionChanged,...
        'Tag','popuplp',...
        'String',...
        {'80 hours', '25 days', ...
        '<html><font color="gray">6 months </font></html>',...
        '<html><font color="gray">1 year </font></html>'});

    hsp = uicontrol('Parent',hf,...
        'Style','popupmenu',...
        'Units','pixels',...
        'Position',[220 60 180 22],...
        'BackgroundColor',get(hf,'Color'),...
        'HandleVisibility','callback',...
        'Callback',@localSelectionChanged,...
        'Tag','popupsp',...
        'String',...
        {'1 second', '<html><font color="gray">1 minute </font></
           html>',...
        '<html><font color="gray">1 hour </font></html>'});

    hsb = uicontrol('Parent',hf,...
        'Style','pushbutton',...
        'Units','pixels',...
        'Position',[169 7 81 26],...
        'BackgroundColor',get(hf,'Color'),...
        'String','Save',...
        'Callback',@localSavePressed,...
        'HandleVisibility','callback',...
        'Tag','buttonsave');

    % Create the handles structure
    ad.handles = guihandles(hf);
    ad.parent = hMainGui;
    ad.hlp = hlp;
    ad.hsp = hsp;

    % Save the application data
    guidata(hf,ad);

    % Position the UI in the centre of the screen
    movegui(hf,'center')
    % Make the UI visible
    set(hf,'Visible','on');
catch ME
    % Get rid of the figure if it was created
    if exist('hf','var') && ~isempty(hf) && ishandle(hf)
        delete(hf);
    end
    % throw up an error dialog
    estr = sprintf('%s\n%s\n\n',...
        'The UI could not be created.',...
        'The specific error was:',...
        ME.message);
    errordlg(estr,'UI creation error','modal');
end
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Callback Function for Save button
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function localSavePressed(hObject,eventdata) %#ok

% get the application data
ad = guidata(hObject);
hMainGui = ad.parent;

timestep = 1;

val = get(ad.hlp,'Value');
if (val == 1)
    timelength = 60*60*80-timestep;
else
    timelength = 60*60*24*25-timestep;
end

ad = guidata(hMainGui);
ad.TimeLength = timelength;
guidata(hMainGui, ad);

delete(get(gcbo,'Parent'));


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Callback Function for Popup menus
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function localSelectionChanged(hObject,eventdata) %#ok

% get the application data
ad = guidata(hObject);

val = get(hObject, 'Value');

switch get(hObject,'Tag')
    case 'popuplp'
        if val > 2
            set(hObject,'Value',1);
        end
    case 'popupsp'
        if val > 1
            set(hObject,'Value',1);
        end
    otherwise
    % shouldn't be able to get in here
        errordlg('Selection Error', 'modal');
end


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Callback Function for deleting the UI
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function localCloseRequestFcn(hObject,eventdata) %#ok

% get the application data
ad = guidata(hObject);
hMainGui = ad.parent;

% destroy the window
delete(gcbo);
```

E.7 loadPowerTraces.m

Listing E.7: loadPowerTraces.m

```
function varargout = loadPowerTraces(hMainGui)

% Create the UI
hf = localCreateUI(hMainGui);
uiwait(hMainGui);

% populate the output if required
if nargout > 0
    varargout{1} = hf;
end


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Function to create the user interface
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function hf = localCreateUI(hMainGui)
try
    ad = guidata(hMainGui);

    % Create the figure, setting appropriate properties
    hf = figure('Tag',mfilename,...
        'Toolbar','none',...
        'MenuBar','none',...
        'IntegerHandle','off',...
        'Position', [0 133 1200 488],...
        'Units','Pixels',...
        'Resize','off',...
        'NumberTitle','off',...
        'HandleVisibility','callback',...
        'Name','Plot Power Traces',...
        'CloseRequestFcn',@localCloseRequestFcn,...
        'Visible','off',...
        'WindowStyle','modal');

    textTags = {'systemTitle','loadTitle'};
    textStrings = {'All Systems', 'System Loads'};
    textPos = {[72 438 178 17], [436 415 123 17]};
```

```
lbTags = {'lbSys','lbLoads'};
lbPos = {[16 87 298 351], [350 87 301 328]};
lbhs = [ 0 0 ];

for idx = 1:2
    hst = uicontrol('Parent',hf,...
        'Style','text',...
        'Units','pixels',...
        'Position', textPos{idx},...
        'String',textStrings{idx},...
        'BackgroundColor',get(hf,'Color'),...
        'HandleVisibility','callback',...
        'Tag',textTags{idx});

    hsl = uicontrol('Parent',hf,...
        'Style','listbox',...
        'Units', 'pixels',...
        'Position',lbPos{idx},...
        'BackgroundColor',get(hf,'Color'),...
        'String','',...
        'Callback',@localListboxSelected,...
        'HandleVisibility','callback',...
        'Tag',lbTags{idx});

    lbhs(idx) = hsl;
end

hsa = axes('Parent',hf,...
    'Units','pixels',...
    'Position',[707 87 477 340],...
    'Visible','off',...
    'Tag','powertraceaxes');

hse = uicontrol('Parent',hf,...
    'Style','text',...
    'Units','pixels',...
    'Position', [707 245 477 16],...
    'String', 'Power trace cannot be plotted!',...
    'BackgroundColor',get(hf,'Color'),...
    'Visible','off',...
    'Tag','powertracewarning');


systemlbtext = cell(numel(ad.AllSystems)+1,1);
systemlbtext{1} = '1SA';
for k = 1:numel(ad.AllSystems)
    systemlbtext{k+1} = ad.AllSystems{k}(4:end);
end

set(lbhs(1),'String',systemlbtext,'Value',1);

% Create the handles structure
ad.handles = guihandles(hf);
ad.parent = hMainGui;
```

```
    % Save the application data
    guidata(hf,ad);

    % Position the UI in the centre of the screen
    movegui(hf,'center');
    % Make the UI visible
    set(hf,'Visible','on');
catch ME
    % Get rid of the figure if it was created
    if exist('hf','var') && ~isempty(hf) && ishandle(hf)
        delete(hf);
    end
    % throw up an error dialog
    estr = sprintf('%s\n%s\n\n',...
        'The UI could not be created.',...
        'The specific error was:',...
        ME.message);
    errordlg(estr,'UI creation error','modal');
end


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Callback Function for Listbox
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function localListboxSelected(hObject,eventdata)
ad = guidata(hObject);

cla;
xlabel('');
ylabel('');
title('');
set(findobj('Tag','powertraceaxes'),'Visible','off');
set(findobj('Tag','powertracewarning'),'Visible','off');

switch get(hObject,'Tag')
    case 'lbSys'
        val = get(gcbo,'Value');
        if (val==1)
            newstr = {'All'};
        else
            str = get(gcbo,'String');
            str = sprintf('ad.%s',str{val});
            tempsys = eval(str);
            subsys = tempsys.Subsystems;
            newstr = {};
            for k = 1:numel(subsys)
                for l = 1:numel(subsys{k})
                    for m = 1:numel(subsys{k}{l})
                        newstr = { newstr{1:end} subsys{k}{l}{m}(4:
                            end) };
                    end
                end
            end
```

```
                newstr = newstr';
            end
            set(findobj('Tag','lbLoads'),'String',newstr,'Value',1);
        case 'lbLoads'
            t = ad.TimeStart:ad.TimeStep:ad.TimeStart+ad.TimeLength-ad.
               TimeStep;
            t = t/(60*60*24);

            str = get(findobj('Tag','lbLoads'),'String');
            if strcmp(str,'All')
                curPowerTrace = ad.powertrace;
                titlestr = '1SA Switchboard';
            else
                val = get(findobj('Tag','lbLoads'),'Value');
                tempLoad = eval(sprintf('ad.%s',str{val}));
                curPowerTrace = tempLoad.PowerTrace;
                titlestr = tempLoad.Name;
            end

            if isempty(curPowerTrace)
                set(findobj('Tag','powertracewarning'),'Visible','on');
            else
                curPowerTrace = curPowerTrace(1:numel(t));
                plot(t,curPowerTrace,'k','LineWidth',1.5);
                xlabel('Time (days)');
                ylabel('Power (kW)');
                title(sprintf('Power Trace for %s',titlestr));
                set(findobj('Tag','powertraceaxes'),'Visible','on');
            end
        otherwise
        % shouldn't be able to get in here
            errordlg('Selection Error', 'modal');
end


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Callback Function for deleting the UI
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function localCloseRequestFcn(hObject,eventdata)

% destroy the window
delete(gcbo);
```

E.8 loadLoadFactors.m

Listing E.8: loadLoadFactors.m

```
function varargout = loadLoadFactors(hMainGui)

% Create the UI
hf = localCreateUI(hMainGui);
uiwait(hMainGui);
```

```
% populate the output if required
if nargout > 0
    varargout{1} = hf;
end


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Function to create the user interface
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function hf = localCreateUI(hMainGui)
try
    ad = guidata(hMainGui);

    % Create the figure, setting appropriate properties
    hf = figure('Tag',mfilename,...
        'Toolbar','none',...
        'MenuBar','none',...
        'IntegerHandle','off',...
        'Position', [127 133 903 488],...
        'Units','Pixels',...
        'Resize','off',...
        'NumberTitle','off',...
        'HandleVisibility','callback',...
        'Name','Edit Ship Systems',...
        'CloseRequestFcn',@localCloseRequestFcn,...
        'Visible','off',...
        'WindowStyle','modal');

    textTags = {'systemTitle','loadTitle'};
    textStrings = {'All Systems', 'System Loads'};
    textPos = {[72 438 178 17], [436 415 123 17]};

    lbTags = {'lbSys','lbLoads'};
    lbPos = {[16 87 298 351], [350 87 301 328]};
    lbhs = [ 0 0 ];

    for idx = 1:2
        hst = uicontrol('Parent',hf,...
            'Style','text',...
            'Units','pixels',...
            'Position', textPos{idx},...
            'String',textStrings{idx},...
            'BackgroundColor',get(hf,'Color'),...
            'HandleVisibility','callback',...
            'Tag',textTags{idx});

        hsl = uicontrol('Parent',hf,...
            'Style','listbox',...
            'Units', 'pixels',...
            'Position',lbPos{idx},...
            'BackgroundColor',get(hf,'Color'),...
            'String','',...
            'Callback',@localListboxSelected,...
```

```
                'HandleVisibility','callback',...
                'Tag',lbTags{idx});

            lbhs(idx) = hsl;
        end


        texthdrTexts = {'Old Load Factor','Estimated Load Factor'};
        texthdrPos = {[688 320 148 22], [688 200 148 22]};
        textPos = {[688 299 148 22], [688 174 148 22]};

        for idx = 1:2
            hp1 = uicontrol('Parent',hf,...
                'Style','text',...
                'Units', 'pixels',...
                'Position',texthdrPos{idx},...
                'BackgroundColor',get(hf,'Color'),...
                'HorizontalAlignment','left',...
                'ForegroundColor',[0 0 0],...
                'HandleVisibility','callback',...
                'String',texthdrTexts{idx});

            ht1 = uicontrol('Parent',hf,...
                'Style','text',...
                'Units', 'pixels',...
                'Position',textPos{idx},...
                'BackgroundColor',get(hf,'Color'),...
                'HorizontalAlignment','left',...
                'ForegroundColor',[0 0 0],...
                'HandleVisibility','callback',...
                'Tag',sprintf('LFtext%d',idx));
        end

        systemlbtext = cell(numel(ad.AllSystems),1);
        for k = 1:numel(ad.AllSystems)
            systemlbtext{k} = ad.AllSystems{k}(4:end);
        end

        set(lbhs(1),'String',systemlbtext,'Value',1);

        % Create the handles structure
        ad.handles = guihandles(hf);
        ad.parent = hMainGui;

        % Save the application data
        guidata(hf,ad);

        % Position the UI in the centre of the screen
        movegui(hf,'center');
        % Make the UI visible
        set(hf,'Visible','on');
    catch ME
        % Get rid of the figure if it was created
        if exist('hf','var') && ~isempty(hf) && ishandle(hf)
```

```
            delete(hf);
        end
        % throw up an error dialog
        estr = sprintf('%s\n%s\n\n',...
            'The UI could not be created.',...
            'The specific error was:',...
            ME.message);
        errordlg(estr,'UI creation error','modal');
    end


    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    % Callback Function for Listbox
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    function localListboxSelected(hObject,eventdata)
    ad = guidata(hObject);

    switch get(hObject,'Tag')
        case 'lbSys'
            val = get(gcbo,'Value');
            tempstr = get(gcbo,'String');
            tempstr = sprintf('ad.%s',tempstr{val});
            tempsys = eval(tempstr);
            subsys = tempsys.Subsystems;
            newstr = {};
            for k = 1:numel(subsys)
                for l = 1:numel(subsys{k})
                    for m = 1:numel(subsys{k}{l})
                        newstr = { newstr{1:end} subsys{k}{l}{m}(4:end)
                            };
                    end
                end
            end
            newstr = newstr';

            set(findobj('Tag','lbLoads'),'String',newstr,'Value',1);
        case 'lbLoads'
            val = get(gcbo,'Value');
            tempstr = get(gcbo,'String');
            tempstr = sprintf('ad.%s',tempstr{val});
            tempload = eval(tempstr);
            set(findobj('Tag','LFtext1'),'String',sprintf('%0.2f',
               tempload.LoadFactor));
            set(findobj('Tag','LFtext2'),'String',sprintf('%0.2f',
               tempload.SimLoadFactor));
        otherwise
        % shouldn't be able to get in here
            errordlg('Selection Error', 'modal');
    end


    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    % Callback Function for deleting the UI
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
function localCloseRequestFcn(hObject,eventdata)

% destroy the window
delete(gcbo);
```

E.9 mystrip.m

Listing E.9: mystrip.m

```
function words = mystrip( tline )
% split line of text into a cell of words
words = regexp(strtrim(regexprep(tline,'\t', '  ')), ' +', 'split');
end
```

E.10 carboncopy.m

Listing E.10: carboncopy.m

```
function clone = carboncopy( original )
% Make a copy of a handle object.

% Instantiate new object of the same class.
clone = feval(class(original));

% Copy all non-hidden properties.
p = properties(original);
for idx = 1:length(p)
    if ~isempty(original.(p{idx}))
        clone.(p{idx}) = original.(p{idx});
    end
end
end
```

E.11 parseCell.m

Listing E.11: parseCell.m

```
function parsedcell = parseCell(cur_cell,ad)
if ischar(cur_cell{1})
    parsedcell = {eval(cur_cell{1}) cur_cell{2:end}};
else
    parsedcell = cell(size(cur_cell));
    for k = 1:numel(cur_cell)
        parsedcell{k} = parseCell(cur_cell{k},ad);
    end
end
```

E.12 getRandomModelValue.m

Listing E.12: getRandomModelValue.m

```
function val = getRandomModelValue(Model)
% val = getRandomModelValue(Model)
% Possible Models:
%          {'deterministic',a}
%            a -> value of constant
%          {'uniform',a,b}
%           a -> left endpoint of interval
%           b -> right endpoint
%          {'exponential',lambda}
%            lambda -> arrival rate
%          {'poisson',lambda}
%            lambda -> arrival rate

val = -1;

while val < 0
    if ( strcmpi(Model{1}, 'deterministic') )
        val = Model{2};
    elseif ( strcmpi(Model{1}, 'uniform') )
        val = rand(1);
        val = val*(Model{3}-Model{2}) + Model{2};
    elseif ( strcmpi(Model{1}, 'normal') )
        val = randn(1,1);
        val = val*Model{3}+Model{2};
    elseif ( strcmpi(Model{1}, 'exponential') )
        val = rand(1);
        val = expinv(val,1/Model{2});
    elseif ( strcmpi(Model{1}, 'poisson') )
        val = rand(1);
        val = poissinv(val,Model{2});
    else
        error('Model has an unknown type');
    end % if
end
```

E.13 getOnTimes.m

Listing E.13: getOnTimes.m

```
function on_times = getOnTimes(data, t)
%getOnTimes Get the start and end times of on runs
%   on_times = getOnTimes(data, t)
%    data : the row vector containing the on/off usage of load
%       t : corresponding time vector
% on_times: Nx2 array of start and end times of on runs

    ind_ons = find(data == 1);
    on_times = [];
    if (~isempty(ind_ons))
        indend = numel(ind_ons);
        curind = 2;
        temp_on_times = [t(ind_ons(1)) 0];

        while (curind <= indend)
            if (ind_ons(curind) ~= ind_ons(curind-1) + 1)
                temp_on_times(2) = t(ind_ons(curind-1));
                on_times = [on_times; temp_on_times];
                temp_on_times(1) = t(ind_ons(curind));
            end
            curind = curind+1;
        end
        on_times = [on_times; [temp_on_times(1) t(ind_ons(end))]];
    end
end
```

E.14 getTimeDependency.m

Listing E.14: getTimeDependency.m

```
function on_times = getTimeDependency(t_arr, TimeLength, TimeStep,
    TimeStart)
%getTimeDependency Get the start and end times of on runs dependent
    on t_arr
%   on_times = getOnTimes(t_arr, TimeLength, TimeStep, TimeStart)
%          t_arr : array of time intervals
%     TimeLength : simulation parameter
%       TimeStep : simulation parameter
%      TimeStart : simulation parameter

t = TimeStart:TimeStep:TimeStart+TimeLength;

t_display = datenum(datestr(t/86400,13));

onoffvector = zeros(size(t_display));

for k = 1:numel(t_arr)
    start_time = datenum(t_arr{k}{1});
    end_time = datenum(t_arr{k}{2});

    onoffvector = onoffvector | ((t_display >= start_time) & (
        t_display <= end_time));
end

on_times = getOnTimes(onoffvector,t);
end
```

E.15 getCurrentModel.m

Listing E.15: getCurrentModel.m

```
function cur_model = getCurrentModel(models)
%getCurrentModel Get the current model based on external conditions
%   cur_model = getCurrentModel(models)
%          models : cell of various models
%       cur_model : output model

% For now, just return the model

cur_model = models;
```

E.16 getTimeDependentPowerTrace.m

Listing E.16: getTimeDependentPowerTrace.m

```
function PowerVector = getTimeDependentPowerTrace(PowerTimeVars,
   VectorLength, tstart)
% PowerVector = getTimeDependentPowerTrace(PowerTimeVars,
   VectorLength)
%    This function breaks the day into 12 2-hr time blocks
%    PowerTimeVars has a 12 element array to describe the mean and
%    standard deviation for each time block.

t = tstart:tstart+VectorLength-1;
day = t/(3600*24);
day_rem = day - floor(day);
hr = floor(day_rem*24);
block_ind = floor(hr/2)+1;

mu = PowerTimeVars{1}(block_ind);
std = PowerTimeVars{2}(block_ind);

PowerVector = randn(1,VectorLength).*std'+mu';
```

E.17 getFSMPowerTrace.m

Listing E.17: getFSMPowerTrace.m

```
function PowerVector = getFSMPowerTrace(PowerFSM,VectorLength)
% PowerVector = getFSMPowerTrace(PowerFSM,VectorLength)

%POWERFSM = { {1, {'normal',2,3}, {'normal',9.25,0.5}, .05} };

PowerVector = zeros(1,VectorLength);
remLen = VectorLength;
curstate = 1;
```

```
while remLen > 0
    rnum = rand(1);
    tempstate = PowerFSM{curstate};
    tempLength = floor(getRandomModelValue(tempstate{2}));
    tempLength = min(tempLength,remLen);

    mu = getRandomModelValue(tempstate{3});
    std = tempstate{4};

    PowerVector(VectorLength-remLen+1:VectorLength-remLen+tempLength
        ) = getNormalSteadyState(mu,std,tempLength);

    probs = cumsum(tempstate{1})>rnum;
    curstate = find(probs==1,1);
    remLen = remLen - tempLength;
end
```

E.18 getSteadyStateVector.m

Listing E.18: getSteadyStateVector.m

```
function SSVector = getSteadyStateVector(SSModel,VectorLength)
% SSVector = getSteadyStateVector(SSModel,VectorLength)
SSVector = zeros(VectorLength,1);

SStype = SSModel{1};

if (strcmpi(SStype,'fingerprint'))
    lenSS = numel(SSModel{2});

    if (lenSS > VectorLength)
        SSVector = SSModel{2}(1:VectorLength);
    else
        N_SS = floor(VectorLength/lenSS);
        mod_SS = mod(VectorLength,lenSS);
        tempstart = 1;
        for k = 1:N_SS
            SSVector(tempstart:tempstart+lenSS-1) = SSModel{2};
            tempstart = tempstart + lenSS;
        end
        SSVector(tempstart:tempstart+mod_SS-1) = SSModel{2}(1:mod_SS
            );
    end
elseif (strcmpi(SStype,'multiple'))
    ind = 1;
    for k = 1:VectorLength
        rnum = rand(1);
        probs = cumsum(SSModel{2}{ind})>rnum;
        ind = find(probs==1,1);
        SSVector(k) = getRandomModelValue(SSModel{3}{ind});
    end
elseif (strcmpi(SStype,'normal'))
```

```
    SSVector = getNormalSteadyState(SSModel{2},SSModel{3},
        VectorLength);
else
    for k = 1:VectorLength
        SSVector(k) = getRandomModelValue(SSModel);
    end
end
```

E.19 getNormalSteadyState.m

Listing E.19: getNormalSteadyState.m

```
function SSVector = getNormalSteadyState(mu,std,VectorLength)
% SSVector = getNormalSteadyState(mu,std,VectorLength)

SSVector = (randn(VectorLength,1)*std)+mu;
inds = find(SSVector<0);
for ind = inds;
    SSVector(ind) = getRandomModelValue({'normal',mu,std});
end
```

E.20 addTemperatureOffset.m

Listing E.20: addTemperatureOffset.m

```
function newpowertrace = addTemperatureOffset(powertrace,
   VectorLength,tstart,Temperatures,Offset)
% newpowertrace = addTemperatureOffset(powertrace,VectorLength,
   tstart,Temperatures,Offset);
%       Offset -> array of parameters
%                 [A,B,C,D,E]
%        For a given value of x, the offset shift value is given by
%        offset =( A(x-B)^C + D)/E

t = tstart:tstart+VectorLength-1;
day = t/(3600*24);
day = floor(day);
Temps = Temperatures(day+1);

A = Offset(1);
B = Offset(2);
C = Offset(3);
D = Offset(4);
E = Offset(4);

offset = (A*(Temps-B).^C+D)./E;

newpowertrace = powertrace + offset;
```

E.21 checkModel.m

Listing E.21: checkModel.m

```
function isModel = checkModel(Model)
  ModelElements = numel(Model);
  if (ModelElements <= 1)
      error('Model must have more than 1 element');
  end % if

  ModelName = Model{1};

  if ( strcmpi(ModelName, 'uniform') )
      if (ModelElements ~= 3)
          error('Model has an incorrect number of parameters');
      end
      if (Model{3} < Model{2})
          error('Model parameters are not possible');
      end
  elseif ( strcmpi(ModelName, 'normal') )
      if (ModelElements ~= 3)
          error('Model has an incorrect number of parameters');
      end
  elseif ( strcmpi(ModelName, 'deterministic') || strcmpi(ModelName,
      'exponential') || strcmpi(ModelName, 'poisson') )
      if (ModelElements ~= 2)
          error('Model has an incorrect number of parameters');
      end
  else
      error('Model has an unknown value');
  end % if
  isModel = true;
end % checkModel
```

E.22 evaluateRule.m

Listing E.22: evaluateRule.m

```
function data = evaluateRule(Rules, TimeLength, TimeStep, level)
% data = evaluateRule(Rules, TimeLength, TimeStep, level)
% Examples of Rule Structure:
%   (x > 0) -> {{{x,'gt',0}}}
%   (x < 0) & (y == 1) -> {{{x,'gt',0}}{{y,'eq',1}}}
%   (x > 0) || (x < 0) & (y == 1) -> {{{x,'gt',0}}{{x,'gt',0}}{{y,'eq
    ',1}}}

if (level == 1)
    data = zeros((round(TimeLength/TimeStep)+1),1);
else
    data = ones((round(TimeLength/TimeStep)+1),1);
end
```

