Development of Real Time Non-Intrusive Load Monitor for Shipboard Fluid Systems

by

Perry L. Branch

B.S., Materials Engineering, Auburn University, 2002

[SD+M]

Submitted to the Department of Mechanical Engineering and Engineering Systems Division in Partial Fulfillment of the Requirements for the Degrees of

Naval Engineer
and
Master of Science in Engineering and Management
at the
Massachusetts Institute of Technology

June 2008

Signature of Author _____

Department of Mechanical Engineering and
System Design and Management Program
May 9, 2008

Certified by _____

Robert W. Cox
Assistant Professor of Electrical and Computer Engineering, UNC Charlotte
Thesis Supervisor

Certified by _____

Steven B. Leeb
Professor of Electrical Engineering and Computer Science & Mechanical Engineering
Thesis Supervisor

Accepted by _____

Pat Hale
Director, Systems Design and Management Fellows Program
Engineering Systems Division

Accepted by _____

Lallit Anand
Chairman, Department Committee on Graduate Students
Department of Mechanical Engineering

Page Intentionally Left Blank

Development of Real Time Non-Intrusive Load Monitor for Shipboard Fluid Systems
by
Perry L. Branch

Submitted to the Department of Mechanical Engineering on May 9, 2008 in Partial Fulfillment of the Requirements for the Degrees of

Naval Engineer
and
Master of Science in Engineering and Management

Abstract

Since the year 2000, the United States Navy has spent an average of half a billion dollars over the congressionally approved budget for shipbuilding. Additionally, most experts project that in order to meet the Chief of Naval Operation's goal of a 313 ship Navy, the annual ship building budget will have to increase by about two thirds. Exacerbating this problem is the rising cost of maintaining the current inventory of ships. The U.S. Navy has long used a requirements driven maintenance program to reduce the number of total system failures by conducting routine maintenance and inspections whether they are needed or not. In order to combat this problem the Navy will inevitably have to turn to a condition based maintenance system. The Non-Intrusive Load Monitor (NILM) is a system that can greatly enhance the ability to monitor the health of engineering systems while incurring a low acquisition cost and low technology risk.

This research focuses on the development of a real time user interface for the current NILM architecture in order to provide useful system information to an operator. Additionally, this research has shown that the NILM can be used effectively and reliably, to monitor equipment health, recognize and indicate abnormal operating conditions and casualties and provide invaluable information for training operators, diagnosing problems and troubleshooting. The NILM is an inexpensive and promising platform for monitoring equipment and reducing maintenance costs.

Thesis Supervisor: Robert W. Cox
Title: Professor of Electrical and Computer Engineering, UNC Charlotte

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

Thesis Supervisor: Pat Hale
Title: Pat Hale, Director, Systems Design and Management Fellows Program

Acknowledgements

The author would like to thank the following organizations and individuals for their assistance. This thesis would not have been possible without them.

- The Office of Naval Research's Control Challenge, ONR/ESRDC Electric Ship Integration Initiative and the Grainger Foundation, all of whom provided funding.
- LT Jennifer Haag, MKC Scott Galvin, EMC Kristin Kiehl and MK1 Coe, of the USCGC ESCANABA, for your support and unlimited patience with the MIT guys. Without you none of this would have been possible.
- Professor Robert Cox for your guidance when I needed it and your reverent patience during the occasional rant.
- Jim Paris for your computer savvy, but mainly for "screen," which saved me buku time.
- Ethan Proper who talked me into taking on the RO UI and then helping me through it.
- Ashley Fuller who was a genius when it came to installing mine and Ethan's crap.
- Professor Steven Leeb for letting a caveman on your team.
- Pat Hale for making SDM a reality for me.
- To Emily and Abby who never got upset when dad ignored you for countless hours while typing away on the couch. During those times of utter complacency you never burned down the house.
- Finally, to my wife Jeni. First Auburn and now MIT and never a foul word. But mainly for your love and your mad proof reading skills. War Eagle!

Contents

List of Figures

List of Tables

Page Intentionally Left Blank

1 Introduction

One of the biggest challenges that large engineering facilities face is monitoring equipment status and health. For example, a United States Navy ship consists of thousands of separate engineering systems. If measured using sensors or other external monitoring systems, then each system would require multiple monitoring points leading to a vast sensor array. Large numbers of sensors greatly increases the overall system complexity and is often a source of failure. Increased equipment monitoring, combined with condition based maintenance, can lower the overall cost of maintaining equipment. The real question is how to improve the monitoring system in order to maintain current system complexity at lower initial cost. The perfect monitoring system would be one that is cheap, easy to install and use, has a minimum number of monitoring points, high reliability and has little or no impact on overall system complexity. The Nonintrusive Load Monitor (NILM) was initially developed at Massachusetts Institute of Technology in 1993 (Leeb, 1993). Since then the concept has been applied to numerous systems and operating platforms. Until now the information gained by NILM systems has been collected and analyzed post event, with no real time input to the operator. This research focuses on applying NILM technology to a particular engineering system, specifically the RC7000 Reverse Osmosis Desalination Plant, used on United States Coast Guard (USCG) medium endurance cutters, in order to enhance the overall performance and lower the costs of maintaining the system. It also focuses on improving the user interface in order to allow real time, useful output, to an untrained operator.

1.1 Motivation for Research – Condition Based Maintenance

Today's military faces a unique challenge, specifically the United States Navy. Currently naval leadership is proposing a 313 ship sustained force that must be designed, built and maintained. This plan is specifically outlined in the Chief of Naval Operation's 30 year plan. However, the Navy faces several road blocks in realistically achieving this goal. Figure 1-4 is the projected number of ships through 2090.

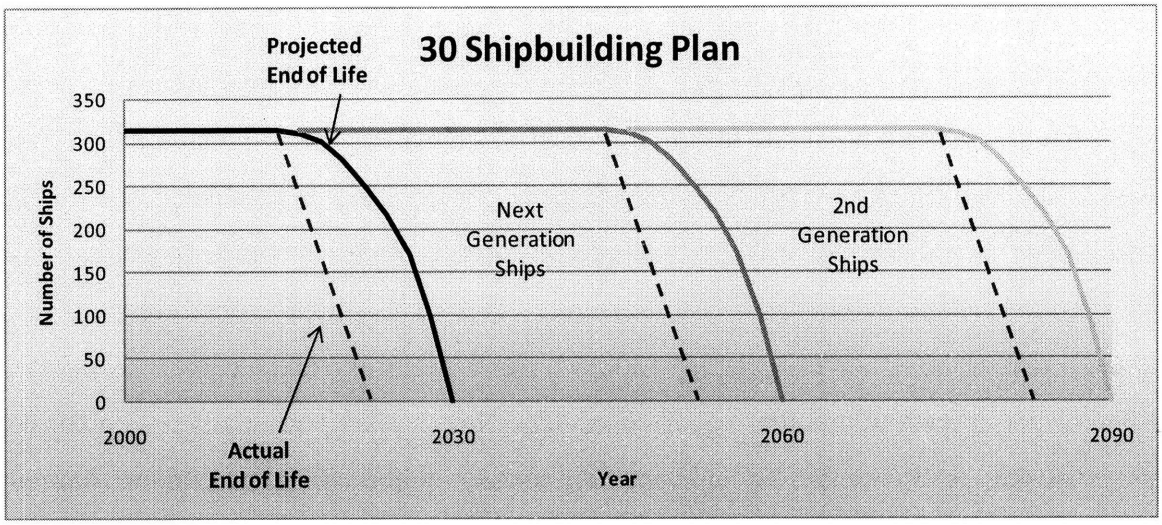


Figure 1-1: 30 Year ship building plan (McCoy, 2008)

The projected end of life down slope illustrates the rate at which existing platforms must be decommissioned in order to meet the projected goals. The red dashed line shows the actual out of service rate for existing platforms based on the projected cost and material condition of these ships. The distance between the two curves illustrates the gap in required force strength and the actual force strength available. These short falls are a result of two fundamental issues:

1. Rising maintenance costs.
2. Rising acquisition costs.

"Between 2000 and 2005, the Navy has spent on average $11.7 billion annually on shipbuilding, according to Congressional Budget Office (CBO) analyst Eric Labs. However, it is notable that the President's FY07 budget allowed for only $11.2 billion for new ship construction and conversions. More disturbing, the plan projects that in order to meet the 313 ship plan, funding levels will have to continue to rise in the coming years: $15.1 billion in FY-08, $14.9 billion in FY-09, $15.9 billion in FY-10, $19.9 billion in FY-11 and $20.5 billion in FY-12. CBO's analysis shows a grim outlook, Labs noted. If the Navy wants to implement its 313-ship plan, the service would need an average of $18.3 billion per year for shipbuilding, or $19.5 billion if nuclear refueling is included, he

said. That would be a two-thirds increase over the recent historical average and about one-quarter more than the Navy's plan (Castelli, 2007)".

Even with innovative cost cutting methods, acquisition costs will inevitably rise in the coming years and thus the Navy must find a way to cut costs in other areas in order to meet this demanding goal. Rear Admiral Kevin M. McCoy, the Chief Engineer of the Navy, has suggested that the Navy must focus on lowering maintenance costs. He specifically points out that yard periods must be used more effectively and efforts must be made to *reduce maintenance requirements*. This requirement driven approach is a direct result of the risk averse nature of today's naval engineers; and a culture change has to come about from within the organization. The Navy uses a preventative maintenance program that is driven by the concept of avoiding major equipment failure through periodic maintenance procedures and inspections. The success of this program is largely based on how accurate the requirement periodicities are set. Too few and the equipment fails catastrophically. Too many and thousands or perhaps millions of dollars are spent needlessly. These issues bring the concept of condition based maintenance to the forefront.

Condition based maintenance (CBM) can be described as using real time data from operating equipment in order to optimize resource allocation. Figure 1-5 is the modern representation of CBM.

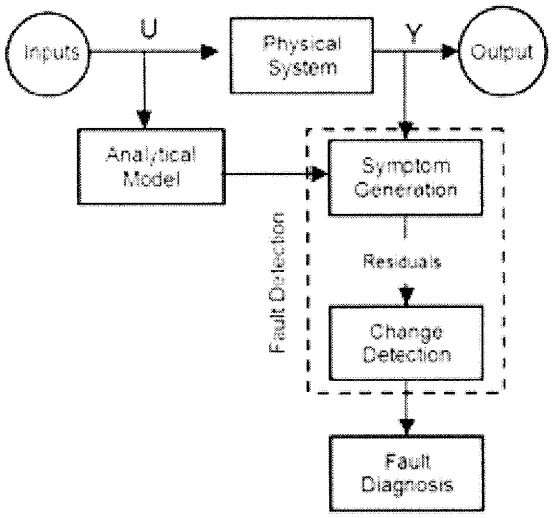


Figure 1-2: Simple Condition Based Maintenance Diagram (Condition Based Maintenance)

There are several methods that can be used in order to provide the impetus for CBM. Currently the Navy is using Integrated Condition Assessment System (ICAS) and vibration analysis to provide real time equipment monitoring information. However, ICAS uses a complex system of sensors and provides little or no diagnostic information that is not dependent on the operator. Likewise, vibration analysis requires extensive technical training and is not permanently installed on the equipment. NILM provides an easy and cheap solution to this problem. NILM has a low initial cost and does not impact the system complexity. This research demonstrates how NILM can be used to effectively monitor equipment and provide real time diagnostics and feedback to an untrained operator.

1.2 NILM Overview

The NILM monitors an aggregate electrical signal at a single point in a system or collection of systems, and disaggregates the signal in order to monitor the state and health of components within the network. The NILM collects a single point voltage from any node that is shared by all the components within a system. Current information is collected from either individual component nodes (during testing) or a common node that is shared by several components. The voltage and current information is then used to develop power traces for the system. Figure 1-1 is a block diagram of a standard NILM. Similar pieces of electrical equipment have unique characteristics or fingerprints that can identify individual components within a network. Currently, system characteristics are collected and analyzed by engineers who then develop and tailor diagnostic packages that best fit each system. The ultimate goal of NILM is to provide the user real time, useful information that can extend the life and lower the cost of maintaining equipment, without the use of a complex physical sensor network.

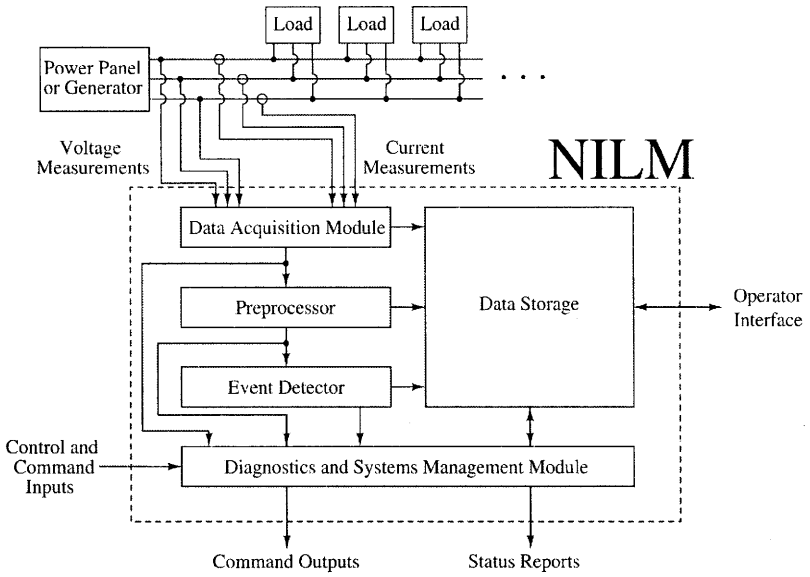


Figure 1-3: Diagram showing the fundamental signal flow path in a NILM (Cox, 2007).

Using measurements of the line voltage and aggregate current, a software-based preprocessor onboard the NILM computes time-varying estimates of the frequency content of the measured line current (Shaw, 2000). Formally, these time-varying estimates, or spectral envelopes, are defined as the quantities (Shaw, 1998)

$$a_m(t) = \frac{2}{T}\int_{t-T}^{t} i(\tau)\sin(m\omega\tau)\,d\tau \qquad (1)$$

and

$$b_m(t) = \frac{2}{T}\int_{t-T}^{t} i(\tau)\cos(m\omega\tau)\,d\tau\,. \qquad (2)$$

These equations are Fourier-series analysis equations evaluated over a moving window of length T (Oppenheim, 1988). The coefficients $a_m(t)$ and $b_m(t)$ contain time-local information about the frequency content of $i(t)$. Provided that the basis terms $\sin(m\omega t)$ and $\cos(m\omega t)$ are synchronized to the line voltage, the spectral envelope coefficients have a useful physical interpretation as real, reactive, and harmonic power (Leeb, 1995).

The spectral envelopes computed by the preprocessor are passed to an event detector that identifies the operation of each of the major loads on the monitored electrical service. In a modern NILM, identification is performed using both transient and steady-state information (Lee, 2003). Field studies have demonstrated that transient details are particularly powerful because the transient electrical behavior of a particular

load is strongly influenced by the physical task that is performed (Leeb, 1995). As shown in Fig. 1-2, for example, the physical differences between an incandescent lamp and an induction machine result in different transient patterns. Figure 1-3 demonstrates the positive identification of an induction motor driving a small vacuum pump.

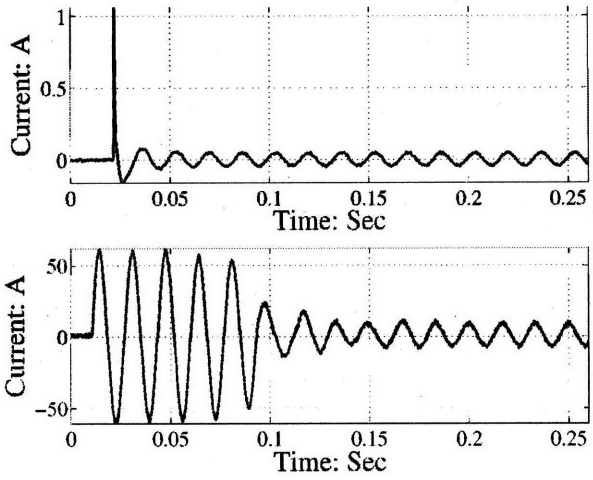


Figure 1-4: Top trace: Current drawn during the start of an incandescent lamp. Bottom trace: Stator current drawn during the start of an unloaded, fractional horsepower induction machine.

The final block in Figure 1-1 is the NILM's diagnostics and systems management module. This software unit assesses load status using any required combination of current data, voltage data, spectral envelopes, and load operating schedules (Cox R., 2006).

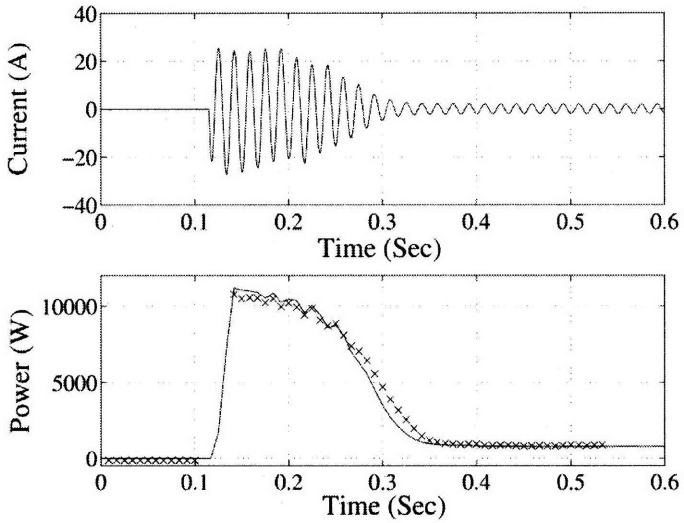


Figure 1-5: Measured current and computed power during the start of 1.7hp vacuum pump motor. Also shown in the power plot is a section of the template that has been successfully matched to the observed transient behavior by the NILM's event detector.

As shown in Fig. 1-1, the NILM is designed to interact with human or automated devices in a number of different ways. For instance, the NILM can use its diagnostic information to command certain loads to either commence or cease operations. Additionally, the NILM can provide regular status reports to human operators. To assist in future maintenance operations, the NILM stores all of the relevant data streams (i.e. currents, voltages, operating schedules, etc.) in either a local or remote database (Paris, 2006). The NILM's vast storage capabilities make it possible for the operator to perform historical data trending. The following chapters describe how a real-time user interface was developed for this NILM application.

2 Reverse Osmosis System Description - NILM Configuration

Previous researchers have deployed prototype NILM systems to monitor the RO units aboard both the USCGC SENECA and the USCGC ESCANABA (Denucci, 2005) (Mitchell, 2007). To date, RO system data has been collected using a Pentium Computer with hard drive (as indicated in Chapter 1) with no user interface and without real time feedback for the operators. To provide useful information in real time, extensive data processing must be performed. Further analysis is also required in order to learn specific indicators for specific casualty situations. These indicators must be tested to guarantee their reliability.

Chapter 2 and Chapter 3 of this thesis document the development of a reliable real-time user interface for a representative shipboard RO plant. This chapter begins the discussion by describing the system itself and by presenting previous work performed by LT Denucci and Mitchell. This chapter also provides key configuration information for the NILM system currently installed aboard the ESCANABA.

2.1 Reverse Osmosis Process

Osmosis is a naturally occurring process in which water diffuses through a semi-permeable barrier, such as a cell wall, from a low concentration solution to a high concentration solution. The osmotic pressure is the pressure required to stop the flow of water through a semi-permeable membrane into the solution of higher concentration. A semi-permeable membrane can be defined as a membrane that will allow pure solution and some types of solids to pass through but that will selectively prevent the passage of other types of solids. Osmotic pressure is a colligative property, meaning that it depends on the number of particles in solution but not on the mass of the particles (Maton & Jean Hopkins, 1997). Therefore, increasing the pressure increases the chemical potential in proportion to the molar volume. When a solute is dissolved in a solution, the mixing increases the overall entropy of the system, thereby decreasing the chemical potential. Since osmotic pressure is independent of the type of particles in solution, it can be represented by:

$$\mu = RTln(1 - x) \tag{3}$$

$$\delta PV = -RTln(1 - x) \tag{4}$$

$$\delta P = RTx/V \tag{5}$$

Where μ is the chemical potential, x is the mole fraction, δP is the osmotic pressure, Vis the molar volume, R is the gas constant (R = 8.314472(15) J · K^{-1} · mol^{-1} and T is the absolute temperature (Maton & Jean Hopkins, 1997). If pressure is increased above the osmotic pressure on the concentrated side of the membrane then the process can be reversed. This is called reverse osmosis. Typical seawater has an osmotic pressure of 350 psi and requires a reverse osmosis pressure of 600 – 1000 psi. This is illustrated in Figure 2-1.

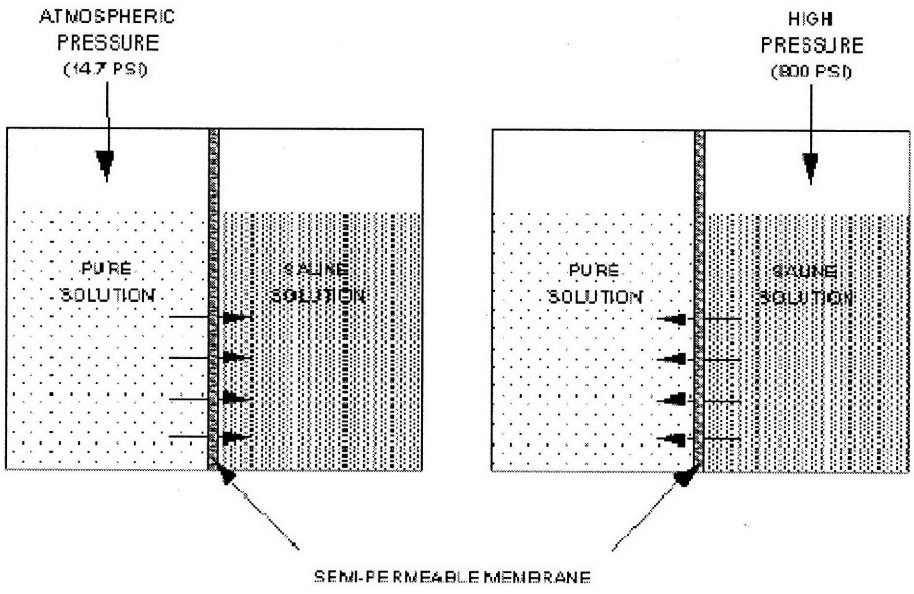


Figure 2-1: Simple Reverse Osmotic System (Operation and Maintenance Manual for USCG Model RC7000 Plus Reverse Osmosis Desalination Plant, 2007)

2.2 RC7000 Plus System Description

The RC7000 Plus produced by Village Marine is the RO system used aboard the Coast Guard's Medium Endurance Cutters and many other warships. This system consists of

two separate skids, a seawater strainer and feed pump skid and a separate RO system skid (Operation and Maintenance Manual for USCG Model RC7000 Plus Reverse Osmosis Desalination Plant, 2007). The seawater strainer and feed pump skid consists of a duplex seawater strainer and a 5HP low pressure centrifugal feed pump. The RO system skid consists of two identical reverse osmotic units (A and B) capable of producing 3,500 gpd each. Figure 2-2 is a simple diagram of the RO system. A more detailed system drawing can be found in Appendix A.

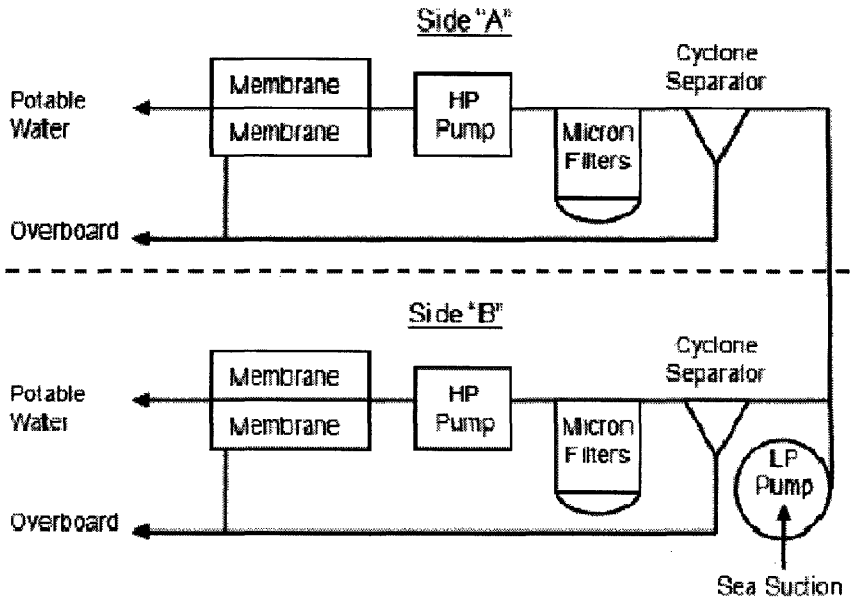


Figure 2-2: Simplified System Diagram (Mitchell, 2007)

The single low pressure feed pump draws suction from the ship's auxiliary seawater header, and supplies pressurized seawater to both plants. At the inlet of the feed pump is a duplex strainer that removes large particulate matter that could potentially damage the pump impellor. The system can be aligned to operate either or both sides at any time. The pressurized seawater flows through the cyclone separators, which remove most suspended solids. The water is then passed through the micron filter array. This array consists of two filters, one 20 micron and one 5 micron filter. The filtered seawater, now called feed water, passes through the high pressure pump. This pump raises the pressure of the feed water to 800 – 1000 psig before it passes into the membrane pressure vessel. The actual system pressure is controlled by a back pressure

regulating valve, which can be adjusted to meet the system needs in accordance with the RC7000 Plus Technical Manual. This pressurized water then flows into the membrane pressure vessel. The membrane pressure vessel consists of two 40 inch membranes, each 6 inches in diameter. When the system is operating, the feed water continuously flows over the membrane. However, once sufficient back pressure has been produced by the high pressure pump and back pressure regulating valve and pressure exceeds the osmotic pressure of the membrane, then pure water is forced through the membrane. The pure effluent then flows into the pure water manifold which contains the salinity monitoring as well as the brominating systems. The brine flows overboard through the back pressure regulating valve and the reject flow meter.

