SOFTWARE FOR MANIPULATING AND EMBEDDING DATA INTERROGATION ALGORITHMS INTO INTEGRATED SYSTEMS

Special Application to Structural Health Monitoring

David W. Allen

Thesis submitted to the faculty of the Virginia Polytechnic Institute and State University in partial fulfillment of the requirements for the degree of

Masters of Sciences

In

Mechanical Engineering

Dr. Daniel J. Inman

Dr. Henry Robertshaw

Dr. Gyuhae Park

August 31, 2004

Blacksburg, Virginia

Keywords: Structural Health Monitoring, JAVA Software, Embedded Systems, Integrated Systems, Data Interrogation, Process Development, Time Series Modeling, Extreme Value Statistics, and Statistical Discrimination

SOFTWARE FOR MANIPULATING AND EMBEDDING DATA INTERROGATION ALGORITHMS INTO INTEGRATED SYSTEMS

Special Application to Structural Health Monitoring

David W. Allen

Abstract

In this study a software package for easily creating and embedding structural health monitoring (SHM) data interrogation processes in remote hardware is presented. The software described herein is comprised of two pieces. The first is a client to allow graphical construction of data interrogation processes. The second is node software for remote execution of processes on remote sensing and monitoring hardware. The client software is created around a catalog of data interrogation algorithms compiled over several years of research at Los Alamos National Laboratory known as DIAMOND II. This study also includes encapsulating the DIAMOND II algorithms into independent interchangeable functions and expanding the catalog with work in feature extraction and statistical discrimination. The client software also includes methods for interfacing with the node software over an Internet connection. Once connected, the client software can upload a developed process to the integrated sensing and processing node. The node software has the ability to run the processes and return results. This software creates a distributed SHM network without individual nodes relying on each other or a centralized server to monitor a structure.

For the demonstration summarized in this study, the client software is used to create data collection, feature extraction, and statistical modeling processes. Data are collected from monitoring hardware connected to the client by a local area network. A structural health monitoring process is created on the client and uploaded to the node software residing on the monitoring hardware. The node software runs the process and monitors a test structure for induced damage, returning the current structural-state indicator in near real time to the client.

Current integrated health monitoring systems rely on processes statically loaded onto the monitoring node before the node is deployed in the field. The primary new contribution of this study is a software paradigm that allows processes to be created remotely and uploaded to the node in a dynamic fashion over the life of the monitoring node without taking the node out of service.

Table of Contents

List of Figures

List of Tables

1 Introduction

Structural Health Monitoring (SHM) is the process of observing a structure over time, identifying a damage sensitive feature in the observations and performing a statistical analysis of these features to determine the health of the observed structure. Structures include examples from mechanical, civil and aerospace engineering systems. The observations consist of periodically sampled dynamic response measurements from an array of sensors deployed on the structure. Damage sensitive features are extracted through modeling of baseline structural observations, while the subsequent statistical analysis leads to quantitative information about the current state of the observed structure. SHM can be used in one of two modes. In the first mode a SHM system periodically checks the health of a structure and assesses the ability of the structure to perform an intended function as the inevitable effects of aging and degradation set in. In a second mode of operation, a SHM system is used for rapid assessment of structural state after an extreme event, such as an earthquake, to provide near real time information about structural integrity.

People depend upon a vast amount of aerospace, civil and mechanical infrastructure of which little is known about the structural state. In the civil engineering sector there are over 600,000 highway bridges (DOT, 2003) in the United States all in varying structural states. There are buildings that can undergo seismic loading that can potentially damage moment resisting steel joints. In the aerospace industry airplanes undergo maintenance on a time-based schedule. Composites, which are hard to diagnose visually, are becoming the building material of choice for aerospace and some civil structures. In each of these areas monitoring the health of the system in a more analytical way and on a more regular basis could lead to improved life-safety benefits and economic savings. The problem of SHM is not "one size fits all". Each structure, component, geometry, or material must often be approached differently.

The field of SHM covers a diverse group of technologies that must be integrated to monitor the structural integrity of bridges, buildings, aerospace, and mechanical structures. Four basic parts of the SHM process are defined in the Los Alamos National Laboratory (LANL) *statistical pattern recognition paradigm* (Farrar, 2000): 1. Operational evaluation, 2. Data acquisition, cleansing and normalization, 3. Feature extraction, and 4. Statistical feature discrimination. The enormity of the SHM problem is more manageable when broken into these four parts. Many research efforts have focused on finding new damage sensitive features in data and developing feature extraction techniques (Doebling, 1998). Another large area of research is in sensor development, which falls under the data acquisition portion of the paradigm. Initial work in data normalization and statistical feature discrimination has begun with forays into neural networks and statistical hypothesis testing, respectfully. There are as many feature extraction techniques, as there are researchers; a plethora of sensors are designed for various materials, environments and structures; and various statistical discrimination techniques exist, each with distinct advantages for specific applications.

1.1 Motivation

As the SHM field grows and matures, the question of "can a structure's health be monitored?" is being replaced by "which permutation of sensing technology, cleansing, normalization, feature extraction, and statistical discrimination yields the best results for the problem outlined in the operational evaluation of a structure?" Next, one must ask how a process can be deployed in a testing scenario on a "real-world" structure. And finally, a solution must be found for how results from this process can be easily interpreted and transmitted to a central structural analysis center. These questions require a set of modular tools for cataloging functions, rapid assembly of SHM processes, embedding processes in remote monitoring hardware, and finally for receiving information about the monitored structure.

This thesis is an introduction to the set of modular software developed by LANL and Virginia Tech to interface with an integrated hardware system assembled by Motorola. This software and hardware fills the void by bringing SHM technologies together in a single package. The integrated set of tools, known as the Husky project[1], is robust and adaptable to varying structural and environmental requirements.

1.2 A Review of Selected literature

There are several sources on creating SHM systems using wired and wireless technology. A SHM system is defined in this document as an integrated package of data interrogation software and hardware with both sensing and processing capabilities. Many systems therefore are not included here because the system focuses are primarily sensing and streaming raw data back to a central sever for analysis. In an effort to cull the papers to a meaningful set, only integrated systems specifically designed for SHM in which data are collected, processed, and a result returned are reviewed.

1.2.1 Wired transmission

Todoroki, (1999) introduces a SHM system called the *Plug & Monitor* system on the Japanese bullet trains. This system incorporates vibration, temperature, and speed sensors as well as cameras under the railroad carriages to confirm the condition of the tracks. Each sub-network on the train features its own CPU to process data locally and minimize the amount of data transmitted, thereby minimizing chances of data collision. Data collision is a problem that results from a network device receiving simultaneous requests to store or retrieve data from another device on the network. As networks expand, the chances of data collision increase. The *Plug & Monitor* operates in an unsupervised learning mode, meaning no *a priori* knowledge of damage is required, with the ability to perform damage detection by simply plugging sensors into the system. The embedded software automatically constructs simple correlations between distributed sensors and finds deviations from the normal measurements. Web-based

[1] The Husky as a breed of dogs has been breed to be small, fast, and yet a work horse in conditions where other animals could not survive. Though the Husky developed as a mode of transportation in colder northern regions, it is equally as happy in warmer climates where its dual layered coat keeps it cool in summer months. This combination of tenacity to perform its job without fail, strength and speed in a small package, and adaptability to the environment are the reason for naming this system the Husky project.

cameras are used to confirm damage. The authors, however, do not specifically state the algorithm or software details of the *Plug & Monitor* system and only state that the software is written with JAVA for Web portability.

Moster (2004) from Datatek Applications, Inc. describes the process of detecting gear faults using frequency domain techniques and support vector machines (SVM) (Haykin, 1998). A SVM is an algorithm that projects data into higher dimensions where it becomes more separable. Gears with simulated cracks are placed into gearboxes and acceleration time history is collected as the gearbox is run. The gears with faults show a periodic spike in the time history that corresponds to one revolution of the gear. Instead of using raw time history as input to the SVM to extract the features, Moster uses the frequency domain analysis to extract features because the periodic spike provides a less ambiguous input. Using spectral correlation as the feature, the SVM algorithm is able to correctly classify the undamaged and damaged motors when the SVM is operated in a supervised learning mode. In a supervised learning mode, examples of damage are included in training. In an unsupervised learning mode, the accuracy of the fault detection was lower, with false positive damage indications in the results. The article is interesting because this advanced data interrogation process has been integrated with data acquisition hardware. Datatek also provides a GUI interface, and visual display of damage states in the DT-3000 autonomous vibration monitoring system.

The *Plug & Monitor* system is included because software is bundled with the hardware. Software allows processing at a local level and the results transmitted to a central monitoring location via the web for review. This approach is similar to the intent of the Husky project; however the *Plug & Monitor* software is static, it does not include a front end for dynamically creating and loading new SHM processes. The SHM process incorporated is also based on simple correlation of data between sensors. The SHM processes that can be developed with the Husky project are much more complex in terms of feature extraction, data normalization, and statistical modeling for feature discrimination. The Husky hardware is also small, portable, and can be wireless.

Similar to the previous article, data collection and processing by the Datatek system are performed by a central processing unit and Ethernet transmits results to a monitoring station. The GUI is for setup and configuration, but the SHM process is still statically embedded in the hardware. The large difference, therefore, between the Husky project and the DT-3000 is the ability to dynamically change the data interrogation process used to detect damage.

1.2.2 Wireless transmission

Stepping away from wired transmission to wireless shows a larger emphasis on processing data at the collection point. This emphasis is because a design goal of wireless systems is typically to run on battery power. The power required to process data is considerably lower than the power required to transmit data. Therefore these systems typically perform some form of data compression locally, and only transmit a limited amount of processed information.

Tanner (2001) used the Crossbow "Mote platform" hardware to implement a simple SHM system. The Motes are a small, integrated sensor, processor, and RF transmitter originally developed at UC Berkley. The processor proved to be very limited, allowing only the most rudimentary data interrogation algorithms to be implemented.

Tanner used a process that monitored the autocorrelation between signals using user-defined thresholds. A binary result could then either be shown on the mote's LEDs or transmitted wirelessly to a base station. The system was demonstrated using a small portal structure with damage induced by loss of pre-load in a bolted joint.

Lynch (2002) presented hardware for a wireless peer-to-peer (P2P) SHM system. Using off the shelf components, Lynch couples sensing circuits and wireless transmission with a computational core allowing a decentralized data collection, analysis, and broadcast of a structural health indicator. The final hardware platform includes two microcontrollers for data collection and computation connected to a spread spectrum wireless modem. The software is tightly integrated with the hardware and includes the wireless transmission module, the sensing module, and application module. The application module implements the AR-ARX time series algorithm described in Sohn, 2001. This integrated data interrogation process requires communication with a centralized sever to retrieve model coefficients. The object of closely integrating the hardware and software with the dual microcontrollers is to produce a power efficient design.

Spencer (2004) in a review of current "smart sensing" technologies compiled summaries of wireless work in the SHM field using small, integrated sensor and processor systems. A smart sensor, as defined by Spencer, is a sensor with an embedded microprocessor and wireless communication. Many smart sensors covered in this article simply sense and transmit data. However, efforts using the Mote platform are discussed and a new generation of Mote is outlined. Two drawbacks of the Mote platform as noted by Tanner were a lack of programming space and processing power. These drawbacks severely limited the ability to program complex data interrogation processes. The new Mote prototype by Intel ® provides a more powerful processor, but does not deliver more programming space.

A comparison, shown in Table 1, illustrates the differences in the various wireless integrated SHM systems found in the previously discussed articles.

The inadequacy of the Mote processing capabilities is clearly illustrated when compared to the Stanford WiMMS system and the Husky project. Both the Stanford system and the Husky project have the ability to perform complex data interrogation processes at each node.

Beyond the processing limitations of the Mote system, implementing new processes on the system requires knowledge of the C programming language and the Tiny © operating system (OS). Both C and Tiny © OS require a steep learning curve. In contrast, the Husky project provides an easy to learn GUI client for creating and loading new processes, and a flexible hardware platform with true computing and sensing power. The physical footprint of the mote system is much smaller than the Husky hardware, but the added software and hardware functionality out weighs size penalty for this application. The mote platform seems to be far too limited to perform true SHM for most applications.

The Stanford WiMMS system provides hardware that is adequate for sampling sensor data from a structure and providing computing power for complex algorithms. There are some differences, however, between the Stanford system and the Husky project. The software included in the Husky system provides a dynamic development platform for creating new SHM processes. The Stanford WiMMS system relies on a process that is loaded

statically and is tightly coupled with a centralized server. Overall, the two platforms differ in that Lynch creates a tightly coupled and optimized system targeted for civil applications. The Husky system is a development and test platform that is configurable in both software and hardware to find optimal solutions for a variety of different SHM problems. While Lynch's integrated hardware is similar to the Husky system, the philosophy of Lynch's system is quite different. The system developed by Lynch is tightly integrated for optimal power consumption purposes. The Husky project is loosely coupled for maximum flexibility and assumes the availability of a continuous power source.

Table 1 - Comparison of wireless systems used for structural health monitoring

	Stanford WiMMS[2]	Crossbow MICA2[3]	Intel Mote[4]	Husky Project
Processing/Programming				
Program Processor	Motorola: 40 MHz 32 bit	Amtel: 16 MHz 8 bit	Intel: 12 MHz 32 bit	Intel: Pentium 233 MHz
Programming Memory	448 Kb	128 Kb	64 Kb	512 Mb[5]
Sampling Processor	Amtel: 4 MHz 8 bit	N/A	N/A	Motorola: 120 MHz 16 bit
Sample Memory	8 Kb	512 Kb	512 Kb	36 Kb
OS	Embedded	Tiny OS	Tiny OS	Linux
Sensing				
ADC	16 bit	10 bit	10 bit	16 bit
Max Sample Freq	1 kHz	*Not Available*	*Not Available*	200 kHz
Sensor Types	Analog	Analog, Digital	Analog, Digital	Analog
Wireless Transmission				
Indoor Range	500 ft.	100 ft.	100 ft.	30 ft.
IEEE MAC[6] Protocol	802.3 (Ethernet)	Software[7]	Software[7]	802.15.4 (ZigBee)
Frequency	2.4-2.483 GHz	433, 868/916 MHz	2.4 GHz	2.4 GHz
Data Rate	1.6 Mbps	38.4 Kbps	57.6 Kbps	230 kbps
Physical				
Volume	20 in^3	0.70 in^3	1.4 in^3	108 in^3
Power[8]	78 mW – 2.4 W	0.5 μW – 50 mW	*Not Available*	6 W
Cost	$500[9]	$150	*Not Available*	$800[9]

[2] Lynch, 2003

[3] www.xbow.com

[4] Spencer, 2004

[5] This memory is in the form of a Compact Flash card. Larger or smaller flash cards can be easily used.

[6] Media Access Control - The interface between a node's logic layer and the network's physical layer.

[7] TinyOS implements the Media Access Control Protocol in software allowing more flexibility, but placing a higher burden on the processor.

[8] The first power rating refers to the system in *sleep* mode while the second represents the system in *active*. If only one number is shown, this represents *active*.

[9] This figure represents an approximate cost of the research prototype materials.

1.3 Scope and overview

The scope of this thesis encompasses the creation of novel software that presents the first method for dynamically creating, implementing and updating a SHM process that is couples with sensing, processing, and telemetry hardware. Such a software tool needs to fulfill four basic steps:

- Present the user with a toolbox of functions including feature extraction, data normalization, and statistical modeling for SHM.
- Allow the user to create a working process from this toolbox of functions.
- Load and run the process on an integrated sensing and processing hardware platform.
- Train the process in an effort to establish appropriate thresholds for the damage indicator.
- Dynamically update the process as new data become available.
- Transmit and receive information pertaining to the state of the structure.

It should be noted that the software and hardware platforms developed in this project have been designed to be flexible and are not simply restricted to SHM applications. SHM provides an interesting problem where a solution was lacking and provides an excellent test bed for the technology.

The thesis layout follows the flow in which the system is used with background on development and implementation at each step. First an overview of the time series and statistical methods employed to create a near real-time SHM process is presented. Then the first look at the client software for prototyping these SHM processes are presented. Finally, the integrated software and hardware for interrogating a structure is discussed. A small proof of concept experimental section is included in which a SHM process is developed, loaded onto the integrated hardware, then monitors a small structure, and a damage classification is returned.

1.4 Contributions

The unique contributions of the research and development work summarized herein are:

Data interrogation:

- The automation of model order selection for time series models for use in the feature extraction process.
- Development of a sum of squared error damage feature.

Software development:

- Cataloging and encapsulation of SHM functions developed by researchers at LANL and Virginia Tech.
- Development of a graphical user interface for assembling functions from varying development languages into processes and running the processes locally.
- Development of a server application for remote execution of assembled processes and transmission of process results using standard Internet protocols.
- Development of a framework for loosely coupled hardware and software through TCP/IP communication to allow for modular co-development of the hardware and software systems.

2 Statistical Pattern Recognition for Structural Health Monitoring

2.1 Introduction

Statistical pattern recognition is a topic of research in engineering, statistics, computer science and even the social sciences. The topic covers the design of systems that recognize patterns in data. In the area of Structural health monitoring (SHM) this process typically involves recognizing patterns that differentiate data collected from a structure in its undamaged and damaged states. Statistical pattern recognition appeals to researchers in the area of SHM because of the ability to quantify and automate the decision making process in the presence of uncertainty. This ability leads naturally to the design of integrated hardware and software systems that can continuously monitor a structure's health.

There are four basic steps to the SHM statistical pattern recognition paradigm: operational evaluation, data cleansing and normalization, feature extraction, and feature analysis (Farrar, 2000).

Operational evaluation is the process of defining the damage that is to be detected and providing justification, typically either economic or safety related, for performing the SHM. The operational evaluation portion of the statistical pattern recognition paradigm is not covered in this chapter because of the dependence upon the sensing hardware and the structure tested. Operational evaluation will be briefly covered in the hardware and experimental chapters.

Data cleansing, normalization, compression and fusion can all be part of the last three steps in this process. During data collection, a sensor array measures the system response such as an acceleration time history at points on the structure. These data are then cleansed (e.g. filtered) and normalized (e.g. subtract the mean and divide by the standard deviation) before features are extracted from the data. Data cleansing and normalization are performed in an effort to separate operational and environmental variability from system response changes caused by damage. For example, a building's air handling unit may produce unwanted acoustic signals at a know frequency that can be detected by the SHM sensing system and mistaken for the onset of damage. Data cleansing filters are then applied to remove these acoustic signals in an effort to reduce false indications of damage.

Next, a feature that is sensitive to the onset of damage in the system is extracted from the data. If, for example, the structure typically behaves in a linear fashion, then a feature that indicates the transition to nonlinear behavior may be extracted and used as an indication of damage.

Feature analysis is performed to identify when changes in the measured response are statistically significant and indicate damage as opposed to changing operational and environmental conditions. Almost all of these statistical methods employed for SHM provide an indication of damage when the damage feature exceeds a statistically determined threshold. To establish the statistical thresholds, the process must go through training on baseline data collected from the structure in an undamaged state.

This chapter describes the various data cleansing, data normalization, feature extraction, and statistical discrimination techniques that are included in the software developed as part of this study.

2.2 Data cleansing and normalization

Know effects that alter the data must be removed to allow accurate modeling of the underlying system response. These know effects can be environmental or operational such as thermal changes throughout the day or a DC offset in the measuring equipment. Such patterns are not of interest in the SHM problem because they are not related to damage in the structure. Often one or more data cleansing and normalization techniques are applied to the data.