```
if (level == 1)
    for k = 1:numel(Rules)
        data = data | evaluateRule(Rules{k}, TimeLength, TimeStep,
           2);
    end
elseif (level == 2)
    for k = 1:numel(Rules)
        data = data & evaluateRule(Rules{k}, TimeLength, TimeStep,
           3);
    end
else
    op = Rules{2};
    var = Rules{1};
    num = Rules{3};

    var = var(1:round(TimeLength/TimeStep)+1);

    if ( strcmpi(op, 'eq') )
        data = data & ( var == num );
    elseif ( strcmpi(op, 'lt') )
        data = data & ( var < num );
    elseif ( strcmpi(op, 'gt') )
        data = data & ( var > num );
    elseif ( strcmpi(op, 'leq') )
        data = data & ( var <= num );
    elseif ( strcmpi(op, 'geq') )
        data = data & ( var >= num );
    else
        error('Rule has an unknown operation');
    end % if
end
```

Appendix F

Dual-Generators Setup

F.1 Introduction

This chapter provides an overview of the design and implementation of a controller for a hardware model of a shipboard electrical distribution system. The aim is to build a scaled model of the zonal electrical distribution system (ZEDS) of the Arleigh Burke class DDG 51 Flight IIA destroyer and improve upon current methods of zonal protection. Chapter 7 discusses the use of multifunction monitors (MFM) and incorporating nonintrusive load monitoring in a zonal electrical distribution system for enhanced protection.

This chapter will enumerate and describe the hardware components used to operate and control the generators. The system identification theory and design of the underlying controllers will also be explained in full detail. Screenshots of software and detailed schematics are accompanied in the text for illustrative purposes. All software code and other schematics are provided in Appendix G. An overall picture of the setup is shown in Fig. F-1.

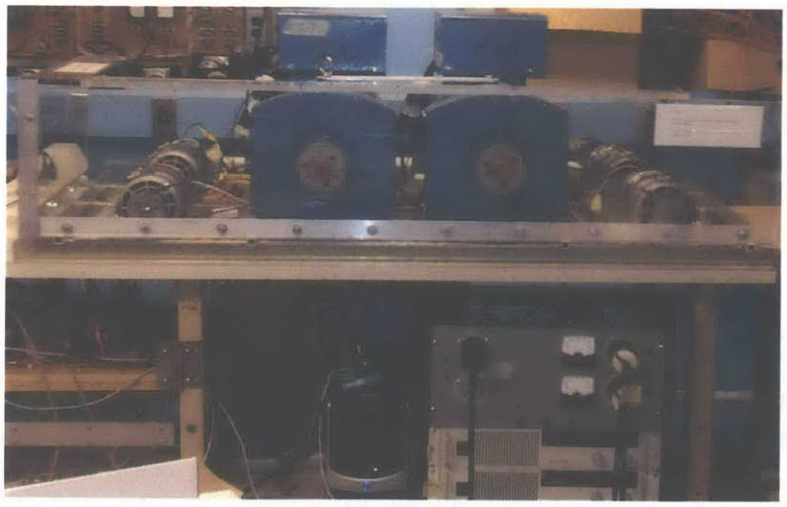


Fig. F-1: Scaled hardware model of a shipboard electrical distribution system.

F.1.1 Generators

The three-phase alternating current (A.C.) generators used in the hardware model of the shipboard electrical system are the 5 kilowatt STC-5 generators manufactured by Fujian Mindong Electric Manufacturing Co., Ltd. shown in Fig. F-2. The information listed on the nameplate is tabulated in Table F.1.

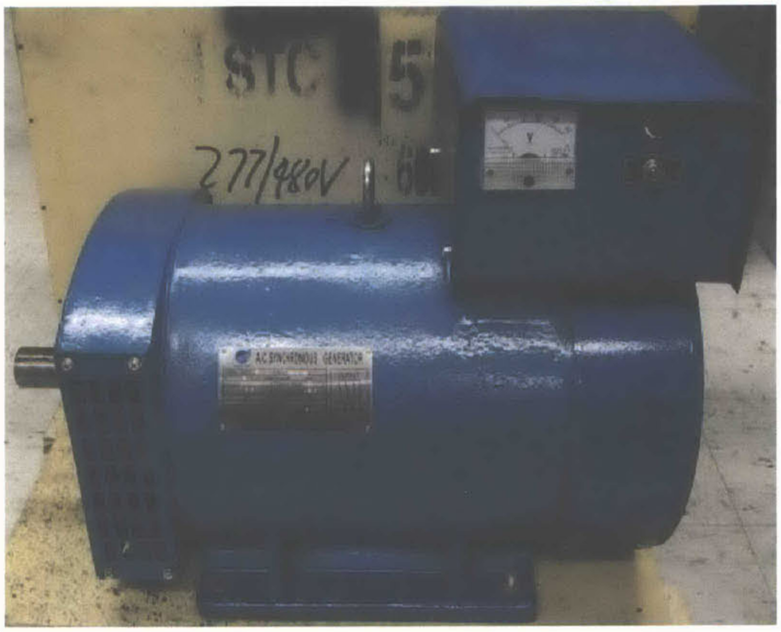


Fig. F-2: Picture of the STC-5 generator [101].

TYPE: STC-5	
P: 5 KW	COS ϕ 0.8
277/480 V	EXCIT. VOLT 80 V
7.5 A	EXCIT. CURR 3.7 A
60 Hz	INS. CL B IP21
1800 r/min	RAT. S1
PHASE: 3	DATE 2008.3
STANDARD JB/T8981-1999	

Table F.1: Nameplate data of the STC-5 generator.

More specifications on the three-phase A.C. synchronous generators are available at [102]. Since the synchronous speed of the generator is 1800 rpm, the generator is a two pole-pair ($p = 2$) machine. The slip ring outputs of the generator are connected in a wye-configuration. The generators will be used to provide a stable 60 Hz, 120 Vrms line-to-neutral (or 208 Vrms line-to-line) voltage. Feedback controllers, discussed in §F.6.2, are to designed to synchronize the phases of the two generators and to parallel them to drive larger loads. An automatic voltage regulator controller, discussed in §F.6.2, is designed to provide the 120 Vrms.

F.1.2 Prime Movers (Engine)

The engine or prime mover used to drive the generators are the permanent magnet DC motors shown in Fig. F-3.

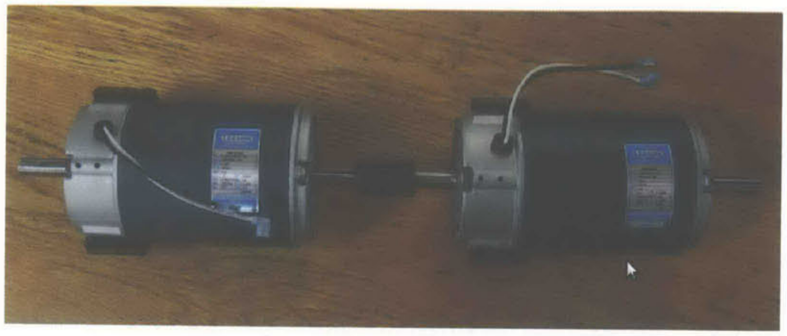


Fig. F-3: Coupled permanent magnet DC motors used as prime movers [101].

1991DAN			
MODEL	C42D340T7A		
CAT. NO.	098231.00		
VOLTS	120		
AMPS	4800		
H.P.	2	FR.	
DUTY	AIROVER	AMB.	40
ENCL.	OPEN	F.F.	1.05
INS.	F3	TYPE	DO
			K99E

Table F.2: Nameplate data of the permanent magnet DC motors.

The corresponding nameplate shown in Table F.2 states that each motor is rated at 2 HP. Thus, two DC motors are coupled in series mechanically and electrically to provide 4 HP of power needed to drive each generator.

Reference [101] provides a full description of the DC motors including armature resistance and motor constant. The measured armature resistance, R_A, and motor constant, K, of the two DC motors in series are

$$R_A = 1.869\ \Omega, \tag{F.1}$$

$$K = 0.432\ V \cdot s. \tag{F.2}$$

As shown in Fig. F-1, the prime movers are mechanically coupled to the generators with L Series timing belt pulleys and a 1/2 inch timing belt. The pulleys on the generators have $N_G = 32$ teeth and the pulleys on the DC prime movers have $N_P = 22$ teeth. For the two pole-pair generators and a desired electrical frequency of 60 Hz, the generators must spin at a mechanical speed of $\omega_G = 30$ rps. The speed ratio between the prime movers and generators is

$$N_P \cdot \omega_P = N_G \cdot \omega_G, \tag{F.3}$$

where ω_P is the mechanical speed of the DC prime movers. Plugging in the known values, the required speed of the prime movers is

$$\omega_P = 43.6364 \text{ rps.} \tag{F.4}$$

F.1.3 Power Supplies

Two Sorensen XFR 150-18 2700 W programmable power supplies, shown in Fig. F-4, are used to power the DC motors. Each has a maximum voltage rating of 150 Vdc and a maximum current rating of 18 A.

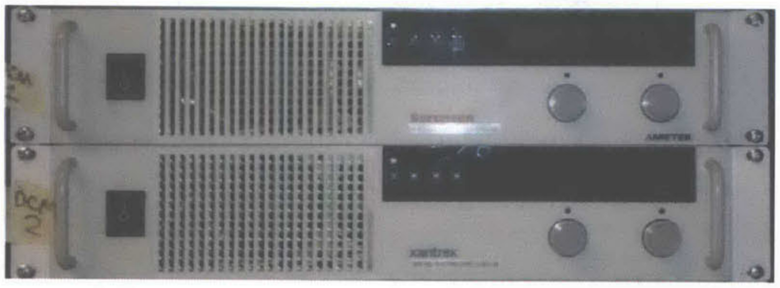


Fig. F-4: DC power supplies. 150 Vdc and 18 A and 2700 W

Fig. F-5: Field Coil DC power supplies. 150 Vdc and 7 A and 1050 W

The field coil windings in the two generators are excited by the Sorensen XHR 150-7 1050 W programmable power supplies shown in Fig. F-5. Both supplies are configured to allow for remote control from a computer. The computer sends a command voltage to each power supply and in turn, each power supply outputs a voltage to either a prime mover or a field coil winding that is proportional to the command voltage. More information is provided in §F.3.

F.1.4 Speed Encoder Mounts

For reliable speed control of the DC prime movers, a robust speed estimate is needed. The US Digital E6 Optical Kit Encoder single-ended output version (S-option) encoder is used to provide the speed estimate. A custom mount was built to attach the encoder to the shaft of the DC prime mover as shown in Fig. F-6. The encoder features two quadrature TTL squarewave channels: A and B and the codewheel inside the encoder provides 1800 pulses per revolution. The two channels provide the information that can be decoded to estimate both speed and direction of the shaft. The two quadrature channels are input to a PCI-4E PCI card, which is discussed in §F.2.2.

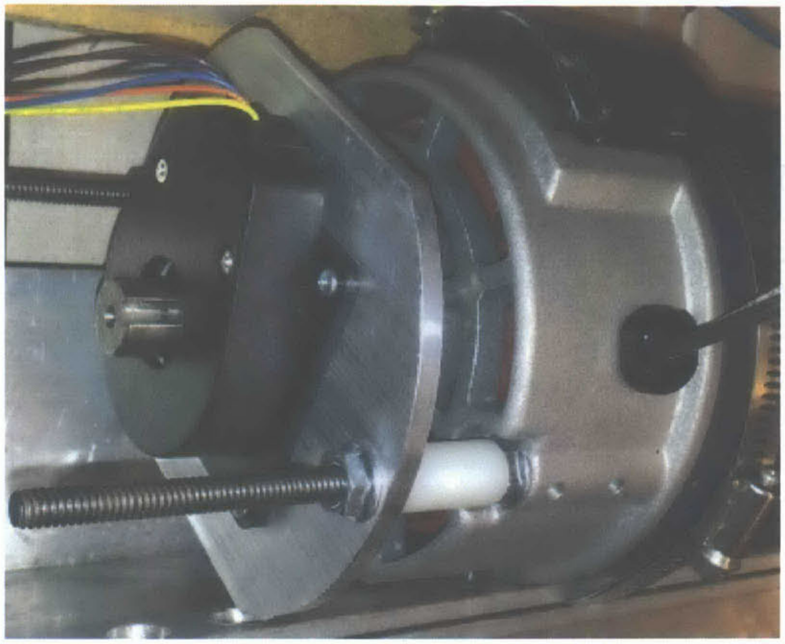


Fig. F-6: Custom mount with shaft speed encoder.

F.1.5 Operational Limits of Generators

The XFR power supplies used to power the DC motors have a 2700 W rating and the XHR power supplies used to power the field coil windings have a 1050 W rating. Therefore, each generator has 3750 W of power delivered to it by the power supplies. Using the nameplate power factor of 0.8 and a 120 Vrms, the maximum current supplied by the generator is 13 Arms. In reality, the power delivered to each generator is less than 3750 W. The max power rating is based on the maximum voltage and maximum current that the power supplies can deliver. However, the power supplies are not able to deliver both at the same time. Moreover, there is power loss in the belt and pulley mechanical coupling used to connect the DC prime movers to the generators.

F.2 Interfacing and Sensing

The previous section described the hardware used to model the shipboard electrical distribution system. This section will discuss the external components and circuitry that sense the currents and voltages from each generator and those that interface with the computer. Figure F-7 shows a block diagram of the interfacing and sensing layer in this testbench model. The computer makes use of the Matlab Simulink® environment to communicate with the interfacing and sensing layer as described in the next section.

F.2.1 Matlab Simulink® with The Real-Time Windows Target™

Simulink® is a powerful interactive graphical environment within the MATLAB programming environment used in model-based design for dynamic and embedded systems. It contains a library of customizable tools to interact with external data acquisition (DAQ) or multifunction cards to design custom filters and to implement feedback systems. While

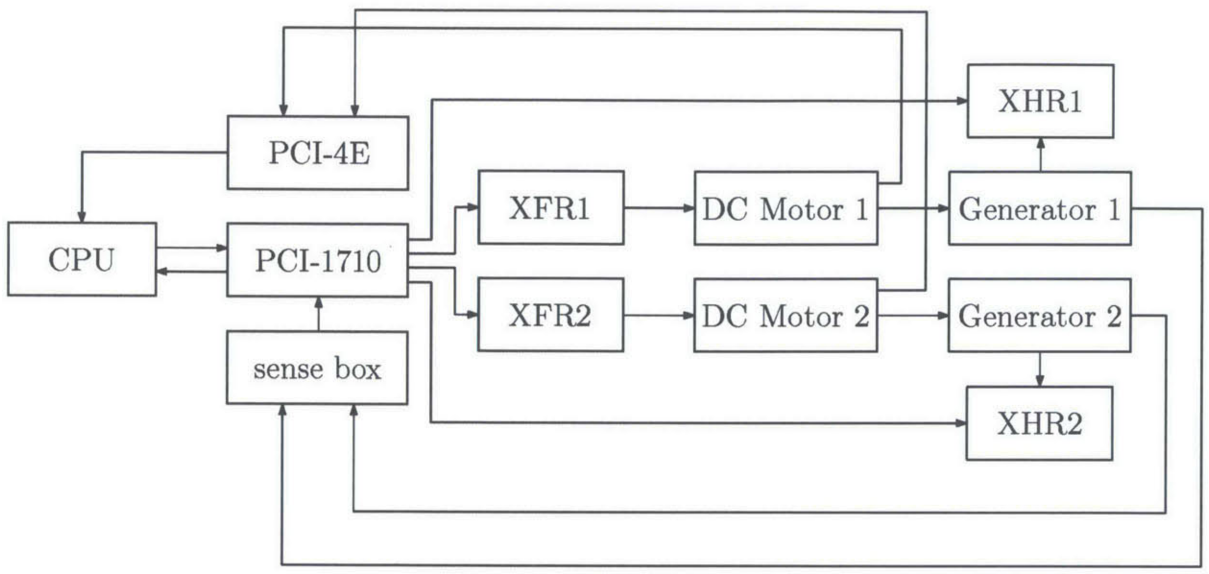


Fig. F-7: Interfacing and sensing layer

Simulink® is a powerful simulation tool, the Real-Time Windows Target™ extends its utility by providing the engine for real-time execution. The tool allows the user the build systems that can perform in real-time and communicate with a range input/output (I/O) boards. More information about the Matlab software, Simulink® and the Real-Time Windows Target™ can be found at the MathWorks® website [103].

F.2.2 PCI-4E-D PCI Interface Card

The US Digital PCI-4E-D PCI Interface Card interfaces with the Matlab Simulink® via the Real-Time Windows Target™ and is responsible for reading the two quadrature channels from each of the two encoders. A picture of the interface card is shown in Fig. F-8.

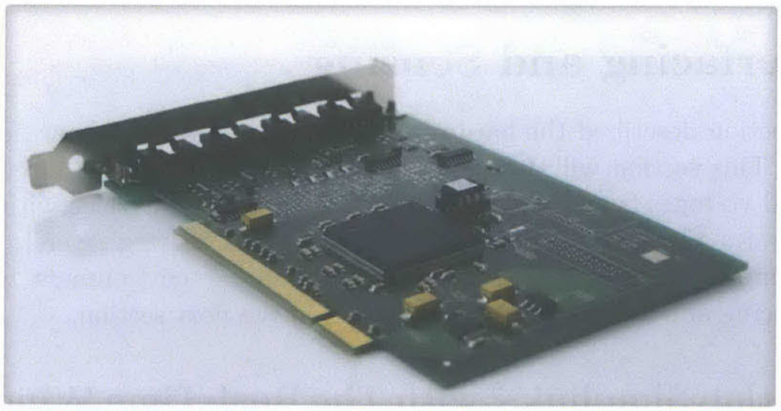


Fig. F-8: Picture of the PCI-4E interface card.

The two US Digital E6 Optical Kits Encoder single-ended output version (S-option) described in §F.1.4 provide the quadrature signals used in estimating the shaft speed of the prime movers. An adapter is used to convert the single-ended output of the encoders to a differential output to the PCI-4E card as differential outputs to provide better noise immunity.

The PCI-4E cards are configured in software to measure the counts of the quadrature signals from each encoder. Using its internal 33 MHz clock, the PCI-4E cards provides periodic 24-bit speed updates used by the simulink model. The C code used to initialize to the card is explained in further detail in §F.5.2.

F.2.3 PCI-1710 Multifunction Cards

The Advantech PCI-1710 is a 12-bit universal PCI multifunction card that provides programmable digital and analog inputs and outputs. A picture of the card is shown in Fig. F-9.

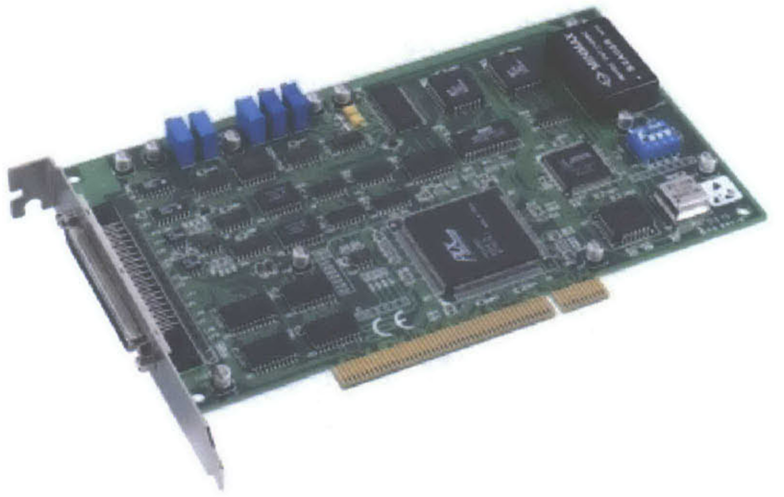


Fig. F-9: PCI-1710 multifunction card.

Each card contains an onboard 4-bit DIP switch used to set its address so that two cards can be differentiated by the computer during the initialization routine of the simulation. To differentiate between the two cards, the first card's switch was set to `0` and will be referred to as **`Card1`** in this text. The second card's switch was set to `1` and will be referred to as **`Card2`** .

A "sense-box" was built from a box enclosure equipped with current and voltage sensors to measure the appropriate signals from the three phases of each generator. A full discussion of the sense-box is provided in §F.2.5. Each card samples the three current and three voltage measurements from its respective generator on six of its 12-bit differential pair analog input channels.

The feedback control of the generator outputs required remote control of the DC motor power supplies to set the speeds of the DC prime movers and the field coil power supplies to excite the field coil windings in the generators. Each PCI-1710 card uses both of its analog output channels. One channel is configured to output 0-5 Vdc to command a Sorensen XFR 150-18 2700 W DC motor power supply (Fig. F-4) to set the speed of the prime movers. The other analog output channel is configured to output 0-5 Vdc to command the Sorensen XHR 150-7 1050 W (Fig. F-5) field coil power supplies to excite the field coil windings of the corresponding generator. More information about the power supplies can be found in §F.3.

F.2.4 SCSI Terminal Blocks

Two 68-pin SCSI screw terminal blocks are used to connect the input and output channels of the PCI-1710 cards with the sense-box and power supplies. These terminal blocks were placed inside a box enclosure and are shown in Fig. F-10.

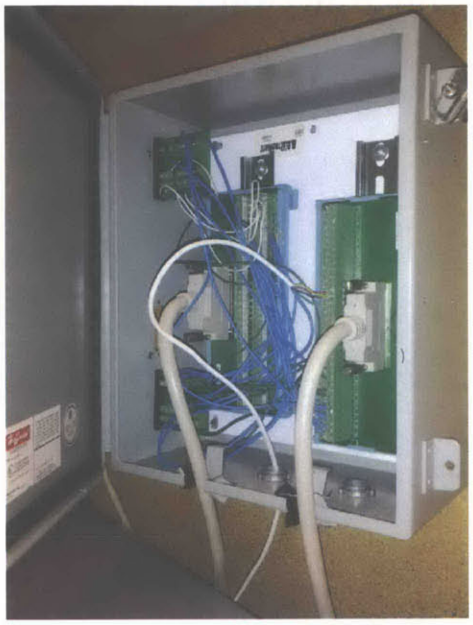


Fig. F-10: SCSI terminal blocks are used to connect the PCI-1710 cards with external components.

The PCI-1710 cards use analog inputs and outputs to interface with the power supplies. A table showing the connection between the power supplies and the PCI-1710 cards is shown in §F.3. Finally, the second PCI-1710 card, `Card2` , also uses three of its digital outputs to connect to relays as described in §F.4.1.

F.2.5 Current and Voltage Sensing

The currents and voltages of each of the three phases from each generator are sensed and passed onto the PCI-1710 multifunction cards. A photograph of the "sense-box" can be found in Fig. F-11

The line-to-neutral voltages of each generator are sensed by the LV 25-P Hall-effect voltage transducer that outputs a current proportional to the input voltage. The datasheet states that the the conversion ratio K_N is 2500:1000. The input voltage is placed across three 470 kΩ resistors in parallel and the resulting current is divided by 2.5 via the turns ratio to produce the output current. The output current is passed through a 750 Ω resistor to produce the voltage proportional to the actual voltage that will be sampled by the PCI-1710 cards. The relationship between the measured voltage V_{meas} and the generator voltage V_{gen} is

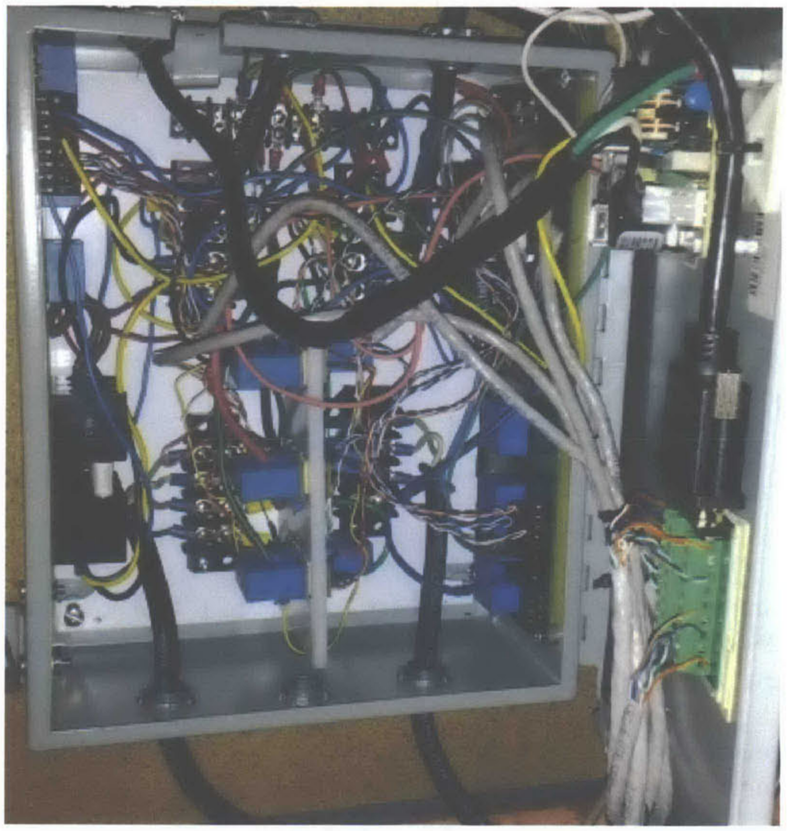


Fig. F-11: Photograph of sense-box

$$V_{meas} = \frac{V_{gen}}{470000/3} \cdot \frac{1}{2.5} \cdot 750 \tag{F.5}$$

$$= \frac{9}{4700} \cdot V_{gen}. \tag{F.6}$$

In steady-state operation, the generators will produce a nominal 120 Vrms line-to-neutral on each phase. Therefore, the output measured voltage will range from -0.324 Vdc to 0.324 Vdc. The analog input channels in the PCI-1710 cards are configured to -5 Vdc to 5 Vdc to properly handle the sensed voltage measurements.

> **NOTE:** The voltage transducers for each generator are placed on separate printed circuit boards. The printed board used for the second generator is a later revision that reversed the polarity of the input voltage so that sensed voltage is the negative of the actual voltage. This error is fixed in software.

The currents are sensed by LA 55-P current transducers that output a current proportional to the input current. The datasheet states that the the conversion ratio K_N is 1:1000 so that the output current is equal to the input current divided by 1000. The output current is passed through a 100 Ω external resistor to produce a voltage proportional to the current that is sampled by the PCI-1710 cards. The relationship between the measured voltage V_{cmeas} and the generator voltage I_{gen} is

$$V_{cmeas} = \frac{I_{gen}}{1000} \cdot 100 \tag{F.7}$$

$$= \frac{1}{10} \cdot I_{gen}. \tag{F.8}$$

The analog input channels are configured to -5 Vdc to 5 Vdc so currents in the range of -50 A to 50 A can be sensed reliably. As stated in §F.1.5, the generators can produce a maximum current of 13 Arms or currents in the range of -18.5 A to 18.5 A.

These sense-box current and voltage measurements will be used as inputs in the feedback control of the generator outputs to keep them paralleled. The schematics for differential pair for the voltage and current measurements are shown in Fig. F-12

F.3 Remote Analog Programming of Power Supplies

The two Sorensen XFR 150-18 power supplies, shown in Fig. F-4 on page 294 and used to drive the prime movers, can be controlled by an analog voltage command from a computer. These power supplies are configured to receive a 0-5 Vdc programming source voltage and no output current limit programming source by properly setting a series of switches and jumpers on the real panel. Refer to the user manual [104] for more information. Once properly configured, the power supply will output a voltage that is proportional to the programming voltage source ranging between 0 Vdc to 150 Vdc.

The field windings are powered by the Sorensen XHR 150-7 1050 W programmable power supplies shown in Fig. F-5 on page 294. These power supplies are similarly configured to

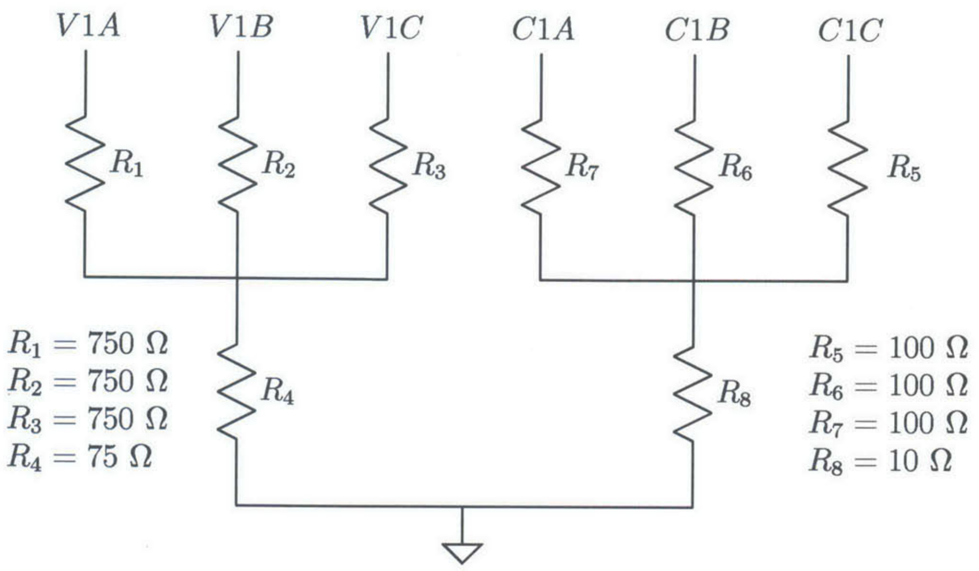


Fig. F-12: Schematics for the differential pair measurements for the voltages and currents.

receive a 0-5 Vdc programming source voltage and no output current limit programming source by properly setting a series of switches and jumpers on the real panel. Refer to the user manual [105] for more information.

The metal box enclosure containing the SCSI terminal blocks is equipped with a second 25-pin D-sub header to connect the PCI-1710 cards to a Jones strip located near the power supplies.

F.4 Front Panel

The front panel provides a user interface during operation of the synchronous generators. A picture of the front panel is shown in Fig. F-13.

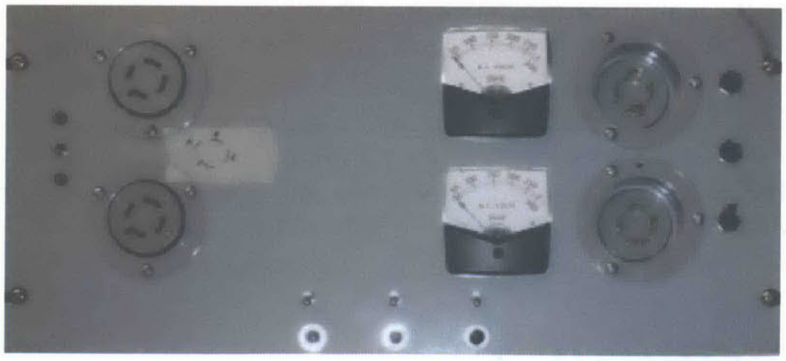


Fig. F-13: Front panel user interface.

The two generators are fitted with female sockets that connect to the two male outlet receptacles on the far right. Two analog voltage panel meters give the voltage readings of the generators. The controllers will be designed so that the generators provide 120 Vrms service.

The switch on the far left is the C/M switch which selects either Computer or Manual control. In the Computer control setting (up position), the green LED above the switch is lit and the relay used to parallel the generators and the relays used to power external loads with the generators are set automatically by the computer during operation when requested by the user in software.

In the manual control setting (down position), the yellow LED below the switch is lit and the user can manually control the setting of the relays via the manual relay switches. The two female flanged receptacles on the left side of the front panel are used to connect any loads to the two generators. The top receptacle is the switched output of generator 1 through relay 1 and the bottom receptacle is the switched output of generator 2 through relay 2.