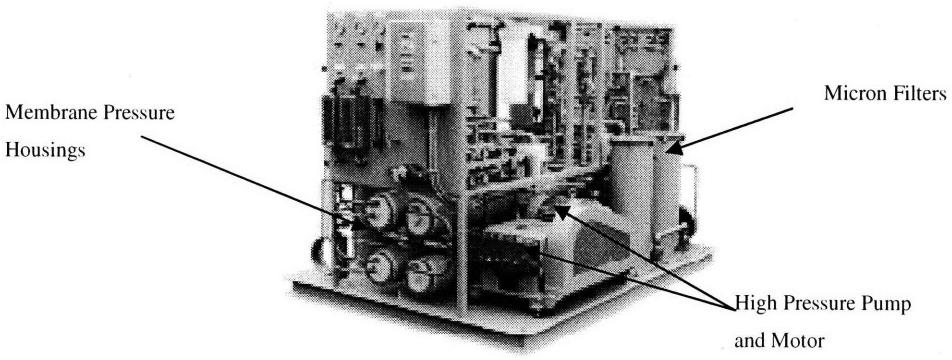


Figure 2-3: RO System Skid

One of the most important aspects of the reverse osmosis units onboard the USCGC ship is that operator actions are required for each step of operation. Further, the order in which the components are operated and how the components are operated is extremely important to the safe and effective use of this equipment. Appendix B describes the start up and shut down procedure for the RC7000 Plus Reverse Osmosis Desalination Plant. The importance of proper operation and being able to determine if the system was operated correctly will be discussed in detail in the diagnosis section of Chapter 5.

2.3 Typical Power Behavior

Figure 2-4 is an example of a real power trace collected using the NILM on USCGC ESCANABA in 2006. The first transient event corresponds to the start of the low pressure pump, which must always be started first. The second transient is caused by the start of one of the high pressure pumps. This event is followed approximately 20 seconds later by a gradual increase in power resulting from the manual closing of the bypass valve. The bypass valve directs the low pressure feed water either overboard (bypass valve open) or through the back pressure regulator (bypass valve shut). The difference in power level is directly proportional to the amount of work being done by the high pressure pump in order to maintain the required pressure. When the bypass valve is open and its associated high pressure pump is operating the system pressure is approximately 60 psig. Once the valve is completely shut, flow is diverted through the back pressure regulator and the system pressure rises to its operating pressure of 800 – 1000 psig. This pressure can be set by adjusting the back pressure regulator in accordance with technical guidance from the manufacturer. Figure 2-4 also shows the start of the other high-pressure pump and the corresponding closing of the bypass valve.

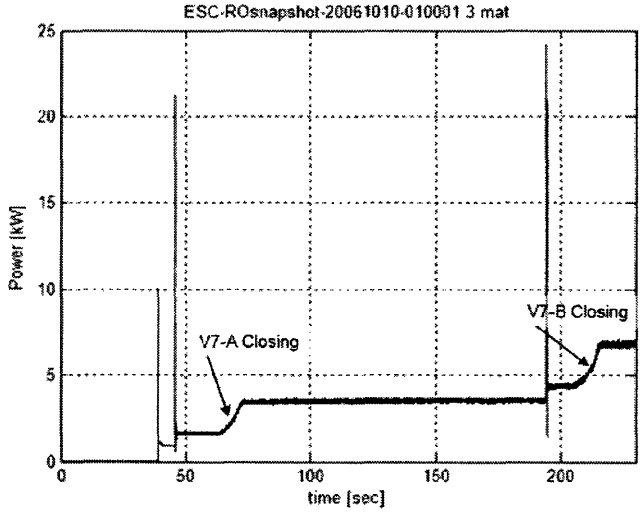


Figure 2-4: System Power Trace (Mitchell, 2007)

2.4 NILM Configuration and Data Collection

The NILM configuration has undergone multiple changes from 2005 to present. The initial install on the USCGC SENECA was set up with three separate current transducers (CT), one for each pump, and a common voltage source. The USCGC ESCANABA was also initially set up with 3 separate current transducers; however one of the CTs was set up to monitor the aggregate system current. Eventually, both ships were configured with only one common current and one common voltage source. Tables 2-1 and 2-2 describe the NILM configuration for both the USCGC SENECA and ESCANABA. The NILM on the USCGC SENECA was removed in September of 2007 for maintenance availability. For information about the role of measuring resistors, gain codes, and other specifics mentioned in Tables 2-1 and 2-2 see (Bennet, 2007).

Table 2-1: USCGC SENECA NILM Setup

	Channel	Component	Resistance	Gain	Data
20060618-20061216	1	B-C Voltage	100 Ω	0	Raw
	2	i-A LP Pump, CT: LA-100P	49.9 Ω	1	
	3	B-C Voltage	100 Ω	0	Raw
	4	i-A HP-A Pump, CT: LA-350S	49.9 Ω	0	
	5	B-C Voltage	100 Ω	0	Raw
	6	i-A HP-B Pump, CT: LA-350S	49.9 Ω	0	
20061216-20070906	1	A-B Voltage	100 Ω	0	Raw
	2	i-C 3-Pump Aggregate, CT: LA-150S	30.1 Ω	0	
20070906-Present	N/A	Uninstalled			

Table 2-2: USCGC ESCANABA NILM Setup

	Channel	Component	Resistance	Gain	Data
20060618-20061216	1	A-B Voltage	100 Ω	0	Raw
	2	i-C LP Pump, CT: LA-100P	100 Ω	0	
	3	A-B Voltage	100 Ω	0	Raw
	4	i-C HP-A Pump, CT: LA-150S	100 Ω	0	
	5	A-B Voltage	100 Ω	0	Raw
	6	i-C 3-Pump Aggregate, CT: LA-150S	100 Ω	0	
20061216-20070906	1	A-B Voltage	100 Ω	0	Raw
	2	i-C 3-Pump Aggregate, CT: LA-150S	30.1 Ω	0	
20070906-Present	1	A-C Voltage	100 Ω	0	Prep
	4	i-B 3-Pump Aggregate, CT: LA-150S	100 Ω	0	

2.4.1 Data Formats

When using the NILM, there are two basic data types that can be recorded – raw data (i.e. current and voltage) and preprocessed spectral envelopes (i.e. real and reactive power). Data is collected in one-hour snapshots. Raw voltage and current are sampled at a rate of 8,000 samples per second; preprocessed power information is sampled at 120 samples per second. Each preprocessed data file contains 8 columns. The first two are the reactive power and real power, respectively. The following 6 columns contain in-phase and quadrature spectral envelopes for the third, fifth, and seventh harmonics of the current. Typically, raw data is collected in the early stages of NILM application. Once the system is well understood and the characteristics and failure modes of individual components within the system are well defined, then it is much more efficient in terms of data collection to collect prep data. Prep data snapshots are 88% smaller than the associated raw data, and much easier and quicker to handle (Mitchell, 2007).

2.5 First Generation Diagnostics and Failure Modes

The first generation Reverse Osmosis NILM was essentially a data collection machine. Failure modes were generally discovered by correlating events in the power data with

ship's logs or information gathered from the crew. Once the typical behavior patterns were establishing it was quite easy to recognize anomalies or unusual operating profiles. In most cases the anomalies were able to be associated with a recorded event and therefore, a diagnostic was developed. LT Gregory Mitchell noted several such conditions in his 2006/2007 research. The most common abnormal conditions were unusual vibrations in the high pressure pump, clogged micron filter, membrane failure, high pressure pump starting without operator input (commonly referred to as a phantom start) and operator error. The last condition, operator error, is generally improper operation of the back pressure regulating valve or not allowing the low pressure pump to reach steady state before starting a high pressure pump. The most common of the first generation NILM diagnostic for the RO are shown in Table 2-3.

Mitchell and DeNucci were able to determine most of this information by either examining the data directly or by using simple programs to identify common events such as pump starts. This is extremely time-consuming and hence most of the data that is obtained cannot be relayed to the operators immediately. However, this front end rigor is essential to understanding the system and identifying equipment tendencies. The research contained in chapters 3, 4 and 5 would not have been possible without their efforts.

Table 2-3: First Generation NILM Common Diagnostics

Failure Mode	*Characteristics*	*Diagnostic*
Vibration in hp pump	• Noise in space • Vibration at pump	• Abnormally high oscillation in power • Large increase in magnitude 8.17 Hz
Clogged micron filter	• D/P across filter >15psig	• Excessive time for low pressure pump to reach steady state
Membrane Failure	• High Salinity • Low effluent production	• Sudden decrease in real power • Abnormal P and Q mismatch
Hp pump start without operator action (*Phantom Start*)	• Hp pump damage or failure	• Hp pump started without lp pump running
Operator Error:		
1. Improper bypass valve operation	• Damage to brine seal in pressure vessel housing	• None
2. Not allowing lp pump to reach steady state	• Damage to hp pump	• None

3 Real-Time Graphical User Interface Development

The previous NILM RO installation did not make use of real-time systems that can provide feedback to the operator. The primary focus of the 2007/2008 team was to develop a modern version of the NILM that does provide this capability and can begin to serve as a marketable device. This chapter details the development of the software needed for a real-time system monitoring the RO units.

A general real-time NILM requires several system components – a rugged, reliable data-acquisition unit, raw data preprocessing, load-classification software, and a user interface. The hardware front-end was designed and built by John Rodriguez of NEMO Metrics, LLC. His system includes sensors, signal conditioning, and data acquisition. This is usually referred to as the *NILM Box*. Output samples from the NILM box are passed to the software preprocessor, which is commonly known as prep. Preprocessed data is fed to the classifying software, which was developed by LCDR Ethan Proper (Proper, 2008). His software, which is known as *Ginzu*, identifies load ON/OFF events and attempts to classify them. Ginzu creates small data files containing power information and classification tags. These files are passed to a graphical user interface (GUI) that provides feedback to the user and performs simple diagnostics. This chapter focuses on the development of the GUI for the RO systems described in Chapter 2.

3.1 UI Development

The typical product development waterfall flows sequentially through the steps of requirements definition, concept exploration, design, testing and delivery. However, this does not always align with software development. There are several reasons for this but probably the single most important reason is that software designs often use embedded functions that are not necessarily a part of a single component or subsystem. This is commonly referred to as "object oriented". Much modern software is written using object-oriented abstractions and makes extensive use of very large software infrastructure objects (Maier & Rechtin, 2002). Because of this abstraction, most software systems cannot be neatly divided into subcategories and subunits. Therefore, it becomes increasingly difficult to map requirements and operations to specific blocks within the

construction of the system. For example, the RO UI requirements specify that the design must be easily modifiable. The requirement itself seems simple; however it is difficult to map modifiability to a specific subsystem or component within the code. Further, modification implies that there are requirements that will come about in the future, thus nullifying the design rigor that is typically done in the early stages of design. This provides a specific challenge when designing a software system.

In the case of a large software project like the RO GUI, one of the most widely used approaches is a spiral type design, where major releases can be represented as vertical jumps or rings and the evolutionary changes or minor revisions circling into them (Maier & Rechtin, 2002). Figure 3-1 is an example of a spiral development approach. This is essentially the method that was used to develop the RO UI. The advantage to this method is that it takes advantage of the software designer's ability to rapidly change and evolve the code through iterative steps rather than forcing a huge design effort in the first two phases of design.

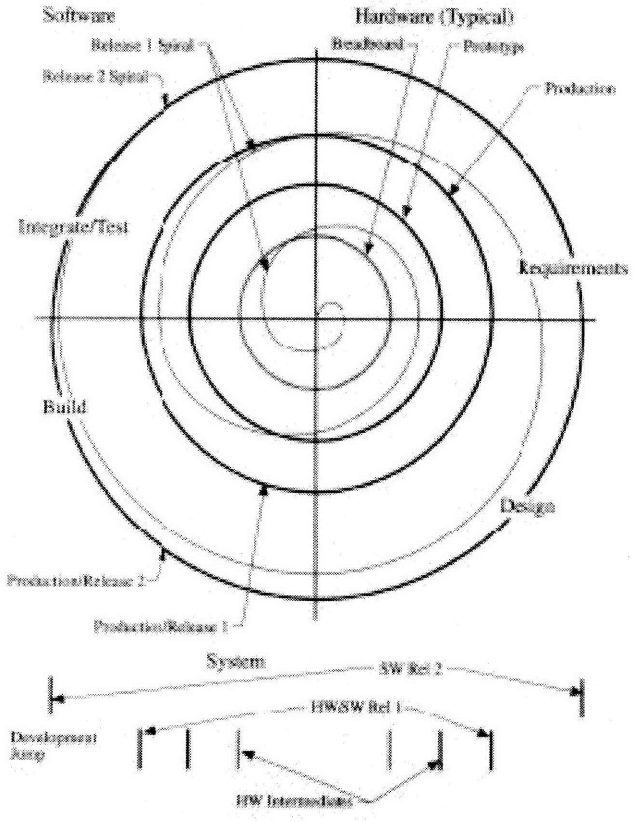


Figure 3-1: Spiral Design Approach (Maier & Rechtin, 2002)

3.2 Requirements Definition

Like any product, the RO GUI had to be developed by adequately and accurately defining the requirements or customer needs. However, the UI is not a standalone system. The UI is just one part of the overall NILM architecture. Hence, the requirements definition includes the entire NILM system. Conversations with the primary stakeholders led to the following eight customer requirements:

1. **Easily modifiable**
    a. The design must allow for easy modification by designer as well as future NILM teams
    b. Evolutionary in nature to allow for updates and future requirements
    c. Common interface so it could be used with any data output
2. **Able to operate in the engine room environment for sustained underway operations**
    a. Water proof
    b. Heat (100 F)
    c. Humid
    d. Power interruptions for drills and casualties
3. **Low cost**
4. **Reliable**
    a. Durable, able to handle the environment of the equipment
    b. Backup power source
5. **Data handling and display**
    a. Provide adequate display in location that operator will receive information in timely and useful manner
    b. Accurately display events in real time
        i. Low pressure pump start/stop
        ii. Inform operator when to steady state has been reached
            1. Time to steady state
        iii. High pressure pump start/stop
            1. Alarm indication if low pressure pump steady state not reached
        iv. Bypass valve operation
        v. Overall System operation
        vi. Indicate and alarm for known casualty conditions
            1. Membrane failure
            2. High pressure pump start without low pressure pump running (phantom start)
            3. Clogged or misaligned sea water strainer
            4. Clogged or dirty micron filters

c. Provide the operator with real time information to assist in operating equipment correctly and safely
d. Provide a supervisor only indication for monitoring operator performance
e. Provide archive for old data storage

6. **Small footprint**
   a. easily removed
   b. Small impact on engine room operations
   c. Low power consumption
7. **Maintain current system complexity level**
8. **Intuitive operation**

3.2.1 Major Decisions

The first decision made during the development of the RO UI was to choose a programming language. One of the major requirements was to ensure modifiability by ensuring that the code could be changed by future design teams. In order to ensure that this requirement was met it was decided that MATLAB could be utilized in developing the UI. MATLAB is an engineering tool that is widely used in research and has been used extensively in the early stages of NILM development. Additionally, MATLAB has a built-in UI development tool called *Guide*, which allows the designer to generate the starting functions and callback identifiers without having to hard code them. *Guide* provides the basic framework for building the UI by auto programming the creation functions for simple applications such as a pushbutton or radio button.

The hardware that interfaces with the RO UI is not part of this product development. The screen that will display the UI was selected by LT Ashley Fuller, USN. Hence the customer requirements 2 – 4 were handled by LT Fuller as part of a different product stream. Additionally, the front end software development was done by LCDR Ethan Proper, USN. He developed the initial event detector, called *Ginzu,* which provides data to the RO user interface.

3.2.2 Develop and UI Architecture

Once the RO GUI requirements were established the next phase of development was the design. In order to design a software package it is important to know exactly where the

major decision points exist. These decisions can usually be determined by examining the system interfaces and common information channels where clogs and errors are most likely to occur. To assist in determining this information for the RO GUI, an initial architecture was developed. Figure 3-2 is an illustration of the initial system architecture. Notice that no major components have been identified and that even subsystem boundaries have yet to be defined. The importance of this architecture is that it physically illustrates what is known, or already exists, to the designer and what must be developed.

The designer can use the initial architecture to identify dependencies and interfaces that will ultimately drive the design. In this case there are two external interfaces and two internal dependencies that must be resolved in order to define a final product. The external interfaces are the data received from Ginzu and the display, which interfaces with the external monitor. The internal dependencies are the data handling structure, which must be defined in order to accurately program functional logic to perform the internal operations of verify, diagnose, classify and display; and finally the actual information that will be displayed to the operator. The information that will be displayed to the operator must be known in order to determine what output must come from the diagnose function.

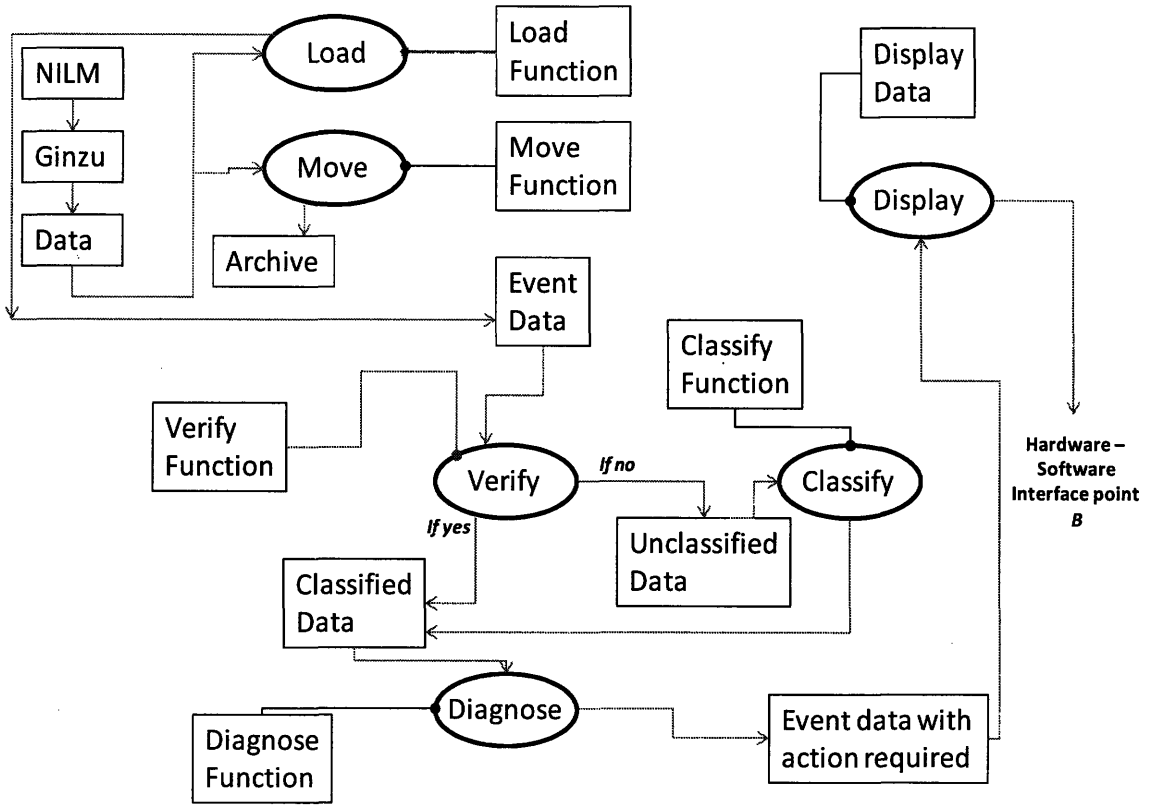


Figure 3-2: Initial UI Architecture

3.2.3 Data

The first blocks in Figure 3-2 include some that are internal to the UI and some that are external. This is important information because it is immediately apparent that the Ginzu output must be in a form that is usable to the UI. This presented the first design decision. Since Ginzu was developed concurrently with the UI the designers were able to work together to determine exactly what output would come from Ginzu. In this case the information needed by the UI is as follows:

- Event classification
- Time of event
- Real Power (P) and Reactive Power (Q) data
- Local index of event within the data file

Next the number of data points (real and reactive power) contained in each output file had to be established. Ginzu needed the files to be at least longer than the longest transient in the RO system, but small enough to not affect the processing speed. The RO UI needed the files to be long enough to provide enough data for all diagnostics but short

enough so that the lag between the event and displaying the event was not excessive. In the end it was the processing speed of Ginzu that drove the length of the data files. This was due to the exponential relationship between the size of the window and the processing time for several applications within Ginzu. Specifically, the processing time required for the primary filtering and the matrix inversion algorithms is equal to the number of data points squared. The output from Ginzu contains 1200 points of P and Q data, actual time of the event, local index of the event, global index of the event and the classification of the event in a single column of text data. These files are definable as "*.evt" files or event files.

An additional architectural problem is that data must be used in real-time processing and simultaneously archived for future use. The second problem can be solved easily by simply moving the event file in its entirety to an archive directory once the useful data has been obtained. The archive itself will be discussed later in this chapter. The first problem is more difficult because it requires one to consider what to do with data once it has cycled through the software's main process. Following the information as it streams through Fig. 3-2 shows that data can be discarded from stored variables only after key state information has been provided to the operator's display screen. This incredibly useful information guides the selection of the data structures used to handle event information within the GUI. From a programming standpoint the local data obtained from within the event file can be handled as intermediate or temporary variables as long as the verify, diagnose, classify and display functions are called by one single function when they are needed. However the state of the system must be maintained globally so that the system's operating profile is always known.

Figure 3-3 is an updated architecture of the data flow. This type of data flow is optimal to the overall design because it is very modular in nature, which allows the designer to change and evolve the functions without affecting the fundamental structure of the code.

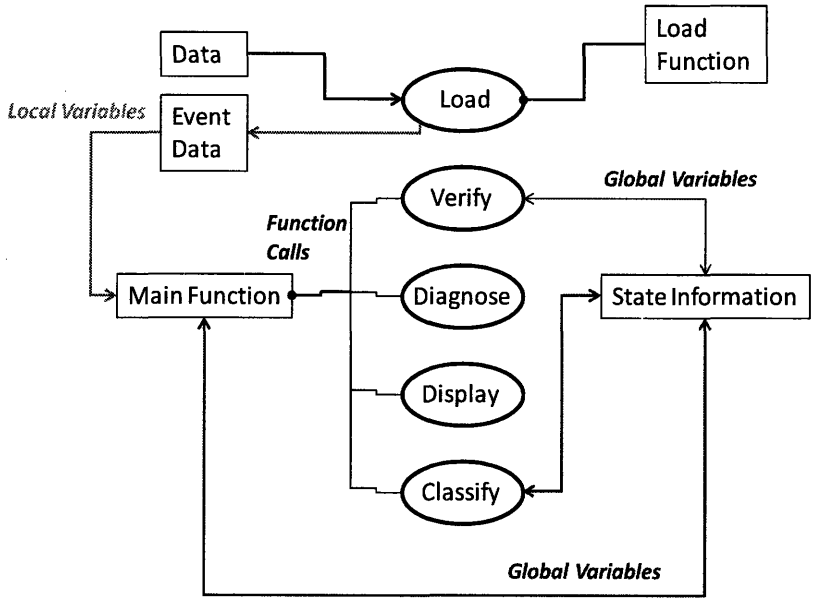


Figure 3-3: Data Flow Architecture

3.2.4 Global Variables

The state of the system has to be known at all times in order to perform certain diagnostics. One of the customer requirements was to be able to detect abnormal operating conditions, such as starting the high pressure pump without allowing the low pressure pump to reach equilibrium, and casualty situations such as phantom starts. The only way to provide this information accurately to the operator is to provide the system with accurate state information. This determination in and of itself becomes the system's first real diagnostic. Figure 3-4 is a representation the state based programming logic that was used to develop the UI. The red sequences represent states that will eventually result in total component failure, while the yellow indicate abnormal system performance or improper operator action, but will not result in system damage. This information will be stored as global variables and passed or updated from any place in the program.

The state array is composed of four component states: [low pressure pump, steady state, high pressure pump/s, bypass valve/s]. For the low pressure pump state a zero indicates that the pump is off and a one indicates that the pump is running. For the steady-state a zero indicates that the low pressure pump has not reached a steady condition (not safe to start a high pressure pump), and a one indicates that the low

pressure pump has reached a steady condition (safe to start a high pressure pump). For the high pressure pump a zero indicates that both pumps are off, a one indicates that one pump is running and a two indicates that both pumps are operating. Finally, for the bypass valve a zero indicates that both bypass valves are open, a one indicates that one bypass valve is shut and a two indicates that both bypass valves are shut.

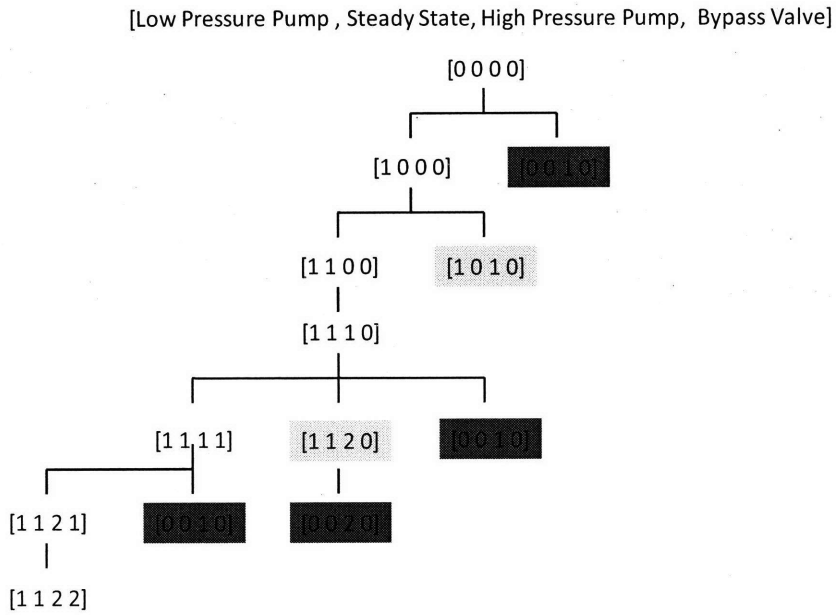


Figure 3-4: System - State Sequence

In addition to the state based information there is other information that must be maintained as global variables in order to provide certain pieces of information to the diagnostic sections of the program. An example is the time-to-steady-state which is a metric defined in (Mitchell, 2007) for determining the state of the micron filters in the RO system. Figure 3-5 illustrates how information is shared within the programming construct.