Data that are collected from natural environments can often display exponential growth, seasonal drift, or cyclic patterns. In some cases only certain frequency ranges may be of interest, or the data may need to be differenced to remove polynomial trends. These techniques all fall under data cleansing, the process of eliminating unwanted data. Possible influences that would require data cleansing include temperature drift, known inputs from machinery, or other environmental influences.

Data that display a shifting of the mean from zero, such as DC offset, or a scalar change in amplitude from one data set to the next can be normalized. Data normalization is the process of scaling the data to facilitate a more direct comparison of different data sets. By subtracting off the mean of the entire data set and dividing by the standard deviation, all of the data sets will be re-scaled to zero mean and common amplitude. Data may also display a logarithmic increase in amplitude that can be removed by a log transformation. More complicated changes in the data may require more sophisticated normalization techniques such as neural networks (Sohn, 2003).

The following example demonstrates the data cleansing and normalization process. The original data are numerically generated from a Gaussian white noise process (Figure 1). The data shown in Figure 2 are the original data with simulated environmental trends, a simple linear trend, added.

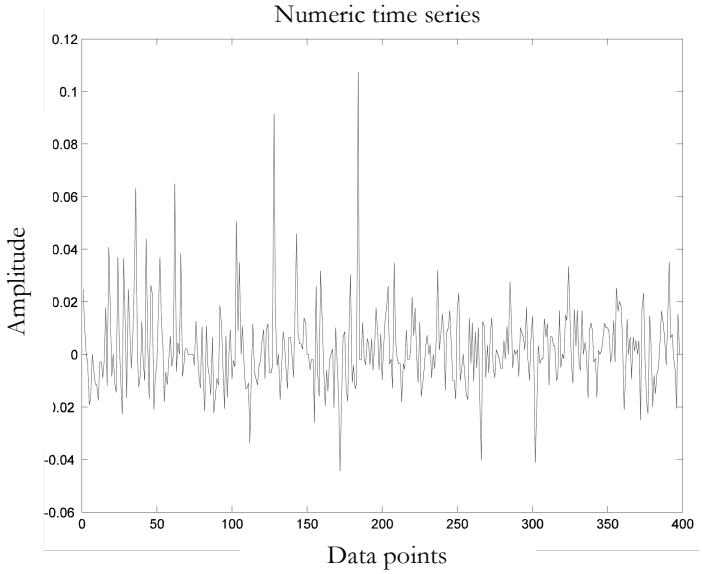

Figure 1 - A numeric Gaussian white noise signal demonstrating a random signal that could be modeled using time series analysis.

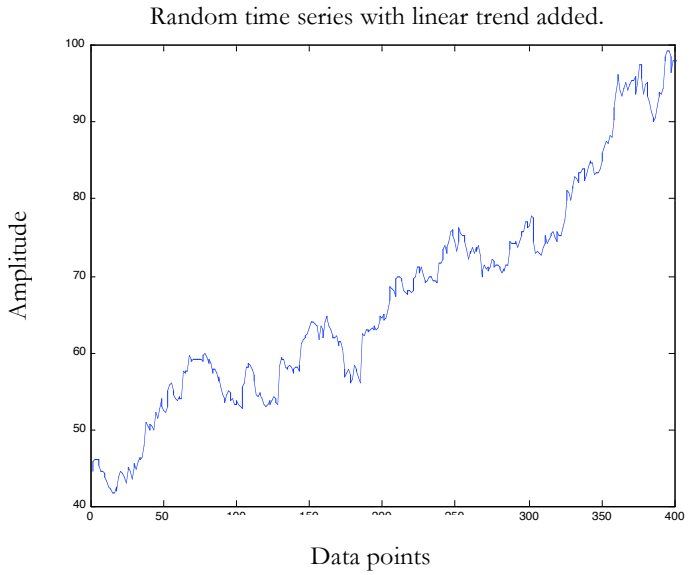

Figure 2 – Original data with simulated environmental trends and operational bias added.

The linear trend is removed by first order differencing the data. Differencing has the same effect as high-pass filtering a linear polynomial from the data and is defined as:

$$y_i = x_i - x_{i+1} \qquad i=1...n-1 \tag{1}$$

where y_i are the transformed data and x_i are the original data. Note that in this transformation a degree of freedom is lost. Higher order differencing can be used to remove higher order polynomial trends. Such a polynomial trend could be introduced into the data by an increasing temperature, humidity or other environmental factor. Figure 3 shows the results of differencing the original data.

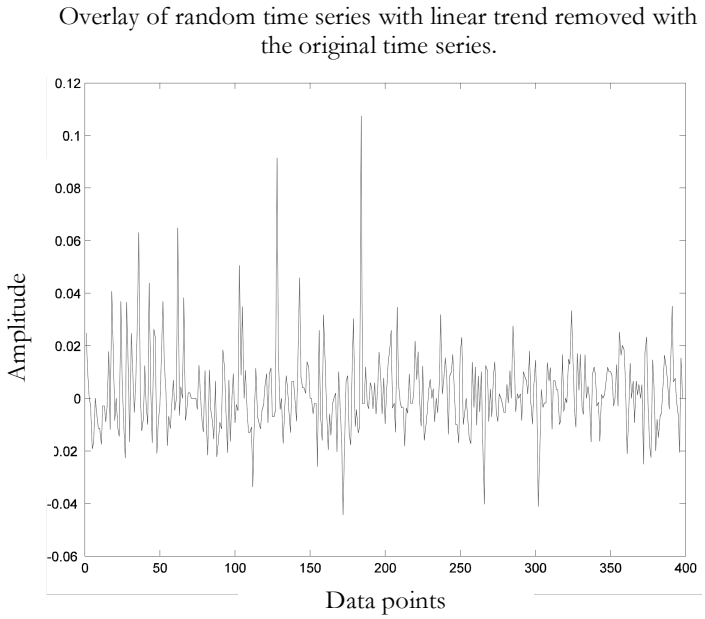

Figure 3 - Difference of the original data to remove linear trend. The resulting data are stationary and are the result of a Gaussian white noise process. Because of the numerical nature of the example, the original and resulting time series are identical.

After the data cleansing process, the data conform to the definition of a stationary process as is necessary for the next portion of analysis. A tool to test if the signal is conditioned well is the periodogram as shown in Figure 4. The periodogram is a measure of the intensity, $I(f_i)$, at frequency f_i as defined by (Box, 1994):

$$I(f_i) = \frac{N}{2}\left(a_i^2 + b_i^2\right) \quad i = 1,2,...,(N-1)/2 \tag{2}$$

where N is the number of observations and a_i and b_i are the coefficients from fitting a Fourier series model to the data. The signal can be considered stationary white noise if the intensity is spread over all of the frequency components. If one or more frequencies dominate the periodogram, then an underlying process may exist.

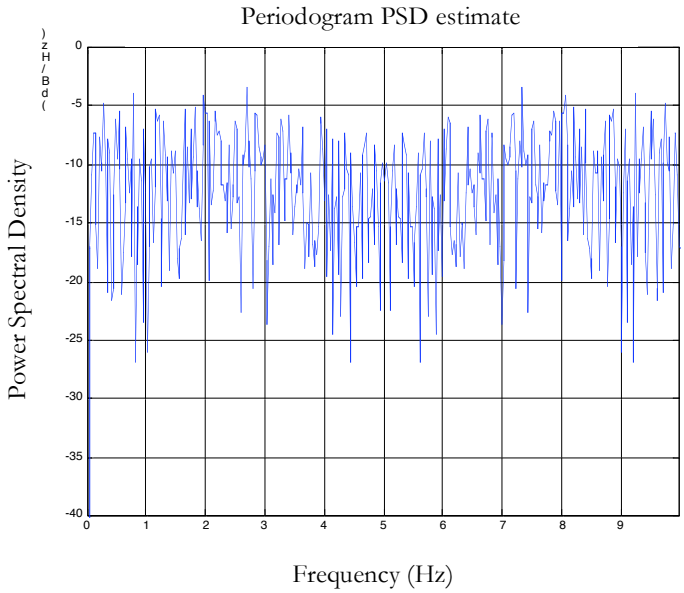

Figure 4 - Periodogram of the cleansed data. Notice that the distribution of the frequency content is spread evenly over the entire range. This spreading is an indication that the data are well-formed, stationary white noise, and that there is no underlying process.

2.3 Feature extraction

Once the environmental and operational effects are removed from the data, models can be applied to identify damage sensitive features. The feature extraction technique used herein is to fit a linear predictive model to data from the undamaged structure and calculate a residual error between the measured data and the data generated by the model. When damage is introduced to the system, it is assumed to exhibit a modified linear or non-linear response. The linear model built from the undamaged state is used to predict data from the damaged system, but the modified linear or non-linear nature of the response will cause an increase in residual error. Therefore, the residual error is the damage-sensitive feature derived from the measured system response data. The particular models used in the feature extraction process are described below.

2.3.1 Time series models

Often data are collected from a structure over time such as $\{x_1, x_2, \ldots, x_n\}$. These data then are referred to as a *time series* and can be modeled using models that assume correlation between sequential points in the time series. Time series models were developed to forecast time series such as temperature or species populations (Box, 1994), but also have many applications in engineering and process control. The motivation for using time domain models in the SHM application is to compress large amounts of data and in the process to extract damage sensitive features from these data. Time domain models assume that a current or future point in the time series is a linear combination of n preceding time points. There are two basic types of time series models: Auto-Regressive (AR) and Moving Average (MA) models. There are also several combinations of these two models that may need to be employed to fit data properly. The AR model is of the form:

$$\tilde{y}_{(t)} = \sum_{n=1}^{p} \phi_n y_{(t-n)} + \varepsilon_{AR} \tag{3}$$

With an AR model the estimate of current point, $\tilde{y}_{(t)}$, is a linear combination of p previous system response terms, $y_{(t-n)}$, weighted by the linear coefficients ϕ_n and the addition of an error term, ε_{AR}. Equation (3) denotes an AR(p) model, where p defines the order, or number of previous time points that are assumed to be strongly correlated with the current time point in the model. A MA model again states that the current point is a linear combination of previous terms, however, instead of previous system responses a MA model uses previous errors:

$$\tilde{y}_{(t)} = \left(1 + \sum_{n=1}^{q} \theta_{t-n}\right)\varepsilon_t \tag{4}$$

$\tilde{y}_{(t)}$, the current point, this time it is a combination of the previous q random errors, ε_n, weighted by linear coefficients θ. This model is then denoted as an MA(q) model.

A combination of the two processes may be present in a time series calling for a mixed model, or Auto-Regressive Moving-Average model denoted ARMA(p,q):

$$\tilde{y}_{(t)} = \sum_{n=1}^{p} \phi_n y_{(t-n)} + \left(1 + \sum_{n=1}^{q} \theta_{t-n}\right)\varepsilon_t \tag{5}$$

The underlying assumptions for these models are that the data are time series ($y_{(t)}$ *and* $\varepsilon_{(t)}$) that are Gaussian and random in nature. The data must therefore be stationary in mean and variance and have any trends, cycles, or bias removed, as previously described in data normalization and cleansing discussion. Once the data are "well behaved" a more accurate model can be estimated.

2.3.2 Automation of model order selection

Time series models are functions of the model order. Using too few coefficients may lead to not capturing the underlying mechanism of the data. If too many coefficients are used, the noise in the data begins to be modeled which degrades the ability of the model to generalize to other data sets. Interestingly, in theory an AR(p) process can also be expressed as a MA(∞) model and vice versa. Often this transformation manifests in an AR process being mistakenly modeled by a large ordered MA process or conversely an MA process being modeled as a large ordered AR process.

Because AR and MA time series models are based on the principal that a current point is a linear combination of p or q past observations, it is natural to look at correlation functions to decide how a current data point is related to n previous points and where this correlation may break down. Two such functions are the autocorrelation function (ACF) and the partial autocorrelation function (PACF). The ACF shows the correlation of a two points separated by a fixed lag n:

$$\rho_n = \frac{E\left[\left(x_t - \mu_x\right)\left(x_{t+n} - \mu_x\right)\right]}{\sigma_x^2} = \frac{\gamma_n}{\gamma_0} \tag{6}$$

where the ACF, ρ_n, is a function of the expected value, $E()$, of the time series x_t, normalized by μ_x and σ_x^2, the mean and variance respectively of x_t. γ_n is the covariance at lag n. Not that the ACF at n=0 should always be 1. The ACF of a MA process will theoretically diminish after lag n (Box, 1994). Therefore, if the ACF approaches zero after n lags, the data are represented with a MA(n) model [Equation (4)]. An example of the ACF for data generated by the following MA(1) model:

$$\tilde{y}_{(t)} = \varepsilon_t - 1.05\varepsilon_{t-1} \tag{7}$$

is found in Figure 5.

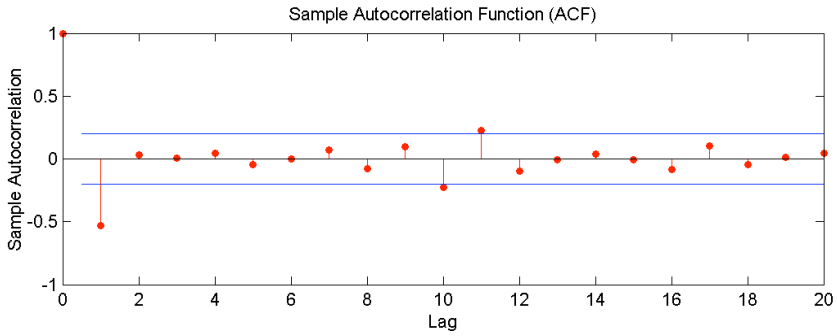

Figure 5 - Autocorrelation function for a moving-average processor order 1. By looking at the autocorrelation function, it can be seen that the process is of order one because of a correlation at lag n=1 and insignificant correlation at all other points. The thresholds represent a 95% confidence interval that a correlation is insignificant.

The Partial autocorrelation function (PACF) quantifies the correlation of a time series point and a point at lag n, with all other correlations removed. The PACF at a time point lagged by n, ϕ_{nn}, can be found recursively through the Yule-Walker equations (Box, 1994):

$$\begin{bmatrix} 1 & \rho_1 & \rho_2 & \cdots & \rho_{n-1} \\ \rho_1 & 1 & \rho_1 & \cdots & \rho_{n-2} \\ \vdots & \vdots & \vdots & \vdots\vdots\vdots & \vdots \\ \rho_{n-1} & \rho_{n-2} & \rho_{n-3} & \cdots & 1 \end{bmatrix} \begin{bmatrix} \phi_{n1} \\ \phi_{n2} \\ \vdots \\ \phi_{nn} \end{bmatrix} = \begin{bmatrix} \rho_1 \\ \rho_2 \\ \vdots \\ \rho_n \end{bmatrix} \tag{8}$$

In the case of an AR process [Equation (3)] the PACF theoretically goes to zero after n lags. Therefore, if the PACF is significantly diminished after n lags, the appropriate order of model would be p=n. In practice, the ACF and PACF only approach zero and some judgment is needed to choose an appropriate model order. An example of the PACF of an AR(2) process can be found in Figure 6

The ACF decaying in an exponential or sinusoidal fashion is an indication of a AR process and the PACF should be checked for confirmation. Alternatively, a MA model will decay exponentially or in a sinusoidal fashion in the PACF. If the ACF and PACF both diminish in exponential or sinusoidal fashion, then the underlying process may be a mixed model, or ARMA(*p,q*).

On the other hand, if neither the PACF nor ACF die out quickly; this indicates that some process exists in the data that violates the underlying assumptions of the time series models. In this case, further data cleansing or a different modeling approach is required before an accurate model can be fit.

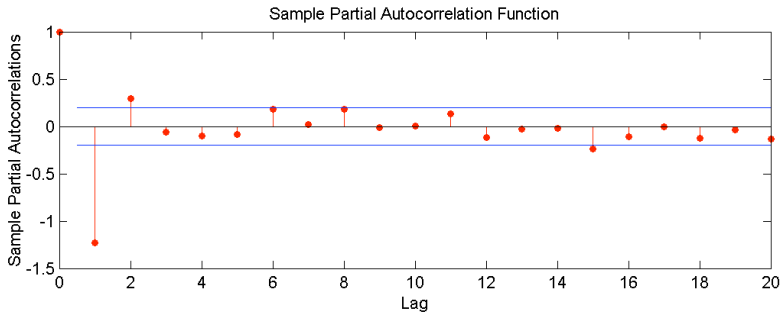

Figure 6 – Partial autocorrelation function for an auto-regressive process. By looking at the partial autocorrelation function, it can be seen that the process is of order 2 because of a correlation at lags n=1 and 2. Thresholds are based on 95% confidence interval that the correlations are insignificant.

To determine if an appropriate model and order has been chosen, the residuals from the model prediction the measured time series data are examined. The residuals should be uncorrelated if the model has performed a reasonable prediction (Box, 1994). A simple ACF and PACF plot of the residuals from the AR(2) model estimation (shown in Figure 7) can be used to determine if the residuals are correlated. As is seen in Figure 7, the residual errors from fitting the AR(2) model are not correlated.

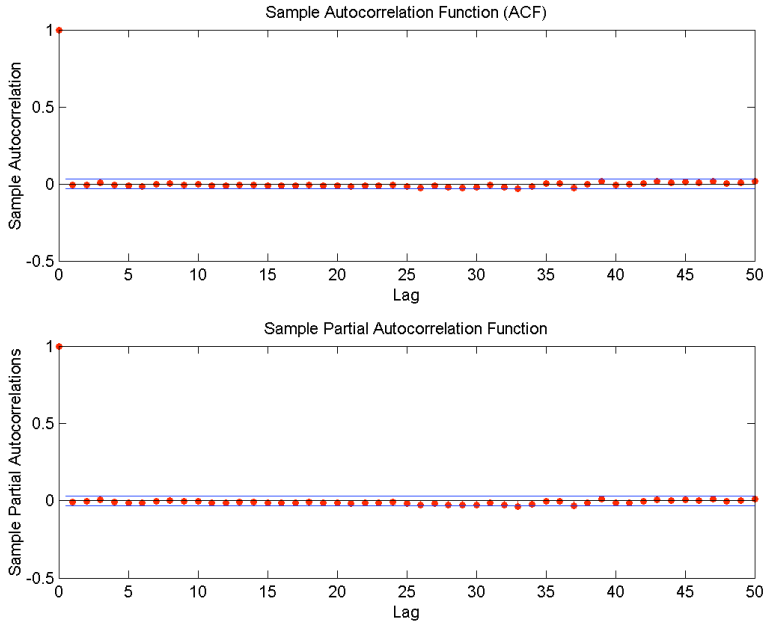

Figure 7 – ACF and PACF of the residuals from the model forecast. There is no indication of any correlation because all points lie within a 99% confidence interval. Note that at Lag 0, a point will always be perfectly correlated with it's self.

Once the type model is chosen for the data, there are diagnostics that can help determine if the model order is appropriate. Two basic quantities are tested: the residual error and a criterion number. A criterion number is a quantitative measurement of how appropriate a model is for a given set of data. Several different criterions exist such as Akaike's Information Criterion (AIC), Final Prediction Error (FPE) and Schwartz-Bayesian Criterion (SBC), all of which take into account a tradeoff between the prediction errors for a model and penalize for the too many fit parameters (Akaike, 1974 and Schwarz, 1978).

