NAVAL POSTGRADUATE SCHOOL

MONTEREY, CALIFORNIA

THESIS

AUTOMATIC WEB-BASED CALIBRATION OF NETWORK-CAPABLE SHIPBOARD SENSORS

by

Charles K. Le

September 2007

Thesis Advisor:	Xiaoping Yun
Second Reader:	Roberto Cristi

REPORT DOCUMENTATION PAGE			*Form Approved OMB No. 0704-0188*
Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instruction, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188) Washington DC 20503.			
1. AGENCY USE ONLY *(Leave blank)*	**2. REPORT DATE** September 2007	**3. REPORT TYPE AND DATES COVERED** Master's Thesis	
4. TITLE AND SUBTITLE Automatic Web-based Calibration of Network-capable Shipboard Sensors			**5. FUNDING NUMBERS**
6. AUTHOR(S) Charles K. Le			
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Naval Postgraduate School Monterey, CA 93943-5000			**8. PERFORMING ORGANIZATION REPORT NUMBER**
9. SPONSORING /MONITORING AGENCY NAME(S) AND ADDRESS(ES) N/A			**10. SPONSORING/MONITORING AGENCY REPORT NUMBER**
11. SUPPLEMENTARY NOTES The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government.			
12a. DISTRIBUTION / AVAILABILITY STATEMENT Approved for public release; distribution is unlimited			**12b. DISTRIBUTION CODE**

13. ABSTRACT (maximum 200 words)

This thesis investigates the feasibility of developing an automatic web-based sensor calibration system with four main objectives. The first objective was to reduce the number of personnel required to calibrate shipboard sensors. The second was to reduce the time required to complete the calibration process. The third was to develop a platform independent and user-friendly interface using the web browser. The fourth was to allow operators to calibrate the sensors remotely from thousands of miles away. This was achieved by using the commercial off the shelf (COTS) products, developing in-house hardware, setting up a web server and developing numerous software programs in Labview and Java languages to allow operators to remotely monitor, and control the calibration process. All communication and control algorithms are handled by two computers. One serves as a web server, equipped with java codes and web pages to interface with an operator. The other serves as a data collector. It collects data from all sensors via the network, passes these data to the web server computer and then to the operator's web browser. It also runs a calibration algorithm on a selected sensor as requested by the user. The two computers communicate with one another via the ship's LAN using UDP packets.

14. SUBJECT TERMS Network-Based Calibration, Wireless LAN, UDP, Labview, Smart Sensors, Pressure Sensors, Tomcat Web Server, Java, Applet, and Servlet.			**15. NUMBER OF PAGES** 135
			16. PRICE CODE
17. SECURITY CLASSIFICATION OF REPORT Unclassified	**18. SECURITY CLASSIFICATION OF THIS PAGE** Unclassified	**19. SECURITY CLASSIFICATION OF ABSTRACT** Unclassified	**20. LIMITATION OF ABSTRACT** UU

NSN 7540-01-280-5500 Standard Form 298 (Rev. 2-89) Prescribed by ANSI Std. 239-18

THIS PAGE INTENTIONALLY LEFT BLANK

AUTOMATIC WEB-BASED CALIBRATION OF NETWORK-CAPABLE SHIPBOARD SENSORS

Charles Khang Duy Le
Lieutenant, United States Navy
B.S. EE, University of Texas at Austin, 1998

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN ELECTRICAL ENGINEERING

from the

NAVAL POSTGRADUATE SCHOOL
September 2007

Author:	Charles K. Le
Approved by:	Xiaoping Yun Thesis Advisor
	Roberto Cristi Second Reader
	Jeffrey B. Knorr Chairman, Department of Electrical Engineering

THIS PAGE INTENTIONALLY LEFT BLANK

ABSTRACT

This thesis investigates the feasibility of developing an automatic web-based sensor calibration system with four main objectives. The first objective was to reduce the number of personnel required to calibrate shipboard sensors. The second was to reduce the time required to complete the calibration process. The third was to develop a platform independent and user-friendly interface using the web browser. The fourth was to allow operators to calibrate the sensors remotely from thousands of miles away. This was achieved by using the commercial off the shelf (COTS) products, developing in-house hardware, setting up a web server and developing numerous software programs in Labview and Java languages to allow operators to remotely monitor, and control the calibration process. All communication and control algorithms are handled by two computers. One serves as a web server, equipped with java codes and web pages to interface with an operator. The other serves as a data collector. It collects data from all sensors via the network, passes these data to the web server computer and then to the operator's web browser. It also runs a calibration algorithm on a selected sensor as requested by the user. The two computers communicate with one another via the ship's LAN using UDP packets.

THIS PAGE INTENTIONALLY LEFT BLANK

TABLE OF CONTENTS

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF FIGURES

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF TABLES

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF ACRONYMS AND ABBREVIATIONS

ASCII	American Standard Code for Information Interchange
CBM	Condition Based Maintenance
CPU	Central Processing Unit
DHCP	Dynamic Host Configuration Protocol
DS	Data Socket
ERP	Enterprise Resource Planning
HM&E	Hull, Mechanical and Electrical
HMI	Human Machine Interface
HTTP	Hypertext Transfer Protocol
I/O	Input/Output
ICAS	Integrated Condition Assessment System
IEEE	International Electrical and Electronic Engineering
IIS	Internet Information Server
ISI	Industrial Sensor Instrument
ISIC	Immediate Superior In Charge
LAN	Local Area Network
LCD	Liquid Crystal Display
MAC	Medium Access Control
MCS	Machinery Control Systems
NCAP	Network Capable Application Processor
NEMIAS	Navy Enterprise Maintenance Automated Information System
NPS	Naval Postgraduate School
PC	Personal Computer
PPC	Portable Pressure Calibrator
PPT	Precise Pressure Transducer
RCM	Reliability Centered Maintenance
RS-232	Recommended Standard 232
SubVI	Subsidiary Virtual Instrument
TCP	Transport Control Protocol
UDP	User Datagram Protocol

URL	Uniform Resource Locator
VI	Virtual Instrument
VPN	Virtual Private Network
WAN	Wide Area Network
W-LION	Wireless Input/Output Node

EXECUTIVE SUMMARY

This thesis reports on the development of an automatic web-based sensor calibration system to implement a new calibration concept. This new sensor calibration system was developed with four objectives. First is to reduce the personnel requirement by at least 50 percent. Second is to reduce the calibration time as much as 75 percent compared to the current calibration procedure used by the US Navy. Third is to further automate the calibration process, and provide a user friendly, web-based GUI to guide an operator through the calibration process. Fourth is to take advantage of the web technologies to allow operators to calibrate any selected shipboard sensor from anywhere on the world-wide-web.

The automatic web-based calibration system is implemented by using the Commercial-off-the-Shelf (COTS) products such as the 3COM router, Netgear PoE switch, Vlinx RS-232 to Ethernet adaptor, and Dell desktop computers. The Vlinx RS-232 to Ethernet adaptor is used to interface sensors with the ship's LAN. The two personal computers are used to interface with remote users (the web server computer) and to automate the calibration process (the calibrating computer). Extensive use of Labview code on the calibrating computer allows it to collect sensor readings from the shipboard LAN, forward the reading to the web server computer, receive calibration commands from the web server computer and execute the calibration process as requested by the user. The web server computer is configured with the Tomcat web server to host the sensor calibration web pages. The calibration capabilities of these web pages are powered by the Java applets and servlets located on the web server computer. These Java codes allow the web server computer to forward sensor readings to an operator and send calibration commands from operators to the calibrating computer. The calibration process is executed automatically by the calibrating computer. No manual intervention is required by operators. The process, therefore, takes much less time than the current calibration process.

THIS PAGE INTENTIONALLY LEFT BLANK

ACKNOWLEDGMENTS

The author would like to acknowledge the financial support of Corona Division, Naval Surface Warfare Center (NSWC), for allowing the purchase of equipment used in this thesis, and to thank the following people for technical and moral support during the course of this thesis effort: Prof. Xiaoping Yun and James Calusdian at NPS, Randy Rupnow and Hal Glick at NSWC Corona.

The author also would like to thank Professor Xiaoping Yun for all his guidance during the thesis. His assistance and advices had helped to overcome challenges along the way. It has been a great learning experience to discover new techniques and solutions to implement this new calibration concept. A lot of valuable skills and knowledge have been gained. Thank you, sir, for this excellent thesis opportunity.

The author also would like to acknowledge James Calusdian for designing and constructing the pressure pipe system on the virtual ship, solenoid controlling circuits, and providing necessary resources to accomplish the thesis work. His strong knowledge of engineering principles and circuit design quickly turned ideas into reality. His assistance was crucial to the successful completion of this thesis.

THIS PAGE INTENTIONALLY LEFT BLANK

I. INTRODUCTION

A. BACKGROUND

In a modern DDG, there are approximately 3,742 hull, mechanical, and electrical (HM&E) sensors to monitor and control systems on the ship. Over times, the readings from these sensors tend to deviate from the actual reading and therefore they need periodic calibration. Most of the sensors are pressure and temperature sensors, switches or gauges. Of these, 1,189 are independent visual types of sensors and 1,480 are integral parts of shipboard Machinery Control Systems (MCS), which can only be read at system consoles located far from the original sensor location [1].

Calibrating all these sensors would require thousands of man-hours from specialized technicians to collect and analyze data from sensors. As an example, at least two technicians are required to calibrate a sensor. The calibration process takes at least 30 minutes to one hour for each sensor. The man-hours and system down time increase dramatically as the complexity of the sensor is increased [1].

With the emphasis being placed on reducing crew size, more sensors are needed for automated systems to reduce the man-hours lost. This increase in number of sensors and reduction in number of maintaining personnel present a major challenge in keeping shipboard sensor calibrated. [2]. A new calibration process or approach is needed to overcome this challenge.

B. CURRENT CALIBRATION PROCESS

The current sensor calibration process is labor intensive, time consuming and requires at least two technicians to complete the task. The displays of the sensor being calibrated and the reference sensor are physically separated. Two technicians are required to read the displays at the two separate locations. Several readings of the two sensors are collected using hand held radio communication between the two technicians. The two sets of data are then plotted for comparison, and calibration constants are derived from the plot. The calibration constants are then applied on the sensor's signal conditioner. The calibration process repeats until the calibrated sensor and the standard sensor display readings within a specified tolerance.

C. OBJECTIVES AND APPROACH

This research work was conducted with four objectives. First is to reduce the personnel requirement by at least 50 percent. Second is to reduce the calibration time as much as 75 percent compared to the current calibration procedure used by the US Navy. Third is to further automate the calibration process, and provide a user friendly, web-based GUI to guide an operator through the calibration process. Fourth is to take advantage of the web technologies to allow operators to calibrate any selected shipboard sensor from anywhere on the world-wide-web.

To achieve the objectives listed above, the current computer and networking technologies were used to maximize automation in the calibration process and allow an operator to remotely calibrate a shipboard sensor. Available commercial off-the shelf (COTS) components were also used to push sensor readings on the ship's LAN where they were collected by a computer for calibration. This computer also controlled an electrical pump and an opening valve to adjust the pressure inside the pipe system during the calibration process. A second computer was used to host a web site to interface with the operator from a remote location or from within the ship. The two computers communicate with one another via the ship's LAN. The system was designed to allow an operator to calibrate a sensor by simply making a few clicks on a web page.

D. RELATED WORK

Previous work on developing a close-loop sensor calibration system was conducted by two former students at NPS, Steven Joseph Perchalski [3] and Eusébio Pedro da Silva [4]. The guidance and technical support of this work are provided by Mr. Randy Rupnow at NAVSEA Corona, and Professor Xiaoping Yun at NPS. Their work consisted of a number of Labview programs communicating with sensors via wireless Ethernet, a RS-232 interface or a Bluetooth interface. These Labview programs collected sensor readings and automatically computed the new calibration constants using a least squares fitting method. This was a closed-loop process that wirelessly calibrated shipboard sensors and reduced the number of personnel and time required to complete the task.

E. BENEFIT OF CONCEPT TO THE NAVY

This research work can potentially be beneficial to the maintenance of US Navy Ships. By maximizing automation and providing a simple user interface, more sensors can be calibrated with much less time and man-power. The user interface is so simple that average sailors can learn to use it in a short period of time. Thus, more sailors can be trained to calibrate their ship's sensors. Therefore, sensors can be calibrated more frequently and will provide more accurate reading. The end result will be a reduction in equipment failure due to sensor errors. This research demonstrates that remote and automatic sensor calibration is achievable. It also shows that thousands of copper wires connecting sensors to a control station can be replaced with a few Ethernet cables. This will save significant space, weight and cost incurred by running sensor cables. Networking also provides the same functionalities with even more redundant routes. More redundancy would greatly increase a ship's survivability.

F. THESIS OUTLINE

This first chapter is the introduction to the thesis, while the second chapter covers the problem statement and proposed approaches. The third chapter concentrates on the hardware components, and it discusses their configurations and functions. The fourth chapter examines the software designed, and discusses the overall functionality and implementation details. The fifth chapter presents the results that were found and discusses what worked and what failed. Finally, the sixth chapter has the conclusions and recommendation for future research. This includes what was accomplished by this thesis and any future work that may be needed. The appendix contains the Java codes that were designed and used for this thesis.

THIS PAGE INTENTIONALLY LEFT BLANK

II. OVERVIEW OF RESEARCH GOALS AND CONTRIBUTIONS

A. PROBLEM STATEMENT AND RESEARCH GOALS

As stated in the introduction, there is a need for a new process to calibrate sensors. The plan was to develop a new process that achieves the following: First is to reduce the personnel requirement by at least 50 percent. Second is to reduce the calibration time as much as 75 percent compared to the current calibration procedure used by the US Navy. Third is to further automate the calibration process, and provide a user friendly, web-based GUI to guide an operator through the calibration process. Fourth is to take advantage of the web technologies to allow operators to calibrate any selected shipboard sensor from anywhere on the world-wide-web. This would allow sensors to be calibrated while the ship is away from her homeport. The approach was to use the web browser to remotely initiate an automatic calibration process on a ship. The goals were to be achieved while following these constraints: the use of current commercial-off-the-shelf (COTS) networking technology, which can be easily added to new ship construction and could be retro-fitted onto older ships. In Figure 1, analog readings from analog sensors are sampled and stuffed into the TCP packets by the Network Capable Application Processors (NCAPs). These packets are transmitted wirelessly to a wireless gateway to the ship's LAN. Therefore, sensor readings can be read by interfacing with the ship's LAN.

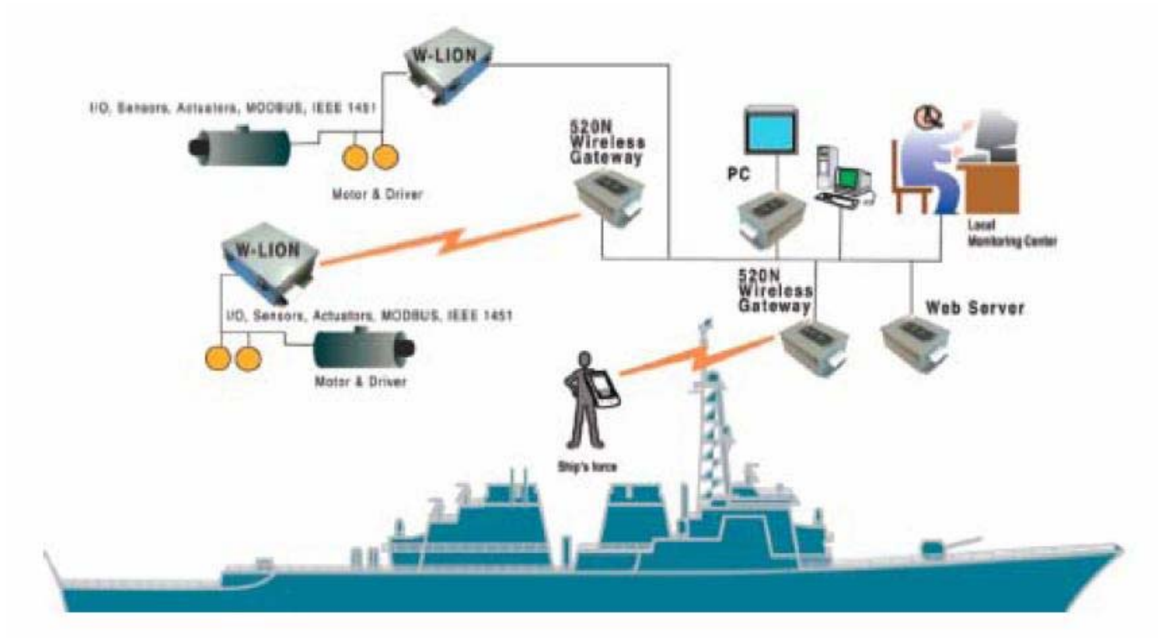
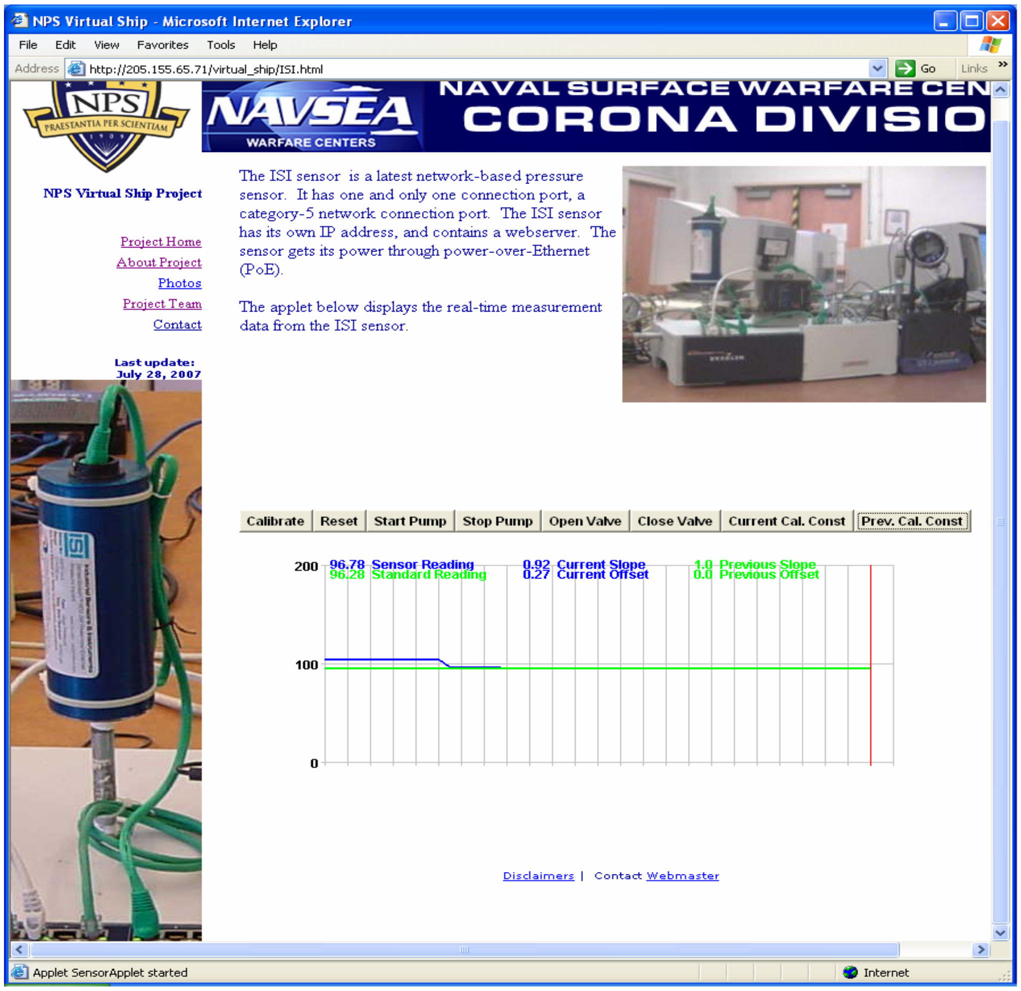


Figure 1. Diagram of a Ship's Network using Wireless NCAP and Gateway [From Ref. 1].

B. OVERVIEW OF CONTRIBUTIONS

1. Reduce Personnel Requirement by at least 50 Percent

The proposed approach reduces the personnel needed from two or more to one by displaying the target sensor reading and the standard sensor reading on the same screen. This removes the need for an additional person needed to read the data from the remote display. As shown in Figure 2, a technician can remotely initiate an automatic sensor calibration process by clicking on the "Calibrate" command button. With multiple automatic calibration systems set up on a ship, a single technician can remotely calibrate multiple sensors on different spaces in a ship or even on different ships simultaneously.



The ISI sensor is a latest network-based pressure sensor. It has one and only one connection port, a category-5 network connection port. The ISI sensor has its own IP address, and contains a webserver. The sensor gets its power through power-over-Ethernet (PoE).

The applet below displays the real-time measurement data from the ISI sensor.

Figure 2. GUI of a Web-based Automatic Calibration System.

2. Reduce Calibration Time by 75 Percent

To achieve the goal of reducing time required, the plan is to make it easier to take measurements and change the calibration constants. All sensor readings are pushed on the ship's LAN using Power-Over-Ethernet (PoE), NCAP, or Vlinx RS-232-to-Ethernet adapter. The calibrating computer collects sensors reading through the ship's LAN, uses an electrical pump to control the pressure on the calibrating sensor, collects pressure data points, runs the calibration algorithm, and applies calibrating constants to the sensor. The whole process is performed by the calibrating computer. The technician, no longer needs to use the hand pump to apply different pressure on the sensors, nor calculating and applying the calibrating constants manually.

3. Enhance User-friendly Interface

One important factor to reduce time and required personnel for a calibration process is to design a user friendly interface that requires minimal input from the technician, and provides guidance throughout the calibration process. Anyone who has used a web browser such as Internet Explorer would be able to get a sensor calibrated using this new calibration system. Figure 3 shows the diagram of a ship's sensor network displayed on a web browser. By clicking on the image of a sensor, a new page is displayed that allows the technician to calibrate the selected sensor, as shown in Figure 2. To start calibration, the technician will only need to click on the Calibrate button on the upper left of the strip chart.

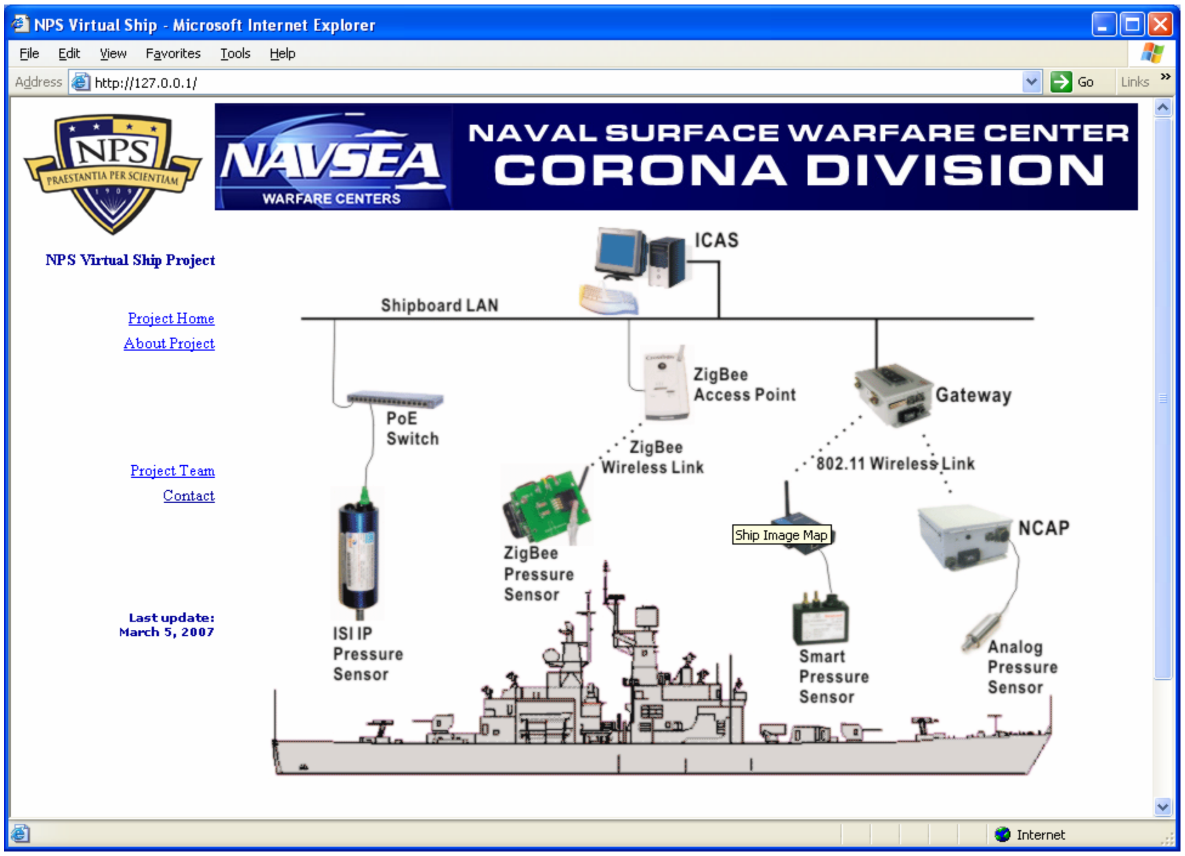


Figure 3. Diagram of a Ship's Sensor Network Displayed on a Web Browser.

4. Automatic and Remote Calibration

With the new calibration process, the technician only needs to click one button (Calibrate button) as shown in Figure 2. The automatic calibration process will, then, be initiated by a computer. The computer will control the electrical pump and open the valve to bring the pressure to the desired point. It, then, collects both the readings of the target sensor and the standard sensor over the ship's LAN. When enough data points are collected, the computer performs the calibration algorithm and sends the calibration constants to the technician via the ship's network and the internet. During the whole process, the reading of the calibrating sensor and the standard sensor are displayed in the strip chart format on the technician's computer. This removes the need to have an additional person needed to read the data from the remote display or use the hand pump to obtain the desired pressure during the data collection process.

C. SUMMARY

In summary the research goal is to demonstrate that an automatic web-based sensor calibration process can be remotely initiated through the internet by a technician using a web browser, and achieve at least 50% reduction in personnel and 75% reduction in calibrating time. These were accomplished by using current networking technology offered by commercial-off-the-shelf (COTS) products. By using the computer, electrical pump, ship's LAN, and the internet, the calibration process can be automated and initiated remotely with a click of a button. In order to prove and test the theory, a virtual ship was set up in the lab by connecting different sensors to pressurized pipes. The research focuses on pressure sensors and uses the portable pressure calibrator (PPC) as the standard sensor. The next chapter examines the hardware that was used in the thesis.

THIS PAGE INTENTIONALLY LEFT BLANK

III. HARDWARE DESCRIPTION AND DISCUSSION

A. INTRODUCTION TO HARDWARE

This chapter discusses the different types of hardware used in this research. Among these hardware are the 3eTI NCAP, the 3eTI Gateway, the Crystal XP2 Digital Test Gauge (PPC), the calibrating computer, the web server computer, Vlinx RS232-to-Ethernet adapters, Omega pressure sensor, Honeywell smart sensor, ISI sensor (the blue sensor) and the networking gears used to implement the ship's LAN (a 3COM router, a Linksys router and a Netgear PoE switch). Figure 4 shows the partial implementation of a virtual ship. The four sensors shown from left to right in Figure 4 are the ISI sensor, the Honeywell Precision Pressure Transducer (PPT), the Zigbee sensor and the Omega analog sensor.

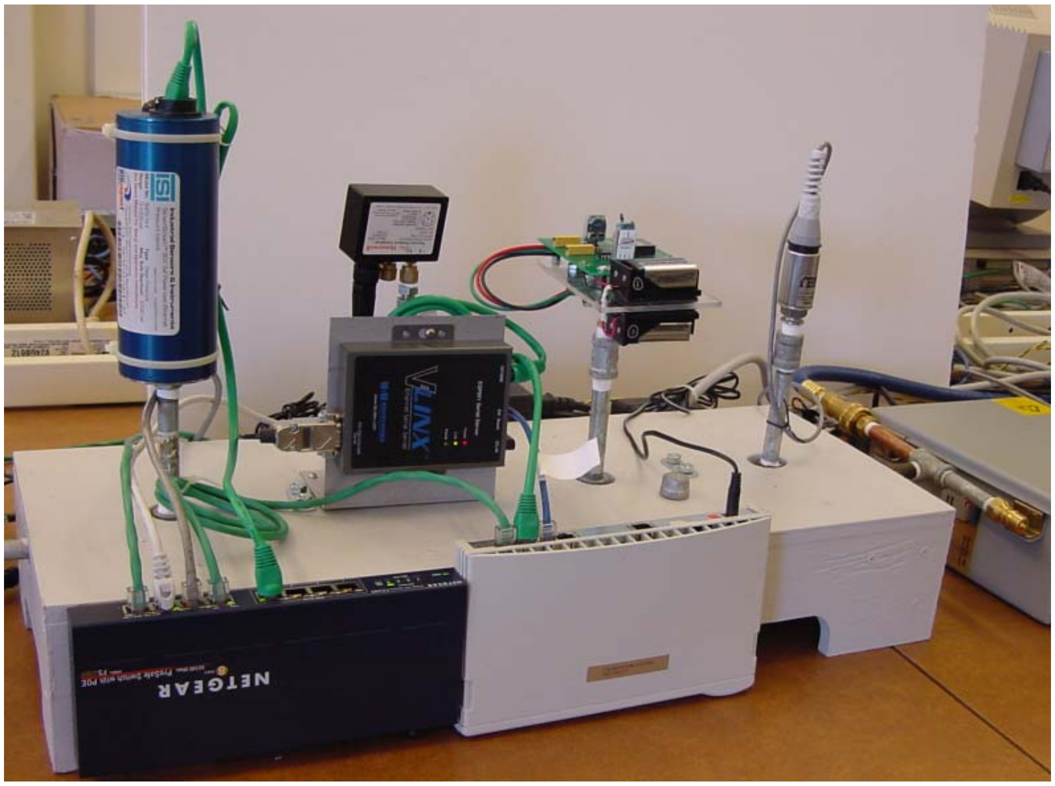


Figure 4. Connection of Pressure Sensors to the Pressurized Pipe System on a Virtual Ship.

B. SHIPBOARD NETWORK

The ship's LAN is implemented using a combination of a 3COM router and a Linksys wireless router to provide wired and wireless LAN access to sensors and other network clients, as shown in Figure 5. The 3COM router is also connected to a Netgear switch to provide PoE interfaces to the ISI sensor. The green Vlinx RS232-to-Ethernet adapters are used to convert an RS-232 interfaces to LAN interfaces. For analog sensors, the NCAP I/O box is used to insert sensors' analog readings to TCP packets and send them to a destination via the ship's LAN. These interface conversions allows the calibrating computer to communicate with all shipboard sensors via a single Ethernet interface.