The manual relay switches are located at the bottom of the panel and above the status LEDs. The first manual switch closes the output relay so that any load plugged into the receptacle is powered by generator 1. The third manual switch does the same for generator 2. The switch in the middle is the paralleling switch. There are three green relay status LEDs located below each manual switch. When a particular relay is closed, the corresponding LED is lit. Finally, there are three relay kill switches located on the far right. For safety reasons, these switches can be use to immediately open the corresponding relay if needed.

F.4.1 Circuit Board

A custom circuit board was designed to provide the functionality of the front panel user interface. The wiring hidden behind the front panel include a circuit board used for the interface, a power supply, the Jones strip discussed in §F.3, and a single-phase 120 V service. The power supply is a third-party component that takes in a 60 Hz, 120 Vrms voltage and outputs +12 Vdc, -12 Vdc and 0 Vdc ground used to power the front panel circuit board. The schematic for this board is provided in Appendix G.

The relays use a nominal voltage of 24 Vdc across the terminals to close. The 24 Vdc supplied by the +12 Vdc and -12 Vdc would suffice. To keep the consistency, the -12 Vdc became the common ground on the board. A 5 V voltage regulator provided -7 Vdc which is 5 volts higher than -12 Vdc to power all the logic chips used in the circuit board.

The three CPU commands to control the relays are sent through an photocoupler to electrically isolate the computer from the relays. The outputs of the photocoupler are inverted for consistency before they are sent to a mux. The mux selects between these CPU commands or the three debounced manual switches based on the position of the C/M switch. The outputs of the mux are sent to a push-pull driver to produce the 24 Vdc signal to close the relays when commanded. The push-pull driver outputs are sent through an override kill switch and this signal is sent directly to the relays. External circuitry is also provided for the CPU or manual LED and the relay status LEDs.

F.5 S-Function Block

The MATLAB S-function is a user-definable block written in C. The block can have an arbitrary number of inputs and outputs and can communicate with PCI cards or any other memory-mapped data acquisition cards. For the purposes of the dual generator setup, an

S-function block is used to communicate with the PCI-1710 cards and the PCI-4E card to acquire needed data.

All of the code written for the S-function is located in the Appendix G. The central file is `plant_controller_960_PCI4E.c`. The S-function block in the Simulink model points to this file.

F.5.1 Initialization

The first method in this file is the `mdlInitialSizes`. When a simulation is started, Simulink uses this method determine the number of inputs, outputs, working vectors and other states for the S-function block.

The block declares seven inputs. The first two inputs are the commanded voltages from speed controller that are used to remotely control the power supplies to the DC prime movers to maintain 60 Hz operation. The next two inputs are the commanded voltages from the field coil controller used to control the power supplies to the field coil windings to maintain 120 Vrms operation. The last three are user inputs to parallel the generators and to open the relay connecting each generator to its load.

The block declares numerous outputs, however the first five are the most critical. The others are used for debugging or for data display. Of the five main outputs, the first two are the speeds of the DC prime movers. The speeds are sent directly into the speed controller. The third output is the phase difference between the voltages of the two generators. This phase difference is sent to the speed controller to be used to parallel the generators. The final two outputs are the measured voltages of the two generators that are sent to the voltage controller.

The final step of the initialization method is to set up working vectors for needed variables. A pointer work vector, explained in the Matlab Documentation Center [106], is necessary to address the memory-mapped PCI-4E card and communicate with it. The defined data work vector is a workspace of variables including the following: the base address of the two PCI-1710 cards, the base address of the PCI-4E card and the length of the allocated memory in the PCI-4E card. The rest of the data work vector is used for variables that may change at each time step. These include a timestamp from the PCI-4E card, counts from each of the four encoders, a vector of the last 100 speed measurements of each generator, the last 20 voltage and current measurements of each generator and the current settings of all relays.

Finally, a `mdlInitializeSampleTimes` routine defines the sample time. With 60 Hz generators, a full cycle of the voltage sampled at $f_s = 960$ Hz would be expressed in 16 samples. As will be shown, to compute the reactive power of the generator a quarter-cycle phase offset is needed. With 16 samples, a quarter-cycle offset is simply 4 samples. Note that 20 samples of the current and voltage measurements are stored in the working vector for this reason. With a sample frequency of 960 Hz, the sample time is $T_s = 1/960$.

F.5.2 Start

The `mdlStart` function is called only once at the start of the simulink model execution. In this function, the PCI cards are initialized and configured.

The first two user defined subroutines called are `find_pci_1710_IOBaseAddresses` and `get_pci_4E_Memory_Info`. These subroutines are defined in the `PCI_1710_4E_Comm.c` file. The PCI-1710 cards are Input/Output PCI cards and their configuration space contains a

Base Address Register (BAR). All input/output registers are defined by an offset from the BAR. The offsets of the registers are found in the user manual. The subroutine, adapted from [107], loops through the all PCI peripherals on the computer to search for the PCI-1710 cards by reading the Device and Vendor IDs. The two PCI-1710 cards are differentiated by an on-board DIP switch.

The PCI-1710 cards are each equipped with two analog output channels, eight differential-pair analog input channels, sixteen digital input and sixteen digital output channels. The analog output channels are configured with an internal reference voltage of $5V$, which allows each channel to output between $0V$ - $5V$ to remotely control all the power supplies. Only six of the differential-pair analog input channels are used on each card for the three current and three voltage measurements of the corresponding generators. Each differential-pair is configured to read $-5V$ to $5V$.

The next step of the start function is to initialize the work vectors. A 60 Hz cosine waveform and a 60 Hz sine waveform sampled at 960 Hz is stored to be used to calculate real and reactive power. Other vectors in the work vector are initialized to zero.

The PCI-4E card is a memory-mapped device which requires different configuration than the port-mapped PCI-1710 card. Versions of MATLAB earlier than R2012b cannot be used to communicate with memory-mapped devices. Using the BAR location of the PCI-4E and length of the card's memory, MATLAB R2012b can use `RTBIO_MapDeviceMemory` to map the device memory and store the address in the pointer work vector. Once the BAR location has been stored, the PCI-4E card is initialized. The encoder is comprised of a wheel designed to provide 1800 counts per revolution and the encoder outputs a quadrature signal. The PCI-4E card is configured to properly receive and decode the signals from the encoders. More information about configuring the PCI-4E card can be found online [108]. The final task in the start routine is to open all generator relays.

F.5.3 Outputs

The `mdlOutputs` function is called every time step to compute the outputs of the S-function.

The speeds of the DC prime movers are calculated by reading the elapsed count and elapsed time from the PCI-4E card. Using an internal 33 MHz clock, a constantly running timestamp is read and stored in the Data Work Vector at each time step. An accurate elapsed time is readily computed by comparing the current timestamp with that of the previous time step. Dividing the elapsed count by the counts per revolution and by elapsed time provides the speed in rotations per second.

The six currents and six voltages measurements are sampled by the two cards. Note that the voltage sensors for the second generator were placed in reverse so a simple software fix is needed is invert them. The currents transducers and voltage sensors were characterized so that the sensed values can be converted to their actual values. To calculate the real (P) and reactive (Q) power the following equations are used,

$$P = \frac{1}{16}\sum_{k=1}^{16} V[k]\cdot I[k] \tag{F.9}$$

$$Q = \frac{1}{16}\sum_{k=1}^{16} V[k]\cdot I[k+4] \tag{F.10}$$

where $V[k]$ and $I[k]$ are the measured voltage and current waveforms.

The phase angle difference of the generators is the third critical output. To do so, the first harmonic of the Fourier coefficients (a_1 and b_1) are computed. A full cycle of a 60 Hz sine wave and cosine wave sampled at 960 Hz are stored in the data work vector. The Fourier coefficients are calculated as

$$a_1 = \frac{\sqrt{2}}{16}\sum_{k=1}^{16} s[k]\cdot \sin[k] \qquad \text{(F.11)}$$

$$b_1 = \frac{\sqrt{2}}{16}\sum_{k=1}^{16} s[k]\cdot \cos[k] \qquad \text{(F.12)}$$

where $s[k]$ is the current or voltage signals, sin is the stored sine vector and cos is the stored cosine vector.

The phase-locking controller and the voltage controller use only the phase difference and rms voltage of the voltages on the B phases of each generator. The phase, ϕ_B, of each generator is found by using the Fourier coefficients as

$$\phi_B = \tan^{-1}\left(\frac{a_1}{b_1}\right). \qquad \text{(F.13)}$$

The rms voltages of the B phases is calculated as

$$V_B\text{rms} = \sqrt{a_1^2 + b_1^2}. \qquad \text{(F.14)}$$

F.5.4 Update

The **mdlUpdate** function is called every time step to update states and performing other meaningful tasks. In this function, the power supplies are controlled based on the outputs of the speed and voltage controllers. The necessary voltages are commanded from the analog outputs of the PCI-1710 cards. If the state of the three relays have changed, the digital outputs are toggled to open and close the relays.

F.5.5 Terminate

The **mdlTerminate** function is called at the termination of the simulation. The PCI-4E card is unmapped and the memory is freed up. All power supplies are commanded to output $0V$ and all relays are opened.

F.6 System Identification

Identification of nonparametric models based on frequency-response data allows for effective control system design. Matlab provides software tools to design feedback control around a known plant to obtain the desired output. The objective is to characterize the behavior of the generators, the power supplies and any delays in the software as a plant of a known frequency response. Once a plant model is known, a controller can be designed around it to maintain 60 Hz, 120 Vrms operation.

Figure F-14 shows a block diagram of the overall system. The overall system (or plant) was broken down into two parallel subsystems. The top (speed) subsystem $H_S(s)$ corre-

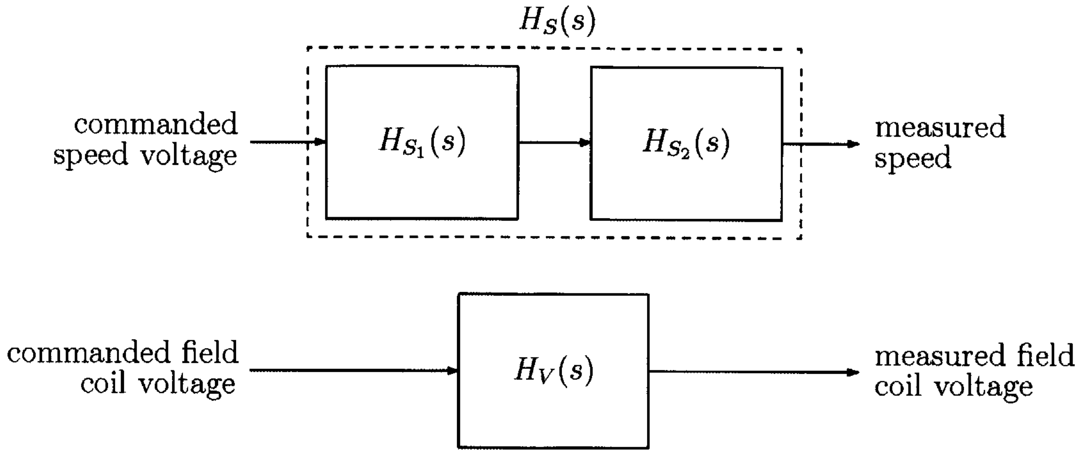


Fig. F-14: Block diagram of overall system (or plant).

sponds to how the software would command the power supplies to output a certain voltage which would turn the DC prime movers. The commanded voltage to monitored voltage transfer function $H_{S1}(s)$ will account for any differences between the voltage commanded by the computer and to the actual voltage of the power supplies. The monitored voltage to DC prime mover speed transfer function $H_{S2}(s)$ is the second component of the speed subsystem. This transfer function describes how the measured speed of the the DC prime movers vary with voltage. Combining the two transfer functions above will yield the commanded voltage to DC prime mover speed transfer function $H_S(s)$.

The bottom (voltage) subsystem $H_V(s)$ corresponds to how the software would command that the power supply output a certain voltage to excite the field coil windings in the generator to output a certain voltage. The commanded voltage to monitored voltage transfer function $H_V(s)$ will account for any differences between the voltage commanded by the computer and the actual voltage sent by the power supply.

In order to maintain 60 Hz, 120 Vrms operation, feedback control is necessary. Once the plant's frequency response is estimated, a feedback control system can be designed for certain design specifications and requirements. The following sections will discuss how the frequency-response of the system was estimated and how the plant controllers were designed.

F.6.1 One Frequency at a Time

The method of one frequency at a time requires that a sinusoid of a known frequency is applied to the system and the magnitude and phase of the output are recorded [109]. This process can be repeated for all desired frequencies to obtain the frequency-response. A chirp signal is a modulated sine wave with a frequency that can grow over a set interval. Using a chirp signal as an input to a system, the frequency response can be computed at once.

As mentioned in §F.1.2, the DC prime movers must move at a speed of 43.6364 rps to maintain a 60 Hz electrical frequency in the generators. A chirp signal that is exponentially swept over frequencies between 0.05 Hz and 50 Hz was used around an open loop operating point near 43.6364 rps.

The chirp signal starts at a instantaneous frequency of 0.05 Hz and exponentially grows to 50 Hz. An exponential sweep rather a linear sweep was chosen as most of the important structures of the frequency response will be prevalent at low frequency and thus high resolution around 50 Hz was not required.

To estimate $H_{S_1}(s)$, the commanded voltage to monitored voltage transfer function, a simple experiment was conducted where the commanded voltage to the power supply for DC prime mover is the chirp signal. The DC prime mover is mechanically coupled to an unenergized generator with a belt and pulley. The monitored voltage reported by the power supply was recorded. With an input signal as the chirp signal and the output signal as the monitored voltage, the system transfer function is estimated using the `sys_transfer_function_finder` function in Appendix G.

To simplify the modeling, a low-order model was estimated using the lowest amount of states that best-fit the actual data over the desired frequency range between 0 and 60 Hz. The low-order continuous-time transfer function is

$$H_{S1-low}(s) = \frac{950}{(s+475)(0.053s+1)}. \tag{F.15}$$

Figure F-15 shows the frequency response for the system governing the commanded voltage to monitored voltage of the power supplies driving the DC prime movers with the solid line. The low-order transfer function is shown as the dotted line and the frequency response closely matches the actual data.

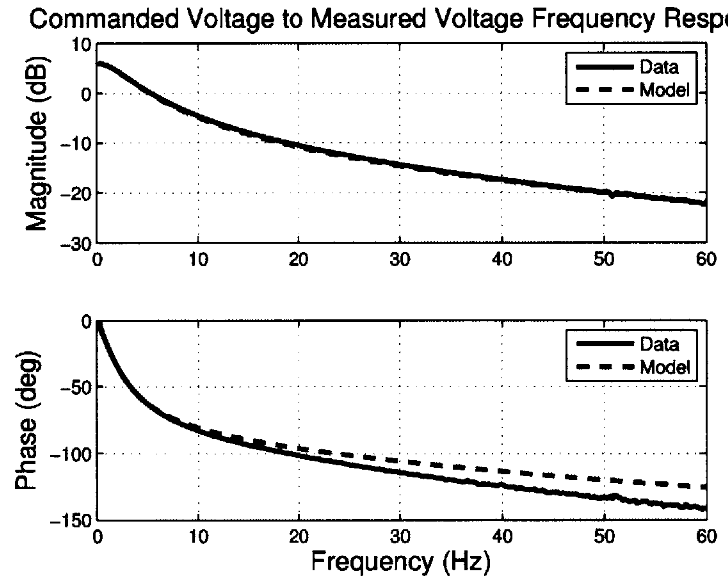


Fig. F-15: Frequency response for the commanded voltage to monitored voltage. The measured response is shown with the solid line. The low-order model is shown with the dotted line.

The same process was done to model $H_{S2}(s)$, the transfer function of the monitored voltage to speed of the DC prime mover. A chirp signal was used but the reported voltage from the power supply was recorded as the input and the output is the speed of the DC prime mover.

The corresponding low-order continuous-time transfer function is

$$H_{S2-low}(s) = \frac{30000(s^2+0.4147s+1720)}{(s+150)(s+3.8)(s^2+10s+1.92e4)}. \tag{F.16}$$

Figure F-16 shows the frequency response for the system governing the monitored voltage to speed with the solid line. The low-order transfer function is shown as the dotted line and the frequency response closely matches the actual data.

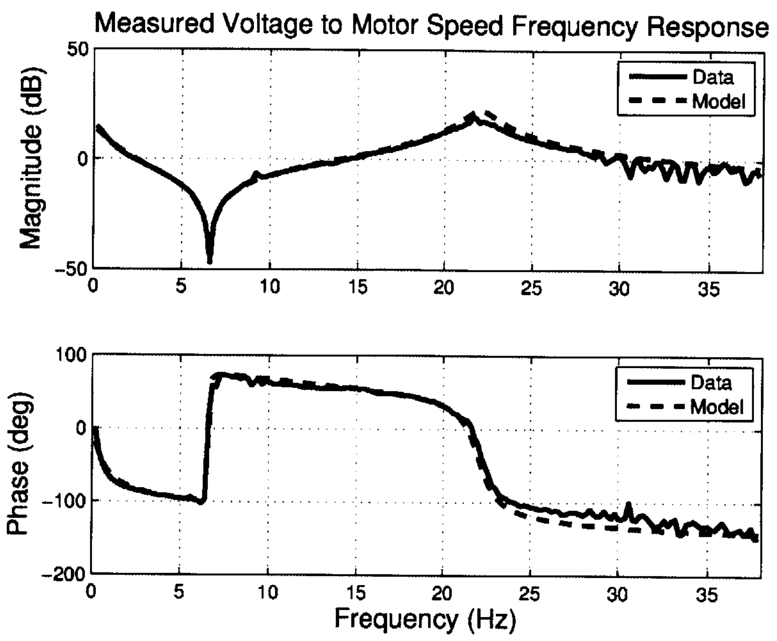


Fig. F-16: Frequency response for the monitored voltage to speed. The measured response is shown with the solid line. The low-order model is shown with the dotted line.

The overall transfer function $H_S(s)$ can be found by multiplying (F.15) and (F.16) as

$$H_S(s) = \frac{28.5e^6(s^2 + 0.4147s + 1720)}{(s + 475)(0.053s + 1)(s + 150)(s + 3.8)(s^2 + 10s + 1.92e4)}. \tag{F.17}$$

Figure F-17 shows the frequency response for the top branch in Fig. F-14 governing the commanded voltage to speed with the solid line. The low-order transfer function is shown as the dotted line and the frequency response matches closely with the actual data.

To model $H_V(s)$ in Fig. F-14, the transfer function of the commanded voltage to field coil voltage, the DC prime mover was set to run open loop near 43.6364 rps. A chirp signal was used as the commanded voltage to the power supply driving the field coil around an operating point to drive 120 Vrms to a 900 W load.

The resulting field coil voltage was measured and recorded. The transfer function for $H_V(s)$ is shown in Fig. F-18 as the solid line. The corresponding low-order continuous-time transfer function, shown as the dotted line, is