The AIC number is a measure of how well the model predicts the data, adjusted by a penalization factor for the number of terms in the model weighted by the number of data points available to fit:

$$AIC = \log(v) + \frac{2d}{N} \tag{9}$$

where v is the maximum likelihood estimate of the prediction error (Box, 1994) for the model, d is the number of coefficients fit, and N is the number of data points fit. For example, a model of order 6 that fits the data well could have a higher AIC than a model of order 4 that fits the data almost as well because of the penalty for extra terms that is associated with this criterion. It is important to note that AIC values are only comparable for different models identified from the same data set. AIC values from different sets of data are not comparable.

Similar to the AIC, the FPE and SBC are also functions of the prediction error, the size of the data set fit and a penalization for the number of fit coefficients. The two other criteria compared are defined as follows:

$$FPE = \frac{\nu(N+d)}{(N-d)} \tag{10}$$

$$SBC = N\log(\nu)+d\log(N) \tag{11}$$

With the criterion defined, the problem of selecting an appropriate model order becomes an optimization problem. An algorithm has been developed to iteratively estimate optimized coefficients for a given model, model order, and data set (Allen, 2003). The criterion numbers are then calculated and the model order that minimizes the prediction error and number of model coefficients is selected.

The method for determining an appropriate model order is computationally intensive. Models are fit, residual errors analyzed for outliers, and criterion numbers are calculated for a range of model orders. Models that pass the residual error test are selected and optimized for the lowest criterion number. The lowest criterion number represents the model that most accurately fits the data with the lowest number of coefficients. This method is easily automated, only requiring a range of model orders to fit.

2.3.3 Damage sensitive feature

Once an appropriate time domain model has been selected, the model coefficients are estimated and the model order optimized, the model can be used to predict the response of the structure. This forecast of the response is then compared to the observed data.

It is known that certain types of damage in a structure cause an initially linear structure to behave in a non-linear manner. Because the time domain models described above are linear and are trained with data obtained when the structure is responding in a linear manner and these training data are assume to be stationary, any non-linear effects will cause errors in the predictive abilities of the model. Therefore, these residual errors can be used as a damage sensitive feature.

A scalar damage sensitive feature is the sum of squares error (SSE). The SSE is a measurement of the difference between predicted data ($f_{i,\,predicted}$) and observed data ($f_{i,\,observed}$) at each measurement point:

$$SSE = \sum_{i=f_0}^{f_n}\left(f_{i,observed} - f_{i,predicted}\right)^2 \tag{12}$$

In the case of data interrogation for structural health monitoring, it is assumed that the response of a healthy structure can be accurately predicted with a properly trained time series model. The $f_{i,observed}$ are observations of the structure in its current (and possibly damaged) state. The $f_{i,predicted}$ is the time domain model prediction of these data based upon a model estimated from the response of the healthy structure. The two responses are subtracted creating a measure of the error between the observed and predicted responses at each time point in the series. These errors are then squared and summed to provide a single positive scalar indication of how different the observed signal is from the expected structural response.

2.4 Statistical modeling for feature discrimination

The task of monitoring changes in the damage sensitive features in a quantitative way leads to the field of Statistical Process Control (SPC) (Montgomery, 1996). SPC has been used for many years in the manufacturing industry to ensure quality control of products. The techniques developed in this field are time tested, geared toward automated decision-making, and can be readily adapted to the SHM problem. SPC operates in an unsupervised learning mode for realistic application of the process. When applied to structural health monitoring, unsupervised learning means that data from the damaged condition are not available to aid in the damage identification process. The objective of unsupervised SPC is to establish a model of the normal system condition and thereafter to signal statistically significant departures from this condition.

First a model must be established from the normal system condition based on the damage-sensitive features extracted from measured system response data. This objective can be accomplished in several ways. Some methods include the use of control charts (Fugate et al., 2001) based on the mean and standard deviation of a damage sensitive feature, or the sequential probability ratio test (Sohn et al., 2002) that monitors a feature based on the ratio of the standard deviation.

SPC with thresholds based on extreme value statistics (EVS) can be used to establish when a significant statistical change in the damage-sensitive feature has occurred. The damaged sensitive feature is collected in a sequential manner, $\{x_1, x_2, \ldots, x_n\}$, and will form a distribution with mean, μ, and variance, σ^2. If the structure is damaged the parameters of the parent distribution, μ and σ^2, are likely to change showing a broadening or shifting of the distribution. Statistical process control provides a framework for monitoring the damage feature and identifying when these types of changes in the feature's distribution occur (Sohn et al., 2003).

2.4.1 Extreme value statistics

When looking at turnkey solutions in SPC, it must be realized that there are underlying assumptions in the statistical models. SPC methods involve establishing thresholds that in the tails of the feature distribution and assume that the damage sensitive feature conforms to a normal distribution. If the feature analyzed is normally distributed, then these SPC methods can be implement "as is", however, if the feature distribution is skewed or is heavy in the tails, then the normal distribution assumption is potentially hazardous. For example, if the feature distribution is heavy tailed, then a decision process based on central statistics (the mean vector and covariance matrix) will set a threshold that results in false-positive indications of damage (i.e. indications of damage when none is present).

A problem with modeling the feature distribution is that often the functional form of the distribution is unknown. There are in fact an infinite number of candidate distributions that may represent the random nature of the feature. The choice of an appropriate distribution is often made based on a knowledgeable data analyst. Parameters of the assumed distribution are then estimated from a set of baseline data. This process is largely subjective and will constrain the tails to that of the assumed distribution. Also, in some cases, the extreme values of an event may be the only data that are recoded because of sensor or storage limitations. Modeling these data as a parent distribution could then bring about erroneous results.

EVS is the portion of statistics that is particularly concerned with modeling the tails of distributions. EVS can be applied to SPC methods to more precisely establish thresholds by accurately modeling the tails of the data. EVS is a branch of order statistics and there are a large number of references in the field. Some are considered classics (Gumbel, 1958; Galambos, 1978), and others are more recent (Embrechts et al., 1997; Kotz and Nadarajah, 2000; Reiss and Thomas, 2001). Castillo (1988) is notable in its concern with engineering problems in fields like meteorology, hydrology, ocean engineering, pollution studies, and strength of materials. Roberts introduced the ideas of EVS into novelty detection in (Roberts, 1998 and 2000) and applied them in the bio-signal processing context. Although EVS has been widely applied in other areas of research, the technique is only now beginning to be applied to SPC and damage identification (Sohn, 2003).

2.4.1.1 Methodology

When analyzing sequential samples $\{X_1, X_2, \ldots, X_n\}$, such as the acceleration response of a structure over time, the resulting time series can have an arbitrary distribution. The most relevant statistics for studying the tails of this parent distribution are the maximum operator, $\max(\{X_1, X_2, \ldots, X_n\})$ and the minimum operator, $\min(\{X_1, X_2, \ldots, X_n\})$, which selects the point of maximum or minimum value from the sample vector. The pivotal theorem of EVS (Fisher, 1928) states that as $n \to \infty$, the induced distribution on the extremes of the samples can only take one of three forms: Gumbel, Weibull, or Frechet.

The fundamental theorem of EVS states (Fisher and Tippett, 1928):

The only three types of non-degenerate distributions *H(x)* satisfying the limits of the maxima and *L(x)* satisfying the limits of the minima are:

Frechet:

$$\text{Maximum} \qquad H_{1,\beta}(x) = \begin{cases} \exp\left[-\left(\dfrac{\delta}{x-\lambda}\right)^{\beta}\right] & \text{if} \quad x \geq \lambda \\ 0 & \text{otherwise} \end{cases} \tag{13}$$

$$\text{Minimum} \qquad L_{1,\beta}(x) = \begin{cases} 1-\exp\left[-\left(\dfrac{\delta}{\lambda-x}\right)^{\beta}\right] & \text{if } x \leq \lambda \\ 1 & \text{otherwise} \end{cases}$$

Weibull:

(14)

Maximum $$H_{2,\beta}(x) = \begin{cases} 1 & \text{if } x \geq \lambda \\ \exp\left[-\left(\frac{\lambda - x}{\delta}\right)^{\beta}\right] & \text{otherwise} \end{cases}$$

Minimum $$L_{2,\beta}(x) = \begin{cases} 0 & x \leq \lambda \\ 1 - \exp\left[-\left(\frac{x - \lambda}{\delta}\right)^{\beta}\right] & x > \lambda \end{cases}$$

Gumbel:

Maximum $$H_{3,0}(x) = \exp\left[-\exp\left(-\frac{x-\lambda}{\delta}\right)\right] \quad -\infty < x < \infty \text{ and } \delta > 0 \tag{15}$$

Minimum $$L_{3,0}(x) = 1 - \exp\left[-\exp\left(\frac{x-\lambda}{\delta}\right)\right] \quad -\infty < x < \infty \text{ and } \delta > 0$$

where λ, δ, and β are the shape and location parameters, which will be estimated from the data and *exp()* is the exponential function.

Now, given samples of maximum data from a number of feature populations, it is possible to select an appropriate limit distribution and fit a parametric distribution model to the data. It is also possible to fit distribution models to portions of the feature distribution's tails as these models are equivalent in the tail to the appropriate underlying distribution. Once the parametric distribution model is obtained, it can be used to compute an effective threshold for a SPC method on the true statistics of the extreme feature data as opposed to statistics based on a blanket assumption of a Gaussian distribution. (Worden et al., 2002)

2.4.1.2 Numeric example

Simulated random data from a lognormal distribution are used to demonstrate the usefulness of the EVS. In the example, the 99% confidence intervals for SPC analysis are computed based on the following three distributions:

(1) The assumed true lognormal parent distribution

(2) A best-fit normal distribution where the sample mean and standard deviation are estimated from the simulated data.

(3) An extreme value distribution, the parameters of which are estimated from the maximum and minimum 10% of the simulated data. Choosing 10% of the data is an arbitrary choice to select a small portion for to represent the tails.

Hereafter, the confidence interval estimation methods based on the above three distributions are referred to as "True", "Normal", and "Extreme", respectively.

Setting a confidence interval on the parent distribution using either the True or Normal cases is fairly trivial. The lower and upper thresholds of the confidence interval are constructed by choosing an upper and lower false-positive threshold, α_U and α_L. This threshold is related to the percentage of false-positive errors in your base line data. For example, an α_L=.05 will correspond to a limit at which the lower 5% of the data will be below the lower threshold. Conversely a value of α_U=.95 will correspond to an upper limit in which the top 5% of the will exceed the upper threshold. By using α_L=0.005 and α_U=0.995 and the inverse cumulative distribution function for either the known distribution or a best fit normal distribution, confidence intervals which encompass 99% of the data can be obtained. The remaining 1% of the data will lie outside the thresholds.

For method 3, the lower and upper thresholds are estimated using the parameters from the parametric modeling of the maxima and minima and a user defined false-positive error bound. Knowing that a Gumbel distribution can model the extreme values of a lognormal (Castillo, 1988) as in Equations (15) the following equation is formulated to estimate the lower limit of the confidence interval:

$$\text{Lower limit: } x_m = \lambda + \delta \ln\left(-\ln\left(1-\alpha\right)\right) \tag{16}$$

where x_M is the threshold, λ and δ are obtained from the Gumbel distribution parameter estimation and α is the false-positive error bound (Worden, 2002). The upper limit of the confidence interval is similarly formulated,

$$\text{Upper limit: } x_M = \lambda - \delta \ln\left(-\ln\left(\alpha\right)\right) \tag{17}$$

Some care must be taken in selecting the α values for the Extreme case in order to obtain thresholds that are comparable when applied to the parent distribution. In these numeric examples, the lower and upper 10% of the data are selected from the parent distribution to be modeled as the extremes. An α value for method 3 that will result in a 1% of the data exceeding the threshold in the parent distribution needs to be selected. When examining the parent distribution, 0.5% of the data will be an outlier, which translates to 5% of the extreme data being outliers. Therefore, the appropriate false-positive error bounds for method 3 would be α=0.05 and α=0.95. This adjustment will allow the thresholds from all of the methods to be comparable.

2.4.1.3 Lognormal distribution

A random variable x has a lognormal distribution if the natural logarithm of x is normal. (Ang and Tang, 1975) For a lognormal distribution, the density function of x becomes;

$$f(x) = \frac{1}{\sqrt{2\pi}\,\sigma x} \exp\left[-\frac{1}{2}\left(\frac{\ln(x)-\mu}{\sigma}\right)^2\right] \tag{18}$$

where ln(x) is the natural logarithm of x. μ and σ are the mean and standard deviation of $ln(x)$, respectively. For this simulation, μ=1.0 and σ= 0.5 are assumed. The associated lognormal density function is displayed in Figure 8. The skewness and kurtosis of this distribution are 1.74 and 8.45, respectively. Note that, for all normal distributions, the values of the skewness and kurtosis should be 0.0 and 3.0, respectively (Wirsching et al., 1995). Therefore, the departure of the skewness and kurtosis values from 0.0 and 3.0 indicates non-Gaussian nature of the data.

The *least-squares return period relative error* method is employed for the maximum of the lognormal data, the minimum, however, is fitted using the *least-squares probability absolute error* method (Castillo, 1988). In this numeric example, several techniques of parameter estimation were employed with the method which best fit the distribution being decided upon visually.

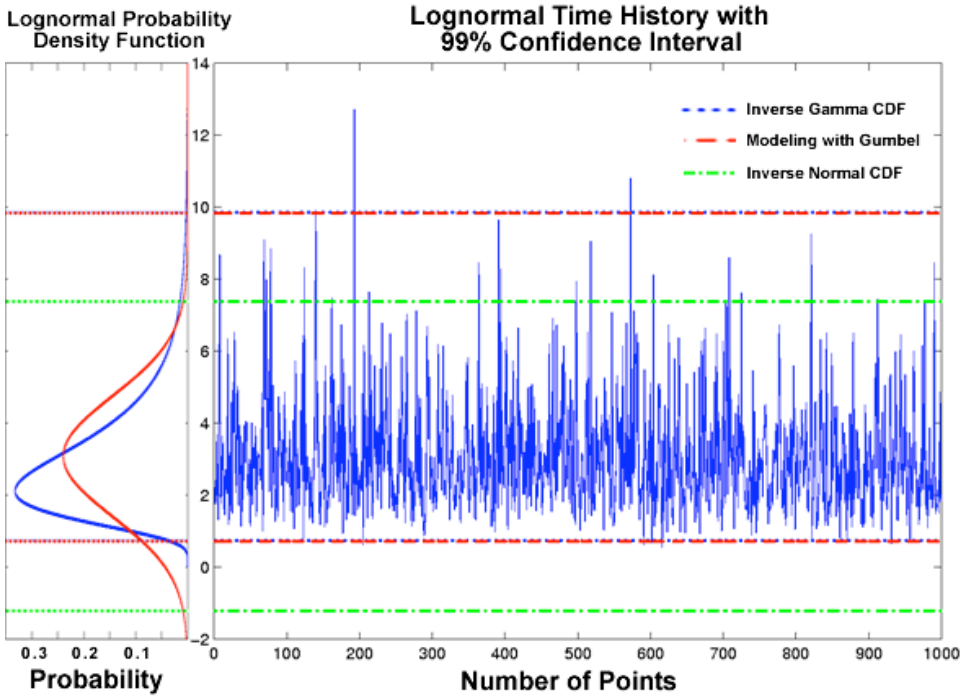

Figure 8 - The exact 99% confidence interval of a lognormal parent distribution compared with those computed from either extreme values statistic or the normality assumption.

Table 2: Estimation of 99% confidence intervals for the 10,000 data points generated from a lognormal parent distribution

Estimation method	Upper confidence Limit	Lower confidence Limit	Number of outliers out of 10,000 samples
True	9.854	0.750	100
Normal	7.378	-1.206	230
Extreme	9.827	0.715	103

A drawback to the extreme values method is that a different method of parameter estimation is used to optimize the distribution fit of the minima and maxima in the example. The advantage is once the extreme values are modeled well there is a noticeable advantage for using this approach to establishing thresholds for SPC (Worden et al., 2002).

2.4.2 Control charts

If the mean (μ_x) and standard deviation (σ_x) of the Gaussian feature distribution are known, a control chart can be constructed by plotting a horizontal line at μ_x and two more horizontal lines representing the upper and lower thresholds. The upper threshold is drawn at $\mu_x + \alpha_U\sigma_x$ and the lower threshold at $\mu_x - \alpha_L\sigma_x$. The number α is chosen so that when the structure is in good condition a large percentage, typically 95% to 99%, of the observed features will fall between the thresholds (Allen et al., 2001).

As each new feature observation is made, the new μ_x is plotted. If the condition of the structure has not changed, the observation should fall between the upper and lower thresholds, with the exact percentage of values falling within the thresholds being determined by the choice of α. If the structure is damaged, there might be a shift in the distribution of the damage feature, which would then be indicated by the number of charted values beyond the thresholds increasing beyond what is consistent with the chosen α. Plotting the individual μ_x observations in this manner is referred to as an *X bar chart* (Montgomery, 1996).

Note that observing an unusual number of observations outside the thresholds does not imply that the structure is damaged, but only that something has happened to cause the distribution of the damage feature to change. In this formulation of the SHM problem, however, it is assumed that a change in the damage feature distribution is caused by an undesirable change in the structure. This assumption implies that changes resulting from varying operational and environmental conditions do not influence the damage-sensitive feature of their influence has been accounted for by data normalization procedures.

2.4.3 Sequential hypothesis tests

One approach to quantifying the existence of damage in a structure is a sequential test. Sequential tests are particularly relevant if the data are collected sequentially such as the acceleration measured at a point on a structure over time. In classical hypothesis testing the number of samples tested or collected is fixed at the beginning of the experiment. After all the data are collected, the analysis is performed and conclusions are drawn. Unlike classical hypothesis testing, in sequential analysis every data point is analyzed directly after being collected. The data collected up to that moment is then compared with threshold values, incorporating the new information obtained from the freshly collected data. This approach leads to forming conclusions about the structural state during the data collection process. By comparing a new test feature to a baseline feature during data collection, a final conclusion can possibly be reached at an earlier stage than is the case in classical hypothesis testing. This sequential analysis is particularly appropriate for continuous SHM.

2.4.4 *Sequential probability ratio test*

Among the various sequential tests, it can be analytically proven that the sequential probability ratio test (SPRT) minimizes the average sample number required to make a decision, thus making the SPRT an optimal sequential test (Ghosh, 1970). The SPRT is also sensitive to minute disturbances in the tested feature distribution. Because of this sensitivity the SPRT has been applied for the critical application of nuclear power plant component surveillance (Gross and Humenik, 1991).

When implementing the SPRT, a trade-off must be considered before assigning values for the false-positive and false-negative error. When there is a large penalty associated with false positive alarms (for example, alarms that evacuate a building), it is desirable to keep α (false positive threshold) smaller than β (false negative threshold). On the other hand, for safety critical systems such as aerospace structures, a false positive alarm is more desirable than a false negative because of the potential for catastrophic failure. In this case, it is not uncommon to specify β larger than α.