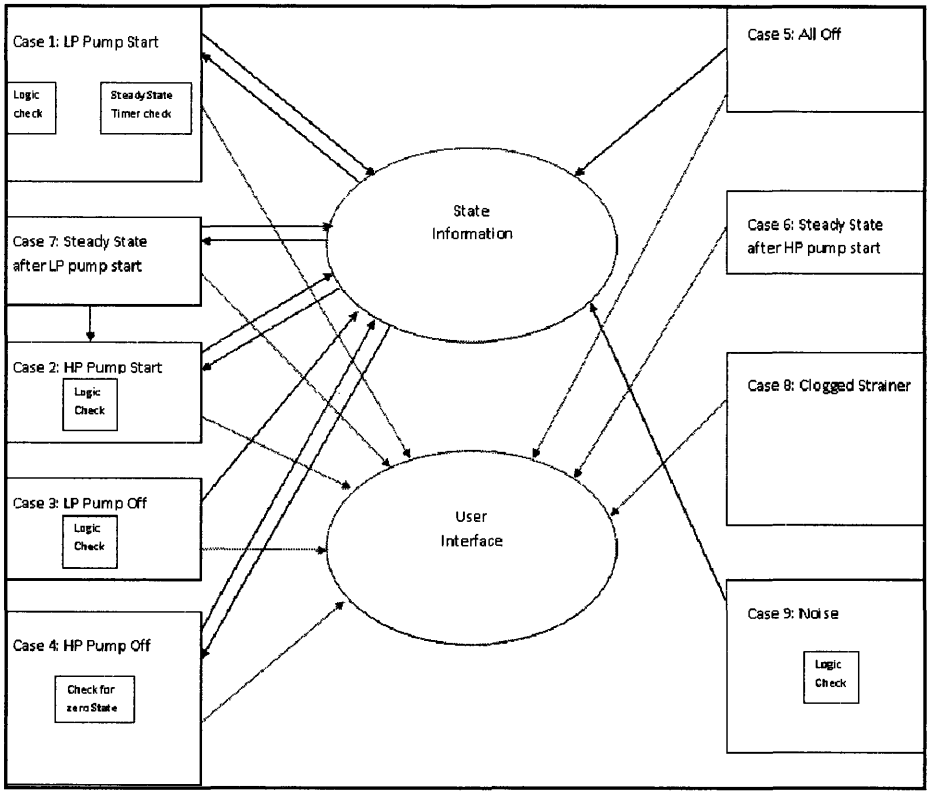


Figure 3-5: State Information Flow

3.2.5 Display

The next step was to determine what information to display to the operator. After careful consideration and discussion with the stakeholders two basic display concepts evolved. The first was an information rich environment, which would provide the operator with specific system information. This display would include an interactive system diagram as well as an alert panel that clearly and distinctly displayed any system abnormalities. The second concept was a visual environment that provided the operator with immediate system health verification but was uncluttered with expansive system information. This concept would include a panel of blocks that would be all white when the system was operating correctly and red when something was abnormal. Figure 3-6 shows an example of the two competing concepts. To determine which of the two concepts would be used the requirements that specifically pertain to the display option were mapped to the designs. Each attribute was assigned a value from 0 to 1 based on how well each requirement was met. Table 3-1 is the concept exploration table where

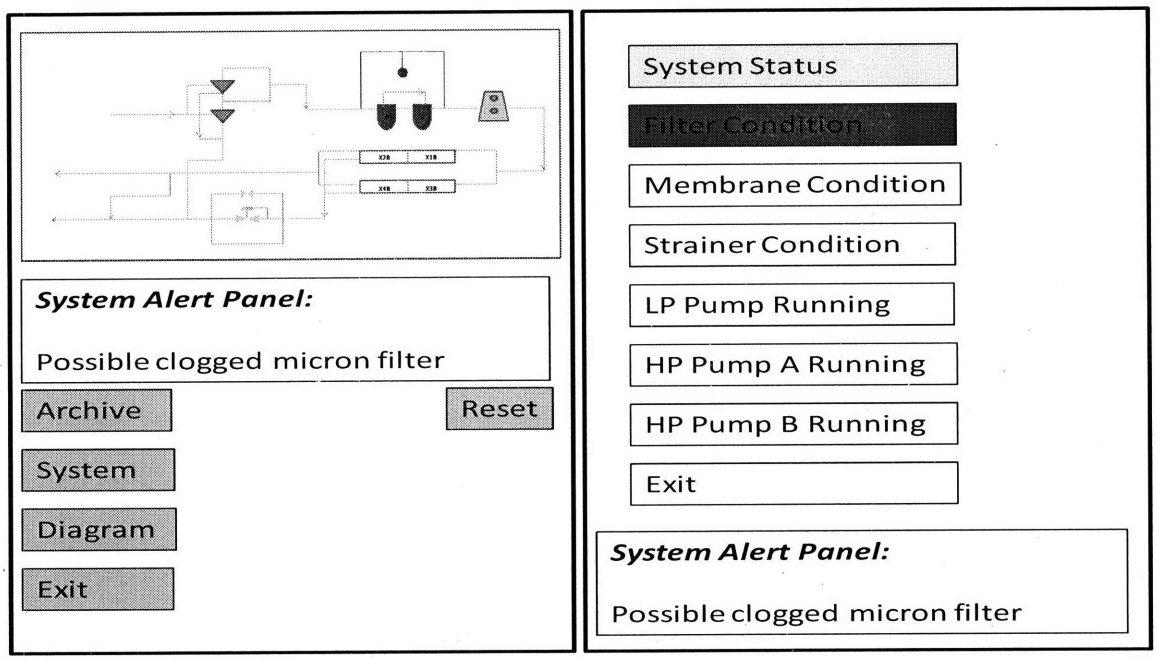


Figure 3-6: Two Competing Display Concepts

concept A was the system diagram approach and concept B was the first glance visual approach. Each concept was given a score of 0 – 1 based on how well it met the customer requirement. Ultimately concept A scored higher and was selected as the display option. This is primarily because proper operation of the RO is solely dependent on the operator taking the correct steps in the right sequence and timing. Thus, providing the operator with real time visual verification of each step proved to be invaluable.

Table 3-1: Concept Exploration Table

Design Requirements	*Concept A*	*Concept B*
Information received by operator	0.8	1
lp pump start / stop	1	1
hp pump start / stop	1	1
lp ss reached	1	1
bp valve indication	1	1
overall system operation indication	0.8	0.2
alarm conditions	1	1
provide operator performance indication	1	0.2
Real time assistance to operator	1	0.5
	8.6	6.9

3.2.6 Internal Function Programming

Now that the key aspects of the overall design were defined the internal UI functions could be defined. The first step was extracting the data from the event file and passing it to the main program. To do this it was essential to understand exactly how MATLAB works within the UI. When the UI is started the program automatically performs tasks that are described inside the opening function. This includes opening the UI, any physical pictures or diagrams that are present in the base UI layout and establishing the initial handles structure. The opening function must also include any global variables that will be used throughout the program. The handles structure contains all the information for each object that is contained with the UI .fig file (Uliana, 2007). The opening function code is described below:

```
% Begin initialization code - DO NOT EDIT
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
'gui_Singleton',  gui_Singleton, ...
'gui_OpeningFcn', @RO_Diagram_OpeningFcn, ...
'gui_OutputFcn',  @RO_Diagram_OutputFcn, ...
'gui_LayoutFcn',  [] , ...
'gui_Callback',   []);
if nargin && ischar(varargin{1})
gui_State.gui_Callback = str2func(varargin{1});
end

if nargout
[varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
gui_mainfcn(gui_State, varargin{:});
end
% End initialization code - DO NOT EDIT

% --- Executes just before RO_Diagram is made visible.
function RO_Diagram_OpeningFcn(hObject, eventdata, handles, varargin)
% Initialize global variables:
global HP;
global LP_StartT;
global ss_7;
global LP;
HP=0;
LP_StartT=0;
ss_7=0;
LP=0;

% Open initial figure (this is the system diagram)
ax4=axes('Position',[0.1  .4  .8  .5]);
[x,map] = imread('Final Gui RO','png');
image(x)
set(gca,'visible','off')
```

Once the UI is opened it will remain unchanged until a function is called to perform a task. This was the first real problem that was encountered by the designer because Ginzu passes an event file into the UI directory but something has to tell the UI to retrieve the data. Ultimately this problem was solved by inserting a timer inside the opening function, which once fired would call a load function to index and read any event files contained inside the working directory. The timer function is a modification of the 'My Timer Function' created by LCDR Ethan Proper, who created the timer for a similar system. Within the timer function the designer can specify how often to look in the directory for a new event file. In order to ensure a small time delay exists between receiving the information and displaying, the timer should be set as low as possible. However, it cannot be set so low that it interferes with the internal functions. For the RO UI the timer was set to .25 seconds. The timer itself must be contained inside the opening function. The timer function code is described below:

```
t=timerfind;delete(t);
% Choose default command line output for RO_Diagram
handles.output = hObject;
handles.evt_index=1;
% Update handles structure
guidata(hObject, handles);

% Start the steady_state timer
handles.steady_state = timer('period',.25);
set(handles.steady_state,'ExecutionMode','fixedrate','StartDelay',2,'BusyMode','drop');
set(handles.steady_state,'timerfcn',{@MyTimerFcn_1, hObject});
start(handles.steady_state);

% Update
guidata(hObject, handles);
```

There are two important pieces to note here. First, at the end of the timer code is the line 'guidata(hObject, handles);' this is to ensure that that the handles data described here is saved each time it fires. Otherwise none of the handles structure data can be passed throughout the body of the program. Second, the timer only calls a function. By itself it does nothing. In this case it calls the function 'MyTimerFcn_1', which is described below:

```
%==========================================================
% --- Timer Callback Function
%==========================================================
function MyTimerFcn_1(steady_state_object,event,hObject)
% increment the number of counts...
```

```
handles = guidata(hObject);
handles.file_names=[];
dir_path=pwd;
dir_struct = dir(strcat(dir_path,'/*.evt'));
[sorted_names,sorted_index] = sortrows({dir_struct.name}');
handles.file_names = sorted_names;
handles.is_dir = [dir_struct.isdir];
handles.sorted_index = sorted_index;
%copyfile('*.evt','event/','f'); %This works.
num_files=length(handles.file_names);

for i=1:num_files
   auto_load(hObject,[],handles,pwd,handles.file_names(i));
end
guidata(hObject, handles);
```

This function is called every 0.25 seconds by the timer. Again it is important to note what exactly is being done here. The 'MyTimerFcn' looks in the current working directory for any '*.evt' files, indexes them sequentially as they appear in the directory and then passes them in order to the main program, which is called 'auto_load.' The 'auto_load' function is described later in this section. Finally the handles structure is preserved by the 'guidata' command at the end of the function.

The first data handling process has been established. Next the design must extract the useful information from the '*.evt' file and pass it to the functions structure described in figure 3-3. Initially the basic structure was to be totally modular. Meaning that each function was self contained and would be called by the 'auto_load' function when it was needed. However, this proved to be unnecessary and actually complicated the overall design. This is primarily because each function contained within the body of the program had to be applied to each classification of event differently. For example, if Ginzu passed the classification of low pressure pump start, then the verification function had to look for parameters that specifically identified the event as a low pressure pump. Essentially every event had a different set of identifying parameters and would require a different verification function. Likewise the 'auto_load' function would first have to identify the classification for each event and call the appropriate verification function. Then if the verification logic did not agree with Ginzu's classification it would have to call the classification command from within the verification function. This would require passing local variables to several different places within the overall program and require an excessive amount of processing time. While the idea of modular functions would be preserved, the benefit of a simple program flow would be lost.

Instead a switching algorithm, which runs within the body of the main program, was developed. In order for the switching algorithm to work efficiently a numeric value was assigned to each classification within Ginzu and passed to the UI. Table 3-2 is a list of the numbers assigned to each event.

Table 3-2: Number assigned to event classification

Event Classification	*Number Assigned*
lp start	1
hp start	2
lp stop	3
hp stop	4
all off	5
bypass valve open	6
lp steady state reached	7
cycling load transient	8
noise	9

The 'auto_load' function extracts the number and assigns it to a local variable. The function also contains the switching algorithm, which will only perform the specific tasks associated with the event. Some modularity is still maintained because the case statements themselves act as individual sub functions within the main program logic. Additionally, the handles structure for each object that is associated with each event is contained outside the 'auto_load' function and is only called when it is needed by the case sub routine. The 'auto_load' was developed by LCDR Ethan Proper and was modified to fit the application here.

```
function auto_load(hObject, eventdata, handles,pathname,filename)
% Must call the global variables
global HP;
global LP_StartT;
global ss_7;
global LP;
pathname=strcat(pathname,'/');
longfilename=char(strcat(pathname,filename));
fid=fopen(longfilename,'r');
[path,name,ext,ver] = fileparts(longfilename);

 % Ethan had two different conventions for how he was time stamping the files
if (ext=='.evt')
   event_t = fgetl(fid);
   try
```

```
   ev_time_num=datenum(event_t,'yyyymmdd-HH:MM:SS');
catch
   ev_time_num=datenum(event_t(5:24),'mmm dd HH:MM:SS yyyy');
end

% Extracting the data needed for the UI
O_file = fgetl(fid);
glob_index = str2num(fgetl(fid));
local_index = str2num(fgetl(fid));
event_class = fgetl(fid);
class=str2num(fgetl(fid));
window_size = str2num(fgetl(fid));
P = zeros(window_size,1);

%Creating P and Q array for display
for i=1:window_size
   P(i)=str2num(fgetl(fid));
end

for i=1:window_size
   Q(i)=str2num(fgetl(fid));
end

ev_min=fix(glob_index*60/432000);
ev_sec=fix((glob_index-ev_min*120*60)/120);
event_time=(num2str(fix(glob_index*60/432000*100)/100));
event_time_min=num2str(ev_min);
event_time_sec=num2str(ev_sec);

done=fclose('all');
movefile(longfilename,'archive/');
```

Note that the last line in the 'auto_load' function is 'movefile.' This accomplishes the move function by physically moving the '*.evt' file to the archive once the useful information has been extracted. Again modularity is partially preserved because the action is accomplished outside of the local data stream, which is passed to the case statement logic.

The final step in developing the UI was to develop the case statement logic routine. For the purposes of discussion each case statement will be treated as a separate sub routine contained within the larger body of the program. The first case is the low pressure pump start. This is by far the most complex routine because it encompasses several customer requirements. The first part of the case one logic is to verify the classification passed by Ginzu. This presents two problems. Number one is that Ginzu looks for a change of mean and then classifies the event based on the state information contained within Ginzu. If Ginzu shows a state of [0 0] (lp pump off, high pressure pump off] and a change of mean occurs then it automatically classifies the event as a '1' (low pressure pump start). Because the RC7000 Plus system has a known problem with phantom high pressure pump starts the UI must correctly identify the event in order to

provide the operator with critical system information. Additionally, Ginzu uses a sliding window approach to identify an event. If an event occurs too quickly after the previous event Ginzu will miss it. Therefore, the UI must sequentially step through the window to ensure that only one event has occurred.

In order to verify that the event being called a low pressure pump start is correct the program compares the peak power to a known value. If the peak power is too large then the program identifies the event as a high pressure pump. If the peak power is too small then the event is ignored due to noise. This part of the program should be improved in the next iteration because it uses information that is specific to this RO unit and may not work for every unit. For example, the peak power always occurs at (local index + 3) and the peak power threshold value is specific to this RO system. Next, the case one routine has to find the time to steady state (indicator of micron filter condition) and pass this information to a .txt file contained in the working directory.

To accomplish this, the case one logic steps through the power data in increments of one, looking for a threshold increase in power. Once the threshold value occurs the program then begins stepping through the data in increments of 20, comparing the mean of each step to the previous step. Once the two steps have corresponding mean values within 1 % of each other then the system is determined to be at steady state and the time is recorded. Finally, the case one statement has to pass pertinent indication information to the UI display. This is done using the handles structure for each object that must be updated on the display. The case one code is described below:

```
switch(class)
      case 1
% set threshold value for p to start looking for steady state
if P(local_index+3)-P(local_index-3) < 20 & P(local_index+3)-P(local_index-3) > 10 %This is set prevent noise from being
classified as pump start or hp start from being classified as lp start
for n = 1:length(P)
   if P(n) < 5  % This is arbitrarily set for now
   else
      event=1;
      break
   end
end

if event ==1
   for m = 1:length(P)-20-n
      avg(m) = [mean(P((m+n-1):(m+n+20-1)))];
   end
   for j = 1:length(avg)
      min_P = min(avg(j:j+20));
      max_P = max(avg(j:j+20));
```

```
% set threshold for what you are defining as steady state

      if (avg(j)*1.01>max_P) & (avg(j)*.99<min_P) | j+20 == length(avg)
         break
      end
         ss_index = j+n-1;
         time = (ss_index-n)/120;
   end
end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
fid = fopen('times.txt','a+');
fprintf(fid,'%s %30.8f\n',O_file,time);
status = fclose(fid);

         set (handles.LPpump,'String','ON');
         set (handles.LPpump,'BackgroundColor','green');
         LP=1;
         LPstartT = time;
            if time >2
               set (handles.system,'String','Check micron filter condition')
               set (handles.system,'BackgroundColor','red')
            end

         set (handles.steady,'String',LPstartT);

         if P(local_index+3)-P(local_index-3) > 50
            set (handles.HPpumpA,'String','ON');
            set (handles.HPpumpA,'BackgroundColor','green');
            HP=HP+1;
            set (handles.system,'String','A high pressure pump start has been detected without the low pressure pump
running');
            set (handles.system,'BackgroundColor','red');
         end
end
```

The threshold value that is set for the steady state routine is system specific and should be revisited to increase the overall system robustness. Figure 3-7 below is a screen capture of the UI with a low pressure pump running.

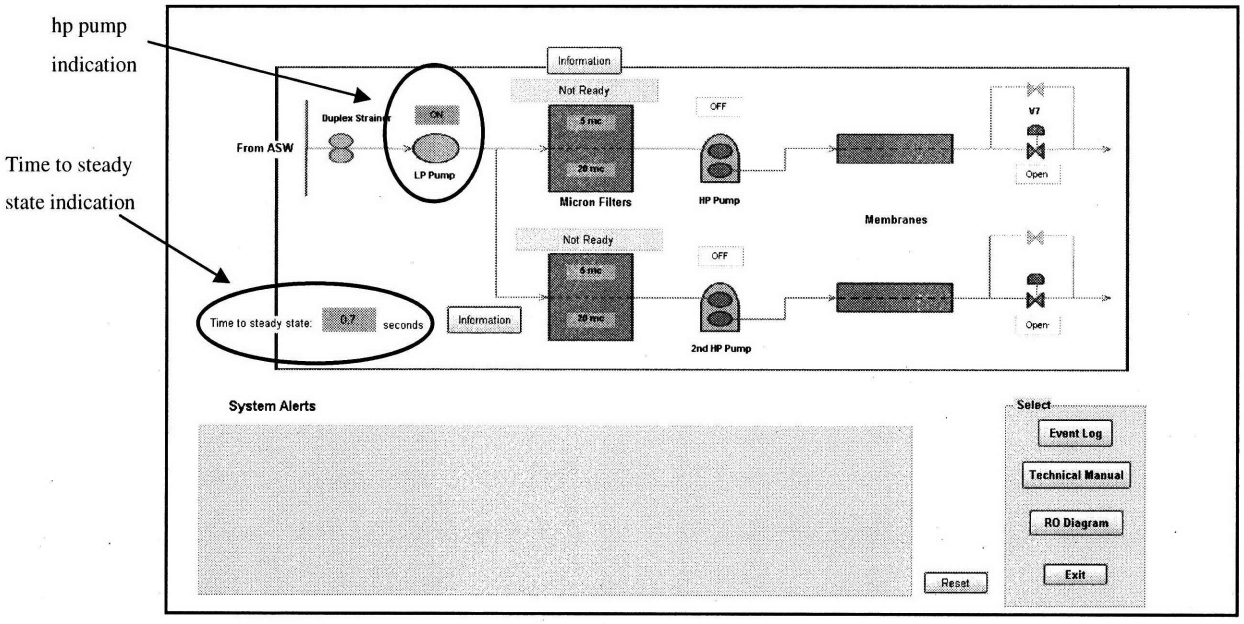


Figure 3-7: RO UI with LP pump running

Next is the case seven (low pressure pump steady state) logic routine. This is done here because sequentially it is the next event that should occur. Note that in the case one statement the low pressure pump index was changed to 1. This is passed as a global variable and is part of the state information. Likewise the case seven logic indexes the global variable 'ss_7' to 1. The case seven statement indicates that the system has reached steady state and that it is safe for the operator to start a high pressure pump. The case seven code is:

```
case 7
        if LP==1
        ss_7=1;
        set (handles.pressureA,'String','Ready to Start');
        set (handles.pressureA,'BackgroundColor','green');
        end
```

Next the system expects to see a high pressure pump start, or case two. If a high pressure pump is started without the state being equal to [1 1 0 0], then an alarm signal is passed to the display to notify the operator of what condition exists.

```
case 2
      if (HP==0)
         set (handles.HPpumpA,'String','ON');
         set (handles.HPpumpA,'BackgroundColor','green');
      else
```

```
    set (handles.HPpumpB,'String','ON');
    set (handles.HPpumpB,'BackgroundColor','green');
  end

  HP=HP+1

  if (ss_7==0)
    set (handles.system,'String','High pressure pump was started before steady state was completely reached');
    set (handles.system,'BackgroundColor','red');
  end
```

The rest of the case statement routine is very similar in structure to the previous cases. The statements that are state sensitive are verified to be in the condition expected and appropriate information is passed to the display.

```
case 3
    set (handles.LPpump,'String','OFF');
    set (handles.LPpump,'BackgroundColor','white');
    LP=0;
    ss_7=0;
    set (handles.pressureA,'String','Not Ready');
    set (handles.pressureA,'BackgroundColor','red');

  case 4
    if (HP==1)
      set (handles.HPpumpA,'String','OFF');
      set (handles.HPpumpA,'BackgroundColor','white');
      set (handles.HPpumpB,'String','OFF');
      set (handles.HPpumpB,'BackgroundColor','white');
      set (handles.bypassA,'String','Open');
      set (handles.bypassA,'BackgroundColor','white');
      set (handles.bypassB,'String','Open');
      set (handles.bypassB,'BackgroundColor','white');
    else
      set (handles.HPpumpB,'String','OFF');
      set (handles.HPpumpB,'BackgroundColor','white');
      set (handles.bypassB,'String','Open');
      set (handles.bypassB,'BackgroundColor','white');
    end
    HP=HP-1
    if (P(1199)<.2)
      HP=0;
      set (handles.LPpump,'String','OFF');
      set (handles.LPpump,'BackgroundColor','white');
      set (handles.HPpumpA,'String','OFF');
      set (handles.HPpumpA,'BackgroundColor','white');
      set (handles.HPpumpB,'String','OFF');
      set (handles.HPpumpB,'BackgroundColor','white');
      set (handles.bypassA,'String','Open');
      set (handles.bypassA,'BackgroundColor','white');
      set (handles.bypassB,'String','Open');
      set (handles.bypassB,'BackgroundColor','white');
      set (handles.pressureA,'String','Not Ready');
      set (handles.pressureA,'BackgroundColor','red');
    end

  case 5
    HP=0;
    ss_7=0;
    LP=0;
    set (handles.LPpump,'String','OFF');
    set (handles.LPpump,'BackgroundColor','white');
    set (handles.HPpumpA,'String','OFF');
```

```
   set (handles.HPpumpA,'BackgroundColor','white');
   set (handles.HPpumpB,'String','OFF');
   set (handles.HPpumpB,'BackgroundColor','white');
   set (handles.bypassA,'String','Open');
   set (handles.bypassA,'BackgroundColor','white');
   set (handles.bypassB,'String','Open');
   set (handles.bypassB,'BackgroundColor','white');
   set (handles.pressureA,'String','Not Ready');
   set (handles.pressureA,'BackgroundColor','red');

case 6
   if HP<=1
      set (handles.bypassA,'String','Shut');
      set (handles.bypassA,'BackgroundColor','green');
   else
      set (handles.bypassB,'String','Shut');
      set (handles.bypassB,'BackgroundColor','green');
   end

case 8
   set (handles.system,'String','Possible clogged or misaligned strainer');
   set (handles.system,'BackgroundColor','red');

case 9
   if (Q(1199)<.2)
      HP=0;
      set (handles.LPpump,'String','OFF');
      set (handles.LPpump,'BackgroundColor','white');
      set (handles.HPpumpA,'String','OFF');
      set (handles.HPpumpA,'BackgroundColor','white');
      set (handles.HPpumpB,'String','OFF');
      set (handles.HPpumpB,'BackgroundColor','white');
   end
```

3.2.7 RO Graphical User Interface – Beta Version

The final beta version of the UI is a culmination of the requirements that were initially established as well as some observations made during the initial testing phase. The UI includes a complete interactive system diagram that indicates all the information that was specified in the requirements definition phase. It also incorporates two information buttons that call separate user interfaces. These were added to help aid the end user in understanding the information that is being presented on the main display.

