5-5-2012

Computer-aided Engineering Tools for Structural Health Monitoring under Operational Conditions

Priscilla O. Mensah-Bonsu
University of Connecticut - Storrs, pom10002@uconn.edu

Recommended Citation
https://opencommons.uconn.edu/gs_theses/278
APPROVAL PAGE

Master of Science Thesis

Computer-aided Engineering Tools for Structural Health Monitoring under Operational Conditions

Presented by
Priscilla Otaa Mensah-Bonsu, B.S/B.A.

Major Advisor

Shinae Jang

Associate Advisor

Richard Christenson

Associate Advisor

Ramesh Malla

University of Connecticut
2012
Abstract

Civil infrastructure such as bridges and pipelines play a central role in a modern economy. As many of these structures age without the resources to replace them, computer-aided non-destructive evaluation and monitoring has become a complement to field inspections as a maintenance approach. Civil structures, unlike mechanical systems, often require monitoring under operational conditions due to the adverse impact of infrastructure closures on economic activities and public safety. As such, non-destructive and autonomous monitoring systems are critical if timely assessment of structural condition is needed. This thesis presents two different applications of computer-aided non-destructive approaches for structures monitored under operational conditions. The first part of the thesis is the development of a graphical user interface (GUI) software for real-time autonomous assessment of structural health condition based on the signature vibration properties of the structure under study. The program, called ConImote2, operates on the wireless sensor network platform called imote2. ConImote2 is created in the Matlab platform for accessibility and to enable extension of its capabilities by the user. The primary goal of creating the program is to overcome the issue of data inundation from SHM systems by developing an autonomous data processing routine for instantaneous feedback on structural health conditions. Lab-scale validations of the program were used to fix bugs and provide important metrics about the sensitivity of the underlining algorithm to real changes in the structure. The second part of the thesis presents a new approach for optimum model selection during vibration-based finite element model updating of civil structures. The goal of this approach is to provide an evidence-based approach to model selection to ensure physical meaning in the non-unique optimum solutions obtained from a numerical optimization process. An algorithm is developed to rank the optimum solutions according to their physical plausibility. The algorithm uses data from static behavior of the structure to decouple the
ranking algorithm from the vibration-based optimization algorithm. The approach is
demonstrated on an in-service highway bridge instrumented with a sparse array of different
sensors.
Acknowledgements

It has been a wonderfully rich experience to study at this great university. I would like to thank my friend, Michael, for introducing me to this university. He has also been the single constant in my life during my time here and I am highly indebted to him for his friendship and real support. My family has also been supportive and patient with me. I am grateful for that.

I would like to thank my major advisor, Dr. Shinae Jang for creating the research and providing me the support for this research. The financial supports for this research from the University of Connecticut Foundation, and Civil Engineering Department are greatly appreciated. I am highly indebted to my associate advisors Dr. Richard Christenson and Dr. Ramesh Malla for inviting me to pursue research in this graduate program and for serving on my thesis committee. Dr. Malla has been extremely supportive by serving as an informal advisor during my time in the graduate program. I am also grateful to Dr. Michael Accorsi for his technical support to me on finite element modeling and his mentorship.

I would like to thank my laboratory colleagues, Sushil Dahal, Jingcheng Li, Raymond Chan, Netania Durogene, Zhaoshuo Jiang for their support and friendship.

Finally, my experience in graduate school has been enriched by serving as a Teaching Assistant during the entire period. I have made many lifetime friends and deepened my understanding of the fundamentals of structural behavior through this process. For this too, I am highly indebted to this fine graduate program.
Table of Contents

1 Chapter 1 Overview of thesis ........................................................................................................10

2 Chapter 2 GUI for continuous wireless monitoring ............................................................. 12
  2.1. Introduction .................................................................................................................. 12
  2.1.1. Hardware and software background ..................................................................... 18
  2.3. ConImote2 : GUI for continuous monitoring on the imote2 ................................... 21
     2.3.1. Remote Sensing functionality form UIUC ISHMP Tool suite ......................... 25
     2.3.2. Criteria for normal behavior ............................................................................ 27
     2.3.3. Outputs ............................................................................................................. 32
     2.3.4. Post-processor for structural condition assessment ......................................... 36
     2.3.5. Utilities ............................................................................................................. 37
  2.4. Validation .................................................................................................................... 39
     2.4.1. SDOF building with mass perturbation ............................................................ 39
     2.4.2. MDOF bridge with loose bolt .......................................................................... 48
  2.5. Troubleshooting ......................................................................................................... 65
  2.6. Future research .......................................................................................................... 68
  2.7. Overview of steps for running the ConImote2 ............................................................ 70

3 Chapter 3 Multi-scale FE updating of skewed in-service highway bridge ..................... 72
  3.1. Introduction ............................................................................................................... 72
  3.2. Test bed description .................................................................................................. 77
  3.3. Statistical system identification .................................................................................. 81
  3.4. Initial finite element model ....................................................................................... 86
     3.4.1. Geometry modeling ......................................................................................... 86
     3.4.2. Parametric modeling ....................................................................................... 87
  3.5. Multi-scale data approach for model updating ........................................................... 91
     3.5.1. Theoretical background of optimization ......................................................... 91
     3.5.2. Model updating Parameters for MOGA ......................................................... 93
     3.5.3. Design of Experiments ................................................................................... 98
     3.5.4. Sensitivity-based model updating results ....................................................... 98
     3.5.5. Ranking algorithm .......................................................................................... 100
Table of Figures

Figure 1 On-line modal frequency monitoring schematic diagram (NI,2012).................................15
Figure 2 On-line tracked resonance frequencies of Donghai Bridge (NI,2012) .....................15
Figure 3 On-line tracked resonance frequencies of Donghai Bridge (NI,2012) .....................16
Figure 4 Graphical User Interface for Guided Ultrasonic Waves (NI,2012).................................16
Figure 5 Corrosion monitoring system GUI from DOD (NI,2012)......................................................17
Figure 6 Schematic of ConImote2 ............................................................................................................23
Figure 7 ConImote2 Layout...................................................................................................................24
Figure 8 Criteria for normal natural frequencies and mode shapes ...................................................28
Figure 9 Four-storey building showing correct ConImote2 mode shape configuration ........29
Figure 10 Six-storey building showing correct ConImote2 baseline mode shape setting ..........31
Figure 11 Health status panel a) before starting b) after a run ..............................................................33
Figure 12 Key codes for overall health status ......................................................................................33
Figure 13 Key codes for health status of individual modes .................................................................34
Figure 14 Stop push button for the ConImote2 ..................................................................................37
Figure 15 File clean-up tool ..................................................................................................................37
Figure 16 Toolbars in ConImote2 .........................................................................................................38
Figure 17 Data selection toolbar used to select a data point in Frequency history plot ..............38
Figure 18 a) Single story building with active mass damper attached b) Single-story with accelerometer ........................................................................................................................................................................39
Figure 19 Test bed for test on lab-scale building .................................................................................42
Figure 20 Two imote2 sensors on the roof of lab-scale building ..................................................43
Figure 21 DAQ for ConImote2 on the lab-scale building .................................................................43
Figure 22 Zoom in of imote2 connection on the roof of lab-scale building ........................................ 44
Figure 23 Arrangement of standard mass on the roof of lab-scale building ........................................ 44
Figure 24 Variation of natural frequency with changes in Mass .......................................................... 46
Figure 25 Historical plot of natural frequency for single storey building ........................................... 47
Figure 26 Lab-scale truss bridge ........................................................................................................ 48
Figure 27 Set-up for lab-scale truss tests .......................................................................................... 49
Figure 28 Bracket for imote2 on a gusset plate ................................................................................ 50
Figure 29 Input force application system using LDS shaker ............................................................. 50
Figure 30 Power spectral density of vertical acceleration using wired sensor (Sensor 6) ................. 52
Figure 31 Power spectral density of vertical accelerations using 11 wired sensors ......................... 52
Figure 32 Mode shape for f = 19Hz for wired sensor ....................................................................... 53
Figure 33 Mode shape for f = 39Hz for wired sensor ....................................................................... 53
Figure 34 Mode shape for f = 70Hz for wired sensor ....................................................................... 54
Figure 35 ConImote2 baseline test showing high fidelity CPSD plot ............................................. 56
Figure 36 Damage Case #1 showing thinner ‘damaged’ element replacing top chord element ... 58
Figure 37 ConImote2 frequency domain plots a) before completely loosened bolts at joint along with damage element in top chord shows clear frequency drops in higher modes .......................... 61
Figure 38b. Variation of resonant frequencies as bolts are progressively loosened on the lab-scale truss bridge ................................................................................................................ 64
Figure 39 Data port initialization error .............................................................................................. 67
Figure 40 Flyover Bridge in Hartford, CT (Google Earth®). .............................................................. 77
Figure 41 Schematic of accelerometers and tilt meters (DeWolf, 2009). ........................................... 79
Figure 42 Schematic of temperature sensors (DeWolf, 2009). ........................................................ 79
Figure 43 Tilt meter .................................................................................................................. 80
Figure 44 Accelerometer ........................................................................................................... 80
Figure 45 Temperature sensor ................................................................................................... 81
Figure 46 Sample acceleration at all the accelerometers (mg) vs. time (seconds). ...................... 82
Figure 47 Sample frequency spectrum at all the accelerometers vs frequency (Hz). ................. 83
Figure 48 Operational deflection shapes. ................................................................................... 85
Figure 49 Workflow for Flyover 3D model drawing .................................................................. 86
Figure 50 ANSYS® Model of Spans 4, 5 and 6 ....................................................................... 87
Figure 51 Bending modes from the initial FE model used for updating ..................................... 90
Figure 52 Tilt profile of the FE model under service load .......................................................... 91
Figure 53 Sensitivity of frequency to concrete properties (%). ................................................... 95
Figure 54 Local Sensitivity (x10^{-5}) of frequency to BCs ......................................................... 95
Figure 55 Trend of tilt (degree) over 4 years (DeWolf, 2009) ..................................................... 97
Figure 56 Trend of diurnal temperature-induced tilt (Virkler, 2004) ......................................... 97
Figure 57 Flow chart of ranking algorithm. ............................................................................. 101
Figure 58 Comparison of natural frequencies (Hz). ................................................................. 103
Figure 59 Torsion in pier caused by shear forces at bearings. ..................................................... 105
Chapter 1 Overview of thesis

Civil infrastructure such as bridges and pipelines play a central role in a modern economy. As many of these structures age without the resources to replace them, computer-aided non-destructive evaluation and monitoring has become a complement to field inspections as a maintenance approach. Civil structures, unlike mechanical systems, often require monitoring under operational conditions due to the adverse impact of infrastructure closures on economic activities and public safety. As such, non-destructive and autonomous monitoring systems are critical if timely assessment of structural condition is needed. This thesis presents two different applications of computer-aided non-destructive approaches for structures monitored under operational conditions. The first part of the thesis is the development of a graphical user interface (GUI) software for real-time autonomous assessment of structural health condition based on the signature vibration properties of the structure under study. The program, called ConImote2, operates on the wireless sensor network platform called imote2. ConImote2 is created in the Matlab platform for accessibility and to enable extension of its capabilities by the user. The primary goal of creating the program is to overcome the issue of data inundation from SHM systems by developing an autonomous data processing routine for instantaneous feedback on structural health conditions. Lab-scale validations of the program were used to fix bugs and provide important metrics about the sensitivity of the underlining algorithm to real changes in the structure. The second part of the thesis presents a new approach for optimum model selection during vibration-based finite element model updating of civil structures. The goal of this approach is to provide an evidence-based approach to model selection to ensure physical meaning in the non-unique optimum solutions obtained from a numerical optimization process. An algorithm is developed to rank the optimum solutions according to their physical plausibility.
The algorithm uses data from static behavior of the structure to decouple the ranking algorithm from the vibration-based optimization algorithm. The approach is demonstrated on an in-service highway bridge instrumented with a sparse array of different sensors.