Using a SPRT, S(b,a), a hypothesis test can be stated as follows (Ghosh, 1970):

A sequence of feature observations are recorded $\{x_i\}$(i = 1,2,..,n) successively, and at stage n;

$$\begin{aligned} &\text{Accept } H_o \text{, i.e. the system is undamaged, if } Z_n \le b \\ &\text{Reject } H_o \text{, i.e. the system is undamaged, if } Z_n \ge a \\ &\text{Continue observing data if } b \le Z_n \le a \end{aligned} \tag{19}$$

where the transformed random variable Z_n is the natural logarithm of the probability ratio at stage n:

$$Z_n = \ln \frac{f(X_n \mid \sigma_1)}{f(X_n \mid \sigma_o)} \text{ for } n \ge 1 \tag{20}$$

where $f(X_n \mid \sigma_0)$ is the probability distribution function (PDF) of the recorded features (X_n) given a baseline standard deviation (σ_0), and $f(X_n \mid \sigma_1)$ is the PDF given a damage case standard deviation (σ_1). Z_n is defined to be zero when $f(X_n \mid \sigma_1) = f(X_n \mid \sigma_0) = 0$. σ_0 and σ_1 are user defined based on previous data. In a supervised learning mode, σ_1 can be quantified based on data, however in an unsupervised learning mode σ_1 will be decided by an informed user based on knowledge of σ_0 .

b and a are the two stopping bounds for accepting and rejecting if the system is undamaged respectively, and they can be estimated by the following Wald approximations (Wald, 1947):

$$b \cong \ln \frac{\beta}{1-\alpha} \text{ and } a \cong \ln \frac{1-\beta}{\alpha} \tag{21}$$

The region $b \le Z_n \le a$ is called the critical inequality of S(b,a) at stage n. Here there is no conclusion that can be drawn about the state of the system, therefore the test remains open.

2.4.5 Application to extreme value distributions

The SPRT formulation can now be extended to incorporate the EVS analysis of the feature distribution. In the previous section, the SPRT is formulated assuming that the features have a normal distribution. Here the Gumbel distribution for maxima values [Equation (15)] is incorporated into the SPRT. Similar formulation of the SPRT can be easily derived for the other types of extreme value distribution and for minima values.

It can be shown that the parameters, λ and σ, of the Gumble maxima distribution are related to its mean μ_M and standard deviation σ_M (Castillo, 1987):

$$\delta = \frac{\sqrt{6}}{\pi}\sigma_M \text{ and } \lambda = \mu_M - 0.57772\delta \tag{22}$$

If the distribution of the maxima is preprocessed such that the mean value is zero, Equation (20) can be rewritten in terms of λ and σ, then if $\{x_i\}$ are independent and identically distributed, substituting in the Gumbel parameters in Equation (15) results in:

$$z_i = \ln \frac{f(x_i \mid \lambda_1, \delta_1)}{f(x_i \mid \lambda_o, \delta_o)} \text{ for } i = 1, 2, \ldots, n \tag{23}$$

Finally, the cumulative sum of the transformed variable, z_i, is monitored against the two stopping bounds, a and b calculated in Equation (21). A z_i value less than a is indicative of acceptance of the hypothesis that the structure is undamaged, while a z_i greater than b indicates an acceptance of the hypothesis that the structure is damaged. For a more detailed explanation of the process and a less general conclusion please refer to Sohn, 2003.

2.4.5.1 Numerical Examples

In this section, the performances of three variations of the SPRT are compared for different types of parent distributions. The three variations of the SPRT include:

(1) The conventional SPRT with the normality assumption of data sets [Equation 23]

(2) A SPRT formulated using a Gumbel distribution for maxima [Equation (15)]

Hereafter, these techniques are referred to as SPRT-1, and SPRT-2 respectively. These two SPRT techniques are applied to data sets generated from a lognormal distribution. First, a set of data consisting of 8192 observations and a known standard deviation of σ_x is. This data set simulates samples of the damage feature from the initial intact condition of the structure. The second data set also consists of 8192 data points has a modified standard deviation of $\sigma_y = F\sigma_x$. Here, F is a multiplication factor varying from 0.90 to 1.00, 1.10, 1.15, 1.45, 1.50, 1.60 and 1.70. This data set represents samples of the feature from test structural conditions.

The damage classification problem is cast in such a way that, if the standard deviation of the test feature σ_y becomes greater than a predetermined upper limit, $1.4 \cdot \sigma_x$, then the new signal is considered to be from a damaged state of the system. On the other hand, if σ_y is less than the predetermined lower limit $1.2 \cdot \sigma_x$, the new signal is then assumed to be from the undamaged condition. Otherwise (when $1.2 \cdot \sigma_x < \sigma_y < 1.4 \cdot \sigma_x$), the damage classifier cannot make a confident decision regarding the current state of the structure and continues collecting additional data. In the numerical examples, the upper and lower thresholds, $1.2 \cdot \sigma_x$ and $1.4 \cdot \sigma_x$, are selected rather arbitrarily. In real applications, the sensitivity of the feature with respect to damage of interest should be first examined to establish these two thresholds. This sequential hypothesis test can be stated in a simplified format:

$$Undamaged: \ \sigma_y \leq 1.2\sigma_x \quad \text{and} \quad Damaged: \ \sigma_y \geq 1.4\sigma_x \tag{24}$$

Because the statistical inference in Equation (24) is imposed only on the unknown standard deviation σ_y, it is assumed that the mean of the signals is zero. Therefore, data normalization that subtracts the mean of each signal is performed before the hypothesis test.

When the SPRT is combined with extreme value statistics (SPRT-2), a moving window of 16 time samples is stepped through the 8192 points of each data set to generate 512 maxima for each condition. That is, the sample size for the maximum value selection is set to be 16 (m=16). The bounds of false positive and false negative errors are set to 0.001. The corresponding two bounds are b = –6.9 and a = 6.9, respectively. It should be noted that because the parent distribution is assumed unknown for SPRT-2, the hypothesis test in Equation (24) cannot be performed and an alternative hypothesis test is conducted on the standard deviation of the "maximum" values;

$$Undamaged: \ \sigma_{M,y} \leq 1.2\sigma_{M,x} \quad \text{and} \quad Damaged: \ \sigma_{M,x} \geq 1.4\sigma_{M,y} \tag{25}$$

where $\sigma_{M,x}$ and $\sigma_{M,y}$ are the standard deviations of the "maximum" values for the first and second sets of signals, respectively.

2.4.5.2 Lognormal parent distribution

The analysis results are summarized in Table 3. Although the formulation of SPRT-1 is based on the normality assumption, SPRT-1 surprisingly performs well even for a lognormal distribution. The performance of SPRT-2 is comparable with the previous result of the normal case. Again, the several misclassifications of SPRT-2 in Table 3 are mainly attributed to the difference between the stated and actual hypothesis tests. The SPRT-2 test assumes that the true maxima will be available for testing, i.e. that the samples will be ordered and the maximum values picked out. In this example, however, the moving window will most likely catch some data that are not true maxima of the set, resulting in a slightly skewed estimate of the maxima distribution.

Table 3: Damage classification results for lognormal distribution data

Hypo	H_o				H_1			
F	0.90	1.00	1.10	1.15	1.45	1.50	1.60	1.70
SPRT-1	100/0/0*	100/0/0	100/0/0	99/1/0	100/0/0	100/0/0	100/0/0	100/0/0

SPRT-2	100/0/0	100/0/0	93/1/6	66/15/19	100/0/0	100/0/0	100/0/0	100/0/0

*The first number denotes the times a correct hypothesis is accepted, and the second number denotes the number of rejected correct hypothesis. The last value is the number of cases where the SPRT cannot draw a conclusion based on the given data sets. For example, 100/0/0 means that, out of 100 simulations, 100 cases are correctly assigned to the true hypothesis and there were no misclassification or undecided cases.

When the cumulative statistics (Z) is plotted graphically, the result is similar to Figure 9. As features are analyzed the algorithm tends toward accepting the hypothesis that the structure is undamaged. It can also be seen from this figure how quickly the SPRT algorithm is able to come to a decision.

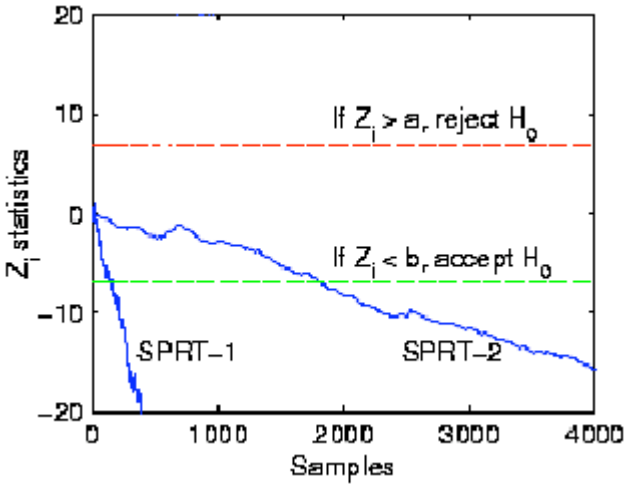

Figure 9: A typical SPRT damage classification result for data sets from a lognormal distribution (Correct decision: accepting H_0)

2.5 Summary

In an effort to develop an automated and quantitative method for unsupervised damage identification, a unique integration of data normalization and cleansing procedures, time series analysis for feature extraction, and feature statistical modeling for feature discrimination that incorporates EVS is undertaken. First, time series modeling techniques, solely based on the measured response signals, are fit in an automated fashion and deployed to extract damage sensitive features. In this study SPC is employed to provide a more automated statistical tool for the decision-making procedure, excluding unnecessary interpretation, such as looking at charts of the observed feature, by users. Finally, the performance and robustness of damage classification is improved by incorporating EVS of the extracted features into SPC. Data normalization and cleansing are introduced as necessary into the process. The framework of the time series analysis and SPC method is well suited for integrating into a continuous monitoring system.

2.6 Contributions

- The development of automated time series order selection using AIC, FPE, and SBC criteria.
- The development of the Sum of Squared Error damage sensitive feature.

- Numerical examples and demonstrations for EVS and integrated EVS-SPRT methods of statistical feature discrimination.

3 Client Side Software Environment

3.1 Introduction

A structural health monitoring (SHM) team at Los Alamos national Laboratory (LANL) conceived the idea that grew into the GLASS, Graphical Linking and Assembly of Syntax Structure, software. The concept was for software that would allow a user to assemble statistical pattern recognition functions into a SHM process in the same manner as assembling a puzzle. The project developed from simple graphical interfaces to a modern piece of software that provides easy user interaction, expandability, and is easy to maintain.

As described in the previous chapter, several tools are being developed in an effort to provide accurate and quantitative SHM. The original LANL toolbox, DIAMOND (Doebling, 1997), is a graphical-user-interface (GUI) driven MATLAB toolbox for experimental modal analysis, finite element model updating and damage identification based on changes in modal properties. This toolbox was developed in the mid 1990's by staff and students in LANL's Engineering Analysis group. With a shift in paradigm from global modal parameter based damage identification (Doebling, 1996) to statistical pattern recognition based SHM (Farrar, 1999), a new tool is needed to reflect this change. The DIAMOND II module and GLASS technology have been created to meet this demand. DIAMOND II, like its predecessor, is a collection of functions based in MATLAB that are assembled to provide SHM data interrogation tools. These tools can be categorized as "Data Collection," "Data Cleansing and Normalization," "Feature Extraction," and "Statistical Discrimination." Functions from these categories are assembled to form a SHM process. The DIAMOND II MATLAB algorithms have been encapsulated so that each is a stand-alone function with defined inputs and outputs. The functions are also based on a single data structure allowing them to be assembled in a "*plug and play*" manner.

With this new catalog of *plug and play* functions, an interface is needed to allow simple assembly of a SHM process. The GLASS client platform facilitates construction of new processes by allowing drag and drop of MATLAB, C, or JAVA functions into a workspace. Variable types, values, and descriptions are displayed and dragging output variables from one function to the input variables of another easily links the two functions. Once assembled, a process can be run in its entirety or selected functions can be run as needed. Processes can then be saved and stored for use in the future, executed remotely, or embedded into microprocessors. GLASS is developed in the JAVA programming language to allow for cross-platform compatibility, and to incorporate modular design allowing for future expansion. In GLASS, DIAMOND II is one of several modules containing data interrogation functions. Other modules include a hardware integration module, a utilities module, and an experimental modal analysis module.

This chapter is compiled in the following order: the creation of the GLASS client software, using GLASS technology with emphasis on process development, and a discussion of the DIAMOND II data interrogation module with an overview of included functionality. Finally, the GLASS software is summarized and a single SHM process is specified for use in the experimental portion of this thesis.

3.2 Development of GLASS Technology

3.2.1 Developing with an object oriented approach

The first GLASS software versions were developed using the MATLAB GUI environment. It was quickly realized that a more powerful language was needed to capture the required functionality. JAVA was chosen because of the ability to compile the software independent of a computing platform, ease of development, and JAVA's Object Oriented (OO) language structure. The use of an OO language allows development of reusable objects, decreasing development and revision time.

OO software emulates abstracted objects from the real world. In the case of the GLASS software, the objects to model come from the functions, and the organization of these functions. The following is a bottom up listing of the objects that are emulated in the GLASS software:

Variables: Each variable object is assigned a name and type corresponding to the MATLAB workspace. The object also has a value and description with which it is associated. An experimentally measured 1024-point acceleration time history is an example of an array variable.

Functions: Function objects encapsulate the ability to execute MATLAB, C or JAVA code. Function objects also contain information pertaining to its description, authorship, purpose, and input and output variables. An algorithm to perform a Fast Fourier Transform (FFT) is an example of a function.

Categories: In the DIAMOND II module, the categories reflect the statistical pattern recognition paradigm for SHM. In GLASS, a tabbed pane represents each category and displays the contained functions. The FFT function might be part of the feature extraction category within DIAMOND II.

Module: A module is the top level of organization in GLASS. DIAMOND II has been developed as a module to allow future expansion and easier assembly of processes. Each module consists of multiple categories. Other modules add functionality in different aspects of structural dynamics such as model validation and uncertainty quantification or experimental modal analysis.

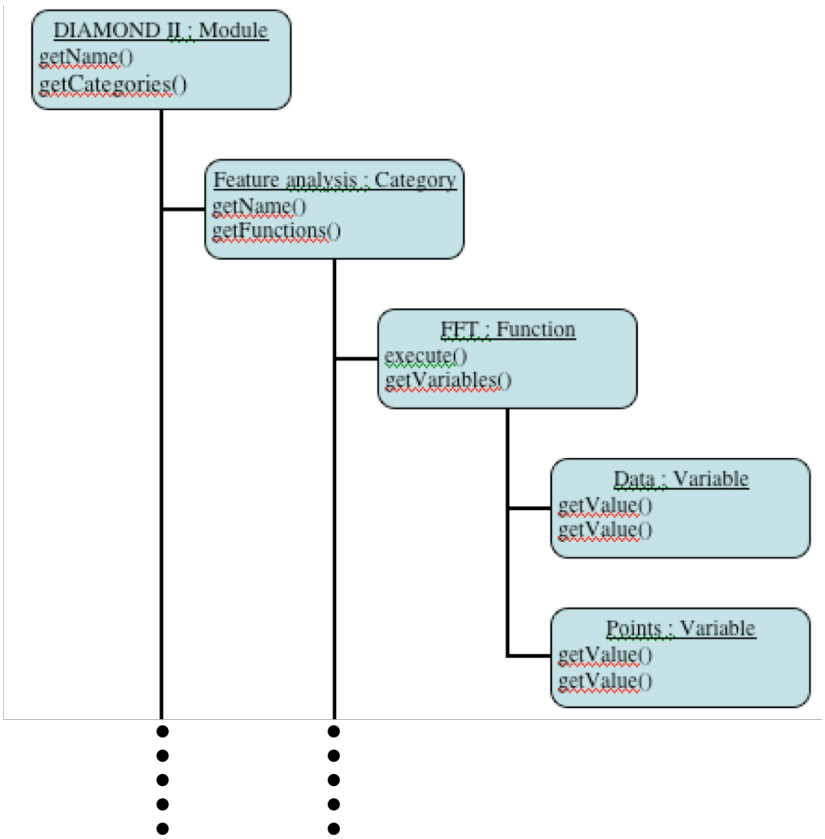

Figure 10 – Universal Markup Language (UML) Diagram for objects that make up the GLASS software system. Each block represents an object in the JAVA language with the name of the object instance, the type of object, and example methods. UML is typically used to diagram software systems, but can be applied to all kinds of processes.

Figure 10 shows how these JAVA objects are connected to create the GLASS environment. The diagram is created using the Universal Markup Language (UML) (Larman, 2002). UML is typically used in describing software, but can be applied to a variety of systems and processes. Each box represents an object. The three portions of the box are the instance name, object type, and methods. Between the objects are connecting lines. For example, FFT is an instance of a *Function* object. The object can return its name or a list of contained variables. A single *Function* object can also contain many *Variable* objects. An object exists that is not shown in Figure 10, the *GlassComponent* object. The *GlassComponent* is an abstract object that contains properties and methods that are common to many of the GLASS objects. Object oriented programming is useful because of the concept of inheritance. The *Module*, *Category*, and *Function* objects all inherit all of the properties and methods of the abstract *GlassComponent*. This inheritance also allows Modules, Categories, and Functions to be treated interchangeably on an abstract level. For example, the *Glass* client requesting the name of a *GlassComponent* does not make any distinction as to whether the component is a *Module*, *Category*, or *Function*.

The variable objects utilize this idea of inheritance as well. An abstract object *Variable* has several realizations: *VarString*, *VarArray*, and *VarScalar*. At an abstract level, all of these variable types are treated the same, however the different types can be used to check if proper input has been provided.

3.2.1.1.1 Integrating a JAVA framework with MATLAB

In order to execute functions assembled from JAVA objects in the MATLAB workspace, a link between the two workspaces must be created.

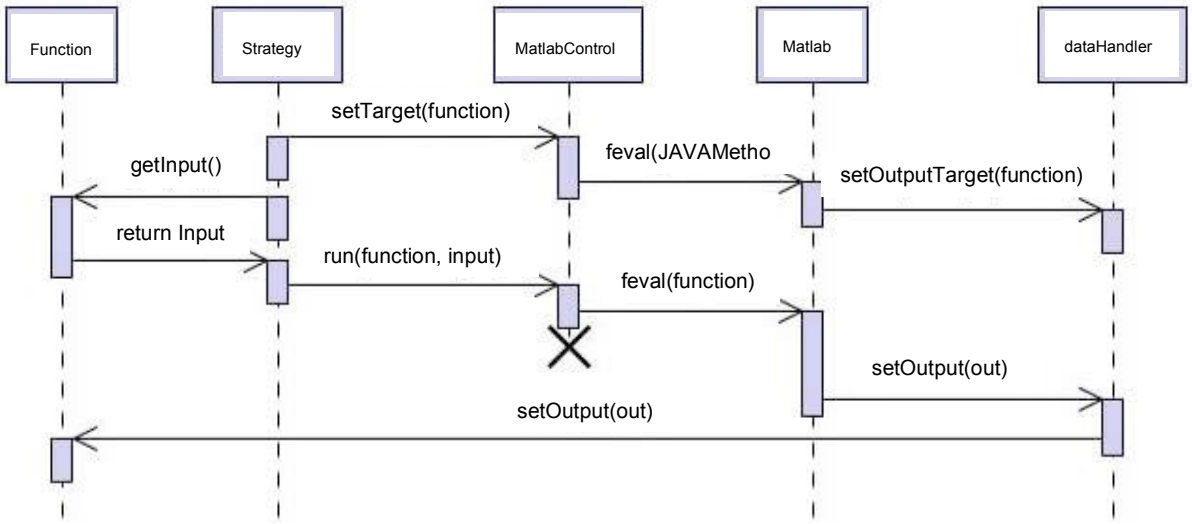

Figure 11 - Execution of MATLAB functions involves JAVA objects in both the JAVA virtual machine and the MATLAB workspace. Because of the JAVA-MATLAB Interface, messages and information can only flow from left to right in MATLAB.