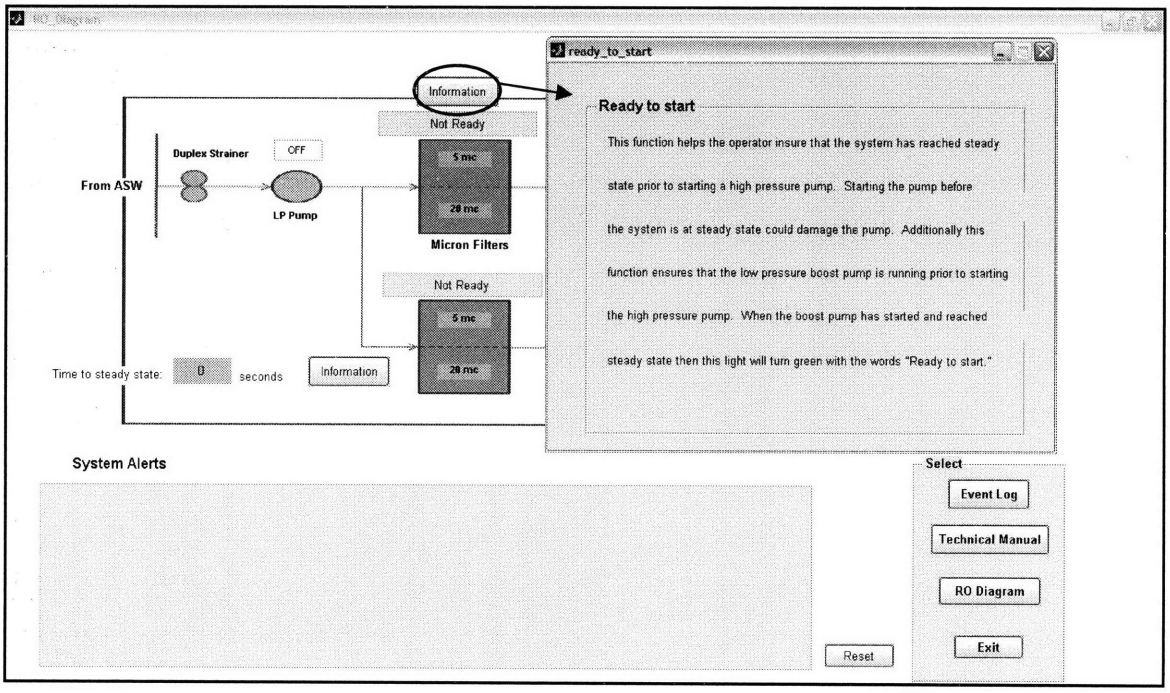


Figure 3-8: Ready To Start UI

The physical display selected by LT Ashley Fuller was an IBM Tablet PC. Since the tablet PC is totally self-contained, it provided the opportunity for latent benefits to evolve out of the overall design. The tablet PC can be removed from its holder and physically carried around the engineering spaces. Because of this ability, a link to the RC7000 Plus Technical Manual was added to the UI. Additionally, a quick reference system diagram was added. Both of these provide excellent opportunities for on-site training of new watch standers, as well as provide technical guidance without having to leave the space or carry and store large books or manuals.

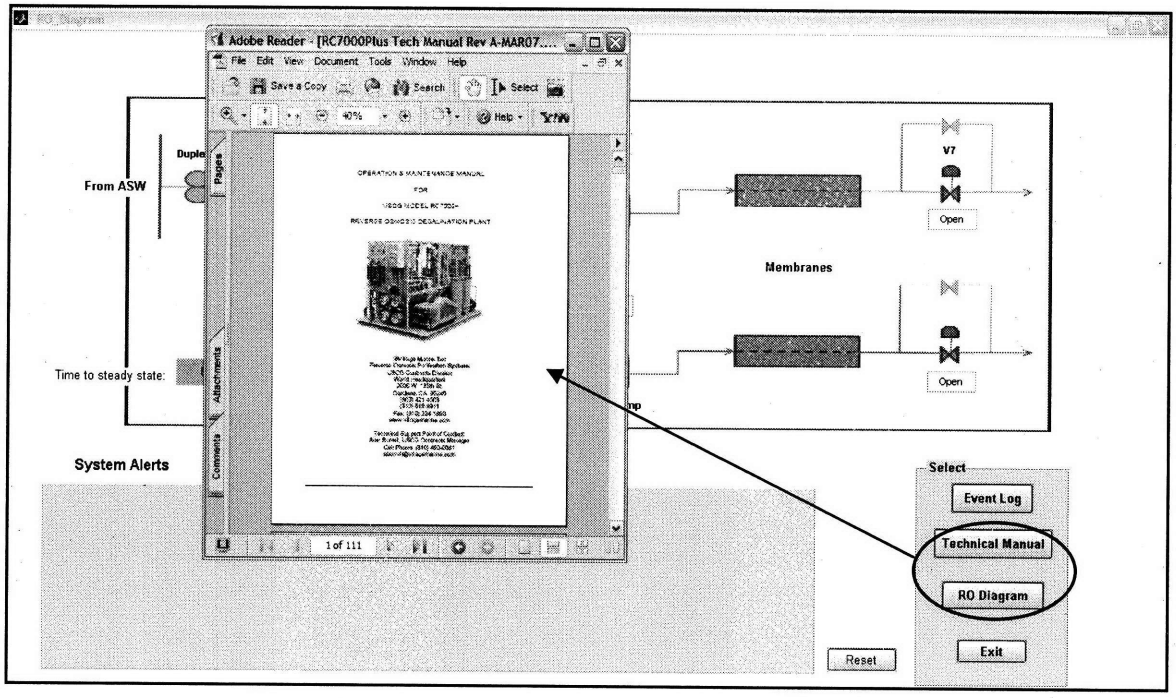


Figure 3-9: RC7000 Plus Technical Manual Link

4 Field Test on USCGC ESCANABA – Winter 2008

In January 2008 the first RO UI was installed on board the USCGC ESCANABA. This installation served as the first operational test for the user interface and the initial diagnostic package. This chapter outlines the specifics of the installation, including the NILM setup, and findings.

4.1 Installation Details

The UI was placed in a position that put it in the direct line of vision of an operator starting the RO unit, from the B side. This location was chosen because the ship's crew normally starts the B side first. The NILM sensor box was placed outboard on the starboard side, just aft of the low pressure air system. The NILM box is located well out of the way, but easily accessible in the event of a casualty requiring the ship's crew to access the panel. The NILM voltage was attached to the A and C channels, while the CT was attached to the B channel. Figure 4-1 is a block diagram of the NILM setup used on USCGC ESCANABA. Figure 4-2 contains several pictures of the install onboard the USCGC ESCANABA and Figure 2-2 provides specifics on how the NILM channels were utilized.

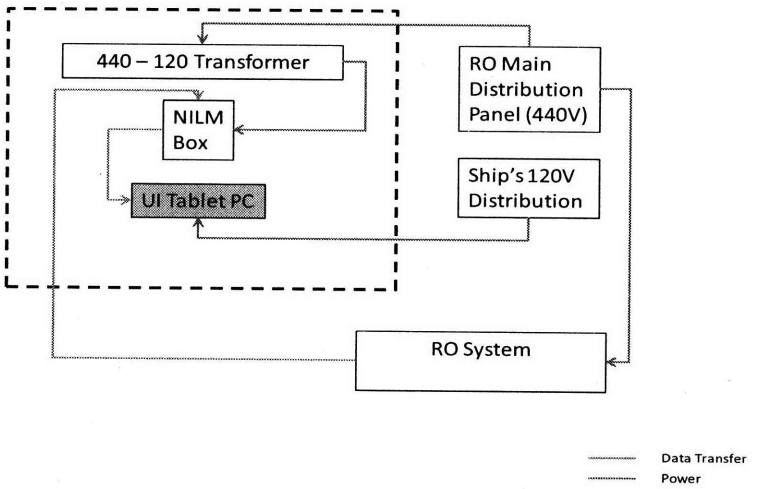


Figure 4-1: NILM Block Diagram

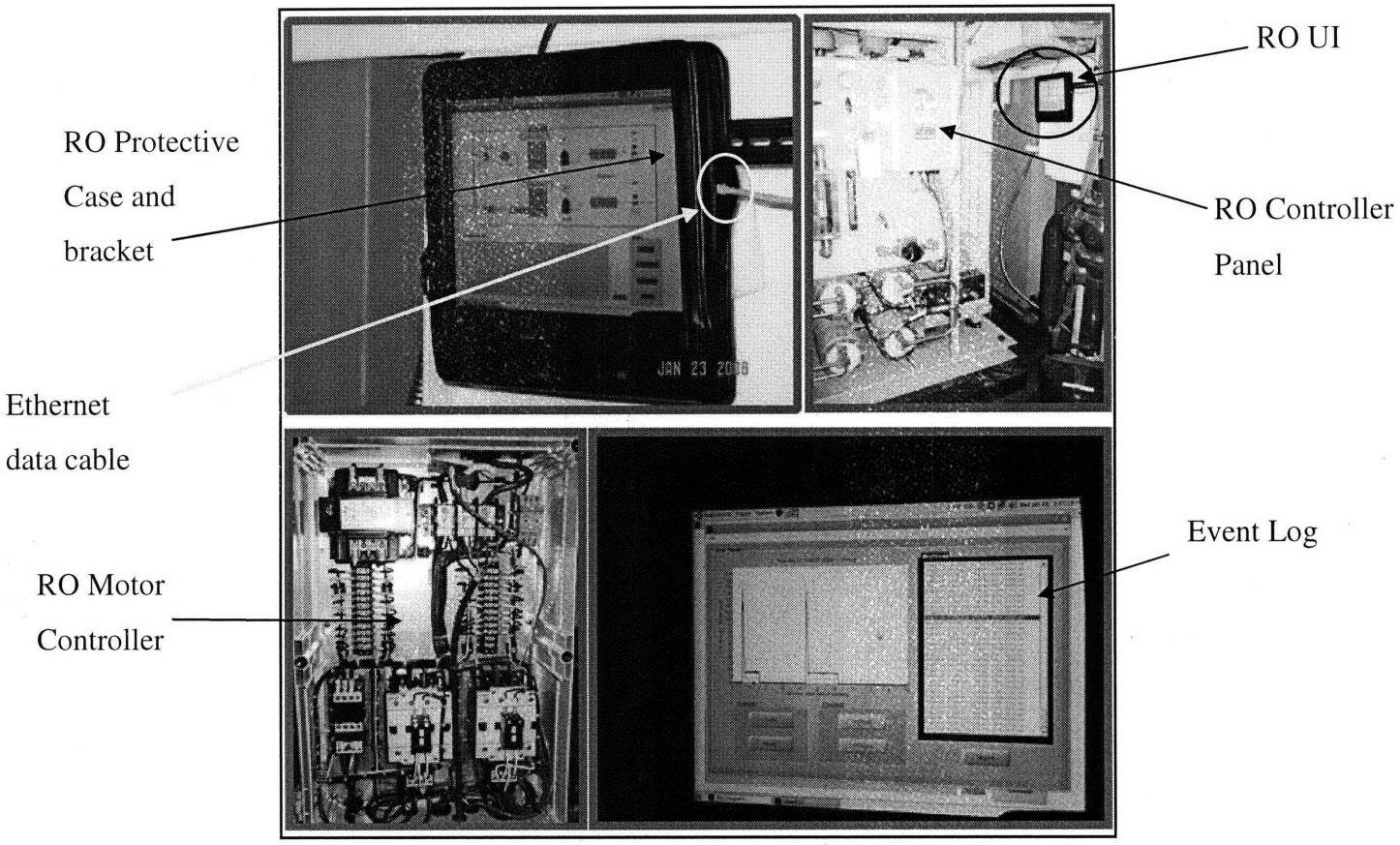


Figure 4-2: USCGC ESCANABA RO Install

4.2 Findings: Winter Cruise Data

Initial testing was performed during ESCANABA's winter cruise. The ship was underway on January 23, 2008 and returned to port on March 23, 2008. The UI itself operated and indicated correctly throughout the patrol. Examination of the data revealed that the NILM was able to detect a number of important issues, including problems caused by operator errors and those caused by system malfunctions. Additionally, these early tests revealed some flaws in both the classification and UI software. This section details both sets of issues.

4.2.1 Detection of Operator Errors

Several key deviations from standard procedure were detected using data from the winter 2008 patrol. The first of these concerned the improper operation of the system. The

proper startup and shut down procedures are outlined in Appendix B. These procedures were established in order to prevent unnecessary damage or wear to the pumps and membranes. Steps 6 and 7 of the start-up procedure are designed to ensure that the system has reached a steady-state condition in which it is safe to start the high-pressure pump. Experience has shown that it takes 1 to 2 seconds for the system to actually reach these conditions and that it takes an operator at least 3 seconds to visually verify this using the procedures outline in Steps 6 and 7 of the start-up process. Figure 4-3 shows the power drawn during a proper start-up. Note that approximately 6 seconds lapse after the low pressure pump is started before starting the high pressure pump.

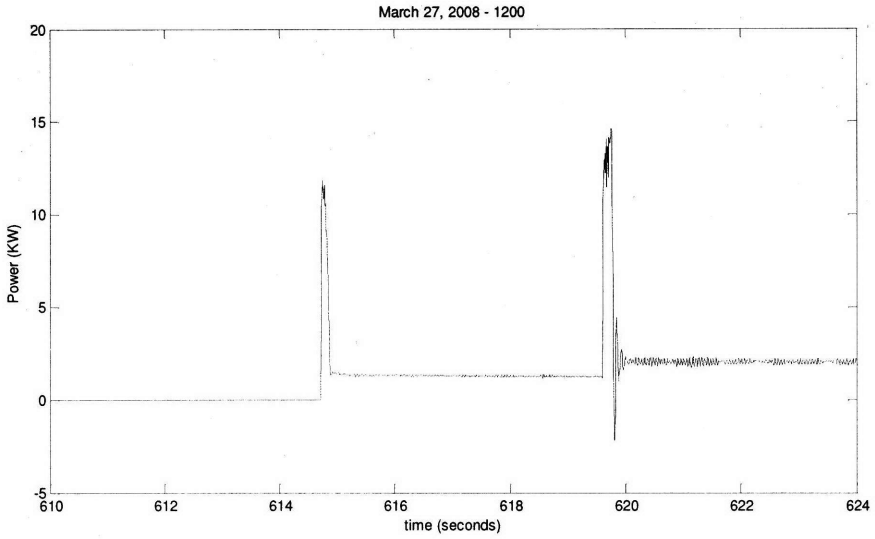


Figure 4-3: Proper operation of lp and hp pumps

Figure 4-4 shows an improper start sequence detected by the NILM on the 2nd of February. In this figure it is clear that the operator neither waited for steady state to be reached nor verified conditions prior to starting the high pressure pump. The low-pressure pump and the high-pressure pump were started within 0.2 seconds of one another. This could introduce air into the high pressure pump cavity causing damaging cavitation. Additionally, by starting the high pressure pump too early it does not allow the system to reach sufficient pump head thereby exacerbating the cavitation problem. The high pressure pump pistons are ceramic and can be damaged easily so care should always be taken to operate the system correctly.

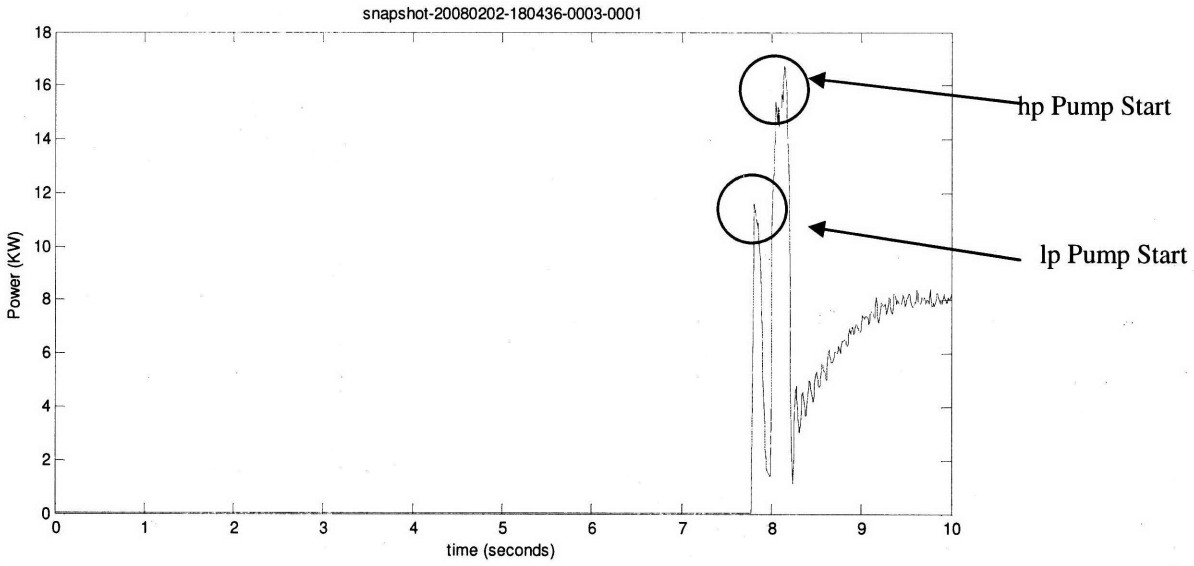


Figure 4-4: Start Sequence from Feb 2, 2008. Note the short time between the start of the LP pump and the start of the HP pump.

Another operator-related issue detected by the NILM relates to the adjustment of the backpressure regulator bypass valve during the start-up process. Once the high pressure pump is running, the operator is instructed to *slowly* shut this valve. This brings the system to its normal operating pressure of 800-1000 psig. Figure 4-5 shows a good bypass valve adjustment as determined by the Machinery Space Supervisor MKC Scott T. Galvin. As the valve is closed the real power slowly increases. When adjusted properly, the valve closing should last for approximately 7 seconds. This has been verified using both NILM data and crew member input.

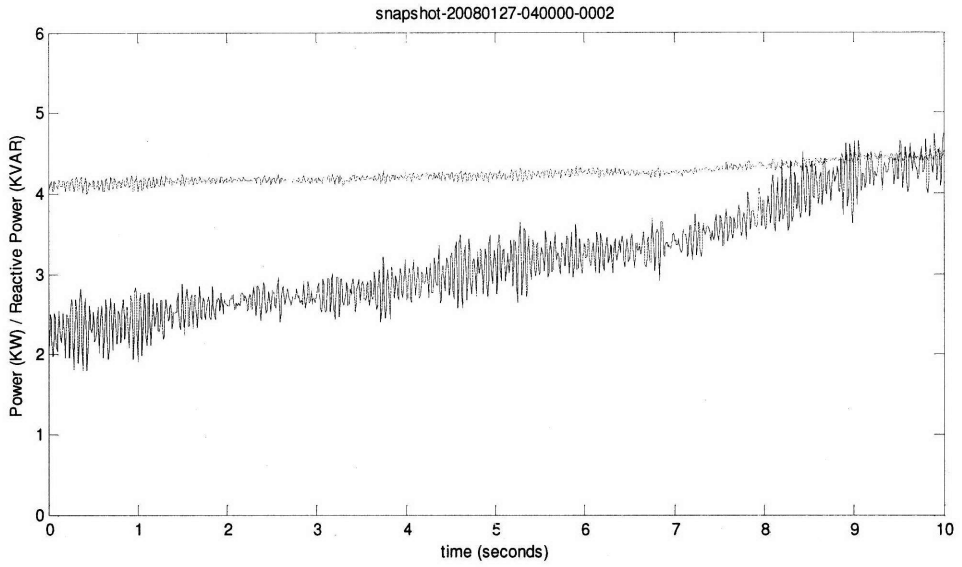


Figure 4-5: Good bypass valve adjustment. The lower trace is the real power; the upper trace is the reactive power

Figure 4-6 shows an instance in which the bypass valve was closed too quickly. In this case the valve was adjusted from fully open to fully shut in about one second. This means that system pressure increased from approximately 60 psig to 1000 psig in one second. Such an extreme change over such a short time could result in damage to the membrane and membrane housing. To understand the possible problems, consider Fig. 4-7, which shows that the membrane housing contains a brine seal on the feed end. This seal prevents the effluent from mixing with the seawater. If the pressure is increased at too high a rate, the seal could be damaged and allow contaminated water to leak into the effluent flow path. Additionally, this sudden increase in pressure could damage the membrane itself. If either problem occurs, the membrane housing must be disassembled in order to make repairs. The disassembly and assembly processes are difficult and time-consuming. The installation of the membrane into the vessel requires a combination of C-clamps to be alternatively tightened until the membrane is fully inserted. The RO UI allows the supervisor to examine every system start-up and shut down to ensure that the system is operated correctly.

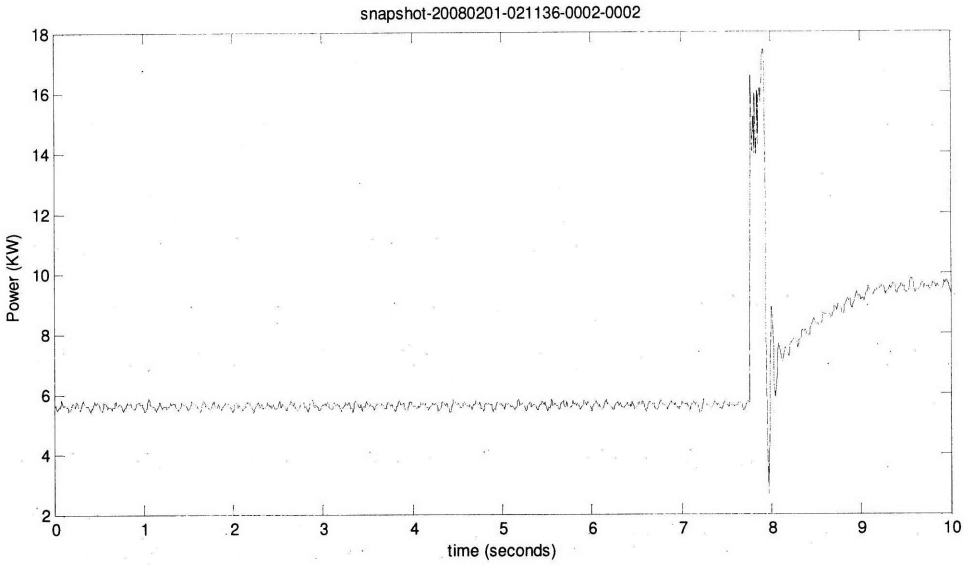


Figure 4-6: Improper bypass valve adjustment. Note that the low-pressure pump is already operating when the high-pressure pump is started at approximately 7.5 seconds. The slow change in power following the transient is due to the closing of the bypass valve.

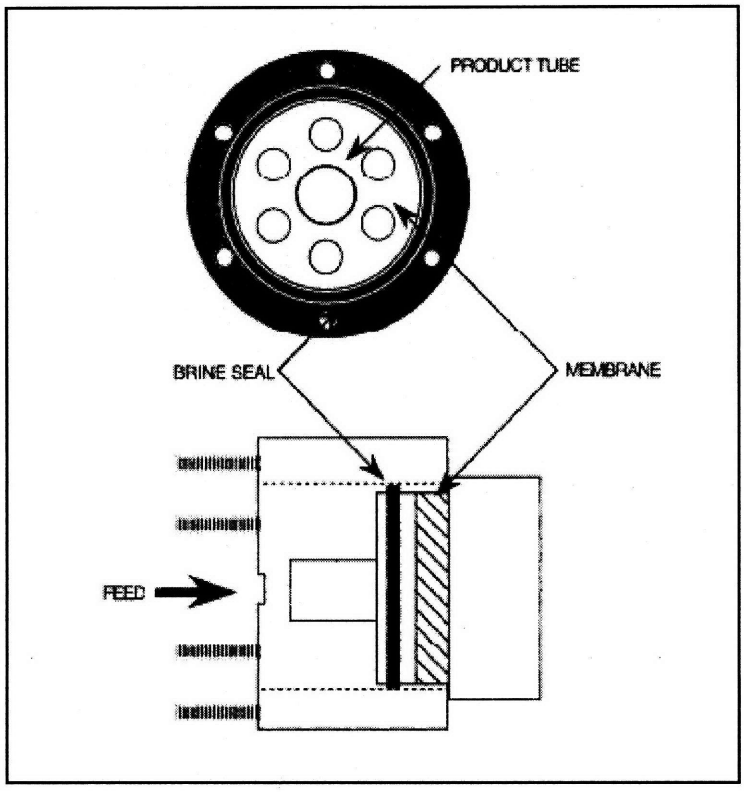


Figure 4-7: Membrane Pressure Vessel End Bell (Operation and Maintenance Manual for USCG Model RC7000 Plus Reverse Osmosis Desalination Plant, 2007)

A third operatorational issue was discovered during shut downs. That procedure requires the operator to first slowly open the bypass valve, then secure the high pressure pump, and then finally secure the low pressure pump. Figure 4-8 shows how the power changes when this procedure is not properly followed. In this example, both A and B sides were operating and then shutdown. All three pumps were secured in approximately .25 seconds. Additionally, neither bypass valve was opened, in contrast to the procedure. Rapidly securing the three pumps, on the other hand, is not specifically disallowed, but it is clearly a bad practice.

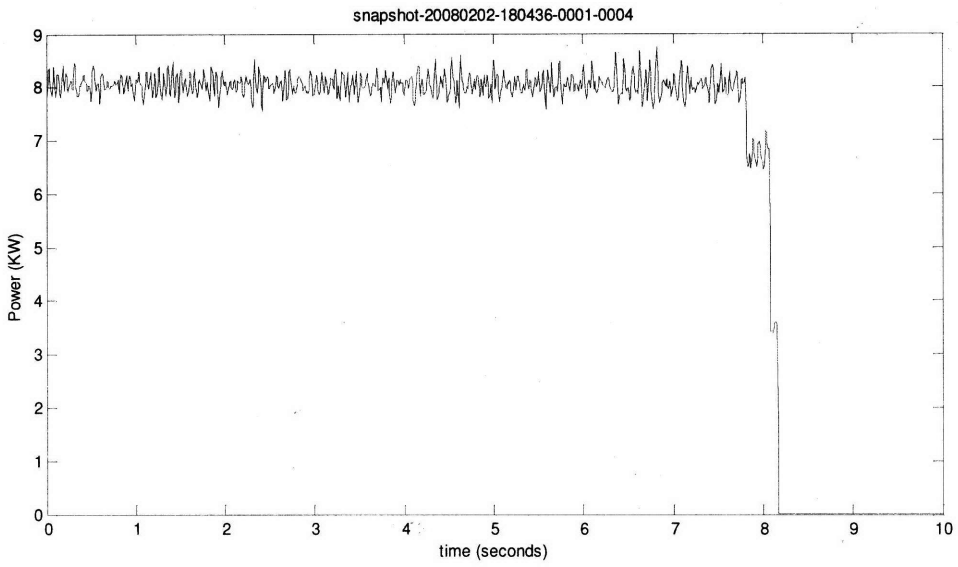


Figure 4-8: Shut down sequence from Feb. 2, 2008. Note that all three pumps were initially running.

All three of these cases provide the Machinery Division Supervisors excellent opportunities to train more junior personnel in the proper operation of equipment. The cases described above can be used to demonstrate exactly how improper operation can affect the system. Additionally, this information provides oversight for operations that are performed when the supervisors are not present. It provides definitive information to the supervisor when the operating procedures are either not being followed or are not clearly understood by the operators. The procedures are specifically designed by the manufacturers to ensure proper operation as well as prolong the life of the equipment.