The Thesis is divided in four chapters. Chapter 1 provides an overview and outline of the main body of the thesis. Chapter 2 presents a review of the development of a GUI for monitoring civil structures. Chapter 3 presents a review of the application of CAE for finite element updating of an in-service highway bridge using multiple but diverse sensors. Finally, Chapter 4 presents a summary of the thesis.
Chapter 2 GUI for continuous wireless monitoring

2.1. Introduction
Civil infrastructure plays a central role in the economic, political, cultural advancement and dynamism of a modern society. The critical need for the safety and reliability of civil infrastructure requires that their design must incorporate a high factor of safety in order to ensure long-term use with minimal interruption. The need for long-term usability, however, leads to ageing structures which can fail catastrophically, as in the case of the Minneapolis interstate highway bridge (I-35W) in 2007. Conversely, new constructions are increasingly designed under optimized engineering performance. As such, construction errors can also lead to catastrophic failures. In the United States, more than 60% of interstate highway bridges in use were built in the 1950s and 1960s. Engineers, policy makers and economists have been debating the sustainable options for maintaining civil infrastructure in the United States.

Structural health monitoring (SHM) is a technology that allows the estimation of the structural state and the detection of structural change that affects the performance of a structure (Kim 2007). Civil infrastructure is monitored with sensors which can characterize a signature property of a structure accurately under repeated evaluation. Properties such as vibration modal properties (frequencies, damping ratio and mode shape), strain and displacement are often used for characterizing civil structures.

The challenges for monitoring civil structures come from their larger size and complexity relative to mechanical systems such as rotors. In addition, unlike many mechanical systems, civil systems have to be monitored while they are in use. The constraints of operational SHM are the limiting factors for the selection of monitoring hardware and software.

A dense array of high-fidelity sensors is required to reliably characterize a large, complex civil structure such as a bridge. Celebi (2002) outlined the high costs of wired sensors for dense
deployment. For example, Celebi outlined that for the Bill Emerson Memorial Bridge in Missouri, the average cost per channel for 84 accelerometer channels was over $15000. Wireless sensors are an alternative approach to system characterization given their relative smaller sizes which minimize installation cost and time. For instance, for the Jindo Bridge in South Korea, the average cost per sensor for 70 sensors was ~$500, a significant cost decrease.

Three factors that have limited the commercial adoption of wireless sensors for civil systems are energy sustainability, the problem of data inundation and accessibility. Continuous, autonomous structural health monitoring requires reliable power supply and efficient, cost-effective power consumption. In addition, the data output from long-term data acquisition (DAQ) inherent in continuous monitoring requires a means of data mining to cull summary information to provide efficient but useful information to the end-user.

The issue of power management has been addressed from the supply and consumption side over the years with different degrees of success. The energy supply challenge for wireless sensors has drawn the interest of researchers from a variety of fields. Of note are attempts at recycling energy produced during vibration of civil structures under monitoring. In addition, there have been attempts to harvest energy from ambient sources. Ambient energy including wind (McEvoy 2011), solar (Miller 2010) has been attempted. The issue of power management has been addressed on the wireless sensor platform called the imote2 by developing a sleep-wake cycle mode of power usage to minimize energy use to periods of data acquisition.

Data inundation is a natural by-product of continuous SHM. Historical data is often needed to track damage initiation and propagation. However, an extensive database can inundate the end user without provide useful actionable intelligence/information. A real-time graphical user interface (GUI) can overcome the issue of data inundation by providing instant graphical
feedback on system performance while efficiently storing useful historical information. An application of real time GUI for monitoring civil structures has been implemented on the Donghai Bridge, the world’s largest bridge located in the East China Sea (NI, 2012). The Donghai Bridge is instrumented with sensors and data acquisition systems on the National Instrument platform. The hardware from NI is complemented with a GUI for rapid real-time feedback on structural health. However, all the sensors on the Donghai Bridge are instrumented with wired sensors. Figure 1 shows the schematic of the vibration-based monitoring of the Donghai Bridge. Figure 2 shows the GUI for Donghai Bridge SHM.

Finally, the issue of platform accessibility is important for SHM because health monitoring of structures should provide clarity about the state of health of a structure. To this end, a graphical interface for structure condition feedback is preferred over command line and computer batch mode output. The Matlab programming environment offers a powerful but accessible platform for developing robust algorithms that can also interface with external devices such as traditional DAQs through Universal Serial Bus (USB) cables. In addition, the Matlab platform supports GUI development with accessible features that an engineer can master with self-tutoring. The cross-platform functionalities of Matlab as well as its accessible programming language make it appropriate software for developing the GUI for SHM. GUIs that have been hosted in the Matlab platform include the Quarc Shake Table Control GUI (Figure 3) and Graphical User Interface for Guided Ultrasonic Waves (GUIUW) (Figure 4). Other existing GUI for SHM includes a corrosion monitoring GUI produced for the US Department of Defense (Figure 5).
Figure 1 On-line modal frequency monitoring schematic diagram (NI,2012)

Figure 2 On-line tracked resonance frequencies of Donghai Bridge (NI,2012)
Figure 3 On-line tracked resonance frequencies of Donghai Bridge (NI,2012)

Figure 4 Graphical User Interface for Guided Ultrasonic Waves (NI,2012)
Figure 5 Corrosion monitoring system GUI from DOD (NI,2012)
2.1.1. **Hardware and software background**

Developments in wireless smart sensor networks (WSSNs) technology for SHM of civil structures have enabled high-fidelity data to be extracted at relatively low costs. At the University of Illinois at Urbana-Champaign (UIUC), the Illinois SHM Project (ISHMP) has developed hardware and software systems aimed at continuous and reliable monitoring of civil structures using a dense array of smart sensors. The collaborative research between civil engineers and computer scientists at UIUC has resulted in an open source programming suite called ISHMP Services Tool suite made up of software foundation, middle-ware service tools and SHM application algorithms to be applied on an a smart wireless sensor hardware platform. The Services Tool suite uses TinyOS as its operating system. TinyOS applications are programmed in NesC, a variant of the programming language C but tailed to the specific constraints of small sensors with relatively small memory footprint.

The hardware utilized in the ISHMP project consists of a commercially available wireless sensor hardware platform from Crossbow and an embedded sensor board designed to meet the specific data acquisition needs for SHM for civil structures (Crossbow, 2012). The imote2 was selected because of its unique ability to satisfy the high-data through-put demands of SHM applications. The imote2 has a low-power processor with multiple processor speeds to optimize power consumption. The memory specifications are 32 RAM, 32MB of SDRAM and 32MB of flash memory. Further specifications about the capacities of the imote2 can be found with the manufacturer (MEMSIC, 2012). The sensor board, developed at by the ISHMP group was design with a flexible array of sampling rates and anti-aliasing filtering capabilities. The sensor capabilities of the sensor board include a 3-axis analog accelerometer for vibration measurement and 1 general purpose analog input channel as well as a digital temperature, light and humidity sensors. The sensor board interfaces with the imote2 via SPI and 12C I/O.
The ISHMP Services Tool suite is designed with a Services-Oriented Architecture that enables customization and addition of numerical applications for further SHM goals. The Tool suite is made up of foundation services for reliable wireless communication and data acquisition; sample application services made up of numerical algorithms for processing extracted data into useful system information; tools and utilities for network testing and debugging to enable assessment of the conditions of wireless communications and the health of the remote sensors; and continuous and autonomous monitoring services such as sleep cycle functionalities and sentry nodes for autonomous sensing capabilities.

The programming language (NesC) and operating system (TinyOS) used to develop the ISHMP Services Tool suite is common to many wireless sensor applications but relatively obscure to the larger engineering community. As such the ISHMP Services Tool suite is designed to minimize the operations to basic Windows routines. Still, the operating platform requires operating in a programming command-line interface which is unfamiliar to many engineers. As such the UIUC SHM team has developed a graphical user interface (GUI) to enhance accessibility of the platform. Extensive documentation on the UIUC ISHMP is available at the project site at UIUC (UIUC SHM, 2012).

The reliability of the ISHMP apparatus has been tested by multiple lab-scale and full scale deployments. In June 2010, the largest full-scale deployment of wireless sensors on a cable-stayed bridge was carried out on the Jindo Bridge in South Korea (Jang, 2010).

- Challenges
  The long-term sustainability and reliability of wireless smart sensors, and the imote2/ISHMP Tool suite in particular, is a critical issue for commercial adoption of this technology.
Issues that have limited commercial adoption of wireless smart sensors for continuous SHM of civil infrastructure include communication reliability including data packet loss, limited communication range of the on-board radio and reliable cost-effective power supply and management. Sim (2011) and Linderman (2010) have investigated the communication performance of the imote2 platform. Linderman investigated the performance of different antenna under varying environmental conditions. Sim developed a decentralized data acquisition and post-processing algorithm suite to overcome the limitations of radio range and minimize data loss.

The issue of platform accessibility is also important for the imote2 because the operating system and programming environment that supports the imote2 wireless sensor is relatively new and requires extensive user interaction. A GUI can overcome the accessibility issues of the TinyOS platform by creating an intuitive user-friendly interface which requires minimal training to use.

- Goals from ISHMP that ConImote2 addresses

The ultimate goal of wireless smart sensor approach to SHM is to enable continuous autonomous monitoring of civil infrastructure which provides early warnings of potential damage. This goal requires a multi-pronged approach including power management, development of feedback and warning systems in addition to the outlined communication and energy supply challenges. This project sought to address the issue of power management and warning systems through the development of a Matlab-based GUI, ConImote2, for continuous autonomous monitoring of civil structures on the imote2 platform.

The ConImote2 provides a graphical interface for continuous real-time data acquisition, post-processing and feedback on structural health. The data-acquisition implements a wireless
sensing routine based on the remote-sensing capabilities of the imote2. The post-processing procedure is a Matlab-based algorithm which extracts structural parameters from the DAQ output file to assess changes in structural health. A graphical alarm about the current state of structural health provides real time actionable information to the end user. Finally, a feedback loop is completed by setting different re-starting of the ConImote2 loop based on the state of structural health.

- Documentation on the Imote2

The imote2 platform was has been developed alongside hardware and software capabilities that minimize the amount of software knowledge for the end user. There is extensive documentation about the configuration of a host computer’s hardware and software environment in order to use the imote2. New users must use the documentation, *Getting Started Guide for New Users*, available from the Illinois Structural Health Monitoring Project (ISHMP) at the University of Illinois, Urbana Champaign to get a comprehensive introduction to the imote2. Adequate knowledge about the imote2 will enable the user to utilize the ConImote2 with understanding.