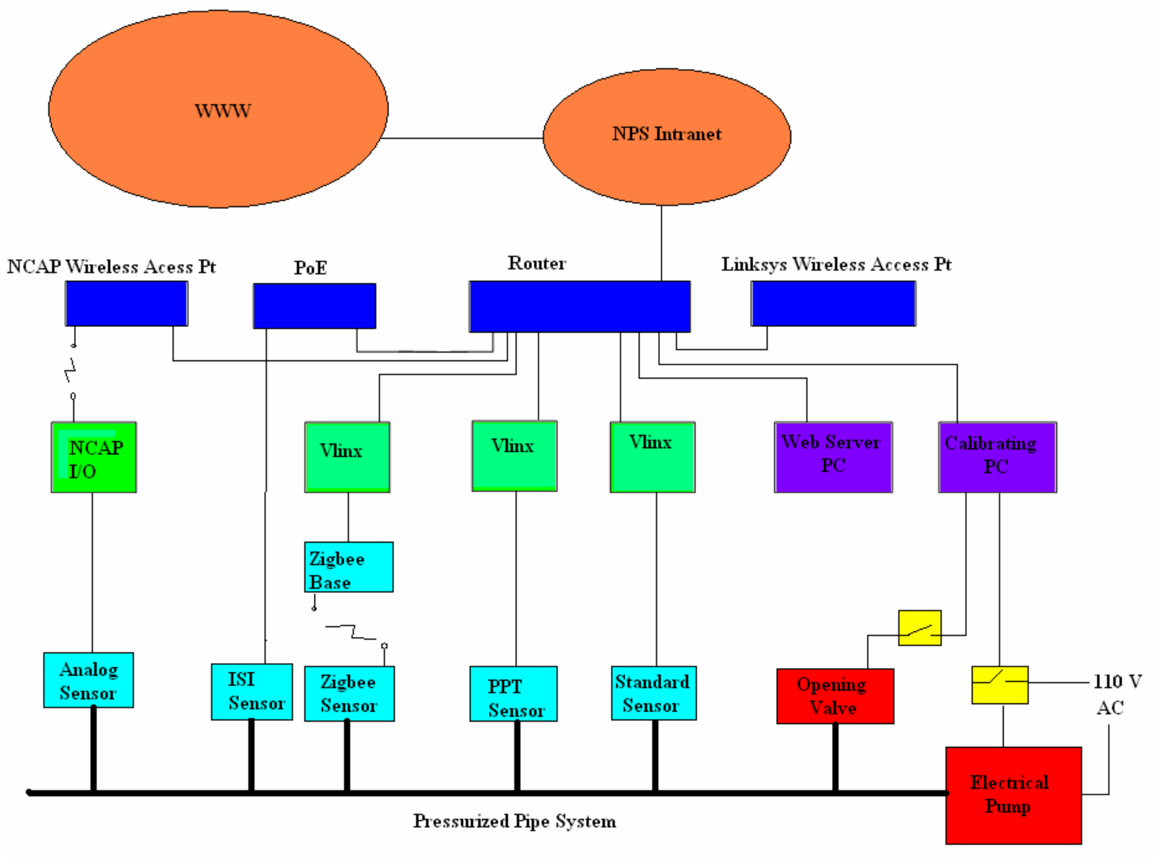


Figure 5. Diagram of a Ship's Integrated Sensor Network.

1. 3COM Router

a. Description

The brain of the ship's LAN is the 3COM router. It is equipped with a firewall and other security measures to ensure a secured data exchange between the ship's LAN and the WAN. It also assigns an IP address to every network interface such as the Vlinx RS232-to-Ethernet adapter, the ISI sensor, the NCAP box or any computer connected to the ship's LAN. It has four LAN ports and one WAN port. The WAN port of the 3COM router is connected to the school network and is configured with the static IP address of 205.155.65.71. The LAN ports are connected to the sensors on the virtual ship. As more sensors are added to the system, there will be a demand for a higher number of LAN ports. More LAN ports can be added by connecting one of the LAN ports to a port on another switch or hub, as shown in Figures 6-7.

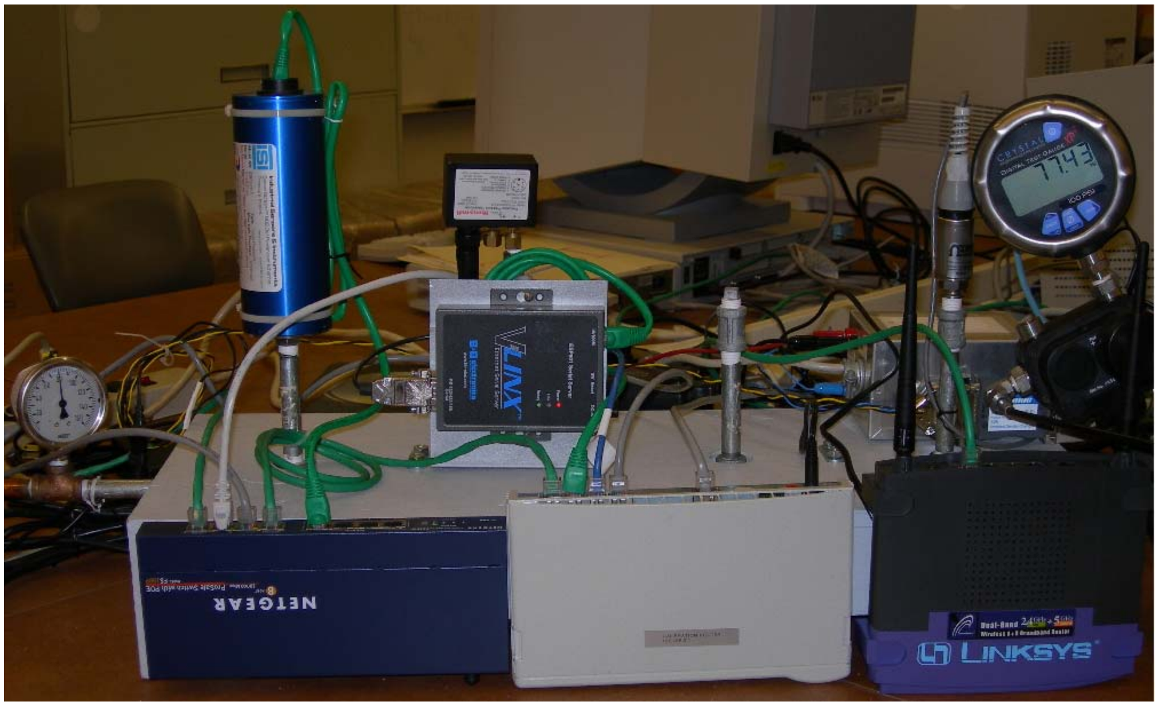


Figure 6. The Virtual Ship's LAN is made of a 3COM Router, a Netgear Switch, and a Linksys Wireless Router.

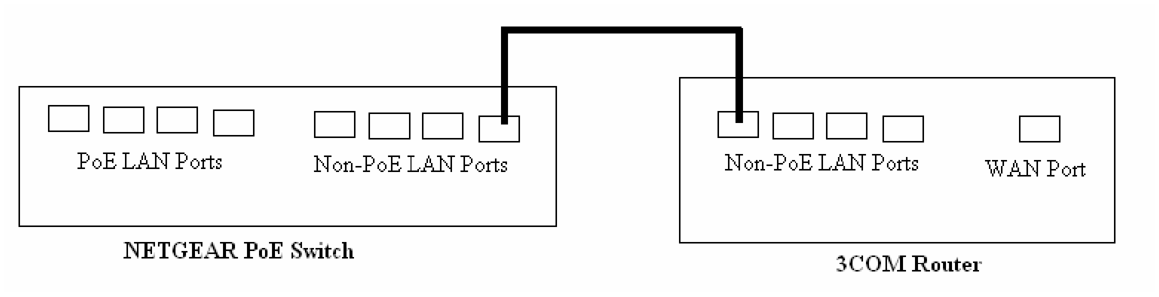
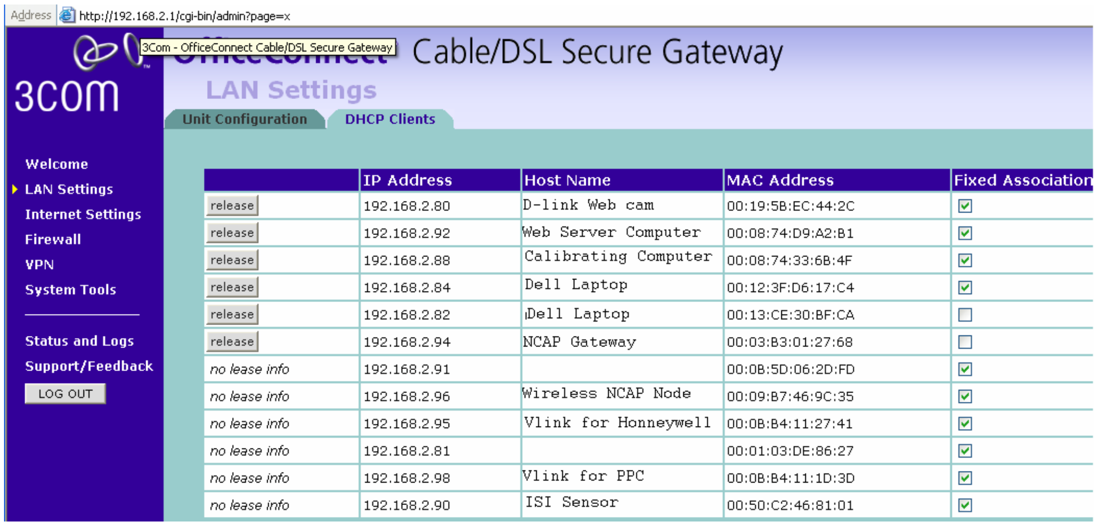


Figure 7. Add PoE and LAN to the Virtual Ship's LAN by Connecting the Router's LAN port to the Netgear Switch's non-PoE LAN port.

b. Configuration

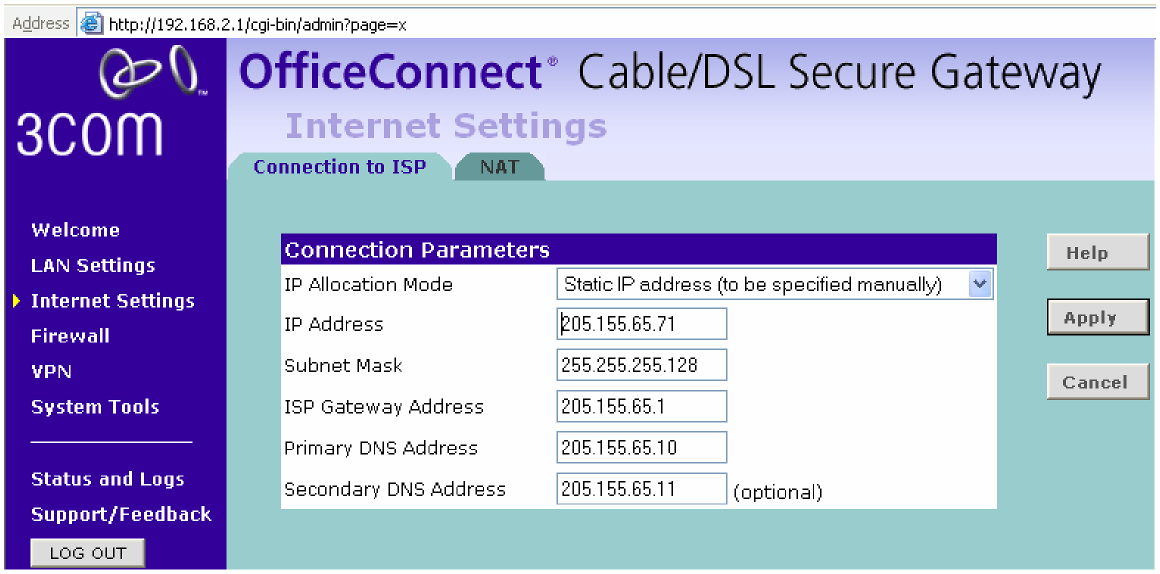
The 3COM router is configured for DHCP operation with the IP address of 192.168.2.1. The router's different LAN configuration tools can be accessed by typing http://192.168.2.1 into the URL box of a web browser, as shown in Figures 8-10. For easier configuration, a client (sensor, Vlinx box, or a computer) should be also configured with DHCP when it is first connected to the 3COM router. DHCP configuration on both the 3COM router and the client would allow the router to assign an IP address to the client automatically. Figure 8 shows all LAN clients that have been assigned a DHCP address. To keep these IP address assignments permanent, one only need to select the corresponding box in the Fixed Association column of the table. If the client does not support DHCP operation, then its IP address and MAC address must be added manually to the 3COM router's client address table.



	IP Address	Host Name	MAC Address	Fixed Association
release	192.168.2.80	D-link Web cam	00:19:5B:EC:44:2C	✓
release	192.168.2.92	Web Server Computer	00:08:74:D9:A2:B1	✓
release	192.168.2.88	Calibrating Computer	00:08:74:33:6B:4F	✓
release	192.168.2.84	Dell Laptop	00:12:3F:D6:17:C4	✓
release	192.168.2.82	Dell Laptop	00:13:CE:30:BF:CA	☐
release	192.168.2.94	NCAP Gateway	00:03:B3:01:27:68	☐
no lease info	192.168.2.91		00:0B:5D:06:2D:FD	✓
no lease info	192.168.2.96	Wireless NCAP Node	00:09:B7:46:9C:35	✓
no lease info	192.168.2.95	Vlink for Honneywell	00:0B:B4:11:27:41	✓
no lease info	192.168.2.81		00:01:03:DE:86:27	✓
no lease info	192.168.2.98	Vlink for PPC	00:0B:B4:11:1D:3D	✓
no lease info	192.168.2.90	ISI Sensor	00:50:C2:46:81:01	✓

Figure 8. IP Addresses assigned to Each Network Interface by the 3COM Router.

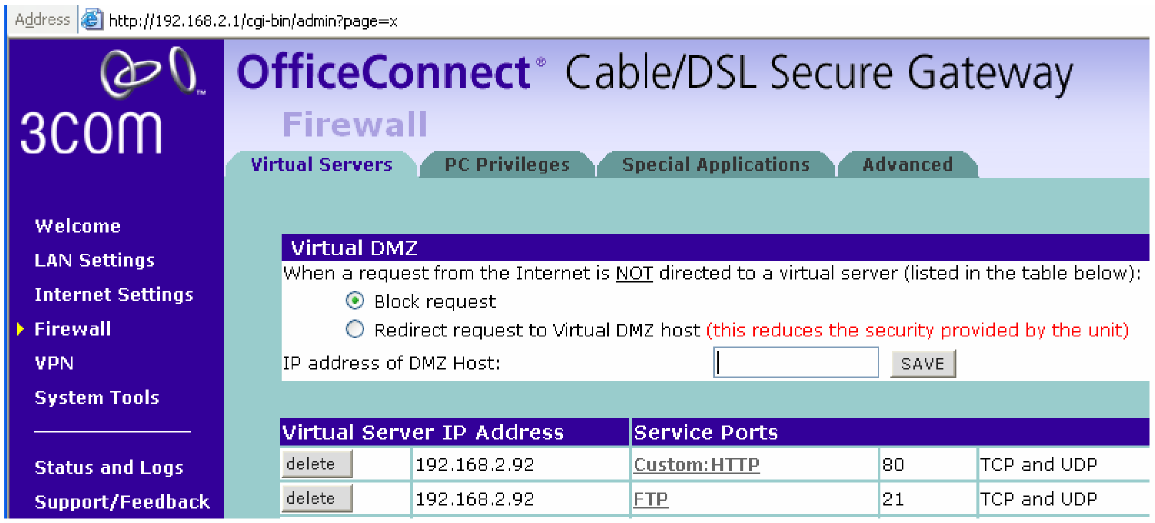
The WAN port of the 3COM router is connected to the school's network via a static IP address of 205.155.65.71. Figure 9 shows the Internet Settings utility is used to configure the 3COM router with a static IP address. This IP address is also used by the technician to access the automatic web-based sensor calibration system.



Connection Parameters	
IP Allocation Mode	Static IP address (to be specified manually)
IP Address	205.155.65.71
Subnet Mask	255.255.255.128
ISP Gateway Address	205.155.65.1
Primary DNS Address	205.155.65.10
Secondary DNS Address	205.155.65.11 (optional)

Figure 9. Configuring a 3COM Router with a Static IP Address using a Web Browser.

The 3COM router also serves as the firewall that filters out data traffic between the ship's LAN and the WAN. The firewall is configured to only allow bidirectional traffic on port 80, as shown in Figure 10. This would allow technicians to access the web site (located on a web server computer, behind the firewall) to calibrate the sensor. Figure 10 also shows that a FTP port is configured on port 21 to allow images from the D-link web camera to be saved on the web server computer.



Virtual DMZ

When a request from the Internet is NOT directed to a virtual server (listed in the table below):

[✓ Block request]

Redirect request to Virtual DMZ host (this reduces the security provided by the unit)

IP address of DMZ Host: SAVE

	Virtual Server IP Address	Service Ports		
delete	192.168.2.92	Custom:HTTP	80	TCP and UDP
delete	192.168.2.92	FTP	21	TCP and UDP

Figure 10. Configuring HTTP Server Port on the Firewall of the 3COM Router.

2. NETGEAR PoE Switch

The Netgear switch, as shown in Figure 11, is used to provide Power-over-Ethernet (PoE) for the ISI sensor and to add additional LAN ports to the ship's LAN. It has four non-PoE LAN ports, and four PoE LAN ports. One non-PoE port is used to connect to the 3COM router as shown in Figure 5 and Figure 7. A similar type of connection can be repeated to add more ports to the ship's LAN, as the ship's LAN grows.

Figure 11. A Netgear PoE Switch is used to provide PoE to sensors and add more Ethernet ports to the virtual ship's LAN.

3. Linksys Wireless Router

A WRT55AG Linksys wireless router, as shown in Figure 12, is used to provide wireless access to the ship's LAN. It also adds three more ports to the ship's network. In this research, it is only used as a wireless access point. The WRT55AG Linksys Wireless Router has four wired LAN ports and one wired WAN port. The WAN port is left unconnected. One of the LAN ports is connected to another LAN port on the 3COM router. This would add three additional LAN ports and a wireless access capability to the ship's LAN.

Figure 12. Different Aspects of a WRT55AG Wireless Router [From Ref. 12].

C. VLINX RS232-TO-ETHERNET ADAPTER

1. Description

The PPT or the PPC communicates through a RS232 interface. In order to integrate these sensors into the ship's LAN, one Vlinx RS232-to-ethernet adapter is used for each sensor. The Vlinx RS232-to-ethernet adapter converts a RS232 interface of a sensor into an Ethernet interface. Figure 13 shows the connection of a serial device, such as the PPT or PPC, to a LAN.

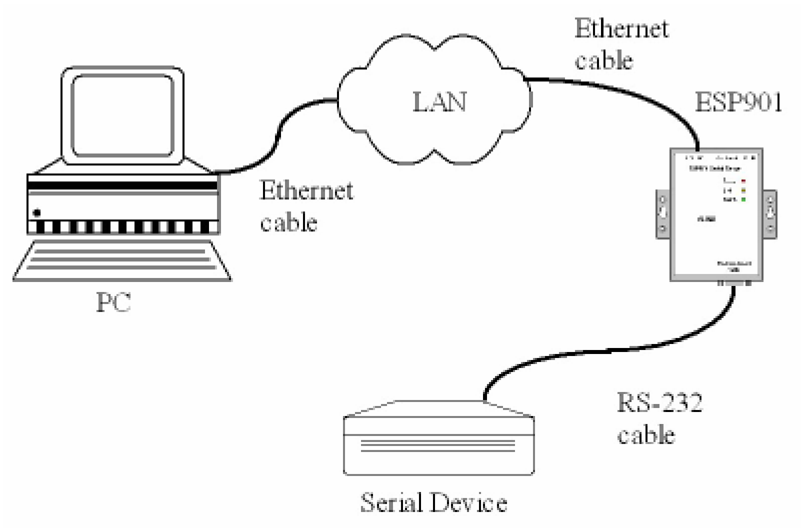
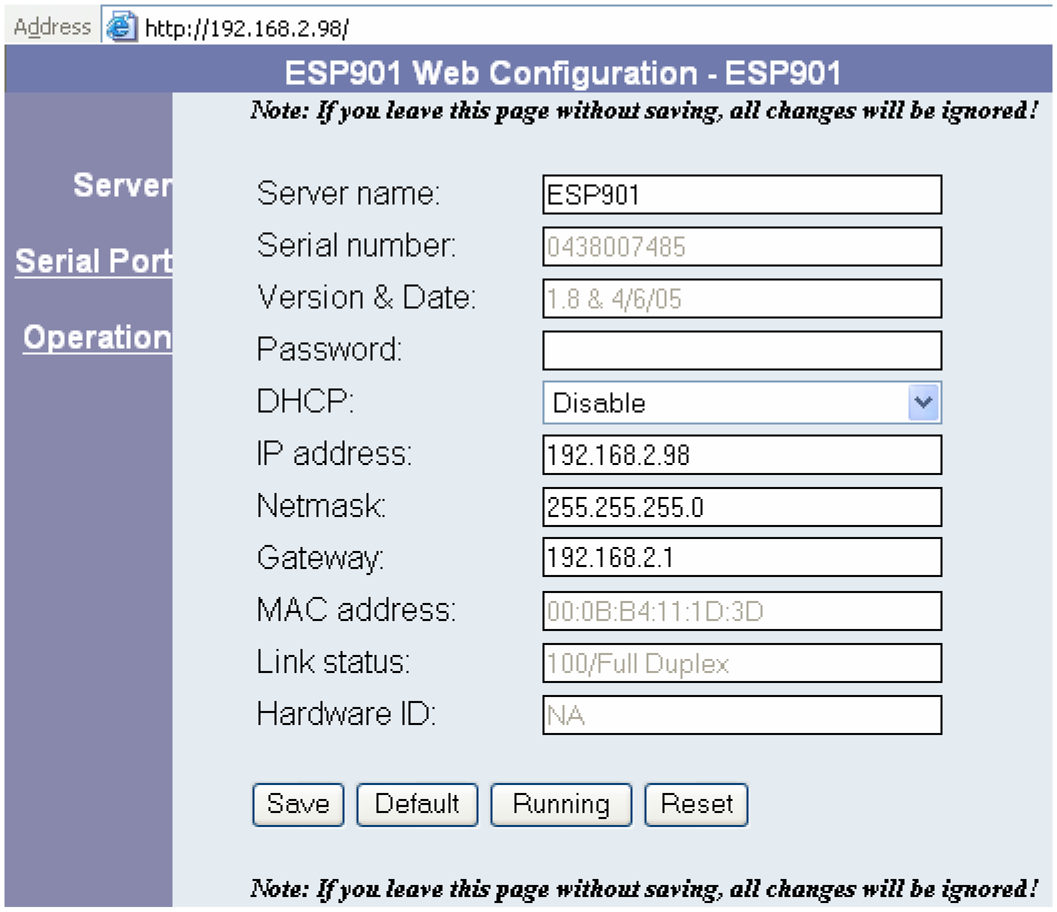


Figure 13. Connection Diagram of RS-232 Interface to the Ship's LAN using the Vlinx RS-232 to Ethernet Adapter [From Ref. 13].

2. Configuration

Inside the Vlinx RS232-to-ethernet adapter is an embedded computer that hosts an embedded web server and TCP/IP or UDP server. This Vlinx RS232-to-ethernet adapter is preconfigured with DHCP operation, which allows it to automatically get the assigned IP address from the 3COM router. In this research, the Vlinx RS232-to-ethernet adapter used with the PPC is assigned with an IP address of 192.168.2.98. The web pages hosted by the Vlinx RS232-to-ethernet adapter can be accessed by entering this IP address in the URL box of a web browser, as shown in Figures 14-16. From this page the

Vlinx RS232-to-ethernet adapter can be configured to send data on the network using UDP or TCP/IP protocol and the port number on which the data packets are sent on. This web page can also be used to configure the baud rate on the RS-232 interface of the adapter. As shown in Figure 15, the Ethernet interface of the adapter is configured to communicate over port 4000 using TCP/IP protocol. The RS-232 interface is configured with a baud rate of 9600 bps, 8 data bits, 1 stop bit and no parity, as shown in Figure 16.



http://192.168.2.98/

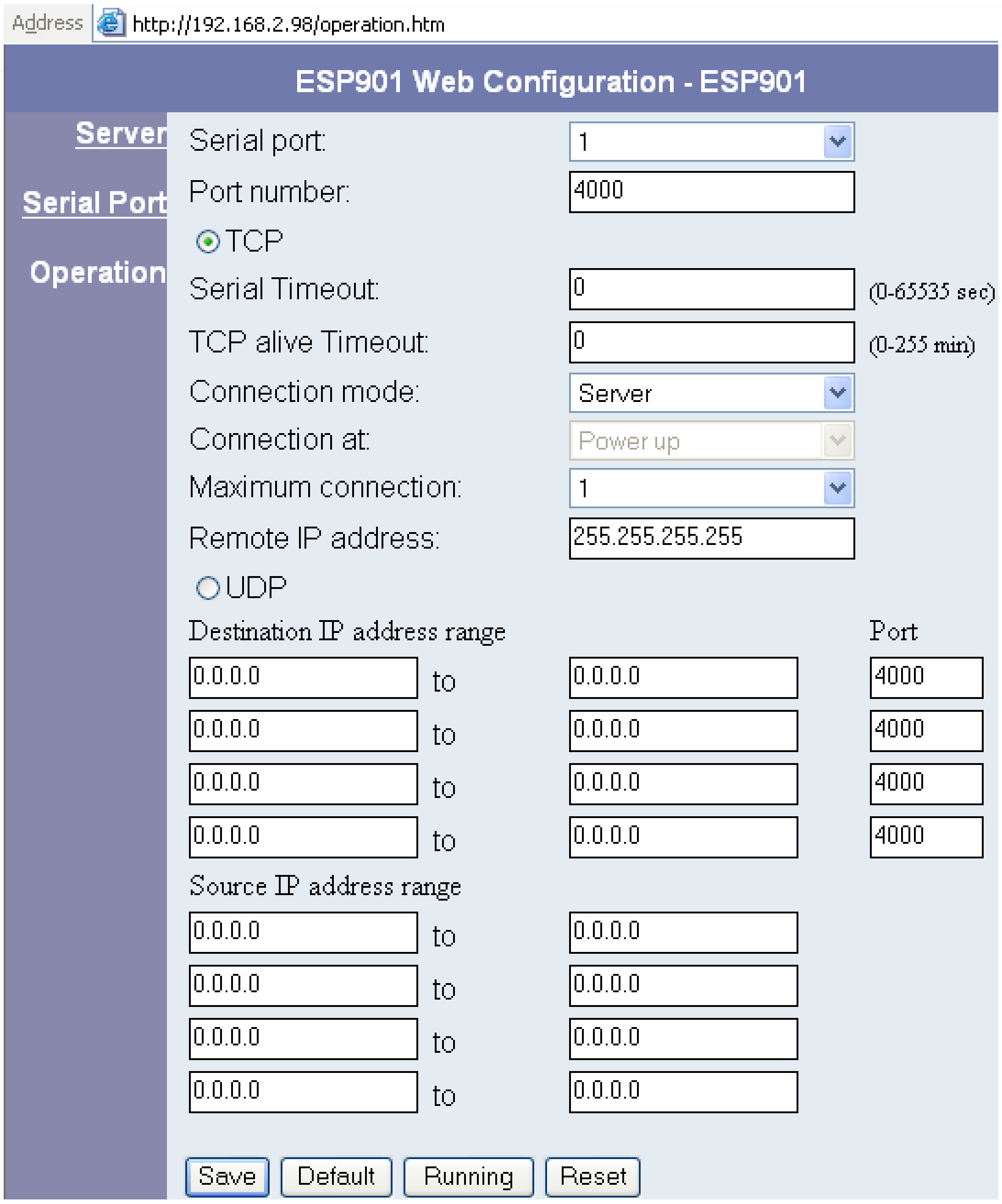
ESP901 Web Configuration - ESP901

Note: If you leave this page without saving, all changes will be ignored!

Server name:	ESP901
Serial number:	0438007485
Version & Date:	1.8 & 4/6/05
Password:	
DHCP:	Disable
IP address:	192.168.2.98
Netmask:	255.255.255.0
Gateway:	192.168.2.1
MAC address:	00:0B:B4:11:1D:3D
Link status:	100/Full Duplex
Hardware ID:	NA

Note: If you leave this page without saving, all changes will be ignored!

Figure 14. Web Page hosted by the Vlinx Box displays the Assigned IP Address.



Address http://192.168.2.98/operation.htm

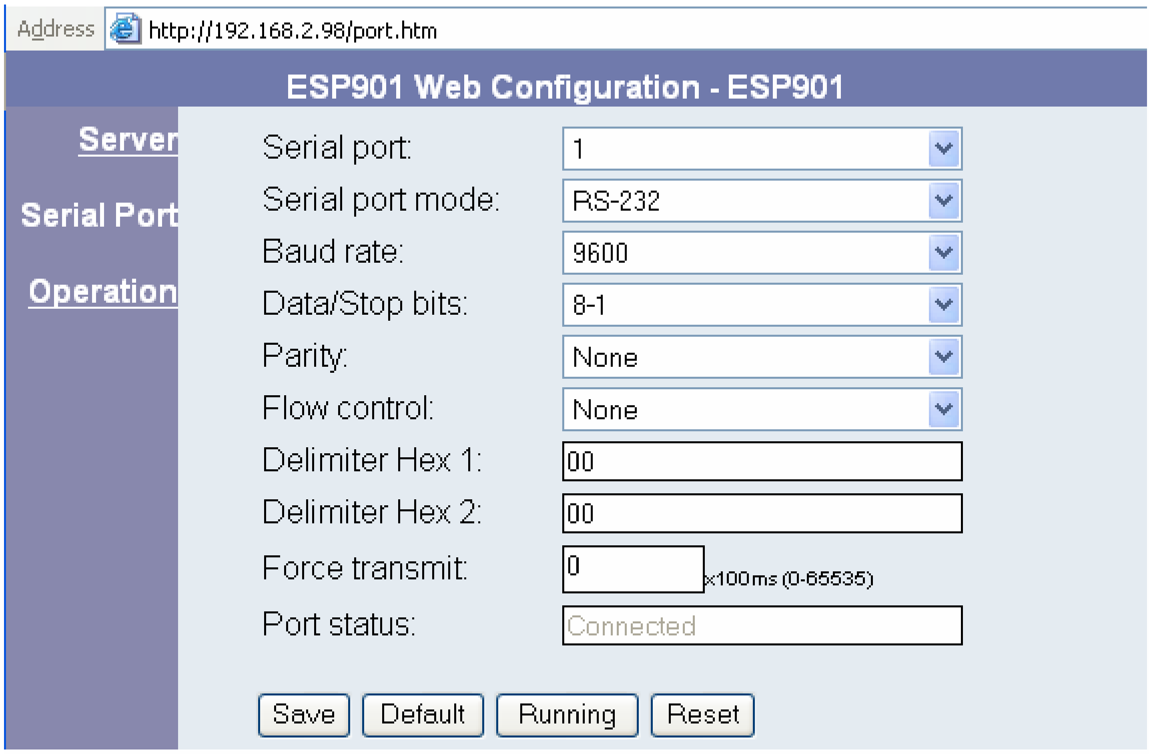
ESP901 Web Configuration - ESP901

Serial port:	1		
Port number:	4000		
● TCP			
Serial Timeout:	0	(0-65535 sec)	
TCP alive Timeout:	0	(0-255 min)	
Connection mode:	Server		
Connection at:	Power up		
Maximum connection:	1		
Remote IP address:	255.255.255.255		
○ UDP			

Destination IP address range			Port
0.0.0.0	to	0.0.0.0	4000
0.0.0.0	to	0.0.0.0	4000
0.0.0.0	to	0.0.0.0	4000
0.0.0.0	to	0.0.0.0	4000

Source IP address range		
0.0.0.0	to	0.0.0.0
0.0.0.0	to	0.0.0.0
0.0.0.0	to	0.0.0.0
0.0.0.0	to	0.0.0.0

Figure 15. Vlinx Configuration Web Page that is used to configure the Ethernet Interface.


Address http://192.168.2.98/port.htm

ESP901 Web Configuration - ESP901

Serial port:	1
Serial port mode:	RS-232
Baud rate:	9600
Data/Stop bits:	8-1
Parity:	None
Flow control:	None
Delimiter Hex 1:	00
Delimiter Hex 2:	00
Force transmit:	0 x100ms (0-65535)
Port status:	Connected

Figure 16. The Vlinx Configuration Web Page that is used to configure the RS-232 Interface.