Figure 11 is a sequence diagram showing how a function would be executed using the GLASS technology. Several new objects have been introduced in this diagram. The *Strategy* object contains the sequence of method calls to execute a function in the MATLAB workspace. The *MATLABControl* object encapsulates the JAVA MATLAB Interface (JMI), allowing JAVA to evaluate commands in the form of a string in the MATLAB workspace. The *dataHandler* object resides in the MATLAB workspace and allows results from a calculation to be retrieved by JAVA objects. Information and objects are passed between objects following the arrow directions and execution time flows in a downward fashion. For example, the first action called is *setTarget* by the *Strategy* object and a *Function* object is passed to the *MATLABControl*.

There is a problem with creating a JAVA interface to MATLAB. JAVA can communicate by evaluating strings in the MATLAB workspace, but MATLAB has no direct way to interact with JAVA objects created outside of the MATLAB workspace. Creating the *dataHandler* object in the MATLAB workspace allows the JAVA object access to the MATLAB workspace variables. Other JAVA objects outside of the MATLAB workspace can then query this JAVA object. This object allows results to be retrieved from the MATLAB workspace and the appropriate *Variable* object to have its value updated.

Another approach to this problem was taken using the Model/View/Controller (MVC) pattern with JMatLink, a JAVA library that calls MATLAB via native expressions. The MVC pattern suffered from synchronization issues. When JMatLink commands are run, the JAVA commands are run first, and then the MATLAB functions are executed second. In this case, GLASS would call all of the functions, MATLAB would execute all of the functions, and then the MVC would attempt to update variables in only the last function. A correct implementation would execute a single function and then update the output before moving to the next function. When implementing JMatLink without the MVC, problems arose such as functions being run out of order. In addition, frequent deadlock has been experienced with the JMatLink implementation. Deadlock is a situation in which the program is waiting for a command that is never issued and "locks" or "hangs" the program. The pure JAVA-MATLAB Interface (JMI) implementation produced similar synchronization issues as the MVC pattern. Deadlock was also experienced with the JMI implementation. Because the JMI is not documented or supported, some commands have never been fully implemented. By using a *thread safe class* to implement the JMI, all synchronization issues were resolved. A thread safe class is an object that is run independent of the main program. If this *thread* experiences deadlock it can be easily terminated without the entire program suffering. This threading of the function execution allowed a list of functions to execute in sequence and correctly pass variable values between them.

With the link between the JAVA and MATLAB workspaces established, functions can be run, variables retrieved, expressions evaluated, and MATLAB commands executed. This method provides an extraordinary tool that allows graphical manipulation of objects via a JAVA interface and subsequent computational execution in MATLAB. JAVA provides a far superior user interface than the native MATLAB GIU toolbox because of the drag and drop functionality, threading, and advanced user controls such as the process tree.

Once MATLAB functions became executable, JAVA classes and C functions were easily encapsulated in the same JAVA function framework. This framework allows functions written in the different languages to be integrated into a single process, sharing variable values and functionality.

With GLASS Technology, there exists the ability to create functions in MATLAB, JAVA or C and then categorize, visually assemble, and have them execute in a process. The next section discusses the assembly of a process from individual functions.

3.2.2 Graphically prototyping algorithms

In GLASS, functions are categorized as belonging to a Module and a Category. This hierarchy allows a separation of functions by developer, project, or method. For example, the DIAMOND II Module is a collection of functions developed by many people in an effort to use the statistical pattern recognition paradigm to tackle the SHM problem. This module is then broken into the categories representing the steps followed to analyze data using time series analysis. Another module, Hardware integration, contains functions for accessing a data acquisition board to collect data and functions for broadcasting data or results over a network.

GLASS modules can be created, stored, and shared among users. Functions from different modules may also be combined together to form new processes. This modular approach was taken in an effort to reduce the number of

functions re-written by individuals when various functions (e.g. importing a specific file type) have already been written.

To assemble a new process, functions are selected from the categories and placed in the workspace. Functions can be re-ordered or inserted at anytime. Functions are then linked by their input and output variables in a cascading fashion. The following example process is used to collect data from hardware, process the data and then return a result.

The first step to assembling a process is data collection. When the data (measured acceleration time histories) are collected live from a DSP board described in Chapter 4, a JAVA class for communicating with the DSP board is used. The function *collectDSPdata* starts the process. In this function, the number of data points, sampling frequency, and IP address of the hardware are specified.

Next data cleansing is performed. Because the DSP board used for data collection is a custom and experimental board, there is a small transient response at the beginning of all the samples taken. The *subset_data* function is the next step in the process and is used to truncate the sample, removing the initial transient data.

A damage sensitive feature must be extracted from the time series. In this example, features are based on fitting a time series model to measured acceleration time-histories. The residual error that results when this model is used to subsequently predict future data sets is considered the damage sensitive feature as described in chapter 2.

Finally, statistical modeling for feature discrimination is obtained using an X-bar control chart. This test is used to determine when there are significant changes in the residual error features. The result of this statistical test is then broadcast by a JAVA function over the network to other clients. Figure 12 shows the GLASS GUI implementation of this process.

This process is given purely as an example of how to assemble a process and return results calculated in MATLAB. For a more in depth look of the process, please refer to Chapter 2.

The internal mechanics of adding functions to the GLASS workspace is shown in Figure 13. The GLASS workspace is designated as the *Routine* object. This object holds the list of *Function* objects making up a process. The *RoutineTree* object is the visual representation of the *Routine* object. The *RoutineTree* displays the functions, variables, and information in a tree format similar to a file browser. All the visual construction of the algorithm interacts with the *RoutineTree* object such as dragging in new functions, connecting functions via variables, and re-ordering objects. The underlying *Routine* object is updated every time the *RoutineTree* object is changed.

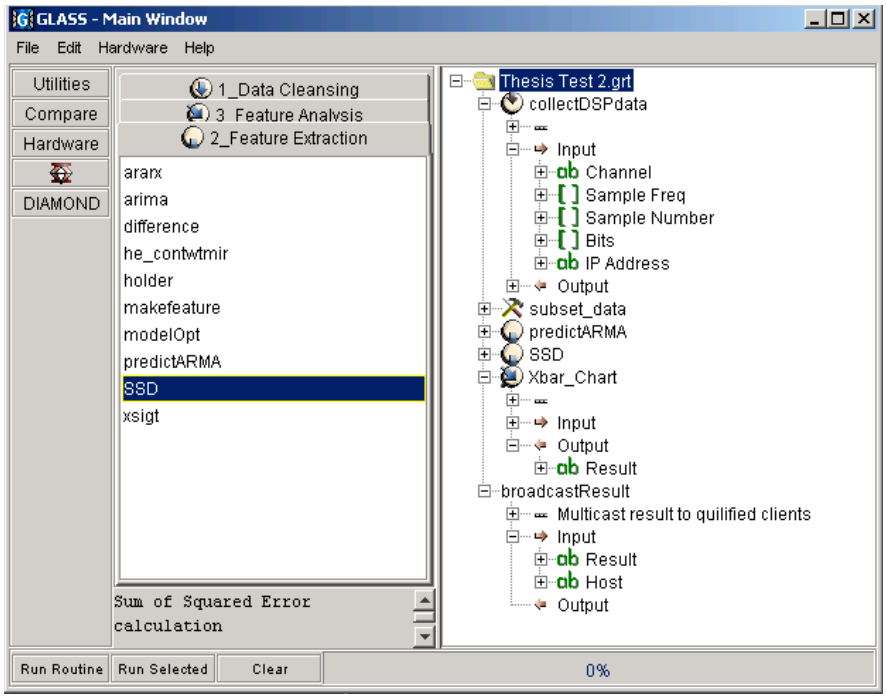

Figure 12 - The GLASS GUI showing an algorithm assembled for performing an SPRT analysis of a frame structure data collected by an HP analyzer.

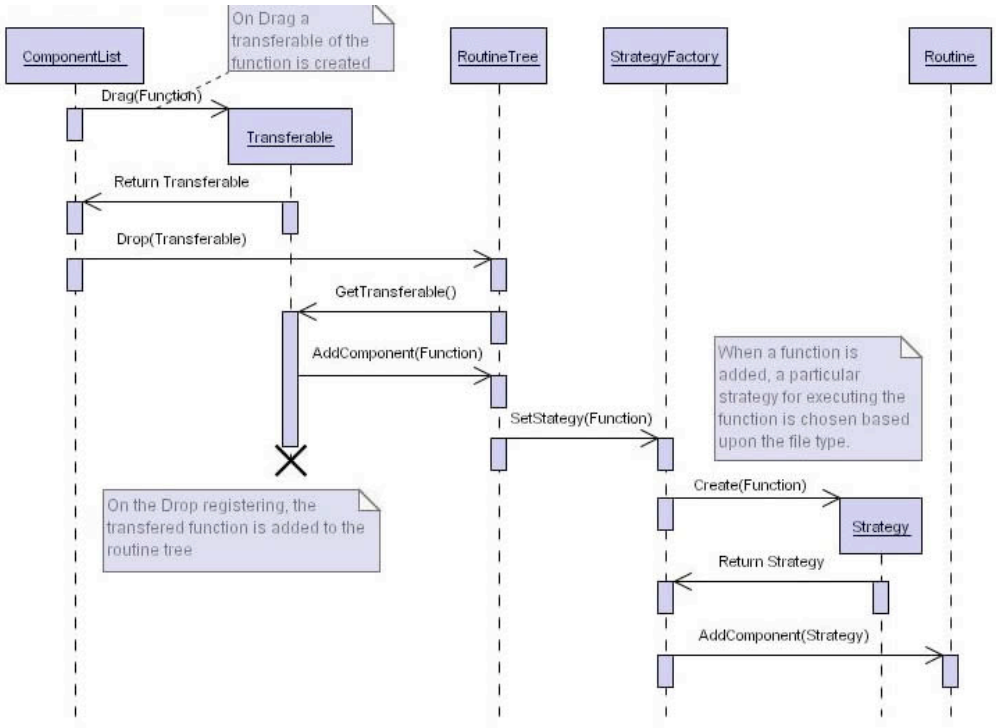

Figure 13 – This figure is of a sequence diagram for adding a function to the GLASS workspace.

In the first steps of dragging and dropping a function the actual object passed is a *Transferable*. A *Transferable* object is an object with a wrapper that allows it to be passed in a basic universal protocol. For example, once an object is made into a *Transferable* object it could be shared with a word processing, graphics, or spreadsheet program. In this case, the *Transferable* object allows the function to be passed from one JAVA object to another using drag and drop.

Once the function has been added to the *RoutineTree*, the visual representation, it must be added to the *Routine* to facilitate functionality. It can be seen in Figure 13 that again the function has been wrapped in another object, a *Strategy* object. The *Strategy* object encapsulates the function with ability for execution. For example, a function may be wrapped in a *Strategy* object that executes the function in MATLAB. Another *Strategy* object may contain functionality for executing C code, or FORTRAN. This approach again makes GLASS a more flexible solution and expandable in the future.

Once assembled, algorithms can be run in their entirety or in selected sequences. The idea is that once an algorithm has been assembled, and run once, small changes to parameters should not require the entire sequence to be run again, only affected functions need to be rerun and the final results recalculated. New functions can also be dragged into the workspace and results recalculated to compare and contrast two methods. Possibilities might include comparing results from two different normalization functions.

Once a process has been created, it can be saved for future use or passed on to other individuals. GLASS Technology has been developed to be an open ended and cooperative endeavor that will save time and promote understanding of different approaches to SHM.

3.3 Summary

Using GLASS Technology allows users to categorize, share, and re-use SHM data interrogation functions. Previously every researcher had their own version (or versions) of a MATLAB function that loaded data from a specific file type. Now a single version of the function can be re-used and shared with others. Functions were also previously pasted together in large and cumbersome master functions that executed a particular algorithm. Integrating a new function often required a lot of cut and paste as well as checking indexing, variable names, and general continuity. GLASS allows assembly of functions into a SHM process by an intuitive drag and drop procedure, similar to moving files around in any modern operating system. Functions written by different individuals for entirely different applications can also be easily incorporated into new algorithms.

GLASS Technology will enable researchers to develop and share functions in a common platform that in turn allows them to quickly prototype new SHM processes. Upon assembly, these processes can be shared for review or placed into a hardware environment for testing.

The DIAMOND II Module makes years of research and development in SHM data interrogation algorithms at LANL and Virginia Tech available to the new user in a very flexible and adaptable software tool. By using the provided functions, new users to the software can quickly assemble and assess various combinations of data cleansing, feature extraction, and feature analysis. The users can then analyze their own data or incorporate their own newly develop function and have a benchmark for comparison.

3.4 Contributions:

- The development of an innovative connection between the JAVA programming language and the MATLAB computational environment to facilitate the transfer of variables between functions.
- The creation of a GUI development interface for the rapid prototyping of new SHM processes from a standardized set of functions.
- The collection and modularization of statistical pattern recognition tools developed by LANL and Virginia Tech to be applied to the SHM problem.

4 Node software integrated with sensing and processing hardware for inline monitoring

4.1 Introduction

The ability to prototype new structural health monitoring (SHM) processes is only half of the solution. To develop a true integrated SHM system, the developed processes must be transferred to embedded software and hardware that incorporates sensing, processing, and the ability to return a result either locally or remotely. Of the off-the-shelf solutions currently available or in development, there is a deficit in processing power that limits the complexity of the software and SHM process that can be implemented. A SHM process is implemented in these systems, often at the detriment of the complexity of the process. Many integrated systems are inflexible because of tight integration between the embedded software, the hardware, and sensing.

To implement a computationally intensive processes such as described in Chapter 2, a single board computer (SBC) is selected to provide true processing power in a compact form. Also included in the integrated system is a Motorola developed digital signal processing (DSP) board with six analog to digital converters (ADC) providing the interface to a variety of sensing modalitites. Finally a Motorola wireless network board provides the ability for the Husky system to relay structural information to a central host, across a network, or through local hardware. Each of these hardware parts are built in a modular fashion and loosely coupled through the transmission control protocol (TCP) or user datagram protocol (UDP) Internet protocols (IP). Building a loosely coupled group of modular hardware and software makes the Husky system extensible and adaptable. By implementing a common interface, changing or replacing a single component does not require a redesign of the entire system.

A node version of the GLASS software is designed to run and communicate with this modular hardware. By allowing processes developed in the GLASS client to be downloaded and run directly in the GLASS node software, the Husky project becomes the first hardware solution where new processes can be created and loaded dynamically. This modular nature does not lead to the most power optimized design, but instead achieves a flexible development platform that is used to find the most effective combination of algorithms and hardware for a specific SHM problem. Optimization for power is of secondary concern and will be the focus of follow-on efforts.

The next section will describe briefly the hardware that is being provided by Motorola. After the hardware is discussed, the GLASS node software that provides the communication between the development platform and the hardware, as well as the functionality provided by the node software will be discussed in depth.

4.2 Hardware

As mentioned in the introduction, the hardware is designed in modular boards. The PC-104 specification (PC-104 consortium, 2003) is implemented for determining the size and design of each of the hardware boards. Drivers are written for each of the hardware portions that allow communication between the hardware boards

through a common protocol over a TCP socket as is seen in Figure 14. This communication setup allows the back end server to communicate with hardware in an encapsulated form. TCP also allows each hardware portion to be accessed individually over an Ethernet connection for testing while the GLASS node software is running in emulation on a desktop platform.

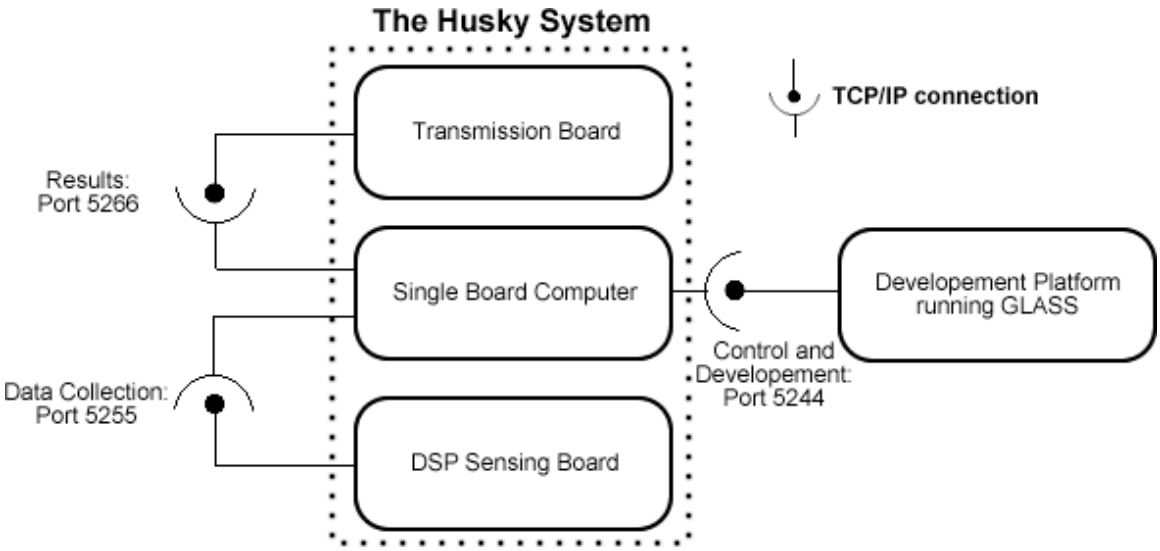

Figure 14 - An overview of the hardware configuration showing the modular approach and using Ethernet protocols for connecting modules.

4.2.1 Single board computer

For the processing center of the Husky project a SBC (Figure 15) is used to provide powerful processing capabilities. The SBC houses a 133 MHz Pentium™ processor, 256 Mb of RAM, and a Compact Flash (CF) card slot that acts as the hard drive. The SBC can support serial, Ethernet and USB communication with other hardware. Developed by Micro/Sys, the SBC adheres to the PC-104 standard and is easily linked to other hardware via the PC-104 bus, or the previously described connections.

4.2.2 Sensing board

The sensing board utilizes a Motorola DSP56858 chip (www.motorola.com) for reading the ADCs and communicating with the SBC. A final production board is shown in Figure 16. A DSP is necessary for sampling the ADCs because of the sampling speed requirements. The DSP is an optimized package able to sample and return samples in four seconds for 1024 samples. The SBC would not be fast enough to read the ADCs and buffer results by itself. The six ADCs are Maxim chips with a maximum sampling speed of 200 kHz.

The DSP board communicates with the SBC, or other command sources, over the serial port through a TCP socket. For example, a command to sample from the DSP board is sent to port 5255 from the CPU, this command is then received on port 5255 and relayed to the serial port and the command is then received by the DSP. The TCP socket is implemented to remove dependence on the serial port. In the future, if the DSP board were to implement the PC-104 bus, the only interfaces that need to be rewritten would be the TCP socket-serial port interface, not the complex code that actually sends the commands.

Because the SBC and DSP board follow the PC-104 standard, they are able to stack on top of one another as seen in Figure 17.