$$H_V(s) = \frac{540.3}{s + 541.1} \tag{F.18}$$

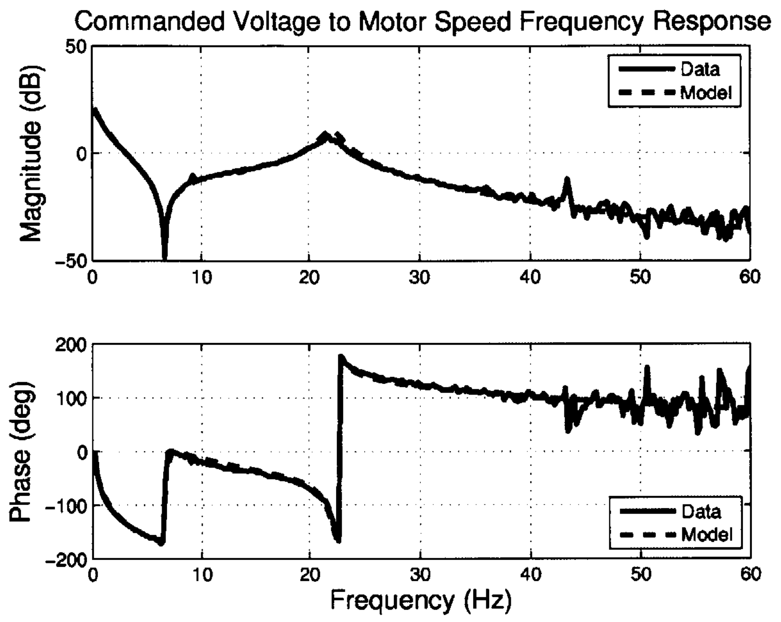


Fig. F-17: Frequency response for the commanded voltage to speed. The measured response is shown with the solid line. The low-order model is shown with the dotted line.

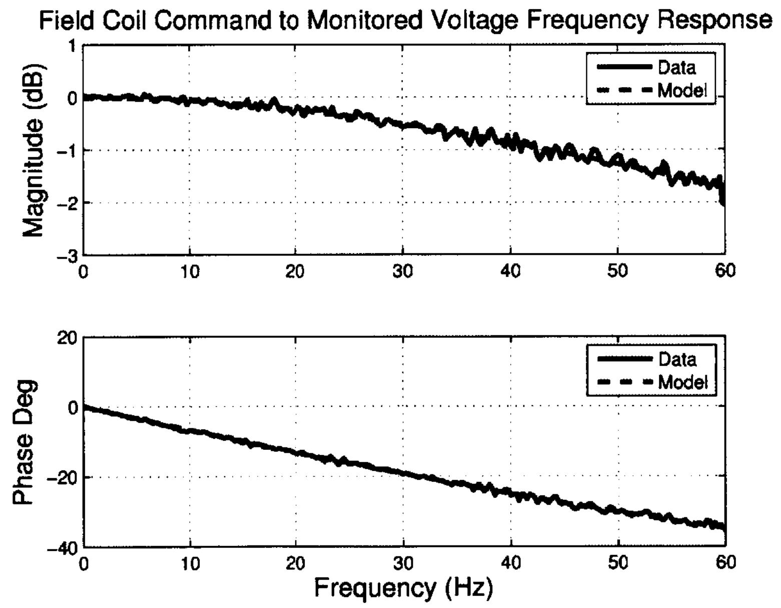


Fig. F-18: Frequency response for the commanded voltage to field coil voltage. The measured response is shown with the solid line. The low-order model is shown with the dotted line.

F.6.2 Matlab Controller Design

With the system identification process complete, the controllers can be designed. The simplest speed controller would include an integrator in order to maintain zero steady-state speed error regardless of the load conditions. This controller is known as an isochronous speed governor. For single generator systems, an isochronous speed governor would work but it cannot be used with other speed controllers are connected to a power system because it will not allow for load sharing [110].

The goal for this experimental setup is to have both generators running at 60 Hz in phase and sharing real and reactive power. To do so, speed droop or speed regulation is needed. In this setup, the speed droop will be a proportional speed controller. Removing the integrator will create a steady-state error but an integrator can be placed outside of the speed droop to compensate. With both generators in parallel, they will share the same electrical frequency and will share loads proportionally to their rating. Specific details and derivations can be found in [110]. Both generators are equivalent in rating, so they will both share loads equally.

Figure F-19 shows the block diagram of the overall speed controller. The inner loop

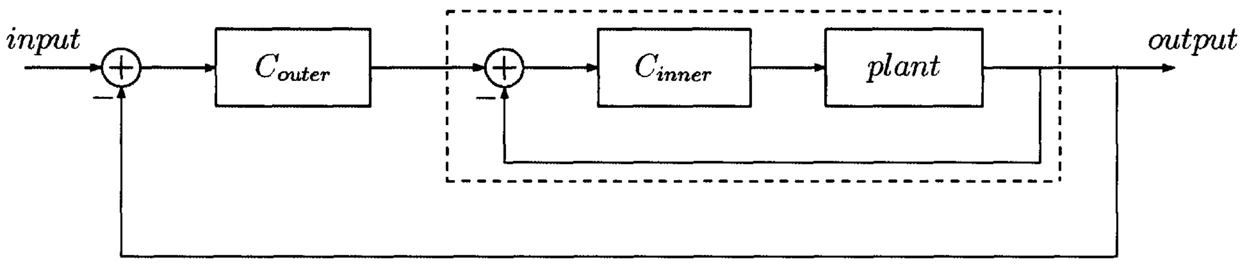


Fig. F-19: Overall block diagram of speed controller.

compensation is the speed droop controller that will allow for equal real power sharing. This inner loop is designed around a plant with the frequency response expressed in (F.17). It needs to be designed such that it can follow its input quickly and without any instability. The outer loop views the inner loop and its corresponding plant as one larger plant. The outer loop includes an integrator to maintain 60 Hz operation.

Using the Matlab Single Input Single Output SISO Tool, the droop controller was designed in the continuous-time space. The tool allows the user to analyze phase margins and step responses so that certain design specifications can be met.

The continuous-time transfer function of the speed controller inner loop is

$$H_{S_{inner}}(s) = \frac{0.19848(s+401.9)(s+12.85)}{(s+85.54)(s+42.59)}. \tag{F.19}$$

With the inner loop transfer function designed, the outer loop transfer function can be designed on the larger plant. The controller designed for this larger plant is

$$H_{S_{outer}}(s) = \frac{9.1936(s+844.381)(s+18.947)}{s(s+99.072)(s+103.584)}. \tag{F.20}$$

A similar approach was taken to design a controller for the field coil voltage or automatic voltage regulator (AVR). In single generator operation, a simple lead-lag compensator is a popular choice. However, for dual generator operation running in parallel, a proportional controller such as the speed droop was chosen to provide equal reactive power sharing. There will be a steady-state error, but an outer integrator will be used to handle this error and maintain 120 Vrms.

The transfer function for the inner loop is

$$H_{V_{inner}}(s) = \frac{3.8601e7(s + 56.256)}{(s + 13.536)} \tag{F.21}$$

and the transfer function for the outer loop is

$$H_{V_{outer}}(s) = \frac{2}{25} \cdot \frac{1}{s} + \frac{1}{10}. \tag{F.22}$$

F.6.3 Simulink Implementation

The controllers designed §F.6.2 were implemented in the Simulink environment. Figure F-20 shows the inner workings of the speed controller block. The block has four inputs. The first two are the speeds of the DC prime movers provided by the s-function block as described in §F.5.1. The third input is a measurement of the difference in phases of the measured generator voltages. The fourth input is a user-provided tuning factor to fine tune the real power shared by the generators in the event loading varies in real-time. The two outputs provided by the droop speed controller block are the commanded voltages that will eventually be sent to the power supplies connected to the DC prime movers.

The two droop controllers are designed such that they can follow the commanded input reference without instability. The inputs for these droop controllers are the errors between the reference speed of 43.6364 and the measured speed of each generator.

Since the control of the generators is done using software, discrete-time models must be used. The sample rate for the software was chosen at $f_s = 960$ Hz. For a 60 Hz voltage, one line cycle of the voltage can be expressed using 16 samples. Therefore, the continuous-time transfer functions are converted to discrete-time using the Forward Euler transformation and a step size of $T = 1/f_s = 1/960$. To convert continuous-time transfer functions, the following expression can be used

$$H(z) = H(s)\Big|_{s=(z-1)/T}\ . \tag{F.23}$$

The discrete-time transfer function for the droop controller is based on the continuous-time representation described in (F.19). Slight parameter modifications were then made to increase performance. The final transfer function $H_{\text{speed\_inner}}(z)$ is

$$H_{\text{speed\_inner}}(z) = \frac{0.1984(z - 0.9866)(z - 0.5814)}{(z - 0.9556)(z - 0.9109)}. \tag{F.24}$$

As previously mentioned, the droop controller is required for parallel operation but there is a steady-state error. A speed steady-state error offset of 14.7 is added for course compensation of this error. If both generators are precisely the same and all other hardware components are the same, then the two droop controllers should share real power equally [109]. Unfortunately, that is not usually the case. To compensate for these imperfections,

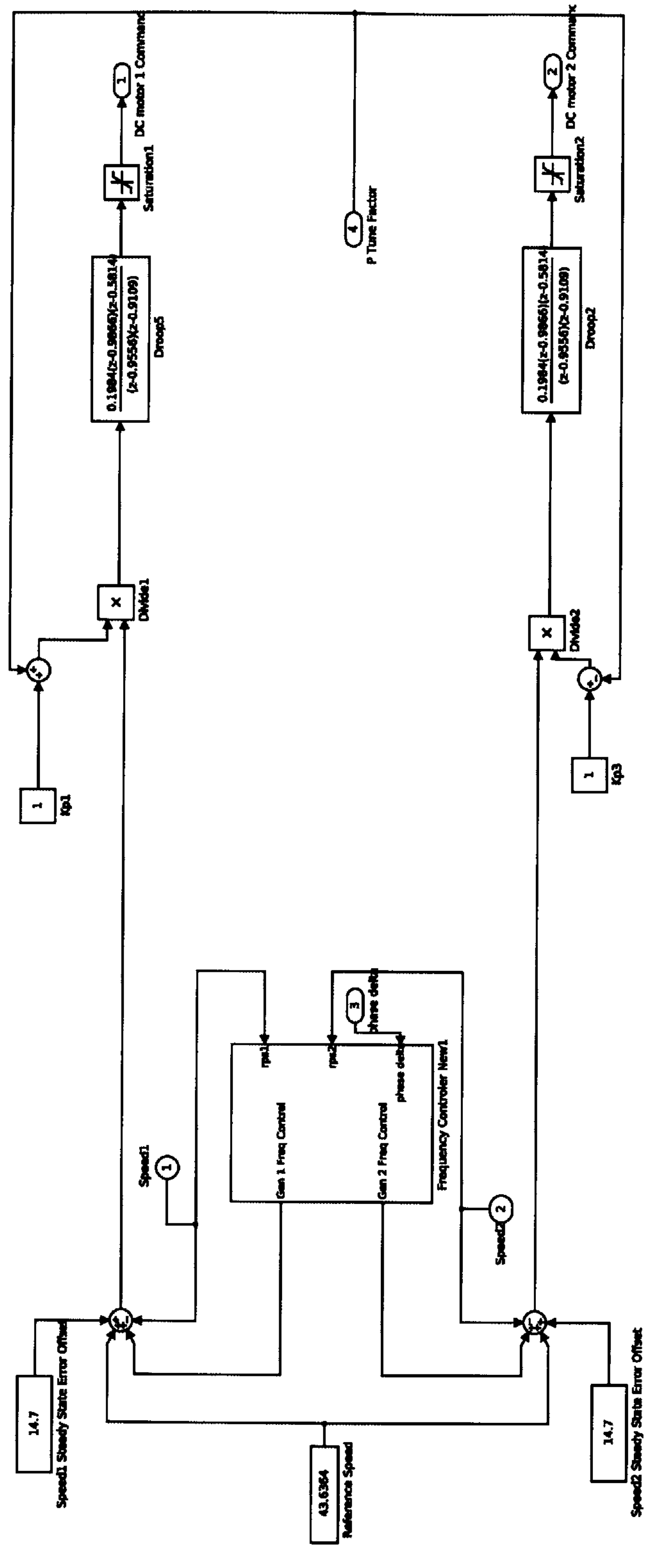


Fig. F-20: Inner frequency block within the speed controller.

the user-provided tuning factor finely adjusts the gains of each droop controller to share real power.

An outer loop is required for fine tuning, maintaining 60 Hz operation and running the generators in parallel if required. The outer loop control is contained in the Frequency Controller block shown in Fig. F-20. It takes as input the two speeds of the generators and also the phase difference of the voltages of the two generators. It outputs the two frequency control feedback parameters. The inner workings of the Frequency Controller block is shown in Fig. F-21.

The bottom two branches of the controller operate on each generator separately when the generators are *not* in parallel mode. Each branch is identical and is derived from the continuous-time controller with an integrator expressed in (F.20). Once again, the a Forward Euler transformation in (F.23) is used with slight parameter changes and using partial fractions to separate the integrator from the other terms, the final transfer function $H_{\text{speed\_outer}}(z)$ is

$$H_{\text{speed\_outer}}(z) = \frac{-0.0053535(z-2.02)}{(z-0.8968)(z-0.8921)} + \frac{0.0149}{z-1}. \tag{F.25}$$

Separating the integrator allows for the compensation of integral windup. The power supplies used to operate the DC prime movers can only output voltages in the range of 0 - 150V. In other words, the power supply cannot supply infinite voltage. If there is a large error between the reference speed and actual speed, the controller can output a large value that is greater than the physical capabilities of the power supplies. Therefore, the integral term is accumulating a significant error during this "windup" that the power supply cannot overcome. The accumulated error causes an overshoot further increasing the error even as the power supply is able to "unwind" the error. The process is then repeated as the integral accumulates error once more.

To prevent this integral windup, the discrete-time integrator is set up with a saturation limit that is below the limits of the power supplies. If the error is very large, the integrator saturates to an upper limit and stays at this level while the error decreases. During this unwinding, the power supply outputs a voltage that is below its maximum capability.

When the generators are not in parallel mode, a phase matching controller block uses a proportional-integrator (PI) controller shown in Fig. F-22 to match the phases of the two generators. The transfer function for this controller is

$$H_{phase}(z) = 2 + \frac{1}{320} \cdot \frac{1}{z-1}. \tag{F.26}$$

When the generators are in parallel mode, the top branch in Fig. F-21 is activated. The parallel top branch uses the same controller as the two separate branches with two differences. The first is that the input uses the error between the reference speed and the average speeds of the two DC prime movers. The second difference is in the integrator. When switching to parallel mode, the integrator would start with zero error which might cause a bumpy transition during operation. To smooth the transition, the integrator is initialized with the average of the two errors of the integrators in the bottom two branches.

Going back to Fig. F-20, a saturation block is added to limit the output within the capabilities of the power supplies connected to the DC prime movers.

The inner and outer voltage controllers are implemented in a similar fashion. The Simulink model for the voltage controllers is shown in Fig. F-23.

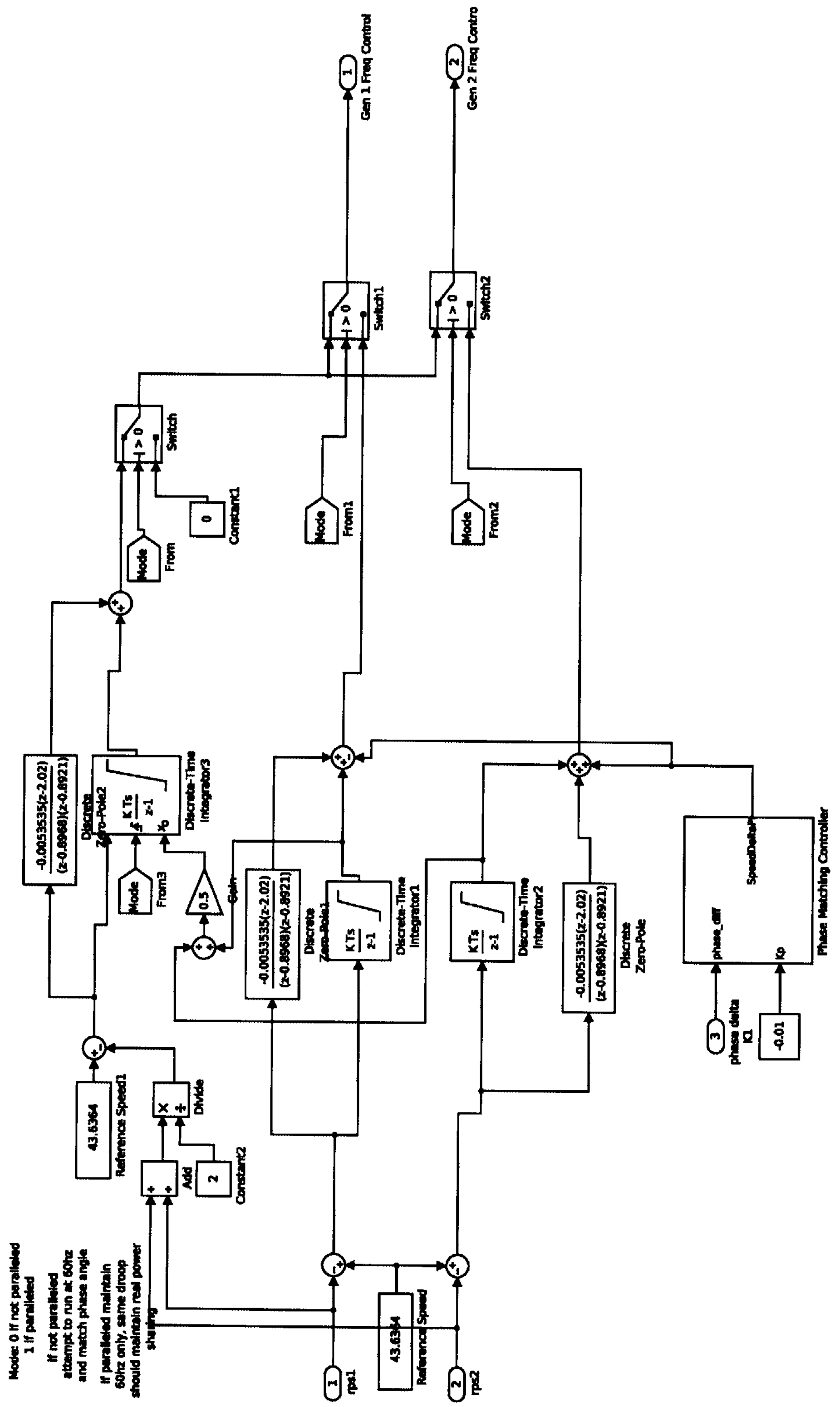


Fig. F-21: Speed controller outer loop block.

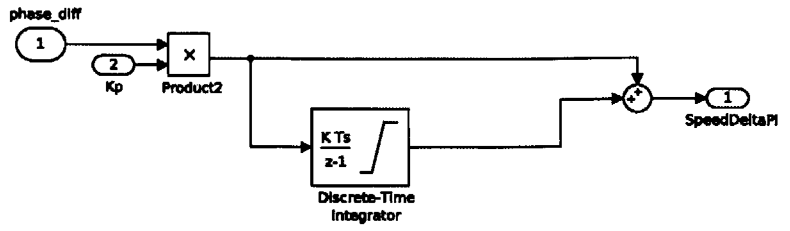


Fig. F-22: Phase matching controller to parallel generators.

The block has three inputs. The first two are the measured voltages of the generators provided by the s-function block as described in §F.5.1. The third input is a user-provided tuning factor to fine tune the reactive power shared by the generators in the event loading varies in real-time. The two outputs provided by the droop field coil controller block are the commanded voltages that will be eventually be sent to the power supplies connected to the field coil windings of the generators.

The two droop controllers are designed such that they can follow the commanded input reference without instability. The inputs for these droop controllers are the errors between the reference voltage of 120 Vrms and the measured voltage of each generator.

The discrete-time transfer function for the droop controller is based on its continuous-time representation described in (F.21). Therefore, the final discrete-time transfer function $H_{\text{voltage\_inner}}(z)$ is

$$H_{\text{voltage\_inner}}(z) = \frac{0.023875(z - 0.9414)}{(z - 0.9859)}. \tag{F.27}$$

This droop controller is required to share reactive power but there is a steady-state error. A speed steady-state error offset of 5 is added for course compensation of this error. A user-provided tuning factor finely adjusts the gains of each droop controller to share reactive power.

An outer loop is required for fine tuning and maintaining 120 Vrms operation. The outer loop control is contained in the Field Coil Frequency Controller block shown in Fig. F-23. It takes as input the two voltages of the generators and outputs the two command voltages sent to the corresponding power supplies. The inner workings of the Field Coil Frequency Controller block is shown in Fig. F-24.

The bottom two branches of the controller operate on each generator separately when the generators are *not* in parallel mode. Each branch is identical and is derived from the continuous-time controller with an integrator expressed in (F.22). The equivalent discrete-time transfer function for the PI-controller is $H_{\text{voltage\_outer}}(z)$ is

$$H_{\text{voltage\_outer}}(z) = \frac{1}{10} + \frac{1}{12000} \cdot \frac{1}{z-1}. \tag{F.28}$$

Separating the integrator allows for the compensation of integral windup.

When the generators are in parallel mode, the top branch in Fig. F-24 is activated. The parallel top branch uses the same controller as the two separate branches with two differences. The first difference is that the input uses the error between the reference voltage and the average voltage of the two generators. The second difference is in the integrator. When switching to parallel mode, the integrator is initialized with the average of the two errors of the integrators in the bottom two branches for a smooth transition.

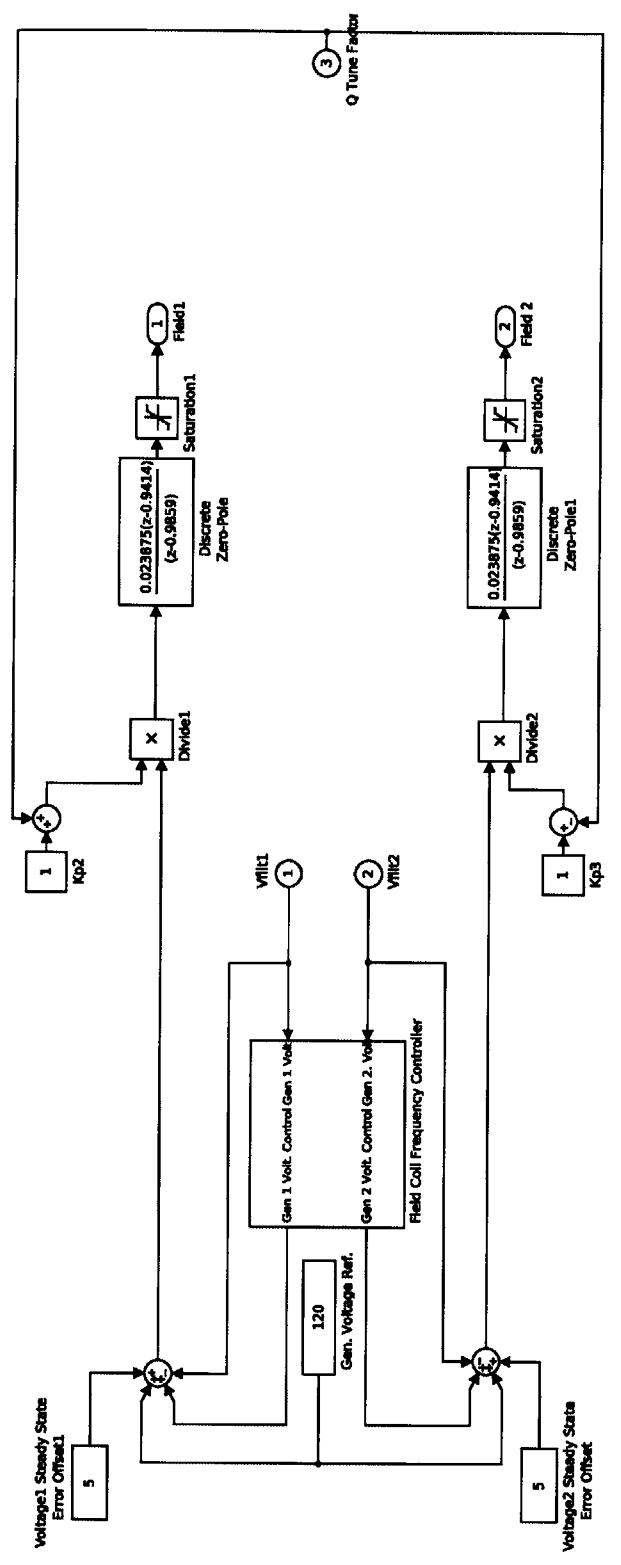


Fig. F-23: Inner block within the voltage controller.

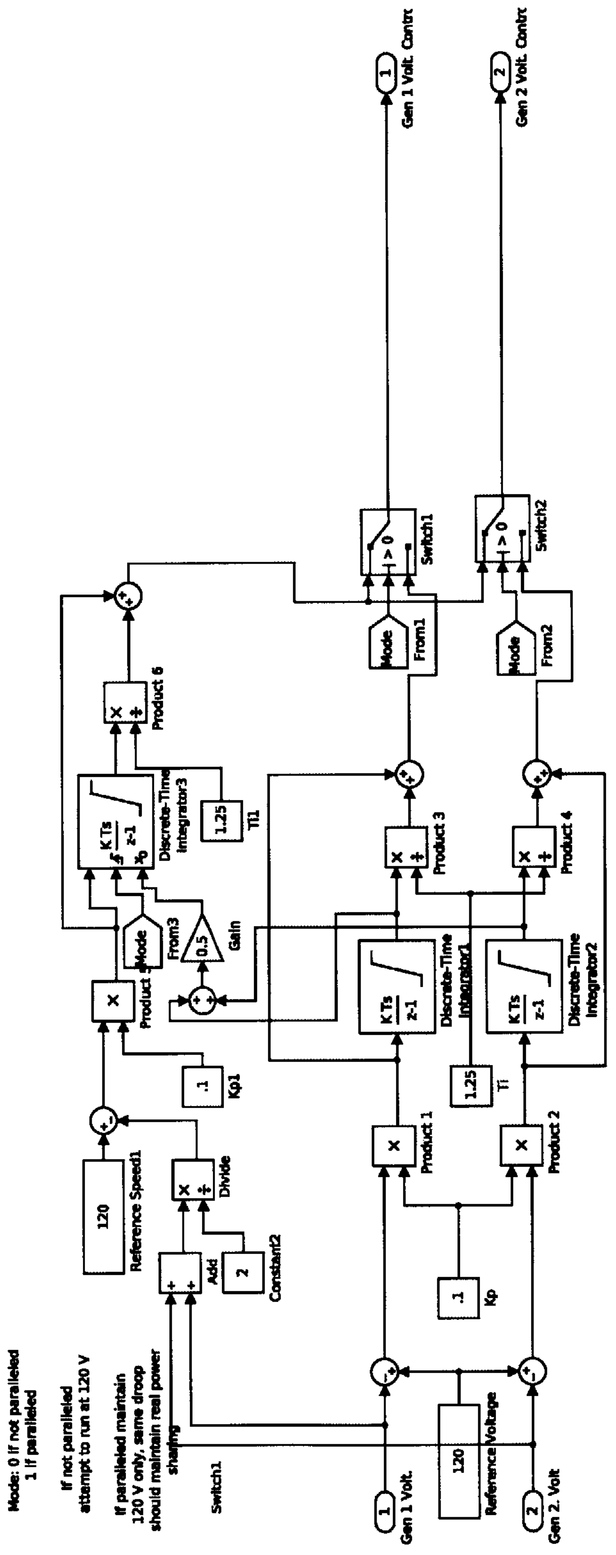


Fig. F-24: Voltage controller outer loop block.

Going back to Fig. F-23, a saturation block is added to limit the output within the capabilities of the power supplies connected to the field coil power supplies.

F.7 Overall Simulink Model

The overall Matlab Simulink model for the dual-generator system is shown in Fig. F-25. The S-Function block to the far left is the brains of the operation as described in §F.5. It takes in as inputs the two commanded voltages from the speed controller block that will be sent to programmable power supplies to adjust the DC prime movers. It also takes in the two commanded voltages from the field coil controller block that will be sent to the power supplies to adjust the field coil voltages on the generators. Furthermore, it takes in three user-provided state variables. The GUI allows the users to open or close a relay to connect each generator to its load. It allows the user to also run the generators individually or in parallel.

F.8 Generator Performance

As mentioned earlier, these two generators are being used to model the shipboard electrical distribution system of DDG 51 class ships. The generators used onboard DDG 51 class ships are required to meet specific design specifications. One specification is that the recovery time for the frequency may not exceed 1.5 seconds [96]. Recovery is defined as returning to within $\pm 1\%$ of the rated frequency or $\pm$ 0.6 Hz for a 60 Hz system. These specifications must be met when designing the controllers.

To test the performance of the generators, 450 Watts of light bulbs were placed across each phase. Figure F-26 shows several variables from the performance test.

The top plot shows the parallel state of the two generators. The generators run individually for the first 20 seconds before they are commanded to run in parallel. The parallel switch is opened at $t = 88.68$ seconds. The second plot shows the state of the load switch that connects the paralleled generators to the light bulbs. The switch is opened between $t = 40.93$ and $t = 68.75$. The final plot illustrates the difference in the voltage angles between the two generators.

Even when each generator runs individually, the simulink model looks to match the voltage angles determined in (F.13). A zoomed in version of the voltage angle difference is shown in Fig. F-27. The controller is able to match the angles at $t = 7$. When the parallel switch is closed at $t = 20.65$ seconds, the angles are already matched and the generators are paralleled immediately.

To analyze the performance of the generators, the frequency must recover quickly to step changes. According to (F.4), the DC prime movers must rotate at $\omega_p = 43.6364$ *rps* to ensure a 60 Hz line frequency. Figure F-28 shows the speed of the DC prime movers around the step changes. The top plot shows that the speed recovers in 0.7 seconds when the load is turned on. The bottom plot shows that the speed recovers in 0.3 seconds when the load is turned off. Both recovery times are less than 1.5 seconds as desired.

Another performance metric for the generators is the ability to share real and reactive powers equally calculated by (F.9) and (F.10). The controllers for speed and field coil voltage were designed such that the two equally sized generators would share power equally. The

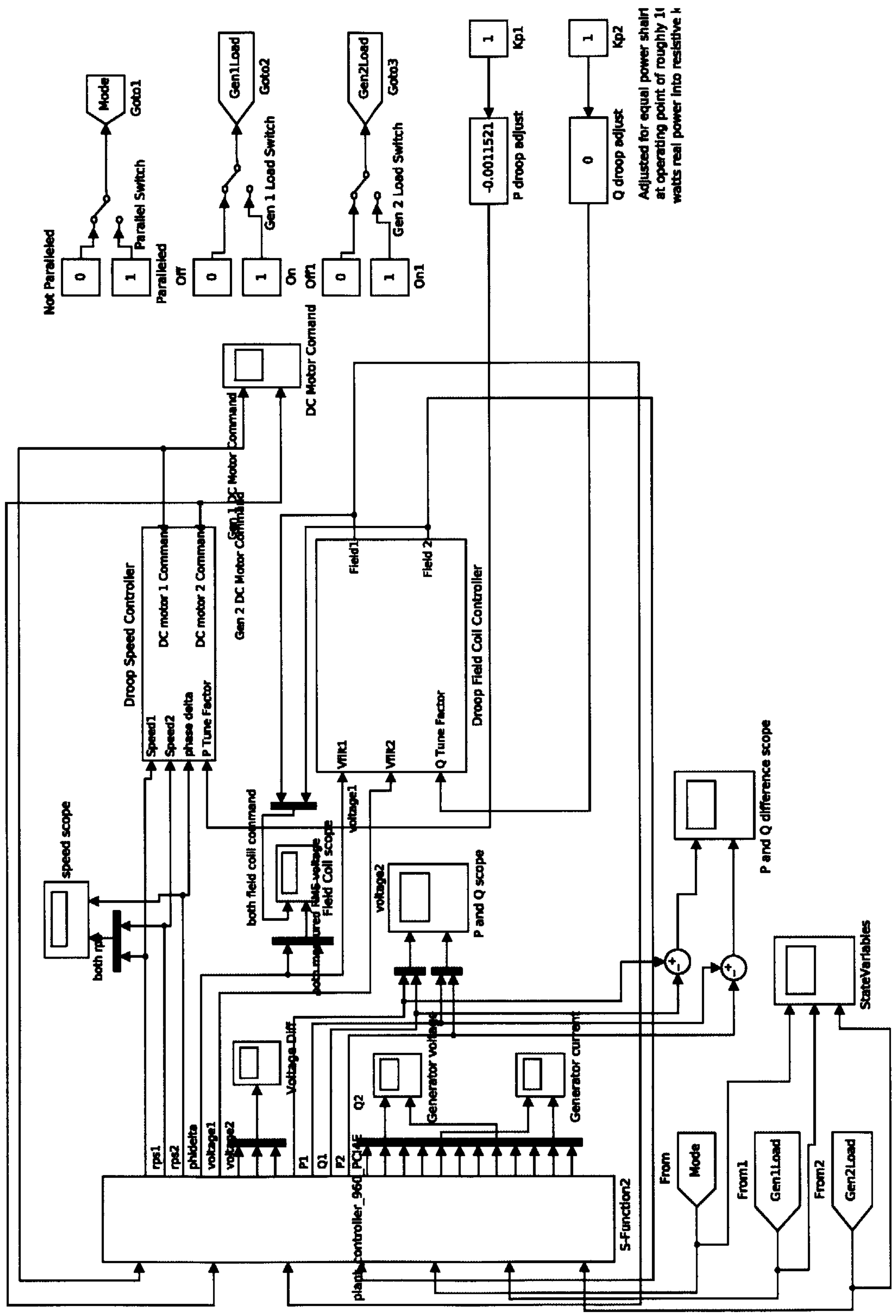


Fig. F-25: Overall Matlab Simulink model

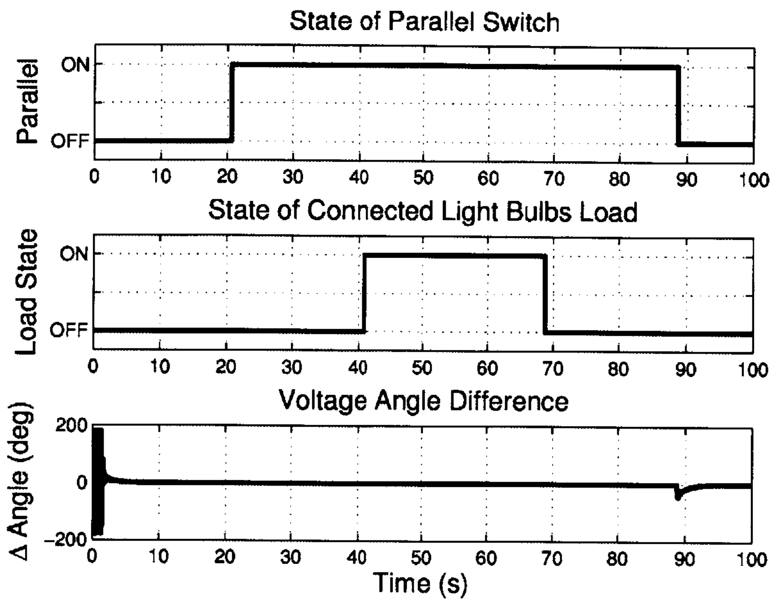


Fig. F-26: Top plot shows the state of the parallel switch. The middle plot is the state for the load switch. The bottom plot is the the angle difference between the voltages.

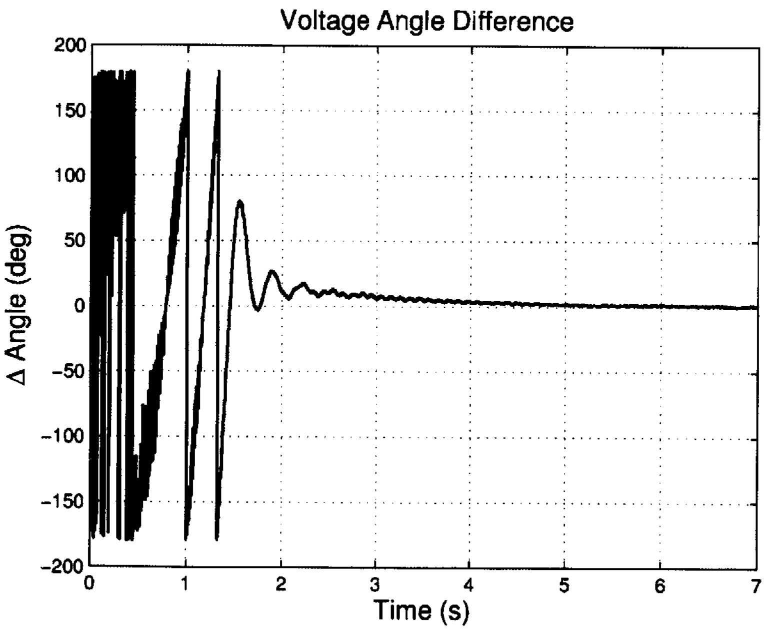


Fig. F-27: Angle difference between the voltages of the two generators.

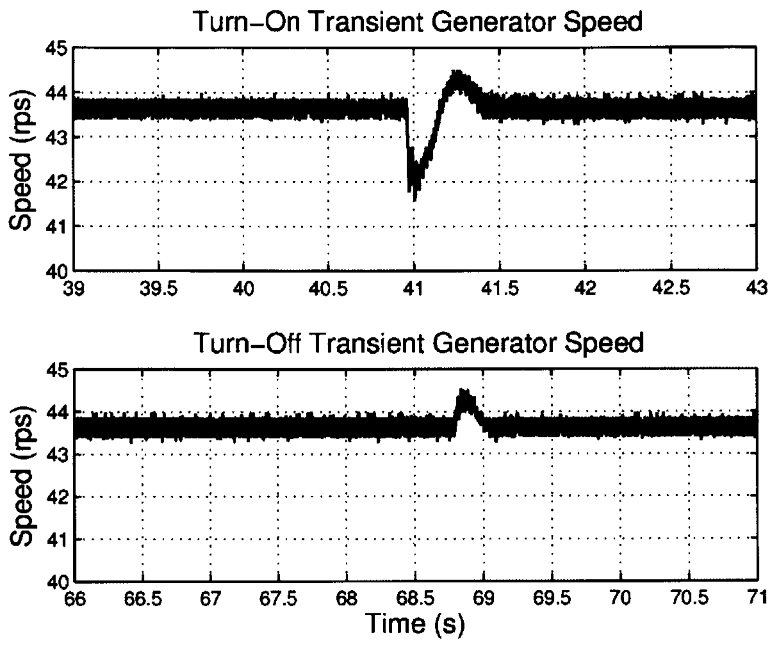


Fig. F-28: Speed of generators in response to a transient event.

real powers drawn by each generator are shown in Fig. F-29 and the reactive power draws are shown in Fig. F-30.

F.9 Conclusion

This chapter presented a detailed overview of the design and implementation of a scaled hardware model of a shipboard electrical distribution system. The components and the design of the underlying controllers used to provide 60 Hz, 120 Vrms operation were discussed. This scaled model provided a testbed to test zonal protection schemes used in the Arleigh Burke class DDG 51 Flight IIA destroyer. Chapter 7 will discuss the use of multifunction monitors (MFM) and incorporating nonintrusive load monitoring in a zonal electrical distribution system for enhanced protection.

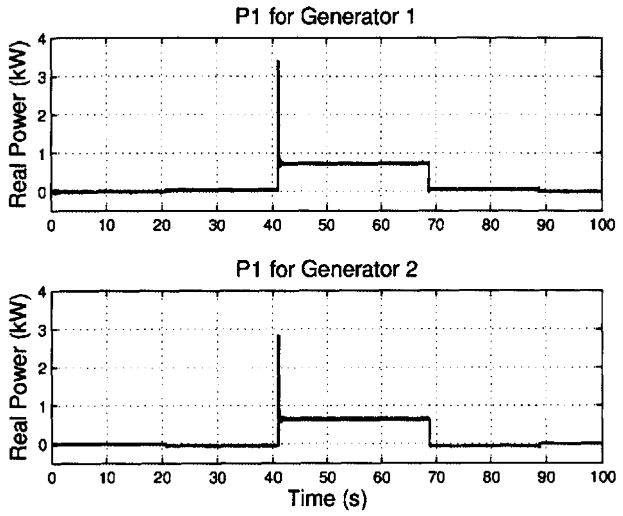


Fig. F-29: Real power provided by each generator

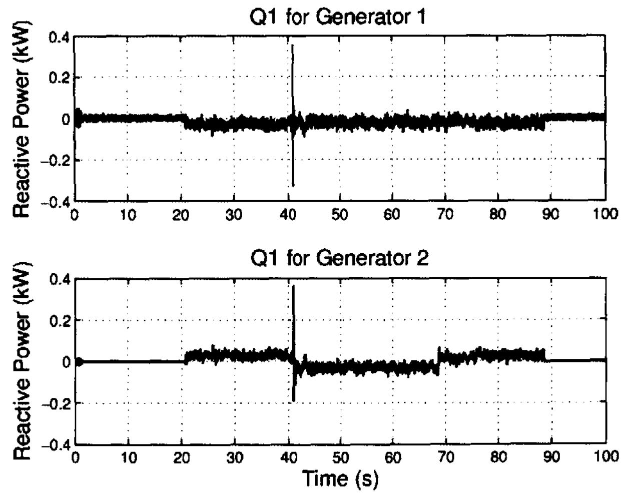


Fig. F-30: Reactive power provided by each generator

Appendix G

Dual Generators Simulink Code

G.1 sys_transfer_function_finder.m

Listing G.1: sys_transfer_function_finder.m