D. WEB SERVER COMPUTER

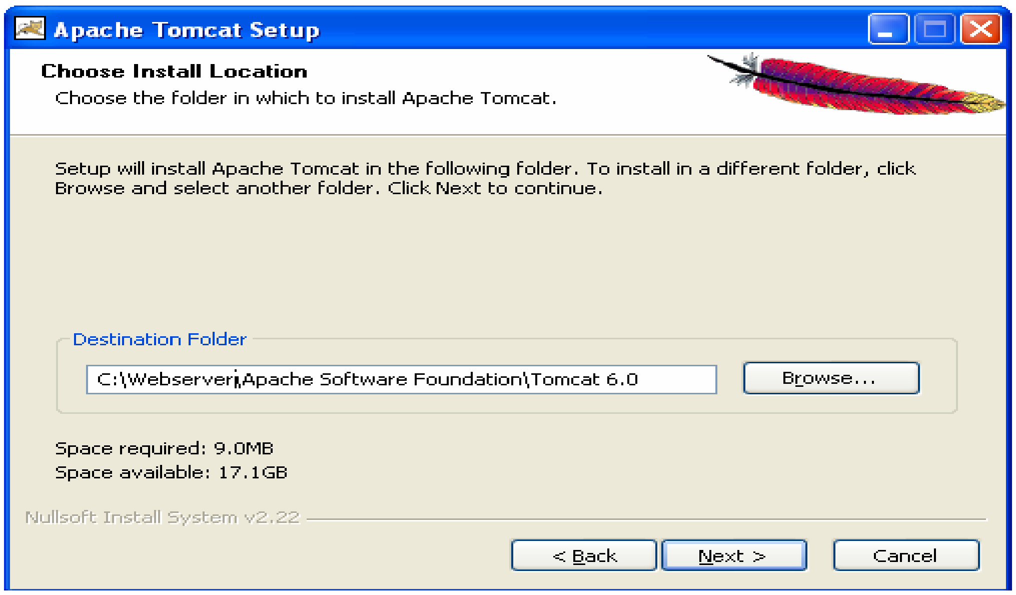
1. Description

The web server computer is a Dell Optiplex 260 that is connected to the 3COM router via a LAN port, as shown in Figure 5. It is loaded with different software and web pages to provide a user friendly interface to the automatic web-based sensor calibration system. The software consists of Tomcat web server for window 6.0.13, Java Development Kit 1.5.0, Java applets and servlets. The Tomcat web server software can be downloaded at http://tomcat.apache.org.

2. Configure the Tomcat Web Server

After being downloaded, the Tomcat web server is installed in the folder *C:\Webserver\ Apache Software Foundation\Tomcat 6.0*, as shown in Figure 17. Once completed, the Tomcat software can be launched by selecting “Configure Tomcat” or

"Monitor Tomcat" program from the startup menu, as shown in Figure 18. After launching the application, a GUI is popped up as shown in Figure 19. The server can be started by clicking on the "*Start*" button on the lower left hand of the GUI. This would display Tomcat's default web page, which is located at the folder *C:\WebServer\ ApacheSoftwareFoundation\Tomcat_6_0\webapps\ ROOT*. In order to for the web server to display the virtual ship's web pages, the working path for the server is changed to the folder *C:\virtual_ship*. This task is accomplished by clicking on the "*Startup*" tab of the GUI and replacing the "*Working Path*" text box with C:\virtual_ship, as shown in Figure 20. Directory *C:\virtual_ship* is where all web pages designed for this research is located at. The contents of this folder is shown in Figure 21. All Java applets are stored in *C:\virtual_ship\applet*. All Java servlets are saved at the folder *C:\virtual_ship\WEB-INF\classes*. The interface between the applets and the servlets are detailed in file *web.xml* located at the folder *C:\virtual_ship\WEB-INF*. More details on file *web.xml* will be discussed in Chapter IV, Section B.


Apache Tomcat Setup

Choose Install Location

Choose the folder in which to install Apache Tomcat.

Setup will install Apache Tomcat in the following folder. To install in a different folder, click Browse and select another folder. Click Next to continue.

Destination Folder

C:\Webserver\Apache Software Foundation\Tomcat 6.0

Space required: 9.0MB
Space available: 17.1GB

Nullsoft Install System v2.22

Figure 17. Installation of the Tomcat Web Server.

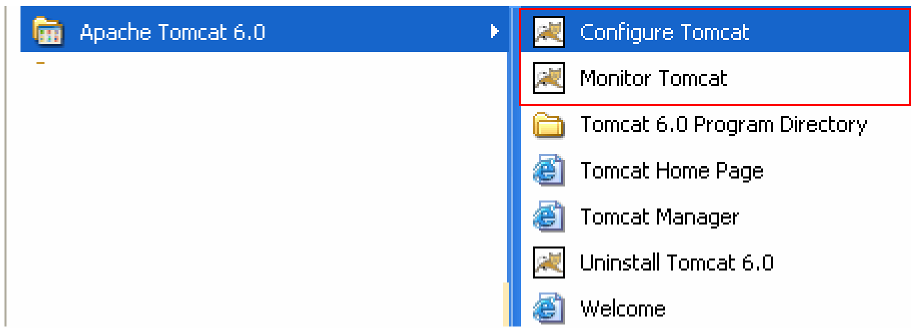
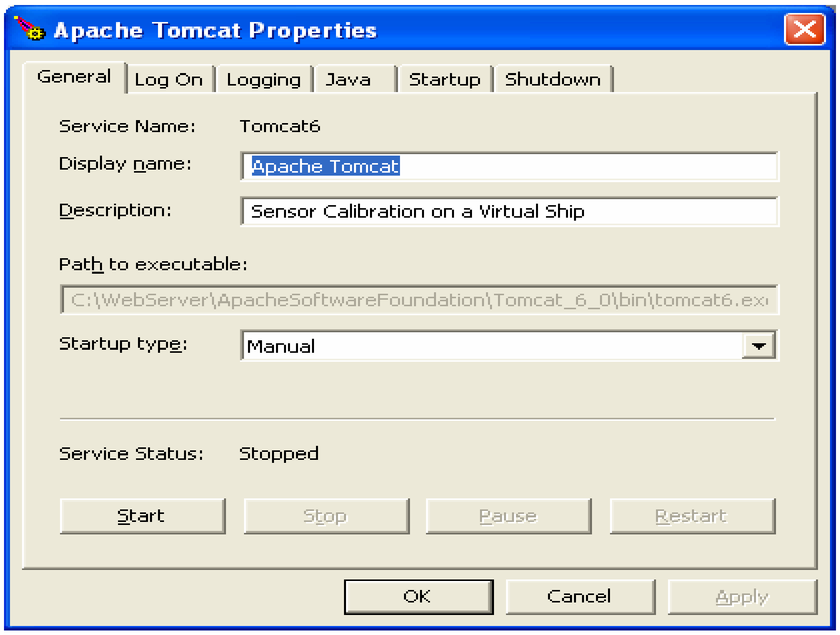
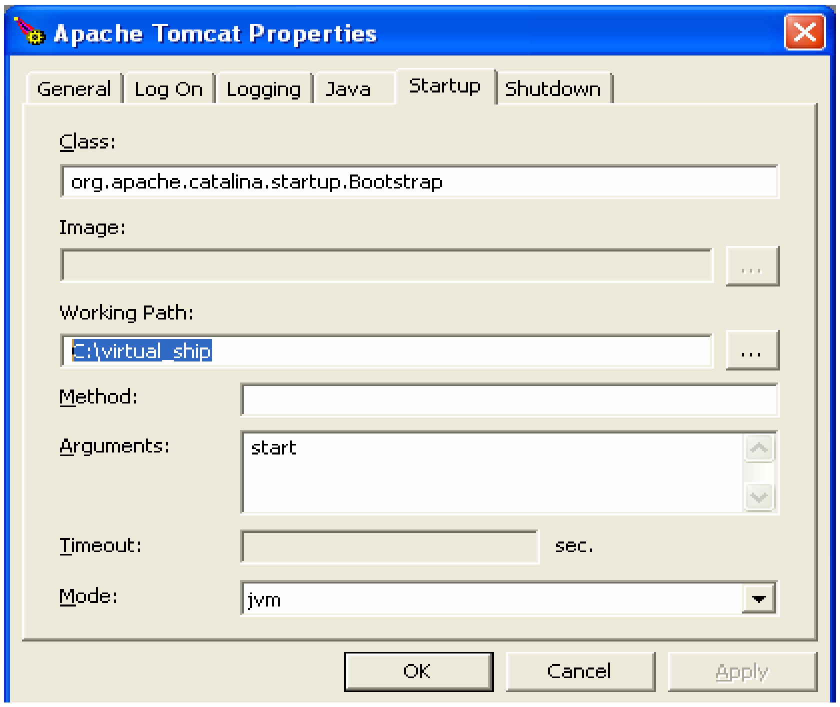


Figure 18. Launching the Tomcat Web Server from the Startup Menu.



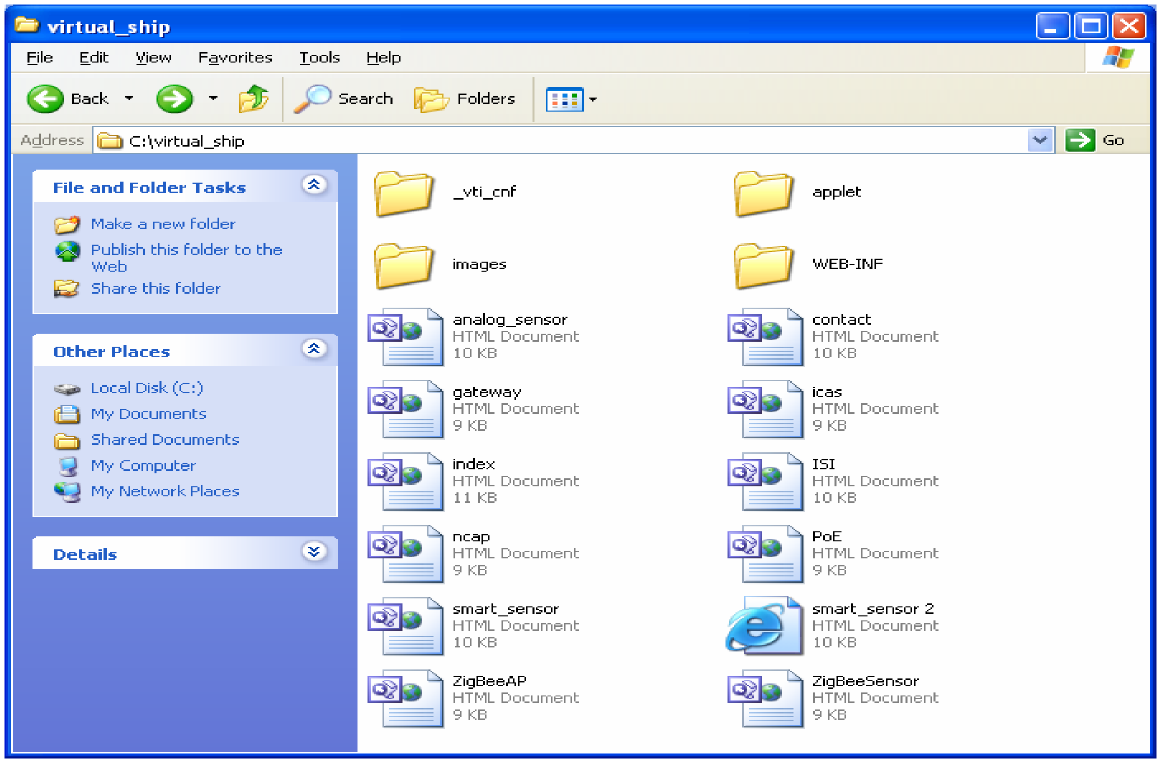
Service Name: Tomcat6
Display name: Apache Tomcat
Description: Sensor Calibration on a Virtual Ship
Path to executable:
C:\WebServer\ApacheSoftwareFoundation\Tomcat_6_0\bin\tomcat6.ex
Startup type: Manual
Service Status: Stopped

Figure 19. Tomcat GUI for Launching the Web Server.



Class: org.apache.catalina.startup.Bootstrap
Image:
Working Path: C:\virtual_ship
Method:
Arguments: start
Timeout: sec.
Mode: jvm

Figure 20. Changing the Working Path to C:\virtual_ship.



Address: C:\virtual_ship

_vti_cnf
applet
images
WEB-INF
analog_sensor HTML Document 10 KB
contact HTML Document 10 KB
gateway HTML Document 9 KB
icas HTML Document 9 KB
index HTML Document 11 KB
ISI HTML Document 10 KB
ncap HTML Document 9 KB
PoE HTML Document 9 KB
smart_sensor HTML Document 10 KB
smart_sensor 2 HTML Document 10 KB
ZigBeeAP HTML Document 9 KB
ZigBeeSensor HTML Document 9 KB

Figure 21. The Virtual Ship Folder.

3. Set up the JAVA Development Environment

In this research, Sun's Java Development Kit 1.5.0 (jdk.1.5.0) is used to develop Java servlets and applets, which allow operators to communicate with the calibrating computer via port 80. Java Development Kit 1.5.0 can be downloaded from Sun's web site at http://www.sun.com. In order to compile the Java servlet and applet codes, all system environments variables listed in Table 1 are set with the corresponding values following the installations of Java Development Kits 1.5.0 and Tomcat web server.

Environment Variables	**Setting Values**
CATALINA_HOME	C:\Inetpub\Tomcat
CLASSPATH	C:\Inetpub\Tomcat\lib\servlet-api.jar
JAVA_HOME	C:\Program Files\Java\jdk1.5.0
PATH	C:\Program Files\Java\jdk1.5.0\bin\;C:\Inetpub\Tomcat\bin

Table 1. Required System Environment Variables for Java Servlet Development.

E. CALIBRATING COMPUTER

The calibrating computer is a Dell Optiplex 260 that is connected to the 3COM router via a LAN port, as shown in Figure 5. It also interfaces with the pump and valve controller on the parallel port (LP1). In this research, the calibrating computer is also used as a Labview software developing computer. It is installed with Labview 6.1 software development package, purchased from National Instrument. If Labview 6.1 is not installed, a Labview 6.1 run-time engine must be installed in order to run the Labview software developed on another computer. The Labview 6.1 run-time engine can be downloaded from National Instrument web site, at http://www.ni.com.

F. ELECTRIC PUMP, OPENING VALVE, AND CONTROLLERS

An electrical pump and an opening valve, as shown in Figures 22-23, are used to control the air pressure applied on the sensor during the calibration process. The pump is used to increase the air pressure, while the opening valve is used to decrease the air pressure in the pipe system. The pump and the opening valve are controlled by separate controlling circuits, as shown in Figure 24. The controlling circuits are input with signals from the parallel port of the calibrating computer. These signals are amplified and used to turn ON/OFF power supply to the electrical pump or the opening valve.

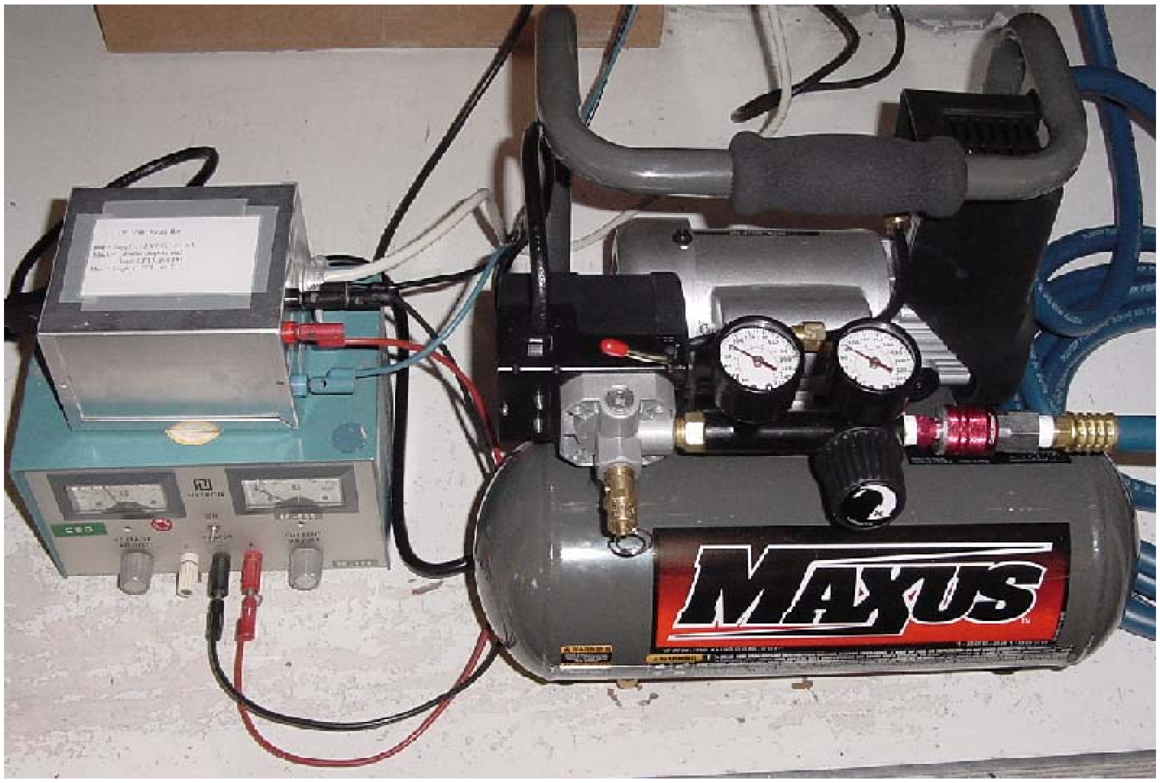


Figure 22. The Electrical Pump, and the Pump Controller.

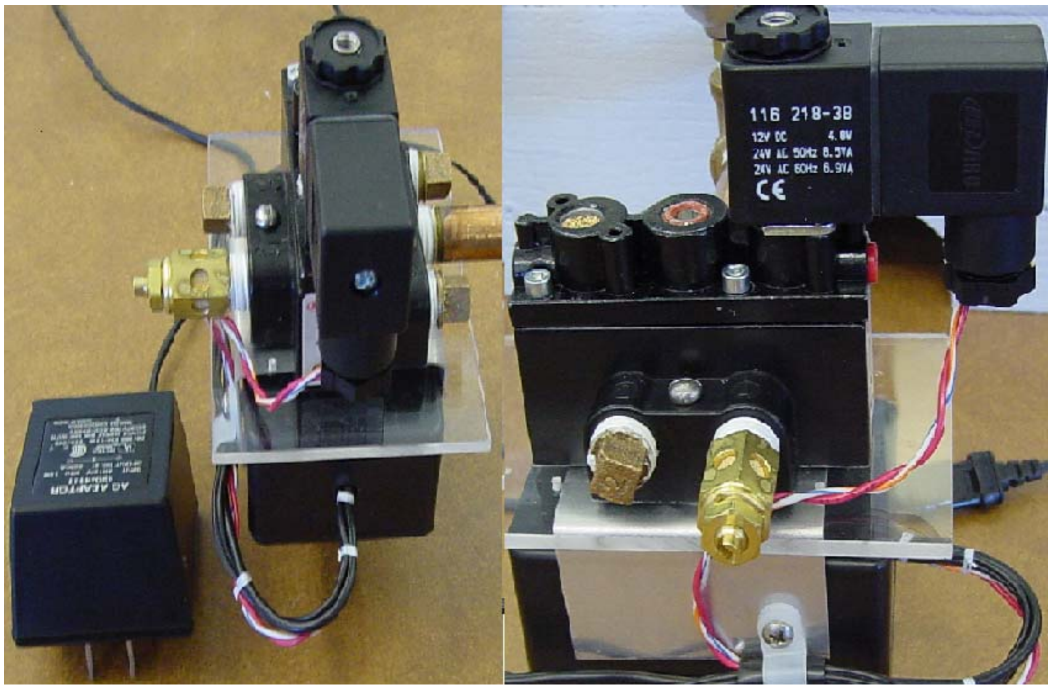


Figure 23. The Opening Valve and the Valve Controller.

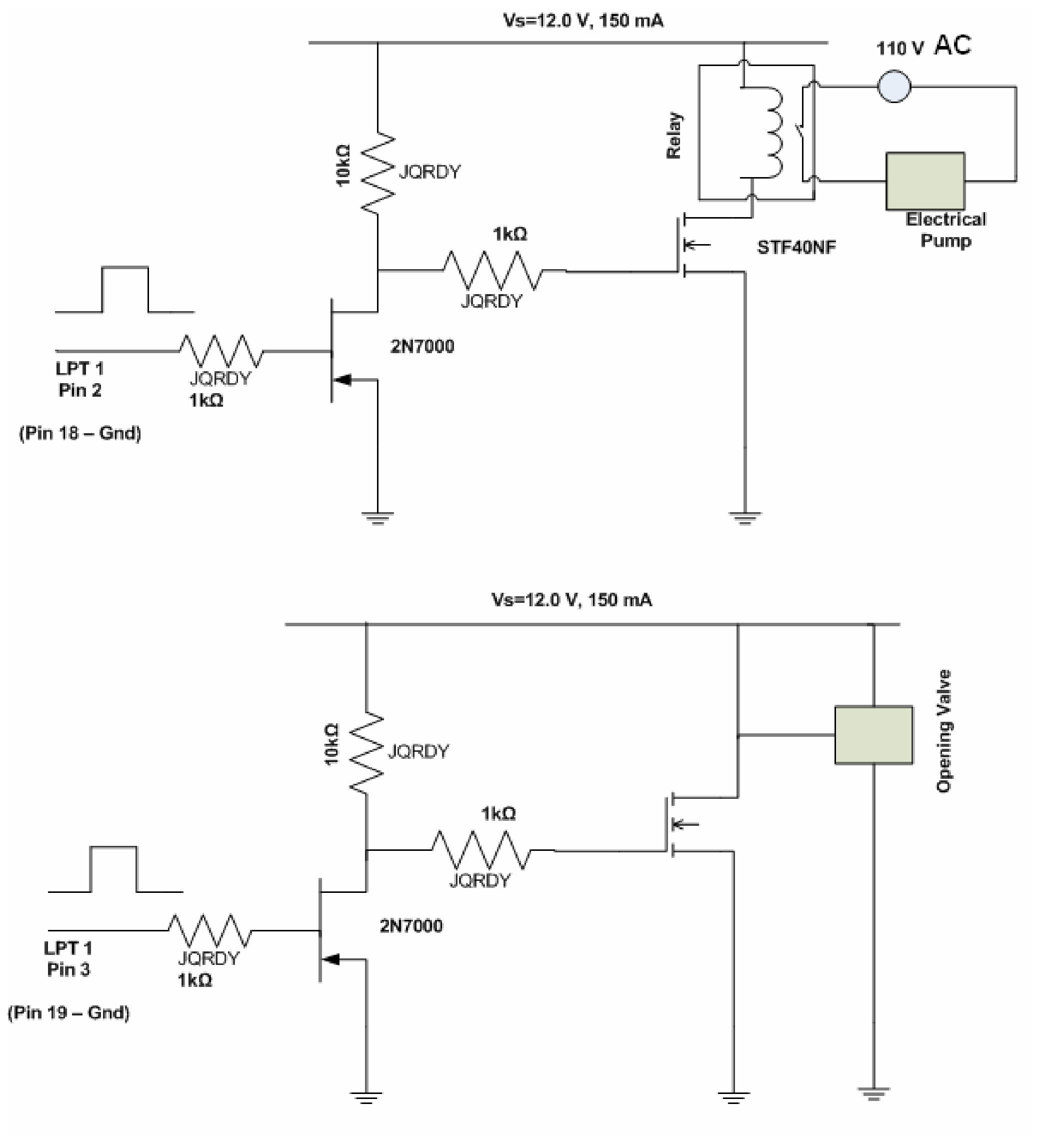


Figure 24. Circuit Diagram of the Pump and Valve Controllers.

G. SENSORS

The virtual ship integrates five different types of sensors: the HonneyWell PPT, the ISI sensor, the PPC, the analog sensor and the Zigbee sensor. Each offers unique advantages and disadvantages. They, however, can co-exist in the same shipboard environment. This lab setup demonstrates the feasibility of using different types of sensors in a sensor network. All sensors except the PPC are used as the target sensor for calibration. The PPC is used as a standard sensor to calibrate the target sensors.

1. Honeywell Precision Pressure Transducer (PPT)

Honeywell Precision Transducer, as shown in Figure 26, is a highly accurate sensor that can provide readings in both digital and analog form. In digital mode, the PPT can interface with either RS-232 or RS-485 port. In this research, the PPT operates in digital mode. Its RS-232 port is configured with the following settings:

- Baud Rate = 9600
- Start Bits = 1
- Data Bits = 8
- Stop Bits = 1
- Parity = None

To collect the sensor reading from the PPT, the query string **"*00P1"** is sent to the PPT. The substring **"00"** is the default address of the sensor, and the substring **"P1"** is the command requesting a single pressure reading in ASCII format. The PPT responses with a sensor reading for each queried command. The responses from the PPT have the format of **"01CP=XX.XX"**, where **XX.XX** is the sensor reading in PSI. The PPT is integrated with the ship's LAN via a Vlink RS232-to-ethernet adaptor, as shown in figure 25. This allows the calibrating computer to communicate with the PPT across the ship's LAN.

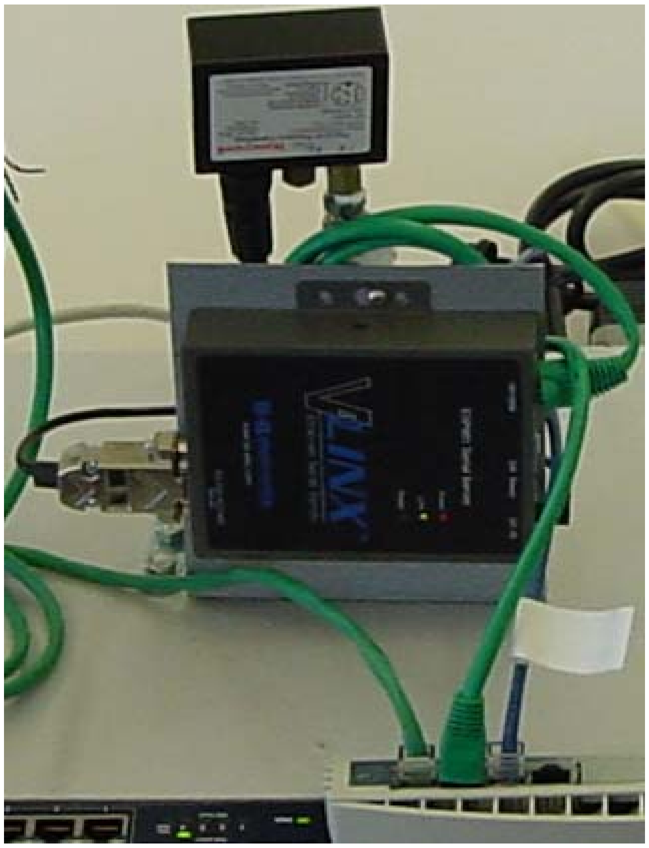


Figure 25. Honneywell Precision Pressure Transducer Connected to a Vlinx Ethernet Adapter.

2. ISI Pressure Sensor

The ISI pressure sensor is a single-cable, Ethernet-ready sensor that does not require a separate power source, as shown in Figure 26. The sensor is connected to the ship's LAN by a Category 5 Ethernet cable via the PoE port of the Netgear switch. The power sources from PoE ports are IEEE 802.3af compliant, and provide adequate power for the ISI sensor's operations.

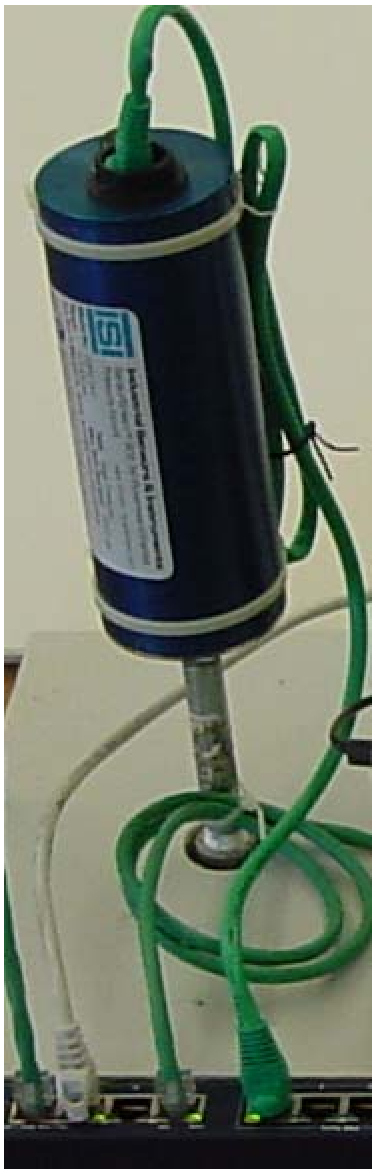


Figure 26. An ISI Pressure Sensor is connected to a PoE Switch.

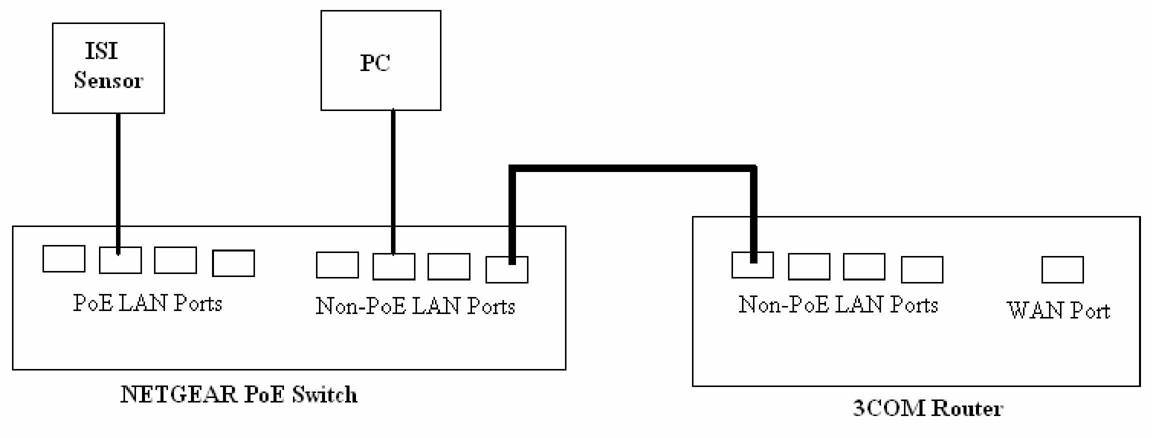


Figure 27. The ISI Sensor interfaces with the Ship's LAN via a Netgear PoE Switch.

The ISI sensor does not support DHCP operation, therefore it is assigned with a static IP address. In preparation for the static IP address assignment, the PC is connected to the non-PoE port, while the ISI sensor is connected to the PoE port of the Netgear switch. Another non-PoE port of the Netgear switch is also connected to the 3COM router, as shown in Figure 27. The ISI sensor's static IP address assignment is started by issuing the command ***arp –s 192.168.2.90 00:50:C2:46:81:01 192.168.2.82*** on the command console. At this point, the web pages hosted by the ISI sensor can be access by entering 192.168.2.90 in the URL box of a web browser, as shown in Figure 28.