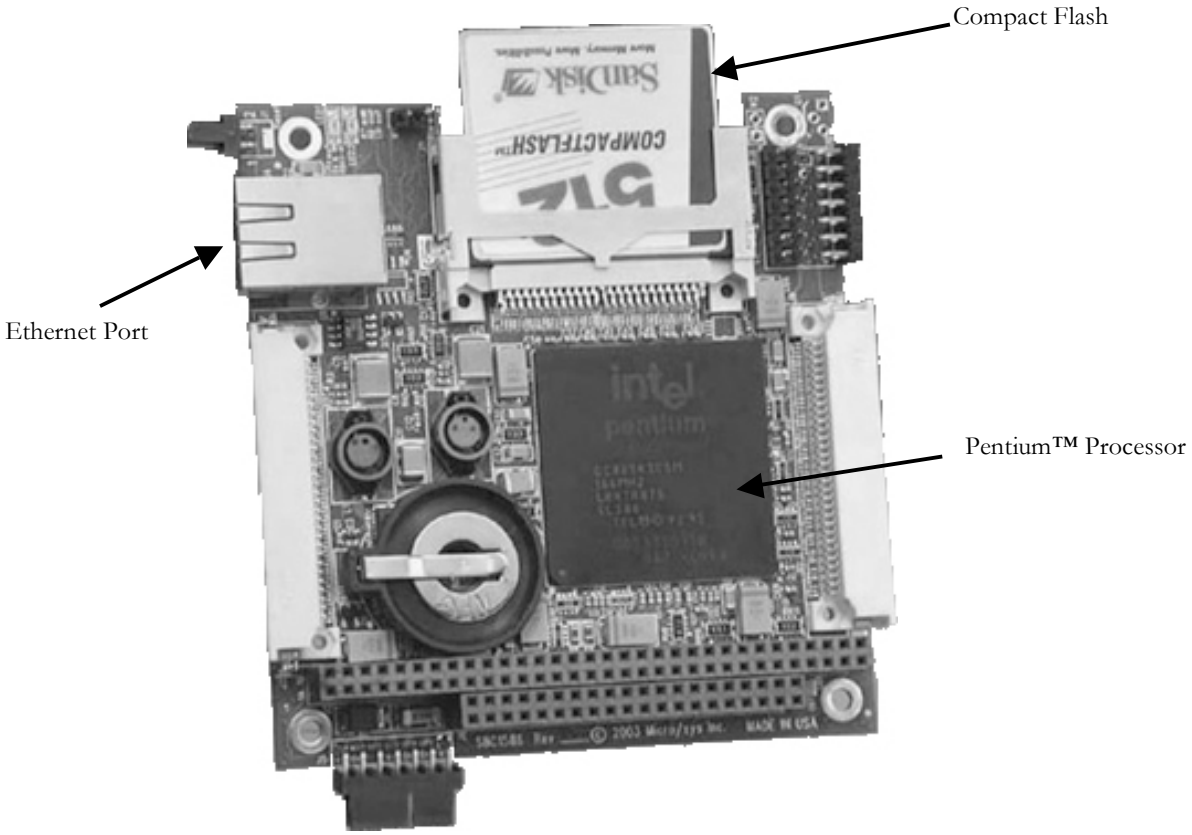

Figure 15 - Single board computer (actual size) showing the processor, the compact flash drive, and Ethernet port.

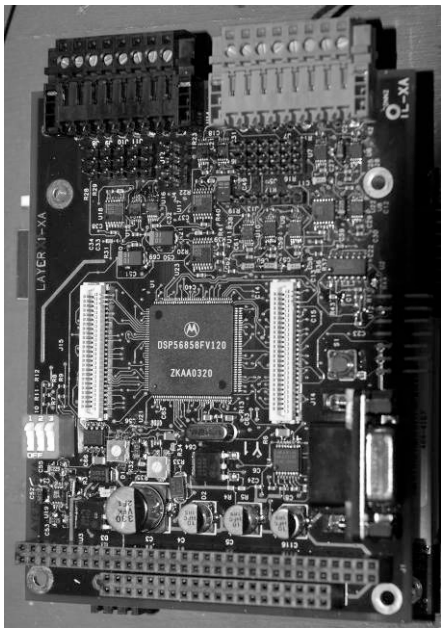

Figure 16 - Final DSP board designed on PC-104 specs and incorporating 6 ADCs.

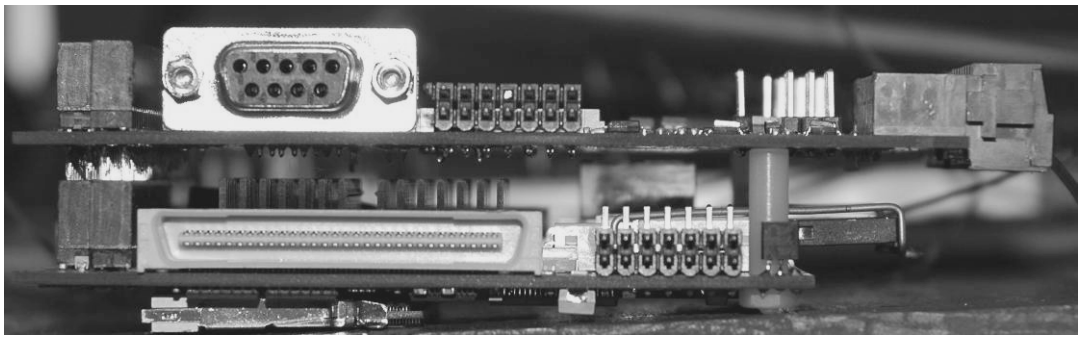

Figure 17 - Showing the stacking of the DSP board on top of the SBC.

4.2.3 Transmission board

The neuRFon™ transmission board developed by Motorola provides a wireless access point to the Husky system. The neuRFon™ board adheres to the IEEE 802.15.4 standard and is designed to be a self-organizing network. For example, if several boards are located within transmission range of each other, a network will be created and data dynamically routed along the most efficient path to a host node. The host node provides connectivity with an external network. The advantage to this network arrangement is if one node becomes disabled or more nodes are added, the network can dynamically reconfigure.

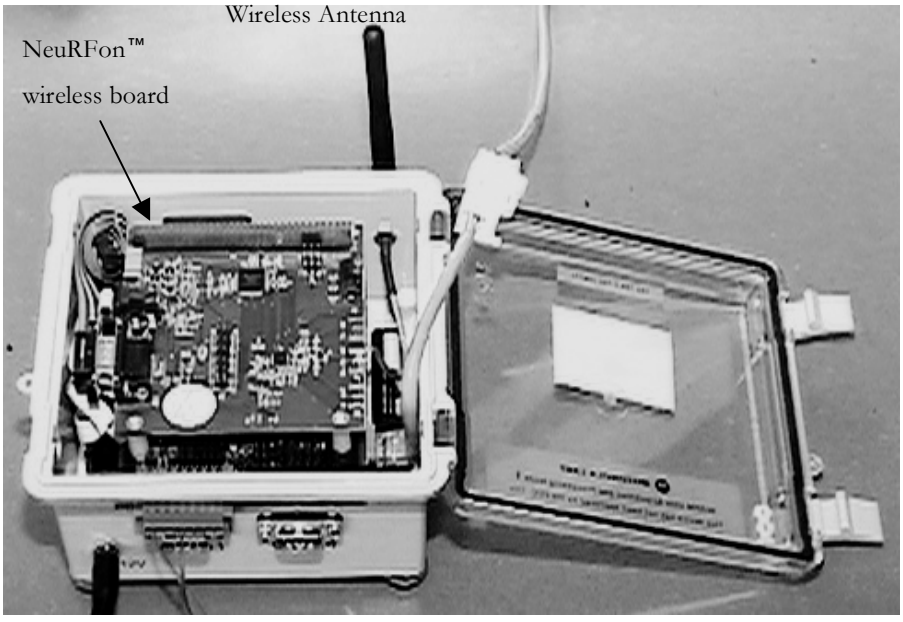

Figure 18 - The Motorola neuRFon™ wireless communication board displayed on top of the prototype system.

Again the wireless board is attached to the SBC through a TCP socket, allowing the neuRFon™ board, another wireless solution, or simply an Ethernet cable to act as a sending/receiving gateway.

4.3 GLASS Node Software

A team from Motorola assembled the hardware. The contribution of this study, therefore, is not in the hardware, but in the software development that allows the hardware to be so flexible, and allows newly created SHM processes to be dynamically loaded for execution on the SBC. This software is referred to as node software because it runs on the monitoring hardware without any user interface. The GLASS graphical user interface (GUI) is the client or software that interacts with a human user.

4.3.1 Embedding overview

Originally, the system design called for the GLASS client software to embed a developed process directly to a DSP chip. In researching the process of embedding developed MATLAB functions into a DSP it was found that no clean and simple solution was available. Some tools existed for targeting the MATLAB functions for DSP chips; however, the conversion produces bloated code and linking external math libraries proved to be difficult. The conclusion was made that the functions would need to be re-written in C if a SHM process were to be easily embedded on a DSP directly from GLASS.

Because of the time already invested in functions developed in MATLAB code, rewriting the functions in C is undesirable. The solution became to implement the full SBC. The SBC solution allows the Linux OS to be run and MATLAB in its entirety. MATLAB functions can now be run without a conversion to C. Harnessing the full power and flexibility of the MATLAB computational engine is also a significant advantage in SHM process development.

Communication with the GLASS node is accomplished from the GLASS client through a TCP/IP socket. This connection allows the Husky system and the process development platform to be physically separated but connected through a local area network (LAN), Wireless LAN, or over the Internet.

The GLASS node allows a process that is created in the GLASS client to be downloaded over this TCP socket and then run in an autonomous and continuous fashion. The process can collect data, process the data, and then send a result repeatedly until a stop command is sent.

4.3.2 GLASS node software

The GLASS node software is both similar too and yet very different from the GLASS client. Where the user of the GLASS client is an engineer assembling SHM processes, the user of the GLASS node is a GLASS client. Because the user is another piece of software, the graphical user interface (GUI) portion of the software that allows a human user to graphically communicate with the software needs to be replaced. Pieces of software communicate with each other through a communications protocol, a set of commands and responses that designate actions.

Because the development of the original GLASS client is object oriented (OO) based, many of the objects could be reused for the node software. The OO development also allows the process and function objects to be easily transferred over an Ethernet socket, allowing the node to share and run objects created on the client.

The node software is also designed to reside on an independent piece of hardware, such as the Husky system or a remote desktop, and continuously run. Once a client has connected, loaded a process, and set it to run, the node will dispatch a thread that will allow the process to run repeatedly until a stop command is sent from a client. In the future, an authentication protocol will be developed to ensure that only users registered with the node can make changes.

4.3.3 Client integration

The original GLASS client was designed solely for constructing SHM processes on a desktop. Now, however, integration in the client is added to allow communication with the GLASS node software to share constructed processes.

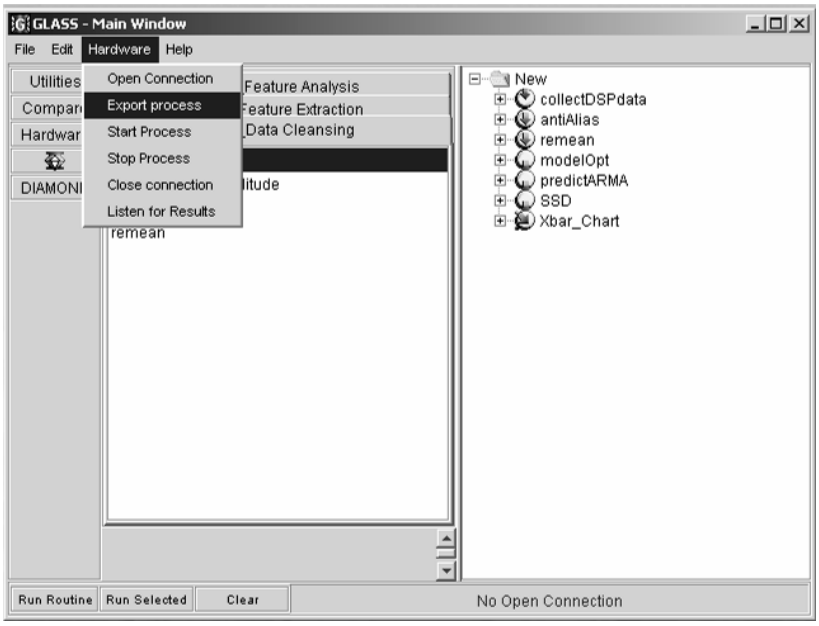

Figure 19 - Screen shot capturing the added Hardware drop down menu. The menu items displayed correspond to commands that can be sent to the node. Notice that confirmation or errors of the commands sent are displayed in the process bar at the bottom.

A "Hardware" menu (Figure 19) is added to facilitate GUI handling of operations such as opening a connection to the hardware, uploading a constructed process, starting and stopping the process remotely, and finally receiving results broadcasted over the network.

4.3.4 Communication and interaction

To facilitate the communication between the GLASS client and node, a simple command and response communications protocol is created. An example of an exchange between the client and node to establish a connection and start a process is similar to communication between people:

Client: Open_Connection
Node: +OK Connection Opened

Client: Send_Process
Node: +OK Send Process
Client: starts transfer of Process object
Node: +OK Process Received

Client: Start_Process
Node: +OK Process Started

Client: Stop_Process
Node: +OK Process Stopped

Client: Close_Connection
Node: +OK Goodbye

Notice that each step is a command from the client followed by a response from the node. Without a "+OK" response, the client will abort and ask the user if they would like to try again. This command and response is implemented to prevent deadlock between the two programs. Similarly, if the node receives a command that is not expected or the received object is not valid it will return a "-ERR" response notifying the client that there was a problem with the communication.

Each step in the above dialog is also tied to an action in the GLASS client hardware menu. To send the Open_Connection command, the user selects "Open Connection" in the Hardware menu. Thereby the user controls each step of the process allowing a connection to be established and possibly several versions of the process to be uploaded before the Start_Process command is sent. Confirmation of each command is displayed in the progress bar at the bottom of the screen (Figure 19).

All communication between the node and clients occurs on port 5244. The node opens a socket on port 5244 and constantly listens for a connection request. While multiple clients can access a single node, the protocol is setup to allow only a single client to connect at one time. This connection mode means that while connected, a client has dedicated and exclusive access to the hardware node.

4.3.5 Execution of a process

The purpose of the GLASS node software is to execute processes on remote machines. This is accomplished by uploading a process object created on a client to the hardware node. The hardware node must be running MATLAB and the GLASS node software. Once uploaded, the process is executed through the same mode as described in Chapter 3. In fact, the objects and method developed for the client were simply reused and wrapped in the command and response protocol described above.

Like the client, the process object only contains information on which functions to execute and how to execute them. The actual M-file, C file, or JAVA class must exist on the node or machine on which it is to be executed. Loading these files onto the node or remote hardware is typically achieved using an FTP client.

4.3.6 Hardware integration

To have data to run a process on, data must be collected from the DSP board. Because the DSP board is wrapped in TCP, communication can again be simplified to a basic command and response communication over a socket. This also allows the DSP board not only to be accessed from the SBC to which it is connected,

but also to be accessed from an outside GLASS client. This TCP socket communication takes place on port 5255.

New functions are created in JAVA to communicate to the board using the socket communication. The class collectDSPdata in the gov.lanl.esawr.glass.hardware package facilitates the interaction with the DSP board. The channel, sampling frequency, number of samples, and IP address of the node are designated as input variables. A vector of data is then returned after the all of the data are collected. This vector of data can then be passed on to other functions in DIAMOND II or other modules for processing.

4.3.7 Communication of results

Once the data are collected and processed a result needs to be returned to a client, mobile device, a display, a central monitoring device, or all of the above. Following the examples of flexibility above, the result is broadcast over a socket opened on port 5266. The difference is that while the above communications were limited to a dedicated connection between a client and the node, the result can be broadcast to multiple recipients simultaneously. Broadcasting of results to multiple recipients is possible using UDP and the JAVA multicasting functionality. This "multicasting" means that several clients, a server for storing the results, and a handheld device can all receive the broadcasted result at the same time and without each making a direct connection to the device.

Once a result is calculated, the node will broadcast the result. Any devices that are connected to the appropriate multicast group will receive the result. Results are typically in the form of 1, 0, or -1 representing the state as damaged, undecided, or undamaged respectively. It is up to the receiving program or device to interpret the result.

The GLASS client is modified to listen for such broadcast results and to display them as a change in the progress bar at the bottom of the screen. Another simple program can be run on the node or client that listens for a result and then records the result with a time stamp to a text file. A simple program to change the state of a green, yellow or red LED cluster, or to show a result on a pocket PC device could also be created.

4.4 Summary

The Husky system is comprised of both custom and off-the-shelf hardware components. What makes the project unique in the SHM community, however, is the ability to create, load, and run processes remotely through the GLASS software and the flexibility of the hardware coupled by TCP/IP.

The design of the system allows for the SHM problem to be broken into two specific steps, training and monitoring. In the first step, the flexibility of the software and hardware allow for baseline data to be downloaded from a remote site onto a more powerful development platform. On the development platform, different cleansing, normalization, feature extraction and statistical modeling techniques can be employed to find the optimal solution. Models and thresholds are developed and formed into a monitoring process from the baseline data.

By creating client and node interaction, the monitoring processes created in the GLASS client can be remotely executed using the GLASS node software. The GLASS node software can reside on another desktop or on specialized hardware as assembled herein by Motorola. The monitoring processes are easily transferred from the development client to the node using a drop down menu. Once a monitoring process is started, it runs continuously collecting data, processing the data, and returning a result until a command to stop is received from a client. The results are broadcast over the local area network and can be received by multiple clients simultaneously.

By creating software that allows dynamic interaction between a client and the hardware, the Husky project facilitates the training phase and overcomes the limitation of a static and limited SHM process.

4.5 Contributions

- Creation of the GLASS node software for remote execution of processes constructed using a GLASS client.
- Conceptualization of loosely coupled hardware using TCP and UDP Internet protocol sockets.
- Development JAVA classes for sampling from a DSP board over a TCP socket and returning data to the GLASS process for analysis.
- Creation of a framework for Multicasting results from a GLASS process to multiple recipients and multiple platforms.

5 Experimental Application

5.1 Introduction

As a demonstration of the Husky system a small structure that simulates real world joints is monitored in both an undamaged and damaged condition. Damage in this case is defined as a loss in preload in a bolted joint. Baseline data are collected and the structural health monitoring (SHM) process is constructed on a remote desktop running the GLASS client. The constructed process is then uploaded to the GLASS node and set to continuously monitor the structure. Results of the process are broadcast back to the client as well as recorded in a text file on the Husky node.

First the test structure, excitation source, and a method of introducing damage into the system are detailed. Next, the SHM process is developed to detect the introduced damage. Finally, the experimental results are presented and a summary of the experimental demonstration is presented.

5.2 Experimental setup

5.2.1 Test structure

The test structure consists of a small four-sided frame. Each of the corners is bolted using an angle iron bracket on the insides with ¼" 10-32 bolts connecting the sides with the angle iron. The four sides are constructed from 6063-Aluminum. The base measures 54 cm by 13 cm and is 0.6 cm thick. Each side measures 26 cm by 4.5 cm and is 0.3 cm thick. The top is 49 cm by 4.5 cm and is 0.3 cm thick. The top is tapped in the middle to thread in the shaker stinger. The stinger transfers excitation from a shaker to the structure.

The structure is secured to the workbench top using two C-clamps. This connection improves the repeatability of the tests. Each of the bolts connecting the sides is tightened to 11.3 N•m. preload. The structure is pictured in Figure 20 and the monitored joint is in the upper right corner of the figure.

Looking closer at the joint in Figure 21, there are two accelerometers located side by side. This location is to provide a comparison between the Husky systems data acquisition board and a commercial data acquisition system. The accelerometers are PCB, model 336C, and have a nominal sensitivity of 1000 mV/g. The accelerometers are mounted to the structure with wax.