### 2.3. ConImote2: GUI for continuous monitoring on the imote2

The *ConImote2* is a GUI for continuous and autonomous real-time SHM of civil structures using the imote2 wireless network. The Goal of the *ConImote2* is to provide an accessible means of continuous SHM and instantaneous feedback on structural health. A schematic of the *ConImote2* is shown in Figure 6. The Matlab-based GUI initiates after a user command to start from a host computer by requesting sensor vibration and other data from a network of imote2 wireless sensors. Active sensors respond and send their data via a gateway imote2 connected to the host computer. The data received from the Gateway imote2 is sent to the *ConImote2* workspace via
USB cables connected between the gateway imote2 and the host PC. The data collected is processed. Properly collected data are processed for further analysis. The modal properties of the collected data are used to gauge the health of the structure using a baseline index of normal modal properties of the structure under study. The program sends feedback to about the structural condition to both the GUI and a sensing schedule function. This feedback is used to either wait periodically for the next scheduled sensing in the case of a healthy structure or to repeat sensing immediately if damage is detected.
Figure 6 Schematic of *ConImote2*
The parts of the GUI are shown in Figure 7. The GUI is made up of the following main parts:

1. Remote Sensing functionality form UIUC ISHMP Tool suite
2. User-defined criteria for normal behavior
3. Outputs
4. Post-processor for structural condition assessment
5. Utilities
6. Toolbars

Figure 7 ConImote2 Layout
2.3.1 Remote Sensing functionality form UIUC ISHMP Tool suite

The goal of ConImote2 is to provide an accessible interface for an alarming system on the structural health of civil structures using the vibration modes as an index of damage. The GUI consists of a sensing panel and feedback panel. The feedback panel in turn consists of a health status panel and plotting panel of historical summary of system information.

The sensing panel is made up of a Remote Sensing panel, which is a graphical implementation of the ISHMP tool called RemoteSensing, a data acquisition system. RemoteSensing consists of sleep/sake functionalities and synchronized sensing between a base station and a network of remote sensors on the imote2.

Sleep/Wake-up

ConImote2 seeks to reduce power consumption by taking advantage of the sleep/wake functionality to reduce the active power usage periods during operation of the deployed nodes. The sleep/wake functionality allows the nodes to be in deep sleep when the nodes are not in use. The nodes are timed to wake up periodically to listen for potential inquiries from the gateway node. As such the Sleep time to Wake time ratio should be optimized to ensure a high probability of communicating between the gateway and remote nodes while minimizing power consumption during the Wake cycle. The nodes automatically go into sleep mode after the end of data-acquisition. These capabilities of the imote2 has been utilized in the ConImote2 to periodically acquire vibration data from the network of remote sensors and return the system to sleep mode or to restart sensing based on the feedback about the system health.
Remote Sensing

The central application used for SHM in the ConImote2 is the RemoteSensing application in the ISHMP Tool suite which acquires synchronized data with varying sensing parameters from a network of sensors. The sensing parameters consist of the degree(s) of freedom for the time history or channel for external sensors, sampling frequencies, size of individual digital data points in the time history, data storage specifications in the limited memory of the imote2 RAM.

AutoComm

The autocomm.exe application from ISHMP Tool suite interfaces with the RemoteSensing application to store sensed data from a gateway imote2 to a host Windows computer. The autocomm.exe application is a command-line implementation of Windows HyperTerminal, a program used to connect a host computer to other computers through a serial port by means of a modem, cable or Ethernet connection. HyperTerminal allows the transfer of large files from a computer onto a portable computer using a serial port. It also enables recorded messages to be passed between the host and guest computer. The autocomm.exe application is required to be in the same folder as the ConImote2 executable files to ensure proper function.
2.3.2 Criteria for normal behavior
The ConImote2 uses the output of 3-axis accelerometers and a fourth external sensor connected to sensor boards mounted on the imote2 to assess the instantaneous health of a structure. The accelerations and displacements (if mounted to the external channel) are used to extract the modal properties of a structure in order to assess them for normal overall structural behavior.

The default criterion for normal behavior is that any of the natural frequencies assigned as the baseline settings is met. The default criterion setting is the setting in effect if none of the buttons in the Check? Panel in Figure 8 is clicked. A user-defined criterion for normal behavior is selected by checking the any number or all of the buttons next to the frequencies for modes of the structure under study. For the user-defined criteria, the modal properties used on the ConImote2 are the natural frequencies and mode shapes of a structure. To establish criteria for normal behavior, the user has to provide the ConImote2 with the baseline natural frequencies and corresponding mode shapes. The panels for user-defined baseline modal properties consist of a table of text editors for the natural frequency, a table of pop-up menus for the mode shape and buttons for establishing strict criteria for checking the health status. Figure 8 shows the panels for establishing baseline modal properties.
The natural frequency ranges are entered into text editors in the frequency pane of Figure 8. The user must ensure that the minimum (Min) and maximum (Max) of the frequency for each mode are ‘bins’ on a frequency domain plot of the acceleration or displacement data. The values that fall on the frequency bins are integer multiples of the resolution of the sampling frequency. The resolution of the sampling frequency is the ratio of the sampling frequency to the NFFT. The maximum value should be greater than or equal to the minimum value of the natural frequency in a given range. In addition, values of frequencies should be positive. Dummy frequencies above the sampling frequency can be assigned in the frequency settings if there are fewer modes than four under study.

The mode shape check includes a) a check of the relative amplitude of adjacent nodes and b) a relative phase check using the first node as a reference sensor. Based on the ConImote2...
algorithm, the imote2 nodes must be arranged in increasing order of Node ID to ensure properly association between the physical nodes and the baseline set up. For this purposes, in the *Mode shape* panel of Figure 8, the numerical order 1, 2, 3… *n* (where *n* is the size of the nodes under study) represents a node arrangement in increasing numerical order. For example, for 5 imote2 nodes with Node IDs 5, 18, 21, 49 and 85, if these nodes are arranged on a five story building with 1 node per floor, the nodes should be arranged in increasing order of their node IDs. Otherwise, the mode shape relative amplitudes and relative phase should be selected in Figure 8 according to the increasing order of the imote2 Node IDs irrespective of the arrangement sequence on the structure under study. Figure 9 illustrates the correct mode shape settings for a four storey building with different arrangements of the imote2. The mode shown is the fundamental mode.

![Figure 9 Four-storey building showing correct *ConImote2* mode shape configuration](image-url)
The amplitude check for mode shapes checks the relative amplitude of adjacent nodes in increasing order of node ID. The first node ID (lowest node ID) always has a mode shape value of 0. If the absolute value of mode amplitude of an adjacent node is higher than the preceding node, the value of the pop-up menu for that node should be set at 1; otherwise it should be set at 0. The algorithm uses these criteria for checking for mode shape amplitude. In this sense, a ‘correct’ mode shape is accepted as a mode shape that meets the relative amplitude criteria. It is envisioned that future releases of the ConImote2 will have functionalities for checking the error of mode shape amplitude with greater quantitative rigor based on mode shape curvature and strain energy.

The mode shape phase check checks the phase of nodes relative to the first Node ID. For a given mode, if a given node is in phase with the reference node, its phase value should be set at 0; otherwise it should be set at 1.

To illustrate proper baseline configuration of the mode shape amplitude and phase, Figure 10 shows the correct configuration for the second bending mode for a six storey building with an imote2 node on each floor. For the mode shape amplitude Node 5 is assigned 0 since it is the lowest node ID. Node 16 has higher absolute amplitude than Node 5; hence it is assigned a value of 1. Node 49 has lower absolute amplitude than Node 16; hence it is assigned a value of 0. Similarly, Node 51 has lower absolute amplitude than Node 49; hence it is assigned a value of 0. Nodes 66 and 85 both have higher absolute amplitudes than their preceding nodes so both are assigned values of 1. Unlike the mode shape amplitude check, the phase check uses a single reference sensor for all sensors. In Figure 10, the reference sensor is Node 5 since it has the lowest Node ID. Since Nodes 16, 49 and 51 are all in phase with Node 5, their phase values are set at 0. Nodes 66 and 85 are both out of phase with Node 5; hence their values are set at 1.
The ConImote2 is currently limited to unidirectional mode shapes. That is, for each mode under study, the acceleration output for each imote2 node under study is separated into the DOFs under study. As such, each mode check is applied to individual DOFs. All DOFs are checked for existence of the modes under study. As such if the mode under study is exhibited by multiple DOFs of the all nodes, the system accepts normal behavior for any DOF which exhibits correct behavior. While it is highly unlikely that a given mode will be exhibited in both natural frequencies, mode shape relative amplitude and mode shape relative phase by multiple DOFs for all imote2 nodes under study, this can happen for 3D modes if there are relatively few nodes under study. In the case of 1 node, for instance, the modes shape relative amplitudes and relative phase values are always 1 and 0 respectively. As such, if the range of the natural frequency is

Figure 10 Six-storey building showing correct ConImote2 baseline mode shape setting
broad, it is likely that multiple DOFs will exhibit the mode under study. This is a limiting factor for the ConImote2 which will be remedied in future releases.

2.3.3 Outputs
Health Status

The innovative aspect of the ConImote2 is its ability to create an instantaneous feedback about the state of structural health in real time and autonomously. To enable this, an algorithm suite uses a baseline defined by the user to determine quantitative and graphical description of the current health status. Using a vibration-based damage detection algorithm based on modal properties, the ConImote2 compares the current extracted modal parameters to a user-defined baseline to determine the possible existence of damage. To prevent false alarms, the ConImote2 carries out multiple sequence of damage assessment if damage is detected to confirm the existence of damage. On the other hand, if the status of the structure is normal, the ConImote2 goes into an inactive mode for a considerable time before returning to the next assessment cycle. The health status panel is shown in Figure 11.
A health status panel consists of graphical feedback and a graphic of health status is prominently located on the GUI to alert the user about the current state of health. In addition, the natural frequencies of the structure are displayed in numerals to provide instant feedback. Figure 12 is an overview of the key code for graphical overall health status and their meanings. The overall health status is based on the criteria set by the user as explained in the Criteria section. The default graphic for the Health status is a start window which shows that the program has not completed a first run since being started. Normal health is shown by a green graphic and a warning graphic shows abnormal behavior.

Figure 11 Health status panel a) before starting b) after a run

Figure 12 Key codes for overall health status
Further status of individual modes is provided using check marks and crosses to represent normal and erroneous behavior. In Figure 13 the default status for each mode is a question mark showing unknown status. A cross sign shows error modes and a check sign shows normal modes. In addition to the graphical feedback, each natural frequency extracted is expressed in numerical form under the headings for natural frequency. An empty value represents error modes.

![Image of key codes](image)

*Figure 13 Key codes for health status of individual modes*

**Time History**

The time history panel provides a visual of the most recent data acquired from the active modes. It allows the user to instantly visualize the level of structural response such as acceleration, displacement or strain.

**Frequency domain**

The plot of the output data in the frequency domain provides an auto-spectral density function of all data collected from the sensors. It allows the user to observe detailed information about the frequency domain behavior of the most recent data collected.
Debug Imote2

The debug panel provides the most recent command line interface that the traditional imote2 displays. It allows the user to track the debugging process of the most recent data acquisition to track sources of errors and gain feedback about network performance that is not displayed graphically.

Status

The status panel provides the real time batch mode output of the ConImote2. It gives a message about all commands to the ConImote2 such as starting, stopping, database clean-up as well as the instantaneous output of the debugging remote sensing commands.
2.3.4 Post-processor for structural condition assessment

Plotters

The plotters provide visual feedback on the long-term behavior of the structure under study. A historical plot of the damage index allows the user to track changes in structural behavior and/or erroneous data acquisition. The historical data is a critical piece of ConImote2 since it allows the user to gain actionable intelligence about long-term changes in the structure in order to enable preventive measures before complete collapse of the structure. Historical frequency plots can also provide immediate information about the effect of cyclical environmental or seasonal factors on the modal behavior of a structure. For example, the effect of daily temperature or wind patterns on natural frequencies can be directly studied with the historical plot. The toolbars for interacting with figures in the ConImote2 allows the user to zoom into the data and study data trends closely.