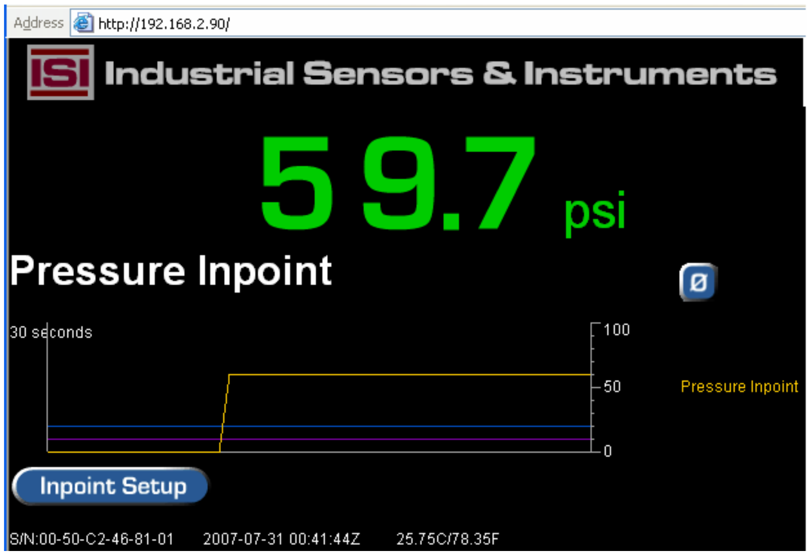
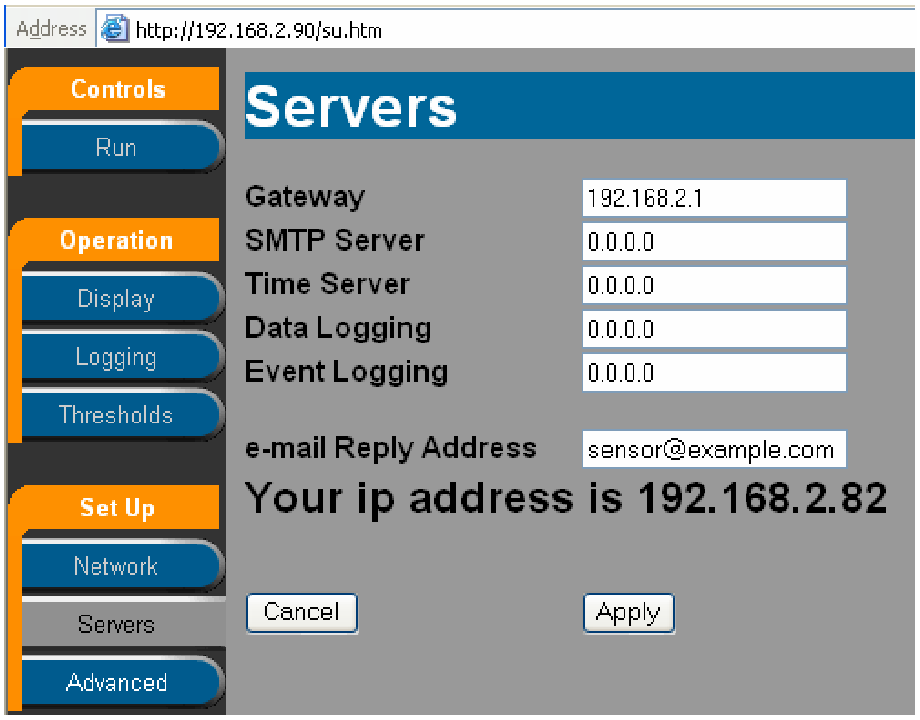

Address http://192.168.2.90/
Industrial Sensors & Instruments
59.7 psi
Pressure Inpoint
30 seconds
100
50
0
Pressure Inpoint
Inpoint Setup
S/N:00-50-C2-46-81-01 2007-07-31 00:41:44Z 25.75C/78.35F

Figure 28. Main Web Page Hosted by the ISI Sensor.

In order for the sensor reading sent by the ISI sensor to reach other clients in the network, server's gateway on the ISI is configured with the 3COM router's IP address. This task is accomplished by clicking on the "*Inpoint Setup*" button on the lower left corner of Figure 28, followed by clicking on the "*Servers*" tab on the lower left corner of the new window. Once the "server" tab is active, as shown in Figure 29, the 3COM router's address (102.168.2.1) is entered on the Gateway text box. The "*Apply*" button is then selected.


Address http://192.168.2.90/su.htm

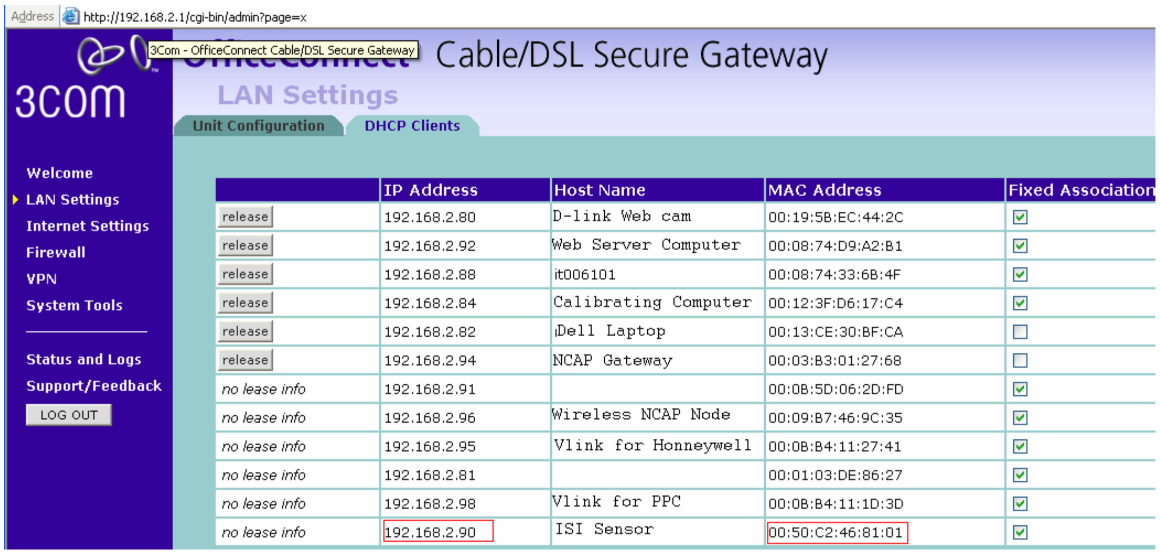
Servers

Gateway 192.168.2.1
SMTP Server 0.0.0.0
Time Server 0.0.0.0
Data Logging 0.0.0.0
Event Logging 0.0.0.0

e-mail Reply Address sensor@example.com
Your ip address is 192.168.2.82

Figure 29. The ISI Sensor has a Static IP Address of 192.168.2.90, and has the 3COM's IP Address as Its Gateway to the Ship's LAN.

At this point, the ISI sensor can see the 3COM router, but the router does not see the ISI sensor as a client of the ship's LAN. To register the ISI sensor's static IP address with the 3COM router, its MAC address and static IP address is manually entered on the 3COM router's routing table. Another browser is opened with "192.168.2.1" entered in the URL to access 3COM router's configuration tools. After logging in, the router's client table is accessed by clicking on the "LAN Settings" tab on the left, followed by clicking on the “DHCP Clients” tab. The ISI sensor’s static IP address and the MAC address can now be added to the DHCP Clients Table, as shown in Figure 30.



	IP Address	Host Name	MAC Address	Fixed Association
release	192.168.2.80	D-link Web cam	00:19:5B:EC:44:2C	✓
release	192.168.2.92	Web Server Computer	00:08:74:D9:A2:B1	✓
release	192.168.2.88	it006101	00:08:74:33:6B:4F	✓
release	192.168.2.84	Calibrating Computer	00:12:3F:D6:17:C4	✓
release	192.168.2.82	Dell Laptop	00:13:CE:30:BF:CA	☐
release	192.168.2.94	NCAP Gateway	00:03:B3:01:27:68	☐
no lease info	192.168.2.91		00:0B:5D:06:2D:FD	✓
no lease info	192.168.2.96	Wireless NCAP Node	00:09:B7:46:9C:35	✓
no lease info	192.168.2.95	Vlink for Honneywell	00:0B:B4:11:27:41	✓
no lease info	192.168.2.81		00:01:03:DE:86:27	✓
no lease info	192.168.2.98	Vlink for PPC	00:0B:B4:11:1D:3D	✓
no lease info	192.168.2.90	ISI Sensor	00:50:C2:46:81:01	✓

Figure 30. ISI Sensor’s Static IP Address and MAC Address are manually added to the Router’s DHCP Client Table.

With the ISI sensor connected to the ship’s LAN, the sensor’s reading can be queried by typing “http://192.168.2.90/-$rdng” in the URL of a web browser. The sensor replies back with the pressure reading in ASCII format. This same concept will be exploited to develop software to communicate with the ISI sensor, in chapter IV. The ISI sensor’s additional functions and capabilities can be found in [11].

3. Crystal XP2 Portable Pressure Calibrator (PPC)

The PPC is a highly accurate pressure sensor manufactured by Crystal Engineering Corporation. The sensor is powered by three AA batteries (4.5 V), as shown in Figure 31. The PPC requires very little power to operate. With three AA batteries, it can operate continuously for two months [8]. The PPC displays pressure reading on the LCD display and sends the reading out on the RS-232 port on the back of the sensor. The RS-232 port is preconfigured with the following default setting:

- Baud Rate = 9600
- Data Bits = 8
- Stop Bits = 1
- Parity = None
- Flow Control = None

The RS-232 port also allows the PPC to be configured remotely. A complete description of the queried and configuration commands can be found in [9]. In this research, the sensor reading can be queried by sending a query string **"?P,U**" to the sensor over the RS-232 link. The PPC responses to the query string with a pressure reading and a pressure unit in ASCII format. This concept can be further exploited by developing software to communicate with the PPC automatically, as discussed in Chapter IV. The PPC is integrated to the ship's LAN using a Vlinx RS232-to-ethernet adapter in the same manner as shown Figure 13. This allows the calibrating computer to communicate with the PPC across the ship's LAN.

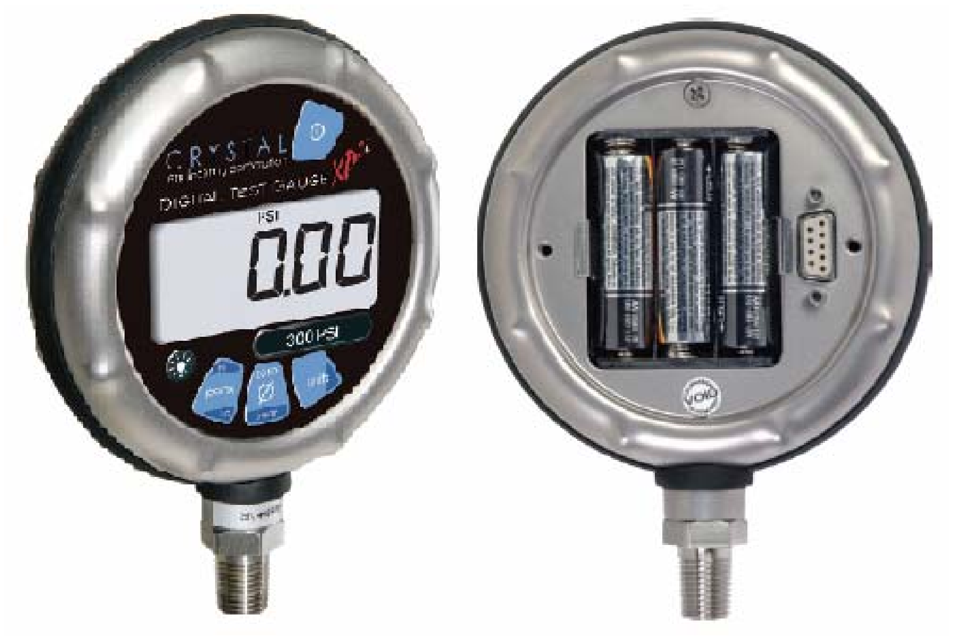


Figure 31. The Sensor Head of a Crystal XP2 Portable Pressure Calibrator (PPC) is powered by three AA Batteries and can communicate over an RS-232 Interface [From Ref 8].

4. Analog Sensor

The analog sensor, Figure 32, used in this thesis is the X205 4 to 20 mA pressure transducer, manufactured by Omega. The sensor is connected to an NCAP (Network Capable Application Processor), where the analog readings are digitized and sent over the ship's LAN to clients. The sensor reading can be retrieved by establishing a TCP connection to the NCAP box with the IP address 192.168.2.97 and port 1501. Once a TCP connection is granted by the TCP server on the NCAP, the sensor's readings are sent to the client periodically.

Figure 32. An Omega Analog Pressure Sensor [Ref 10].

5. Zigbee Sensor

The ZigBee sensor is a custom-made wireless pressure sensor utilizing Zigbee technology and Commercial-off-the-Shelf components, as shown in Figure 33. ZigBee is a wireless technology based on the IEEE 802.15.4 standard. It is intended for monitoring and control applications that require low data rate and low power consumption. The sensor transmits unprocessed data wirelessly to the base station (Figure 34) where the data is processed and made available on the ship's LAN [14].

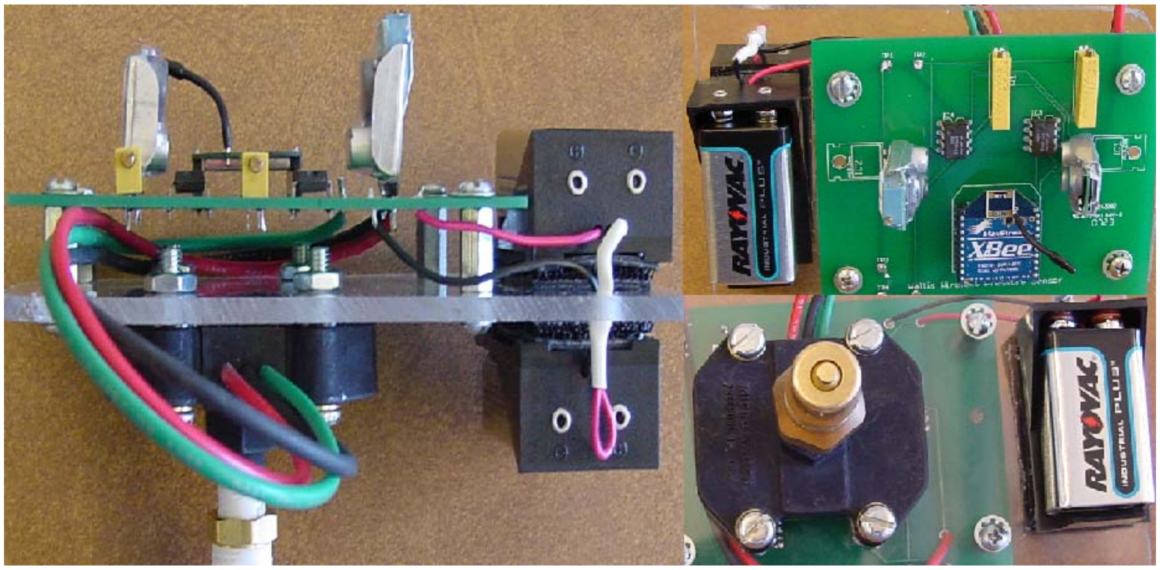


Figure 33. Top, Bottom, and Side View of the Custom-made Zigbee Wireless Sensor [14].

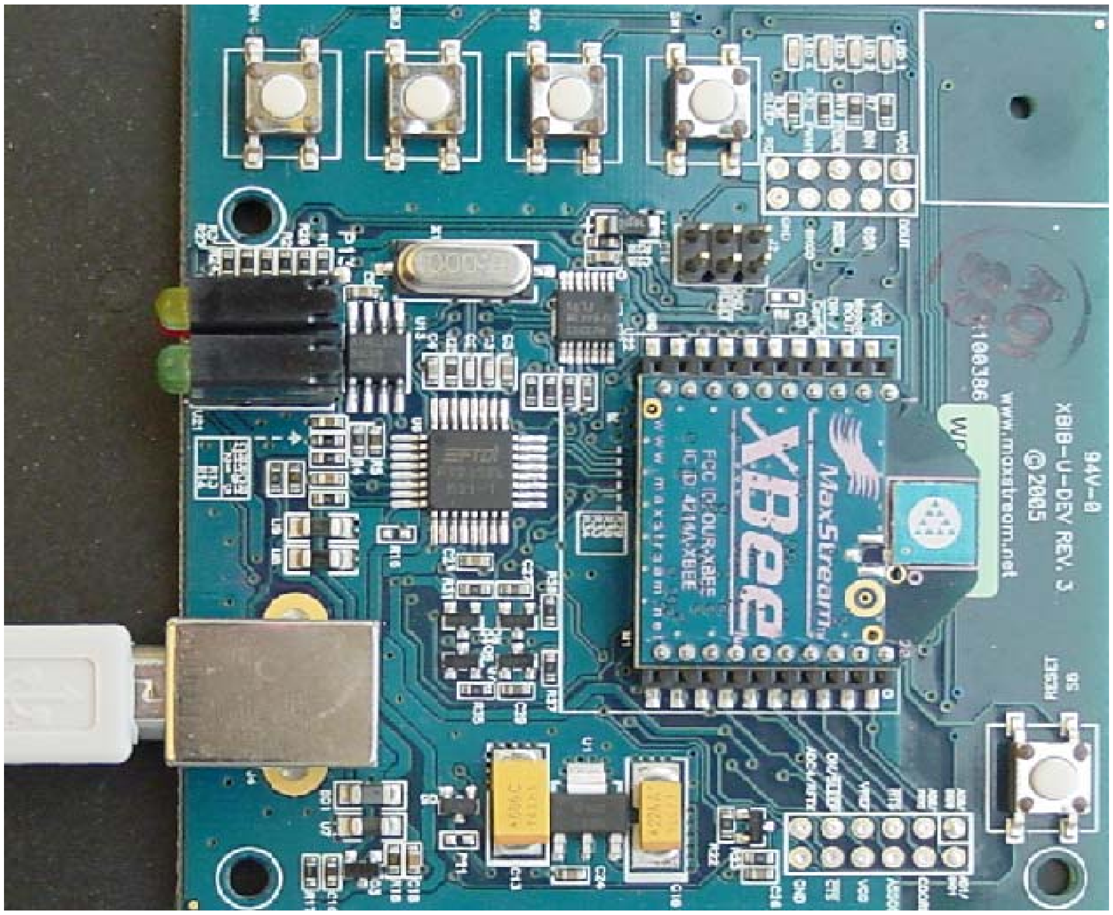


Figure 34. A Zigbee Base Station [14].

THIS PAGE INTENTIONALLY LEFT BLANK

IV. SOFTWARE

A. INTRODUCTION

The software can be classified into three main categories: HTML codes, Java codes, and Labview codes. HTML codes and Java codes are hosted on the web server computer, while the Labview codes are hosted on the calibrating computer. The codes are developed using Microsoft Front Page, Sun's Java Software Development Kits 1.5.0, and National Instruments' Labview 6.1. The whole operation of all developed codes can be summarized in a diagram shown in Figure 35. When the sensor calibrating page is accessed, Java applets are loaded and run on the operator's computer to establish communication with the Tomcat web server. The Java servlets located on the web server computer will act as a data translator, converting and relaying data messages between the applets and the calibrating computer. Labview codes are hosted on the calibrating computer to communicate with the web server and execute commands requested by the operator. This chapter discusses algorithms implemented by software, using the top-down approach starting from the web pages, then the Java codes, and finally the Labview codes. Only snapshots of important source codes are displayed or discussed. Complete copies of the source codes, which are rather lengthy, can be found in the appendix.

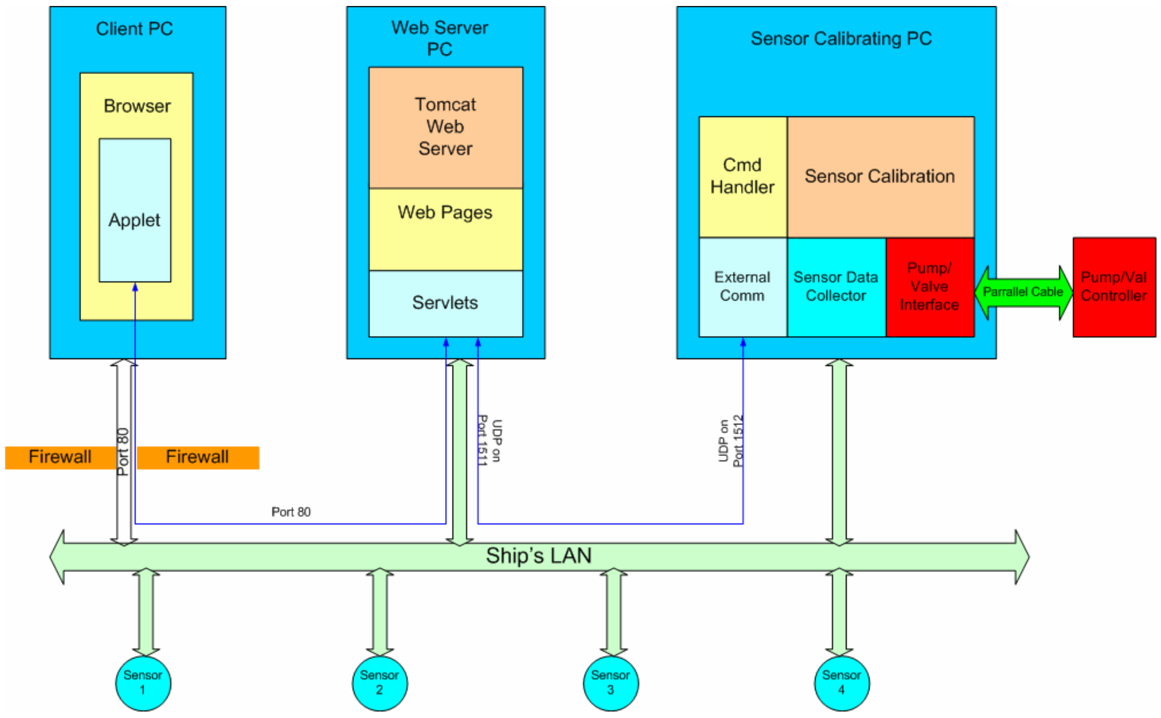


Figure 35. Communication between different Software Components in the Automatic Web-based Calibration System.

B. COMMUNICATION BETWEEN COMPONENTS

A large portion of the software is developed to implement the bidirectional data communication between the operator and the calibrating computer. This communication link allows the operator to initiate the calibration process monitor the sensor readings in real time. The data exchanges between the user and the calibrating computer are in binary format, while the data exchanges between the calibrating computer and the sensors are in ASCII format.

1. Communication between a Web Page and the Web Server Computer

When both the user and the virtual ship are located on two separate secured networks, only port 80 is available for the user to initiate a bidirectional communication channel with the web server computer. The problem is the operator's web browser is also communicating with the web server computer on port 80. A normal socket communication on port 80 would not work. To overcome this problem, a Java applet is developed to send the POST requests from the client to the web server and wait for web server's response to the POST request before continuing.

a. Communication from an Applet to the Web Server

POST request are sent from two different applets: SensorApplet.class and ImageApplet.class. POST request from the applet ImageApplet.class has only one type of data request, which is the image from the web camera. The POST requests from the applet SensorApplet, however, have different meanings, depending on what is filled in the [Command] field of the requests. The data payload of POST request from the the applet SensorApplet.class is arranged in the format shown in Table 2.

Data Format	Command	Sensor ID	Min Reading	Max Reading
Size in bytes	2	2	8	8
Data Type	Unsigned Short	Unsigned Short	Double	Double

Table 2. Payload Data Format of the POST Requests from SensorApplet.class to SensorServlet.class.

Starting from left to right of Table 2, the request contains a command, sensor ID, the lower bound, and upper bound of the pressure range supported by a sensor. The [Command] field and the [Sensor ID] field, each is two-byte long. Each is implemented by an unsigned short variable. The [Min Reading] field and the [Max Reading] field, each is eight-byte long. Each is implemented by a double variable. There are a total of eight different values that could be assigned to the [Command] field. The complete definition of each command is listed in Tables 3.

Commands	**Meaning**
400	Request Sensor Reading of the Sensor with the Attached ID
401	Calibrate the Sensor with the Attached ID
402	Start Pump
403	Stop Pump
404	Open Valve
405	Close Valve
406	Reset Calibration Constants of Sensor with the Attached ID
407	Get Current Calibration Constants of Sensor of the given ID
408	Get Previous Calibration Constants of Sensor of the given ID

*Sensor ID = Each sensor has an ID, starting from 0
*Min/Max Reading = Min/Max reading of the reading range that a sensor supports

Table 3. Table of Command Definitions.

b. Communication from the Web Server to an Applet

POST requests from the applets SensorApplet.class and ImageApplet.class are responded by the servlets SensorServlet.class and ImageServlet.class, respectively. The response from the ImageServlet.class is an image object, which is a live webcam image of the virtual ship. The image file is saved on the web server computer by the IP web camera. The responses from the SensorServlet.class contain data payloads arranged in the format shown in Table 4.

Data Format	Message 1	Reading 1	Message 2	Reading 2
Size in bytes	2	8	2	8
Data Type	Unsigned Short	Double	Unsigned Short	Double

Table 4. Payload Data Format of the Responses from SensorServlet.class to SensorApplet.class.

Each of the fields [Reading 1] and [Reading 2] has a different meaning depend on the content of the [Message] field on its right. When the [Message] field is equal to 300, the fields [Reading 1] and [Reading 2] are the sensor readings of the target sensor and the standard sensor (PPC). When it is equal to 301 or 302, the fields [Reading 1] and [Reading 2] are the calibration constants of the target sensor, as shown in Table 5.

Message	**Meaning of [Reading 1] [Reading 2]**
300	[Target Sensor's Reading][Standard Sensor's Reading]
301	[Current Slope][Current Offset] of Target Sensor
302	[Previous Slope][Previous Offset] of Target Sensor

Table 5. Different Meaning of [Reading 1][Reading 2] for Different Values of Message.

2. Communication between the Web Server Computer and the Calibrating Computer

Communication between the web server computer and the calibrating computer is implemented using the UDP protocol. The web server computer only sends UDP packets to the calibrating computer when it receives POST requests from SensorApplet.class. The calibrating computer, however, periodically packs sensor readings from all sensors on the virtual ship into a UDP packet and sends it to the web server. Normally, this type of packet is ignored by the web server computer. It is only read by the servlet SensorServlet.class when a POST request from the applet SensorApplet.class is received by the web server.

a. Communication from the Web Server Computer to the Calibrating Computer

When a POST request from the applet SensorApplet.class is received, the servlet SensorServlet.class extracts the data content and examines the [command] field, filters out unnecessary data field. The relevant data fields are then forwards to the calibrating computer in a UDP packet. Therefore, the UDP packet payloads for different commands have different payload data formats, as shown in Table 6.

Command ID	**Format of the UDP Data Payload**	**Packet Length**
400	No Packet sent to the calibrating computer	N/A
401	[Command][ID][Max Rreading][Min Reading]	20
402	[Command = Start Pump]	2
403	[Command = Stop Pump]	2
404	[Command = Open Valve]	2
405	[Command = Close Valve]	2
406	[Command = Reset Cal Const][ID]	4
407	[Command = Get Current Cal Const][ID]	4
408	[Command = Get Previous Cal Const][ID]	4

Table 6. The Payload Data Format of the UDP packets sent to the Calibrating Computer.

b. Communication from the Calibrating Computer to the Web Server Computer

Ten times per second, the calibrating computer packs sensor's readings from all sensors of the virtual ship into a UDP packet and sends it to the web server computer. During normal operation, the data payload of the UDP packet has the same data format as shown in Table 7. The [command] fields are filled with two-byte long value of 300, followed by eight-byte long sensor readings. When a sensor calibration is completed, one UDP packet with a different format is sent to the web server computer. The data payload of this UDP packet is organized in a format as shown in Table 8.

Data Format	300	Sensor 1 Reading	300	Sensor 2 Reading	300	Sensor 3 Reading	300	Sensor 4 Reading	…
Size in bytes	2	4	2	4	2	4	2	4	…

Table 7. The Payload Data Format of the UDP Packets sent to the Web Server Computer during the Normal Operation.

Data Format	301	Sensor 1's Slope	300	Sensor 2 Reading	300	Sensor 3 Reading	301	Sensor 1 's Offset	…
Size in bytes	2	4	2	4	2	4	2	4	…

Table 8. The Payload Data Format of the UDP Packets sent to the Web Server Computer after Sensor Calibration of Sensor 1 is completed.

3. Communication between the Calibrating Computer and Sensors

The communication link between the calibrating computer and sensors is bidirectional. It is, however, device specific. The query strings and response strings are different from sensor to sensor. Therefore, different software components must be developed for each sensor. The query string and response string for each sensor are listed in Table 9. Table 9 also shows the data format of the payload and the communication protocol used for data exchange with sensors.

Sensor	**Query String**	**Response String**	**Data Format**	**Protocol Used**
PPT	*00P1	01CP= <Reading>	ASCII	TCP
PPC	?P,U	<Reading> PSI	ASCII	TCP
ISI	http://ip_address/-$rdng	<Reading>	ASCII	HTTP

Table 9. Summary of Query Strings and Responses from each Sensor Used on the Virtual Ship.

4. Communication from the Calibrating Computer to Controllers of the Pump and the Valve

The calibrating computer interfaces with the controllers of the electrical pump and the opening valve via its parallel port. Pin 2 and 3 of the parallel port are connected to the controllers of the electrical pump and the opening valve, respectively. The communication is only one way, from the calibrating computer to the controllers. The data bit 0 and data bit 1 of the parallel port (pins 2-3) are independently toggled to control the power to the pump and the opening valve.

C. CALIBRATION WEB PAGES

The main page of the virtual ship, Figure 36, is a diagram of a ship's sensor network. To calibrate a sensor, the technician first needs to click on the target sensor of interest (ie. PPT, ISI sensor, or the analog sensor). The sensor calibration page is then displayed on the browser. Each calibration web page has two embedded Java applets, as shown in Figures 37. One applet displays the live webcam image of the virtual ship. The other simultaneously displays sensor reading of the target sensor and the standard sensor in digital and on a strip chart format. It also displays command buttons that allow operator to initiate a calibration process, reset calibration result and control the pressure applied on sensors.