4.2.2 *Detection of RO System Problems*

During the Winter 2008 cruise, the NILM also detected a number of problems in the RO system. These problems were either inherently related to the RO based on its design, or they were related to the way in which the RO operates as an individual subsystem aboard the ship.

Figure 4-9 demonstrates one issue detected during the winter patrol. To provide further clarity Figure 4-10a shows a typical RO power trace recorded over a one-hour period, and Figure 4-10b shows an abnormal trace recorded on March 9, 2008. This atypical pattern was noticed on multiple occasions throughout the patrol. Figure 4-9 is an example of an extreme case. That figure also presents frequency spectra demonstrating that there are significant changes to the frequency content when the variations are observed. Appendix E contains the MATLAB script used to produce the spectral graphs.

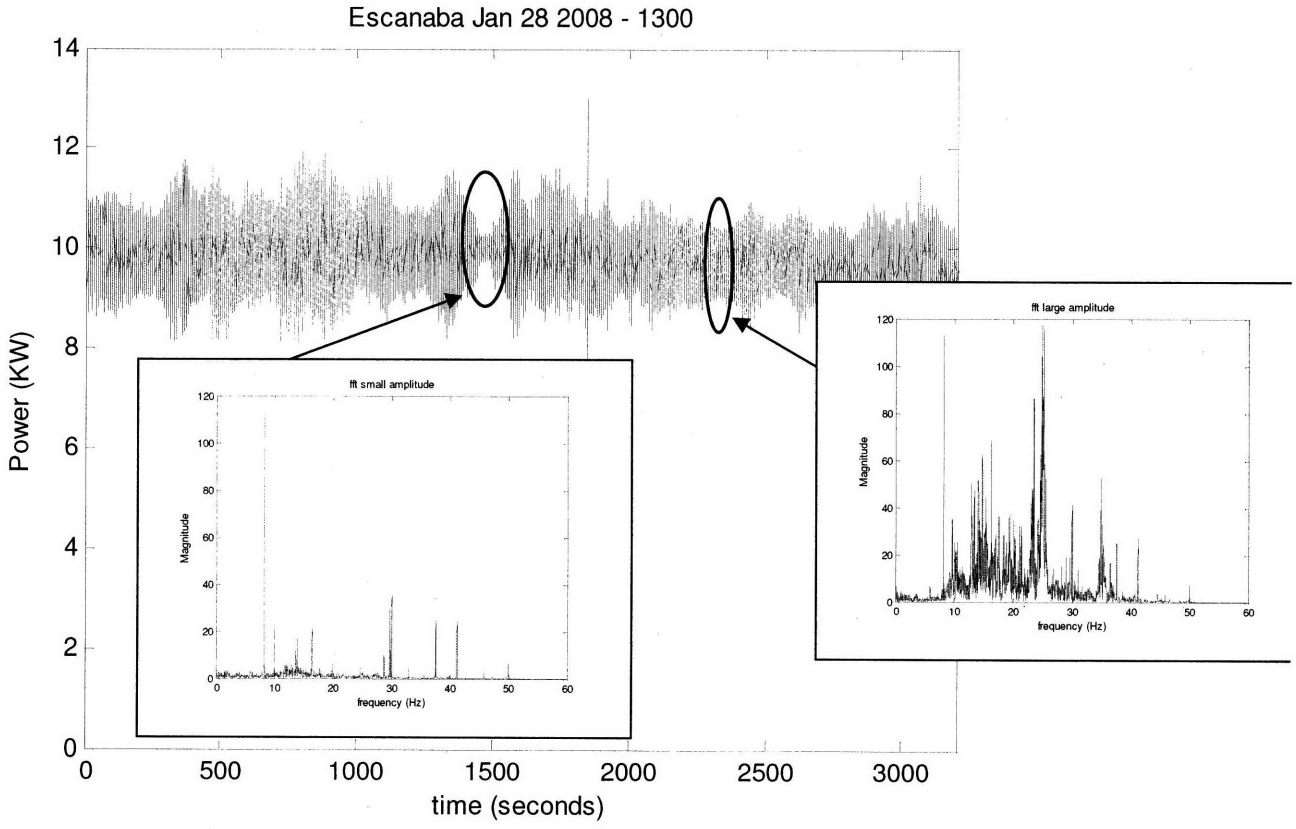


Figure 4-9: Unusual power signal from the RO system aboard USCGC ESCANABA. The two inset plots show the frequency spectrum of the power signal during two different time periods.

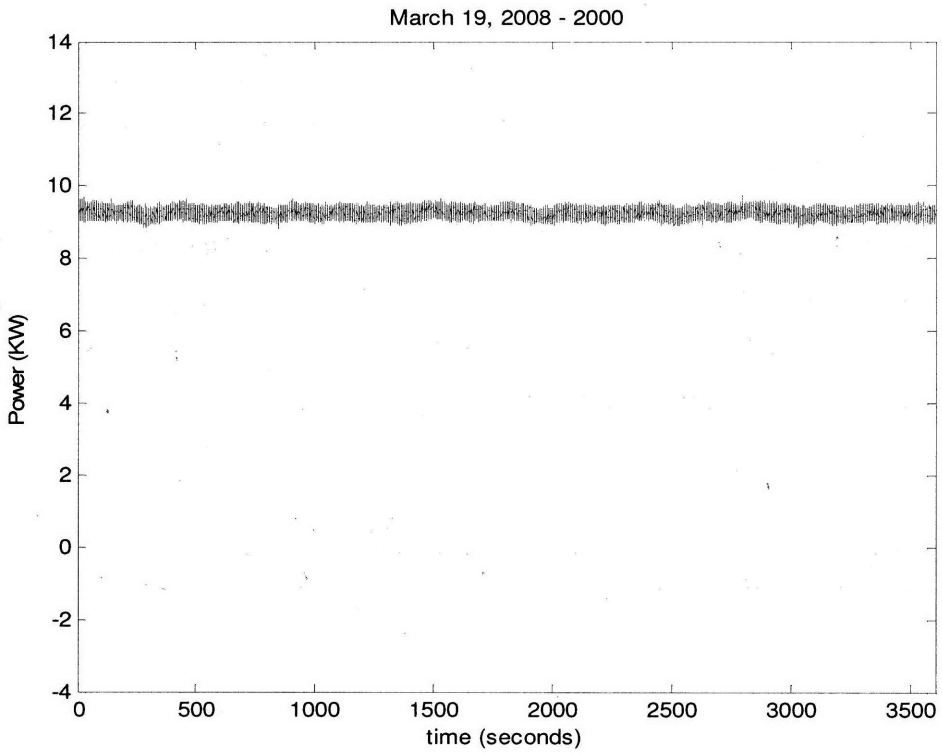


Figure 4-10a: Real power demand during normal system operation.

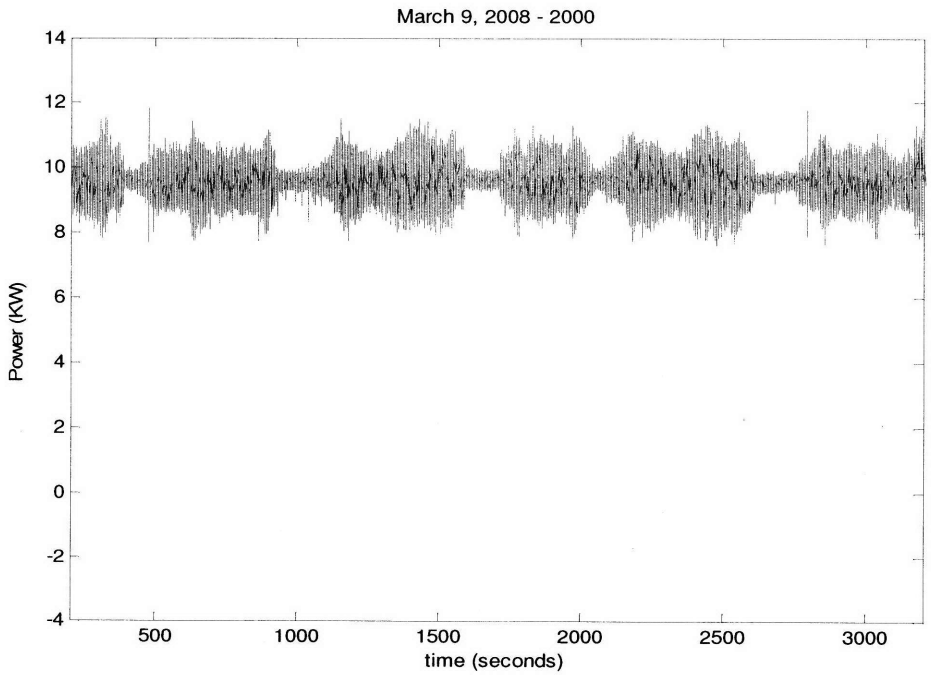


Figure 4-10b: Real power demand during an hour with unusual power disturbances.

After noticing the oscillations shown in Fig. 4-10a and Fig. 4-9, several investigations were performed while the ESCANABA was in port. Several tests were performed to determine if this problem could be related to an external piece of equipment fed from the same voltage source. Upon examining the engineering logs and comparing the events with start and stop times for other engineering components throughout the auxiliary machinery spaces, it was determined that only one piece of equipment had an operating schedule that correlated with the unusual power trace. This was the submersion heaters that are responsible for heating the entire ship. The heaters do not always run, and when they are operating they cycle up or down in 15 kW increments to maintain the set temperature. When all six heaters are energized the system draws 90 kW.

To determine if the heaters were in fact the problem some additional tests were performed, including energizing and securing the submersion heaters while the system was operating, and starting the RO system with heaters secured and with heaters energized. Figure 4-11 illustrates the affect of energizing the heaters while the RO system is operating. Note that when the heaters are energized the maximum power oscillation is approximately seven times the normal power trace. Figures 4-12 and 4-13 illustrate how the heaters affect the system before and after the bypass valve is shut. The actual reason why the submersion heaters cause such a tremendous oscillation in the RO's power signal has not been determined, but the crew has been informed of the problem and is pursuing corrective measure.

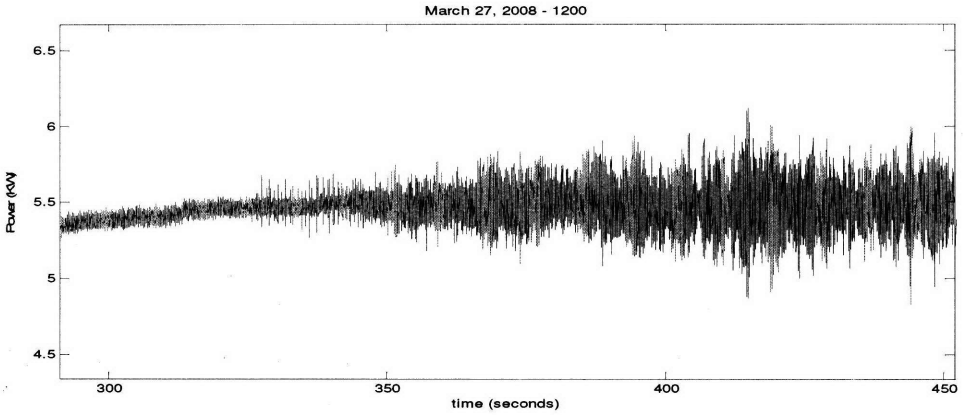


Figure 4-101: RO power trace during submersion heater testing. The heaters were started at approximately 330 sec.

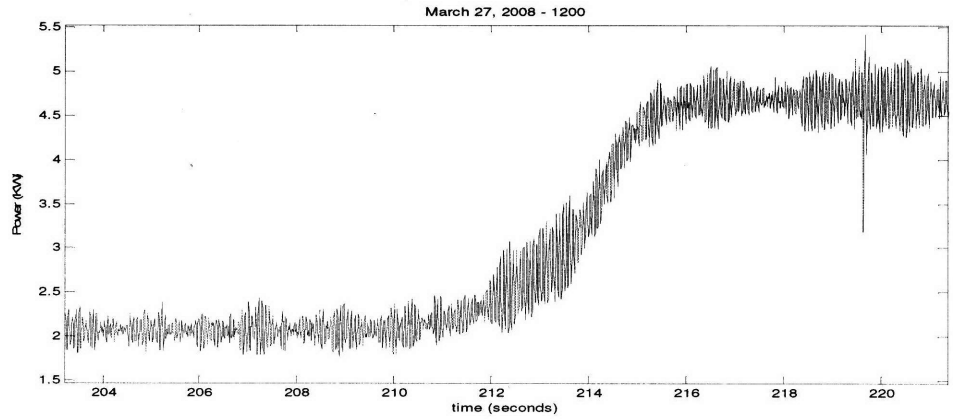


Figure 4-112: RO power trace with the heaters energized. Note that the bypass valve was adjusted starting at approximately 210 seconds.

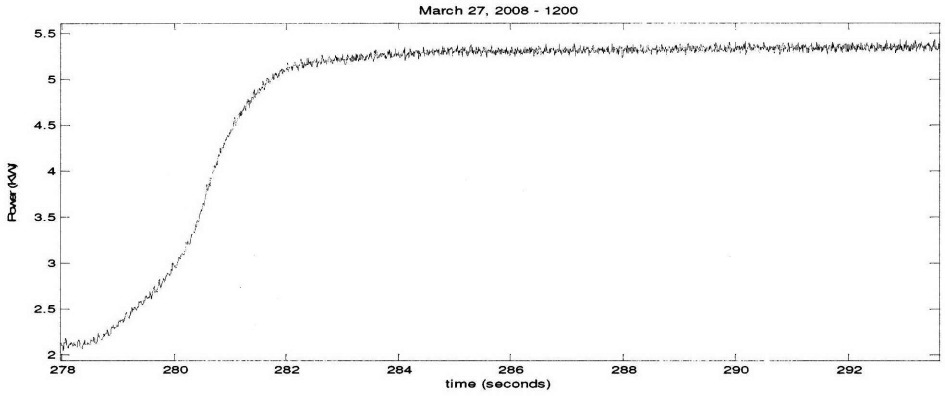


Figure 4-123: RO power trace with the heaters de-energized. Note that the bypass valve was adjusted starting at approximately 278 seconds.

Another issue discovered during the cruise was the observance of a power-quality issue. Numerous times, data analysis has revealed the presence of small "spikes" in the reactive power trace. These spikes seem to occur randomly, but they have been found in all reactive power traces investigated to date. Occasionally, these "spikes" are also noticed in real power. In most cases analyzed to date, the anomaly occurs singularly at random times. On occasion, the effect is noticed to occur in groups of multiple "spikes." Figure 4-14 illustrates the varying degrees of the anomaly.

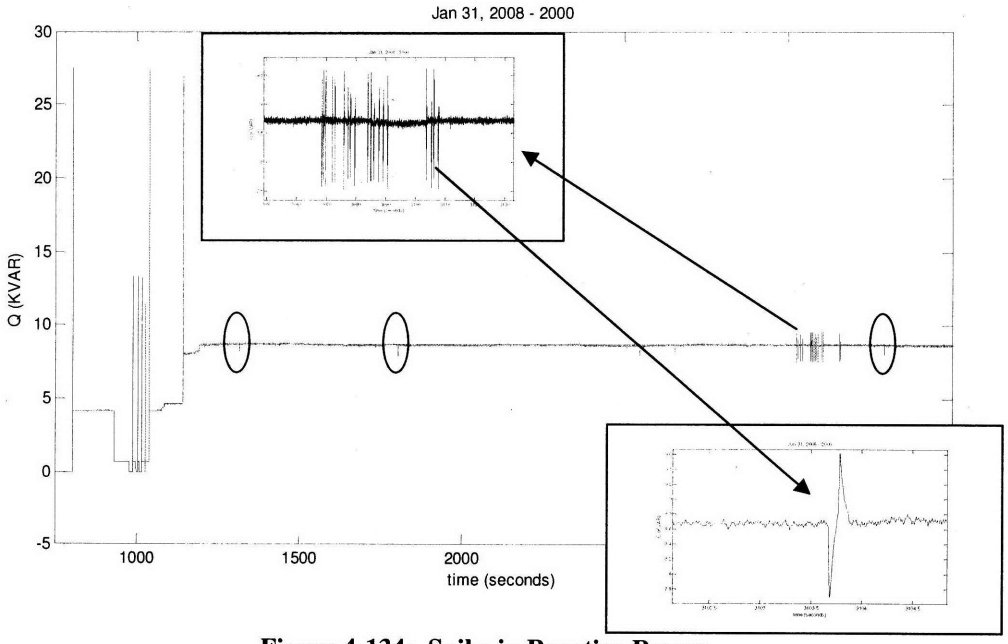


Figure 4-134: Spike in Reactive Power

This problem may be related to the phantom start issue noted previously. The stray spikes could be an indication of a power-quality issue that is causing the system controller to start the HP pump improperly. Because phantom starts occur on all Famous Class Cutters, it was decided to examine SENECA data to look for similar "spikes." Figure 4-15 shows reactive power traces from both the ESCANABA and the SENECA. Note that random spikes are present on both ships. The actual cause of this unusual behavior is not known, however the crew is aware of the problem and is actively trying to determine the source.

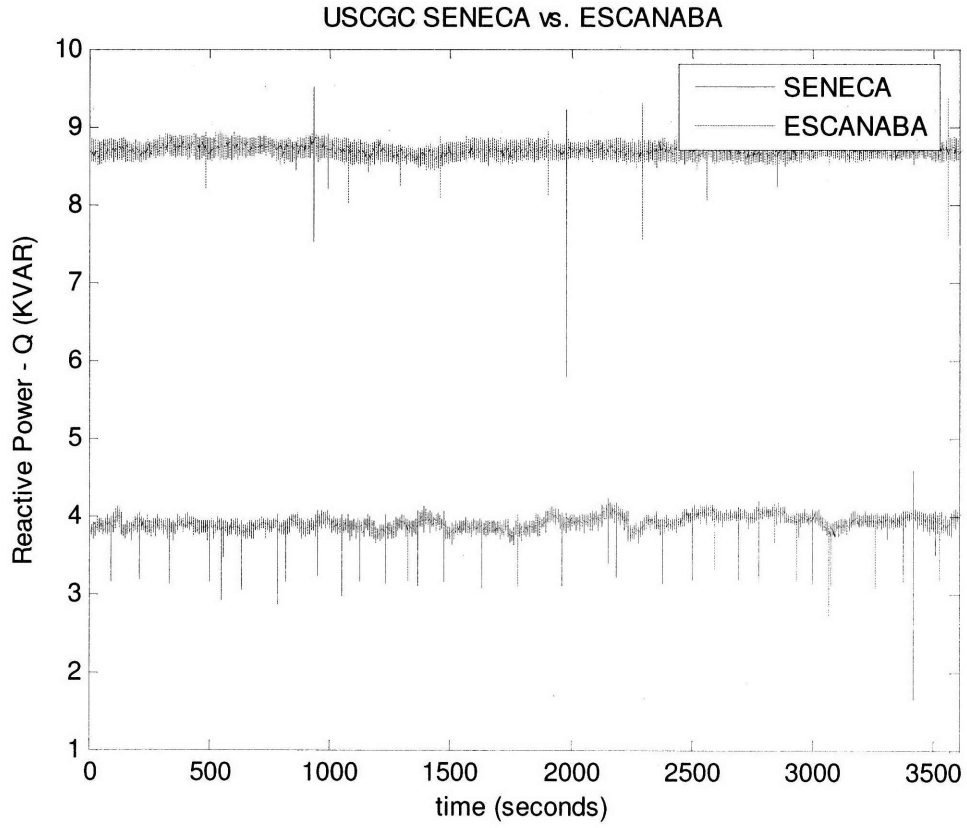


Figure 4-145: Representative reactive power data from both ESCANABA and SENECA. The top trace is from the ESCANABA with both hp pumps running and the bottom trace is from the SENECA with only one hp pump running.

A third issue noticed during the Winter 2008 patrol was an abnormally large oscillation in the power trace while the B-side was running. This phenomenon, which can be seen in Figure 4-16, was seen previously by LT Gregory Mitchell (Mitchell,

2007). When the problem was discovered by LT Mitchell, he noted that there was a large frequency component at 8.26Hz and its harmonics. In 2007 LT Gregory Mitchell discovered that the high pressure pump has a distinct rotational frequency at 8.26 Hz, which is due to the rotational speed of the pump, 495.7 rpm (Mitchell, 2007). On March 23, 2008 the USCGC ESCANABA experienced a very similar occurrence. The oscillation lasted for almost 2 hours before dissipating. Since the incident, the crew has reported that the B-side pump has experienced several problems including a loud knocking noise on start up, which disappears after a few seconds. The pump's leak rate has become excessive, and the crew has scheduled a complete rebuild for this pump, to be completed this in-port maintenance period. Note that the oscillation noted here can be distinguished from the submersion heater problem by the presence of the uniform 8.26 Hz. In the case of the submersion heater issue, there is no effect on the frequency content at that frequency.

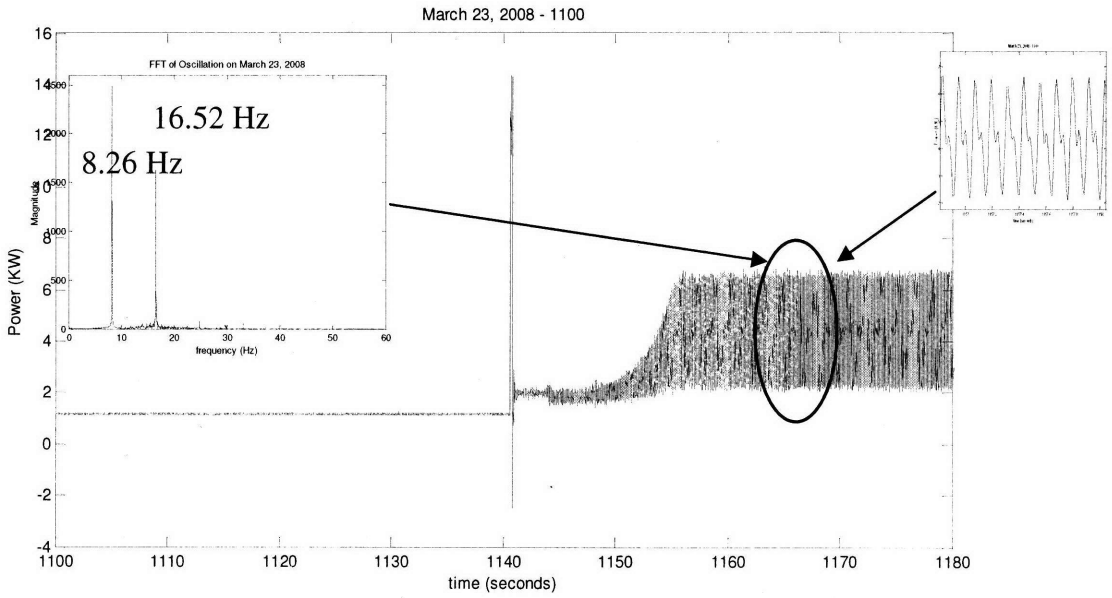


Figure 4-156: B-side High Pressure Pump Oscillation

In all of the instances mentioned above the NILM can be used to detect the problem and provide immediate indications to the operator. Additionally, diagnostics can be designed and implemented to not only display the problems, but also to alert the

operator that an abnormal condition exists. This type of implementation is critical to an effective conditioned based maintenance program.

5 Diagnostics

One of the main customer requirements for the development of the NILM software was to make the system easy to use and understand. This requirement combined with a fairly aggressive time constraint led to an initial prototype system with minimal diagnostic capabilities. The NILM is capable, however, of performing sophisticated analysis. Additionally, the NILM could interact with the system to provide additional safety features and more efficient operating profiles. This chapter describes several diagnostic metrics that could be deployed as a part of the RO system UI. The primary goal of the 2007-2008 NILM team was to develop a real-time user interface for the current NILM system. Based on the tremendous success of this beta version test the USCGC ESCANABA Engineering Officer has agreed to continue the program and assist in further developing the NILM program. The 2008-2009 team will be to working to design and implement the diagnostics described in this chapter, and a working diagnostics update will be installed by the end of 2008.

5.1 Start Sequence Figure of Merit (FOM)

One useful piece of information that the NILM can provide to the RO operator is feedback about the start-up process. Recall from Section 4.2.1 that although it is critical to perform these operations in a certain order, the crew does not always do so. To prolong component life and prevent excessive failures, it would be helpful for the NILM to analyze each start sequence and provide a measure of how effective the operator was at starting the system. In this case a score or figure of merit (FOM) would be assigned to each and every start so that the supervisor could determine how effectively the system is being operated. This FOM would aide in preventing damage to the unit, as well as provide a basis for structured training on the operation of the RO unit.

The FOM would be determined by assigning a score to each portion of the start that is sequential or time dependent. The start can be broken into 6 separate steps: low pressure pump start; micron filter condition; low pressure pump reaching steady state before the high pressure pump is started; first bypass valve operation; second high pressure pump start; second bypass valve operation. Each step can be assigned a numeric value, which when summed together equal 1. Hence, the best total FOM is 1 and the

lowest is 0. The number assigned to each step is described in Figure 5-1. If the start sequence does not include the starting of a second high pressure pump then the FOM score will be out a total of .7, normalized to 1.

Table 5-1: FOM Values for Start Sequence

LP Start	Filter Condition	Operator wait for S.S. / HP Start	Bypass Valve Operation	HP Start	Bypass Valve Operation
.2	.2	.2	.1	.2	.1

The most important part of this algorithm was determining what constitutes a failure. The filter condition was considered adequate if the time to steady state was less than three seconds, see section 5.2.3. The operator was given full credit for the 'wait for steady state' step as long as the high pressure pump was started after the timer determined the system had reached steady state. The bypass valve was determined satisfactory as long as the valve operation (from fully open to fully shut) occurred over a time of greater than 4 seconds. This value was determined by the Machinery Room Supervisor MKC Scott T. Galvin. If the step was done correctly then the operator would get full credit for that particular step, however if any portion of the step was done improperly then no credit will be given.