In addition there is a graphic window about the statistics of each mode. The mean and standard deviation for each mode enables the user to gain a quantitative expression for the normal variations of frequency over a period under study. This window allows the user to study the range of frequencies. In addition, for long-term monitoring purposes, the statistics window provides insight into the level of change which can lead to catastrophic failure of a structure. A statistical summary of damage indices provides a measure of data variation and confidence level in the data acquisition process.
2.3.5 Utilities

Stopper
A stopper is added to stop the program to allow refreshing and re-configuration of the deployed network. After stopping the program, the user should allow the current sensing and post-processing cycle to complete in order to have access to inactive GUI functionalities which become inactive to the user during runs. It is recommended that the program be stopped close to the finish of a run. If there is a need to immediately stop the program, it can be stopped in the Matlab command window using Control + C and refreshing the gateway node. Figure 14 shows the Stop push button for the ConImote2

![STOP!](image)

Figure 14 Stop push button for the ConImote2

File clean-up
File clean-up functionality removes old output files to de-clutter the folder that contains ConImote2 files in order to speed-up routines and prevent the Matlab allocated memory from becoming exceeded. Figure 15 shows the File clean-up tool.

![Clean Up](image)

Figure 15 File clean-up tool

Toolbars
Toolbars are added to the ConImote2 to assist the user in interacting with the outputs of the GUI. Specifically, a legend toolbar, a zoom in and zoom out functionality, a data point picking toolbar, a screen printer and pan toolbar have been added. A legend toolbar enables the user to find the data from a specific sensor in a network. The zoom in and zoom out toolbar is handy given the
relatively small windows used for plotting output in the *ConImote2*. The data point picking toolbar enables the user to select specific data points for further study. The printer prints the *ConImote2* window and can be configured in the windows printer. The printer works like any windows printer. Figure 16-17 shows the Toolbars in the *ConImote2*.

![Figure 16 Toolbars in *ConImote2*](image)

**Figure 16 Toolbars in *ConImote2***

![Figure 17 Data selection toolbar used to select a data point in Frequency history plot](image)

**Figure 17 Data selection toolbar used to select a data point in Frequency history plot**
2.4 Validation

2.4.1 SDOF building with mass perturbation

Test bed description

The test bed is a lab-scale single-story building commercially produced by Quanser (Quanser 2011). The building, shown in Figure 18, is equipped with an accelerometer to measure the top (roof) acceleration relative to the fixed base. The structure frame is made from steel and has a flexible façade.

![Figure 18 a) Single story building with active mass damper attached b) Single-story with accelerometer](image)
The building was fixed to a shake table by four bolts at the corners of the building for vibration tests. For all tests, the input function consisted of a continuous chirp function with 0.2cm amplitude which provided adequate excitation to the building roof. An experiment was set-up to assess the ability of the ConImote2 to detect structural changes that affect the modal properties of the building. The structural change on the building consisted of adding masses to the mass of the roof successively. The masses were standard steel disc masses. The values of the masses were selected to cause dramatic changes in the single natural frequency of the building. Given that the single-story building can be approximated as lumped mass-spring system, the natural frequency, in hertz (Hz) for the single degree of freedom (SDOF) system with roof mass, \( m \) in kg and frame stiffness, \( k \) in N/m is expressed in Equation 1.

\[
f(\text{Hz}) = \frac{1}{2\pi} \sqrt{\frac{k}{m}}
\]  

(1)

The top floor mass included fixed masses given by the mass of the roof, the Quanser accelerometer as well as variable masses from the imote2 nodes, attachment brackets as well as the standard steel masses. All masses were attached to the roof floor to create lumped masses for the SDOF system. The mass of the plastic roof was 0.68kg; the Quanser accelerometer had an additional mass of 0.2kg. Each imote2 node with the bracket for attachment had a combined weight of 0.177kg. The standard masses included a 397.1g, 758g and a third mass made from combining the two masses into an 1155.1g mass. Figure 23 shows the masses and the attachments. The effective lumped masses at the roof floor of the single story building are summarized in Table 1. The stiffness of the frame was assumed to stay constant at the design-specified value of 500 N/m. The natural frequencies of predicted by Equation 1 are also summarized in Table 1.
The design specifications of a complete unit of the building consist of a single story building attached to an active mass damper (AMD). Details about the structure are given in the Quanser Manual (Quanser 2011). The complete unit of the building attached to the AMD has a natural frequency of 2.5Hz. However, in the tests conducted with the ConImote2, the AMD was not part of the structure. As such a new system identification of the new structure without the AMD was carried out to establish a baseline structure for further testing.

Preliminary tests were conducted to establish reasonable sensing parameters such as the number of samples and sampling frequency for the RemoteSensing application in ConImote2. The first preliminary test consisted of attaching two imote2 nodes to the roof. Based on the relatively low natural frequency, 3.25 Hz of the new structure, a sampling rate of 25Hz with an Nfft of 64 and 1000 data samples to ensure 40 seconds of data was used for all subsequent tests. This sampling frequency was adequate since the natural frequency of the structure with additional masses was expected to decrease. To make room for the standard masses in future tests, it was determined that only one imote2 node could be used for the tests. The decreased lumped mass as a result of the removing one imote2 unit resulted in a 0.25Hz drop in the natural frequency from ConImote2. Equation 1 confirmed this drop. Hence, the preliminary tests confirmed that the ConImote2 was reliable in its assessment of structural behavior. Figure 19- Figure 23 show the set-up of the lab-scale building with the imote2 sensor and the fixed mass attached by tape to the roof of lab-scale building.

The tests included 5 stages of progressively adding masses to the roof in the order stated in Table 1. For all tests, one imote2 node was used. The baseline structure consisted of the building with one imote2 attached. A numerical predication of the natural frequencies was used to set a bound of the natural frequencies for the single mode. The frequency range was set at 2Hz
to 4Hz in order to create a visual of the change of the structure with additional masses in the historical frequency plot as well as the frequency statistics plot. The masses were added in increasing order of their values. Masses were secured by tape to the roof. The masses were added after about 8 runs with each mass. Sample screen shots of the program during the runs are shown in Figure 25. Table 1 summarizes the frequencies determined by ConImote2 for each additional mass.

Figure 19 Test bed for test on lab-scale building
Figure 20 Two imote2 sensors on the roof of lab-scale building

Figure 21 DAQ for ConImote2 on the lab-scale building
Figure 22 Zoom in of imote2 connection on the roof of lab-scale building

Figure 23 Arrangement of standard mass on the roof of lab-scale building
Table 1 Tests on progressive mass changes on lab-scale single story building

<table>
<thead>
<tr>
<th>CASE</th>
<th>Masses</th>
<th>Mass (kg)</th>
<th>Natural frequency (Hz)</th>
<th>Frequency error (Hz)</th>
<th>Δ mass (kg) (Δ%)</th>
<th>% change in frequency</th>
</tr>
</thead>
<tbody>
<tr>
<td>Quanser Design</td>
<td>Roof + AMD</td>
<td>2.0</td>
<td>2.52</td>
<td>2.5</td>
<td>0.02</td>
<td>0.943 (+89%)</td>
</tr>
<tr>
<td>Trial</td>
<td>Roof + 2 Imote2</td>
<td>1.234</td>
<td>3.20</td>
<td>3.25</td>
<td>0.05</td>
<td>0.177 (+16%)</td>
</tr>
<tr>
<td>Baseline</td>
<td>Roof + 1 Imote2</td>
<td>1.057</td>
<td>3.46</td>
<td>3.5</td>
<td>0.04</td>
<td>Baseline (Baseline)</td>
</tr>
<tr>
<td>Baseline</td>
<td>Mass 1</td>
<td>1.4541</td>
<td>2.95</td>
<td>3</td>
<td>0.05</td>
<td>0.3971 (-38%)</td>
</tr>
<tr>
<td>CASE 3</td>
<td>Baseline + Mass 2</td>
<td>1.815</td>
<td>2.64</td>
<td>2.5 - 2.75</td>
<td>0.015</td>
<td>0.758 (-72%)</td>
</tr>
<tr>
<td>CASE 4</td>
<td>Baseline + Mass 3</td>
<td>2.2121</td>
<td>2.39</td>
<td>2.25</td>
<td>0.140</td>
<td>1.1551 (-109%)</td>
</tr>
</tbody>
</table>

Results and Discussion
The natural frequencies for each test were recorded from the ConImote2 Feedback pane. The variation of natural frequency of the system with changes in mass showed an inverted exponential curve. Overall, there was strong agreement between the theoretical prediction and the ConImote2 results as shown in Figure 24. The rate of change of the natural frequency decreased with increasing mass. This characteristic of the SDOF system affected the quality of the results as the mass increased. This was because the resolution of the frequency was fixed at a ratio of the sampling frequency to the NFFT, so at higher masses, the relatively smaller change in natural frequency was less than the resolution of the sampling frequency. In Table 1, for the case of the baseline and Mass 2, the theoretical natural frequency was 2.64 Hz, Since the resolution of the frequency curve was 0.25 Hz, the ConImote2 results showed a range of natural frequency from 2.5Hz to 2.75Hz.
Figure 25a) shows the ConImote2 historical plot of the natural frequency as the mass changes were applied. A zoom-in on the historical plot is shown in Figure 25b). Stage A represents the preliminary trial test with the building and 2 imote2. Stage B represents a trial test with Mass 1. Stages C to F represent the Baseline, Case 3, Case 4 and Case 5 respectively. It is seen in Stage E that the natural frequency toggles between 2.5 and 2.75 because of the low resolution of the sampling frequency and NFFT. For Case 5, the ConImote2 results had a greater error from the theory prediction because the resolution was not high enough to correctly predict the actual natural frequency. The ConImote2 results were the next lower frequency bin at 2.25Hz. Based on the results of this term it is clear that a higher NFFT and sampling frequency is necessary to for high fidelity structural change at low frequencies.
Figure 25  Historical plot of natural frequency for single storey building
2.4.2 MDOF bridge with loose bolt

Test bed description

The test bed is a lab-scale modular truss bridge made from steel elements (Figure 26-Figure 27). The axial members are connected through gusset plates. The superstructure is a 3D skeletal frame is without a deck. The single span Howe truss consists of twelve longitudinal bays and a single transverse bay. The bottom chords at opposite ends are supported by a pin support at one end and a roller support at the other end. The supports consists of 3 feet high steel frames which are connected to the ground by 5-foot long bolts and tightened until they are snug tight. A sequence of structural changes was applied to the bridge to progressively induce damage.