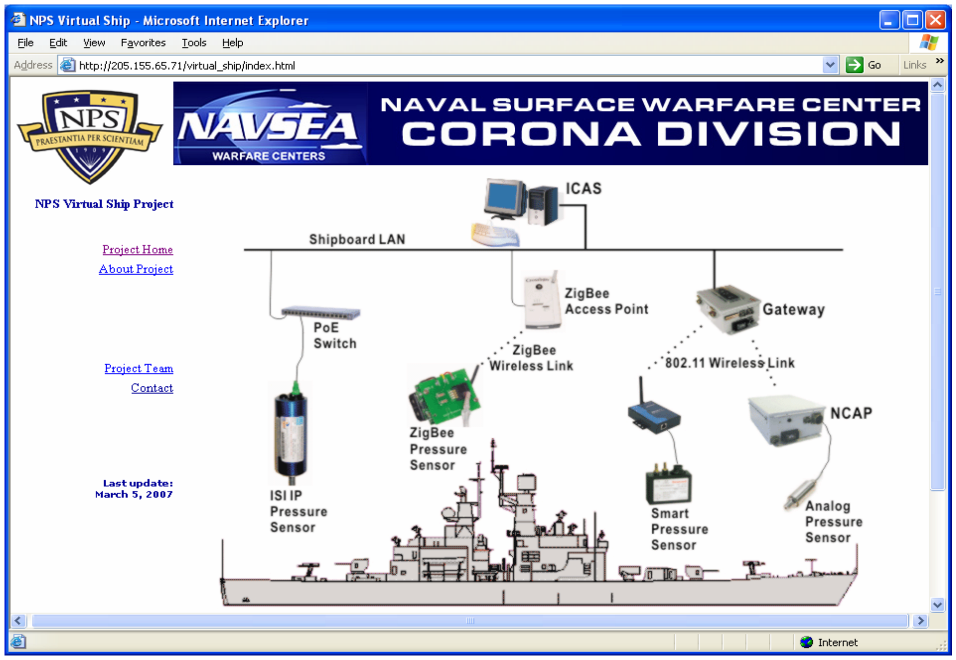
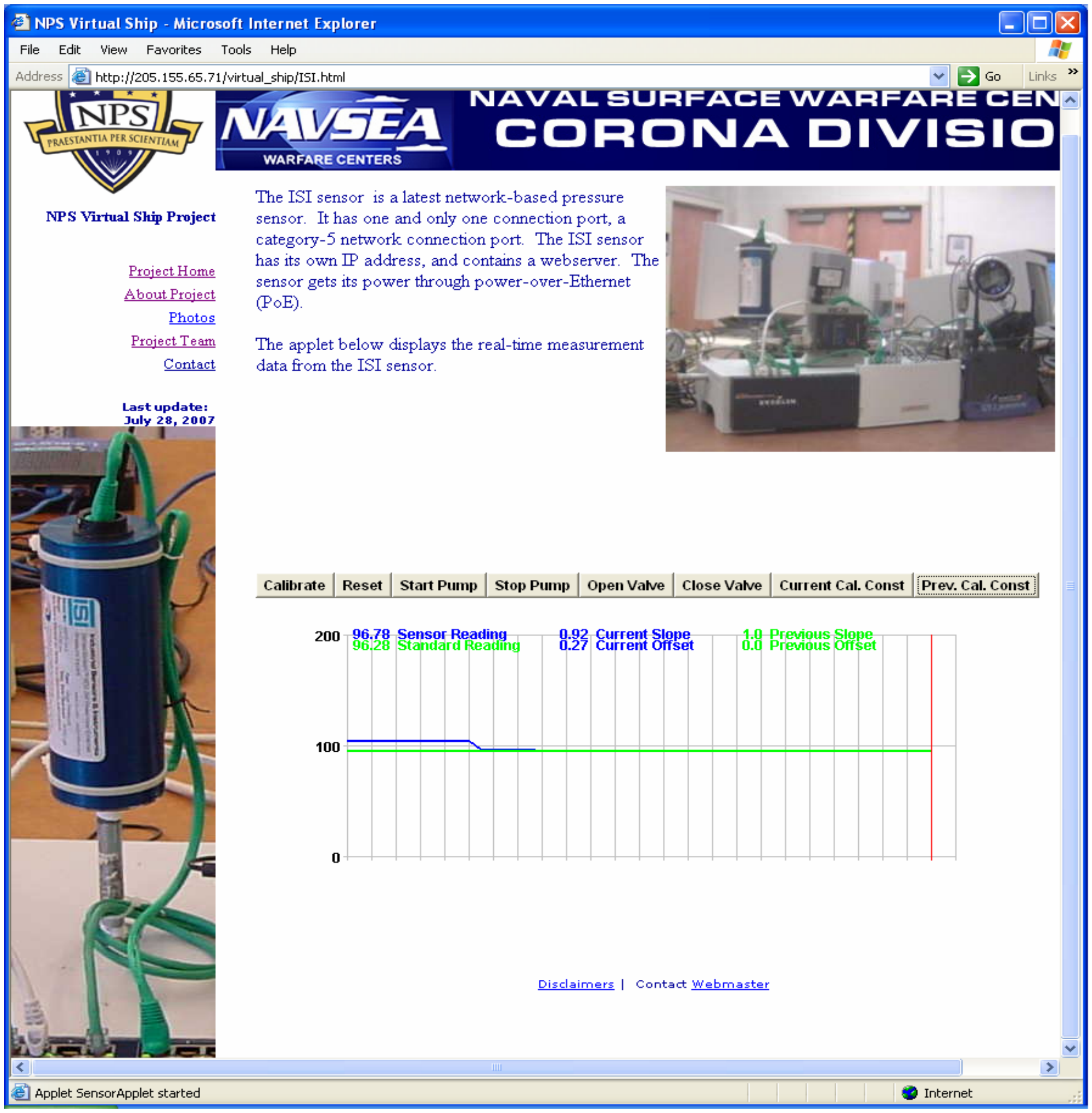


Figure 36. Main Web Page of the Virtual Ship's Automatic Web-based Sensor Calibration System.


NPS Virtual Ship - Microsoft Internet Explorer

http://205.155.65.71/virtual_ship/ISI.html

NPS Virtual Ship Project

The ISI sensor is a latest network-based pressure sensor. It has one and only one connection port, a category-5 network connection port. The ISI sensor has its own IP address, and contains a webserver. The sensor gets its power through power-over-Ethernet (PoE).

The applet below displays the real-time measurement data from the ISI sensor.

96.78 Sensor Reading
96.28 Standard Reading
0.92 Current Slope
0.27 Current Offset
1.0 Previous Slope
0.0 Previous Offset

Applet SensorApplet started

Figure 37. ISI Sensor's Calibration Page.

D. JAVA CODES

1. Introduction

The Java code is stored on the web server computer. It can be classified into two main categories: applet and servlet. The applet is downloaded and run on the client's computer when the web page is accessed. The servlet, however, does not run on its own. It is only invoked by the web server to service POST requests from clients, and it runs on the web server computer. The servlets and the applets communicate with one another via port 80, which is the only port that remains open for bidirectional communication between the virtual ship's LAN and the WAN.

2. Java Applets

When an operator accesses a sensor calibration page, two applets are downloaded from the web server and run on the operator's computer. The two applets are image applet, and the sensor applet. The image applet displays the live images of the virtual ship's lab setup, and the sensor applet displays sensor reading in strip chart format. The details of the source code are listed in files ImageApplet.java and SensorApplet.java of the appendix.

a. Image Applet

The image applet functions as an image display of a web camera. The image applet periodically sends in POST requests to the web server. When a POST request is received, the web server invokes the image servlet to attach the image file to the response of that POST request and send to the image applet. After receiving the response, the image applet displays the received image on a web browser, as shown in Figure 37.

b. Sensor Applet

The sensor applet works in a similar way as the image applet. There are nine different commands and three different messages that are exchanged between the calibrating computer and the web server. The commands are sent from the web server to the calibrating computer, and the messages are sent in the reverse direction. At initialization, the sensor applet interfaces with the target sensor's web page (in HTML format), to obtain the sensor specific information such as sensor ID, pressure reading range, and size of the strip chart (Figures 38-39). Periodically, the sensor applet sends POST request with command 400 (Table 2) to the web server to request the sensor reading of the target sensor. The POST requests are serviced by the sensor servlet, which attaches the target sensor readings and the standard sensor reading in the responses. The responses are then sent to the sensor applet. The sensor applet extracts the sensors' readings from the responses and displays both readings on the same strip chart. The target sensor readings are displayed in blue, and the standard sensor readings are displayed in green, as shown in Figure 43. When a command button on the GUI is clicked, a corresponding POST request with corresponding command sent to the web server. The complete list of commands is shown in Table 2.

```
<applet codebase="applet"
code=SensorApplet.class id=SensorApplet width="680" height="250" >
 <param name=sensorID value="0">
 <param name=bufSize value="50">
 <param name=chartHeight value="200">
 <param name=xStep value="10">
 <param name=ySpace value="100">
 <param name=speed value="1">
 <param name=maxReading value="100">
 <param name="minReading" value="15">
</applet>
```

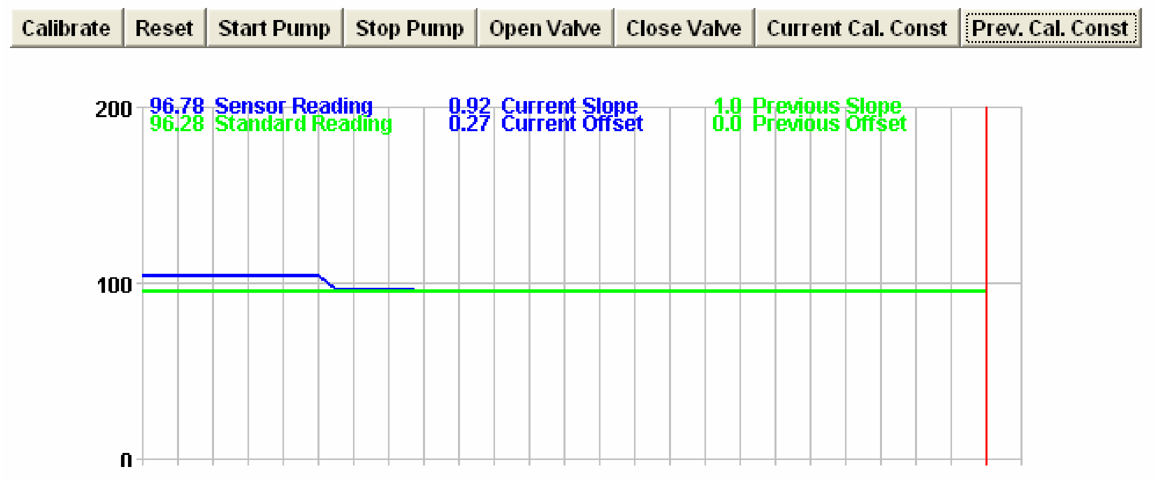
Figure 38. Html Code uses "param" to interface with Sensor Applet.

```
param = getParameter("sensorID");
if (param != null)
     sensorID = (short) Integer.parseInt(param);

//=================================
param = getParameter("speed");
if (param != null)
     m_speed = Integer.parseInt(param);


//=================================
param = getParameter("bufSize');
if (param != null) {
     try {
          bufSize = Integer.parseInt(param);
     } catch (NumberFormatException e){
          bufSize = 1;
     }
} else {
     bufSize = 1;
}
```

Figure 39. Examples of Applet Interfaces with an HTML file to Get Sensor Specific Information.



96.78 Sensor Reading
96.28 Standard Reading
0.92 Current Slope
0.27 Current Offset
1.0 Previous Slope
0.0 Previous Offset

Figure 40. The GUI of the Sensor Applet.

3. Java Servlet

Java servlets are Java codes that are invoked by the web server to service the POST and GET requests from clients. In this research, only POST requests are used. Most of them are sent by the applets. There are two servlets: image servlet and sensor servlet. The image servlet services POST requests from the image applet, and the sensor servlet services POST requests from the sensor applet. The details of the source code are listed in files ImageServlet.java and SensorServlet.java of the appendix.

a. Image Servlet

The implementation of the image servlet is fairly simple. When invoked by the web server, the image servlet attaches the image file (saved by the web camera) to the response to the POST request and send it to the image applet.

b. Sensor Servlet

The sensor servlet functions as a data translator, converting and relaying data between the sensor applet and the calibrating computer. For each POST request received from the applet (except command 400 – sensor reading), the content of the request is extracted, filtered, repackaged into the UDP packet and sent to the calibrating computer, as shown in Table 6. Similarly, content of UDP package from the calibrating computer is extracted, filtered, attached to the response to the POST and forwarded to the sensor applet, as shown in Tables 7-8.

E. LABVIEW CODE

1. Description

The Labview code is developed to be run on the calibrating computer. It is designed to perform four main functions. It has to maintain a bidirectional communication with the web server computer, and collect readings from the sensors used in the virtual ship. It also interprets commands from the servlets and invoke the corresponding functional block to carry out the command execution. Finally, it calibrates the target sensor using the automatic calibration process. The Labview code, therefore, is divided into four modules: external communication module, data collector module, command handler, and sensor calibration module. The four modules are integrated in a multi-threaded Labview program called SensorServerFinal.vi, as shown in Figures 42-44. As shown in Figure 41, the external communication module receives UDP packets from the web server computer, and passes the packet's payload to the command handler. The command handler interprets the commands and invokes the appropriate software module to execute the command. Figure 41 also shows that the outputs of the sensor calibration module are the sensor calibration constants. These constants are used to adjust the sensor readings before being sent to the web server by the external communication module.

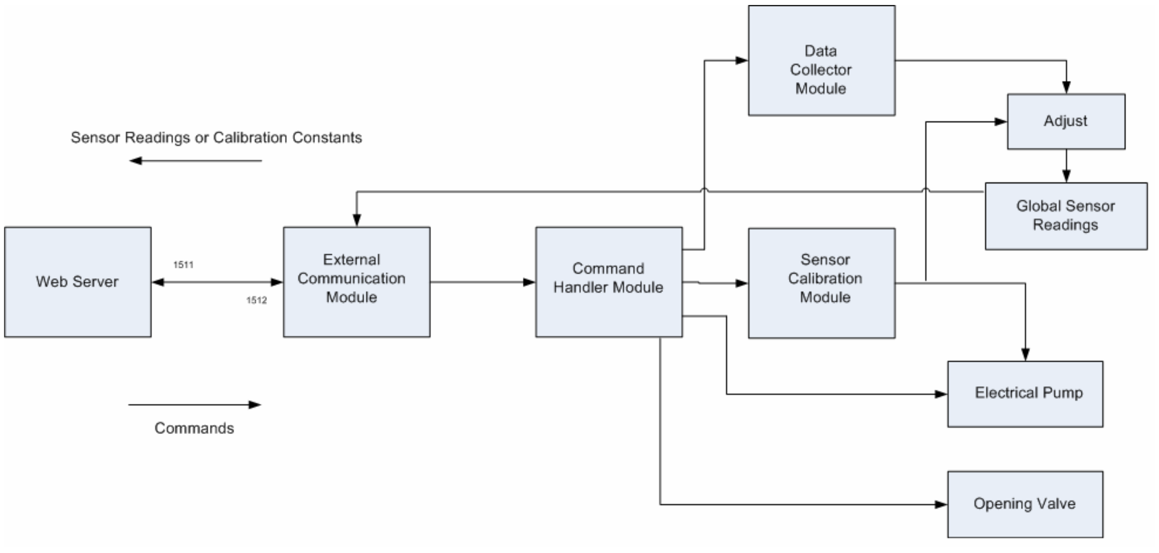


Figure 41. Block Diagram demonstrates Data Flow between the Four Labview Modules.

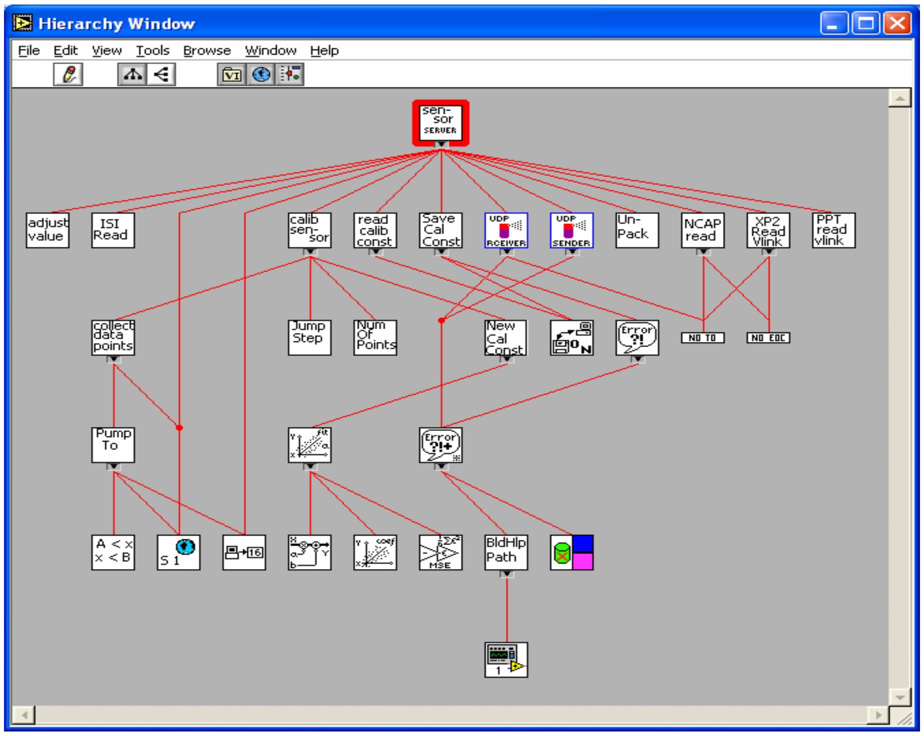
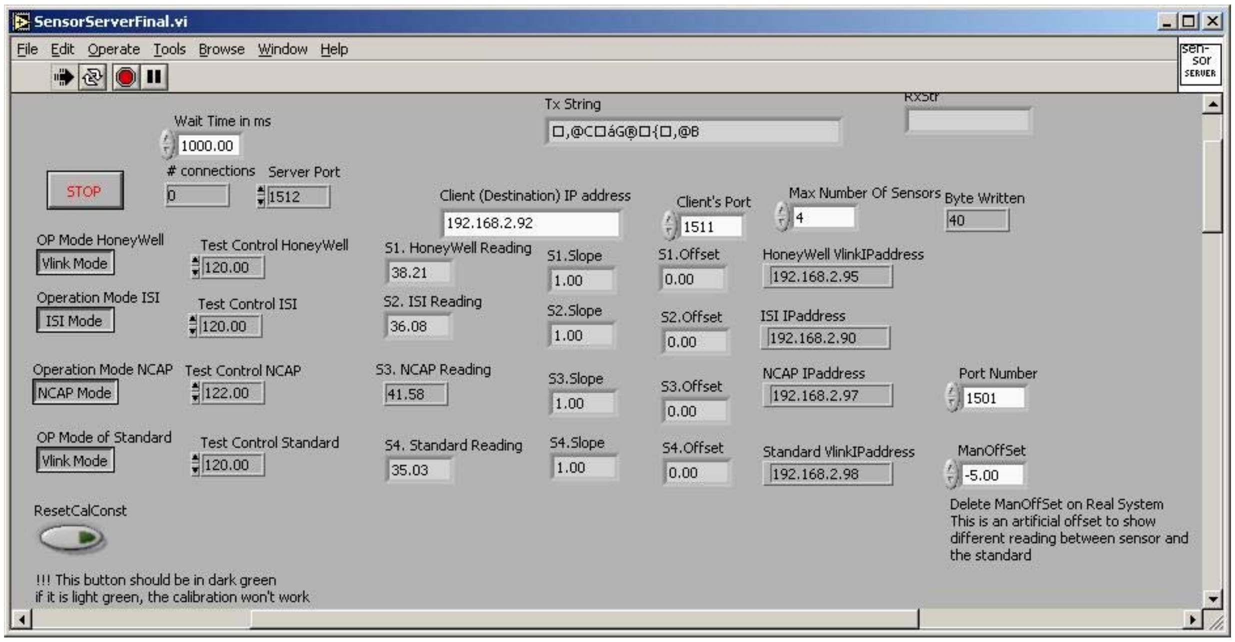


Figure 42. The Tree Diagram shows different subVIs that are used to implement the four main functions of the subVI *SensorServerFinal.vi*.



Wait Time in ms 1000.00

connections 0

Server Port 1512

Tx String □,@C□áG®□{□,@B

RxStr

Client (Destination) IP address 192.168.2.92

Client's Port 1511

Max Number Of Sensors 4

Byte Written 40

OP Mode HoneyWell Vlink Mode

Test Control HoneyWell 120.00

S1. HoneyWell Reading 38.21

S1.Slope 1.00

S1.Offset 0.00

HoneyWell VlinkIPaddress 192.168.2.95

Operation Mode ISI ISI Mode

Test Control ISI 120.00

S2. ISI Reading 36.08

S2.Slope 1.00

S2.Offset 0.00

ISI IPaddress 192.168.2.90

Operation Mode NCAP NCAP Mode

Test Control NCAP 122.00

S3. NCAP Reading 41.58

S3.Slope 1.00

S3.Offset 0.00

NCAP IPaddress 192.168.2.97

Port Number 1501

OP Mode of Standard Vlink Mode

Test Control Standard 120.00

S4. Standard Reading 35.03

S4.Slope 1.00

S4.Offset 0.00

Standard VlinkIPaddress 192.168.2.98

ManOffSet -5.00

Delete ManOffSet on Real System
This is an artificial offset to show
different reading between sensor and
the standard

ResetCalConst

!!! This button should be in dark green
if it is light green, the calibration won't work

Figure 43. Front Panel of the SubVI SensorServerFinal.vi.

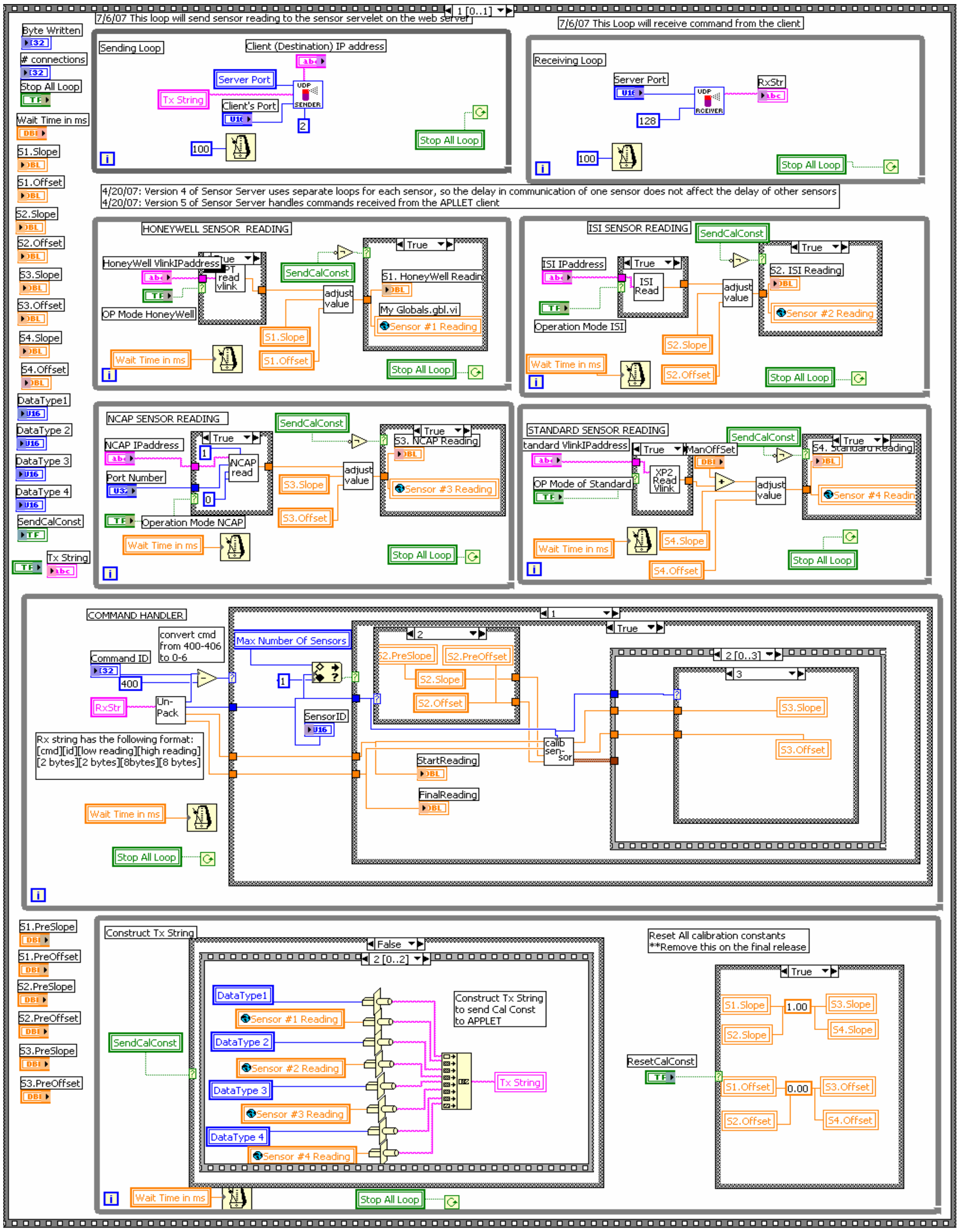

7/6/07 This loop will send sensor reading to the sensor servelet on the web server
7/6/07 This Loop will receive command from the client
Sending Loop
Receiving Loop
Client (Destination) IP address
Server Port
Client's Port
Tx String
Stop All Loop
RxStr
4/20/07: Version 4 of Sensor Server uses separate loops for each sensor, so the delay in communication of one sensor does not affect the delay of other sensors
4/20/07: Version 5 of Sensor Server handles commands received from the APLLET client
HONEYWELL SENSOR READING
ISI SENSOR READING
NCAP SENSOR READING
STANDARD SENSOR READING
COMMAND HANDLER
convert cmd from 400-406 to 0-6
Rx string has the following format: [cmd][id][low reading][high reading] [2 bytes][2 bytes][8bytes][8 bytes]
Max Number Of Sensors
Construct Tx String
Construct Tx String to send Cal Const to APPLET
Reset All calibration constants **Remove this on the final release
ResetCalConst
SendCalConst
Wait Time in ms

Figure 44. Block Diagram of the SubVI SensorServerFinal.vi.

2. External Communication Module

The external communication module is designed to maintain a bidirectional communication link with the servlets on the web server computer. The module is implemented by three threads in the file *SensorServerFinal.vi*, as shown in Figures 45-46. The first thread loads whatever is in the local variable *Tx String* into a UDP packet and sends it to the web server computer's IP address on port 1511. The second thread keeps listening on port 1512 of the calibrating computer and copies the data payload of the received UDP packet into the local variable *Rx String*, which will be processed by the command handler module. The third thread constructs the local variable *Tx String* in the same format as previously shown in Tables 7-8. The third thread also monitors the button *ResetCalConst*, and resets all calibration constants when the button is pressed on on the front panel of the Labview program *SensorServerFinal.vi*.

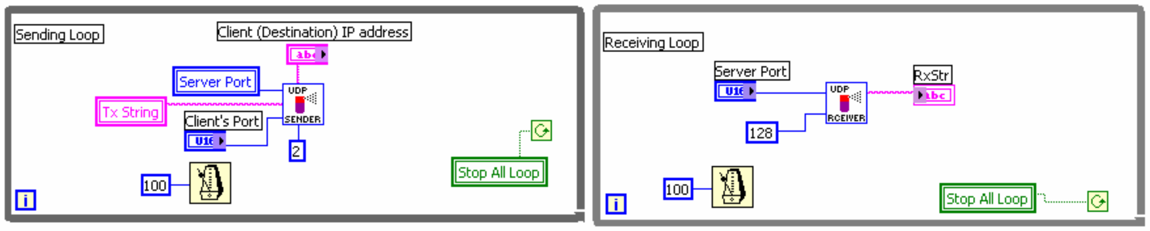


Figure 45. The UDP Transmit and Receive Loops of the External Communication Module.

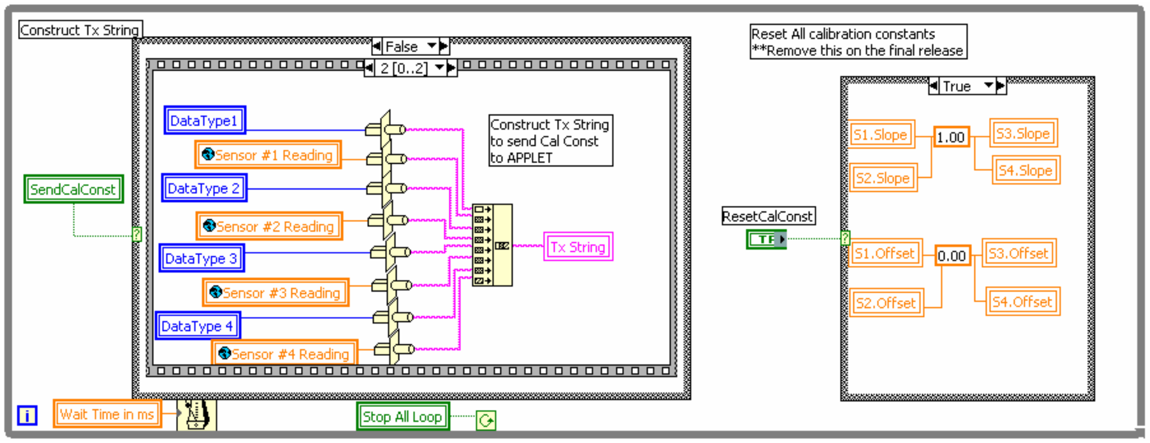


Figure 46. The Tx String Construction Thread of the External Communication Module.

3. Sensor Data Collector Module

The sensor data collector is designed to retrieve the sensor readings from all sensors in the virtual ship, adjusts them using the most recent calibration constants and stores them in a series of global variables *Sensor #X* (X is the sensor ID). Each sensor reading is monitored by a separate thread in the Labview program *SensorServerFinal.vi*. The virtual ship has totally four sensors. Therefore the sensor data collector module uses four different threads. The advantage of this design is that the readings of each sensor are not affected when one or more of the sensors in the virtual ship fail to response.

As shown in Figure 47, each thread of the sensor data collector performs similar tasks. Each communicates with the sensor via a TCP connection. For this connection, the sensor is the TCP server, and the sensor data collector is the TCP client. After retrieving the sensor readings from the sensor, the sensor data collector adjusts the sensor reading using the most recent calibration constants. As seen in Figure 41, these calibration constants are the outputs of the sensor calibration module. The adjusted sensors readings are stored in the global variables, so they can be accessed by lower level subVIs, ie the sensor calibration module during the calibration process.

The sensor data collector has two modes of operation: running and testing mode. In running mode, the sensor readings are retrieved from the sensor via a TCP connection. In testing mode, the sensor readings are provided by the controls on the front panel. The operation mode can be individually set for each sensor. As shown in Figure 43, the buttons on the left side of the figure are used to toggle the operation mode for each sensor. This feature is useful for testing and debugging during the development process.

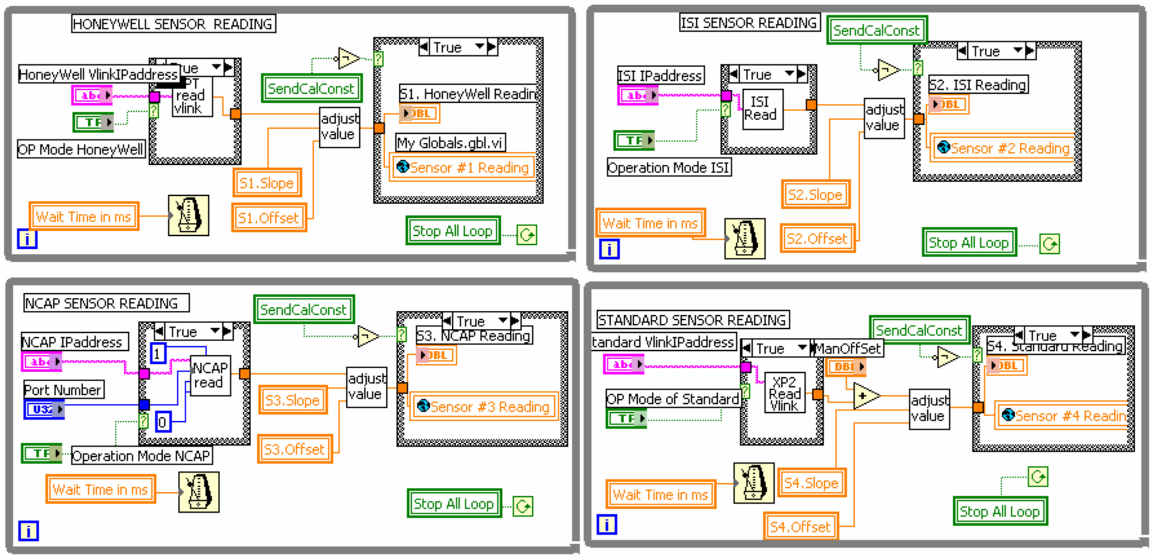


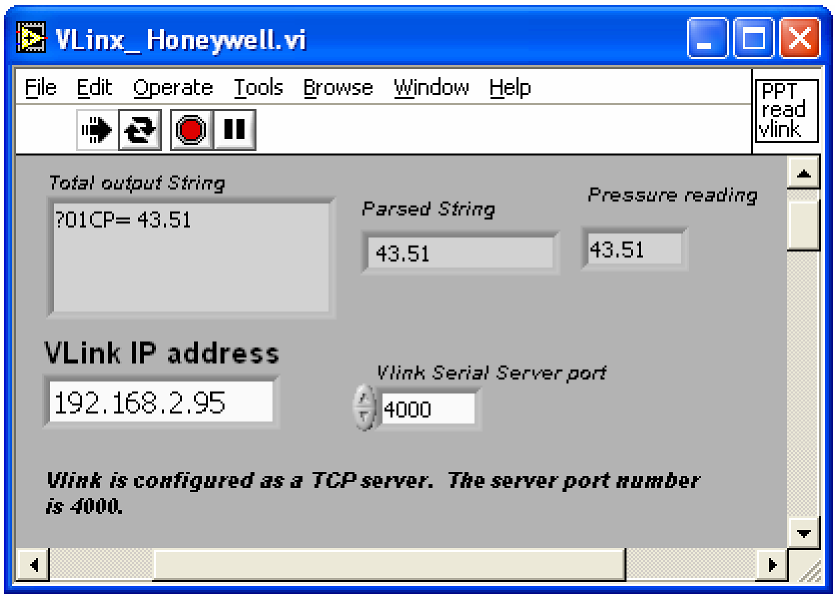
Figure 47. Four Sensor Monitoring Threads used on a Virtual Ship.