Applying this algorithm to all the starts in this patrol yields the following results. Out of 70 recorded starts during the 60 day patrol, the average FOM was .847. There were 42 perfect starts. The lowest FOM of .2 occurred twice, .4 was scored six times and the remaining 20 starts were given a FOM of between .6 and .8. The most common problem was improper operation of the bypass valve.

5.2 Oscillations Due to the HP Pump

In order for NILM to be used effectively as a part of a condition-based maintenance program, the system has to be able to report on the relative health of individual components. One way that this can be done is by establishing baseline information from

individual components and comparing real time output to the baseline data. Provided that the baseline data is correct, this could provide valuable information to the operator. This information could be utilized to schedule maintenance upkeeps as needed in order to prevent a total system failure, without having to conduct expensive and time consuming periodic inspections. The characteristic 8.26 Hz signal originating from the high pressure pumps is one tool that can be used to track the relative health of those pumps.

As mentioned in the previous Chapter, the 8.26 Hz signal was initially investigated by LT Gregory Mitchell, USN, who discovered that the frequency exactly matches the rotating frequency of the high pressure pumps (Mitchell, 2007). Mitchell also noted that the signal's relative magnitude provides a general indication of how healthy the pump is. Figure 5-1 illustrates the abnormal condition that occurred on the ESCANABA in 2006.

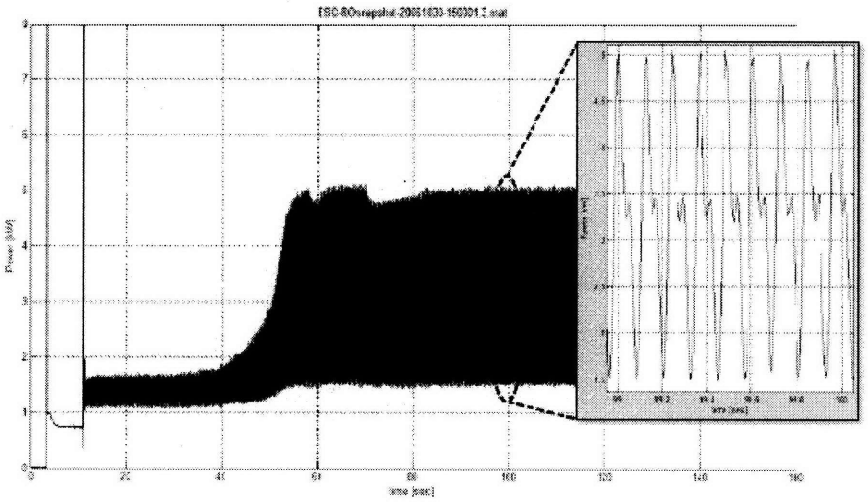


Figure 5-1: HP pump power extreme amplitude (Mitchell, 2007).

Figure 5-2 shows the frequency spectrum of the real power waveform presented in Figure 5-1. Note that there is significant spectral content at approximately 8.26 Hz and its harmonics (Mitchell, 2007).

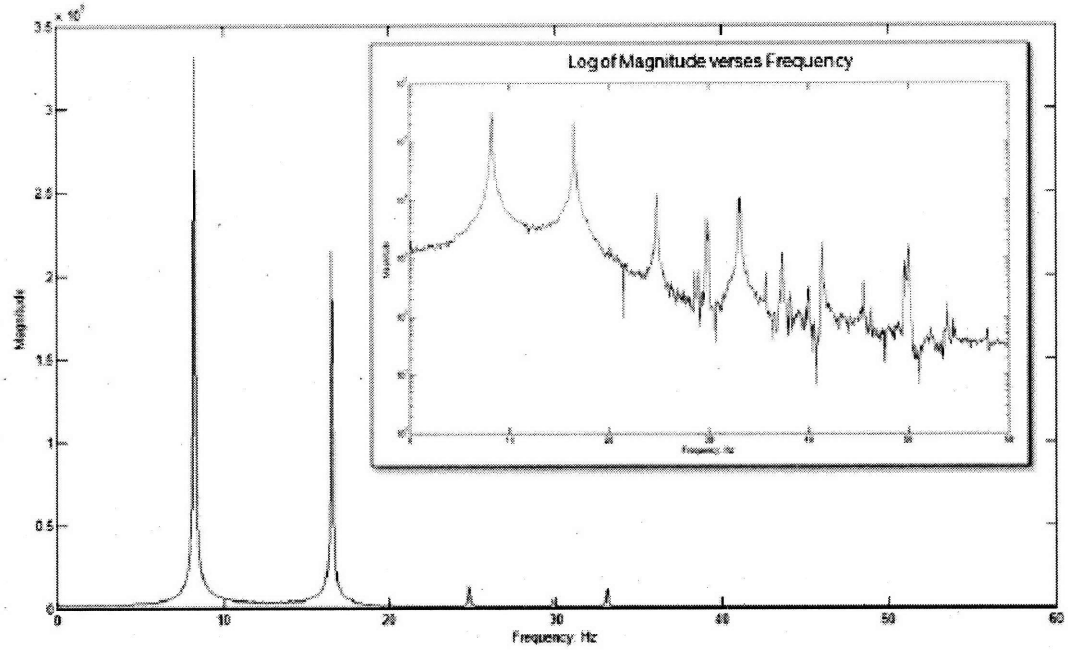


Figure 5-2: Frequency spectrum analysis of Figure 5-1 (Mitchell, 2007).

Additionally he noted that the amplitude of the 8.26 Hz could be used as an indication of relative pump health. Through analysis of high pump starts before and after a pump replacement he was able to show how the magnitude changes. Most he noted that the pump was replaced shortly after the extreme amplitude condition occurred.

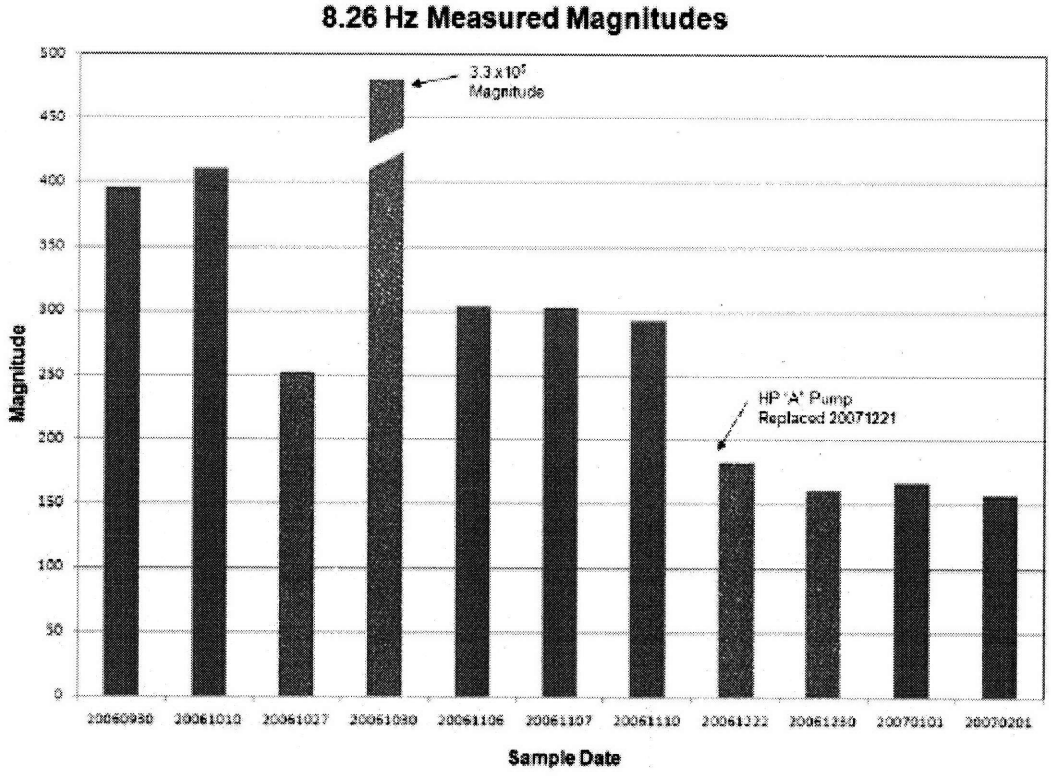


Figure 5-3: 8.26 Hz magnitude trending for RO unit hp pumps (Mitchell, 2007).

As noted by LT Mitchell, there is a clear change in the magnitude of the 8.26Hz signal over time. One could establish a diagnostic that trended the magnitude of this signal for each pump. Such a diagnostic would likely require a change-of-mean filter that would search for a relative change in the magnitude of the 8.26Hz signal.

5.2.1 High Pressure Pump Real Power Waveform Analysis

The fact that the pump's rotational frequency shows up so strongly in the power signal suggests that more powerful diagnostics may be possible. For instance, one may be able to track the action of the pistons inside the actual pump. Such information might better indicate pump health and even provide an insight into the status of other components in the system, i.e. filters, membranes, etc. To consider this possibility, it was decided to analyze the power trace more carefully.

A more consistent diagnostic tool would be to examine the power signal in the time domain and determine which parts of the signal can be attributed to the high

pressure pump. For example, using MATLAB Simulink, it should be possible to recreate the typical power trace, thereby identifying each contributor as they are added in. A simple example of this technique is to start with the trace in Figure 5-1, when the oscillations were present. The assumption has been that the oscillation in the high pressure pump is causing the unusual waveform. If this is true then one should be able to recreate the waveform using Simulink. To conduct this only the components of the waveform known to be associated with the high pressure pump were used. The 8.26 spike is the rotational frequency of the pump. The 16.5 spike can be assumed to be a harmonic of the rotational frequency. All three pumps in the RO system are driven by 1800 rpm motors, therefore there should be some affect around 30 Hz. Lastly, the high pressure pumps are 5 piston pumps, hence there should be some spectral contribution around 41 Hz. To ensure that these frequencies are present in the frequency domain an fft of the abnormal power oscillation that occurred in March 2008 was examined. Figure 5-4 is the frequency spectrum analysis of the abnormal waveform.

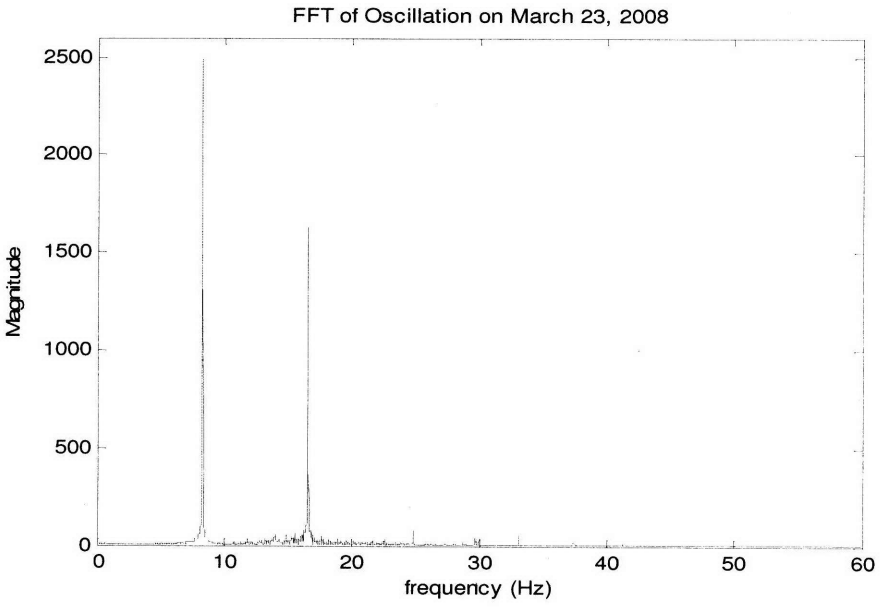


Figure 5-4: FFT of Power during abnormal oscillation

If these frequencies are approximated correctly in Simulink then the output should be similar to Figure 5-1. Figure 5-5 is the Simulink model used, and it is apparent the modeled signal is a good approximation of the actual signal.

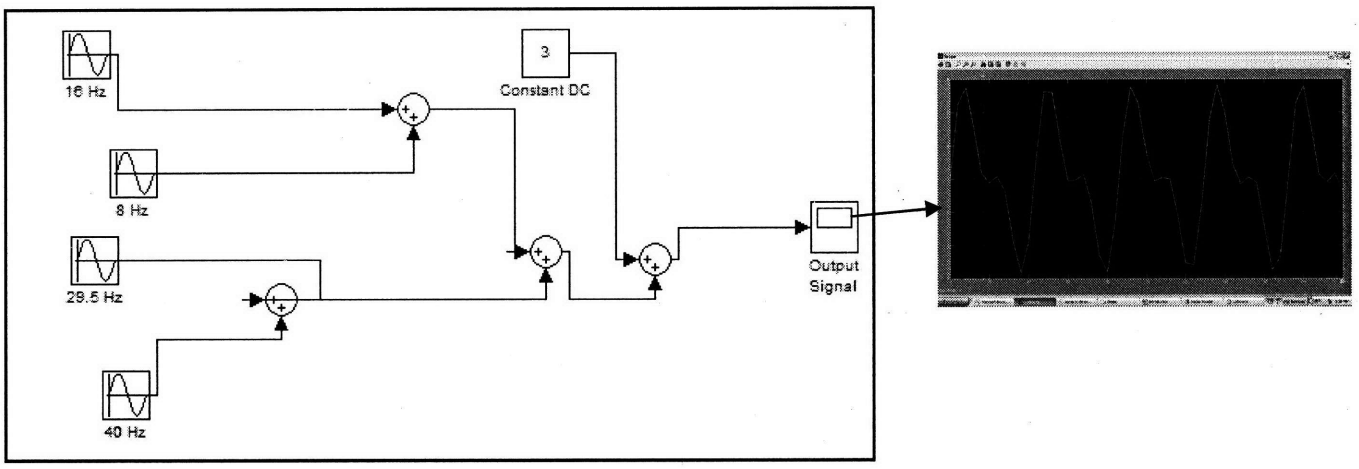


Figure 5-5: Simulink Model of abnormal power waveform

The next step is to determine what makes up the non casualty power trace. A typical RO aggregate power signal can be seen in Figure 5-7. Note that there is not a readily discernable pattern in the trace.

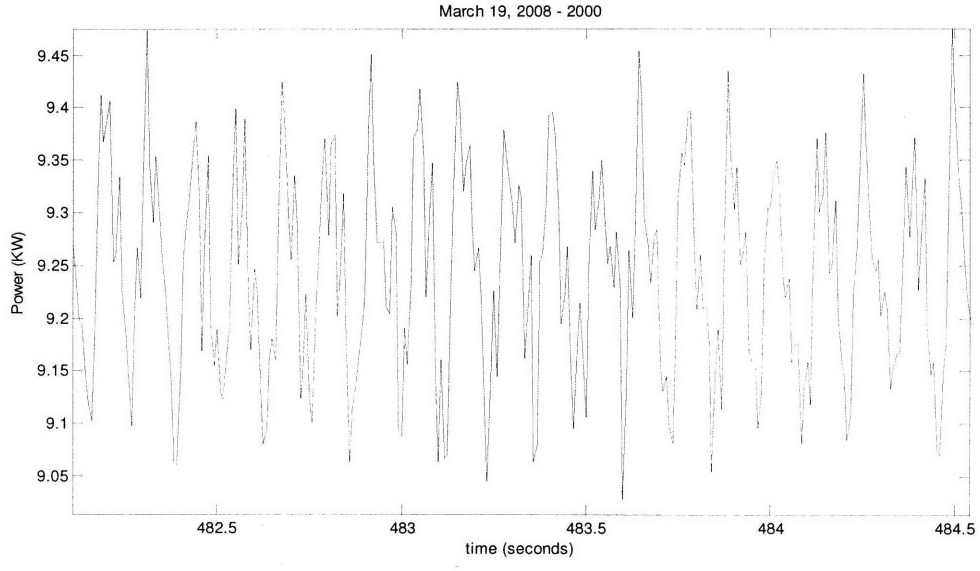


Figure 5-6: Close up of normal (good) power trace

To determine if the same high pressure pump components are present in the non casualty power waveform a spectral analysis of Figure 5-6 was performed. This can be seen in Figure 5-7.

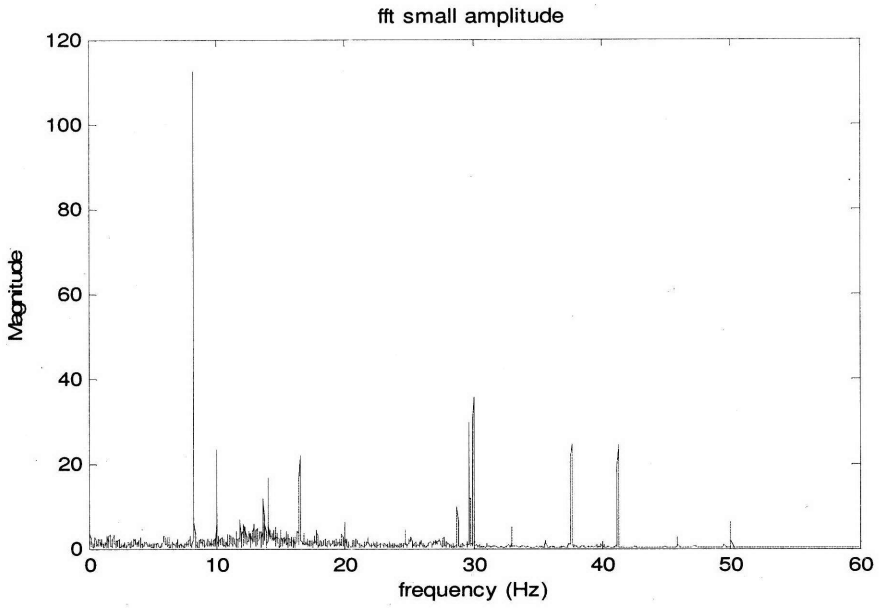


Figure 5-7: Spectral analysis of good power waveform.

From Figure 5-7 (FFT of normal power trace) it is apparent that the same signals are present; however they are modulated by a lot of other signals. Some of these signals are from the high pressure pumps, while many of them may be unrelated to the high pressure pumps and further distort what is happening. One solution is to simply filter out what is desired and ignore the rest. Assuming that under normal conditions both high pressure pumps contribute equally to their associated frequencies and that the each pump accounts for one-third of the motor frequency, then a filter can be designed to remove the desired information. Figure 5-8 is the filtered signal.

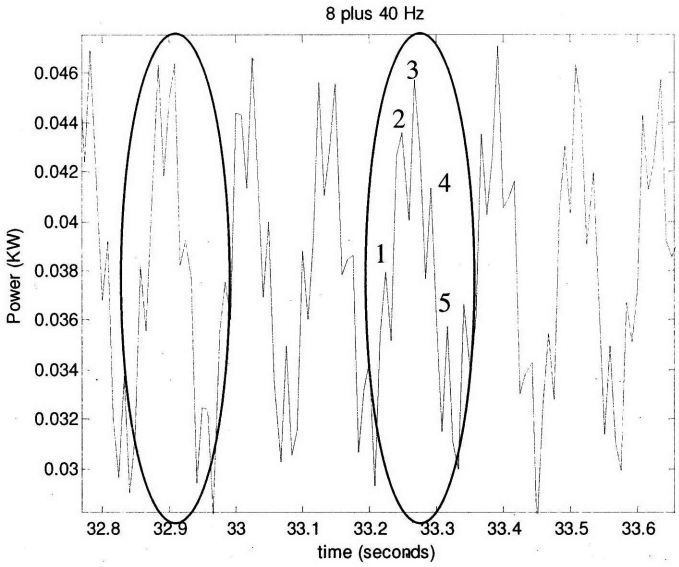


Figure 5-8: Filtered Power Signal

Unlike in the unfiltered power signal, Figure 5-8 shows that a clear pattern is emerging. The pattern appears in peak pairs of five. Examining the filtered trace in Figure 5-8 the pattern appears to repeat itself every 0.121 seconds. This is consistent with the piston theory. The pump rotates at a speed of 495.7 rpm; therefore all five pistons should go through 8.261 rotations per second or one full rotation every 0.12105 seconds. The most important part of this discovery is that if the piston order is known then the operator will be able to determine which, if any, of the pistons or cylinders are damaged or not operating.

5.3 Membrane Failure

One of the most crippling casualties that can occur when using a reverse osmosis system is a membrane failure. These problems are difficult to diagnose, as well as expensive and time consuming to repair. There are two basic symptoms that will accompany a membrane failure; low potable water production and high total dissolved solids (TDS). These symptoms are not specific to a membrane failure, however, as they can be caused by other component failures. Table 5-2 lists all the membrane related faults having symptoms similar to those of a complete failure.

Table 5-2: RO System Membrane-Related Failures

Casualty	*Symptoms*	*Correction*
Membrane Failure	1. High TDS 2. Low effluent production 3. Low membrane array pressure	Determine which membrane is damaged and replace
HP Pump Suction and Discharge Valve Failure	1. High TDS 2. Low reject flow rate (FM1)	If any wear is present must replace all suction and discharge spring and valve assemblies
Pressure Housing Failure	1. High TDS 2. Low membrane array pressure	Inspect and replace o-ring seals in end cap and housing interconnectors
Product Water Relief Valve (V-9) Failure	1. Low effluent 2. Low membrane array pressure 3. Low differential pressure between normal an dump modes at PG5	Adjust relief valve setting IAW Tech Man

Another issue that further complicates the diagnosis of complete membrane failure is that once a failure has been identified, more testing is needed to determine which membrane failed. This requires taking product water samples from each membrane housing while the system is operating. There are four membrane housings.

The NILM could greatly simplify membrane-failure diagnosis. A membrane failure is immediately recognizable in the power trace because a change in membrane conditions will greatly affect flow conditions. As soon as the failure occurs, the real power in the system is drastically reduced. The reactive power also decreases, but by a much smaller fraction. . Both of these effects are visible in Figure 5-10, which presents data recorded before and after a membrane failure aboard the SENECA on 11 October 2006.

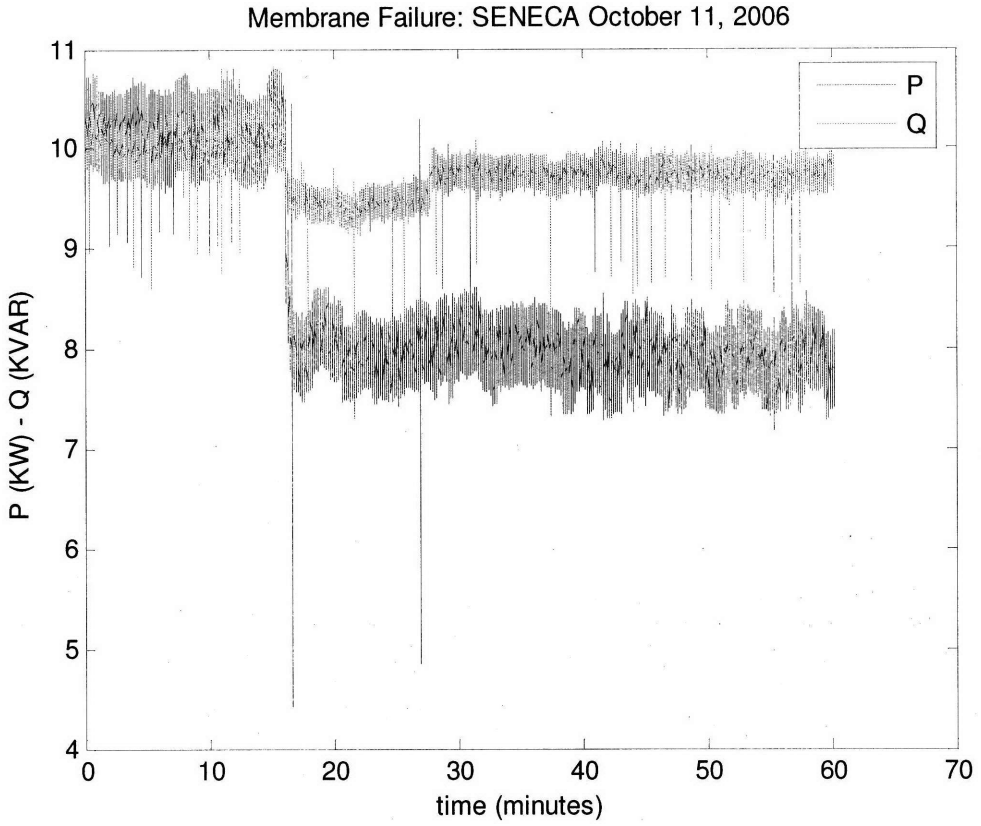


Figure 5-9: Real and Reactive power recorded before and after a membrane failure aboard the USCGC SENECA, October 2006

The data shown in Fig. 5-10 suggests that a membrane-failure diagnostic should make use of a simple change-of-mean detector. The NILM could display this information to the operator. If the operator needed to determine which side the failure affected, they would need to perform off-line testing. To do so, the operator would simply start one side at a time and shut the bypass valve. If the bypass event is detected this means that the failure is not on that side. However, if the valve manipulation is not detected then the membrane failure is on that side. This is because the real power is indicative of the flow conditions in the system. If a membrane failure has occurred then shutting the bypass valve will have little or no affect on the real power and will not be detected by Ginzu. If the membrane is intact shutting the bypass valve will cause the system pressure to increase from 60 psig to 1000 psig and will be readily detected by Ginzu. The only diagnostic left is to determine which pressure vessel contains the damaged membrane. This can be done by testing the product water from each vessel.

5.4 Filter Condition

As mentioned before LT Gregory Mitchell was the first to notice that there may be some correlation between the time it takes for the low pressure pump to reach steady state and the condition of the micron filter. He theorized that as the micron filters became fouled and dirty, that the low pressure pump would take longer to reach steady state. Figure 5-11 is a graph of start times from the USCGC ESCANABA in January 2007.