Figure 26 Lab-scale truss bridge
A network of wireless sensors and wired sensors was installed on the bridge in order to assess the ability of the ConImote2 to detect damage using a network of wireless sensors. The wireless sensors consisted of up to 8 imote2 sensors and 12 wired accelerometers. The imote2 sensors included 7 sensors for multi-axial acceleration measurement and 1 sensor connected to an external displacement sensor. The imote2 sensors were connected to the truss by modular brackets shown in Figure 28. The wired accelerometers were connected by magnetic connectors.
A series of vibration tests were performed on the instrumented truss by applying random excitation to the middle joint of the bottom chord by means of a LDS shaker (Figure 29). The random excitation was programmed using a force application and data acquisition (DAQ) system from m+p International. The hardware component of the DAQ, called VibPilot, consisted of signal conditioners, filters and analog-to-digital converters (ADC) while the software component, called SO Analyzer, consist of customizable settings for sampling parameters and data storage systems.
The tests on the truss consisted of preliminary tests and 4 main tests involving progressive damage of the truss bridge. The preliminary tests were used to select sampling parameters and to diagnose bugs during network monitoring on the ConImote2. During the preliminary tests the wired accelerations were also collected in order to compare the results of both tests. The preliminary results for the z-direction of the wired sensors are shown in a frequency domain plot in Figure 30 and Figure 31. The peaks in the frequency domain were dominant at frequencies of 19Hz, 24Hz, 39Hz, 70Hz and 90Hz. The mode shapes for the wired sensors were confirmed using two system identification techniques. A time domain called Eigenvalue realization algorithm (ERA) was used in addition to the frequency domain to screen out noise modes. From the five frequencies extracted from the frequency domain method, it was confirmed that the 24Hz mode was a noise mode. The mode shapes at the dominant frequencies were extracted using a mode shape building algorithm built in Matlab by the author for routine modal analysis. Figure 32 to Figure 33 show the mode shapes for 19Hz, 39Hz and 70Hz. The mode shape for 19Hz is the first vertical bending mode. The mode shape for 39Hz is a torsional mode which has a single bending in the z-direction. The mode shape at 70Hz is the second bending mode. These modes served as the baseline modal properties which were used for damage assessment in the remaining tests.
Figure 30 Power spectral density of vertical acceleration using wired sensor (Sensor 6)

Figure 31 Power spectral density of vertical accelerations using 11 wired sensors
Figure 32  Mode shape for f = 19Hz for wired sensor

Figure 33  Mode shape for f = 39Hz for wired sensor
The tests on the truss bridge with wired sensors and the ConImote2 were conducted simultaneously to confirm the robustness of the ConImote2 relative to traditional health monitoring. As such the tests included the following:

1. Baseline structure test
2. Damage 1: Replacing a single truss top chord element with a ‘damaged’ version
3. Damage 2: Sequential loosening of all bolts at a single joint until complete release
Baseline structure test

A baseline of the truss bridge’s modal frequencies was established before the damage scenarios were applied. The baseline test with the ConImote2 and the wired sensor DAQ replicated the frequencies identified in the preliminary studies. The sampling specifications for the ConImote2 and the wired DAQ are summarized in Table 2. The highest sampling frequency available for the sensor boards used for the Imote2 is 280Hz. The ConImote2 resampled the data collected with a 280Hz to 256Hz for easier data post-processing. For the baseline tests, the mode at ~90Hz was not reliably collected by the ConImote2 because of the NFFT of 512 used. The choice of a low NFFT was chosen in order to get single peaks at resonant frequencies for the peak peaking algorithm used for the ConImote2. As a result of losing the ~90Hz due to this trade-off, the ConImote2 algorithm was refined to overcome the issue of finding the exact resonant peak. This problem has since been resolved. However, at the time of the test it was decided to use 19Hz, 39Hz and 70Hz and put the noise mode in as a dummy mode without requiring that its mode shape be checked. For the wired DAQ, the sampling frequency used was 512 Hz with an NFFT of 4096. Using these sampling properties resulted in high fidelity frequency domain resonant modes.

The results of the baseline tests showed that the ConImote2 and the wired DAQ system gave equal results of resonant frequencies at 19.5Hz, 39.5Hz and 70.5Hz. In addition, the wired DAQ showed higher modes including the mode at 90.75 Hz. Table 2 summarizes the results and their statistics for running the ConImote2 for 30mins. The ConImote2 interface for a sample run for the baseline is shown in Figure 35. From Figure 35 it is clear that for all 7 sensors, the frequency domain has a well-defined peaks that coincide at the same resonant frequencies. This is a mark of an ‘undamaged’ baseline structure.
Table 2 Baseline results of ConImote2 and Wired DAQ for truss bridge

<table>
<thead>
<tr>
<th></th>
<th>Fs</th>
<th>NFFT</th>
<th>Mode 1 (Hz)</th>
<th>Mode 2 (Hz)</th>
<th>Mode 3 (Hz)</th>
<th>Mode 4 (Hz)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Mean</td>
<td>SD</td>
<td>Mean</td>
<td>SD</td>
<td>Mean</td>
<td>SD</td>
</tr>
<tr>
<td>ConImote2</td>
<td>256</td>
<td>512</td>
<td>19.5</td>
<td>0</td>
<td>39.5</td>
<td>0</td>
</tr>
<tr>
<td>Wired DAQ</td>
<td>512</td>
<td>4096</td>
<td>20</td>
<td>-</td>
<td>39.65</td>
<td>-</td>
</tr>
</tbody>
</table>

Figure 35  ConImote2 baseline test showing high fidelity CPSD plot
Damage 1: Replacing a single truss top chord element with a ‘damaged’ version

The first test involved replacing a single top chord element with an element with a cross sectional area that is half of the original area (Figure 36). For a truss element, the axial stiffness, is given by

\[ k = \frac{EA}{l} \] (2)

where \( E \) = Elastic modulus, \( A \) is the cross-sectional area and \( l \) is the length of the member. Hence the stiffness is reduced by 50% for a 50% decrease in the cross sectional area. Ideally, a numerical or finite element model of the truss should have been used to predict the gross effect of the local change in element stiffness but time constraints prevented this portion of the research. Hence there was no basis for comparing the experimental behavior with a theoretical prediction.

After several runs of both ConImote2 and the wired sensors, there was small decrease in all natural frequencies for both the ConImote2 and the wired DAQ. The drop was small enough that the new frequencies of the bridge were well within the user-defined bound of normal frequencies. Nonetheless, the user could see that the structure had changed. Given that there was no numerical model to compare the experimental results, there was no basis to assess the impact of the damaged element on the overall static and dynamic behavior of the truss bridge. Future research will be aimed at documenting the load carrying and distribution pattern of the truss after damage. This will enable the user to create an informed bound of normal natural frequencies. On the other hand, the lesson learnt was that for large scale structures, in order for damage to be observed, the damage has been severe. Hence it can be deduced that if a frequency domain technique detects damage, the damage level has been severe. This property of the frequency domain makes it a strong candidate for long-term SHM, which is the goal of creating the
ConImote2. The conclusion and lesson from this test was that a more quantitatively rigorous criterion for damage is necessary in order for the modal properties-based damage detection to be useful for diagnosing minor damage. Table 3 is a summary of the change in frequencies after replacing the damage element.

Figure 36 Damage Case #1 showing thinner ‘damaged’ element replacing top chord element
## Table 3 Results for Damage 1 for ConImote2 and Wired DAQ

<table>
<thead>
<tr>
<th></th>
<th>Fs</th>
<th>NFFT</th>
<th>Mode 1 (Hz)</th>
<th>Mode 2 (Hz)</th>
<th>Mode 3 (Hz)</th>
<th>Mode 4 (Hz)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>Mean</td>
<td>SD</td>
<td>Mean</td>
<td>SD</td>
</tr>
<tr>
<td><strong>ConImote2</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Baseline</td>
<td>256</td>
<td>512</td>
<td>19.5</td>
<td>0</td>
<td>39.5</td>
<td>0</td>
</tr>
<tr>
<td>Damage 1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ConImote2</td>
<td>256</td>
<td>512</td>
<td>19</td>
<td>0</td>
<td>39</td>
<td>0</td>
</tr>
<tr>
<td>Wired DAQ</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Baseline</td>
<td>512</td>
<td>4096</td>
<td>20</td>
<td>-</td>
<td>39.65</td>
<td>-</td>
</tr>
<tr>
<td>Damage 1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Wired DAQ</td>
<td>512</td>
<td>4096</td>
<td>19.63</td>
<td>-</td>
<td>39.15</td>
<td>-</td>
</tr>
</tbody>
</table>
**Damage 2: Sequential loosening of all bolts at a single joint until complete release**

The next test involved loosening the bolts of one joint sequentially. The sequence of tests followed the baseline test in which all bolts were tightened to ensure the torque was over 200Ib-in. After this a single top joint was loosened until the torque in all pin joints at that gusset plate was 50 Ib-in. Finally, all the joints bolts were fully released to induce the case of complete loose joint.

A preliminary test was conducted with a completely loosened bolt at one of the joints where the thinner damaged element was connected to the top chord. This test was performed to assess the impact of connection on the modal frequencies. To ensure that the structure retains its initial elevation after loosening the bolts, a support was provided to the bays around the joint. Firstly, only two sensors were placed at the damage location to assess local effects. The sensors were placed at the opposite joints of the damage member. After running the ConImote2 for up to 30mins, it was observed that the natural frequencies for the higher modes had seen significant drops. Specifically, the resonant frequency at the baseline 39.5Hz had dropped to a range of 38 and 38.5Hz. The resonant frequency at the baseline 70.5Hz also dropped to a range of 69 to 69.5Hz.

Figure 37 ConImote2 frequency domain plots a) before completely loosened bolts at joint along with damage element in top chord shows clear frequency drops in higher modes shows a comparison of the ConImote2 Frequency plots before and after loosening the bolts at the damaged element. It is clear, especially from the CPSD plots that the structure had become more flexible at the damage location.
After the preliminary test, a more structured test on the effect of joint damage was conducted by sequentially reducing the torque applied to all the bolts at the damage location. To focus exclusively on connections, the healthy top chord element was returned. The results showed gradually decreasing natural frequencies. The results are summarized in Table 4 and Figure 38. In Table 4 and Figure 38, the damage is the percentage change in natural frequency between the damaged state and the baseline. It was observed that the percentage damage increased as the bolts were loosened. The error was higher in the absolute change of frequency as at higher resonant modes. The results between the wired DAQ and the ConImote2 showed strong agreement. From the results it was clear that the higher modes were more sensitive to damage than the lower modes as seen in the relatively higher rate of change of damage for the resonant frequency at the baseline of 90.75Hz versus that at 20Hz for the wired DAQ.
Table 4 Results for Single bolt on ConImote2 and Wired DAQ

<table>
<thead>
<tr>
<th></th>
<th><em>ConImote2</em> Baseline</th>
<th>200 lb-in <em>ConImote2</em></th>
<th>50 lb-in <em>ConImote2</em></th>
<th>0 lb-in <em>ConImote2</em></th>
<th><em>Wired DAQ</em> Baseline</th>
<th>200 lb-in Wired DAQ</th>
<th>50 lb-in Wired DAQ</th>
<th>0 lb-in Wired DAQ</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fs (Hz)</td>
<td>256</td>
<td>256</td>
<td>256</td>
<td>256</td>
<td>512</td>
<td>512</td>
<td>512</td>
<td>512</td>
</tr>
<tr>
<td>NFFT</td>
<td>512</td>
<td>512</td>
<td>512</td>
<td>512</td>
<td>4096</td>
<td>4096</td>
<td>4096</td>
<td>4096</td>
</tr>
<tr>
<td>f1 (Hz)</td>
<td>19.5</td>
<td>19</td>
<td>19.5</td>
<td>19</td>
<td>20</td>
<td>19.5</td>
<td>19.38</td>
<td>19.25</td>
</tr>
<tr>
<td>% Damage f1</td>
<td>0.00</td>
<td>2.56</td>
<td>0.00</td>
<td>2.56</td>
<td>0.00</td>
<td>2.50</td>
<td>3.10</td>
<td>3.75</td>
</tr>
<tr>
<td>f2 (Hz)</td>
<td>39.50</td>
<td>39.00</td>
<td>38.50</td>
<td>38.50</td>
<td>39.65</td>
<td>39.25</td>
<td>38.42</td>
<td>38.59</td>
</tr>
<tr>
<td>% Damage f2</td>
<td>0.00</td>
<td>1.27</td>
<td>2.53</td>
<td>2.53</td>
<td>0.00</td>
<td>1.01</td>
<td>3.10</td>
<td>2.67</td>
</tr>
<tr>
<td>f3 (Hz)</td>
<td>70.50</td>
<td>70.00</td>
<td>69.00</td>
<td>69.00</td>
<td>70.95</td>
<td>70.53</td>
<td>69.23</td>
<td>69.11</td>
</tr>
<tr>
<td>% Damage f3</td>
<td>0.00</td>
<td>0.71</td>
<td>2.13</td>
<td>2.13</td>
<td>0</td>
<td>0.59</td>
<td>2.42</td>
<td>2.59</td>
</tr>
<tr>
<td>f4 (Hz)</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>90.75</td>
<td>89.45</td>
<td>86.63</td>
<td>86.63</td>
</tr>
<tr>
<td>% Damage f4</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>0</td>
<td>1.43</td>
<td>4.54</td>
<td>4.54</td>
</tr>
</tbody>
</table>
Figure 38a. Natural Frequencies for 4 modes with progressively loosened bolts
Figure 38b. Variation of resonant frequencies as bolts are progressively loosened on the lab-scale truss bridge
2.5 Troubleshooting

The key requirement for autonomous operation of Con Imote2 is the ability to restart the loop if a minor error disrupts the program. Common errors that crash the program includes malfunctioning nodes, low battery power and data loss. Malfunctioning nodes can result from communication errors due to unfavorable operating environment (Linderman, 2010). Nodes can also lose functionality over time. Low battery power shuts down the communication and data transfer capacities of the nodes. Finally, data loss can result from unspecified factors (Sim, 2011). This error can disrupt the algorithm. Results from extensive lab-scale tests were used to create safeguard features to minimize the disruptions caused by the outlined errors.