4. Sensor Interfaces

As discussed in Section IV.B.3, communication between the calibrating computer and sensors is bidirectional and device specific. The query strings and reading responses are different from sensor to sensor. A separate VI must be designed for each type of sensor as shown below.

a. PPT Sensor Interface

The subVI *Vlinx_Honeywell.vi*, as shown in Figures 48-49, is designed to retrieve sensor readings from the PPT. As stated in Section III.B.1, the PPT is integrated to the ship's LAN by a Vlinx RS232-to-Ethernet adapter. Therefore, the subVI *Vlink_Honeywell.vi* is designed to establish a TCP connection to the Vlinx adaptor on port 4000. On this connection, the Vlinx RS232-to-Ethernet adapter is the TCP server, and the subVI *Vlinx_Honeywell.vi* is the client. After a TCP connection being established, the query string **"00P1"** is sent to the sensor in ASCII format to request the sensor reading. The response from the sensor has the format of **"01CP=XX.XX"**. **XX.XX** is then parsed after the **"="** sign to extract the sensor reading.



Vlink is configured as a TCP server. The server port number is 4000.

Figure 48. Front Panel of the SubVI *Vlink_Honeywell.vi*.

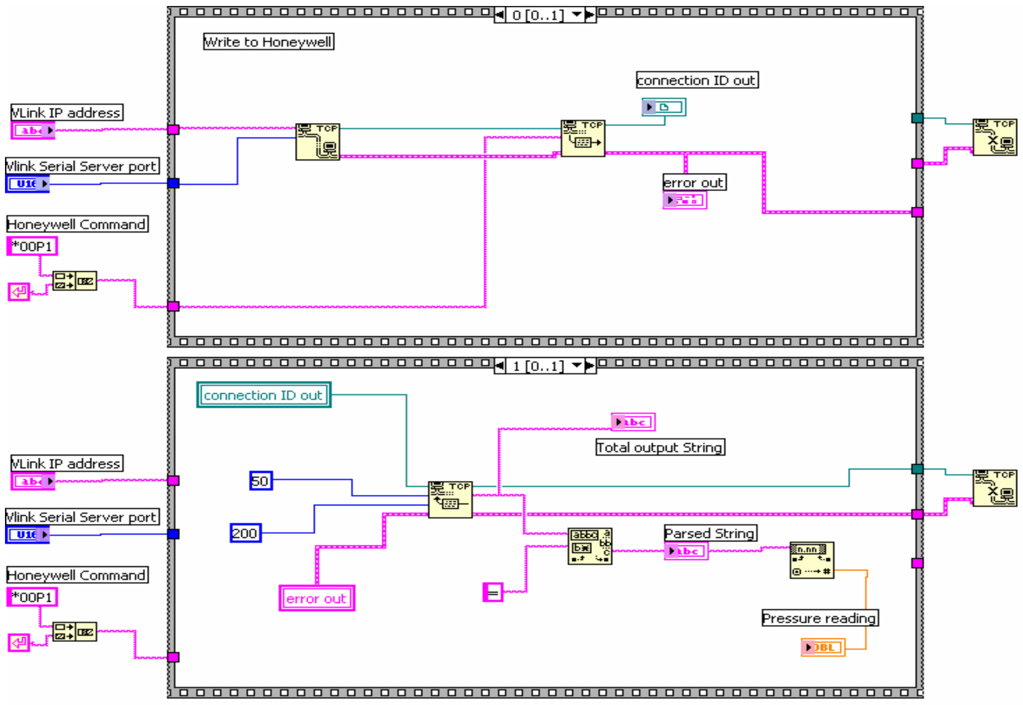
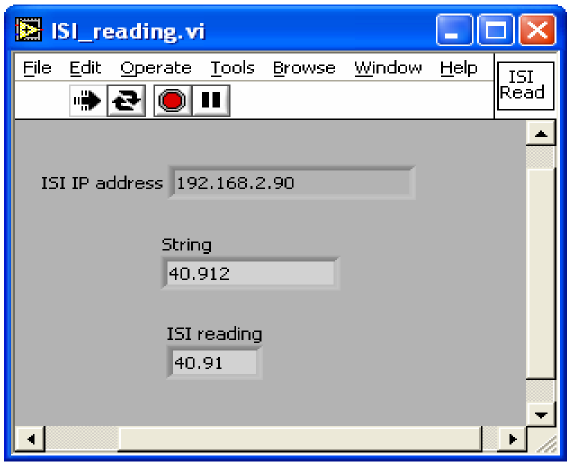


Figure 49. Block Diagram of the SubVI *Vlink_Honeywell.vi*.

b. *ISI Sensor Interface*

The subVI *ISI_reading.vi*, as shown in Figures 50-51, is designed to retrieve sensor readings from the ISI sensor. The subVI is designed based on the concept discussed in Section III.G.2. A data socket is used to establish a URL connection to the web server hosted on the ISI sensor. When a URL connection is established, the string "http://192.168.2.92/-$rdng" is sent over the URL connection. The ISI sensor responses back with the sensor reading in ASCII format. This reading is then converted into a double value and returned as an output of the subVI.



ISI IP address 192.168.2.90

String
40.912

ISI reading
40.91

Figure 50. Front Panel of the SubVI *ISI_reading.vi.*

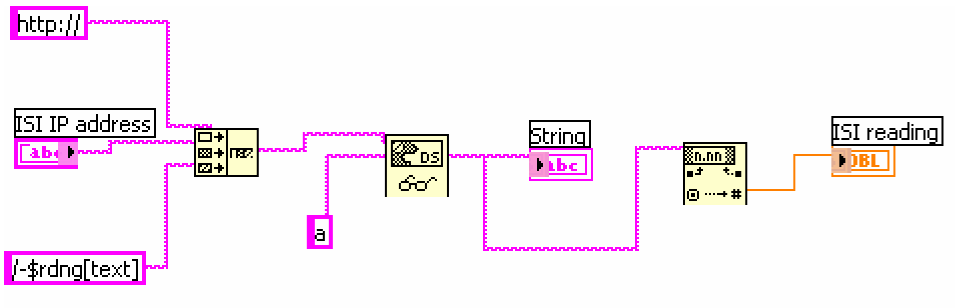
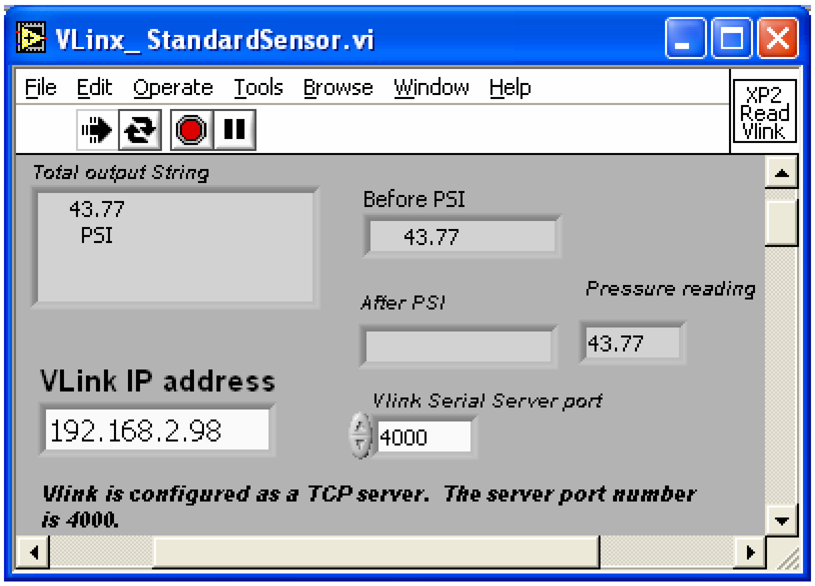


Figure 51. Block Diagram of the SubVI *ISI_reading.vi.*

c. PPC Interface

Interfacing with the PPC is similar to interfacing with the PPT, since the Vlinx RS232-to-ethernet adaptors are used to integrate both sensors to the ship's LAN. The only difference is the syntax of the command string and the response used to communicate with the sensor. As shown in Figures 52-53, the subVI *Vlink_StandardSensor.vi* is designed to implement the concept previously discussed in Section III.G.3. After establishing a TCP/IP connection to the Vlinx RS232-to-ethernet adapter on port 4000 of the IP address 192.168.2.98, a query string **"?P,U"** is sent to the PPC. The subVI then waits for the response from the PPC. The response from the PPC is returned will have the format of "XXX.XX PSI" in ASCII format. The sensor reading is extracted out of the response string, converted to a double value and returned as an output of the subVI.



Total output String: 43.77 PSI
Before PSI: 43.77
After PSI:
Pressure reading: 43.77
VLink IP address: 192.168.2.98
Vlink Serial Server port: 4000

Vlink is configured as a TCP server. The server port number is 4000.

Figure 52. Front Panel of the SubVI *Vlink_StandardSensor.vi.*

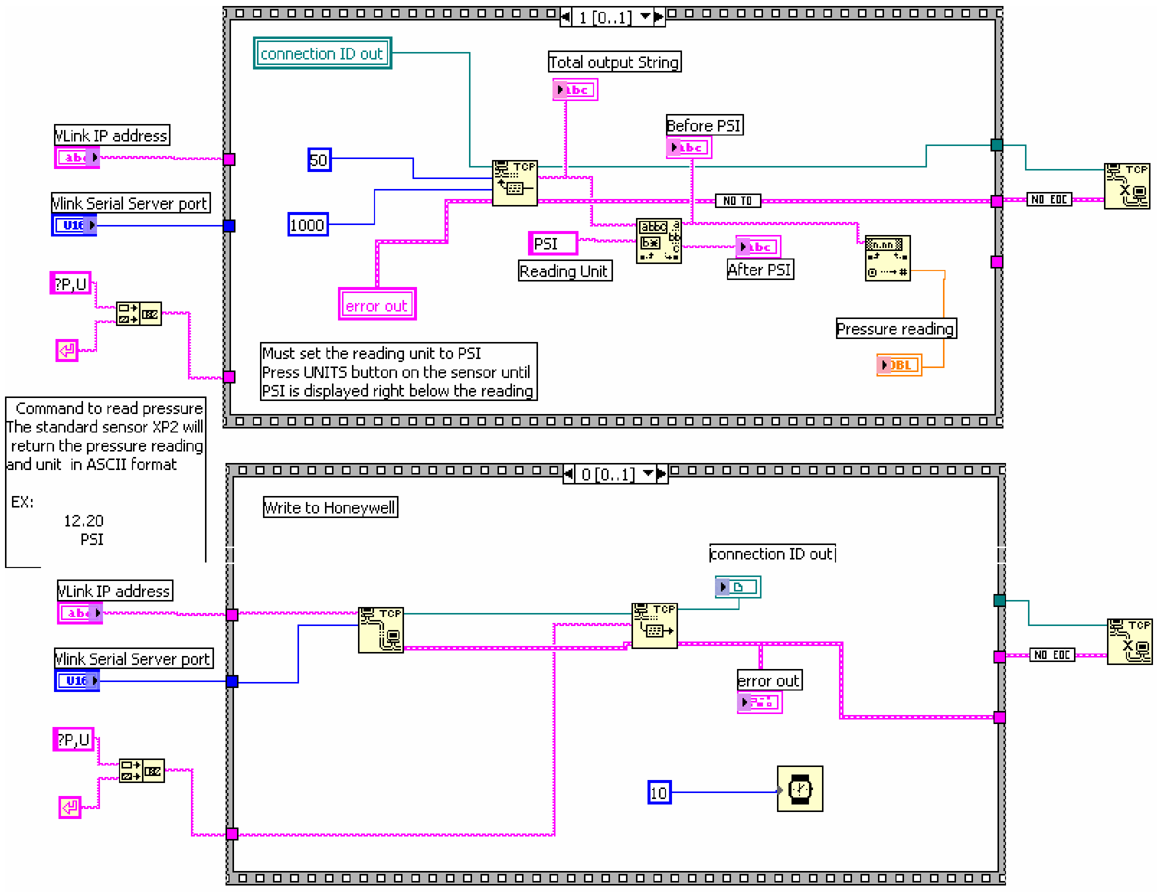


Figure 53. Block Diagram of the SubVI *Vlink_StandardSensor.vi.*

d. Analog Sensor Interface

The subVI *NCAPsensorReading.vi* is designed to retrieve sensor readings from the analog sensors connected to the NCAP box. As previously discussed in Section III.G.4, the NCAP box converts the analog readings from all eight analog sensors to digital readings and makes them available on the TCP/IP server. Unlike other sensors, all data exchanges between the client and the NCAP server are in binary format. After establishing a TCP/IP connection with the TCP/IP server on the NCAP box, a data packet with the format, as shown in Table 10, is sent to the NCAP box for authentication:

Format of Data Payload	Unit ID	Net Message
Size in byte	4	4
Data Type	Unsigned Long	Unsigned Long
Value	0	1

Table 10. Format of the Data Packet sent to the NCAP Server for Authentication.

When access is granted, the NCAP TCP server periodically sends readings of all eight channels to subVI *NCAPsensorRead.vi*. The sensor readings arranged in the format shown in Table 11.

Format of Data Payload	Packet Length	Sensor 1	Sensor 2	Sensor 3	Sensor 4	Sensor 5	Sensor 6	Sensor 7	Sensor 8
Size in byte	4	8	8	8	8	8	8	8	8
Data Type	Unsigned Long	Double	Double	Double	Double	Double	Double	Double	Double

Table 11. Format of the Data Packet from the NCAP TCP Server.

5. Command Handler Module

The Command handler module is a thread in the subVI *SensorServerFinal.vi*, as shown in Figure 54. It is designed to interpret the received commands and invoke appropriate modules or subVIs to execute the commands. The data payload of the received packet is organized in the format as shown in Table 6. The command handler module uses the subVI *UnpackRxString.vi* to extract each field out of the data payload. The command field (the first two bytes) is checked for the meaning before executing.

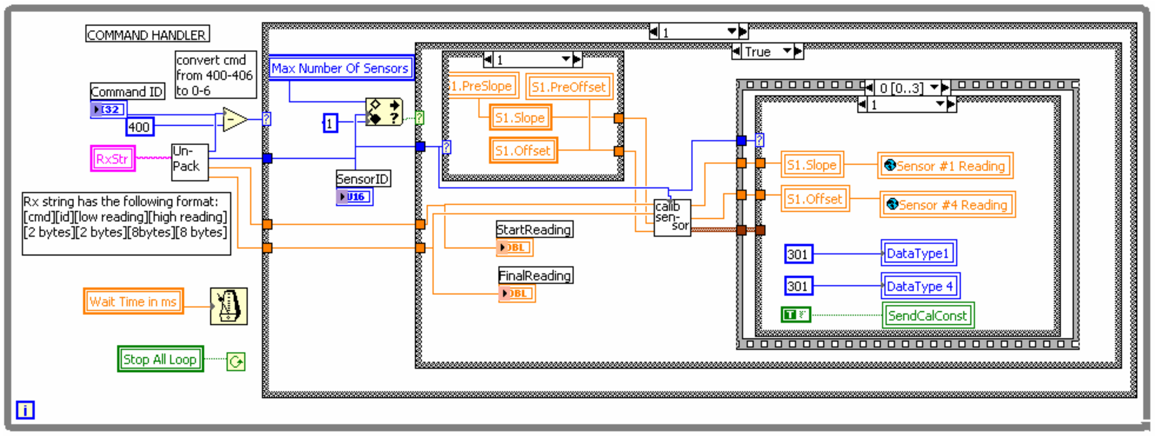


Figure 54. Block Diagram of the Command Handler Thread in the subVI *SensorServerFinal.vi*.

6. Sensor Calibration Module

Most of the function of the sensor calibration module is carried out by the subVI *CalibrateSensor.vi*, as shown in Figures 55-56. It is invoked by the command handler when the received command is 401 (Calibrate Sensor, Table 3). The subVI *CalibrateSensor.vi* are input with the sensor ID, minimum reading, maximum reading, and current calibration constants and returns the new calibration constants as the output. The new calibration constants are then saved into the sensor data file and sent to the operator. The sensor data files are currently decided to be stored in the folder C:\. The file names will have the format of *sensor#<sensor ID>data.txt*. If the file does not exist at run-time, a new file is created before storing new calibration constants.

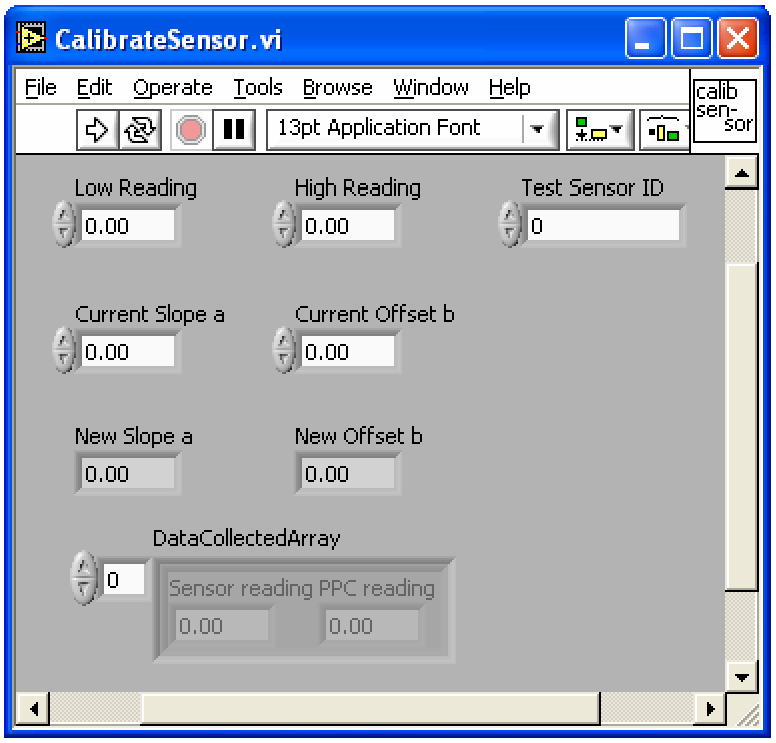


Figure 55. Front Panel of the subVI CalibrateSensor.vi.

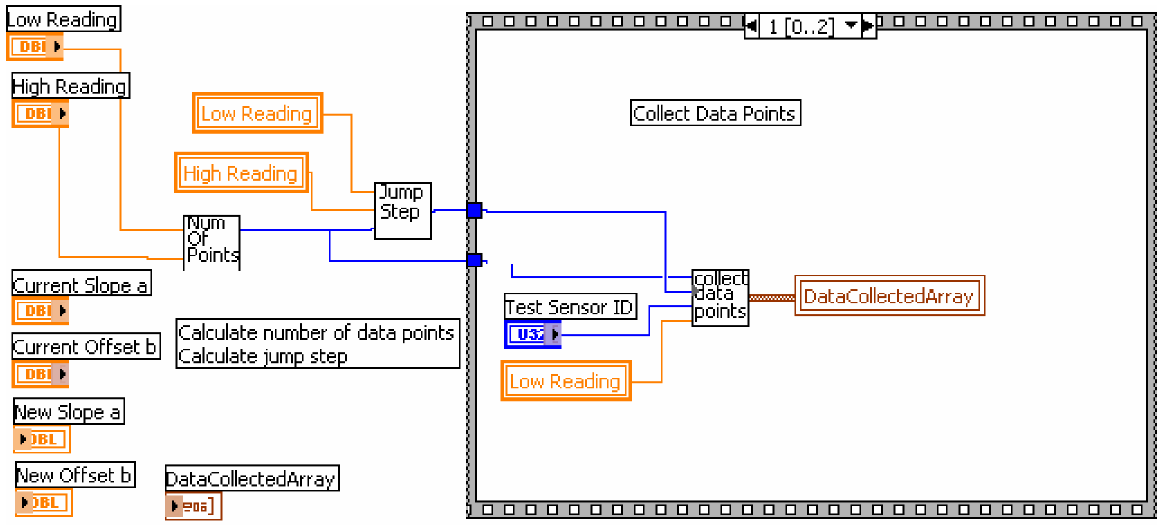
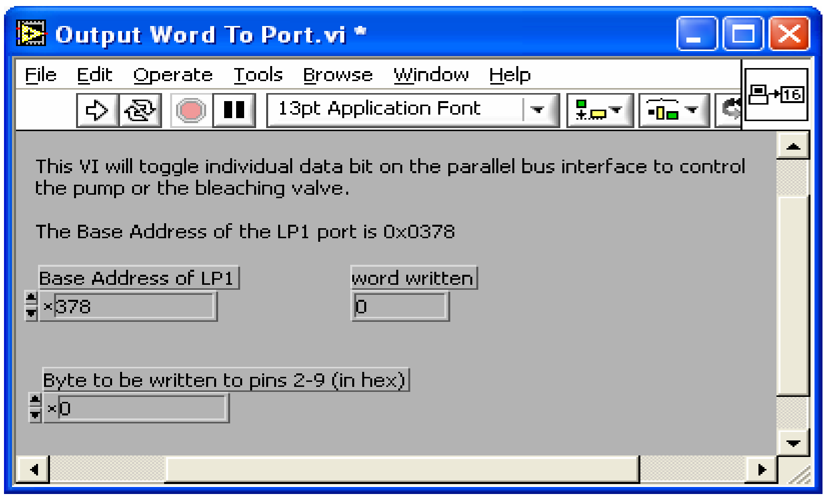


Figure 56. Block Diagram of the subVI CalibrateSensor.vi.

a. Interfacing with the Electrical Pump and the Opening Valve

The calibration module interfaces with the pump and the opening valve via the parallel port of the calibrating computer. The parallel port interface is enabled by the use of the subVI *OutputWordToPort.vi*, as shown in Figures 57-58. This subVI takes two inputs: the physical address of the parallel port and the content of the byte to be written to the port. On the calibration computer, the physical address of the parallel port is 0x378 (hex). The physical address of the parallel port can be found in the device manager, as shown in Figure 59. In Figure 60, the output on the parallel port has eight data bits. Only bit 0 and bit 1 are used to control the pump and the opening valve. To turn the pump ON, bit 0 is set LOW. To turn it OFF, bit 0 is set to HIGH. Similarly for the valve, bit 1 is set LOW to open the valve, and HIGH to close it. The complete list of output words in hexadecimal and their functions can be found in Table 12.



This VI will toggle individual data bit on the parallel bus interface to control the pump or the bleaching valve.

The Base Address of the LP1 port is 0x0378

Base Address of LP1: x378

word written: 0

Byte to be written to pins 2-9 (in hex): x0

Figure 57. Front Panel of the subVI *OutputWordToPort.vi*.

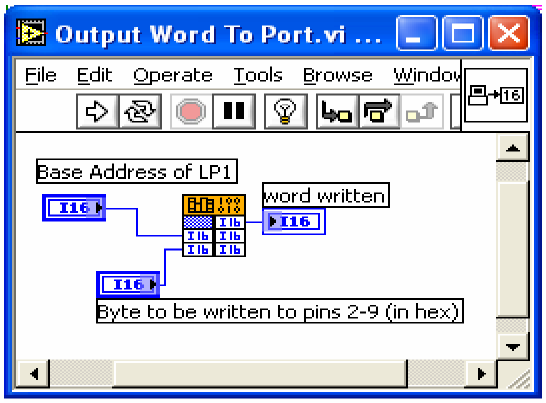
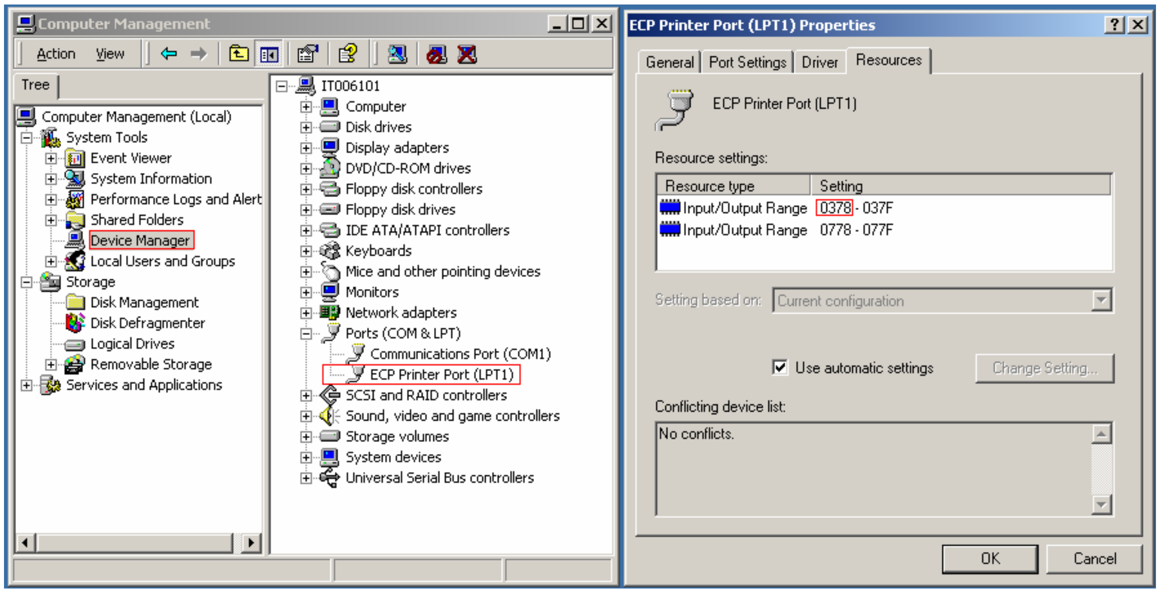

Output Word To Port.vi ...
File Edit Operate Tools Browse Window
Base Address of LP1
I16
word written
I16
I16
Byte to be written to pins 2-9 (in hex)

Figure 58. Block Diagram of of the subVI *OutputWordToPort.vi*.



ECP Printer Port (LPT1)

Resource settings:

Resource type	Setting
Input/Output Range	0378 - 037F
Input/Output Range	0778 - 077F

Setting based on: Current configuration

☑ Use automatic settings

Conflicting device list:

No conflicts.

Figure 59. Device Manager GUI shows the Physical Address of the Parallel Port on the Calibrating Computer.

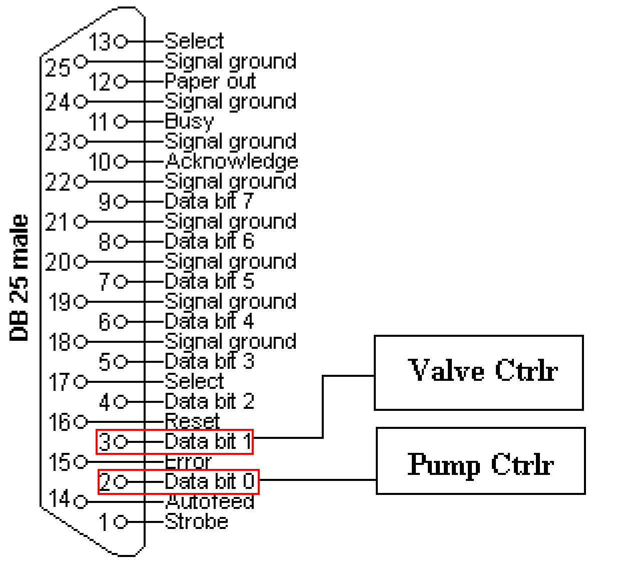


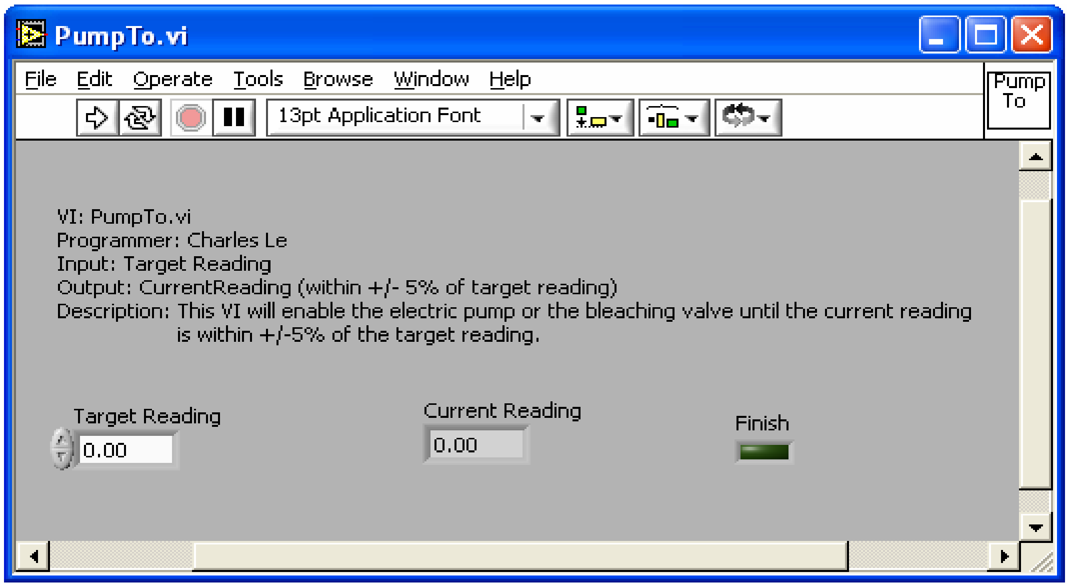
Figure 60. Pin Diagram of the Parallel Port.

Function	**Output Word in Hexadecimal**	**Output Word in Binary**
Pump ON	0x02	000000**10**
Open Valve	0x01	000000**01**
Pump OFF / Close Valve	0x03	000000**11**

Table 12. Output Words used to Control the Pump and the Opening Valve.

b. Pump and Valve Control Algorithm

During the calibration process, a number of data points are collected at various pressure points. A closed control loop is input with the PPC sensor reading and the target pressure reading to control the pump and the opening valve, as shown in Figures 61-62. If the PPC sensor reading is less than 95% of the target pressure, the pump will be turned ON, while the opening valve remains closed. If the PPC sensor reading is more than 105% of the target pressure, the opening valve will be opened, and the pump is turned OFF. The loop is repeated every two seconds, until the target pressure reading is reached.



VI: PumpTo.vi
Programmer: Charles Le
Input: Target Reading
Output: CurrentReading (within +/- 5% of target reading)
Description: This VI will enable the electric pump or the bleaching valve until the current reading is within +/-5% of the target reading.

Target Reading: 0.00
Current Reading: 0.00
Finish

Figure 61. Front Panel of the subVI *PumpTo.vi.*

.