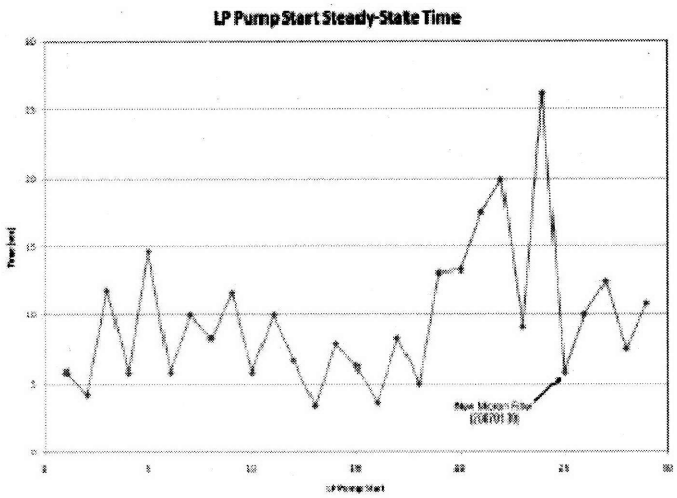


Figure 5-10: LP Pump steady state times for January 2007 (Mitchell, 2007)

LT Mitchell noted that the steady-state times increased just before the filter elements were replaced and suddenly decreased after the replacement. In order to determine a useful diagnostic for this failure, it is necessary to determine all the factors that affect the time to steady state. It is clear from the graph that the correlation is not consistent until just before the filters are replaced. He noted specifically that the correlation was strongest as the differential pressure across the micron filters increased to above 12 psig. Before reaching this high differential pressure the times vary slightly around a relatively constant value. Although this variation would make it difficult to present a running real-time filter condition value, the data suggests that it might be possible to indicate an impending failure. To be able to make such a statement on a

definitive basis, it is important to determine how other factors affect the time to steady state.

To create a useful diagnostic for detecting impending membrane failures, all additional factors affecting time to steady state must first be defined and second each factor's effect must be determined. Based on a study of the system, three control variables were identified. These are the filter differential pressure, the filter inlet pressure (or low-pressure pump outlet pressure), and the temperature of the seawater. Changes in all of these variables would be expected to cause changes in the time to steady state.

To test the effect of each of the control variables identified previously, data was compiled from the Engineering Logs of the USCGC ESCANABA during their Fall 2007 cruise. In order to determine the effect of these control variables, a simple regression analysis was performed in MATLAB. MATLAB solves a set of simultaneous linear equations that represent the system using a least-squares fit. In this case, the time to steady state can be represented using the equation

$$y = a0 + a1x1 + a2x2 + a3x3 \quad (6)$$

where $x1$ is the differential pressure across the micron filter, $x2$ is the pump outlet pressure, $x3$ is the seawater temperature, and y is the time to steady state. Multiple regression solves for the unknown coefficients $a1$, $a2$ and $a3$. Additionally, MATLAB solves for an error term represented by $a0$. The error term is treated simply as a random variable that represents unexplained variation in the response (Berk, 2004). The values of each of the parameters are presented in Table 5-3. The data used to determine these parameters can be found in Appendix D.

Table 5-3: Regression analysis result.

Variable	*Value*
a0	1.8426
a1	.0584
a2	.0114
a3	-.0235
y	Response Variable

Figure 5-12 shows a plot of both the predicted and actual time to steady-state values for the ESCANABA's Fall 2007 cruise. Note that there is a strong deviation around start 1 and start 55. These deviations appear to occur after extended in port periods. A suspected cause for this deviation is that it likely takes several starts before all of the entrapped air can be removed from the filter. Figure 5-13 illustrates the fit data with the initial starts after prolonged in port periods removed.

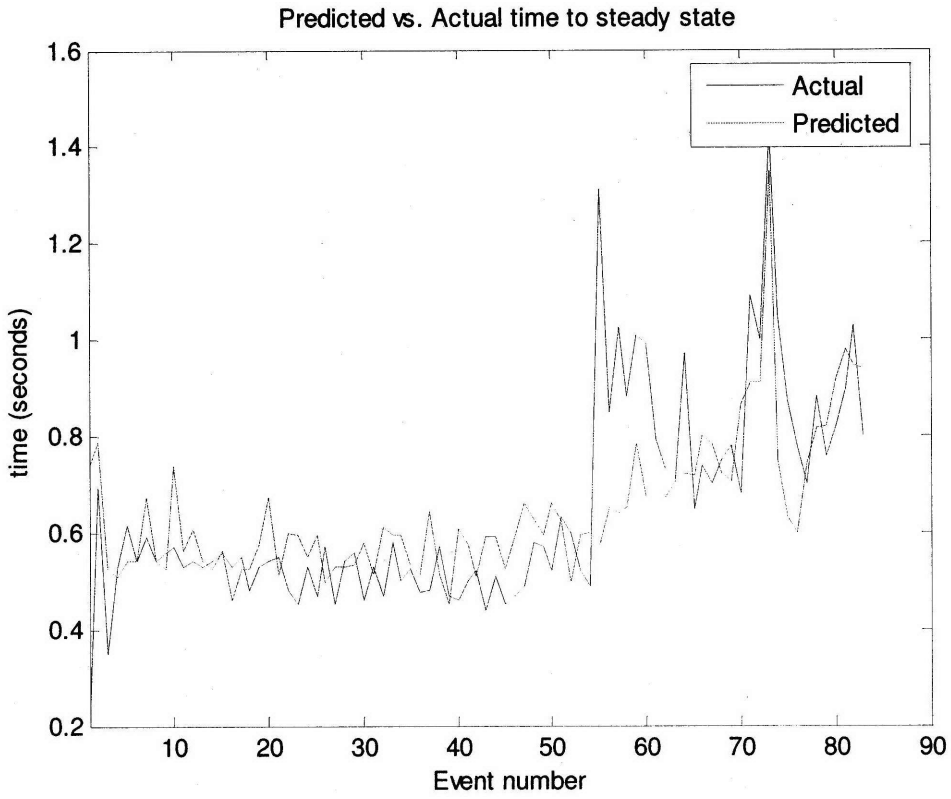


Figure 5-11: Predicted vs. Actual time to steady state

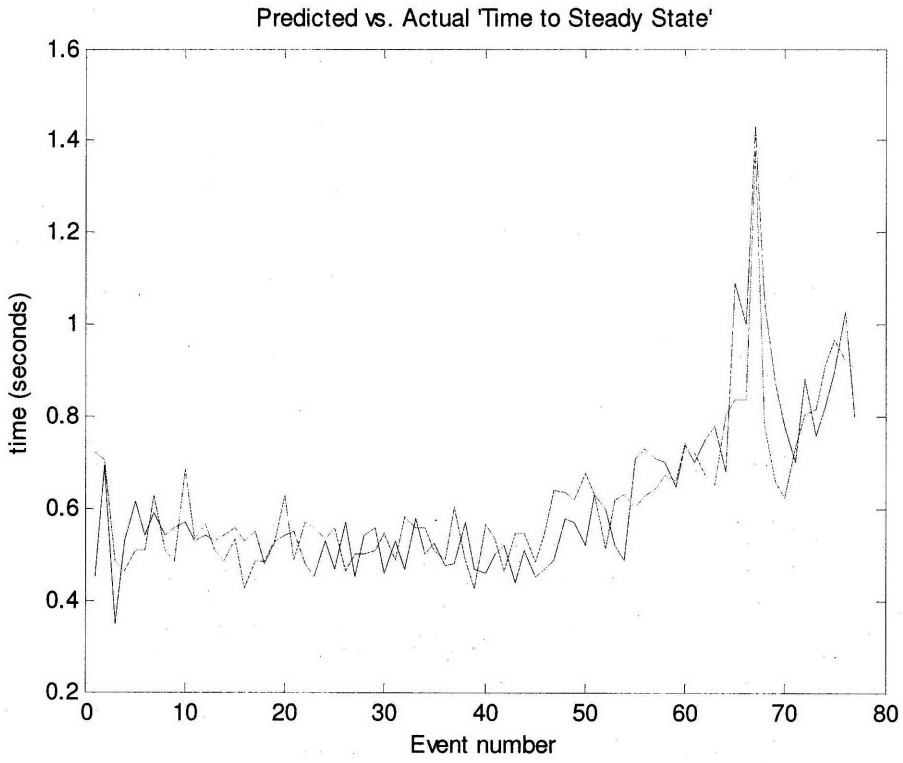


Figure 5-12: Predicted vs. Actual time to steady state times without extended layup period starts

Using this information, the next step is to determine what to pass on to the operator. The value of this analysis is that the regression model provides a much clearer picture of the interactions within the system. For example, the temperature of the seawater has a negative proportional effect on the time. Hence, it can be seen that the lower the temperature the longer the time to steady state. Adversely, the pump pressure has a proportional effect. Based on the operating profile of the ship it can be assumed that the ship will never operate in water less than 48 F or greater than 90 F, and that the pump pressure will never exceed 65 psig and never be less than 35 psig. Additionally, according to the RC7000 Plus Reverse Osmosis Technical Manual the micron filter element should not be allowed to exceed a differential pressure of 15 psig. Using this information and worst case temperature and pump pressure data it can be determined that if the time to steady state is less than 1 second, then the filter does not need to be replaced. However, under best case conditions the time to steady state may reach as high as 2.3 seconds.

There are two approaches to determining a useful diagnostic with this information. The first is to simply place the alarm set point to 1 second. This will ensure that no matter what the conditions are the system will be safe. However, this approach is not optimal from a condition based maintenance approach because it could lead to premature replacement of the filters. The second approach is to establish temperature groups that can be adjusted by the operator. Figure 5-12 is an example of how the temperatures could be set up in zones, each with a different alarm set point.

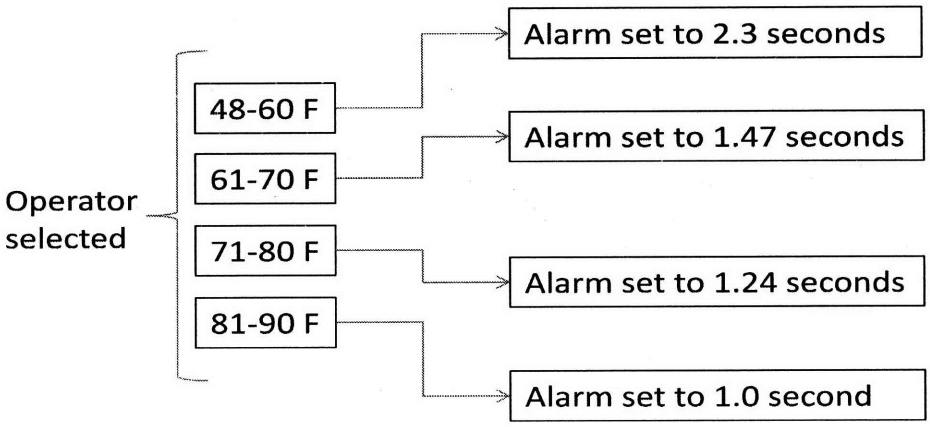


Figure 5-13: Temperature alarm zones for diagnostic

Next the information has to be passed in the case statement logic. When the user interface is initially started the set point would have to set to 1 second to ensure that the system is safe. However, once the system is operating the operator can input which temperature zone the system is operating in. This input could then be updated in case statement logic sequence. Figure 5-13 depicts how the information would be passed and updated within the user interface. The zone would most likely have to passed as a global variable that can be updated throughout the UI as conditions change.

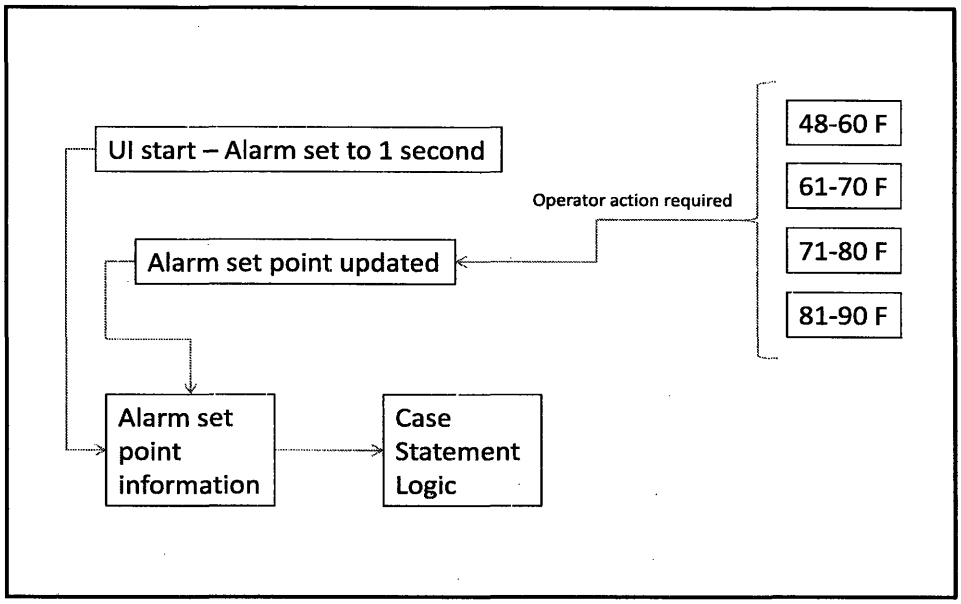


Figure 5-14: Map of temperature zone information update within the case statement logic routine

6 Conclusion

It is clear that the United States Navy will not have enough funding to support the vision of a 313 ship Navy without some significant changes in how maintenance is performed on board ships. The Non Intrusive Load Monitor combined with an effective condition based maintenance system can effectively address this problem. More importantly, NILM does not require an elaborate, unreliable and costly sensor network in order to operate.

This research has shown that the NILM can be used effectively and reliably, to provide real time system information to the operator. This has been done through the development, testing and implementation of a user interface for the RC7000 Plus Reverse Osmosis Desalination Plant on USCGC ESCANABA. The only current road block to full NILM implementation is the time it takes to learn the system characteristics in order to provide useful system information. However, as the technology pushes forward and this learning curve is shortened or automated, the NILM is a cheap and proven method of monitoring equipment and reducing maintenance costs.

7 Bibliography

Bennet, P. L. (2007). *Using Non-Intrusive Load Monitor for Shipboard Supervisory Control.* Cambridge, MA: Massachusetts Institute of Technology.

Berk, R. A. (2004). *Regression Analysis: A Constructive Critique.* Sage Publications.

Castelli, C. (2007, Feb). *Navy Shipbuilding Plan Called Unrealistic.* Retrieved Mar 6, 2008, from Military.com: http://www.military.com/features/0,15240,85654,00.html
Condition Based Maintenance. (n.d.). Retrieved Mar 6, 2008, from Robotics Research Group: http://www.robotics.utexas.edu/rrg/research/conditionb/

Cox, R., M. Piber, G. Mitchell, P. Bennet, J. Paris, W. Wichakool, S. Leeb. 2007. *Improving Shipboard Maintenance Practices Using Non-Intrusive Load Monitoring.* Philadelphia.

Cox, R. (2006). *Minimally Intrusive Strategies for Fault Detection and Energy Monitoring.* Cambridge, MA: Massachusetts Institute of Technology.

Denucci, T. W. (2005). *Diagnostic Indicators for Shipboard Systems Using Non-Intrusive Load Monitoring.* Cambridge, MA: Massachusetts Institute of Technology.

Maier, M. W., & Rechtin, E. (2002). *The Art of Systems Architecting.* Boca Raton, FL: CRC Press LLC.

Maton, A., & Jean Hopkins, S. J. (1997). *Cells Building Blocks of Life.* Upper Saddle River, NJ: Prentice Hall.

McCoy, Kevin M, Rear Admiral, USN (2008). *2N MIT Brief.* Cambrindge, MA.

Mitchell, G. R. (2007). *Shipboard Fluid System Diagnostics using Non-Intrusive Load Monitoring.* Cambrige, MA: Massachusetts Institute of Technology.

Lee, K. (2003). *Electrical Load Information System Based on Non-Intrusive Power Monitoring.* Cambridge, MA: Massachusetts Institute of Technology.

Leeb, S. B. (1993). *A Conjoint Pattern Recognition Pattern to Non-Intrusive Load Monitoring.* Cambridge, MA: Massachusetts Institute of Technology.

Leeb, S., S. Shaw, and J. Kirtley. 1995. Transient Event Detection in Spectral Envelop Estimates for Non-Intrusive Load Monitoring. *IEEE Trans. On Power Delivery* , 1200-1210.

(2007). *Operation and Maintenance Manual for USCG Model RC7000 Plus Reverse Osmosis Desalination Plant.* Gardena, CA.

Oppenheim, A., A. Willinsky, and I. Yound. 1988. *Signals and Systems.* Englewood Cliffs, NJ: Addison Wellesley.

Paris, J. (2006). *A Framework for Non-Intrusive Load Monitoring and Diagnostics.* Cambridge, MA: Massachusetts Institute of Technology.

Piber, M. A. (2007). *Improving Shipboard Maintenance Practices Using Non-Intrusive Load Monitoring.* Boston, MA: Massachusetts Institute of Technology.

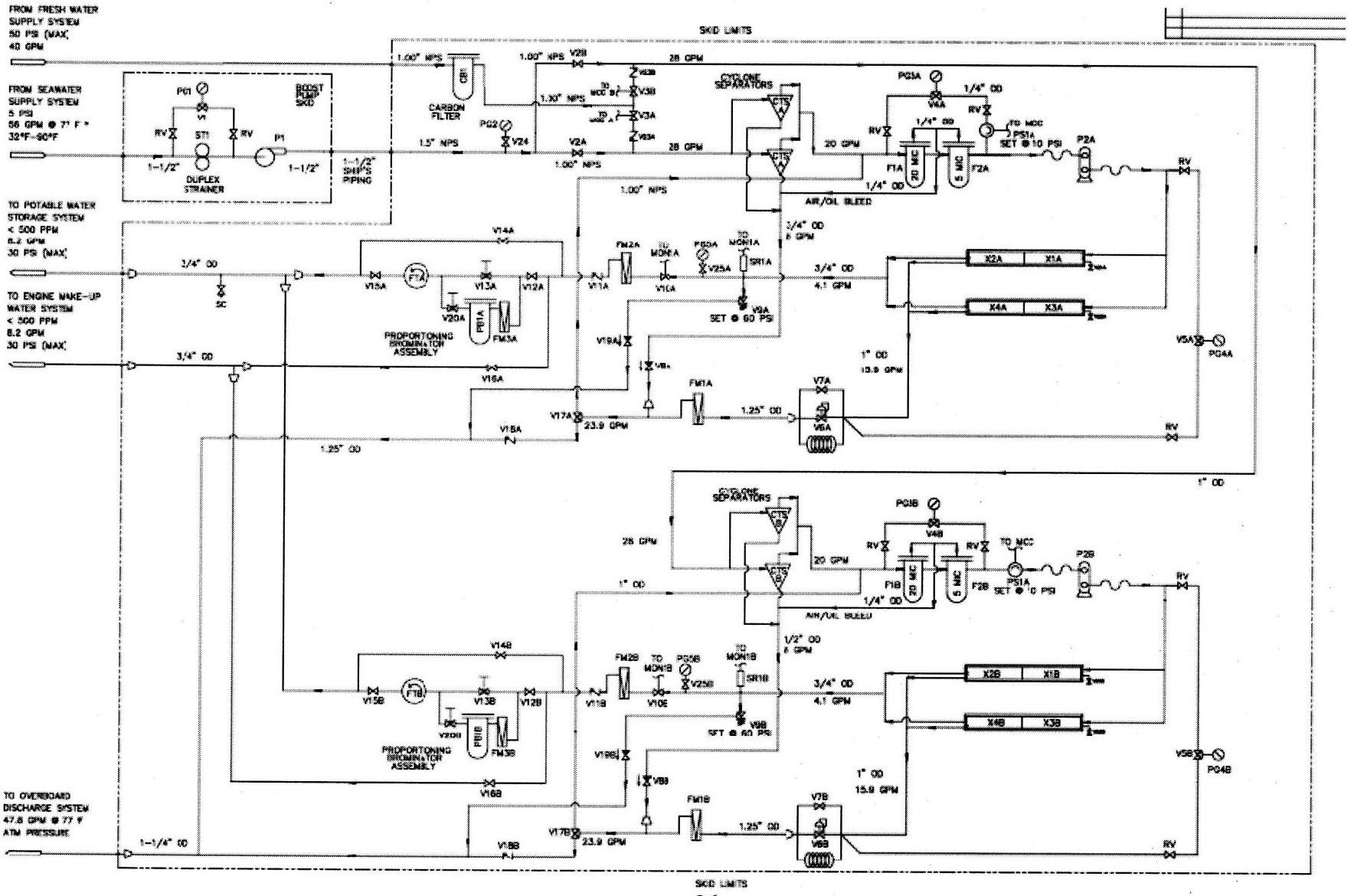
Proper, E. R. (2008). *Automated Classification of Power Signals.* Cambridge, MA: Massachusetts Institute of Technology.

Shaw, S., C. Abler, R. Lepard, D. Luo, S. Leeb, and L. Norford. 1998. Instrumentation for High Performance Non-Intrusive Electrical Load Monitoring. *ASME Journal of Solar Energy Engineering* , 224-229.

Shaw, S. (2000). *System Identification Techniques and Modeling for Non-Intrusive Load Diagnosis.* Cambridge, MA: Massachusetts Institute of Technology.

Uliana, D. (2007, Nov 13). *Matlab GUI Tutorial.* Retrieved Dec 12, 2007, from blinkdagger:http://www.blinkdagger.com/matlab/matlab-gui-tutorial-a-brief-introduction-to-handles

Appendix A RC7000 Plus System Diagram (Operation and Maintenance Manual for USCG Model RC7000 Plus Reverse Osmosis Desalination Plant, 2007)



Appendix B RC7000 Plus Start Up and Shut Down Procedure (Operation and Maintenance Manual for USCG Model RC7000 Plus Reverse Osmosis Desalination Plant, 2007)

4.2 NORMAL STARTUP PROCEDURE

These instructions apply when either side of the RO unit is being started after a normal (short-term) shutdown. If both units are to be started, complete the startup procedure for one unit and then repeat for the second.

1. Place the RO valves and switches in positions indicated in Table 4.3.
2. Check the HP pump oil level by observing the sight gauge located on the pump.
3. Open the appropriate raw water isolation valve (V2).
4. Turn on the electrical supply to the RO unit, if required. If the system is already energized, verify that the message center display indicates "STANDBY MODE." If any other message is displayed, cycle the electrical power OFF and then ON to reset.

ID	Description	Position
V2	Raw Water Isolation Valve	Open
V7	High-Pressure Bypass Valve	Open
V14	Proportioning Brominator Bypass Valve	Closed
V16	Engine Water Make-Up Isolation Valve	Closed
V17	Cleaning Valve	Normal

Table 4.3 – Valve/Switch Line Up – Normal Operation/Startup

CAUTION

Failure to open the high-pressure bypass valve (V7) (which is required to bleed any entrapped air) can result in hydraulic shock to the system.

WARNING

Failure to set the cleaning valve (V17) in the NORMAL position will result in excessive pressure in the RO reject line and possible failure of the RO reject flow meter (FM1).

5. Start the LP boost pump by pressing the LP PUMP pushbutton located on the MCC.
6. Start the HP pump by pressing the HP PUMP pushbutton located on the MCC. At least 10 psig must be indicated on the discharge side of the micron filter array duplex pressure gauge (PG3).
7. When flow through the reject discharge flow meter appears to be free of air bubbles, slowly close the HP bypass valve (V7). It is important to monitor the pressure indicated on the membrane array pressure gauge (PG5).

WARNING

Pressure, as indicated on the membrane array duplex pressure gauge (PG4) should never exceed 1,000 psig on the inlet.

Appendix B RC7000 Plus Start Up and Shut Down Procedure (Operation and Maintenance Manual for USCG Model RC7000 Plus Reverse Osmosis Desalination Plant, 2007)

4.3 BROMINE SYSTEM STARTUP AND OPERATION

The following chart is provided as an aid for the operator in determining the proper flow rate through the bromine cartridge:

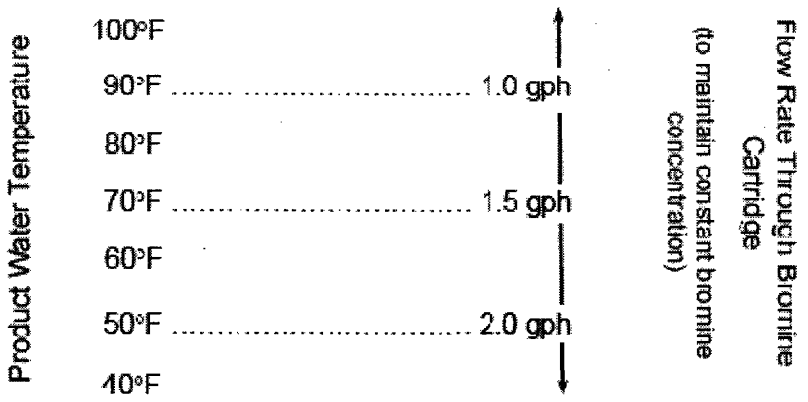


Figure 4.1 – Product Temperature vs. Bromine Cartridge Flow at 3,500 GPD.

The chart illustrates how the flow rate through the cartridge should be changed in order to maintain the recommended 1.0 ppm of bromine as the product temperatures varies. The required flow and observed temperature have an inverse relationship. As the temperature decreases the required flow rate increases and as the temperature increases the required flow rate decreases. This is an important concept to understand when attempting to "dial in" the necessary flow rate required for safe bromination. The specific procedure for setting the proper bromine cartridge flow rate is as follows.

1. Determine the product temperature (this value can be obtained on the water quality monitor display). The desired flow rate at 70 °F is 1.5 gph. For every 10 °F increase in product temperature, the flow through the bromine cartridge should be decreased by 0.25 gph. Conversely, for every 10 °F decrease in product temperature, the flow through the bromine cartridge should be increased by 0.25 gph. Flow through the bromine cartridge, as read on the bromine flow meter, can be adjusted as follows:

   a. Use the proportioning brominator flow control valve (V20) to adjust the flow rate. This is accomplished by slowly rotating the valve counterclockwise to reduce the flow rate and rotating the valve clockwise to increase the flow rate.

   b. Use the proportioning brominator backpressure control valve (V13) to adjust the flow at lower flow rates beyond the range of V20. This is accomplished by slowly rotating the valve clockwise to increase product water backpressure and rotating the valve counterclockwise to decrease product water backpressure.

2. After setting the flow rate in accordance with step 1, take a sample of the brominated product water and measure the bromine concentration using the pocket bromine colorimeter. If the bromine concentration is greater than 1.0 ppm, slowly reduce the flow rate. If the bromine concentration is lower than 0.7 ppm, slowly increase the flow rate. Repeat this step until a constant bromine concentration of 1.0 ppm is achieved.