- **Erroneous nodes**
  Erroneous nodes are caused by low battery power, poor wireless communication due to antenna damage or unfavorable medium. A key requirement for the RemoteSensing application is that specified remote nodes stay awake with adequate power supply throughout the period of communication. An inactive requested node delays data acquisition of the other nodes in the network. The RemoteSensing application ‘waits’ on a non-responsive node for a few milliseconds before it resumes the remaining operations. A dysfunctional node, therefore, does not negatively impact other nodes beyond delaying the overall network operation.

- **Timeouts**
  Unlike the RemoteSensing application, ConImote2 relies on Microsoft Windows serial port communications protocols which require operations to be timed through a setting called Timeout. Timeout options, in milliseconds for serial port communications range from a default
Timeout of 500 or user-defined values greater than 0. An option to prevent a Timeout is specified by the setting ‘InfiniteTimeout’. The Timeout specification is critical to intelligent operation of the ConImote2. Data acquisition is only meaningful if it occurs during the active phase of the structure under study. More importantly, an alarming system must provide timely information to be useful. As such, speed of data acquisition and processing requires optimized Timeout periods. Factors that affect the speed of serial port communications are size of network nodes, number of data samples requested. Higher values of each of these factors result in longer operating times which require longer Timeout periods. The InfiniteTimeout setting is used to permit multiple attempts at communication if erroneous behavior is detected.

- **Data port initialization errors**

The autocomm.exe application requires a Microsoft DOS command prompt to request data output from the imote2 to the gateway computer. The execution time to the batch operation of this step in the DAQ stage needs to be managed for proper completion of the ConImote2 routines. A command to delay proceeding events after this operation was needed to prevent serial port initialization errors which disrupt the program loop. Figure 39 shows the error.
Figure 39 Data port initialization error

- **Clean-up files to prevent data inundation and prevent Matlab crush**

*ConImote2* is created in the Matlab programming environment. It is thus subject to the speed limitations of the Matlab compiler as well as limitations on the size of files. Two Matlab-related issues that affect the performance reliability of *ConImote2* are the efficient use of memory allocated to Matlab files and speed of the Matlab compiler. Many routines in the *ConImote2* program require reading large data structures and sorting data files. To increase the efficiency, file clean-up functionality has been added to the program to remove very old files that do not provide actionable information. Specifically, the *Clean Up* button removes output data files older than the 10 most recent files. The newer files are retained to provide the end user recent unaltered data for further analysis.
2.6 Future research

- Develop a trigger

The ConImote2 in its current form does not use a trigger for initiation data acquisition. Instead it uses a self-starter based on a timer. The timer initiates data acquisition periodically. If the ConImote2 returns a damage start, data acquisition occurs more frequently to confirm or dispute the damage state. Otherwise, if the structure under study is healthy at the time of the normal periodic assessment, the ConImote2 goes into an inactive mode for a longer period of time. This is the main trigger system in the ConImote2. Future extension of the ConImote2 will be aimed at adding a trigger system provides feedback if an exciter is approaching the structure. The exciter for a bridge, for instance, is a speeding vehicle. Interference-based systems already exist for highway patrol and can be adopted as an external interface to the gateway computer of the ConImote2 to serve as a trigger to the ConImote2.

- Full-scale tests

Full-scale tests are needed to diagnose bugs that might exist for under uncontrolled conditions such as on a highway. Extensive testing on the lab-scale bridge brought to light network communication related problems such as data loss which might not be observed under controlled conditions. It is therefore import to test the ConImote2 on a full scale in-service highway bridge or structure.

- Multi-scale assessment

Currently, the ConImote2 has been tested for vibration-based sensors. It is the goal of SHM to combine multiple sensors for fuller system monitoring. The external channel on the UIUC
sensor board allows any analog sensor to be connected to it. A brief test with a displacement sensor was carried out on the ConImote2. This test showed that the functionality works, hence the ConImote2 can be extended for multiple sensors.

- Multiple system identification and damage detection methods

Finally, it is important to verify damage by using different damage detection methods and system identification (ID) algorithms. The ConImote2 uses the Peak-picking method for modal-analysis based damage detection. For the validation test on the truss bridge, the ERA method confirmed all the results of the ConImote2 algorithm. However, multiple system ID methods and damage detection techniques will increase confidence in the results.
2.7 Overview of steps for running the ConImote2

1) Remote Sensing Panel

A. Connect a gateway imote2 node to an interface board and a network of imote2 remote sensors as directed in the user’s guide for the imote2 (Refer to the Getting Started Guide for New Users from the UIUC ISHMP).

B. Enter the Node IDs of the imote2 (Refer to the Getting Started Guide for New Users for finding Node IDs of the imote2 and entering it in the correct format).

C. Node IDs can be repeated if one mistakenly selects more nodes than exist in the network. **NB: Any non-zero selection is assumed to be an existing imote2 in the network so use repetition to correct Node ID selection errors.**

D. Select the windows COM port IDs as directed in the user guide.

E. Enter the number of time history data points per cycle of data acquisition in the Samples window.

F. Enter the sampling frequency and NFFT for frequency domain analysis.

G. Select the channels (x, y, z and external channel) for the sensors.

H. Select the synchronizing and data storage options for the RemoteSensing functionality of the ISHMP tool suite using the buttons for Synch and Data block.

2) Healthy Modal Properties Panel

A. Enter a range for each resonant mode being used as a damage index.

B. Select the button under Check? for each mode which is being enforced as a damage index. Leave Check? Unchecked for all if any correct mode is enough for health status.
Refer to section on *Criteria for normal behavior* for better understanding of this functionality.

C. Select the mode shape and phase description using the key code described in the *Criteria for normal behavior* section.

D. Review the information

E. Click *Start imote2 now* to start running.

F. Click Stop to stop running. It is recommended to wait for the program run to near completion before stopping. Refer to the notes on Utilities for more about starting and stopping.
Chapter 3 Multi-scale FE updating of skewed in-service highway bridge

Vibration-based finite element (FE) model updating method has drawn significant attention for structural health monitoring (SHM) in decades. In vibration-based FE model updating methods, the optimum model is selected by minimizing the error between modal properties of the model and the real structure. A major drawback of model selection is the probability that the optimum model has lack of physical connectivity with the real structure given that a large set of updating parameters is required to increase accuracy. Model selection is especially difficult for structures monitored under operational conditions with many uncertainties. The multi-scale data approach provides an evidence-based method to ensure that the optimum model retains physical connectivity to the real structure. In this research, FE model updating based on multi-scale data is demonstrated on a skewed in-service highway bridge. Multi-scale data including acceleration, temperature and tilt were measured from previous bridge monitoring projects for this highway bridge. A comprehensive output-only modal analysis based on nearly 700 ambient vibration data has been conducted to set up multi-scale physical evidence for model updating. An evidence-based approach to model selection using temperature-induced tilts is implemented in which the cyclical static behavior of a bridge is used to create a bound of possible models. This approach has a strong potential to be applicable to large civil structures with a sparse array of sensors.

3.1 Introduction

The finite element (FE) model updating of a structure is based on assumptions of material and geometric properties, boundary conditions and in the governing physical relations which might differ from the in situ properties. These uncertainties about the material as well as defects in the material and geometry during construction might lead to measurable differences in the vibration characteristics between the structure and the model. To reduce this error, the in situ data has to
be calibrated with the FE model data by adjusting the geometric and material properties used to model the FE model. These adjustments must be grounded in laws of vibration in order to achieve physically meaningful updated data.

The accuracy of FE model based modal analysis and updating has been demonstrated on a complex and non-linear cable stayed bridge by Hu et al. (2006) using the commercial FE model software, Ansys. Hu demonstrated that a FE model based non-linear model of the Owensboro Bridge over the Ohio River accurately predicted the modal properties of the real structure. The field modal properties were derived from frequency domain analysis of the ambient vibration acceleration at critical sections the Owensboro cable-stayed bridge. The field results yielded stable natural frequencies and operational deflection mode shapes normalized to reference sensors. In their model, the FE model based mode shapes were normalized to unity instead of the mass matrix in order to match the operational modes shapes from the field. Using direct updating technique in which the geometric and material properties of the FE model mesh elements were individually and iteratively changed until a good match of the modal properties were found, Hu et al. were able to establish a baseline FE model of the bridge which is expected to be used for future updating.

The direct updating method used by Hu et al. is limited for applications of the baseline model to predict local damage. The need to revise individual parameters is time exhausting and requires some prior knowledge of existence and location of damage in the real structure-knowledge that might not be available to inspectors. There is also the danger that the resulting updated matrices reproduce the measured structural modal properties exactly but do not generally maintain structural connectivity and the corrections suggested are not always physically meaningful.
Friswell (1995) and Link (1999) used iterative methods of FE model updating to minimize the afore-stated error by using the sensitivity of the modal properties to parameters. Automated iterative methods of model updating are increasingly being used for complex structures such as long-span composite deck and cable-stayed bridges as shown in the work of Wang et. al (1997) and Mordini (2008). Jaishi and Ren (2006) developed an algorithm for sensitivity based FEM model updating by minimizing an objective function which is a residual of the modal flexibility matrix of the real structure and FE model of it. Jaishi and Ren demonstrated good updating results on lab-scale structures with ambient vibration data by applying Guyan-Reduced mass matrix normalization technique to convert the operational deflection shapes to mode shapes.

In the vibration-based FE model updating procedure, a baseline of the structure at the time of initial monitoring is established by changing structural parameters in order to minimize the error between field-observed modal properties and those from the model. The optimization processes lead to non-unique solutions unless a robust method creates a bound of plausible models. In this process, retaining physical connectivity of the model to the real structure is required. The problem of model selection in the context of FE model updating has been explored widely in the literature by Link et al. (2002), Beck and Yuen (2004), Mthembu et al. (2011), and others. The above methods employed the Bayesian selection algorithms which rank optimum models. The Bayesian approach can lead to highly complex model classes which nonetheless lack evidence of physical plausibility. The accuracy of FE model updating has been demonstrated extensively. Hu et al. (2006) demonstrated that a FEM-based nonlinear model of the Owensboro Bridge over the Ohio River accurately predicted the modal properties. Hu iteratively modified many parameters of the bridge until the best match of modal properties was
achieved. This approach can give very accurate matches; however, physical model connectivity is not necessarily guaranteed.