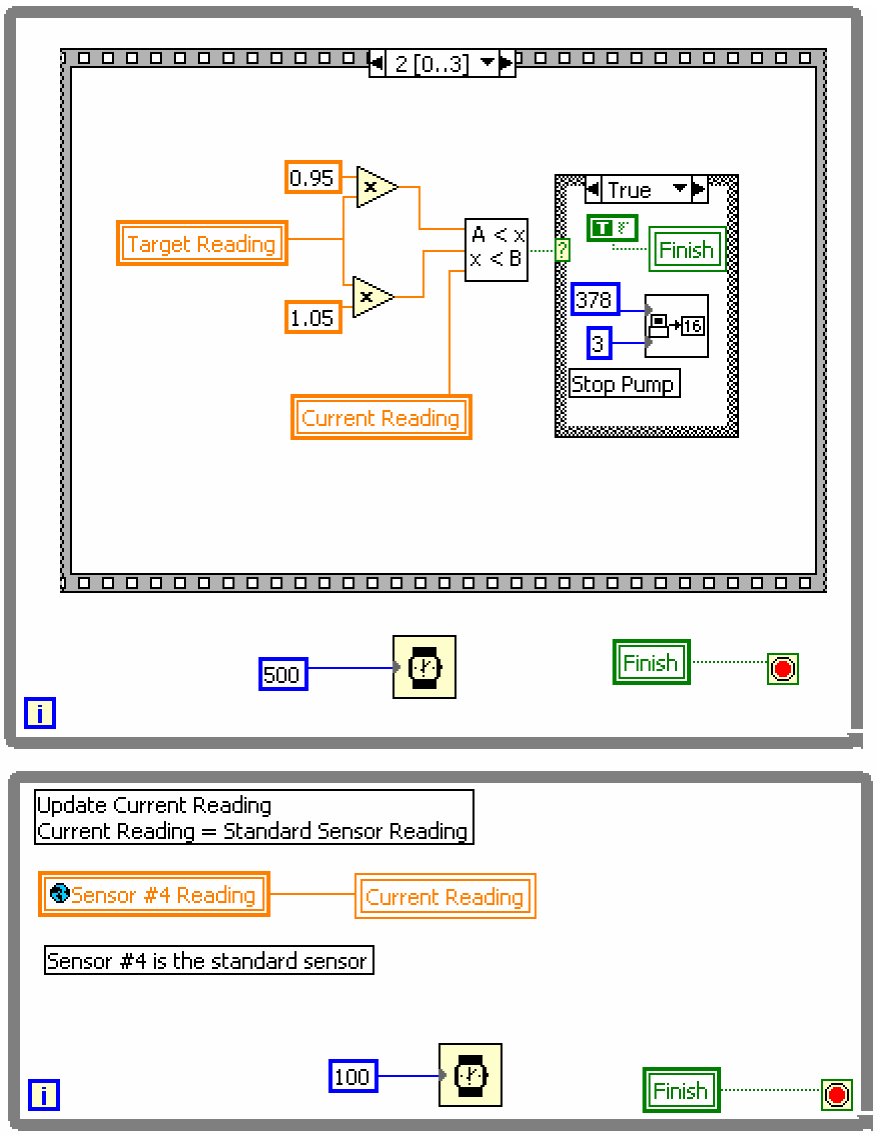


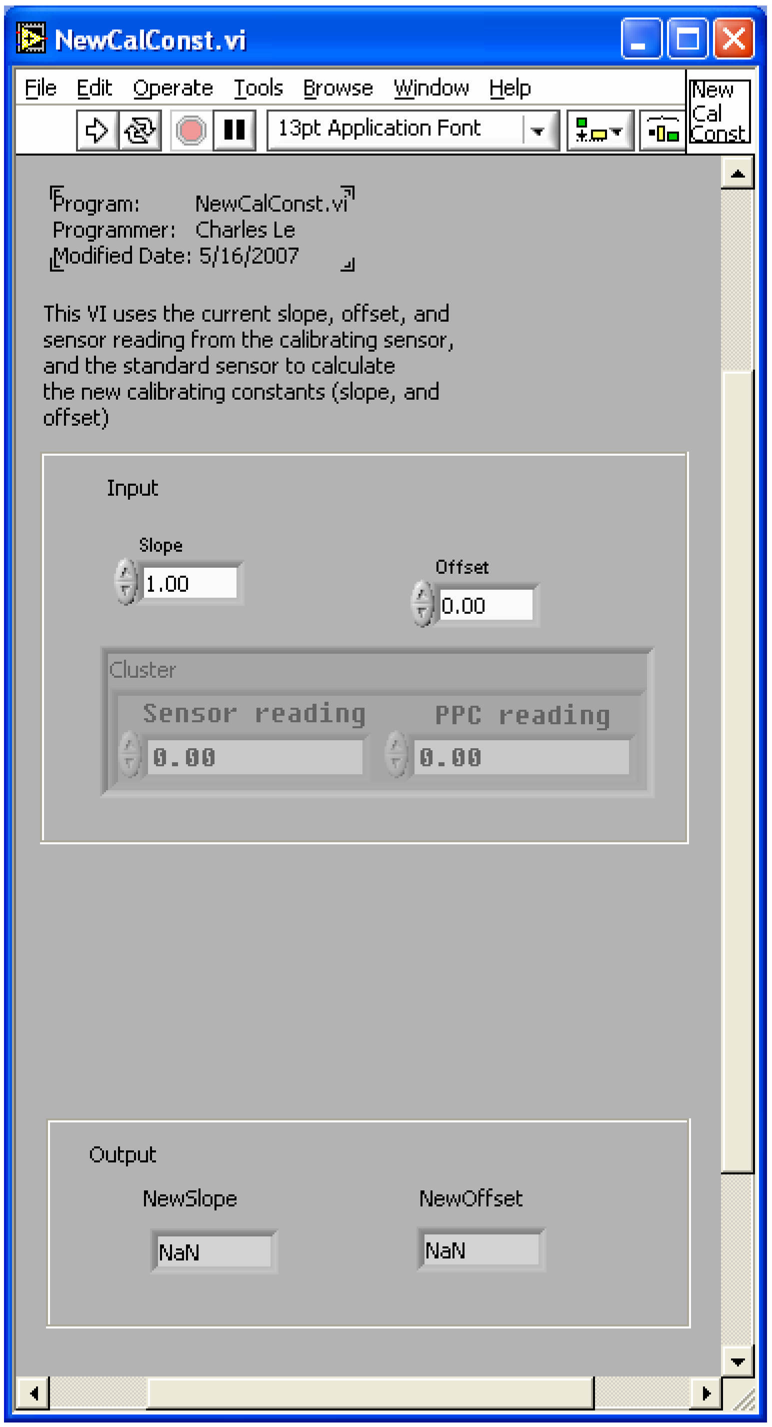
Figure 62. Block Diagram of the subVI *PumpTo.vi.*

c. *Calibration Algorithm*

The calibration algorithm begins by determining the number of data points to be collected based on the reading range of the sensor, as shown in Table 13. The jump step, the space between two consecutive collected data points is then calculated by rounding down the ratio of (Reading Range)/(Number of Data Point). The next target pressure reading is calculated by adding the current target pressure reading with the jump step. It is then input to the *subVI PumpTo.vi* to apply that newly calculated pressure on the sensor. Sensor readings of the target sensor and the standard sensor (PPC) are collected and stored in an array. After all data points are collected, the new calibration constants are calculated by the subVI *NewCalConstant.vi* (Figures 63-64), using the current calibration constants and the result of the linear fit of PPC sensor readings versus the target sensor readings. The previous calibration constants are updated with the current calibration constants, and the current calibration constants are updated with the new calibration constants. A record of the new calibration constants and collected data points are stored in the sensor data file by the subVI *SaveCalConst.vi* (Figures 68-69). Finally, they are sent to the web server computer in same format as shown in Tables 14, where the calibration constants are inserted into the same locations reserved for the readings of the target sensor and the standard sensor (PPC).

Sensor Reading Range in PSI Range = Max Reading – Min Reading	**Number of Data Points to be Collected**
80-100	5
40-79	4
20-39	3
Less than 20	0 (No Calibration)

Table 13. Number of Data Points for each Reading Range.



Program: NewCalConst.vi
Programmer: Charles Le
Modified Date: 5/16/2007

This VI uses the current slope, offset, and sensor reading from the calibrating sensor, and the standard sensor to calculate the new calibrating constants (slope, and offset)

Input

Slope	Offset
1.00	0.00

Cluster

Sensor reading	PPC reading
0.00	0.00

Output

NewSlope	NewOffset
NaN	NaN

Figure 63. Front Panel of the subVI *NewCalConst.vi.*

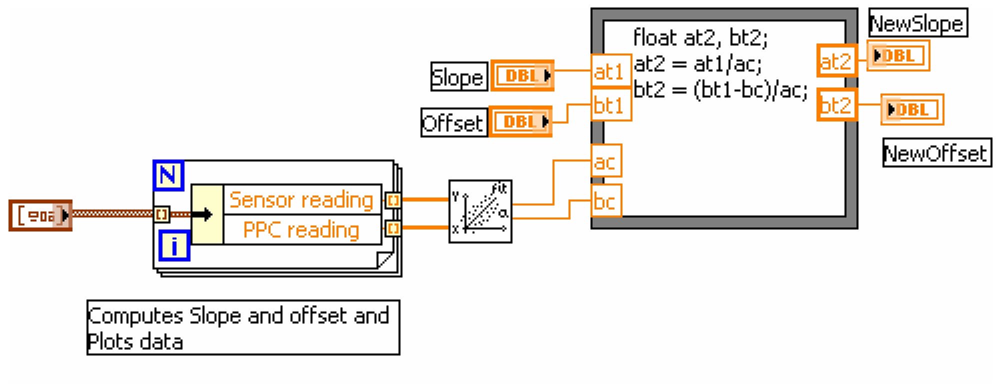
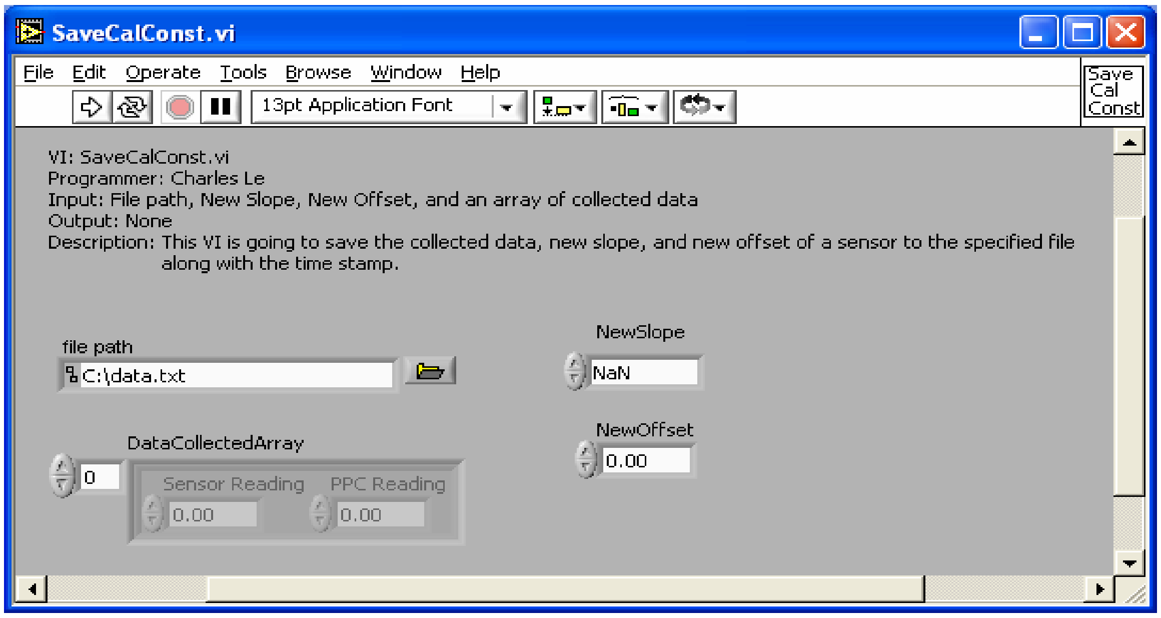


Figure 64. Block Diagram of the subVI *NewCalConst.vi.*



VI: SaveCalConst.vi
Programmer: Charles Le
Input: File path, New Slope, New Offset, and an array of collected data
Output: None
Description: This VI is going to save the collected data, new slope, and new offset of a sensor to the specified file along with the time stamp.

file path: C:\data.txt
NewSlope: NaN
NewOffset: 0.00
DataCollectedArray: 0
Sensor Reading: 0.00
PPC Reading: 0.00

Figure 65. Front Panel of the subVI *SaveCalConst.vi.*

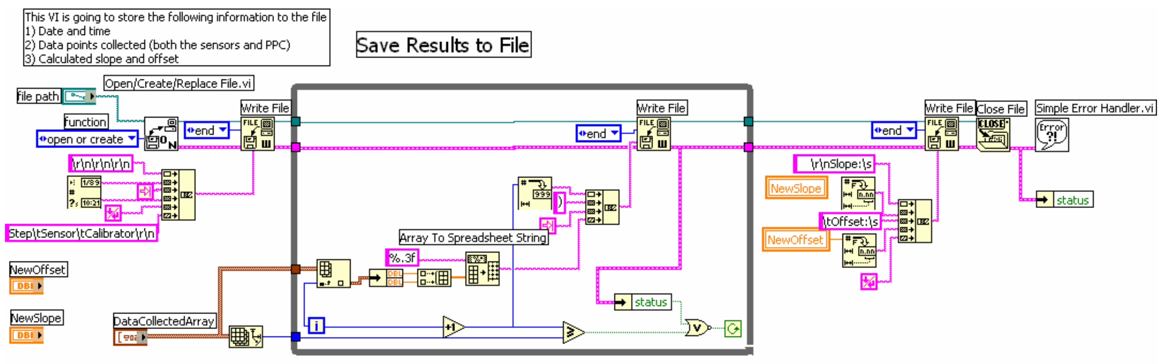


Figure 66. Block Diagram of the subVI *SaveCalConst.vi.*

	Message 1	**Reading 1**	**Message 2**	**Reading 2**	**Message 3**	**Reading 3**	**Message 4**	**Reading 4**
Data Written	301	Current Slope of Sensor 1	300	Sensor Reading 3	300	Sensor Reading 3	301	Current Offset of Sensor 1
Size in bytes	2	8	2	8	2	8	2	8
Data Type	Unsigned Short	Double	Unsigned Short	Double	Unsigned Short	Double	Unsigned Short	Double

Table 14. The Detail Format of the UDP Packet’s Payload that is used to send Calibration Constants of Sensor 1 to the Web Server Computer.

V. RESULTS AND LESSONS LEARNED

A. INTRODUCTION

Previous chapters have covered the algorithm, hardware, and software used to design and implement the automatic web-base sensor calibration system. This chapter sequentially analyzes the final product and design process from start to finish.

B. THE CONCEPT

This research was conducted with four main objectives. First is to reduce the personnel requirement by at least 50 percent. Second is to reduce the calibration time as much as 75 percent compared to the current calibration procedure used by the US Navy. Third is to further automate the calibration process, and provide a user friendly, web-based GUI to guide an operator through the calibration process. Fourth is to take advantage of the web technologies to allow operators to calibrate any selected shipboard sensor from anywhere on the world-wide-web. To safeguard the calibrating computer from a possible network attack, a second computer is used to host the web server and web pages. This means that three links of communication must be maintained at all time: between the calibrating computer and sensors, between the calibrating computer and the web server computer, and between the web server computer and the operator at some remote location. The sensor data flows from sensors to the calibrating computer, next to the web server computer, and finally to the operator. The commands from the operator flow in the reversed direction. The final demonstrated result is only 70 percent of the total work that had been put in the project. A good portion of the work had to be discarded due to unforeseen restrictions and additional requirements.

C. THE BEGINNING

Initially, the calibrating computer communicates with sensor via a variety of interfaces: Ethernet (RJ-45), RS-232, and Bluetooth. The decision is made to use the Vlinx RS232-to-ethernet adapter to convert all RS-232 and RS-485 interfaces to Ethernet. This new approach allows the calibrating computer to query sensors' readings via a single Ethernet interface instead of three different types of interfaces. This also makes it much easier to communicate with sensors, because they all begin with a TCP connection to the Vlinx RS232-to-ethernet adapter. A sensor interface module can easily be modified to interface with another sensor that also uses the Vlinx RS232-to-ethernet adapter for integration with the ship's LAN.

After detail analysis, it is found that most of the steps in the calibration process can be automated by software, except controlling the pressure applied on sensors. Initially, the PPC was used to manually pump or release pressure on sensors. This step could not be automated by software with the current hand pump. A decision was made to purchase the electrical pump and the solenoid valve. Two separate controller circuits were built in house to digitally turn on/off the power to the electrical pump and open/close the opening valve. The calibrating computer interfaces with the controller board via the parallel port. A closed-loop calibrating subVI was then developed to allow the operator to calibrate the selected sensor. The subVI was tested successfully with simulating data received from the web server computer, which was not yet supported by the web server computer. The automatic calibration is, therefore, only available locally at this time.

At this stage of the development, the concept of communication between the calibrating computer and the operator was relied on the TCP server running at all times on the web server computer. The TCP server is a server on port 4001, and a client on port 1512. When started, it initiates a TCP connection with the calibrating computer on port 1512 to get sensor readings. When a sensor web page is accessed by an operator, a Java applet is downloaded and run on the operator's computer. At initialization, this applet initiates a TCP connection to the TCP server on port 4001 of the web server computer. The project was almost complete based on this concept. The concept was working beautifully when demonstrated on NPS campus, and VPN connection. The sponsor at NAVSEA, however, was unable to see the demonstration. A series of tests was run, and it was determined that both the sponsor and the web server are on two separate secured networks. The firewall only allows the bidirectional traffic on port 80. Port 80, however, was used by the Internet Information Server (IIS) component of the

Window XP operating system to provide web service to clients. The TCP server can not use the same port to communicate with the applet.

D. THE ADJUSTMENT PHASE

It is decided that the project needs to rely on a different communication concept to communicate with the applet running on the operator’s computer. The new communication is found, but the change is quite radical. It discounts 90 percent of the work done on the web server computer, and 20 percent the work done on the calibrating computer. The new communication concept uses servlets and POST requests to establish communication between the applet and the calibrating computer. The IIS, however, does not support this feature. Therefore, it is replaced by the Tomcat web server, which can be downloaded at no cost at http://tomcat.apache.org. With the Tomcat web server, the bidirectional communication is re-established using POST requests and Java servlets. The new concept requires the calibrating computer to periodically send UDP packets containing sensor readings to the web server computer. When accessed by the operator, the applet periodically sends the POST commands to the web server, which invokes the servlet to response with sensor readings. The applet, then, extracts the sensor reading from the responses and displays them on the strip chart. A command button was added to the applet to allow operator sending calibration command to the calibrating computer. A small modification was made on the calibrating computer to integrate with this new function. The integration works seamlessly on the first time being integrated with the servlet. This time the sponsor at NAVSEA, San Diego Corona, CA is not only able to see the live update of sensor readings of a sensor located at Naval Postgraduate School, 500 miles away, but he is also able to calibrate it by one simple mouse click on the “*calibrate*” button on the browser. Two to three minutes, after clicking on the *calibrate* command button, the target sensor is calibrated, and the new calibration constants are saved on the hard drive of the computer. Comparing to the current calibration procedure, this is at least 90 percent improvement in calibration time. In the calibration process currently used by US Navy, three minutes are not even enough to hand pump the pressure to the first desired pressure point of the data collection process. At this point, all objectives listed in the executive summary have been met.

E. THE FINISHING TOUCHES

The same concept is further expanded to add the view of a web camera, and additional command buttons to allow the operator to see the lab setup, apply different pressures on sensors and request for calibration constants. With the manual control of the pump and the opening valve, the operator can control the pressure applied on the sensors to verify the result of the calibration. A reset button is also added to allow the operator to undo the result of the calibration, so the calibration process can be demonstrated again. The automatic web-base calibration system is made to be available online 24/7 for any NPS student, staff, or registered users. Non NPS users need to register their IP address to ITACS before being allowed to access the web site. This is the initial security measure to protect the system from the possible network attack.

F. LESSONS LEARNED

Quite a few lessons are learned throughout the development process. The five most important ones are:

- New features and functions must work with the existing system
- Unforeseen problems
- Strength and Weaknesses of Labview Programming Language
- Synchronization is important
- Poor performance on the NCAP box

1. Working with the Existing System

A new function or capability, regardless how advance it is, is usually useless if it does not interoperate with the existing system. Unfortunately, this is the common mistake made by developers. A lot of efforts is spent on developing new and advanced features. The interoperability is often under emphasized. Fortunately, in this project, the problem was detected early in the development process and corrected in time to demonstrate the feasibility of the project.

2. Unforeseen Problems

In many projects, major delays are caused by unforeseen problems. This project is no exception. Not knowing that the firewall restrictions are applied on both the virtual ship's network and the operator's network had discounted a good portion of the previous work. In the business world, this would translate into additional cost, missing deadline, and possible loss of contract. It is very important to make sure that all requirements are considered before initiating development, because changing requirements during development process can be very costly.

3. Strength and Weakness of Labview Programming Language

Labview is an excellent tool to interface with peripheral devices. Its graphical programming concept is easy to learn to program and to understand the existing code. If designed properly, Labview code can be highly reusable. Features provided by Labview make software reusability, portability, and multi-threading programming a lot easier than other programming languages. Their nice features, however, don't come without a cost. The major problem that almost every Labview developer faces is the backward compatibility problem. Labview software designed in older version of Labview development software no longer works on a different versions of Labview run-time engines. The graphical programming concept takes more time to modify or develop complex software.

4. Synchronization

In a complex system, synchronization is very important. Without synchronization, subsystems won't be able to work together to perform more complex tasks. In this project, there is a high demand for synchronization on the connectionless communication between the calibrating computer and the web server computer. There is a good chance that UDP packets sent by the web server computer are not received by the calibrating computer or UDP packets sent by the calibrating computer not received by the web server computer. This implies that a connection between the operator and the sensors calibration system is interrupted or lost. This problem is quickly corrected by making sure the receivers wait at least three to five times the transmission interval before timing out.

5. NCAP Box Performance Problem

The NCAP box was working fine with the software developed by NPS students. It does not perform like an embedded system when loaded with a Labview software package developed by 3eTI. At boot up, Window XP Embedded pops up a window indicating that the hard disk space is low. The software continues to run after the OK button is clicked. After five minutes, Window XP Embedded, again warns that the virtual memory is running low, and the Labview software package is frozen. A new NCAP box with latest software package is currently on order.

VI. CONCLUSIONS AND RECOMMENDATIONS

A. SUMMARY

This research has demonstrated the feasibility of an automatic web-based sensor calibration concept. With a strong emphasis on the uses of software, current networking technologies and commercial-off-the-shelf products, the tedious work of calibrating a sensor can be replaced by a few mouse clicks on a web browser. This allows an operator to calibrate a sensor from thousands of miles away.

B. CONCLUSION

After extensive testing and demonstration, it is concluded that this research has achieved all objectives listed in the executive summary. The results from testing and demonstration are quite optimistic. With this new concept of calibration, very little effort is required from the operator to calibrate a sensor, and the operator is not even required to be on the ship. The result of this research appears to provide good approach to tackle the problem of calibrating the ever-increasing number of sensors used on new ships.

C. RECOMMENDATION

The current work has demonstrated a high degree of feasibility and operational efficiency. There are, however, many opportunities for improvement. Future work needs to focus on portability, interoperability, and the network security aspects of the project. It is also important to develop a robust software architecture to collect readings from thousands of sensors.

1. Portability

The very nature of the sensor calibration business is its portability. The smaller, the lighter, and the fewer of connected wires the better. This concept needs to be applied on the automatic web-base sensor calibration system as much as possible using the COTS products and current technologies. The portability of the current work can be improved as following:

- Replace the AC powered pump with a DC powered pump
- Use batteries to power the pump, standard sensor, embedded computer, and the opening valve
- Use an IEEE 802.11 capable embedded computer system to control the pump and the valve via digital I/Os
- Use an IEEE 802.11 capable ultra-mobile PC to run the web server and host web pages

This new concept would reduce the current one-hundred pound automatic web-based calibration system to a fifteen-pound system. The size would be reduced from ten cubic feet to two cubic feet. With this improvement, the operator could bring on board the ship a light weight and self contained system that only requires a pipe connection to the target sensor and wireless access to the ship's LAN. This type of system would be much more deployable than the current system.

2. Interoperability

Higher interoperability implies easily deployable and shorter integration time. Future study needs to focus on the interoperability between the automatic web-based sensor calibration system and the existing hardware and software currently used on the ship. The software also needs to interoperate with other software used by SYSCAL, and provide answers to questions such as what software SYSCAL uses to store the calibration constants and what data format it expects if automated data entry is available. Interoperability with other software currently used by SYSCAL would prove that the automation concept could be expanded beyond the ship.

3. Network Security

Network security has always been a concern for network administrators and users. The availability of an automatic web-based sensor calibration system to a remote user also exposes the ship's LAN to external threats. The data exchange between users and web server could be altered or imitated by malicious hackers. Additional studies on this problem are needed. A quick way to solve this problem is to encode and decode all data exchanges between operators and the web server. A better approach would be incorporating the current security technology adopted by US Navy.

4. Robust Software Architecture

A robust software architecture is important in developing a complex and evolving software. All requirements and restrictions need to be carefully considered before laying down the software architecture, including possible additional functions for the later versions of software. One area that needs special attention is the multi-thread feature. The current architecture uses a separate thread for each sensor. This architecture, however, might face implementation problems when the number of sensors increases to hundreds or thousands. A different software architecture is needed to efficiently allocate CPU resources, so the system can maintain its reliability in the shipboard environment.

It is also important to choose a software development tool and platform that requires minimal efforts for modification or scalability. It is highly recommended that the C/C++ compiler running on Window environment is used for software development. C/C++ language offers the following advantages:

- Multi-thread programming are easy to implement
- Network programming and peripheral interface can be easily done
- There are a lot of available resources on the web
- Less effort is required for modification and scalability
- No incompatibility between different versions

All the advantages, however, do not come without a cost. The disadvantages of using programming in C or C++ are:

- It is a little more difficult (compared to Labview) to learn to program in C or C++

- It is more difficult to determine the logic implemented by another programmer

The disadvantages can be compensated by proper documentation of the code and good software development discipline.

APPENDIX - JAVA CODES

A. IMAGESERVLET.JAVA

```
/*  This is the ImageServlet.java, which reads a jpg image file
 *  and sends over to the calling Applet.
 *  Version 1.0
 *  June 27, 2007
 *  by Xiaoping Yun
 */

import java.io.*;
import java.awt.*;
import java.awt.image.*;
import javax.servlet.ServletException;
import javax.servlet.http.*;
import javax.servlet.*;
import javax.imageio.*;

public class ImageServlet extends HttpServlet {

public void doPost(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException {

        response.setContentType("application/octet-stream");

        try {
                File f = new File("C:\\Inetpub\\ftproot\\live\\image900.jpg");


                BufferedImage bufi = ImageIO.read(f);


                OutputStream os = response.getOutputStream();
                if (bufi != null) {  ImageIO.write(bufi, "jpg", os);  }

                os.flush();
                os.close();

                } catch (Exception e) {
                        //e.printStackTrace();
                        System.out.println("Error reading the image file. ");   }
        }

}
```

B. IMAGEAPPLET.JAVA

```
import java.applet.Applet;
import java.awt.*;
import java.awt.event.*;
import java.io.*;
import java.net.*;
import java.util.Date;

import java.awt.image.*;
import javax.imageio.*;

//////////////////////////////////////////////////////////////////////////

public class ImageApplet extends Applet implements Runnable {
            BufferedImage bufi;
            Font f = new Font("TimesRoman",Font.BOLD, 14);
            Date theDate;
            Thread imageLoadingThread;

///////////////////////////////////////////////////////////////////////////
            public void init() {

            }

///////////////////////////////////////////////////////////////////////////
public void start() {
            if (imageLoadingThread == null) {
                        imageLoadingThread = new Thread(this);
                        imageLoadingThread.start();
            }
}

////////////////////////////////////////////////////////////////////////////
public void stop () {
            if (imageLoadingThread != null) {
                        imageLoadingThread.stop();
                        imageLoadingThread = null;
            }
}

////////////////////////////////////////////////////////////////////////////

public void run() {
            while (true) {
                        theDate = new Date();
                        onSendData();
                        repaint();
                        try {
                          Thread.sleep(400);
                        }   catch (InterruptedException e) {  }
            }
}
///////////////////////////////////////////////////////////////////////////
public void paint(Graphics g) {
            g.setFont(f);
            g.setColor(Color.red);
            g.drawImage(bufi,0,0,this);
}
/////////////////////////////////////////////////////////////////////////////
public void update(Graphics g) {
		paint(g);
}
//////////////////////////////////////////////////////////////////////////////
private void onSendData() {
			try {
				// get input data for sending
				String input = "test 123";

				// send data to the servlet
				URLConnection con = getServletConnection();

				OutputStream outstream = con.getOutputStream();
				ObjectOutputStream oos = new ObjectOutputStream(outstream);
				oos.writeObject(input);
				oos.flush();
				oos.close();

				// receive result from servlet
				InputStream instr = con.getInputStream();

				bufi = ImageIO.read(instr);


			} catch (Exception ex) { ex.printStackTrace(); }
		}
////////////////////////////////////////////////////////////////////////////////////////////////
private URLConnection getServletConnection()
			throws MalformedURLException, IOException {

			URL urlServlet = new URL(getDocumentBase(), "image2");
			URLConnection con = urlServlet.openConnection();
			con.setDoInput(true);
			con.setDoOutput(true);
			con.setUseCaches(false);
			con.setRequestProperty(
				"Content-Type",
			//	"application/x-java-serialized-object");
				"application/octet-stream");
			return con;
		}
}
```