```
function [sys, w] = sys_transfer_function_finder(in, out, freq_rez,
    fs, input_cutoff_freq, doplot)
% [sys, w] = sys_transfer_function_finder(in, out, freq_rez, fs,
    input_cutoff_freq, doplot)
% This function takes an input/output pair according to the graphic
    below.
%           _ _ _
%          |     |
%  In --> | sys | --> Out
%          |_ _ _|
%
% It returns the frequency response of the system sys.
%
% Inputs:
%   in - electrical current data vector
%   out - waveform
%   freq_rez - desired frequency resolution
%   fs - sample frequency
%   input_cutoff_freq - stopband-edge frequency
%   doplot - boolean flag to plot frequency responses
%
%   Output:
%   sys - frequency response of system
%   w   - frequency vector

if nargin<6
    doplot = 1;
end

L = length(in);
N = length(out);
if (L ~= N)
    error('inputs not equal length');
```

```
end

window_size = round(fs/freq_rez);
overlap = round(window_size*0.5);

[b, a] = cheby2(9,30,(input_cutoff_freq*2)/(fs/2));

outf = filtfilt(b,a,out);
inf = filtfilt(b,a,in);
[OI, w] = cpsd(outf, inf,hanning(window_size),overlap,window_size,fs
    );
[II, w] = cpsd(inf, inf,hanning(window_size),overlap,window_size,fs)
    ;
sys = OI./II;

if (doplot)
    figure
    subplot(2,1,1)
    semilogx(w,20*log10(abs(sys)));
    xlim([w(1),input_cutoff_freq])
    subplot(2,1,2)
    semilogx(w,180/pi*unwrap(angle(sys)))
    xlim([w(1),input_cutoff_freq])
end
end
```

G.2 plant_controller_960_PCI4E.c

Listing G.2: plant_controller_960_PCI4E.c

```
/*
 * You must specify the S_FUNCTION_NAME as the name of your S-
    function
 */

#define S_FUNCTION_NAME  plant_controller_960_PCI4E
#define S_FUNCTION_LEVEL 2

#include <conio.h>
#include <stdlib.h>
#include <math.h>
#include <simstruc.h>
#include <rtwintgt.h>
#include "Plant_Controller_Definitions.h"
#include "PCI_1710_4E_Comm.c"
#include "vectormath.c"


/*====================*
 * S-function methods *
 *====================*/
```

```
/* Function: mdlInitializeSizes
   ================================================
 * Abstract:
 *    The sizes information is used by Simulink to determine the S-
   function
 *    block's characteristics (number of inputs, outputs, states,
   etc.).
 */
static void mdlInitializeSizes(SimStruct *S)
{
    uint_T k;
    /* See sfuntmpl_doc.c for more details on the macros below */
    ssSetNumSFcnParams(S, 0);  /* Number of expected parameters */
    if (ssGetNumSFcnParams(S) != ssGetSFcnParamsCount(S)) {
        /* Return if number of expected != number of actual
           parameters */
        return;
    }

    ssSetNumContStates(S, 0);
    ssSetNumDiscStates(S, 0);

     /* Table of Inputs
      *
      * Index |     Description      | Width
      * ------------------------------------
      * 0   |  DCMotor1Voltage   | 1
      * 1   |  DCMotor2Voltage   | 1
      * 2   | FieldCoil1Voltage | 1
      * 3   | FieldCoil2Voltage | 1
      * 4   | ParallelRelayCmd   | 1
      * 5   | Gen1LoadRelayCmd   | 1
      * 6   | Gen2LoadRelayCmd   | 1
      */

    if (!ssSetNumInputPorts(S, 7)) return;
    for(k=0;k<7;k++)
    {
      ssSetInputPortWidth(S, k, 1);
      ssSetInputPortDirectFeedThrough(S, k, 0);
      ssSetInputPortRequiredContiguous(S, k, 1);
    }

     /* Table of Outputs

        Index | Description | Width
      ------------------------------------
           0  |     rps1     |  1
           1  |     rps2     |  1
           2  |   phidelta   |  1
           3  |   voltage1   |  1
           4  |   voltage2   |  1
      */
    if (!ssSetNumOutputPorts(S, 24)) return;

    for(k=0;k<24;k++)
    {
        ssSetOutputPortWidth(S, k, 1);
    }

    ssSetNumSampleTimes(S, 1);

    // Point vector for memory-mapped PCI device capability
    ssSetNumPWork(S, 1);