The application of FE updating to civil structures for SHM faces unique challenges compared to mechanical systems. Firstly, unlike mechanical systems which can often be tested under controlled conditions in a lab, civil systems like bridges are often tested in operational conditions under uncertainties. Operational analysis of civil structures can compromise computational flexibility and robustness. For example, vibration-based operational modal analysis is limited to un-scaled mode shapes which limit its damage sensitivity. Furthermore, civil systems are built through manual labor without the precision achieved with industrial craftworks. As such, the design specifications can often deviate from the ‘as-built’ conditions. Because of these unique challenges, refined FE model updating while retaining physical connectivity is difficult. In addition, optimization techniques in model updating have computational constraints related to efficient sampling of design points and 3D curve-fitting algorithms. This constraint has limited the efficiency of this procedure for routine adoption as a damage detection technique. The application of FE model updating to civil structures is especially computationally expensive because Civil structures are relatively large, with a high number of uncertain parameters when tested under operational conditions.

The above challenges may be overcome with multi-scale data which combine (1) dynamic data such as acceleration which represents structural characteristics, (2) static data such as tilt, strain and displacement which represent structural behaviors, and (3) ambient data such as temperature variation and humidity. Multi-scale data enables model updating to simultaneously limit the size of the candidate model class and enables the updating procedure to retain physical
meaning. Using this procedure, the optimum model classes can be ranked according to evidence of plausibility.

For comprehensive SHM, a dense array of sensors is desirable because structural damage is intrinsically local. Some researchers deployed dense arrays of wired sensors on bridge structures such as Tsing-Ma Bridge, Ting Kau Bridge, Khap Shui Mun Bridge (Wong, 2004), Owensboro Bridge (Hu et al, 2006), Stonecutters Bridge (Ni et al, 2011), and others. However, the logistical difficulties of installing wired sensors as well as the high costs of such systems have limited installation of high density of sensors. With multi-faceted knowledge about the structural behavior of a bridge, a sparse array of sensors can also be used to capture the signature properties that are likely to change with damage (Raich, 2011). Also, a refined 3D FE model based on as-built design specifications can be used to select optimum spatial arrangement of sensors to capture the behavior of a bridge.

In this research, a multi-scale model updating strategy is developed for a skewed in-service highway bridge. The model updating strategy is based on measured multi-scale data to enhance model convergence and overcome the problem of physical connectivity. Firstly, an initial 3D FE model of the bridge is built in ANSYS® (2012) based on as-built design specifications. A sensitivity-based optimization algorithm is implemented to select a class of optimum models that satisfy the modal properties measured in the field. In this process, the support conditions, material properties of the bridge are dynamically updated on an objective function which minimizes the error between the natural frequencies and mode shapes of the FE model and the bridge. The temperature-induced tilt of the bridge is used to create a bound of plausible optimum models using a ranking algorithm. It is demonstrated that a sparse array of
vibration sensors can provide an accurate update of global health status of a complex bridge if the monitoring system is supplemented with static sensors.

### 3.2 Test bed description

The test bed is a skewed flyover highway bridge which connects I-84 East to I-91 North, in downtown Hartford, Connecticut (Figure 40). This bridge is a multi-span, continuous, double steel box-girder bridge with composite deck. The whole bridge consists of a total of nine spans with three sets of continuous spans which are simply supported. The bridge is supported by tall circular reinforced concrete columns. The two spans of the middle continuous span was instrumented as part of a bridge monitoring project initiated by the University of Connecticut in collaboration with the Connecticut Department of Transportation since 2001 (DeWolf, 2009).

![Figure 40 Flyover Bridge in Hartford, CT (Google Earth®).](image)

A data acquisition system and sensors were deployed on the test bed bridge. The bridge monitoring system consists of 8 piezoelectric accelerometers with ±1.5g peak amplitude and bandwidth of 0.01 to 1200 Hz, 8 RTD temperature sensors, and 6 tilt meters with a ± 3 degree range.
Figure 41 shows the schematic of sensors; AV, AH, and T represent vertical accelerometer, horizontal accelerometer and tilt meter, respectively. The acceleration measurements are collected at a 91.91 Hz rate for 30 seconds for triggers which exceed a threshold of 0.0095 g, which represents bridge vibration level induced by heavy trucks. Additional details of the truck parameter detection algorithm can be found in DeWolf (2009). The uniaxial tilt meter measures the rotation with respect to the longitudinal axis of the bridge. Temperature sensors are located at mid-span of Span 4, and co-located with the array of tilt meters 1, 2, 4, 5 and 6. Tilt and temperature measurements were collected every 10 minutes, synchronically. Of the eight temperature transducers: two measure the concrete deck temperature, four measure the temperature in the steel box tubs, and the other two record ambient temperatures both inside and outside of the box (see Figure 42). The temperature transducers were originally placed in this configuration to study the effects of temperature differential on tilt in the cross-section. Using the deployed systems, field data collection began on this bridge in 2001. The sensors are shown in Figure 43 to Figure 45.
Figure 41 Schematic of accelerometers and tilt meters (DeWolf, 2009).

Figure 42 Schematic of temperature sensors (DeWolf, 2009).
Figure 43 Tilt meter

Figure 44 Accelerometer
Given the relative newness of the bridge, major damage was not expected to be detected through modal analysis. Nonetheless, visual inspection had noted that there were significant cracks in Piers 4 and 5. Thus, it was of interest to build a numerical model of the bridge in order to enable predictive studies of the bridge as well as possible diagnosis of the noted cracks.

3.3 Statistical system identification

A comprehensive system identification of the test bed bridge based on long-term measurement data has been conducted to set up baseline information for model updating. Nearly 700 data samples measured in November 2001 and in November 2003 have been employed for this research, because the most dramatic temperature variations and structural behavior changes occurred in the winter months. Figure 46 shows a sample time history record representing 30 seconds of vibration. The higher temperature differential in the winter occurs because the sun is
lower in the winter and tends to heat both the upper deck and lower steel box girders, while the concrete deck effectively shades the steel box girders in the summer. Statistical analysis of the modal properties was carried out for data sets measured in each year under similar temperatures since temperature varied from 50 °F to 102 °F.

Figure 46  Sample acceleration at all the accelerometers (mg) vs. time (seconds).
Figure 47  Sample frequency spectrum at all the accelerometers vs frequency (Hz).

An automated modal parameter extraction algorithm in Matlab was built and used on all data samples. The algorithm used the peak-picking method to extract the natural frequencies and mode shapes. In addition, four system identification techniques were applied to the field data using a graphical user-interface for operational modal analysis (Beskhyroun, 2009) in order to increase the confidence in output-only modal properties. These methods included two frequency domain identification methods: the peak picking (PP) method, the frequency domain decomposition (FDD), and two time domain techniques; Eigensystem realization algorithm (ERA) and the stochastic subspace identification (SSI) (Beskhyroun, 2009). Table 5 shows a sample natural frequency extracted using the 4 different modal analysis methods. Extensive statistical analysis of natural frequencies was carried out on the extracted modal properties. It was observed that beyond 3 Hz, modal properties extracted did not exhibit repeatability because a low-pass filter with a bandwidth of 3 Hz was used in the data acquisition system. As a result, the first three reliable natural frequencies were identified. All three modes represent the 1st
bending mode in each span; however, the phase between the adjacent spans alternates. The modal properties determined using the 4 methods were variable up to 18%.

The statistical operational deflection shapes (ODS) of the first three modes with means and standard deviations are shown in Figure 48. In this plot, the fundamental mode shape of Span 5 is consistent for over 100 data samples, while that of Span 4 showed great variation. Since the fundamental mode of the bridge is directly related to its static deformed shape, it can be deduced that the static response of the bridge to heavy trucks interact with its transient response in Span 4 but have negligible steady-state effects in the spans ahead of the travel direction. From this observation, it can be deduced that for medium-to-large span bridges, reference sensor at each span is appropriate for extracting high fidelity mode shapes. Further, the results show that the trigger sensor’s span is likely to show greater error in the mode shape due to coupling of dynamic and static effects. The possible reason for the variability in mode shapes is that the mode shapes determined by the output-only system identification are not scaled in terms of mass without input measurement.

<table>
<thead>
<tr>
<th>SI method</th>
<th>Mode 1</th>
<th>Mode 2</th>
<th>Mode 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>PP</td>
<td>1.5708</td>
<td>2.0195</td>
<td>2.8722</td>
</tr>
<tr>
<td>FDD</td>
<td>1.5708</td>
<td>2.0195</td>
<td>2.8722</td>
</tr>
<tr>
<td>ERA</td>
<td>1.5056</td>
<td>1.8966</td>
<td>2.9866</td>
</tr>
<tr>
<td>SSI</td>
<td>1.6418</td>
<td>1.6493</td>
<td>3.3881</td>
</tr>
</tbody>
</table>
(a) Mode 1: 1.57Hz.

(b) Mode 2: 2.01Hz.

(c) Mode 3: 2.87 Hz.

Figure 48 Operational deflection shapes.
3.4 Initial finite element model

3.4.1 Geometry modeling

The bridge was drawn in the Autodesk Civil/Structures suite to take advantage of the robust options for road geometry reconstruction and 3-Dimensional drawing capacities. The workflow for the bridge 3D drawing is shown in Figure 49 below.

![Figure 49 Workflow for Flyover 3D model drawing](image)

Ansys provides a robust platform for parametric modeling. Ansys Classic (Mechanical APDL) is a parameter based FEM platform. The new Ansys Workbench platform also provides a platform that allows the material, geometric and results of a model to be parameterized.
3.4.2 *Parametric modeling*

An initial model of the instrumented simply-supported spans from Pier 3 to Pier 6 of the bridge has been constructed in ANSYS (2012) with the engineering design drawing specifications (see Figure 50). Given the highly curved nature of the bridge, finite elements with 6 DOF were required to accurately represent structural behavior. ANSYS solid elements with 6 DOF (SOLID 186) were used for the deck and steel boxes. The piers were modeled as rigid given the much higher stiffness of the piers relative to the superstructure. ANSYS mass elements (Mass 21) and stiffness element (Stiff 21) are made from lumped mass and rotational inertia. In addition, the 6 DOF stiffness of the piers is located at their center of gravity and center of rigidity, respectively.

![Figure 50 ANSYS® Model of Spans 4, 5 and 6](image)

The design specifications for the boundary conditions (BCs) were used to model connections between parts of the ANSYS model. Piers 3, 5 and 6 were connected to the composite deck by multi-rotational bearings. Further, expansion joints were installed in the deck at these locations to permit deck translation. Pier 4 was connected to the composite deck by similar multi-rotational bearings; however, longitudinal translation was constrained. In addition, deck translation was prevented due to absence of expansion joints. To model the stated BCs, the parts of the bridge were connected together by appropriate joint elements. Lagrangian-based
contact elements connect the concrete deck and steel boxes through a bonded connection to simulate the composite action at the interface. Multi-rotational bearings connect the superstructure to the piers. The bearings are modeled by equivalent 6 DOF springs. The i-th bearing stiffness matrix is modeled as a 6x6 diagonal stiffness matrix with 3 translational DOFs and 3 rotational DOFs:

\[
\begin{bmatrix}
k_x & \cdots & 0 \\
k_y & k_z & \\
0 & \cdots & k_i \\
\end{bmatrix}
\]

where, \(k_x, k_y, k_z\) are the translational stiffness in \(x, y, z\) directions, respectively. \(r_x, r_y, r_z\) are the rotational stiffness in \(x, y, z\) directions, respectively. \(i\) is the index of each bearing. Finally, the foundations of all piers were modeled as fixed connections.