## C.	SENSORSERVLET.JAVA

```
/////////////////////////////////////////////////////////////////////////////////////////////////////
// Program:	SensorServlet.java
// Programmer: Xiaoping Yun and Charles Le
// Input: UDP packets from the calibrating computer, request from client,
//						and empty response to client
// Output: update response to client, and forward POST request to
//						the calibrating computer
// Description: This program is invoked by the web server to service a POST
//					request from client.
//						1) First step: It checks the received and store sensor reading
```

```
//                                          received from the calibrating computer.
//          2) Second step: extract content of the POST request, stuff
//                                                  the content to UDP package and send to the
calibrating
//                                                  computer.  Only the CALIBRATING command has a
full length
//                                                  packet of 20 byte.  Most of the packet is only 2 byte
long
//                                  3) Third step:    Send stored sensor reading to the applet
////////////////////////////////////////////////////////////////////////////////////////////////////


import java.io.*;
import java.net.*;
import java.util.*;


import javax.servlet.ServletException;
import javax.servlet.http.*;

/**
 * SensorServlet.
 */
public class SensorServlet extends HttpServlet {

        //Define constants
        private short SEND_SENSOR_READING = 400;
        private short CALIBRATE_SENSOR    = 401;
        private short START_PUMP          = 402;
        private short STOP_PUMP           = 403;
        private short OPEN_VALVE          = 404;
        private short CLOSE_VALVE         = 405;
        private short RESET_SENSOR        = 406;
        private short LABVIEW_SERVER_PORT = 1512;
        private short SENSOR_SERVLET_PORT = 1511;
        private short MAX_SENSOR_NUMBER   = 4;
        private short STANDARD_SENSOR_ID  = 3;
        private short UDP_RX_BUFF_SIZE    = 40;    // this is exactly for 4 sensors
        private short UDP_TX_BUFF_SIZE    = 1024;
        private short DEBUG               = 0;     //set to 1 for debug messages
        private short UDP_TIME_OUT        = 12000;

        DatagramSocket ReceiveFromLabviewSocket;
        DatagramSocket TxToLabviewSocket;
        byte[] buffer = new byte[UDP_RX_BUFF_SIZE];
        byte[] txbuffer = new byte[UDP_TX_BUFF_SIZE];
        InetAddress LabviewServerAddress;
        DatagramPacket packet, txpacket;


        short sensorID;        // received from applet.
        short sensorCommand;   // received from applet.
        double minReading = 0.0;
        double maxReading = 100.0;
        double[] sensorValueDouble = new double[MAX_SENSOR_NUMBER];
```

```
short[]  sensorStatus = new short[MAX_SENSOR_NUMBER];

////////////////////////////////////////////////////////////////////////////////////////////////////
// Function init()
// Programmer: Xiaoping Yun and Charles Le
// Input:
// Output:
// Description: This function creates an UDP socket to the calibrating computer
//            IP = 192.168.2.88, port=1511
////////////////////////////////////////////////////////////////////////////////////////////////////
public void init()
{
        try {
          ReceiveFromLabviewSocket = new DatagramSocket(SENSOR_SERVLET_PORT);
          ReceiveFromLabviewSocket.setReceiveBufferSize(UDP_RX_BUFF_SIZE);
          //ReceiveFromLabviewSocket.setSoTimeout(12000);


        } catch (SocketException e) {System.out.println("Failed to open UDP socket on port
                                                  1511."); }

        try {
          TxToLabviewSocket = new DatagramSocket(LABVIEW_SERVER_PORT);
          LabviewServerAddress = InetAddress.getByName("192.168.2.88");
        } catch(UnknownHostException e){ System.out.println(e); }
         catch(IOException e){ System.out.println(e); }
         //catch (InterruptedException e)  {}
}


////////////////////////////////////////////////////////////////////////////////////////////////////
// Function doPost()
// Programmer: Xiaoping Yun and Charles Le
// Input: request from client, and empty response to client
// Output: update response to client
// Description: This function is called by the web server to service a POST
//                              request from client.
//          1) First block: It checks the received and store sensor reading
//                        received from the calibrating computer.
//           2) Second block: extract content of the POST request, stuff
//                     the content to UDP package and send to the calibrating
//                     computer.  Only the CALIBRATING command has a full length
//                     packet of 20 byte.  Most of the packet is only 2 byte long
//          3) Third block:    Send stored sensor reading to the applet
////////////////////////////////////////////////////////////////////////////////////////////////////
public void doPost(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException
{

try { // receive sensor data from UDP Server of the calibrating computer

        ReceiveFromLabviewSocket.setSoTimeout(12000);
        packet = new DatagramPacket(buffer, buffer.length);
```

```
            ReceiveFromLabviewSocket.receive(packet);

            ByteArrayInputStream byteIn = new ByteArrayInputStream (packet.getData());
            DataInputStream dataIn = new DataInputStream (byteIn);

            for (int i=0; i<MAX_SENSOR_NUMBER; i++) {
                        sensorStatus[i] = dataIn.readShort();
                        sensorValueDouble[i] =  dataIn.readDouble();
            }
} catch (IOException e)
            {
              System.out.println("UDP connection timeout.");
              for (int i=0; i<MAX_SENSOR_NUMBER; i++) {
                        sensorStatus[i] = 0;
                        sensorValueDouble[i] =  0.0;
              }

            }


// Forward POST request to the calibrating computer
try {

            response.setContentType("application/x-java-serialized-object");

            // read data sent by applet.
            InputStream in = request.getInputStream();
            ObjectInputStream inputFromApplet = new ObjectInputStream(in);


            sensorCommand = inputFromApplet.readShort();
            sensorID = inputFromApplet.readShort();


            // print out the command sent by applet and sent it to Labview server.
            if (sensorCommand == CALIBRATE_SENSOR) {
              minReading = inputFromApplet.readDouble();
              maxReading = inputFromApplet.readDouble();
              if (DEBUG>0)
              System.out.println("sensor Command:  " + sensorCommand + "Sensor ID:  " +
                                  sensorID + " min:   " + minReading + " max:   " + maxReading);
              sendToLabview(CALIBRATE_SENSOR);
            }
            else if (sensorCommand != SEND_SENSOR_READING)
            {
              minReading = 0;
              maxReading = 0;
              if (DEBUG>0)
              System.out.println("sensor Command:  " + sensorCommand + "Sensor ID:  " +
                        sensorID + " min:   " + minReading + " max:   " + maxReading);
              sendToLabview(sensorCommand);
            }
            // send data to applet
            OutputStream outstr = response.getOutputStream();
            ObjectOutputStream oos = new ObjectOutputStream(outstr);

            oos.writeShort(sensorStatus[sensorID]);
            if (DEBUG>0)
                        System.out.println("sensorStatus["+sensorID+"] = "+ sensorStatus[sensorID]);
            oos.writeDouble(sensorValueDouble[sensorID]);
            oos.writeDouble(sensorValueDouble[STANDARD_SENSOR_ID]);
            oos.flush();
            oos.close();

            } catch (Exception e) {
                        e.printStackTrace();
            }
}


////////////////////////////////////////////////////////////////////////////////////////////////////////
// Function sendToLabview()
// Programmer: Charles Le
// Input: use class variables: CommandID, SensorID, minReading, and MaxReading
// Output: Send those variable over UDP socket to the Labview Sensor Server
// Description: This function send UDP packet to the Labview Sensor Server using
//                         the following format
//                      <cmd ID> <Sensor ID> <minReading> <maxReading>
// Size in bytes:   [2]                 [2]               [8]                    [8]
////////////////////////////////////////////////////////////////////////////////////////////////////////
public void sendToLabview(short Cmd) {
int i = 0, j=0;

try {

            // Send data to Labview Server via UDP


            // Construct UDP payload
            ByteArrayOutputStream byteOut = new ByteArrayOutputStream();
            DataOutputStream dataOut = new DataOutputStream (byteOut);
            dataOut.writeShort(Cmd);
            dataOut.writeShort(sensorID+1);  //convert sensor ID from 0-k to 1-(k+1)

            dataOut.writeDouble(minReading);
            dataOut.writeDouble(maxReading);

            // convert into byte array
            txbuffer = byteOut.toByteArray();

            //put payload buffer into the UDP packet and send it
            txpacket = new DatagramPacket(txbuffer, txbuffer.length, LabviewServerAddress ,
                                    LABVIEW_SERVER_PORT);
            TxToLabviewSocket.send(txpacket);
            dataOut.flush();
            dataOut.close();
  }
```

```
                catch(UnknownHostException e){ System.out.println(e); }
                catch(IOException e){ System.out.println(e); }
                //catch (InterruptedException e)  {}

  }


}
```

D. SENSORAPPLET.JAVA

```
//******************************************************************************
// SensorApplet.java:         Applet
//
// Version: 1.0
//
// Date: November 12, 2006
//
//
//******************************************************************************

import java.applet.*;
import java.awt.*;
import java.awt.event.*;
import java.net.*;
import java.io.*;
import java.util.*;

public class SensorApplet extends Applet implements Runnable
{



        Thread    sensorThread = null;

        //Define constants
        private short SEND_SENSOR_READING = 400;
        private short CALIBRATE_SENSOR    = 401;
        private short START_PUMP          = 402;
        private short STOP_PUMP           = 403;
        private short OPEN_VALVE          = 404;
        private short CLOSE_VALVE         = 405;
        private short RESET_SENSOR        = 406;
        private short GET_CURR_CAL_CONST  = 407;
        private short GET_PREV_CAL_CONST  = 408;

        private short MSG_SENSOR_READING  = 300;
        private short MSG_CURR_CAL_CONST  = 301;
        private short MSG_PREV_CAL_CONST  = 302;


        //private  String dataSource;             // the cgi URL
        // sensor ID: SmartSensor = 0, ISI = 1, analog = 2, Zigbee = 3
        // calibration = 400, ...
```

```
    private  short sensorID;
           private  short sensorCommand = SEND_SENSOR_READING;
           private  double minReading=0.0;
           private  double maxReading=100.0;
           private  short  DataMsg       = 300;
           private  double  CurrentSlope = 1.0;
           private  double  CurrentOffset = 0.0;
           private  double  PreviousSlope = 1.0;
           private  double  PreviousOffset = 0.0;
           private boolean  CurrCalConstFlg = false;
           private boolean  PrevCalConstFlg = false;

           private  int m_speed = 5;    // seconds between counter data retrievals     (parameter)

           private  Image offImage;                 // off-screen display for all
           private  Graphics offGraphic;            // zoomed displays

           private  int chartWidth;       // width of chart (grid) section in pixels (computed)
           private  int chartHeight;       // height of the chart (grid) section  (parameter)
           private  int xStep;                          // pixel length of line segments when plotting the data
  (parameter)
           private  int ySpace;                         // space above/below the max/min plotted values (parameter)

           private  int appWidth;                       // applet widow width (computed)
           private  int appHeight;                      // applet window height (computed)

           private  String theCount = ""; // the data vallues read from the source on a given execution cycle
           private  int cntWidth;                       // width of the y-axis labels (set now for 7 char max)
           private  final  int charSize = 12;        // font info
           private  final  int halfCharSize = 6;
           private  Dimension chartLLC;                     // position of the Lower Left Cornet of the grid
           private  Dimension chartTLC;                     // Top Left Corner position
           private  Dimension chartLRC;                     // Lower Right Corner position
           private  Dimension chartTRC;                     // Top Right Corner position
           private  Dimension maxLabelPos;                  // position of the max Y scals lebel
           private  Dimension midLabelPos;                  // position id the mid Y scale label
           private  Dimension minLabelPos;                  // position of the min Y scale label
           private  Dimension cntLabelPos;                  // position of the current count (X axix) label


           private  int[] theData;                    // the displayed data (a circular buffer)
           private  int[] theData2;
           private  int bufSize = 1;      // number of elements in theData buffer
           private  int inData;                       // index of the position of the next input element in theData
           private  int outData;                      // index of the first output position in theData
           private  boolean bufFull;

           private  int chartMaxY;                    // Max Y value (in data Units)
           private  int chartMinY;                    // Min Y value (in data units)


           private  Font font1;                        // the font to be used
           private  FontMetrics fontM;
```

```
            int sensorValueInt;
            double sensorValueDouble;
            String sensorValueString;

            int standardValueInt;
            double standardValueDouble;
            String standardValueString;

            int CurrSlopeInt;
            String CurrSlopeString;
            int CurrOffsetInt;
            String CurrOffsetString;

            int PrevSlopeInt;
            String PrevSlopeString;
            int PrevOffsetInt;
            String PrevOffsetString;


            Random rnum = new Random();




    // Parameter names.  To change a name of a parameter, you need only make
            // a single change.  Simply modify the value of the parameter string below.
    //--------------------------------------------------------------------------
            private final String PARAM_sensorID = "sensorID";
            private final String PARAM_bufSize = "bufSize";
            private final String PARAM_chartHeight = "chartHeight";
            private final String PARAM_xStep = "xStep";
            private final String PARAM_ySpace = "ySpace";
            private final String PARAM_speed = "speed";
        private final String PARAM_minReading = "minReading";
            private final String PARAM_maxReading = "maxReading";

            public SensorApplet()
            {
            }


////////////////////////////////////////////////////////////////////////////
public void init()
{

            setLayout(new FlowLayout(FlowLayout.LEFT, 0, 0));
            Button calibrationButton = new Button("Calibrate");
            Button resetButton = new Button("Reset");
            Button StartPumpButton = new Button("Start Pump");
            Button StopPumpButton = new Button("Stop Pump");
            Button OpenValveButton = new Button("Open Valve");
            Button CloseValveButton = new Button("Close Valve");
            Button GetCurrCal = new Button("Current Cal. Const");
            Button GetPrevCal = new Button("Prev. Cal. Const");
```

```
add(calibrationButton);
add(resetButton);
add(StartPumpButton);
add(StopPumpButton);
add(OpenValveButton);
add(CloseValveButton);
add(GetCurrCal);
add(GetPrevCal);

calibrationButton.addActionListener( new ActionListener() {
        public void actionPerformed(ActionEvent e) {
                onButtonClick(CALIBRATE_SENSOR);
        }
});
GetCurrCal.addActionListener( new ActionListener() {
        public void actionPerformed(ActionEvent e) {
                onButtonClick(GET_CURR_CAL_CONST);
        }
});
GetPrevCal.addActionListener( new ActionListener() {
        public void actionPerformed(ActionEvent e) {
                onButtonClick(GET_PREV_CAL_CONST);
        }
});

resetButton.addActionListener( new ActionListener() {
        public void actionPerformed(ActionEvent e) {
                onButtonClick(RESET_SENSOR);
        }
});
StartPumpButton.addActionListener( new ActionListener() {
        public void actionPerformed(ActionEvent e) {
                onButtonClick(START_PUMP);
        }
});
StopPumpButton.addActionListener( new ActionListener() {
        public void actionPerformed(ActionEvent e) {
                onButtonClick(STOP_PUMP);
        }
});
OpenValveButton.addActionListener( new ActionListener() {
        public void actionPerformed(ActionEvent e) {
                onButtonClick(OPEN_VALVE);
        }
});
CloseValveButton.addActionListener( new ActionListener() {
        public void actionPerformed(ActionEvent e) {
                onButtonClick(CLOSE_VALVE);
        }
});



String param;
```

```
param = getParameter(PARAM_sensorID);
if (param != null)
        sensorID = (short) Integer.parseInt(param);

//System.out.println("SensorID:  " + sensorID);


//=================================
param = getParameter(PARAM_speed);
if (param != null)
        m_speed = Integer.parseInt(param);


//=================================
param = getParameter(PARAM_bufSize);
if (param != null) {
        try {
                bufSize = Integer.parseInt(param);
        } catch (NumberFormatException e){
                bufSize = 1;
        }
} else {
        bufSize = 1;
}

//=================================
param = getParameter(PARAM_chartHeight);
if (param != null) {
        try {
                chartHeight = Integer.parseInt(param);
        } catch (NumberFormatException e){
                chartHeight = 0;
        }
} else {
        chartHeight = 0;
}

//=================================
param = getParameter(PARAM_xStep);
if (param != null) {
        try {
                xStep = Integer.parseInt(param);
        } catch (NumberFormatException e){
                xStep = 2;
        }
} else {
        xStep = 2;
}

//=================================
param = getParameter(PARAM_ySpace);
if (param != null) {
        try {
                yShpace = Integer.parseInt(param);
        } catch (NumberFormatException e){
                ySpace = 10;
        }
} else {
        xStep = 10;
}
chartMinY = ySpace;
chartMaxY = ySpace * 4;
//=================================
param = getParameter(PARAM_minReading);
if (param != null) {
        try {
                minReading = Double.parseDouble(param);
        } catch (NumberFormatException e){
                 minReading=0;
        }
} else {
        minReading = 0;
}
//=================================
param = getParameter(PARAM_maxReading);
if (param != null) {
        try {
                maxReading= Double.parseDouble(param);
        } catch (NumberFormatException e){
                 maxReading=0;
        }
} else {
        maxReading = 0;
}

//=================================
font1 = new Font("Arial", Font.BOLD, charSize);
setFont(font1);
fontM = getFontMetrics(font1);
cntWidth = fontM.stringWidth("MMMMMMM");

//=================================
Dimension dim = getSize();
appWidth = dim.width;
appHeight = dim.height;

if (chartHeight > 0) {
        chartWidth = (bufSize * xStep);

        chartLLC = new Dimension(cntWidth + 5, chartHeight + halfCharSize);
        chartTLC = new Dimension(chartLLC.width, halfCharSize);
        chartLRC = new Dimension(chartWidth + chartLLC.width, chartLLC.height);
        chartTRC = new Dimension(chartLRC.width, chartTLC.height);
        maxLabelPos = new Dimension(0, charSize);
        midLabelPos =  new Dimension(0, (chartLLC.height + chartTLC.height) / 2
                                                        + halfCharSize);
        minLabelPos = new Dimension(0, chartLLC.height + halfCharSize);
        cntLabelPos = new Dimension(chartLRC.width - (cntWidth / 2),
                                                                chartLLC.height + 5 + charSize);

        } else {
                chartWidth = 0;
                appHeight = charSize;
        }

        //=================================
        theData = new int[bufSize];
        theData2 = new int[bufSize];
        for (int i = 0; i < bufSize  ; i++)
                { theData[i] = ySpace * 2;  theData2[i] = ySpace * 2; }

        inData = 0;
        outData = 0;
        bufFull = false;
        //=================================


        //=================================

        resize(appWidth, appHeight);

}


/////////////////////////////////////////////////////////////////////////////

public void paint(Graphics g)
{
        g.drawImage(offImage, 0, 50, null);
}



/////////////////////////////////////////////////////////////////////////////

public final synchronized void  update(Graphics g)
{
        // override the default update method in order to
        // use double buffering of the graphics and avoid
        // watching the screens be drawn

        int i, j;
        int xFrom, xTo, yFrom, yTo;



                // communicate with the servlet
                try {


                URLConnection con = getServletConnection();
                SendPOSTrequest(con, SEND_SENSOR_READING);
                ReceivePOSTresponse(con);
```

```
// get integer and string value of the sensor reading
//for plotting purpose.
sensorValueInt = (int) sensorValueDouble;
String tmpString = Double.toString(sensorValueDouble);
int positonofperiod = tmpString.indexOf('.');

if (tmpString.length()>=positonofperiod+3) {
            sensorValueString = tmpString.substring(0, positonofperiod+3);
}
else
{
            sensorValueString = tmpString;
}

// get integer and string value of the standard reading
//for plotting purpose.
standardValueInt = (int) standardValueDouble;
tmpString = Double.toString(standardValueDouble);
positonofperiod = tmpString.indexOf('.');

if (tmpString.length()>=positonofperiod+3) {
             standardValueString = tmpString.substring(0, positonofperiod+3);
}
else
{
            standardValueString = tmpString;
}

// get integer and string value of the Current Slope
//for plotting purpose.
CurrSlopeInt= (int) CurrentSlope;
tmpString = Double.toString(CurrentSlope);
positonofperiod = tmpString.indexOf('.');

if (tmpString.length()>=positonofperiod+3) {
             CurrSlopeString = tmpString.substring(0, positonofperiod+3);
}
else
{
            CurrSlopeString = tmpString;
}

// get integer and string value of the Current Offset
//for plotting purpose.
CurrOffsetInt= (int) CurrentOffset;
tmpString = Double.toString(CurrentOffset);
positonofperiod = tmpString.indexOf('.');

if (tmpString.length()>=positonofperiod+3) {
             CurrOffsetString = tmpString.substring(0, positonofperiod+3);
}
else
{
            CurrOffsetString = tmpString;
```

```
		}

		// get integer and string value of the Current Slope
		//for plotting purpose.
		PrevSlopeInt= (int) PreviousSlope;
		tmpString = Double.toString(PreviousSlope);
		positonofperiod = tmpString.indexOf('.');

		if (tmpString.length()>=positonofperiod+3) {
			 PrevSlopeString = tmpString.substring(0, positonofperiod+3);
		}
		else
		{
			PrevSlopeString = tmpString;
		}

		// get integer and string value of the Current Offset
		//for plotting purpose.
		PrevOffsetInt= (int) PreviousOffset;
		tmpString = Double.toString(PreviousOffset);
		positonofperiod = tmpString.indexOf('.');

		if (tmpString.length()>=positonofperiod+3) {
			 PrevOffsetString = tmpString.substring(0, positonofperiod+3);
		}
		else
		{
			PrevOffsetString = tmpString;
		}





	} catch (Exception ex) {
		ex.printStackTrace();
	}


	if (offImage == null) {
		offImage = createImage(appWidth, appHeight);
		offGraphic = offImage.getGraphics();
		offGraphic.setFont(font1);
	}

	offGraphic.setColor(Color.white);
	offGraphic.fillRect(0,0,appWidth, appHeight);


	if (chartHeight > 0) {
		// draw the chart

		// get the new data value
		try {
```

```
		//theData[inData] = Integer.parseInt(sensorValueString);
		theData[inData] = sensorValueInt;
		theData2[inData] = standardValueInt;
	} catch (NumberFormatException e){
		theData[inData] = ySpace;
		theData2[inData] = ySpace;
	}

	inData = (inData + 1) % bufSize;
	if (inData == bufSize - 1)
		bufFull = true;


	// adjust the min/max Y values and scale values
	// if needed to keep a good data resolution visible

	chartMaxY = ((getMaxData() / ySpace) * ySpace) + ySpace;
	chartMinY = ((getMinData() / ySpace) * ySpace);
	if (chartMaxY == chartMinY)
		chartMinY -= ySpace / 2;

	double yM = (((double) chartHeight) /
					(double)(chartMaxY - chartMinY));
	double yA = - ((yM * (double)chartMaxY) - (double)chartHeight);

	// now draw the labels

	offGraphic.setColor(Color.black);
	offGraphic.drawString(""+chartMaxY, cntWidth -
					fontM.stringWidth("" + chartMaxY), maxLabelPos.height);
	offGraphic.drawString(""+ (chartMaxY - ((chartMaxY - chartMinY)/ 2)),
	cntWidth - fontM.stringWidth("" + (chartMaxY / 2)), midLabelPos.height);
	offGraphic.drawString(""+chartMinY, cntWidth - fontM.stringWidth("" +
					chartMinY), minLabelPos.height );

	// now draw the grid
	offGraphic.setColor(Color.lightGray);
	offGraphic.drawLine(chartTLC.width - 3, chartTLC.height,
					chartTRC.width, chartTRC.height);
	offGraphic.drawLine(chartTLC.width - 3, midLabelPos.height -
		halfCharSize, chartTRC.width, midLabelPos.height - halfCharSize);
	offGraphic.drawLine(chartTLC.width - 3, chartLLC.height, chartTRC.width,

		chartLLC.height);
	for (i = chartTLC.width; i <= chartTRC.width; i += xStep*2)
		offGraphic.drawLine(i, chartTLC.height, i, chartLLC.height + 3);

	// now Draw the Data
	offGraphic.setColor(Color.blue);

	// the data trace
	xFrom = cntWidth + 5;
	xTo = xFrom + xStep;

	for (i = outData, j = ((i + 1) % bufSize), yFrom = chartLLC.height -
            (int)(((double)theData[outData]) * yM + yA); j != inData;
            i = ((i + 1) % bufSize), j = (j + 1) % bufSize) {

            yTo = chartLLC.height - (int)(((double)theData[j]) * yM + yA);
            offGraphic.drawLine(xFrom, yFrom, xTo, yTo);
            offGraphic.drawLine(xFrom, yFrom+1, xTo, yTo+1);
            yFrom = yTo;
            xFrom = xTo;
            xTo += xStep;
}

// trace the standard data
offGraphic.setColor(Color.green);

xFrom = cntWidth + 5;
xTo = xFrom + xStep;
for (i = outData, j = ((i + 1) % bufSize), yFrom = chartLLC.height -
            (int)(((double)theData2[outData]) * yM + yA); j != inData;
            i = ((i + 1) % bufSize), j = (j + 1) % bufSize) {

            yTo = chartLLC.height - (int)(((double)theData2[j]) * yM + yA);
            offGraphic.drawLine(xFrom, yFrom, xTo, yTo);
            offGraphic.drawLine(xFrom, yFrom+1, xTo, yTo+1);
            yFrom = yTo;
            xFrom = xTo;
            xTo += xStep;
}



if (bufFull)
            outData = (outData + 1) % bufSize;

// finally, draw the cur data vert line and the count value
xTo -= xStep;
offGraphic.setColor(Color.red);
offGraphic.drawLine(xTo, chartTLC.height, xTo, chartLLC.height + 3);

// draw the legend
offGraphic.setColor(Color.blue);
offGraphic.drawString(sensorValueString + "  Sensor Reading",  80, 10);

offGraphic.setColor(Color.green);
offGraphic.drawString(standardValueString + "  Standard Reading",80,20);

if (CurrCalConstFlg)
{
            // draw the legend
            offGraphic.setColor(Color.blue);
            offGraphic.drawString(CurrSlopeString  + "  Current Slope",250, 10);

            offGraphic.setColor(Color.blue);
            offGraphic.drawString(CurrOffsetString  + "  Current Offset",250,20);
}
```

```
                if (PrevCalConstFlg)
                {
                        // draw the legend
                        offGraphic.setColor(Color.green);
                        offGraphic.drawString(PrevSlopeString + " Previous Slope", 400,10);

                        offGraphic.setColor(Color.green);
                        offGraphic.drawString(PrevOffsetString + " Previous Offset",400,20);
                }

        }
        else {
                // just display the counter value
                offGraphic.setColor(Color.black);
                offGraphic.drawString(sensorValueString, 0, appHeight);
        }

        paint(g);
}




////////////////////////////////////////////////////////////////////////////

        private int getMaxData()
        {
                int max = Integer.MIN_VALUE;
                int i,j;
                if (!bufFull && inData == 1)
                        return theData[outData] + ySpace;
                for (i = outData; i != inData; i = (i + 1) % bufSize)
                if (max < theData[i])
                        max = theData[i];
                return  max;
        }

////////////////////////////////////////////////////////////////////////////
        private int getMinData()
        {
                int min = Integer.MAX_VALUE;
                int i,j;
                if (!bufFull && inData == 1)
                        return theData[outData] - ySpace;;
                for (i = outData; i != inData; i = (i + 1) % bufSize)
                if (min > theData[i])
                        min = theData[i];
                return min;
        }




////////////////////////////////////////////////////////////////////////////
```

```
        public void destroy()
        {



        }


////////////////////////////////////////////////////////////////////////////////////

        public String getAppletInfo()
        {
                return "Name: Sensor Applet \r\n" +
                                "2007 by Xiaoping Yun and Charles Ler\n" +
                                "Version 1.02.\r\n";
        }


/////////////////////////////////////////////////////////////////////////////////////

public String[][] getParameterInfo()
{
        String[][] info =
        {
                {PARAM_sensorID, "String", "ID assigned to each sensor"},
                {PARAM_bufSize, "String", "size of the data buffer"},
                {PARAM_chartHeight, "String", "height in pixels of the chart section"},
                {PARAM_xStep, "String", "pixels per data element when plotted"},
                {PARAM_ySpace, "String", "space above/below the max/min data values"},
                {PARAM_speed,"String","number of seconds b/w counter data retrievals"},
        };
        return info;
}

/////////////////////////////////////////////////////////////////////////////////////
        private URLConnection getServletConnection()
                throws MalformedURLException, IOException {

                // Assuming that html file containing this applet is
                // located at htpp://..../app/, then url-pattern
                // in web.xml should be  /anything/echo2 or /anything/*
                // If the html file containing this applet is located at
                // http://.../app/test/, then url-pattern in web.xml should
                // be  /test/anything/echo2.

                URL urlServlet = new URL(getDocumentBase(), "anything/sensorServlet");

                URLConnection con = urlServlet.openConnection();

                // konfigurieren
                con.setDoInput(true);
                con.setDoOutput(true);
                con.setUseCaches(false);
                con.setRequestProperty(
                        "Content-Type",
                        "application/x-java-serialized-object");

                // und zurückliefern
                return con;
        }


////////////////////////////////////////////////////////////////////////////////////////
        private void onButtonClick(short ButtonCmd) {

                System.out.println("Started the automated calibration ... ");


                        // communicate with the servlet
                        try {

                        URLConnection con = getServletConnection();
                        SendPOSTrequest(con, ButtonCmd);
                        ReceivePOSTresponse(con);



                } catch (Exception ex) {
                        ex.printStackTrace();
                }




        }

///////////////////////////////////////////////////////////////////////////////
private void SendPOSTrequest(URLConnection con, short Cmd) {
        try {
                // send data to the servlet
                //URLConnection con = getServletConnection();
                OutputStream outstream = con.getOutputStream();
                ObjectOutputStream oos = new ObjectOutputStream(outstream);
                oos.writeShort(Cmd);
                oos.writeShort(sensorID);
                oos.writeDouble(minReading);
                oos.writeDouble(maxReading);
                oos.flush();
                oos.close();
        } catch (Exception ex) {
                ex.printStackTrace();
        }
}
///////////////////////////////////////////////////////////////////////////////
private void ReceivePOSTresponse(URLConnection con) {
        try {
                // receive sensor readings from servlet
                InputStream instr = con.getInputStream();
                ObjectInputStream inputFromServlet = new ObjectInputStream(instr);
                DataMsg = inputFromServlet.readShort();
```

```
				//System.out.println(DataMsg  + "  DataMsg ");
				if (DataMsg==MSG_SENSOR_READING)
				{
					sensorValueDouble = inputFromServlet.readDouble();
					standardValueDouble = inputFromServlet.readDouble();
				}
				else if (DataMsg==MSG_CURR_CAL_CONST)
				{
					CurrentSlope = inputFromServlet.readDouble();
					CurrentOffset = inputFromServlet.readDouble();
					CurrCalConstFlg = true;
				}
				else if (DataMsg==MSG_PREV_CAL_CONST)
				{
					PreviousSlope = inputFromServlet.readDouble();
					PreviousOffset = inputFromServlet.readDouble();
					PrevCalConstFlg = true;
				}

				inputFromServlet.close();
				instr.close();
			} catch (Exception ex) {
				ex.printStackTrace();
			}
		}
		///////////////////////////////////////////////////////////////////////

		public void start()
		{
			if (sensorThread == null)
			{
				sensorThread = new Thread(this);
				sensorThread.start();
			}
		}


		///////////////////////////////////////////////////////////////////////

		public void stop()
		{
			if (sensorThread != null)
			{
				sensorThread = null;
			}
		}





		///////////////////////////////////////////////////////////////////////

		public void run()
		{
```

```
            Thread thisThread = Thread.currentThread();
            while (sensorThread == thisThread)
            {
                        try
                        {
                                    repaint();
                                    thisThread.sleep(m_speed * 200);
                        }
                        catch (InterruptedException e)
                        {
                                    stop();
                        }
            }
}  //end of run()

}   //end of SensorApplet class
```

E. WEB.XML

```
<!DOCTYPE web-app
   PUBLIC "-//Sun Microsystems, Inc.//DTD Web Application 2.3//EN"
   "http://java.sun.com/dtd/web-app_2_3.dtd">

<web-app>

   <!-- General description of your web application -->
   <display-name>Echo Servlet</display-name>
   <description>
   Echo Servlet
   </description>

   <!-- define servlets and mapping -->
   <servlet>
      <servlet-name>sensorServlet</servlet-name>
      <servlet-class>SensorServlet</servlet-class>
   </servlet>
   <servlet-mapping>
      <servlet-name>sensorServlet</servlet-name>
      <url-pattern>/anything/sensorServlet</url-pattern>
   </servlet-mapping>

<servlet>
      <servlet-name>image2</servlet-name>
      <servlet-class>ImageServlet</servlet-class>
   </servlet>


   <servlet-mapping>
      <servlet-name>image2</servlet-name>
      <url-pattern>/image2</url-pattern>
   </servlet-mapping>
</web-app>
```

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF REFERENCES

[1] R. Rupnow, S. Perchalski, X. Yun, D. Greaves, and H. Glick, .“New Calibration Standings for Next Generation Ship’s Monitoring Systems,” presented at the Thirteenth International Ship’s Control Systems Symposium (SCSS). Orlando, FL, 2003.

[2] C. Vern, “Sea Power 21, Projecting Decisive Joint Capabilities,” in Naval Institute Proceedings, 2002, pp. 1-5.

[3] S. Perchalski, “Shipboard sensor closed-loop calibration using wireless LANS and DataSocket transport protocols,” *Washington Post*, 7 May, 2002.

[4] E. Silva, “Network-Based Control, Monitoring, and Calibration of Shipboard Sensors,” Master’s Thesis, Naval Postgraduate School, Monterey, California, September 2003.

[5] Y. Noguchi, Rockville Firm Hoping to Help the Navy Go Wireless, *Washington Post*, 7 May, 2002.

[6] 3e Technologies International, *3e-550I W-LION Industrial Wireless IO Node*, 3e Technologies International, 2002.

[7] 3e Technologies International, *3e-521N Series Wireless Dual Mode Gateway User’s Guide,* 3e Technologies International, 2002.

[8] CRYSTAL Engineering Corporation, PN: 2975 Rev A, *XP*$_2$ *Digital Test Gauge Operational Manual*, CRYSTAL Engineering Corporation, 2003.

[9] Honeywell, *Precision Pressure Transducer PPT and PPTR User’s Manual Version 2.4*, Honeywell, Inc., 2000.

[10] OMEGA, *PX202,PX203,PX205,PX212,PX213,PX215 Pressure Transducers M2165/0395*, Omega, 1995.

[11] Network I/O, NIOengine1ie, User’s Manual, Network I/O, Inc., 2005

[12] Linksys, Dual-*Band Wireless A+G Broadband Router*, Linksys, 2003

[13] B&B Electronics Manufacturing Company, *Multi-Interface Ethernet Serial Servers*, B&B Electronics Manufacturing Company, 2004

[14] D. Wallis, “Vibration Analysis via Wireless Network,” Master’s Thesis, Naval Postgraduate School, Monterey, California, September 2007.

THIS PAGE INTENTIONALLY LEFT BLANK

INITIAL DISTRIBUTION LIST

1. Defense Technical Information Center
   Ft. Belvoir, VA

2. Dudley Knox Library
   Naval Postgraduate School
   Monterey, CA

3. Chairman, Code EC
   Department of Electrical and Computer Engineering
   Naval Postgraduate School
   Monterey, CA

4. Professor Xiaoping Yun, Code EC/YX
   Department of Electrical and Computer Engineering
   Naval Postgraduate School
   Monterey, CA

5. Professor Roberto Cristi, Code EC
   Department of Electrical Engineering
   Naval Postgraduate School
   Monterey, CA

6. Professor Don Wadsworth, Code EC
   Department of Electrical Engineering
   Naval Postgraduate School
   Monterey, CA

7. LCDR Ken Macklin, Code EC
   Department of Electrical Engineering
   Naval Postgraduate School
   Monterey, CA

8. James Calusdian, Code EC
   Department of Electrical Engineering
   Naval Postgraduate School
   Monterey, CA