    /* Table of DWork
     *
     * Index |          Description          | Width
     * ----------------------------------------------
     *   0   |       IOBaseAddress1          | 1
     *   1   |       IOBaseAddress2          | 1
     *   2   |      PCI4EBaseAddress         | 1
     *   3   |        MemoryLength           | 1
     *   4   |       Old TimeStamp           | 1
     *   5   |   Old Encoder 0 Count         | 1
     *   6   |   Old Encoder 1 Count         | 1
     *   7   |   Old Encoder 2 Count         | 1
     *   8   |   Old Encoder 3 Count         | 1
     *   9   |       Gen 1 Speeds            | 100
     *   10  |       Gen 2 Speeds            | 100
     *   11  |            V1A                | 20
     *   12  |            V1B                | 20
     *   13  |            V1C                | 20
     *   14  |            C1A                | 20
     *   15  |            C1B                | 20
     *   16  |            C1C                | 20
     *   17  |            V2A                | 20
     *   18  |            V2B                | 20
     *   19  |            V2C                | 20
     *   20  |            C2A                | 20
     *   21  |            C2B                | 20
     *   22  |            C2C                | 20
     *   23  |       Cosine vector           | 16
     *   24  |        Sine vector            | 16
     *   25  |   Parallel Relay State        | 1
     *   26  |   Gen1 Load Relay State       | 1
     *   27  |   Gen2 Load Relay State       | 1
     *   28  |    Old Relay Commands         | 1
     */

    ssSetNumDWork(S, 29);
    for(k=0; k<=8; k++)
    {
      ssSetDWorkWidth(S, k, 1);
      ssSetDWorkDataType(S, k, SS_INT32);
    }

    for(k=GEN_SPEEDS_LOC; k<GEN_SPEEDS_LOC+2; k++)
    {
      ssSetDWorkWidth(S, k, N_RPS);
      ssSetDWorkDataType(S, k, SS_DOUBLE);
    }

    for(k=VC_LOC; k<VC_LOC+12; k++)
    {
      ssSetDWorkWidth(S, k, N_MEAS);
      ssSetDWorkDataType(S, k, SS_DOUBLE);
    }

    for(k=COS_SIN_VEC_LOC; k<COS_SIN_VEC_LOC+2; k++)
    {
      ssSetDWorkWidth(S, k, N_TRIG);
      ssSetDWorkDataType(S, k, SS_DOUBLE);
    }

    for(k=PAR_RLY_LOC; k<PAR_RLY_LOC+3; k++)
    {
        ssSetDWorkWidth(S, k, 1);
        ssSetDWorkDataType(S, k, SS_INT8);
    }

    for(k=OLD_RELAY_CMDS_LOC; k<OLD_RELAY_CMDS_LOC+1; k++)
    {
        ssSetDWorkWidth(S, k, 1);
        ssSetDWorkDataType(S, k, SS_INT8);
    }

    /* Specify the sim state compliance to be same as a built-in
       block */
    ssSetSimStateCompliance(S, USE_DEFAULT_SIM_STATE);

    ssSetOptions(S, 0);
}



/* Function: mdlInitializeSampleTimes
    =========================================
 * Abstract:
 *    This function is used to specify the sample time(s) for your
 *    S-function. You must register the same number of sample times
    as
 *    specified in ssSetNumSampleTimes.
 */
static void mdlInitializeSampleTimes(SimStruct *S)
{
  ssSetSampleTime(S, 0, SAMPLE_TIME);  // 960 hz
  ssSetOffsetTime(S, 0, 0.0); // on offset
}

```

```
#define MDL_START  /* Change to #undef to remove function */
#if defined(MDL_START)
/* Function: mdlStart
   =======================================================
 * Abstract:
 *    This function is called once at start of model execution. If
    you
 *    have states that should be initialized once, this is the place
 *    to do it.
 */
static void mdlStart(SimStruct *S)
{
  uint32_T *DWorkPointer = (uint32_T*) ssGetDWork(S,0);
  uint32_T IOBaseAddress1, IOBaseAddress2, tempBaseAddress;
  uint32_T MemoryBaseAddress, MemoryLength;
  volatile uint32_T* BAR0Address;

  uint_T error_code, k, n;
  uint8_T ucRegVal, *tempPointer;
  real_T *tempvec;

  uint8_T ucGain, ucUniPolar, ucDifferential; //Base+2, channel
     range setting
  uint8_T ucTrigSrc, ucIrqEn, ucFh, ucCnt0;   //Base+6, Control
     register

#ifndef MATLAB_MEX_FILE

  error_code = find_pci_1710_IOBaseAddresses(&IOBaseAddress1,&
     IOBaseAddress2);
  get_pci_4E_Memory_Info(&MemoryBaseAddress, &MemoryLength);

  DWorkPointer[0] = IOBaseAddress1;
  DWorkPointer[1] = IOBaseAddress2;
  DWorkPointer[PCI_BAR_LOC] = MemoryBaseAddress;
  DWorkPointer[PCI_BAR_MEM_LEN_LOC] = MemoryLength;
  DWorkPointer[TS_LOC] = 0;             // Old TimeStamp;
  DWorkPointer[ENC_COUNTS_LOC+0] = 0; // Old Encoder 0 Count;
  DWorkPointer[ENC_COUNTS_LOC+1] = 0; // Old Encoder 1 Count;
  DWorkPointer[ENC_COUNTS_LOC+2] = 0; // Old Encoder 2 Count;
  DWorkPointer[ENC_COUNTS_LOC+3] = 0; // Old Encoder 3 Count;

  /* Setup Analog Input and Output Channels */
  for(n=0;n<2;n++)
  {
    tempBaseAddress = DWorkPointer[n];

    //Setup Output Channels
    ucRegVal = 0x00; //Internal ref., ref. voltage 5V for both
       Analog
                      //Output Channels
    outp(tempBaseAddress+14, ucRegVal);

    //Setup Input Channels
    outp(tempBaseAddress+6, 0);  //Clear Trigger source, interrupt.
    outp(tempBaseAddress+9, 0);   //Clear FIFO

    ucGain = 0 ;                 // -5V to 5V
    ucUniPolar = 0;              //Bipolar
    ucDifferential = 1;          //Differential
    ucGain = (ucGain) | (ucUniPolar<<4) | (ucDifferential<<5);

    for(k=0;k<6;k++)
    {
      // set these even channels as differential
      outp(tempBaseAddress+4, k*2);     //Start channel
      outp(tempBaseAddress+5, k*2);     //Stop channel
      outp(tempBaseAddress+2, ucGain); //Set each channel
    }

    ucDifferential = 0;          //Single-ended
    ucGain = (ucGain) | (ucUniPolar<<4) | (ucDifferential<<5);

    for(k=12;k<16;k++)
    {
      // set these channels as single-ended
      outp(tempBaseAddress+4, k);     //Start channel
      outp(tempBaseAddress+5, k);     //Stop channel
      outp(tempBaseAddress+2, ucGain); //Set each channel
    }

    outp(tempBaseAddress+4, 0);      //Start channel
    outp(tempBaseAddress+5, 10);     //Stop channel


    ucTrigSrc = 1;         //Software trigger
    ucIrqEn = 0;           //Not use interrupt;
    ucFh = 0;              //One data interrupt
    ucCnt0 = 0;            //Internal crystal
    ucTrigSrc = (ucTrigSrc) | (ucIrqEn<<4) | (ucFh<<5) | (ucCnt0<<6)
        ;
    outp(tempBaseAddress+6, ucTrigSrc);
  }

  //Initialize work vectors to 0
  for(k=GEN_SPEEDS_LOC; k<GEN_SPEEDS_LOC+2; k++)
  {
    tempvec = ssGetDWork(S,k);
    for(n=0; n<N_RPS; n++)
    {
        tempvec[n] = 0;
    }
  }

  for(k=VC_LOC; k<VC_LOC+12; k++)
  {
      tempvec = ssGetDWork(S,k);
      for(n=0; n<N_MEAS; n++)
```

```
        {
            tempvec[n] = 0;
        }
    }

    // create sampled Fourier basis functions
    for(k=0; k<N_TRIG; k++)
    {
      tempvec = ssGetDWork(S,COS_SIN_VEC_LOC);
      tempvec[k] = cos(2*PI*60.0*k/FS);

      tempvec = ssGetDWork(S,COS_SIN_VEC_LOC+1);
      tempvec[k] = sin(2*PI*60.0*k/FS);
    }

    for(k=PAR_RLY_LOC; k<PAR_RLY_LOC+3; k++)
    {
      tempvec = ssGetDWork(S,k);
      tempvec = 0;
    }

    for(k=OLD_RELAY_CMDS_LOC; k<OLD_RELAY_CMDS_LOC+1; k++)
    {
      tempvec = ssGetDWork(S,k);
      tempvec = 0;
    }

     /* map device memory and store the mapped address to PWork */
    ssSetPWorkValue(S, 0, RTBIO_MapDeviceMemory(MemoryBaseAddress,
        MemoryLength));
    BAR0Address = ssGetPWorkValue(S, 0);
    initialize_PCI4E(BAR0Address);


    // set Relays pins HI (inverted) to open mechanical relays
    // and set bit 3 to HI (indication that model is running)
    outp(IOBaseAddress1+16, 0x0F);
#else
    /* we are in Simulink, don't do board I/O */
#endif

}
#endif /*  MDL_START */


/* Function: mdlOutputs
    =====================================================
 * Abstract:
 *     In this function, you compute the outputs of your S-function
 *     block.
 */
static void mdlOutputs(SimStruct *S, int_T tid)
{
    real_T *rps1 = ssGetOutputPortSignal(S,0);
    real_T *rps2 = ssGetOutputPortSignal(S,1);
    real_T *phidelta = ssGetOutputPortSignal(S,2);
    real_T  *voltage1 = ssGetOutputPortSignal(S,3);
    real_T  *voltage2 = ssGetOutputPortSignal(S,4);

    real_T  *y5 = ssGetOutputPortSignal(S,5);
    real_T  *y6 = ssGetOutputPortSignal(S,6);
    real_T  *y7 = ssGetOutputPortSignal(S,7);
    real_T  *y8 = ssGetOutputPortSignal(S,8);
    real_T  *y9 = ssGetOutputPortSignal(S,9);
    real_T  *y10 = ssGetOutputPortSignal(S,10);
    real_T  *y11 = ssGetOutputPortSignal(S,11);
    real_T  *y12 = ssGetOutputPortSignal(S,12);
    real_T  *y13 = ssGetOutputPortSignal(S,13);
    real_T  *y14 = ssGetOutputPortSignal(S,14);
    real_T  *y15 = ssGetOutputPortSignal(S,15);
    real_T  *y16 = ssGetOutputPortSignal(S,16);
    real_T  *y17 = ssGetOutputPortSignal(S,17);
    real_T  *y18 = ssGetOutputPortSignal(S,18);
    real_T  *y19 = ssGetOutputPortSignal(S,19);
    real_T  *y20 = ssGetOutputPortSignal(S,20);
    real_T  *y21 = ssGetOutputPortSignal(S,21);
    real_T  *y22 = ssGetOutputPortSignal(S,22);
    real_T  *y23 = ssGetOutputPortSignal(S,23);

#ifndef MATLAB_MEX_FILE
   uint32_T *DWorkPointer = (uint32_T*) ssGetDWork(S,0);
   volatile uint32_T* BAR0Address = ssGetPWorkValue(S, 0);
   uint32_T IOBaseAddress1, IOBaseAddress2, MemoryBaseAddress;
   uint32_T tempBaseAddress;

   uint8_T k, n, ucStatus;
   uint16_T wRawData;

   uint8_T  j;
   uint16_T temp, debug[24], debug_count=0;

   real_T VCvec[2][6];
   real_T *tempvec, *tempvec2;
   real_T a1[12], b1[12];
   real_T *cos_vec, *sin_vec;
   real_T temp_rms_total[12], phis[6];
   real_T P[6], Q[6];

   IOBaseAddress1 = DWorkPointer[0];
   IOBaseAddress2 = DWorkPointer[1];
   MemoryBaseAddress = DWorkPointer[PCI_BAR_LOC];
   cos_vec = (real_T*) ssGetDWork(S,COS_SIN_VEC_LOC);
   sin_vec = (real_T*) ssGetDWork(S,COS_SIN_VEC_LOC+1);

   /* Get Encoder Speeds from PCI4E */
   getEncoderSpeeds(BAR0Address, DWorkPointer, rps1, rps2);
   *phidelta = 0;
```

```
// Read in differential generator currents and voltages
for(k=0;k<6;k++)
{
    for(n=0;n<2;n++)
    {
        tempBaseAddress = DWorkPointer[n];

        outp(tempBaseAddress, 0);               //Software trigger
        //Wait conversion complete
        do
        {
            ucStatus = inp(tempBaseAddress+7) & 1;
        }while(ucStatus == 1);

        if (ucStatus == 0)
        {
            wRawData = inpw(tempBaseAddress);
            wRawData = wRawData & 0xfff;
            VCvec[n][k] = ((5.0-(-5.0)) * (wRawData&0xfff)/0xfff)
               + (-5.0);
        }
    }
}


// Software fix to invert the voltage analog input channels
//  for the second generator
for(k=0;k<3;k++)
{
    VCvec[1][k] = -VCvec[1][k];
}

// Scale voltages and currents
for(n=0;n<2;n++)
{
    VCvec[n][0] = (VCvec[n][0] - 0.0297583)*84.461;
    VCvec[n][1] = (VCvec[n][1] - 0.109715)*84.186;
    VCvec[n][2] = (VCvec[n][2] - 0.011871)*83.981;
    VCvec[n][3] = (VCvec[n][3] + 0.006839)*10.109;
    VCvec[n][4] = (VCvec[n][4] + 0.019846)*10.044;
    VCvec[n][5] = (VCvec[n][5] + 0.007979)*10.098;
}

*y5 = VCvec[0][0]-VCvec[1][0];
*y6 = VCvec[0][1]-VCvec[1][1];
*y7 = VCvec[0][2]-VCvec[1][2];

// Automatic Voltage Regulator
// Update speeds vector and VCmatrix
tempvec = (real_T*) ssGetDWork(S,GEN_SPEEDS_LOC);
updateVector(tempvec,*rps1,N_RPS);

tempvec = (real_T*) ssGetDWork(S,GEN_SPEEDS_LOC+1);
updateVector(tempvec,*rps2,N_RPS);
```

```
for(n=0;n<2;n++)
{
     for(k=0;k<6;k++)
     {
          tempvec = (real_T*) ssGetDWork(S,VC_LOC+6*n+k);
          updateVector(tempvec,VCvec[n][k],N_MEAS);
     }
}

  // Compute the P and Q values for each phase
  // phase mapping : 1A, 1B, 1C, 2A, 2B, 2C

for(n=0;n<2;n++)
{
     for(k=0;k<3;k++)
     {
          tempvec = (real_T*) ssGetDWork(S,VC_LOC+6*n+k); // voltage
          tempvec2 = (real_T*) ssGetDWork(S,VC_LOC+3+6*n+k); //
              current
          P[(n*3)+k] = (1/16.0)*dotProduct(&tempvec[4],&tempvec2[4],
              N_TRIG);
          Q[(n*3)+k] = (1/16.0)*dotProduct(&tempvec[4],&tempvec2[0],
              N_TRIG);
      }
 }


for(n=0;n<2;n++)
{
     // use dot product to compute the a1 and b1 Fourier
         coefficients
     // each will be a row vector of length 12
     // a1, b1 mapping V1A, V1B, V1C, C1A, C1B, C1C, ...
     //                V2A, V2B, V2C, C2A, C2B, C2C
     for(k=0;k<6;k++)
     {
          tempvec = (real_T*) ssGetDWork(S,VC_LOC+6*n+k);
          a1[6*n+k] = (sqrt(2)/16.0)*dotProduct(cos_vec,tempvec,
              N_TRIG);
          b1[6*n+k] = (sqrt(2)/16.0)*dotProduct(sin_vec,tempvec,
              N_TRIG);
     }
}



// Compute phase angles of voltages
// phis will be a row vector of length 6

for(n=0;n<2;n++)
{
     for(k=0;k<3;k++)
```

```
        {
              phis[3*n+k] = atan2(a1[6*n+k],b1[6*n+k])*180/PI;
        }
    }


    // Compute the rpsdelta for phase locking
    *phidelta = (phis[1]-phis[4]);
    if (*phidelta < -180)
      *phidelta += 360;
    else if (*phidelta > 180)
      *phidelta -= 360;

    // Compute the sensing parameter for the Automatic Voltage
       Regulator
    // using only the B phase

    // RMS total of current and voltages
    for(k=0;k<12;k++)
    {
        temp_rms_total[k] = sqrt(pow(a1[k],2) + pow(b1[k],2));
    }

    *voltage1 = temp_rms_total[1];
    *voltage2 = temp_rms_total[7];


    *y8 = P[0]+P[1]+P[2];
    *y9 = Q[0]+Q[1]+Q[2];
    *y10 = P[3]+P[4]+P[5];
    *y11 = Q[3]+Q[4]+Q[5];

    *y12 = VCvec[0][0]; // V1A
    *y13 = VCvec[0][1]; // V1B
    *y14 = VCvec[0][2]; // V1C
    *y15 = VCvec[0][3]; // C1A
    *y16 = VCvec[0][4]; // C1B
    *y17 = VCvec[0][5]; // C1C
    *y18 = VCvec[1][0]; // V2A
    *y19 = VCvec[1][1]; // V2B
    *y20 = VCvec[1][2]; // V2C
    *y21 = VCvec[1][3]; // C2A
    *y22 = VCvec[1][4]; // C2B
    *y23 = VCvec[1][5]; // C2C

  #else

  #endif

  }



  #define MDL_UPDATE  /* Change to #undef to remove function */
```

```
#if defined(MDL_UPDATE)
/* Function: mdlUpdate
   ======================================================
 * Abstract:
 *    This function is called once for every major integration time
    step.
 *    Discrete states are typically updated here, but this function
    is useful
 *    for performing any tasks that should only take place once per
 *    integration step.
 */
static void mdlUpdate(SimStruct *S, int_T tid)
{
    const real_T *DCMotor1Voltage =  ssGetInputPortRealSignal(S,0);
    const real_T *DCMotor2Voltage =  ssGetInputPortRealSignal(S,1);
    const real_T *FieldCoil1Voltage = ssGetInputPortRealSignal(S,2);
    const real_T *FieldCoil2Voltage = ssGetInputPortRealSignal(S,3);
    const real_T *ParallelRelayCmd  = ssGetInputPortRealSignal(S,4);
    const real_T *Gen1LoadRelayCmd = ssGetInputPortRealSignal(S,5);
    const real_T *Gen2LoadRelayCmd = ssGetInputPortRealSignal(S,6);

    uint32_T *DWorkPointer = (uint32_T*) ssGetDWork(S,0);
    uint32_T IOBaseAddress1, IOBaseAddress2;

    //uint8_T relay_cmds=0, old_relay_cmds;
    uint8_T relay_cmds= 0x07, old_relay_cmds;
    uint16_T wRawData;

    IOBaseAddress1 = DWorkPointer[0];
    IOBaseAddress2 = DWorkPointer[1];

    old_relay_cmds = DWorkPointer[OLD_RELAY_CMDS_LOC];

    #ifndef MATLAB_MEX_FILE

    // Write voltage out to DC motor power supplies
    wRawData = (DCMotor1Voltage[0]/(5.0 - 0.0)) * 0xfff;
    outpw(IOBaseAddress1+10, wRawData);
    wRawData = (DCMotor2Voltage[0]/(5.0 - 0.0)) * 0xfff;
    outpw(IOBaseAddress2+10, wRawData);

    // Write voltage out to Field Coil power supplies
    wRawData = (FieldCoil1Voltage[0]/(5.0 - 0.0)) * 0xfff;
    outpw(IOBaseAddress1+12, wRawData);
    wRawData = (FieldCoil2Voltage[0]/(5.0 - 0.0)) * 0xfff;
    outpw(IOBaseAddress2+12, wRawData);


    // Set Relays
    // relay_cmd = 0x07;
    relay_cmds = 0x0F;

    if(Gen2LoadRelayCmd[0]>0.5)
    {
            // relay_cmds = relay_cmds | 0x04;
            relay_cmds = relay_cmds & 0xFB;
        }
        if (ParallelRelayCmd[0]>0.5)
        {
            // relay_cmds = relay_cmds | 0x02;
            relay_cmds = relay_cmds & 0xFD;
        }
        if (Gen1LoadRelayCmd[0]>0.5)
        {
            // relay_cmds = relay_cmds | 0x01;
            relay_cmds = relay_cmds & 0xFE;
        }

        if (old_relay_cmds != relay_cmds)
        {
            outp(IOBaseAddress1+16, relay_cmds);
            DWorkPointer[OLD_RELAY_CMDS_LOC] = relay_cmds;
        }

        #else

#endif

}
#endif /* MDL_UPDATE */



#undef MDL_DERIVATIVES  /* Change to #undef to remove function */
#if defined(MDL_DERIVATIVES)
  /* Function: mdlDerivatives
       =================================================
   * Abstract:
   *    In this function, you compute the S-function block's
     derivatives.
   *    The derivatives are placed in the derivative vector, ssGetdX
     (S).
   */
  static void mdlDerivatives(SimStruct *S)
  {
  }
#endif /* MDL_DERIVATIVES */



/* Function: mdlTerminate
    =====================================================
 * Abstract:
 *    In this function, you should perform any actions that are
    necessary
 *    at the termination of a simulation.  For example, if memory
    was
 *    allocated in mdlStart, this is the place to free it.
 */
static void mdlTerminate(SimStruct *S)
{
    #ifndef MATLAB_MEX_FILE
        void* devaddr = ssGetPWorkValue(S, 0);
        uint32_T *DWorkPointer = (uint32_T*) ssGetDWork(S,0);
        uint32_T IOBaseAddress1, IOBaseAddress2;
        uint16_T wRawData;

        /* unmap the device address */
        if (devaddr!=0)
            RTBIO_UnmapDeviceMemory(devaddr, DWorkPointer[
               PCI_BAR_MEM_LEN_LOC]);

        IOBaseAddress1 = DWorkPointer[0];
        IOBaseAddress2 = DWorkPointer[1];

        /* Write voltage out to Field Coil power supplies */
        wRawData = 0.0 * 0xfff;
        outpw(IOBaseAddress1+12, wRawData);
        wRawData = 0.0 * 0xfff;
        outpw(IOBaseAddress2+12, wRawData);

        /* Write voltage out to DC motor power supplies */
        wRawData = 0.0 * 0xfff;
        outpw(IOBaseAddress1+10, wRawData);
        wRawData = 0.0 * 0xfff;
        outpw(IOBaseAddress2+10, wRawData);

        /* Set Relays */
        outp(IOBaseAddress1+16, 0x07);

    #endif /* MDL_TERMINATE */
}



/*=====================================================*
 * See sfuntmpl_doc.c for the optional S-function methods *
 *=====================================================*/

/*=============================*
 * Required S-function trailer *
 *=============================*/

#ifdef  MATLAB_MEX_FILE    /* Is this file being compiled as a MEX-
   file? */
#include "simulink.c"      /* MEX-file interface mechanism */
#else
#include "cg_sfun.h"       /* Code generation registration function
   */
#endif
```

G.3 Plant_Controller_Definitions.h

Listing G.3: Plant_Controller_Definitions.h

```
#ifndef __PCI4E_SPEED_FUNCTION_DEF_INCLUDED
#define __PCI4E_SPEED_FUNCTION_DEF_INCLUDED

/* Location within the DWorkPointer of various variables */
#define PCI_BAR_LOC 2
#define PCI_BAR_MEM_LEN_LOC 3
#define TS_LOC 4
#define ENC_COUNTS_LOC 5
#define GEN_SPEEDS_LOC 9
#define VC_LOC 11
#define COS_SIN_VEC_LOC 23
#define PAR_RLY_LOC 25
#define GEN_ONE_RLY_LOC 26
#define GEN_TWO_RLY_LOC 27
#define OLD_RELAY_CMDS_LOC 28

/* Constants used for size of DWorkPointers vectors */
#define N_RPS 100
#define N_MEAS 20
#define N_TRIG 16

/* Sample period and time */
#define FS  960.0
#define SAMPLE_TIME 0.0010416
#define PI 3.14159265359

/* PCI4E Functions Constants */
#define PRESET_OFFSET 0
#define OUTPUT_OFFSET 1
#define CONTROL_OFFSET 3
#define STATUS_OFFSET 4
#define TRANSFER_OFFSET 6
#define COMMAND_REG 7
#define TSLATCH_REG 15


#define COUNTS_PER_REV 1800.0
#define X1 1
#define X2 2
#define X4 4
#define FORWARD_DIR 1
#define MAX_COUNTS_DIFFERENCE 16383 //2^14-1

#define MULTIPLIER X4

#endif // __PCI4E_SPEED_FUNCTION_DEF_INCLUDED
```

G.4 PCI_1710_4E_Comm.c

Listing G.4: PCI_1710_4E_Comm.c

```
/*
    C Code to find the two PCI-1710 card's IOBaseAddress and
    to connect to the PCI4E counter card.

    Some code form PCI.c dos example code included with PCI-1710 CD
        .