Appendix B RC7000 Plus Start Up and Shut Down Procedure (Operation and Maintenance Manual for USCG Model RC7000 Plus Reverse Osmosis Desalination Plant, 2007)

WARNING

Failure to manually sample the bromine level of the product water and verify the bromine level at 1.0 ppm may result in an incorrect bromine concentration in the potable water. Do not rely entirely on the flow rates described in Figure 4.1

4.4 SHUTDOWN PROCEDURES

4.4.1 Shutdown Procedure (Short Term)

This shutdown procedure applies if the RO unit will be shut down for a period of time greater than 24 hours. If the anticipated shutdown period will be 5 days or more, refer to Section 4.4.2, Shutdown Procedure (Extended).

1. Release the pressure from the system by turning the high-pressure bypass valve counterclockwise to the OPEN position.
2. Secure the HP pump by pressing the HP PUMP pushbutton located on the MCC.
3. Secure the LP pump by pressing the LP PUMP pushbutton located on the MCC.

NOTE

A single low-pressure boost pump feeds both Alpha and Bravo units. If either unit is running the boost pump should be left on.

4. Shut the appropriate raw water isolation valve (V2).
5. Flush the system in accordance with Section 4.5, Freshwater Flush.

CAUTION

The development and growth of bacteria is accelerated in warm temperatures. To safeguard against the potential fouling of the membrane surface, VMT recommends that the freshwater flush cycle be performed at least every 48 hours when operated in warm water or climates even if the system has not been operated since the last flush.

4.4.2 Shutdown Procedure (Extended)

If the RO unit is to be secured for periods exceeding five days, stagnant water in the system will breed bacteria and other biological material even if the system is flushed with fresh water in accordance with the procedures. These organisms will not directly attack the membranes or other components but can, in sufficient numbers, foul the membrane surface by blocking the product water channels resulting in a membrane flux loss (decrease in product water output). In most cases the fouling can be cleaned from the membrane surface by circulating chemical solutions through the system, but results of the cleaning process are not guaranteed. VMT recommends the membrane preservation process be performed IAW Section 5.6, RO Element Preservation.

```
function varargout = RO_Diagram(varargin)
% RO_DIAGRAM M-file for RO_Diagram.fig
%      RO_DIAGRAM, by itself, creates a new RO_DIAGRAM or raises the existing
%      singleton*.
%
%      H = RO_DIAGRAM returns the handle to a new RO_DIAGRAM or the handle to
%      the existing singleton*.
%
%      RO_DIAGRAM('CALLBACK',hObject,eventData,handles,...) calls the local
%      function named CALLBACK in RO_DIAGRAM.M with the given input arguments.
%
%      RO_DIAGRAM('Property','Value',...) creates a new RO_DIAGRAM or raises the
%      existing singleton*.  Starting from the left, property value pairs are
%      applied to the GUI before RO_Diagram_OpeningFunction gets called.  An
%      unrecognized property name or invalid value makes property application
%      stop.  All inputs are passed to RO_Diagram_OpeningFcn via varargin.
%
%      *See GUI Options on GUIDE's Tools menu.  Choose "GUI allows only one
%      instance to run (singleton)".
%
% See also: GUIDE, GUIDATA, GUIHANDLES

% Edit the above text to modify the response to help RO_Diagram

% Last Modified by GUIDE v2.5 21-Jan-2008 14:18:35

% Begin initialization code - DO NOT EDIT
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
   'gui_Singleton',  gui_Singleton, ...
   'gui_OpeningFcn', @RO_Diagram_OpeningFcn, ...
   'gui_OutputFcn',  @RO_Diagram_OutputFcn, ...
   'gui_LayoutFcn',  [] , ...
   'gui_Callback',   []);
if nargin && ischar(varargin{1})
   gui_State.gui_Callback = str2func(varargin{1});
end

if nargout
   [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
   gui_mainfcn(gui_State, varargin{:});
end
% End initialization code - DO NOT EDIT

% --- Executes just before RO_Diagram is made visible.
function RO_Diagram_OpeningFcn(hObject, eventdata, handles, varargin)
% This function has no output args, see OutputFcn.
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% varargin   command line arguments to RO_Diagram (see VARARGIN)

% Initialize global variables:
```

Appendix C RO UI Program Code

```
global HP;
global LP_StartT;
global ss_7;
global LP;
HP=0;
LP_StartT=0;
ss_7=0;
LP=0;
%handles.hp=0;

t=timerfind;delete(t);
% Choose default command line output for RO_Diagram
handles.output = hObject;
handles.evt_index=1;
% Update handles structure
guidata(hObject, handles);

% Start the steady_state timer
handles.steady_state = timer('period',.25);
set(handles.steady_state,'ExecutionMode','fixedrate','StartDelay',2,'BusyMode','drop');
set(handles.steady_state,'timerfcn',{@MyTimerFcn_1, hObject});
start(handles.steady_state);

% Update
guidata(hObject, handles);

% Open initial figure (this is the system diagram)
ax4=axes('Position',[0.1  .4  .8  .5]);
[x,map] = imread('Final Gui RO','png');
image(x)
set(gca,'visible','off')



% UIWAIT makes RO_Diagram wait for user response (see UIRESUME)
% uiwait(handles.figure1);



% --- Outputs from this function are returned to the command line.
function varargout = RO_Diagram_OutputFcn(hObject, eventdata, handles)
% varargout  cell array for returning output args (see VARARGOUT);
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Get default command line output from handles structure
varargout{1} = handles.output;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%
% This is the create functions for the unicontrol objects

% --- Executes on button press in infoA.
function infoA_Callback(hObject, eventdata, handles)
```

```
% hObject    handle to infoA (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
Filter_info

function LPpump_Callback(hObject, eventdata, handles)
% hObject    handle to LPpump (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of LPpump as text
%        str2double(get(hObject,'String')) returns contents of LPpump as a double


% --- Executes during object creation, after setting all properties.
function LPpump_CreateFcn(hObject, eventdata, handles)
% hObject    handle to LPpump (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

function HPpumpA_Callback(hObject, eventdata, handles)
% hObject    handle to HPpumpA (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of HPpumpA as text
%        str2double(get(hObject,'String')) returns contents of HPpumpA as a double


% --- Executes during object creation, after setting all properties.
function HPpumpA_CreateFcn(hObject, eventdata, handles)
% hObject    handle to HPpumpA (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end


function bypassA_Callback(hObject, eventdata, handles)
% hObject    handle to bypassA (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
```

```
% Hints: get(hObject,'String') returns contents of bypassA as text
%        str2double(get(hObject,'String')) returns contents of bypassA as a double


% --- Executes during object creation, after setting all properties.
function bypassA_CreateFcn(hObject, eventdata, handles)
% hObject    handle to bypassA (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end


function HPpumpB_Callback(hObject, eventdata, handles)
% hObject    handle to HPpumpB (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of HPpumpB as text
%        str2double(get(hObject,'String')) returns contents of HPpumpB as a double


% --- Executes during object creation, after setting all properties.
function HPpumpB_CreateFcn(hObject, eventdata, handles)
% hObject    handle to HPpumpB (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end


function bypassB_Callback(hObject, eventdata, handles)
% hObject    handle to bypassB (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of bypassB as text
%        str2double(get(hObject,'String')) returns contents of bypassB as a double


% --- Executes during object creation, after setting all properties.
function bypassB_CreateFcn(hObject, eventdata, handles)
% hObject    handle to bypassB (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called
```

```
% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end


function system_Callback(hObject, eventdata, handles)
% hObject    handle to system (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of system as text
%        str2double(get(hObject,'String')) returns contents of system as a double


% --- Executes during object creation, after setting all properties.
function system_CreateFcn(hObject, eventdata, handles)
% hObject    handle to system (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

function pressureA_Callback(hObject, eventdata, handles)
% hObject    handle to pressureA (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of pressureA as text
%        str2double(get(hObject,'String')) returns contents of pressureA as a double


% --- Executes during object creation, after setting all properties.
function pressureA_CreateFcn(hObject, eventdata, handles)
% hObject    handle to pressureA (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

% --- Executes on button press in reset.
function reset_Callback(hObject, eventdata, handles)
% hObject    handle to reset (see GCBO)
```

```
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
set (handles.system,'String',' ');
set (handles.system,'BackgroundColor','yellow');


function steady_Callback(hObject, eventdata, handles)
% hObject    handle to steady (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of steady as text
%        str2double(get(hObject,'String')) returns contents of steady as a double


% --- Executes during object creation, after setting all properties.
function steady_CreateFcn(hObject, eventdata, handles)
% hObject    handle to steady (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor'))
   set(hObject,'BackgroundColor','white');
end

% --- Executes on button press in drawing.
function drawing_Callback(hObject, eventdata, handles)
% hObject    handle to drawing (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

open RO_drawing.pdf;

% --- Executes on button press in info2.
function info2_Callback(hObject, eventdata, handles)
% hObject    handle to info2 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

ready_to_start

% --- Executes on button press in sys_diagram.
function sys_diagram_Callback(hObject, eventdata, handles)
% hObject    handle to sys_diagram (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

open RC7000'Plus Tech Manual Rev A-MAR07.pdf';

% --- Executes on button press in archive.
function archive_Callback(hObject, eventdata, handles)
```

```
% hObject    handle to archive (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

Ginzu_R0;
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function auto_load(hObject, eventdata, handles,pathname,filename)

global HP;
global LP_StartT;
global ss_7;
global LP;
pathname=strcat(pathname,'/');
longfilename=char(strcat(pathname,filename));
fid=fopen(longfilename,'r');
[path,name,ext,ver] = fileparts(longfilename);


if (ext=='.evt')
   event_t = fgetl(fid);
   try
      ev_time_num=datenum(event_t,'yyyymmdd-HH:MM:SS');
   catch
      ev_time_num=datenum(event_t(5:24),'mmm dd HH:MM:SS yyyy');
   end

   O_file = fgetl(fid);
   glob_index = str2num(fgetl(fid));
   local_index = str2num(fgetl(fid));
   event_class = fgetl(fid);
   class=str2num(fgetl(fid));
   window_size = str2num(fgetl(fid));
   P = zeros(window_size,1);
   Q = zeros(window_size,1);

   for i=1:window_size
      P(i)=str2num(fgetl(fid));
   end

   for i=1:window_size
      Q(i)=str2num(fgetl(fid));
   end

   ev_min=fix(glob_index*60/432000);
   ev_sec=fix((glob_index-ev_min*120*60)/120);
   event_time=(num2str(fix(glob_index*60/432000*100)/100));
   event_time_min=num2str(ev_min);
   event_time_sec=num2str(ev_sec);

   done=fclose('all');
   % delete (longfilename);
```

```
   movefile(longfilename,'archive/');
   %disp(['Processing ' event_class]);



   switch(class)
      case 1
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
   % set threshold value for p to start looking for steady state
if P(local_index+3)-P(local_index-3) < 20 & P(local_index+3)-P(local_index-3) > 10 %This is set prevent
noise from being classified as pump start or hp start from being classified as lp start
for n = 1:length(P)
   if P(n) < 5  % This is arbitrarily set for now
   else
      event=1;
      break
   end
end

if event ==1
   for m = 1:length(P)-20-n
      avg(m) = [mean(P((m+n-1):(m+n+20-1)))];
   end
   for j = 1:length(avg)
      min_P = min(avg(j:j+20));
      max_P = max(avg(j:j+20));

% set threshold for what you are defining as steady state

      if (avg(j)*1.01>max_P) & (avg(j)*.99<min_P) | j+20 == length(avg)
         break
      end
          ss_index = j+n-1;
          time = (ss_index-n)/120;
   end
end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
fid = fopen('times.txt','a+');
fprintf(fid,'%s %30.8f\n',O_file,time);
status = fclose(fid);

         set (handles.LPpump,'String','ON');
         set (handles.LPpump,'BackgroundColor','green');
         LP=1;
         LPstartT = time;
            if time >2
               set (handles.system,'String','Check micron filter condition')
               set (handles.system,'BackgroundColor','red')
            end

         set (handles.steady,'String',LPstartT);
```

```
        if P(local_index+3)-P(local_index-3) > 50
           set (handles.HPpumpA,'String','ON');
           set (handles.HPpumpA,'BackgroundColor','green');
           HP=HP+1;
           set (handles.system,'String','A high pressure pump start has been detected without the low
pressure pump running');
           set (handles.system,'BackgroundColor','red');
        end
end

     case 7
        if LP==1
        ss_7=1;
        set (handles.pressureA,'String','Ready to Start');
        set (handles.pressureA,'BackgroundColor','green');
        end

     case 2
        if (HP==0)
           set (handles.HPpumpA,'String','ON');
           set (handles.HPpumpA,'BackgroundColor','green');
        else
           set (handles.HPpumpB,'String','ON');
           set (handles.HPpumpB,'BackgroundColor','green');
        end

        HP=HP+1

        if (ss_7==0)
           set (handles.system,'String','High pressure pump was started before steady state was
completely reached');
           set (handles.system,'BackgroundColor','red');
        end

     case 3
        set (handles.LPpump,'String','OFF');
        set (handles.LPpump,'BackgroundColor','white');
        LP=0;
        ss_7=0;
        set (handles.pressureA,'String','Not Ready');
        set (handles.pressureA,'BackgroundColor','red');

     case 4
        if (HP==1)
           set (handles.HPpumpA,'String','OFF');
           set (handles.HPpumpA,'BackgroundColor','white');
           set (handles.HPpumpB,'String','OFF');
           set (handles.HPpumpB,'BackgroundColor','white');
           set (handles.bypassA,'String','Open');
           set (handles.bypassA,'BackgroundColor','white');
           set (handles.bypassB,'String','Open');
           set (handles.bypassB,'BackgroundColor','white');
        else
           set (handles.HPpumpB,'String','OFF');
```

```
        set (handles.HPpumpB,'BackgroundColor','white');
        set (handles.bypassB,'String','Open');
        set (handles.bypassB,'BackgroundColor','white');
    end
    HP=HP-1
    if (P(1199)<.2)
        HP=0;
        set (handles.LPpump,'String','OFF');
        set (handles.LPpump,'BackgroundColor','white');
        set (handles.HPpumpA,'String','OFF');
        set (handles.HPpumpA,'BackgroundColor','white');
        set (handles.HPpumpB,'String','OFF');
        set (handles.HPpumpB,'BackgroundColor','white');
        set (handles.bypassA,'String','Open');
        set (handles.bypassA,'BackgroundColor','white');
        set (handles.bypassB,'String','Open');
        set (handles.bypassB,'BackgroundColor','white');
        set (handles.pressureA,'String','Not Ready');
        set (handles.pressureA,'BackgroundColor','red');
    end

case 5
    HP=0;
    ss_7=0;
    LP=0;
    set (handles.LPpump,'String','OFF');
    set (handles.LPpump,'BackgroundColor','white');
    set (handles.HPpumpA,'String','OFF');
    set (handles.HPpumpA,'BackgroundColor','white');
    set (handles.HPpumpB,'String','OFF');
    set (handles.HPpumpB,'BackgroundColor','white');
    set (handles.bypassA,'String','Open');
    set (handles.bypassA,'BackgroundColor','white');
    set (handles.bypassB,'String','Open');
    set (handles.bypassB,'BackgroundColor','white');
    set (handles.pressureA,'String','Not Ready');
    set (handles.pressureA,'BackgroundColor','red');

case 6
    if HP<=1
        set (handles.bypassA,'String','Shut');
        set (handles.bypassA,'BackgroundColor','green');
    else
        set (handles.bypassB,'String','Shut');
        set (handles.bypassB,'BackgroundColor','green');
    end

case 8
    set (handles.system,'String','Possible clogged or misaligned strainer');
    set (handles.system,'BackgroundColor','red');

case 9
    if (Q(1199)<.2)
        HP=0;
        set (handles.LPpump,'String','OFF');
```

```
            set (handles.LPpump,'BackgroundColor','white');
            set (handles.HPpumpA,'String','OFF');
            set (handles.HPpumpA,'BackgroundColor','white');
            set (handles.HPpumpB,'String','OFF');
            set (handles.HPpumpB,'BackgroundColor','white');
        end




    end
end



%=========================================================
% --- Timer Callback Function
%=========================================================
function MyTimerFcn_1(steady_state_object,event,hObject)
% increment the number of counts...
handles = guidata(hObject);
handles.file_names=[];
dir_path=pwd;
dir_struct = dir(strcat(dir_path,'/*.evt'));
[sorted_names,sorted_index] = sortrows({dir_struct.name}');
handles.file_names = sorted_names;
handles.is_dir = [dir_struct.isdir];
handles.sorted_index = sorted_index;
%copyfile('*.evt','event/','f');  %This works.
num_files=length(handles.file_names);

for i=1:num_files
    auto_load(hObject,[],handles,pwd,handles.file_names(i));
    % movefile(handles.file_names(i),'archive/');
    % delete (handles.file_names(i));
end


% else if (handles.evt_index>num_files)
%     handles.evt_index=1
% end
guidata(hObject, handles);


% --- Executes on button press in exit.
function exit_Callback(hObject, eventdata, handles)
% hObject    handle to exit (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
close;
closer
```

Table 7-1: B-side LP Pump Start Data

Date			*Time*	D/P B	LP	Temp
snapshot-20070730-150001.1.gz	1.53333333		1.53	4	50	82
snapshot-20070731-110001.1.gz	1.30833333		1.31	3	49	87
snapshot-20070731-230002.1.gz	0.85000000		0.85	4	50	87
snapshot-20070804-050001.1.gz	1.02500000		1.025	4	50	87.9
snapshot-20070804-110001.1.gz	0.88333333		0.88	4	50	87
snapshot-20070807-000001.1.gz	1.00833333		1.008	6	50	87
snapshot-20070807-120001.1.gz	0.99166667		0.992	4	50	86
snapshot-20070807-210001.1.gz	0.79166667		0.71	4	50	87
snapshot-20070808-030001.1.gz	0.72500000		0.73	4	50	86
snapshot-20070808-180001.1.gz	0.70833333		0.71	5	50	87.7
snapshot-20070811-120001.1.gz	0.96666667		0.7	5	49	86
snapshot-20070817-140001.1.gz	0.65000000		0.65	5	50	87
snapshot-20070818-040001.1.gz	0.74166667		0.74	6	50	86
snapshot-20070820-130001.1.gz	0.70000000		0.7	6	50	87
snapshot-20070827-010001.1.gz	0.75000000		0.75	6	46	87
snapshot-20070901-170001.1.gz	0.78333333		0.78	6	46	88
snapshot-20070906-070001.1.gz	0.67500000		0.68	7	50	86
snapshot-20070907-230001.1.gz	1.09166667		1.09	8	50	87
snapshot-20070909-070001.1.gz	1.00000000		1	8	50	87
snapshot-20070913-180001.1.gz	1.43333333		1.43	9	51	67
snapshot-20071106-210001.1.gz	1.95833333		1.95	4	43	75
snapshot-20071108-130001.1.gz	1.05000000		1.05	4	43	76
snapshot-20071108-220001.1.gz	0.87500000		0.875	3	43	78.7
snapshot-20071110-200001.1.gz	0.77500000		0.78	3	42	79.7
snapshot-20071114-070001.1.gz	0.70000000		0.7	6	42	82.5
snapshot-20071114-150001.1.gz	0.88333333		0.88	7	42	82
snapshot-20071116-150001.1.gz	0.75833333		0.76	7	41	81
snapshot-20071119-170001.1.gz	0.81666667		0.82	8	42	80
snapshot-20071120-000001.1.gz	0.90000000		0.9	9	42	80
snapshot-20071120-130001.1.gz	1.03333333		1.03	9	42	82
snapshot-20071130-100002.1.gz	0.83333333		0.8	7	49	81

* Yellow highlighted starts are after long in port layup period

Table 7-2: A-side LP Pump Start Data

Date			*Time*	D/P A	LP	Temp
snapshot-20070730-040001.1.gz	1.46666667		1.47	2	50	82.9
snapshot-20070730-220001.1.gz	0.25000000		0.45	4	50	82
snapshot-20070801-170001.1.gz	0.69166667		0.692	3	62	86
snapshot-20070806-020001.1.gz	0.35000000		0.35	2	50	87
snapshot-20070809-200001.1.gz	0.53333333		0.53	2	50	88
snapshot-20070810-070001.1.gz	0.61666667		0.617	2	50	86
snapshot-20070810-160001.1.gz	0.54166667		0.541	2	50	86
snapshot-20070810-230001.1.gz	0.59166667		0.592	4	50	86
snapshot-20070811-100001.1.gz	0.54166667		0.54	2	50	86
snapshot-20070811-180001.1.gz	0.55833333		0.56	2	50	87
snapshot-20070812-000001.1.gz	0.56666667		0.57	5	50	86
snapshot-20070812-060001.1.gz	0.53333333		0.53	2	50	85
snapshot-20070812-170001.1.gz	0.54166667		0.54	3	50	86
snapshot-20070813-130001.1.gz	0.53333333		0.53	2	50	86
snapshot-20070813-220001.1.gz	0.54166667		0.54	2	50	87
snapshot-20070814-140001.1.gz	0.55833333		0.56	2	50	85
snapshot-20070815-010002.1.gz	0.52500000		0.53	1	50	87
snapshot-20070818-150001.1.gz	0.55000000		0.55	2	50	87
snapshot-20070819-030001.1.gz	0.48333333		0.48	2	50	87
snapshot-20070819-180001.1.gz	0.52500000		0.53	3	49	87
snapshot-20070820-060001.1.gz	0.54166667		0.54	4	50	86
snapshot-20070821-010001.1.gz	0.55000000		0.55	3	45	87
snapshot-20070821-100001.1.gz	0.48333333		0.48	4	45	86
snapshot-20070821-190001.1.gz	0.45000000		0.45	4	46	87
snapshot-20070822-130002.1.gz	0.53333333		0.53	3	45	85
snapshot-20070823-020002.1.gz	0.46666667		0.467	4	46	87
snapshot-20070823-170001.1.gz	0.56666667		0.57	3	45	88
snapshot-20070825-000001.1.gz	0.45000000		0.45	3	46	87
snapshot-20070825-110001.1.gz	0.54166667		0.54	3	46	87
snapshot-20070825-190001.1.gz	0.55833333		0.56	3	45	86
snapshot-20070826-090001.1.gz	0.45833333		0.46	4	45	87
snapshot-20070826-140001.1.gz	0.53333333		0.53	3	45	87
snapshot-20070831-100001.1.gz	0.46666667		0.47	4	46	86
snapshot-20070831-210001.1.gz	0.58333333		0.58	4	46	87
snapshot-20070901-100001.1.gz	0.50000000		0.5	4	46	87
snapshot-20070902-070001.1.gz	0.52500000		0.525	3	45	86
snapshot-20070902-160001.1.gz	0.47500000		0.475	3	45	87
snapshot-20070903-020001.1.gz	0.48333333		0.48	5	45	87
snapshot-20070903-130001.1.gz	0.56666667		0.57	3	45	87
snapshot-20070904-020001.1.gz	0.46666667		0.47	2	45	87
snapshot-20070905-150001.1.gz	0.45833333		0.46	3	50	86
snapshot-20070906-130001.1.gz	0.50000000		0.5	3	49	87
snapshot-20070906-210001.1.gz	0.51666667		0.52	2	50	88
snapshot-20070907-130002.1.gz	0.44166667		0.44	3	50	87
snapshot-20070909-180001.1.gz	0.50833333		0.51	3	50	87

Table 7-3: A-side LP Pump Start Data Continued

Date			*Time*	D/P A	LP	Temp
snapshot-20070910-050001.1.gz	0.45000000		0.45	2	50	87
snapshot-20070910-230001.1.gz	0.46666667		0.47	3	50	87
snapshot-20070912-140001.1.gz	0.49166667		0.49	3	50	83
snapshot-20071106-010002.1.gz	1.23333333		1.23	3	43	59
snapshot-20071107-030001.1.gz	0.35833333		0.36	1	42	76
snapshot-20071115-220001.1.gz	0.57500000		0.58	4	42	81.7
snapshot-20071121-180001.1.gz	0.56666667		0.57	3	42	80
snapshot-20071123-030001.1.gz	0.51666667		0.52	4	42	80
snapshot-20071126-080001.1.gz	0.62500000		0.63	4	42	81.9
snapshot-20071126-150001.1.gz	0.60000000		0.6	2	42	82
snapshot-20071127-000001.1.gz	0.51666667		0.52	3	42	80
snapshot-20071127-180001.1.gz	0.49166667		0.49	4	38	80

Appendix E MATLAB FFT Script

```
% This is simple code to take the fft of a file.
% You must define the data set

p = data % specify data set;
p1 = -p(330000:430000); % specify what points are being manipulated

a1 = detrend(p1); % Removes the mean (dc)

n = 4096*2; % Number of point in the fft
f_sample = 120; % Sample frequency of the data set (prep data)

% Hanning is actually obsolete now but it is the same as the
% hann function.  I basically reduces aliasing in the fft
fft_a1 = fft(a1.*hanning(length(a1)),n);

f = [0:1:(n/2)-1]*(f_sample/n);

plot(f,abs(fft_a1(1:length(fft_a1)/2)))
xlabel('Frequency: Hz');
ylabel('Magnitude');
```

Appendix F MATLAB Filter Script

```
% Bandpass Filter

clear all

% dlmread is a function that allow you to specify exactly what part of a
% data set that you want, it is especially useful for really big files that
% would take an excessive time to load the whole file
ss = dlmread('snapshot-20070509-160001-005-000.evt',' ', [8 0 1207 0]);

% the filter allows you to specify the outer bounds of the range that you
% want to filter out (inner numbers). The outer two numbers specify how
% sharp you want the bounds cut off. Example: [5 8 9 12] -- here you are
% isolating the frequency range between 8 and 9, the filter begins to
% isolate at 5 and 12 (not sharp).
[n,Wn,bta,filtype] = kaiserord( [5 8 9 12], [0 1 0], [0.1 0.01 0.1], 120 );
b = fir1(n, Wn, filtype, kaiser(n+1,bta), 'noscale');
[n,Wn,bta,filtype] = kaiserord( [15 16 17 19], [0 1 0], [0.1 0.01 0.1], 120 );
c = fir1(n, Wn, filtype, kaiser(n+1,bta), 'noscale');
t = [0:1:length(ss)-1]*(1/120);

y=filter(b,1,ss);
z=filter(c,1,ss);

plot (t,y+z)
```