As seen in Figure 22, the bolt on the monitored joint is equipped with a piezoelectric (PZT) stack actuator located between the structure and the bolt head. This stack actuator is a model HPST 1000/25-15/15 produced by Piezomechanik and has a maximum stroke of 15 μm for 1000V. The PZT actuator changes the bolt preload of a joint without disturbing the structure. By varying the input voltage to the actuator from –200 V to +1000 V, a 4 kN change in the bolt tension can be achieved. The PZT actuator allowed for modeling gradual degradation or deterioration of a structural system.

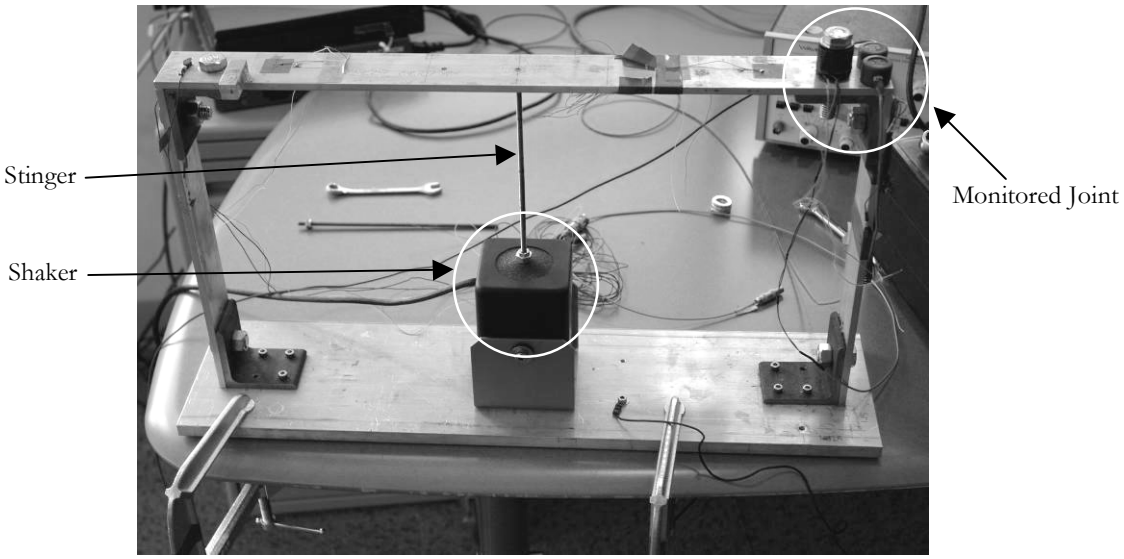

Figure 20 - Overall view of the experimental structure. The structure has bolted connections at the joints and is excited by a small shaker.

Figure 21 - Two accelerometers were placed side by side on the test structure. The closer accelerometer is attached to the Husky system, while the far accelerometer is attached to a commercial data acquisition system.

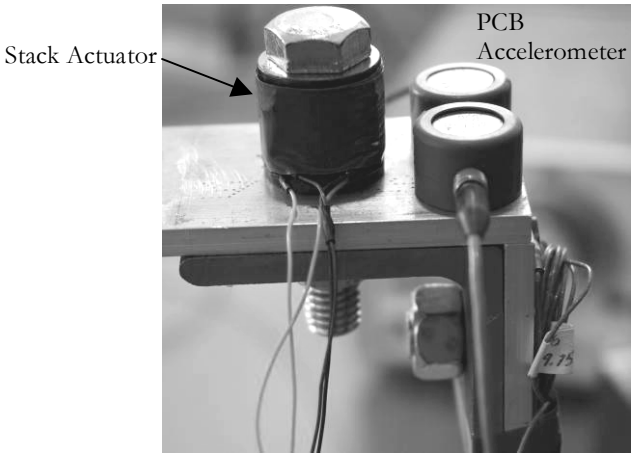

Figure 22 - The bolt holding the joint tight is equipped with a piezoceramic stack actuator. The actuator is used to apply and remove tension in the joint.

Figure 23 - The laptop based remote development platform.

Several other pieces of equipment are required in creating the undamaged to damaged actuation. First a Wavetek function generator produces a square wave. This wave is used to alternate a Piezomechanik high voltage power source from ~1000 V on the high end to ~ -200 V on the low end. This voltage change in turn actuates the stack actuator and either tightens or loosens the joint respectively. A Tektronix oscilloscope is also attached to monitor the signal. A photo of the addition equipment is shown in Figure 24.

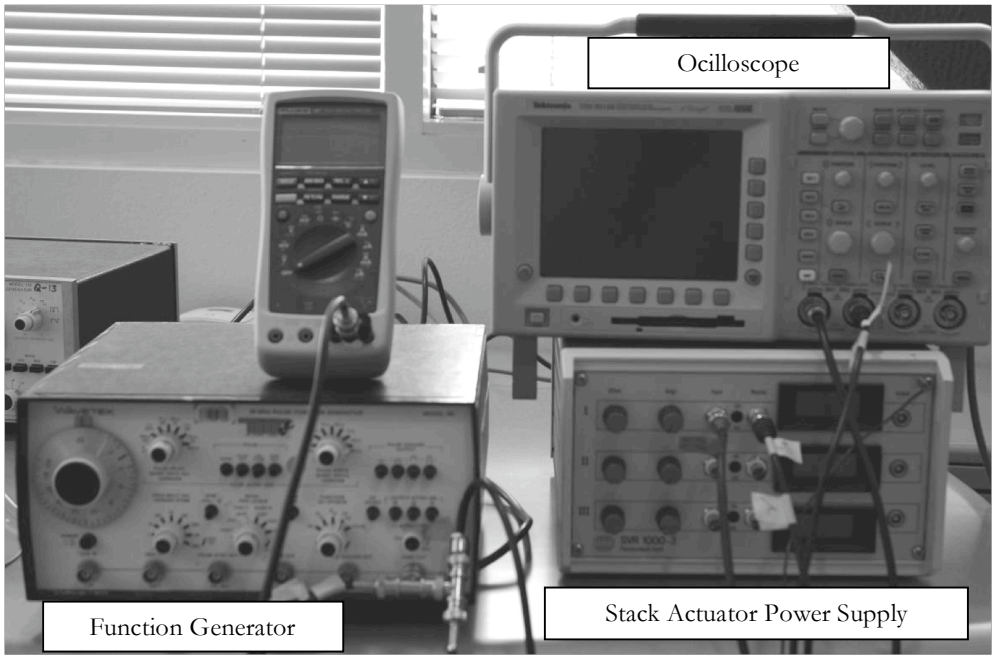

Figure 24 - The test equipment. The function generator provided a square wave to the stack actuator power supply. This square wave alternates the power from high to low, effectively actuating the joint from tight to loose. The Oscilloscope is used to monitor the power output. The multi-meter is used by a parallel experiment.

5.3 Benchmarking

Two methods of benchmarking the Husky system's ability to collect data accurately are implemented. First, a sine wave from a frequency generator is input to the first channel on the data acquisition board. Sine waves at different frequencies are recorded by the system. An example of these test results is shown in Figure 25 – A 80 kHz sampling, 8000 data points and 10 Hz sine wave. Both clipping at the top peak of the wave and small abnormalities are displayed in this figure. The clipping is caused by excess DC gain in the ICP power. The abnormalities were caused by small bits of electrical interference in part of the board. This was corrected..

Two problems are noted when reading in the sine wave. First, the tops of the peaks are clipped. By adjusting the DC offset, the center of the sine wave is lowered so the full range of the analog to digital converter (ADC) is used and the signal experienced no clipping. Secondly, at some points of the wine wave, the samples experience spikes. This problem was caused by electrical interference between wires on the circuit board. After appropriate shielding, this problem was eliminated.

Once the DSP board correctly samples a known electrical signal, an accelerometer is attached via a small signal-conditioning box. The box provides the ICP power for the accelerometer. The PCB accelerometer is attached to a 1g handheld shaker. The output of the shaker is sampled and compared to the known calibration of the accelerometer. When adjusted for gain, the system sampled a peak voltage of 996 mV, which when compared to 994.6 mV/g calibration, shows less than 1% error.

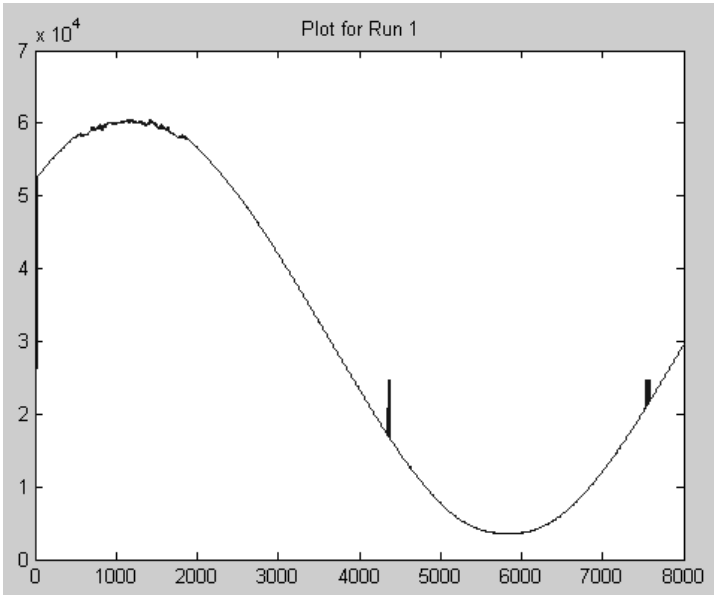

Figure 25 – A 80 kHz sampling, 8000 data points and 10 Hz sine wave. Both clipping at the top peak of the wave and small abnormalities are displayed in this figure. The clipping is caused by excess DC gain in the ICP power. The abnormalities were caused by small bits of electrical interference in part of the board. This was corrected.

These benchmarking tests allow troubleshooting of the hardware data collection and confirmation that the system collects data accurately. Once the system is shown to collect data accurately, a SHM test of the structure is performed.

5.4 Structural health analysis

There are two steps that must be taken to determine if a structure has changed. First, a baseline condition must be established. This process requires collection of baseline data, a damage sensitive feature be extracted, and that statistical modeling be applied to determine confidence intervals on the baseline data. The data is collected from the structure using the Husky hardware, but the baseline feature extraction and statistical modeling is performed on the development platform using the GLASS client.

Once a baseline model and confidence intervals have been established, the process can be uploaded to the GLASS server. The server then takes sample datum, applies the baseline model, extracts a feature and compares this feature against the baseline confidence intervals. If the feature is statistically similar to the baseline data the structure is determined to be unchanged, or undamaged. If the feature is statistically abnormal, then the structure is determined to have changed, or is indicative of damage.

The steps of the statistical pattern recognition paradigm described in chapter 2 are followed in defining the demonstration SHM problem presented herein. The data acquisition, feature extraction, and statistical modeling are present in both the baseline and testing phases, but are slightly different. The following sections outline each of the steps in the paradigm.

5.4.1 Operational evaluation

The test structure in question simulates a basic bolted or welded joint. In the case of a bolted joint it is desirable to detect a loss of preload that would cause a joint to loosen. It is also desirable to detect too much preload because excess stress on the bolt may cause premature failure. This joint can also be seen to abstract cracking in a welded joint. When a joint with a crack responds to a dynamic load the crack with open and close, much in the same way as a bolt that is loosing preload. Depending on the severity of preload loss and the external loading, damage may or may not affect the global response of the structure, but will most likely affect the local features. Therefore, a feature that is able to detect a loss of preload, or too much preload, in the bolt on a local level is desirable.

The Husky system can collect up to 8000 data points, which is more data than is required for this test. The current system also has no permanent way of storing data locally, but baseline modeling will be performed on a development system, therefore the Husky system will only be required to process a single data set at a time negating any need for permanent storage.

Because of the controlled nature of the laboratory, environmental and operational conditions are not considered in this test.

The purpose for monitoring such joints can be both economic and life safety related. For this particular application the purpose is to graduate, however, in a larger view detecting a loosening joint has safety and

economic implications that are pervasive across mechanical, civil, and aerospace fields. Detecting a bolt loosening in an assembly line, for instance, can prevent failure and down time by alerting a technician to this condition before failure occurs.

5.4.2 Data acquisition

Data is acquired by the Husky system through the Motorola digital signal processor (DSP) board attached to the analog PCB accelerometers. The raw acceleration time history is used in the SHM process. When deciding on the sampling frequency, several factors came under consideration. First, the damage is considered local and not global. Global effects are typically noted in the lower wavelengths where waves are longer. Local effects can be seen at higher frequencies because of the short wavelengths interacting with local damage. By sampling at a high frequency, effects these local effects are recorded with high resolution.

To determine a frequency at which to sample, a chirp signal is sent through the structure. The chirp signal ranges from 10 Hz to 10 kHz. No response in the structure is observed over 8 kHz and amplitude of response is decreased after 7.5 kHz. Therefore, a Nyquist frequency of 7.5 kHz is chosen.

A short time sample will allow the underlying assumptions of the time series ARMA modeling to hold true. These assumptions are that the signal is not changing in mean or variance. By taking a small slice of time, it is unlikely that any effects will be seen to affect the mean or variance. 1050 samples (0.07 seconds) are taken. The first 26 samples are discarded to cleanse the data of a small transient as the ADC switches on to sample. This leaves 1024 samples for analysis. The data acquisition process is shown in Figure 26.

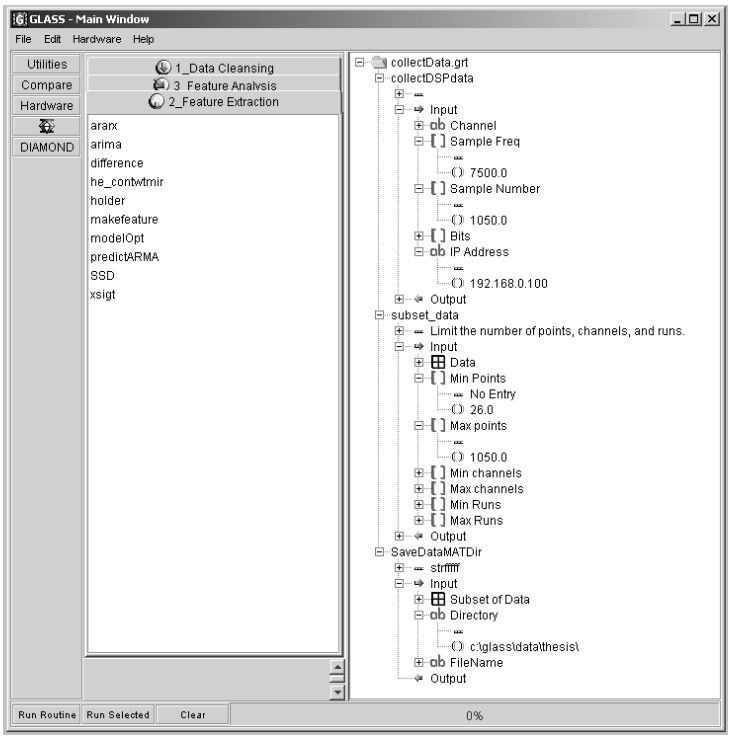

Figure 26 - GLASS client window showing the data collection process used to collect baseline data from the test structure.

5.4.3 Training feature extraction

Once an acceleration time history is collected, an auto-regressive moving-average (ARMA) model is applied. The Akakie information criteria (AIC) from chapter 2 are used to determine the optimal order of ARMA model. The model optimization function in DIAMOND II searched for the optimal model with the number of AR and MA coefficients varying between one and ten. In this test, the optimal ARMA model is found to be AR=2 and MA=3 for a total of five coefficients. The process can be seen in Figure 27.

Once the model is established, 60 more training condition acceleration time histories are recorded. The baseline model is then applied to each of the baseline histories to attempt to predict each point. The sum of the squared difference between the true history and the predicted history becomes a feature. In theory, a model based on a training structural state should predict other data sets from the same structural state. However, if damage is introduced into the system, the model will fail to predict the histories and will result in a large error.

The 60 data sets provide a distribution of the sum of squared error (SSE) feature that can be used to set confidence intervals. The SSE feature is of dimension one.

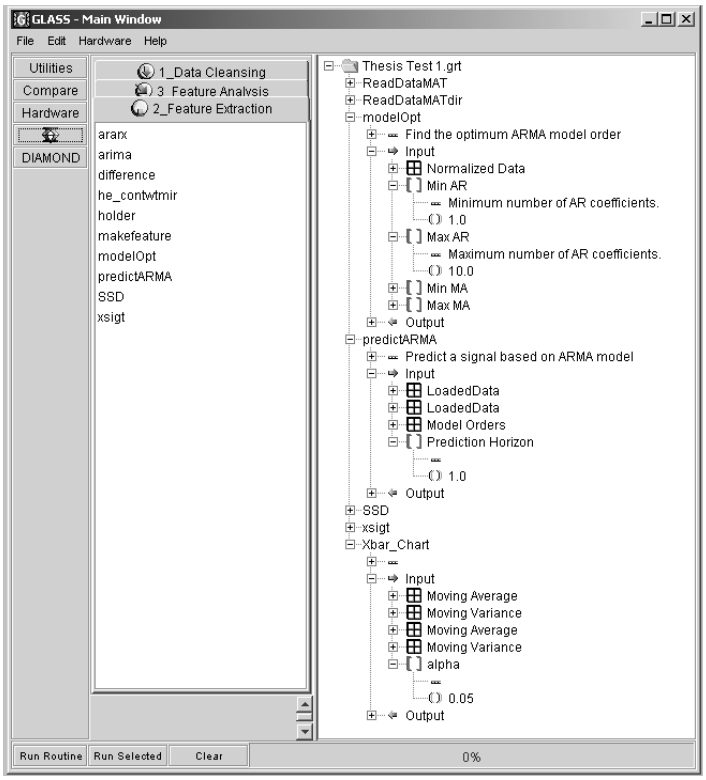

Figure 27 - GLASS client showing the process used for feature extraction and statistical modeling of the baseline data sets.

5.4.4 Training statistical modeling

For statistical modeling, control charts, specifically X-bar charts, are designed using a moving mean of six calculations the SSE. A 95% confidence interval is determined using extreme value statistics on the maximum and minimum 5%, totaling 10% of the data, as described in chapter 2. A Gumbel distribution is used to model the tails of this distribution.

5.4.5 Testing process

Once the baseline model and thresholds are established, the testing process can be implemented. Data are collected using the same test parameters as those used in collecting the baseline data. This data set is stored in memory on the Husky system instead of being uploaded to the development system.

Once the data are collected, the optimal ARMA model from the baseline data is applied to predict each data point. The SSE feature is obtained after the prediction. Again, this feature extraction is performed at each individual Husky node, not on a central server.

Once the feature for a data set is extracted, it is analyzed with the baseline statistical model. The feature will either lie within the confidence intervals, indicating the structure is unchanged from the baseline, or the feature will lie outside of the intervals, indicating a change in the structure. Because the feature that is selected is sensitive to loosening of the joint, the structural change is assumed to be from the joint loosening. The entire process uploaded to the Husky node is shown in Figure 28.