    Written by Chris Schantz and Uzoma Orji
*/

#include "PCI4E_Speed_Function_Definitions.h"

#ifndef __PCI_1710_4E_COMM_INCLUDED
#define __PCI_1710_4E_COMM_INCLUDED

/* Functions */

int find_pci_1710_IOBaseAddresses(uint32_T *IOBaseAddress1, uint32_T
    *IOBaseAddress2)
{
    uint8_T Board_ID;
    uint_T bus, device, found = 0;
    uint32_T p_in, p_out, tempIOBaseAddress;

    for(bus = 0; bus<255; bus++)
    {
        for(device = 0; device<32; device++)
        {
            p_in = 0x80000000+bus*0x10000+(device*8)*0x100;
            outpd(0xcf8,p_in);
            p_out = inpd(0xcfc);
            if(p_out == 0x171013FE) /* Device and Vendor ID for
                PCI_1710 */
            {
                outpd(0xcf8, p_in+0x18);
                p_out = inpd(0xcfc);
                tempIOBaseAddress = p_out&0xfffffffc;
                Board_ID = inp(tempIOBaseAddress+20);
                if(Board_ID==00)
                {
                    *IOBaseAddress1 = tempIOBaseAddress;
                }
                else
                {
                    *IOBaseAddress2 = tempIOBaseAddress;
                }
                found++;
            }
            if(found==2)
```

```
                break;
            }
            if(found==2)
                break;
        }
        return found;
    }


    int find_pci_1710_IOBaseAddress(uint32_T *IOBaseAddress)
    {
        uint_T bus, device, found = 0;
        uint32_T p_in, p_out;

        for(bus = 0; bus<255; bus++)
        {
            for(device = 0; device<32; device++)
            {
                p_in = 0x80000000+bus*0x10000+(device*8)*0x100;
                outpd(0xcf8,p_in);
                p_out = inpd(0xcfc);
                if(p_out == 0x171013FE) /* Device and Vendor ID for
                    PCI_1710 */
                {
                    outpd(0xcf8, p_in+0x18);
                    p_out = inpd(0xcfc);
                    *IOBaseAddress = p_out&0xfffffffc;
                    found++;
                    break;
                }
                if(found)
                    break;
            }
            if(found)
                break;
        }
        return found;
    }


    int get_pci_4E_Memory_Info(uint32_T *BaseAddress, uint32_T *
        MemoryLength)
    {
        uint_T bus, device, found = 0;
        uint32_T p_in, p_out, ConfigSpaceAddress, TempBaseAddress;

        for(bus = 0; bus<255; bus++)
        {
            for(device = 0; device<32; device++)
            {
                p_in = 0x80000000+bus*0x10000+(device*8)*0x100;
                outpd(0xcf8,p_in);
                p_out = inpd(0xcfc);
                if(p_out == 0x57471892) /* Device and Vendor ID for
                   PCI4E */
                {
                    ConfigSpaceAddress = p_in;
                    found++;
                    break;
                }
                if(found)
                    break;
            }
            if(found)
                break;
        }

        if(found) {
            // get Address and Length of BAR0
         outpd(0xcf8, ConfigSpaceAddress + 0x10); // unmasked location
            of BAR0
            TempBaseAddress = inpd(0xcfc);
            outpd(0xcfc, 0xFFFFFFFF);
            p_out = inpd(0xcfc) & 0xFFFFFFF0;
            *MemoryLength = ~p_out + 1;                  //side of
               memory
            outpd(0xcfc,TempBaseAddress);
         *BaseAddress = TempBaseAddress & 0xFFFFFFF0; //masked location
            of BAR0
        }

        return found;
}

void PCI4E_SetRegisterValue(volatile uint32_T *BaseAddress, uint8_T
    Encoder, uint8_T RegisterOffset, uint32_T val)
{
        BaseAddress[Encoder*8+RegisterOffset] = val;
}

void PCI4E_SetMiscRegisterValue(volatile uint32_T *BaseAddress,
    uint8_T Register, uint32_T val)
{
        BaseAddress[Register] = val;
}

void PCI4E_GetRegisterValue(volatile uint32_T *BaseAddress, uint8_T
    Encoder, uint8_T RegisterOffset, uint32_T *val)
{
        *val = (uint32_T) BaseAddress[Encoder*8+RegisterOffset];
}

void PCI4E_GetMiscRegisterValue(volatile uint32_T *BaseAddress,
    uint8_T Register, uint32_T *val)
{
        *val = BaseAddress[Register];
}
```

```
void PCI4E_GetCountValue(volatile uint32_T *BaseAddress, uint8_T
    Encoder, uint32_T *val)
{
        PCI4E_SetRegisterValue(BaseAddress, Encoder, OUTPUT_OFFSET,
            0);
        PCI4E_GetRegisterValue(BaseAddress, Encoder, OUTPUT_OFFSET,
            val);
}


void PCI4E_SetMultiplier(volatile uint32_T *BaseAddress, uint8_T
    Encoder, uint8_T val)
{
    uint32_T ctrl_val;

    PCI4E_GetRegisterValue(BaseAddress, Encoder, CONTROL_OFFSET, &
        ctrl_val);  // Get current Control Register

    switch( val )
    {
        case 0:
            ctrl_val = ctrl_val | 0x0000;
            break;
        case X1:
            ctrl_val = ctrl_val | 0x4000;
            break;
        case X2:
            ctrl_val = ctrl_val | 0x8000;
            break;
        case X4:
            ctrl_val = ctrl_val | 0xC000;
            break;
    }

    PCI4E_SetRegisterValue(BaseAddress, Encoder, CONTROL_OFFSET,
        ctrl_val);   // Set the Control Register
}


void PCI4E_SetCounterMode(volatile uint32_T *BaseAddress, uint8_T
    Encoder, uint8_T Mode)
{
    // CounterMode: Governs counter behavior and limits:
        //      0 = acc. acts like a 24 bit counter
        //      1 = acc. uses preset register in range-limit mode
        //      2 = acc. uses preset register in non-recycle mode
        //      3 = acc. uses preset register in modulo-N mode.

        uint32_T ctrl_val;

    PCI4E_GetRegisterValue(BaseAddress, Encoder, CONTROL_OFFSET, &
        ctrl_val);   // Get the current control mode
    ctrl_val = ctrl_val & 0xFCFFFF;                           // Disabled
       bit 16 and 17.

    switch ( Mode )
    {
        case 0:
            // Do nothing...                  // Both bits 16 and 17
               should be disabled.
            break;
        case 1:
            ctrl_val = ctrl_val | 0x10000;                // Enable bit
                16.
            break;
        case 2:
            ctrl_val = ctrl_val | 0x20000;                // Enable bit
                17.
            break;
        case 3:
            ctrl_val = ctrl_val | 0x30000;                // Enable
               bits 16 and 17.
            break;
    }

    PCI4E_SetRegisterValue(BaseAddress, Encoder, CONTROL_OFFSET,
       ctrl_val);   // Set the Control Register
}


void PCI4E_SetCount(volatile uint32_T *BaseAddress, uint8_T Encoder,
     uint32_T val)
{
    uint32_T preset_val;

    PCI4E_GetRegisterValue(BaseAddress, Encoder, PRESET_OFFSET, &
       preset_val); // Get preset value

    // Write new position value to preset register...
    PCI4E_SetRegisterValue(BaseAddress, Encoder, PRESET_OFFSET, val
       );

    // Write value to transfer preset register...
    PCI4E_SetRegisterValue(BaseAddress, Encoder, TRANSFER_OFFSET,
       0);

    // Write old preset value back to preset register...
    PCI4E_SetRegisterValue(BaseAddress, Encoder, PRESET_OFFSET,
       preset_val);
}

void PCI4E_SetEnableAccumulator(volatile uint32_T *BaseAddress,
    uint8_T Encoder, boolean_T val)
{
    // Enabled: Master enable for accummulator (must be set to
       count).
```

```
        uint32_T ctrl_val;

        // Get the current control value...
        PCI4E_GetRegisterValue(BaseAddress, Encoder, CONTROL_OFFSET, &
            ctrl_val);

        if( val == 0 && (ctrl_val && 0x40000) ) // new value is False
            and old is True
        {
              ctrl_val = ctrl_val & 0xFBFFFF;    // Disabled bit 18.
              PCI4E_SetRegisterValue(BaseAddress, Encoder,
                  CONTROL_OFFSET, ctrl_val);
        }
        else                        // new value is True and old is False
        {
              ctrl_val = ctrl_val | 0x40000;     // Enable bit 18.
              PCI4E_SetRegisterValue(BaseAddress, Encoder,
                  CONTROL_OFFSET, ctrl_val);
        }
    }

    void PCI4E_SetForward(volatile uint32_T *BaseAddress, uint8_T
        Encoder, boolean_T val)
    {
        // Forward: swap a and b quadrature Foward = True, Reverse =
            False.
        uint32_T ctrl_val = 0;

        // Get the current control mode...
        PCI4E_GetRegisterValue(BaseAddress, Encoder, CONTROL_OFFSET, &
            ctrl_val);

        if( val == 0 && (ctrl_val && 0x80000) ) // new value is False
            and old is True
        {
              ctrl_val = ctrl_val & 0xF7FFFF;    // Disabled bit 19.
              PCI4E_SetRegisterValue(BaseAddress, Encoder,
                  CONTROL_OFFSET, ctrl_val);
        }
        else                        // new value is True and old is False
        {
              ctrl_val = ctrl_val | 0x80000;     // Enable bit 19.
              PCI4E_SetRegisterValue(BaseAddress, Encoder,
                  CONTROL_OFFSET, ctrl_val);
        }
    }

    void PCI4E_GetTimeStamp(volatile uint32_T *BaseAddress, uint32_T *
        timestamp)
    {
        uint32_T cmd_val = 0;

        // To get the TimeStamp counter to latch to the TimeStamp
            Output Latch
        // a transition bit 5 of the CMD_REGISTER from 0 -> 1 must
           occur.

        // Read command register so that it can be preserved.
        PCI4E_GetMiscRegisterValue(BaseAddress, COMMAND_REG, &cmd_val);

        PCI4E_SetMiscRegisterValue(BaseAddress, COMMAND_REG, (0
           xFFFFFFDF & cmd_val));

        PCI4E_SetMiscRegisterValue(BaseAddress, COMMAND_REG, (0x20 |
           cmd_val));

        PCI4E_GetMiscRegisterValue(BaseAddress, TSLATCH_REG, timestamp)
           ;
    }

    void PCI4E_CaptureTimeAndCounts(volatile uint32_T *BaseAddress,
       uint32_T *countvalues, uint32_T *timestamp)
    {
        uint8_T Encoder;
        uint32_T cmd_val = 0;

        // To get the TimeStamp counter to latch to the TimeStamp
           Output Latch
        // and trigger a capture transition bit 4 of the CMD_REGISTER
           from 0 -> 1.

        // Read command register so that it can be preserved.
        PCI4E_GetMiscRegisterValue(BaseAddress, COMMAND_REG, &cmd_val);

        PCI4E_SetMiscRegisterValue(BaseAddress, COMMAND_REG, (0
           xFFFFFFFF & cmd_val));

        PCI4E_SetMiscRegisterValue(BaseAddress, COMMAND_REG, (0x10 |
           cmd_val));

        PCI4E_GetTimeStamp(BaseAddress, timestamp);

        for(Encoder=0; Encoder < 4; Encoder++)
        {
            PCI4E_GetCountValue(BaseAddress, Encoder, countvalues+
               Encoder);
        }
    }

    void PCI4E_ResetTimeStamp(volatile uint32_T *BaseAddress)
    {
        uint32_T cmd_val = 0;

        // To reset the TimeStamp counter stop the counter by setting
           bit 6 of the CMD_REGISTER
        // to 1 and then restart the counter by setting bit 6 back to
           0.
        // Note: while bit 6 is 1 the TimeStamp counter value is zero.
        // When the TimeStamp is started, we also set bit 5 to 1.

        // Read command register so that it can be preserved.
        PCI4E_GetMiscRegisterValue(BaseAddress, COMMAND_REG, &cmd_val);

        PCI4E_SetMiscRegisterValue(BaseAddress, COMMAND_REG, (0
            xFFFFFFDF & cmd_val) | 0x40);

        PCI4E_SetMiscRegisterValue(BaseAddress, COMMAND_REG, (0
            xFFFFFFBF & cmd_val) | 0x20);
}

void initialize_PCI4E(volatile uint32_T *BaseAddress)
{
      uint8_T k;

      // Configure encoder channel k
      for(k=0; k<2; k++)
      {
         PCI4E_SetRegisterValue(BaseAddress, k, PRESET_OFFSET,
            COUNTS_PER_REV-1);  // Set the preset register to the CPR
            -1
         PCI4E_SetMultiplier(BaseAddress, k, MULTIPLIER);
            // Quadrature x 4
         PCI4E_SetCounterMode(BaseAddress, k, 0);              // 24 bit
             counter
         PCI4E_SetCount(BaseAddress,k,0);                     // Set the
            encoder count to zero
         PCI4E_SetEnableAccumulator(BaseAddress,k,1);   // Enables the
             counter **IMPORTANT**
         PCI4E_SetForward(BaseAddress,k,FORWARD_DIR);                //
            Optional: determines the direction of counting.
      }
      PCI4E_ResetTimeStamp(BaseAddress);
}

void getEncoderSpeeds(volatile uint32_T *BaseAddress, uint32_T *
    DWorkPointer, real_T *rps0, real_T *rps1)
{
      uint32_T OldTimeStamp = DWorkPointer[TS_LOC];
      uint32_T NewTimeStamp, Count[4];

      real_T ElapsedTime, rps[2];
      int32_T  ElapsedCount[2];
      real_T CLK_SPEED = 33000000.0;
      uint8_T k;

      PCI4E_CaptureTimeAndCounts(BaseAddress, &Count, &NewTimeStamp);

      if ( NewTimeStamp > OldTimeStamp ) {
           ElapsedTime = ((NewTimeStamp - OldTimeStamp)*1.0)/CLK_SPEED;
      }
      else {
            ElapsedTime = ((4294967296 - OldTimeStamp + NewTimeStamp)
                *1.0)/CLK_SPEED;
        }

        for(k=0; k<2; k++) {
            ElapsedCount[k] = Count[k] - DWorkPointer[ENC_COUNTS_LOC+k];

            if ( abs(ElapsedCount[k]) > MAX_COUNTS_DIFFERENCE ) {
                if ( Count[k] > DWorkPointer[ENC_COUNTS_LOC+k] ) {
                    // Assume it spun backwards
                    ElapsedCount[k] = -(16777216 - Count[k] +
                        DWorkPointer[ENC_COUNTS_LOC+k]);
                }
                else {
                    // Assume it spun forwards
                    ElapsedCount[k] = 16777216 - DWorkPointer[
                        ENC_COUNTS_LOC+k] + Count[k];
                }
            }
            rps[k] = (ElapsedCount[k]/(COUNTS_PER_REV*MULTIPLIER))/
                ElapsedTime;
        }

        // Update DWorkPointer for next cycle
        DWorkPointer[TS_LOC] = NewTimeStamp;
        for(k=0;k<2;k++) {
            DWorkPointer[ENC_COUNTS_LOC+k] = Count[k];
        }

        *rps0 = rps[0];
        *rps1 = rps[1];
}
#endif // __PCI_1710_4E_COMM_INCLUDED
```

G.5 PCI4E_Speed_Function_Definitions.h

Listing G.5: PCI4E_Speed_Function_Definitions.h

```
#ifndef __PCI4E_SPEED_FUNCTION_DEF_INCLUDED
#define __PCI4E_SPEED_FUNCTION_DEF_INCLUDED

/* Location within the DWorkPointer of various variables */
#define PCI_BAR_LOC 2
#define PCI_BAR_MEM_LEN_LOC 3
#define TS_LOC 4
#define ENC_COUNTS_LOC 5
#define GEN_SPEEDS_LOC 9
#define CV_LOC 11
#define CV_MAP_LOC 23
#define COS_SIN_VEC_LOC 25
#define PAR_RLY_LOC 27
#define GEN_ONE_RLY_LOC 28
```

```
#define GEN_TWO_RLY_LOC 29

/* Sample period and time */
#define FS  960.0
#define SAMPLE_TIME 0.0010416
#define PI 3.14159265359

/* PCI4E Functions Constants */
#define PRESET_OFFSET 0
#define OUTPUT_OFFSET 1
#define CONTROL_OFFSET 3
#define STATUS_OFFSET 4
#define TRANSFER_OFFSET 6
#define COMMAND_REG 7
#define TSLATCH_REG 15

#define COUNTS_PER_REV 1800.0
#define X1 1
#define X2 2
#define X4 4
#define FORWARD_DIR 1
#define MAX_COUNTS_DIFFERENCE 16383 //2^14-1

#define MULTIPLIER X4

#endif // __PCI4E_SPEED_FUNCTION_DEF_INCLUDED
```

G.6 vectormath.c

Listing G.6: vectormath.c

```
/*
    C Code to perform needed vector and other mathematical
       operations

    Written by Uzoma Orji April 9, 2012
*/

#ifndef __VECTORMATH_INCLUDED
#define __VECTORMATH_INCLUDED

/* Functions */
real_T min2(real_T num1, real_T num2)
{
    if(num2<num1)
    {
        return num2;
    }
    else
    {
        return num1;
    }
```

```
}

real_T max2(real_T num1, real_T num2)
{
    if(num2>num1)
    {
        return num2;
    }
    else
    {
        return num1;
    }
}

real_T mean(real_T *vecptr, uint8_T len)
{
    real_T avg = 0;
    uint8_T k;
    for(k=0;k<len;k++)
    {
        avg = avg + vecptr[k];
    }
    avg = avg / len;

    return avg;
}

real_T round2(real_T num)
{
    return floor(num+0.5);
}

real_T myabs(real_T num)
{
    if(num<0)
    {
        return num * -1.0;
    }
    else
    {
        return num;
    }
}

uint8_T minIndex(real_T *vec, uint8_T len)
{
    uint8_T minind, k;
    real_T minval;

    minind = 0;
    minval = vec[minind];

    for(k=1;k<len;k++)
    {
            if (vec[k]<minval)
            {
                minind = k;
                minval = vec[k];
            }
        }

        return minind;
}


real_T dotProduct(real_T *basisptr, real_T *CVvecptr, uint8_T N)
{
        // *basisptr is the basis pointer of the cosine of sine basis
            function
        // *CVvecptr is the CV vector pointer
        // N is the number of points in basis vector and number of
            recent
        //  points in CV vec

        uint8_T k;
        real_T val=0;

        for(k=0;k<N;k++)
        {
            val = val + basisptr[k]*CVvecptr[k];
        }

        return val;
}

void updateVector(real_T *vecptr, real_T val, uint8_T N)
{
        // update vector by inserting val at the top (0-index)
        // *vecptr is the vector to update
        // val is the most recent value to insert into *vector
        // N is length of vector

        uint8_T k;

        for(k=N-1;k>0;k--)
        {
            vecptr[k] = vecptr[k-1];
        }

        vecptr[0] = val;
}

#endif // __VECTORMATH_INCLUDED
```

G.7 Front Panel Circuit Board Schematic

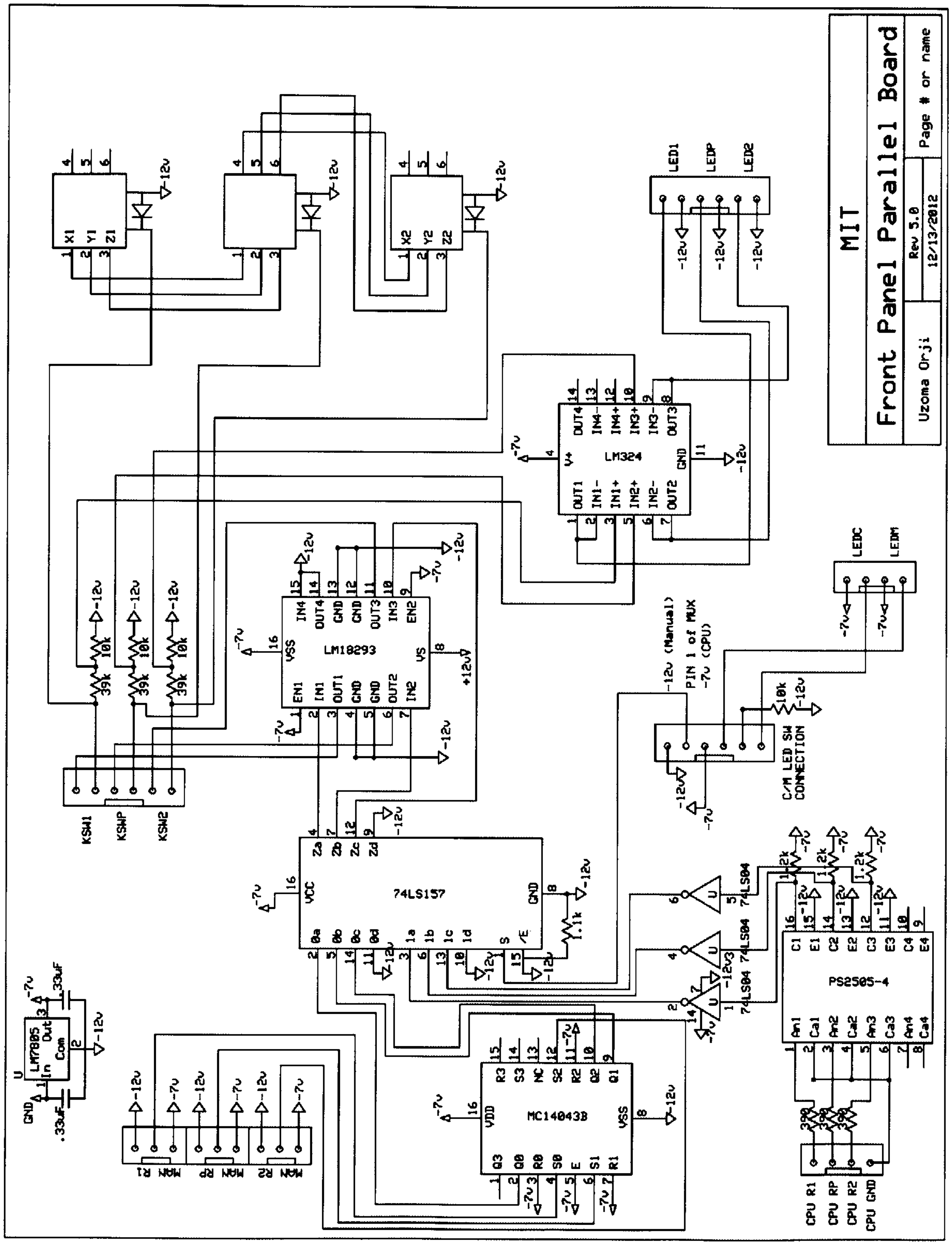


Fig. G-1: Front panel circuit board schematic.

Appendix H

Table of Acronyms

AAW	Anti-Air Warfare
AC	Air Conditioning
ACB	Air Circuit Breakers
AHU	Air Handling Unit
ASUW	Anti-Surface Warfare
ASW	Anti-Submarine Warfare
CBM	Condition-Based Maintenance
CDF	Cumulative Distribution Function
CG	Guided Missile Cruiser
CPP	Controllable Pitch Propeller
DDG	Guided Missile Destroyer
DDS	Data Design Sheet
EPLA	Electrical Power Load Analysis
FFT	Fast Fourier Transform
FOSP	Fuel Oil Service Pump
GTG	Gas Turbine Generator
GTM	Gas Turbine Module
GUI	Graphical User Interface
HED	Hybrid Electric Drive
HIF	High Impedance Faults
HFAC	High Frequency Alternating Current
ICAS	Integrated Condition Assessment System
IES	Industrial Electric Shop
IPCS	Integrated Protective Coordination System
IPS	Integrated Power Systems
LBES	Land-Based Engineering Site
MCMAS	Machinery Control Message Acquisition System
MFM	Multi-Function Monitor
MLO	Main Lubricating Oil
MMF	Magnetomotive Force
MRG	Main Reduction Gear
MVAC	Medium Volt Alternating Current
MVDC	Medium Volt Direct Current

NAVSEA	Naval Sea Systems Command
NGIPS	Next Generation Integrated Propulsion Systems
NILM	Non-Intrusive Load Monitoring
NITC	Non-Intrusive Transient Classifier
PDF	Probability Distribution Function
PSH	Principal Slot Harmonic
QOS	Quality of Service
RMD	Restricted Maneuvering Doctrine
SHP	Shaft Horsepower
USCG	United States Coast Guard
USN	United States Navy
VAMPIRE	Vibration Assessment Monitoring Point with Integrated Recovery of Energy
ZEDS	Zonal Electrical Distribution System

Bibliography

[1] U. S. Department of Energy, "The Smart Grid: An Introduction," World Wide Web electronic publication. [Online]. Available: http://www.oe.energy.gov/1165.htm

[2] S. R. Shaw, S. B. Leeb, L. K. Norford, and R. W. Cox, "Nonintrusive Load Monitoring and Diagnostics in Power Systems," *IEEE Trans. Instrum. Meas.*, vol. 57, no. 7, pp. 1445–1454, Jul. 2008.

[3] J. S. Ramsey *et al.*, "Shipboard Applications of Non-Intrusive Load Monitoring," in *ASNE Conference on Survivability and Reconfiguration*, Feb. 2005.

[4] T. DeNucci, R. Cox, S. B. Leeb, J. Paris, T. J. McCoy, C. Laughman, and W. C. Greene, "Diagnostic indicators for shipboard systems using non-intrusive load monitoring," *IEEE Electric Ship Technologies Symposium*, pp. 413–420, Jul. 2005.

[5] W. Greene, R. J. S., R. Cox, and T. DeNucci, "Non-intrusive monitoring for condition-based maintenance," *Proc. ASNE Reconfiguration and Survivability Symposium*, Feb. 16 2005.

[6] W. Wichakool, "Advanced Nonintrusive Load Monitoring System," Ph. D. Defense, Sep. 13 2010.

[7] S. B. Leeb, "A Conjoint Pattern Recognition Approach to Nonintrusive Load Monitoring," Ph.D. dissertation, Massachusetts Institute of Technology, Cambridge, MA, Feb. 1993.

[8] S. B. Leeb, S. R. Shaw, and J. L. Kirtley Jr., "Transient Event Detection in Spectral Envelope Estimates For Nonintrusive Load Monitoring," *IEEE Trans. Power Delivery*, vol. 10, no. 3, pp. 1200–1210, Jul 1995.

[9] S. R. Shaw, "System identification techniques and modeling for nonintrusive load diagnostics," Ph.D. dissertation, Massachusetts Institute of Technology, Cambridge, MA, Feb. 2000.

[10] R. Cox, S. Leeb, S. Shaw, and L. Norford, "Transient event detection for nonintrusive load monitoring and demand side management using voltage distortion," in *Twenty-First Annual IEEE Applied Power Electronics Conference and Exposition, 2006. APEC '06.*, march 2006.

[11] K. D. Hurst and T. G. Habetler, "Sensorless Speed Measurement Using Current Harmonic Spectral Estimation in Induction Machine Drives," *IEEE Trans. Power Electron.*, vol. 11, no. 1, pp. 66–73, 1996.

[12] A. Ferrah, K. G. Bradley, and G. M. Asher, "Sensorless Speed Detection of Inverter Fed Induction Motors Using Rotor Slot Harmonics and Fast Fourier Transform," vol. 1. IEEE Power Electronics Specialist Conference, Jun. 29 - Jul. 3 1992, pp. 279–286.

[13] A. Ferrah, P. J. Hogben-Laing, K. J. Bradley, G. M. Asher, and M. S. Woolfson, "The Effect of Rotor Design on Sensorless Speed Estimation Using Rotor Slot Harmonics Identified by Adaptive Digital Filtering Using The Maximum Likelihood Approach," in *Industry Applications Conference, 1997. Thirty-Second IAS Annual Meeting, IAS'97., Conference Record of the 1997 IEEE*, vol. 1, Oct. 1997, pp. 128–135.

[14] A. Ferrah, K. J. Bradley, P. J. Hogben-Laing, M. S. Woolfson, G. M. Asher, M. Sumner, J. Cilia, and J. Shuli, "A Speed Identifier For Induction Motor Drives Using Real-Time Adaptive Digital Filtering," *IEEE Trans. Ind. Applicat.*, vol. 34, no. 1, pp. 156–162, 1998.

[15] K. D. Hurst and T. G. Habetler, "A Comparison of Spectrum Estimation Techniques For Sensorless Speed Detection in Induction Machines," *IEEE Trans. Ind. Applicat.*, vol. 33, no. 4, pp. 898–905, 1997.

[16] M. E. H. Benbouzid, "A Review of Induction Motors Signature Analysis as a Medium For Faults Detection," *IEEE Trans. Ind. Electron.*, vol. 47, no. 5, pp. 984–993, 2000.

[17] G. B. Kliman, R. A. Koegl, J. Stein, R. D. Endicott, and M. W. Madden, "Noninvasive Detection of Broken Rotor Bars in Operating Induction Motors," *IEEE Trans. Energy Conversion*, vol. 3, no. 4, pp. 873–879, 1988.

[18] G. B. Kliman and J. Stein, "Methods of Motor Current Signature Analysis," *Electric Machines and Power Systems*, vol. 20, no. 5, pp. 463–474, 1992.

[19] S. Nandi, H. A. Toliyat, and X. Li, "Condition Monitoring and Fault Diagnosis of Electrical Motors - A Review," *IEEE Trans. Energy Conversion*, vol. 20, no. 4, pp. 719–729, 2005.

[20] N. M. Elkasabgy, A. R. Eastham, and G. E. Dawson, "Detection of Broken Bars in the Cage Rotor on an Induction Machine," *IEEE Trans. Ind. Applicat.*, vol. 28, no. 1 Part 1, pp. 165–171, 1992.

[21] W. T. Thomson and M. Fenger, "Current Signature Analysis to Detect Induction Motor Faults," *IEEE Ind. Appl. Mag.*, vol. 7, no. 4, pp. 26–34, 2001.

[22] R. R. Schoen, T. G. Habetler, F. Kamran, and R. G. Bartfield, "Motor Bearing Damage Detection Using Stator Current Monitoring," *IEEE Trans. Ind. Applicat.*, vol. 31, no. 6, pp. 1274–1279, 1995.

[23] R. R. Schoen and T. G. Habetler, "Effects of Time-Varying Loads on Rotor Fault Detection in Induction Machines," *IEEE Trans. Ind. Applicat.*, vol. 31, no. 4, pp. 900–906, 1995.

[24] R. R. Schoen, B. K. Lin, T. G. Habetler, J. H. Schlag, and S. Farag, "An Unsupervised, On-line System For Induction Motor Fault Detection Using Stator Current Monitoring," in *Conference Record of the 1994 IEEE Industry Applications Society Annual Meeting, 1994.*, 1994, pp. 103–109.

[25] M. E. H. Benbouzid, M. Vieira, and C. Theys, "Induction Motors' Faults Detection And Localization Using Stator Current Advanced Signal Processing Techniques," *IEEE Trans. Power Electron.*, vol. 14, no. 1, pp. 14–22, 1999.

[26] J. R. Cameron, W. T. Thomson, and A. B. Dow, "Vibration and Current Monitoring for Detecting Airgap Eccentricity in Large Induction Motors," *IEE Proceedings B [see also IEE Proceedings-Electric Power Applications] Electric Power Applications*, vol. 133, no. 3, pp. 155–163, 1986.

[27] H. Guldemir, "Detection of Airgap Eccentricity Using Line Current Spectrum of Induction Motors," *Electric Power Systems Research*, vol. 64, no. 2, pp. 109–117, 2003.

[28] S. Nandi, S. Ahmed, and H. A. Toliyat, "Detection of Rotor Slot and Other Eccentricity Related Harmonics in a Three Phase Induction Motor With Different Rotor Cages," *IEEE Trans. Energy Conversion*, vol. 16, no. 3, pp. 253–260, 2001.

[29] M. Ishida and K. Iwata, "A New Slip Frequncy Detector of an Induction Motor Utilizing Rotor Slot Harmonics," *IEEE Trans. Ind. Applicat.*, pp. 575–582, 1984.

[30] D. S. Zinger, F. Profumo, T. Lipo, and D. W. Novotny, "A Direct Field-Oriented Controller for Induction Motor Drives Using Tapped Stator Windings," *IEEE Trans. Power Electron.*, vol. 5, no. 4, pp. 446–453, 1990.

[31] S. Nandi, "Modeling of Induction Machines Including Stator and Rotor Slot Effects," in *Industry Applications Conference, 2003. Conference Record of the 38th IAS Annual Meeting.*, vol. 2, 2003.

[32] A. V. Oppenheim, R. W. Schafer, and J. R. Buck, *Discrete-time Signal Processing.* Prentice Hall Englewood Cliffs, NJ, 1989.

[33] A. Cowan, "Review of Recent Commercial Rooftop Unit Field Studies in the Pacific Northwest and California," New Buildings Institute, PO Box 653, White Salmon,WA, 98672, Tech. Rep., Oct. 8 2004.

[34] R. J. Mowris, A. Blankenship, and E. Jones, "Field measurements of air conditioners with and without txvs," 2004.

[35] D. Hales, A. Gordon, and M. Lubliner, "Duct Leakage in New Washington State Residences: Findings and Conclusions," *ASHRAE Transactions-American Society of Heating Refrigerating Airconditioning Engineers*, vol. 109, no. 2, pp. 393–402, 2003.

[36] K. Srinivasan, "Measurement of Air Leakage in Air-Handling Units and Air Conditioning Ducts," *Energy & Buildings*, vol. 37, no. 3, pp. 273–277, 2005.

[37] C. Laughman, "Fault Detection Methods for Vapor-Compression Air Conditioners Using Electrical Measurements," Ph.D. dissertation, Massachusetts Institute of Technology, Cambridge, MA, Sep. 2008.

[38] C. J. Schantz, "Non-Intrusive Fault Detection in Reciprocating Compressors," Master's thesis, Massachusetts Institute of Technology, Cambridge, MA, Jun. 2011.

[39] J. Paris, "A comprehensive system for non-intrusive load monitoring and diagnostics," Ph.D. dissertation, Massachusetts Institute of Technology, Department of Electrical Engineering and Computer Science, September 2013.

[40] J. S. Donnal, "Home NILM: A Comprehensive Energy Monitoring Toolkit," Master's thesis, Massachusetts Institute of Technology, Cambridge, MA, Jun. 2013.

[41] C. Hargis, B. G. Gaydon, and K. Kamash, "The Detection of Rotor Defects in Induction Motors," in *Proc IEE EMDA Conf, London*, 1982, pp. 216–220.

[42] E. Linton, "Coast Guard RCM/CBM Initiatives." Presented at the 2005 Department of Defense Maintenance Symposium and Exhibition, Birmingham, AL, Oct. 24 2005.

[43] W. T. Thomson, R. A. Leonard, A. J. Milne, and J. Penman, "Failure Identification of Offshore Induction Motor Systems Using On-condition Monitoring," *Reliability Engineering*, vol. 9, no. 1, pp. 49–64, 1984.

[44] U. A. Orji, Z. Remscrim, C. Laughman, S. B. Leeb, W. Wichakool, C. Schantz, R. Cox, J. Paris, J. L. Kirtley, and L. K. Norford, "Fault Detection and Diagnostics for Non-Intrusive Monitoring using Motor Harmonics," in *Applied Power Electronics Conference and Exposition (APEC), 2010 Twenty-Fifth Annual IEEE*, Palm Springs, CA, Feb. 2010, pp. 1547–1554.

[45] U. A. Orji, C. Schantz, D. E. Barber, S. B. Leeb, R. Cox, A. Goshorn, and K. Thomas, "Electrical-based Condition Monitoring Using Shaft Speed Oscillation Harmonics," in *American Society of Navel Engineers (ASNE), ASNE Day 2011 "Thinking Outside the Hull"*, Arlington, VA, Feb. 2011.

[46] M. DiUlio, C. Savage, B. Finley, and E. Schneider, "Taking the Integrated Condition Assessment System to the Year 2010," in *13th Int. Ship Control Systems Symposium*, Orlando, FL., 2003.

[47] J. O. Smith III, *Mathematics of the Discrete Fourier Transform (DFT): with Audio Applications – Second Edition.* W3K Publishing, 2007.

[48] J. Donnal, U. A. Orji, C. Schantz, J. Moon, S. B. Leeb, J. Paris, A. Goshorn, K. Thomas, J. Dubinsky, and R. Cox, "VAMPIRE: Accessing a Life-Blood of Information for Maintenance and Damage Assessment," in *American Society of Navel Engineers (ASNE), ASNE Day 2012 "Naval Warfare - Critical Engineering Challenges"*, Arlington, VA, Feb. 2012.

[49] J. S. Webster, H. Fireman, D. A. Allen, A. J. Mackenna, and J. C. Hootman, "Alternative propulsion methods for surface combatants and amphibious warfare ships," *Transactions of the Society of Naval Architects and Marine Engineers*, vol. 115, pp. 224–262, Jan. 2007.

[50] N. Doerry and K. McCoy, "Next Generation Integrated Power System: NGIPS Technology Development Roadmap," DTIC Document, Tech. Rep., 2007.

[51] "DDS 310-1 Electric Power Load Analysis (EPLA) For Surface Ships," Naval Sea Systems Command (NAVSEA), Washington Navy Yard, DC 20376-5124.

[52] N. Doerry, "Electric Power Load Analysis," *Naval Engineers Journal*, vol. 124, no. 4, pp. 45–48, Dec. 2012.

[53] N. H. Doerry and J. Amy, "Implementing quality of service in shipboard power system design," in *Electric Ship Technologies Symposium (ESTS), 2011 IEEE.* IEEE, 2011, pp. 1–8.

[54] C. B. Office, "An Analysis of the Navy's Fiscal Year 2013 Shipbuilding Plan," Jul. 2012.

[55] T. McCOY, J. Zgliczynski, N. W. Johanson, F. A. Puhn, and T. W. Martin, "Hybrid electric drive for ddg-51 class destroyers," *Naval Engineers Journal*, vol. 119, no. 2, pp. 83–91, 2007.

[56] D. S. Cusanelli and G. Karafiath, "Hydrodynamic energy saving enhancements for ddg 51 class ships," DTIC Document, Tech. Rep., 2012.

[57] US NAVY, "United States Navy Fact File: Destroyers-DDG," Department of the Navy.

[58] ——, "DDG 51 Hull number 91 and higher AC ZEDS training," Department of the Navy.

[59] Naval Sea Systens Command, "Guide to MFM Operations," Surface Warface Center, Carderock Division, Philadelphia, Tech. Rep., 2003.

[60] I. McNab and F. C. Beach, "Naval Railguns," *IEEE Transactions on Magnetics*, vol. 43, no. 1, pp. 463–468, 2007.

[61] T. Goodridge, "Baseline Report of the USS Spruance for Bath Iron Works," Alaris Companies, 140 Second Street Suite 250 Petaluma, CA 94952, Tech. Rep., Nov. 2010.

[62] ——, "Supplementary Report Measured Data for Bath Iron Works. Special Ship Study USS Spruance," Alaris Companies, 140 Second Street Suite 250 Petaluma, CA 94952, Tech. Rep., Nov. 2010.

[63] B. Sievenpiper, T. Anderson, and K. Gerhard, "Operational Ship Utilization Modeling of the DDG-51 Class." Presented at the ASNE Day 2013, Crystal City, VA, 2013.

[64] J. A. Cairns, "DDG51 Class Land Based Engineering Site (LBES) – The Vision and the Value," *Naval Engineers Journal*, vol. 123, no. 2, pp. 73–83, 2011.

[65] P. L. Bennett, "Using the Non-intrusive Load Monitor for Shipboard Supervisory Control," Ph.D. dissertation, Massachusetts Institute of Technology, Cambridge, MA, Jun. 2007.

[66] K. Douglas, "Shipboard Aggregate Power Monitoring," Master's thesis, Massachusetts Institute of Technology, Cambridge, MA, Jun. 2009.

[67] R. Jones, "Improving Shipboard Application of Non-Intrusive Load Monitoring," Master's thesis, Massachusetts Institute of Technology, Cambridge, MA, Jun. 2008.

[68] "Ship Information Book DDG 100 USS KIDD, Part 1 Propulsion Plant," Naval Sea Systems Command (NAVSEA), Jul. 28 2010.

[69] "Ship Information Book DDG 100 USS KIDD, Part 2 Book 3 Auxiliary Machinery, Piping and Environmental Pollution Control. Chapters 11-13," Naval Sea Systems Command (NAVSEA), Jul. 28 2010.

[70] R. O'Rourke, "Navy ddg-51 and ddg-1000 destroyer programs: Background and issues for congress," in *Congressional Research Service.* DTIC Document, Apr. 19 2011.

[71] Bath Iron Works, "DDG-51 Class Electric Load Analysis," Department of the Navy, Tech. Rep.

[72] US Coast Guard, "Maritime Security Cutter, Medium (WMSM)."

[73] M. D. Coutto Filho, A. Leite Da Silva, V. Arienti, and S. Ribeiro, "Probabilistic load modelling for power system expansion planning," in *Probabilistic Methods Applied to Electric Power Systems, 1991., Third International Conference on.* IET, 1991, pp. 203–207.

[74] V. Neimane, "Distribution network planning based on statistical load modeling applying genetic algorithms and monte-carlo simulations," in *Power Tech Proceedings, 2001 IEEE Porto*, vol. 3. IEEE, 2001, pp. 5–10.

[75] M. Meldorf, T. Taht, and J. Kilter, "Stochasticity of the electrical network load," *Oil Shale*, vol. 24, no. 2, pp. 225–236, 2007.

[76] D. P. Bertsekas and J. N. Tsitsiklis, *Introduction to probability.* Athena Scientific Belmont, MA, 2008, vol. 2.

[77] D. G. Kendall, "Stochastic processes occurring in the theory of queues and their analysis by the method of the imbedded markov chain," *The Annals of Mathematical Statistics*, pp. 338–354, 1953.

[78] Naval Sea Systems Command, "Naval Ships' Technical Manual Chapter 320, Electrical Power Distribution Systems," Naval Surface Warfare Center, Carderock Division, Tech. Rep., Apr. 1998.

[79] L. Plesnick, T. Hannon, and D. Devine, "An Intelligent Fault Detection Device For Shipboard Power Systems," Naval Surface Warfare Center, Carderock Division, Tech. Rep.

[80] D. P. Wipf and B. E. Parker Jr., "Demonstration and Assessment of High Speed Relay Algorithm," Barron Associates, Inc., 1160 Pepsi Place, Suite 300 Charlottesville, VA 22907-0807, Tech. Rep., Jun. 30 1997.

[81] P. C. Krause, O. Wasynczuk, and S. D. Sudhoff, *Analysis of Electric Machinery and Drive Systems.* IEEE Press New York, 2002.

[82] R. Christie, H. Zadehgol, and M. Habib, "High impedance fault detection in low voltage networks," *Power Delivery, IEEE Transactions on*, vol. 8, no. 4, pp. 1829–1836, 1993.

[83] M. Aucoin, B. Russell, and C. Benner, "High impedance fault detection for industrial power systems," in *Industry Applications Society Annual Meeting, 1989., Conference Record of the 1989 IEEE*, 1989, pp. 1788–1792 vol.2.

[84] A. Girgis, W. Chang, and E. Makram, "Analysis of high-impedance fault generated signals using a kalman filtering approach," *Power Delivery, IEEE Transactions on*, vol. 5, no. 4, pp. 1714–1724, 1990.

[85] C. Benner and B. Russell, "Practical high impedance fault detection for distribution feeders," in *Rural Electric Power Conference, 1996. Papers Presented at the 39th Annual Conference*, 1996, pp. B2–1–B2–6.

[86] S.-J. Huang and C.-T. Hsieh, "High-impedance fault detection utilizing a morlet wavelet transform approach," *Power Delivery, IEEE Transactions on*, vol. 14, no. 4, pp. 1401–1410, 1999.

[87] D. Hou and N. Fischer, "Deterministic high-impedance fault detection and phase selection on ungrounded distribution systems," in *Power Systems Conference: Advanced Metering, Protection, Control, Communication, and Distributed Resources, 2006. PS '06*, 2006, pp. 112–122.

[88] D. Hou, "Detection of high-impedance faults in power distribution systems," in *Power Systems Conference: Advanced Metering, Protection, Control, Communication, and Distributed Resources, 2007. PSC 2007*, 2007, pp. 85–95.

[89] T. Cui, X. Dong, Z. Bo, A. Klimek, and A. Edwards, "Modeling study for high impedance fault detection in mv distribution system," in *Universities Power Engineering Conference, 2008. UPEC 2008. 43rd International*, 2008, pp. 1–5.

[90] H. Rickover and P. Ross, "Fault protection on shipboard ac power-distribution systems," *Electrical Engineering*, vol. 63, no. 12, pp. 1099–1120, 1944.

[91] M. Michalik, W. Rebizant, M. Lukowicz, S.-J. Lee, and S.-H. Kang, "High-impedance fault detection in distribution networks with use of wavelet-based algorithm," *Power Delivery, IEEE Transactions on*, vol. 21, no. 4, pp. 1793–1802, 2006.

[92] C. S. Por, K. L. Choo, and L. Y. Jian, "A study of arc fault current in low voltage switchboard," in *Sustainable Utilization and Development in Engineering and Technology (STUDENT), 2012 IEEE Conference on*, 2012, pp. 52–56.

[93] S. Liao, R. Zhang, Y. Huang, and H. Xia, "Research of low-voltage arc fault classification based on support vector machine," in *Automatic Control and Artificial Intelligence (ACAI 2012), International Conference on*, 2012, pp. 1690–1693.

[94] J. A. Momoh and A. S. Ishola-Salawu, "A new arcing fault modeling and detection technique for navy ips power system," in *Power Engineering Society General Meeting, 2006. IEEE*. IEEE, 2006, pp. 7–pp.

[95] D. Jeerings and J. Linders, "Unique aspects of distribution system harmonics due to high impedance ground faults," *Power Delivery, IEEE Transactions on*, vol. 5, no. 2, pp. 1086–1094, 1990.

[96] C. N. Tidd, "Hardware Model of a Shipboard Zonal Electrical Distribution System (ZEDS): Alternating Current/Direct Current (AC/DC)," Master's thesis, Massachusetts Institute of Technology, Cambridge, MA, Jun. 2010.

[97] P. L. Alger, *Induction Machines, Their Behavior and Uses.* Gordon & Breach Science Publishers, 1970.

[98] P. Vas, *Parameter Estimation, Condition Monitoring, and Diagnosis of Electrical Machines.* Oxford University Press, USA, 1993.

[99] S. Nandi, R. M. Bharadwaj, and H. A. Toliyat, "Performance Analysis of a Three Phase Induction Motor Under Mixed Eccentricity Condition," *IEEE Power Eng. Rev.*, vol. 22, no. 7, pp. 49–49, 2002.

[100] J. L. Kirtley Jr., *6.685 Electric Machines Lecture Notes: Fall 2005.*

[101] G. L. Elkins, "Hardware Model of a Shipboard Generator," Master's thesis, Massachusetts Institute of Technology, Cambridge, MA, Jun. 2009.

[102] China Electromechanics, "STC SERIES THREE-PHASE A.C. SYNCHRONOUS GENERATORS - China Electromechanics–motor, generator, pump, brake, auto, electrical & mechanics, Alternators, Generating Sets, Scissor, Valve, Hardware, Massager," World Wide Web electronic publication. [Online]. Available: http://en.ccmotor.cn/myjh/showproduct.asp?q=xx&id=7403&mylb=sell

[103] The MathWorks, Inc, "MathWorks - MATLAB and Simulink for Technical Computing," World Wide Web electronic publication. [Online]. Available: http://www.mathworks.com

[104] AMETEK, Inc, "XFR 2800 Watt Series Programmable DC Power Supply Operation Manual," World Wide Web electronic publication. [Online]. Available: http://www.sorensen.com/products/XFR/downloads/XFR2-8_Operation_Manual_D__TM-F2OP-C1XN-01.pdf

[105] ——, "XHR 1000 Watt Series Programmable DC Power Supply Operation Manual," World Wide Web electronic publication. [Online]. Available: http://www.sorensen.com/products/XHR/downloads/XHR_Operation_Manual_TM-XROP-01XN.pdf

[106] The MathWorks, Inc, "MATLAB Documentation Center," World Wide Web electronic publication. [Online]. Available: http://www.mathworks.com/help/documentation-center.html

[107] L. Miao, G. Zou, P. Shi, and X. Jiao, "Development of Hardware Driver for MATLAB/Simulink Real-Time Simulation," in *International Workshop on Intelligent Systems and Applications, 2009. ISA 2009*, May 2009, pp. 1–4.

[108] US Digital, "US Digital | Support >> Software >> PCI-4E," World Wide Web electronic publication. [Online]. Available: http://www.usdigital.com/support/software/pci-4e

[109] G. F. Franklin, J. D. Powell, and M. Workman, *Digital Control of Dynamic Systems.* Ellis-Kagle Press Half Moon Bay, CA, 1998.

[110] I. Boldea, *Synchronous Generators.* CRC Press, 2006.