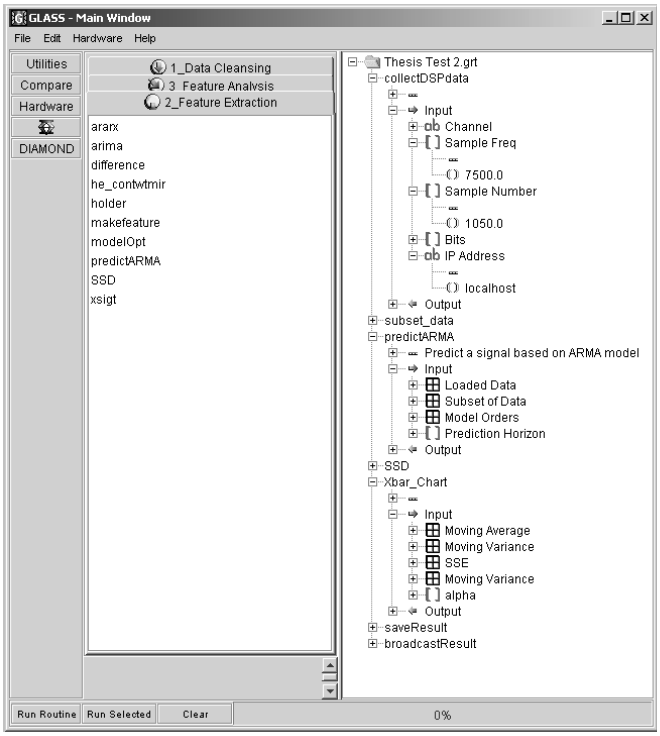

Figure 28 - GLASS Client displaying the process uploaded to the hardware to monitor structural health. The model and control chart parameters based on the baseline data are contained in the function variables.

Once the statistical assessment is complete and the feature is determined to lie inside or outside of the confidence intervals, the result is saved to a file on the node for later retrieval. The result is also broadcast back to the client software.

5.5 System Performance

The laptop and a single node are connected on a Local Area Network (LAN). The laptop is used as a development platform, acquiring a set of 60 baseline data sets of 1050 data points over the LAN from the node. Each data set is collected and saved on the laptop in an average of seven seconds.

Once the data are collected, the baseline feature extraction and statistical modeling process is run on the development platform. In this test, the optimal ARMA model is found to be AR=2 and MA=3 for a total of five coefficients, which is reasonable. The statistical thresholds are determined by a 95% confidence interval placed on a moving mean of the extracted SSE feature. The moving mean window is 4 data points wide. This process takes approximately 2 minutes on a standard laptop to run with the 60 data sets.

The testing process is constructed using the optimal ARMA model and the 95% confidence interval. The process is uploaded to the node and set to run. Each cycle of the process run on the node takes eight seconds from data collection to the client receiving the result. The testing process was run for eight minutes in the undamaged condition with the stack actuator at 1000 V. During this period of time, false positive indications consistent with the 95% confidence intervals are noted.

The stack actuator is then reduced to –250 V to simulate the damage. The process on the node is run continuously while making the adjustment. The process correctly identifies the structural state as being changed to a damaged condition after a short lag. This lag is because the change in structural state takes place during processing the previous result. Results from the test are shown in Figure 29.

During the test, the Husky system rebooted when high voltage was returned to the stack actuator for the first time. The cause for this reboot is unknown. The Husky system, however, performed well under the loss of power. The process restarted and began to immediately classify the state of the structure correctly. Subsequent changes in the stack actuator did not cause the system to reboot.

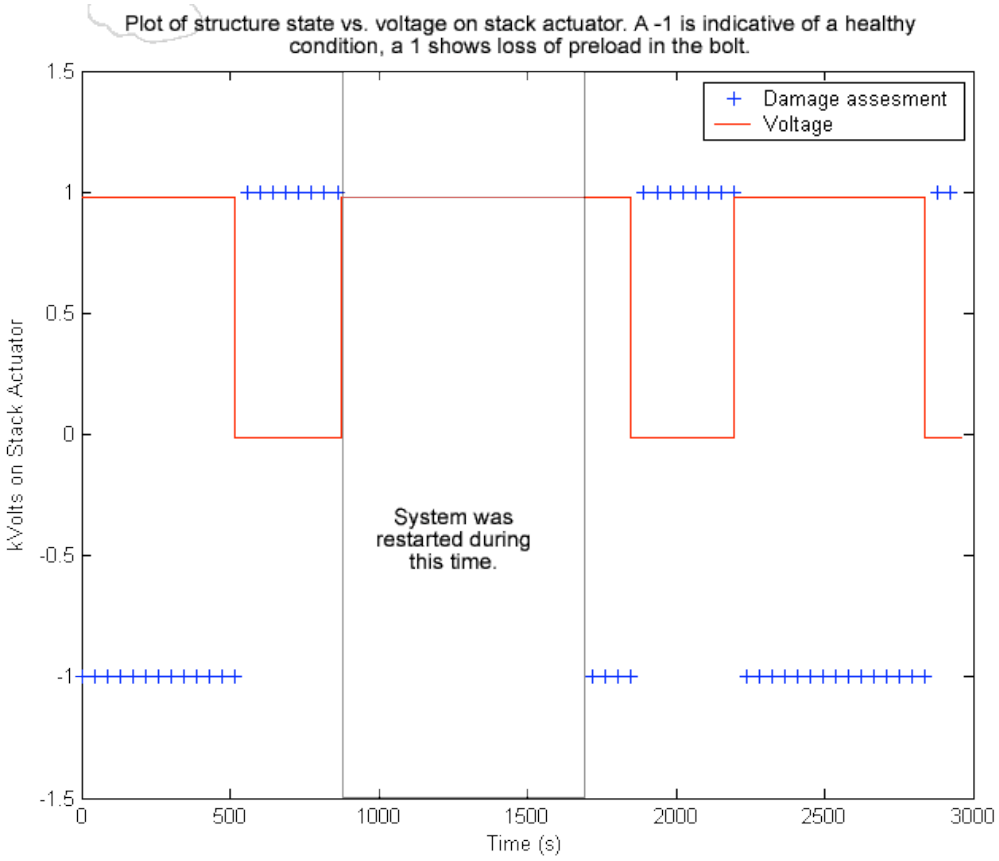

Figure 29 - A plot of the test results showing the structure in both undamaged and damaged states. The solid line is a plot of the voltage. The blue crosses represent the results received by the client from the node process. It can be seen that there is a 1 or 2-cycle lag in the damage assessment because of processing time. A high voltage should correspond to a –1 result, while a low voltage should correspond to a +1 result.

5.6 Summary

In this demonstration, the GLASS client, server and integrated Husky system met all of the design goals. Data collection, baseline analysis, and testing processes are easily assembled in the GLASS client from catalogs of encapsulated data interrogation functions. These processes then interface with a hardware system to collect and analyze baseline data. A final testing process can then be assembled and dynamically loaded onto the hardware node.

The processes created are capable of detecting changes in the test structure that correlated to a loss of preload in a bolted joint. The system also accurately assessed when the structure returned to baseline operating conditions.

5.7 Contributions

- Completed a successful demonstration using the GLASS client to graphicaly assemble a SHM processes.

- Demonstrated successfully the embedding of SHM processes into the Husky hardware from a remote development platform.
- Demonstrated a near real-time SHM process that correctly classified changes in a structure related to loss of preload in a bolted connection.

6 Summary

This study involved:

1. The collection and creation of data interrogation functions.
2. The development of a software package for assembling a Structural Health Monitoring (SHM) process from the collected functions.
3. The development of node software and hardware packages for remotely running processes.

Together, these solutions provide a general and flexible framework for developing future SHM processes.

In an effort to develop an automated and quantitative method for damage identification, the statistical pattern recognition paradigm is followed. A unique integration of data normalization and cleansing, time series analysis for feature extraction, and statistical discrimination that incorporates extreme value statistics (EVS) is undertaken within the framework of the DIAMOND II module. Time series analysis techniques, solely based on the observed vibration signals, are first automated and deployed to extract damage sensitive features from a structure. In this study, a control chart is employed to provide a more automated statistical tool for this decision-making procedure, excluding unnecessary interpretation of the observed feature by users. Finally, the performance and robustness of damage classification is improved by incorporating extreme values statistics of the extracted features into the control chart. This framework is well suited for a continuous monitoring system.

Using the GLASS Client allows users to categorize, share, and re-use SHM data interrogation functions. Previously every researcher had a version (or versions) of a commonly performed function. Now a single version of the function can be re-used and shared with others. Functions were also previously pasted together in large and cumbersome master functions that executed a particular process. Integrating a new function often required a lot of cut and paste as well as checking indexing, variable names, and general continuity. GLASS allows assembly of functions into a SHM process by an intuitive drag and drop procedure, similar to moving files around in any modern operating system. Functions written by diverse people in various languages for entirely different applications can also be easily incorporated into a new process.

GLASS Technology will enable researchers to develop and share functions in a common platform to quickly prototype new SHM processes. Upon assembly, these processes can be shared for review, placed into a hardware environment for testing, or compared with other processes.

The Husky system is comprised of both custom and off-the-shelf components. What makes the project unique in the SHM community, however, is the ability to create, load, and run processes remotely through the GLASS node software.

By creating client and node interaction, processes created in the GLASS client can be remotely executed using the GLASS node. The GLASS node can reside on another desktop or on specialized hardware as assembled herein by Motorola. The processes are easily transferred from the client to the node using a drop down menu. Once a process is started it runs continuously to collect data, process the data, and return a result until a stop command is

received from a client. The results are broadcast over the local area network and can be received by multiple clients simultaneously.

By creating software that allows dynamic interaction between a client and the hardware, the Husky project overcomes the limitation of non-adaptation that other embedded SHM systems face.

In this demonstration, the GLASS client, node and integrated Husky system met all of the design goals. Data collection, baseline analysis, and testing processes are easily assembled in the GLASS client from catalogs of encapsulated data interrogation functions. These processes then interface with an integrated hardware system to collect and analyze baseline data. A final testing process is then assembled and dynamically loaded onto the hardware node. The processes are shown capable of detecting changes in the test structure that correlated to a loss of preload in a bolted joint. The system is also accurate in assessing the structure as undamaged when the bolt is returned to the baseline preload.

6.1 Contributions:

- The automation of ARMA order selection for feature extraction using the AIC.
- The use of the Sum of Squared Error damage sensitive feature.
- Development of a numerical proof of concept for EVS and SPRT in statistical modeling for SHM.
- Development of an innovative connection between the JAVA programming language and the MATLAB computational environment.
- The creation of a GUI client interface for rapid prototyping of new SHM processes.
- The collection and modularization of statistical pattern recognition tools.
- GLASS node software development for remote execution of constructed processes.
- Conceptualization of loosely coupled hardware using TCP and UDP Internet protocol sockets.
- JAVA classes for sampling from a DSP board over a TCP socket and returning data to a GLASS process for analysis.
- Facilitation of multicasting results from a GLASS process to multiple recipients and multiple platforms.
- Demonstration of graphical assembly of SHM processes using the GLASS client.
- Demonstration of embedding SHM processes into integrated hardware from a remote client.
- Demonstration of near real-time SHM process correctly classifying changes in a structure related to damage in a bolted connection.

References

1. Ayres, J.W., C. Rogers, Z. Chaudhry. "Qualitative Health Monitoring of a Steel Bridge Joint via Piezoelectric Actuator/Sensor Patches.", 1996.
2. Adams, D.R and Farrar, C.R. "Application of Frequency Domain ARX Features for Linear and Nonlinear Structural Damage Identification," *Proceedings of SPIE's 9th Annual International Symposium on Smart Structures and Materials,* San Diego, CA, March 17-21, 2002.
3. Akaike, H., "A new look at the statistical model identification", IEEE Transactions on Automatic Control, AC-19(6), pp. 716-723, 1974.
4. Akaike, H., "Information measures and model selection", Bulletin of the International Statistical Institute, Vol. 50, pp. 277-290, 1983.
5. Allen, D.W., Castillo, S., Cundy, A.L., Farrar, C.R., McMurray, R.E., "Damage Detection in Building Joints by Statistical Analysis", Proc. of IMAC, Orlando, 2001.
6. Allen, D.W., Limback, N., Los Alamos National Laboratory: Structural Health Monitoring Website, www.lanl.gov/damage_id, January, 2003.
7. Allen, D.W., Sohn, H., Worden, K., Farrar, C.R., "Utilizing the Sequential Probability Ratio Test for Building Joint Monitoring", Proc of SPIE Smart Structures Conference, San Diego, March 2002.
8. Ang, A.H-S. and Tang, W.H. *Probability Concepts in Engineering Planning and Design*, John Wiley & Sons, Inc., New York, NY, 1975.
9. Benjamin J.R. & Cornell C.A., *Probability, Statistics and Decision for Civil Engineers*, McGraw-Hill, Inc. New York, NY, 1970.
10. Box, G. E., Jenkins, G. M., and Reinsel, G. C., Time Series Analysis: Forecasting and Control, Third Edition, Prentice-Hall, Inc., NJ, 1994.
11. Box, G. E., Jenkins, G. M., and Reinsel, G. C. *Time Series Analysis: Forecasting and Control*, Third Edition, Prentice-Hall, Inc., NJ, 1994.
12. Castillo, E. (1987) *Extreme Value Theory in Engineering*, Academic Press, Inc., San Diego, CA.
13. Chaudhry, Z., T. Joseph, F. Sun, and C. Rogers, "Local-area health monitoring of aircraft via piezoelectric actuator/sensor patches." SPIE Vol. 2443. 1996
14. Crawley, E.F., and E.H. Anderson, "Detailed Models of Piezoceramic Actuation of Beams." Journal of Intelligent Material Systems and Structures. Vol. 1, pp. 4-25. 1990.
15. Doebling, S.W., Farrar, C.R., and Cornwell, P.J., "DIAMOND: A Graphical User Interface Toolbox for Comparative Modal Analysis and Damage Identification," in Proc. of Sixth International Conference on Recent Advances in Structural Dynamics, pp. 399-412. Southampton, UK, July 1997.
16. Doebling, S. W., Farrar, C. R., Prime, M. B., and Shevitz, D. W. "A Review of Damage Identification Methods that Examine Changes in Dynamic Properties," *Shock and Vibration Digest* 30 (2), pp. 91-105. 1998.
17. Embrechts, P., Kluppelberg, C. and Mikosch, T. *Modeling Extremal Events*, Springer-Verlag, New York, NY. 1997.
18. Farrar, C. R. and S. Doebling, "The State of the Art in Vibration-Based Structural Damage Identification, A Short Course." Los Alamos Dynamics, Ltd. 2000.
19. Fisher, R.A. and Tippett, L.H.C. "Limiting Forms of the Frequency Distributions of the Largest or Smallest Members of a Sample", *Proceedings of the Cambridge Philosophical Society,* 24, pp. 180-190, 1928.
20. Galambos, J. *The Asymptotic Theory of Extreme Order Statistics*, John Wiley and Sons, New York, NY. 1978.
21. Gamma, E., Helm, R., Johnson, R., Vlissides, J. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley. 1995

22. Ghosh, B.K. *Sequential Tests of Statistical Hypotheses*, Addison-Wesley, Menlo Park, CA. 1970.

23. Gross, K.C. and Humenik, K.E. "Sequential Probability Ratio Tests for Nuclear Plant Component Surveillance," *Nuclear Technology*, Vol. 93, pp. 131-137. 1991.

24. Gumbel, E.J. *Statistics of Extremes*, Columbia University Press, New York, NY. 1958.

25. Haykin, S. Neural Networks: A Comprehensive Foundation, Prentice Hall, 1998.

26. Hicks, C.R., Holt, Rinehart and Winston. Fundamental Concepts in the Design of Experiments, New York, 1973.

27. Humenik, K.E. and Gross, K.C. "Sequential Probability Ratio Tests for Reactor Signal Validation and Sensor Surveillance Applications," *Nuclear Science and Engineering*, Vol. 105, pp. 383-390. 1990.

28. Kotz, S. and Nadarajah, S. *Extreme Value Distributions: Theory and Applications,* Imperial College Press, London, UK. 2000.

29. Larman, C., *Applying UML and Patterns*, Prentice Hall, New Jersey, USA, 2002.

30. Ljung, L. *System Identifications: Theory for the User*, Prentice Hall, Englewood Cliffs, NJ. 1987.

31. Matlab, *The Language of Technical Computing*, The Math Works Inc., 1998.

32. Meyer, Bertrand. *Object Oriented Software Construction, Second Edition*, Prentice Hall, New Jersey, USA, 1997

33. Montgomery, D. C., *Introduction to Statistical Quality Contro*l, John Wiley & Sons, Inc., New York, 1996.

34. Moster, Paul C. *Gear fault detection and classification using learning machines*, Sound and Vibration MFPT show issue, pp. 22-27. March, 2004.

35. Park, G., H.H. Cudney, D.J. Inman. "An Integrated Health Monitoring Technique using Structural Impedance Sensors".

36. Peairs, D. M., Park, G. and Inman, D. J., 2002, "Low Cost Impedance Monitoring Using Smart Materials", Proceedings of the 1st European Workshop on Structural Health Monitoring, Paris, France, pp. 442-449, July 9-12, 2002.

37. Pickands III J., *Statistical inference using extreme order statistics*, Annals of Statistics 3 pp.119-131, 1975.

38. Reiss R.-D. & Thomas M., *Statistical Analysis of Extreme Values with Applications to Insurance, Finance, Hydrology and Other Fields*, Birkhauser Verlag, 2001.

39. Roberts S., *Novelty detection using extreme value statistics*, IEEE Proceedings in Vision, Image and Signal Processing 146 pp.124-129. 1998.

40. Roberts S., *Extreme value statistics for novelty detection in biomedical signal processing*, IEEE Proceedings in Science, Technology and Measurement 147 pp.363-367, 2000

41. Schwarz, G, "Estimating the dimension of a model", The annals of Statistics, Vol. 6, pp. 461-464, 1978.

42. Sohn, H, D. W. Allen, K. Worden, C. R. Farrar "Statistical Damage Classification using Sequential Probability Ratio Tests." International Journal of Structural Health Monitoring. Vol. 2, March. 2003

43. Sohn, H. and Farrar, C.R. "Damage Diagnosis Using Time Series Analysis of Vibration Signals," *Journal of Smart Materials and Structures*, 10, pp. 446-451. 2001.

44. Sohn, H., Farrar, C.R., Hunter, N.F., and Worden, K. "Structural Health Monitoring Using Statistical Pattern Recognition Techniques," *ASME Journal of Dynamic Systems, Measurement and Control: Special Issue on Identification of Mechanical Systems,* 123(4), pp. 706-711. 2001.

45. Sohn, H, C. R. Farrar, N. F. Hunter and K. Worden "Applying the LANL Statistical Pattern Recognition Paradigm for Structural Health Monitoring to Data from a Surface-Effect Fast Patrol Boat," Los Alamos National Laboratory report, 2000.

46. Sun, F., Z. Chaudhry, C. Liang, and C. Rogers. "Truss Structure Integrity Identification using PZT Sensor-Actuator." Journal of Intelligent Material Systems and Structures. Vol. 6. 1995.

47. Wald, A. *Sequential Analysis*, John Wiley and Sons, New York, NY. 1947.

49. Wirsching, H., Paez, T. L., Ortiz, K. *Random Vibrations Theory and Practice*, John Wiley, New York, NY. 1995

50. Webb. Peter. "MATLAB Programming Patterns: Integrating JAVA Components into MATLAB", MATLAB News & Notes, February 2002.

51. Worden, K., Allen, D. W., Sohn, H., and Farrar, C.R. "Damage Detection in Mechanical Structures using Extreme Value Statistics," *Proceedings of SPIE's 9th Annual International Symposium on Smart Structures and Materials,* San Diego, CA, March 17-21. 2002.

52. Worden K., Manson G. & Fieller N.J., *Damage detection using outlier analysis*, Journal of Sound and Vibration 229 pp.647-667. 2000