The initial values of stiffness are selected by trial and error by quantitatively expressing the stated design BCs. For example, the initial stiffness for fully constrained vertical translation is set as \(10^7\) lb/in, a high value which leads to insignificant displacement. On the other hand, the initial stiffness for fully released longitudinal rotation was set at 0 lb/in. Based on this initial assumption, the initial FE model was built with the specifications of Table 6. Figure 51 shows the three resonant modes of the initial model that were used for the updating procedure. Figure 52 shows the tilt profile of the FE model under service load.
Table 6 Initial BCs.

<table>
<thead>
<tr>
<th>BCs</th>
<th>Pier 3</th>
<th>Pier 4</th>
<th>Pier 5</th>
<th>Pier 6</th>
</tr>
</thead>
<tbody>
<tr>
<td>Longitudinal translation (lb/in)</td>
<td>10</td>
<td>10^6</td>
<td>10</td>
<td>10</td>
</tr>
<tr>
<td>Longitudinal rotation (lb in/degree)</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Vertical stiffness (lb/in)</td>
<td>10^7</td>
<td>10^7</td>
<td>10^7</td>
<td>10^7</td>
</tr>
</tbody>
</table>
Figure 51 Bending modes from the initial FE model used for updating
3.5 Multi-scale data approach for model updating

3.5.1 Theoretical background of optimization

The model updating was carried as an optimization process using the Ansys Optimization module (ANSYS, 2012). The module implements a multi-objective genetic algorithm (MOGA). The MOGA algorithm is a goal-based, weighted, aggregation-based optimum design ranking technique. The algorithm has a combined single objective function created from a weighted aggregate sum of multiple individual objective functions. The individual objective functions are functions of the input and output parameters. The candidate designs are ranked by ascending magnitudes of the values of the combined objective function. The candidate designs are collectively termed the optimum model class, assuming non-unique solutions to the combined objective function.
Given \( n \) input parameters, \( m \) output parameters, and their individual objective functions, the combined objective function, \( \Phi \), is given by the following:

\[
\Phi = \sum_{i=1}^{n} w_i \left( \frac{|x_t - x_i|}{x_u - x_l} \right) + \sum_{j=1}^{m} w_j \left( \frac{|y_t - y_j|}{y_{max} - y_{min}} \right)
\]  

(4)

where, \( x_i \) is the current input parameter, \( x_t \) is the target input parameter, \( y_j \) is the current output parameter, \( y_t \) is the target output parameter. \( x_l \) and \( x_u \) are the lower and upper bounds of the input parameters respectively. \( y_{min} \) and \( y_{max} \) correspond to the lower and upper bounds of the output parameters, respectively. \( w_i \) and \( w_j \) are the weights for optimization for indices of the input and output parameters, \( i \) and \( j \), respectively. The weights can be determined based on the importance such that,

\[
w_i, w_j = \begin{cases} 
1.00 & \text{if the importance is higher} \\
0.66 & \text{if the importance is average} \\
0.33 & \text{if the importance is lower}
\end{cases}
\]  

(5)

The importance factor represents the level of confidence in the input and output parameters. A higher value of importance is placed on the parameters with relative lower uncertainties. For example, the output parameters, the modal properties, have relatively low uncertainties, while the input parameters such as the BCs and material properties are higher uncertainties. The vibration-based FE updating creates an initial model class which lacks evidence of physical plausibility.

To retain physical connectivity, a second step, called a ranking procedure, is used to create a bound of plausible optimum models by filtering the model class through constraint
functions. The plausibility criterion is quantitatively expressed as ‘hard’ or ‘soft’ constraints on the combined objective functions. Mathematically, a ‘hard’ constraint rejects solutions to the objective function which do not absolutely satisfy a given constraint, whereas a ‘soft’ constraints ranks solutions to the objective function according to their aggregate error from a given tolerance factor. In this research, hard and soft constraint equations are built from the static properties of the bridge in order to provide plausibility evidence for the optimum models created from vibration-based FE updating.

The constraint condition for ranking optimum models is given by functions as follows:

\[ \Psi_k - f(\varphi) < tol \] (6)

\[ \Psi_l - f(\varphi) < Tol \] (7)

where, \( \Psi_i, \Psi_j \) are field data used to constraint the optimum model class. For the flyover bridge, the soft constraints are the original tilt profile under service loads and the hard constraints are the daily temperature induced tilt; \( \varphi \) are the input parameters; \( tol \) and \( Tol \) are tolerance levels where \( Tol > tol \), \( k \) and \( l \) are the indices for soft and hard constraints, respectively.

### 3.5.2 Model updating Parameters for MOGA

The initial FE model has been updated using a two stage MOGA. The input parameters for updating were the elastic modulus and the density of the concrete deck and piers and the BCs of the superstructure. The material properties of concrete were uncertain given the varying field conditions of in-situ concrete. The steel material properties of the box girders were not considered, because the girders were factory manufactured; hence, it was assumed that the design specifications and the as-built properties were similar. For the BCs, the translational and rotational bearing stiffness were chosen as input parameters, because the stiffness of the bearings
is uncertain. The design specifications described stated that all bearings were multi-rotational bearings. Also, all piers under study have expansion-joints to permit translation of the deck, except at Pier 4. The exact level of longitudinal translation of the Piers 3, 5 and 6 was uncertain. In addition, observed torsional hairline cracks observed in Pier 4 and Pier 5 created uncertainties about the as-built rotational stiffness.

Given the computational costs of multi-parameter optimization, a preliminary sensitivity analysis was used to limit the number of the updating parameters. The preliminary sensitivity analysis showed that the BCs were more critical for correct structural behavior compared to material properties. This is because the material properties had uniform effects on global behavior while the effects of different BCs were localized. Figure 53 shows the absolute value of sensitivity of natural frequencies of the first three modes for a 10% increase in the Young’s modulus, concrete density, and Poisson ratio. The natural frequencies have a positive proportionality constant with the Young’s modulus and a negative proportionality constant with the concrete density. A local sensitivity plot of the BCs in Figure 54 shows the absolute percentage change in natural frequencies for the range of stiffness considered while the other BCs stay constant. The sample local sensitivity plot showed that different pier stiffness had different effects on the natural frequencies. The preliminary sensitivity analysis further showed that transverse displacement should be limited at the end piers (Pier 3 and Pier 6) but allowed at the inner piers in order to prevent unreasonable transverse displacements while preventing excessive transverse stiffness of the highly curved bridge. After the sensitivity analysis, the updating parameters for BCs were reduced to the longitudinal translation stiffness at Pier 3, 4 and 6 as well as the rotational stiffness about the longitudinal axis of the bridge at Pier 4 and Pier 5. The longitudinal stiffness was critical to the correct tilt behavior as observed in the field.
first objective function sought to use the modal properties and static deformation data to update the BCs. The model was further optimized by updating the global material properties.

![Figure 53 Sensitivity of frequency to concrete properties (%).](image)

![Figure 54 Local Sensitivity (x10^-5) of frequency to BCs.](image)

The bound of the input parameters were determined in accordance with practicality of structural behavior. For the material properties, a 10% error of the design specifications was used given the relatively newness of the bridge and the lack of evidence of serious fatigue damage. The upper bound for longitudinal and transverse bearing stiffness was set to 10% of the vertical
stiffness capacity of the bearings according to common engineering specifications. The lower bound was set at 0 for roller support condition. The upper limit of the rotational stiffness was set according to 10% of the rotational stiffness needed to prevent rotations in excess of the AASHTO moderate rotation limit of 1.5% radian (AISI, 2006).

The output parameters were selected based on the field data. The output parameters for updating included the first three bending frequencies determined from the system identification of the field data. The tilt under dead load and tilt induced by daily temperature changes were also used as soft and hard constraints respectively as detailed above. The tilt profile of the bridge under self-weight was recorded at intervals of 10 minutes from 2001 to 2005. The tilts recorded in 2001 were consistently at -0.58, 0, +0.05, -0.25 degree for Pier 3, 4, 5 and 6, respectively (Figure 55). Field results showed that global temperature increase resulted in increased negative tilt at the piers. The highest temperature increase of 30°F from a baseline of 72°F was recorded in the winter. A field experiment (Virkler, 2004) focused on thermal effects on tilts showed that Pier 3 recorded a 0.02 degree negative tilt as girder temperature rose by approximately 20°F (Figure 56). These tilt behavior were simulated on the FE model in a combined thermal structural analysis and modal analysis.
Figure 55 Trend of tilt (degree) over 4 years (DeWolf, 2009)

Figure 56 Trend of diurnal temperature-induced tilt (Virkler, 2004)
3.5.3 **Design of Experiments**
A design of experiments (DOE) step is needed in the MOGA procedure to sample design points used for the curve fitting in the optimization algorithm. A DOE is a method of sampling the input parameters for an optimization procedure in order to efficiently select an optimum number of sample design points for simulation purposes. An optimum size of design points is needed to accurately represent the design space while minimizing the computational cost. A FE model with $n$ parameters and $m$ design points for each parameter requires $n^m$ sample design points in order to enable accurate curve fitting of the sensitivity curve required in optimization (Ansys, 2012). The computational costs and trade-offs of the optimization procedure for large-scale civil structures are concentrated in the DOE stage. The application of FE model updating to civil structures is especially computationally expensive. Civil structures are relatively large, with a high number of uncertain parameters when tested under operational conditions. For example, for a 64-bit Windows® platform with 8GB of RAM and 2.66GHz processor, a DOE computation with ‘p’ design points requires ~0.83$p$ minutes of computational time. Under these conditions, a model with 10 updating parameters (a modest parameter size for a large scale structure) and 3 elements in each parameter bound requires $10^3$ minutes (16 hours) for a single optimization run.

3.5.4 **Sensitivity-based model updating results**
The optimization step for selecting optimum BCs is outlined below. The input parameters for the first optimization function were the stiffness of the bearings at the piers. The objective function is defined based on the input parameters and updating parameters chosen from the previous section in Eqn (4).

In Eqn. (4), $x_i$ is the initial BCs based on design specifications, $x_t$ is the a sample design point obtained from the DOE sampling procedure; $y_j$ is natural frequencies as at a given design
point and \( y_i \) is the field natural frequencies; \( x_l \) and \( x_u \) are the lower and upper values of the input parameters respectively; \( y_{\text{min}} \) and \( y_{\text{max}} \) correspond to lower and upper bounds of the output parameters obtained from the statistical analysis of the field properties, respectively. For the output parameters, a higher weight (1.00) was assigned to the second mode due to its relatively lower variance while lower weights (0.66) were assigned to Modes 1 and 2 due to their greater variance. Equal weight (1.00) was assigned to all input parameters.

The threshold for error of the selected optimum models was 10% for the output parameters. Sample optimum models are shown in Table 7. The table shows the input parameters \( k_{lx} \) represents the translational stiffness at a given Pier, ‘x’; \( k_{rx} \) for rotational stiffness about the longitudinal axis at a given Pier, ‘x’. The corresponding output parameters are represented by \( f_x \) for natural frequencies for a given mode, ‘x’. For all sample optimum models the maximum error of the frequencies was 7%.

Table 7 Results of updated stiffness of the bearings.

<table>
<thead>
<tr>
<th>Optimum model</th>
<th>kl3 (lb/in)</th>
<th>kl4 (lb/in)</th>
<th>kl6 (lb/in)</th>
<th>kr4 (lb/in)</th>
<th>kr5 (lb/in)</th>
<th>f1 (Hz)</th>
<th>f2 (Hz)</th>
<th>f3 (Hz)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Model 1</td>
<td>0</td>
<td>10^6</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1.52</td>
<td>2.06</td>
<td>2.93</td>
</tr>
<tr>
<td>Model 2</td>
<td>10^6</td>
<td>10^6</td>
<td>10^6</td>
<td>10^6</td>
<td>10^6</td>
<td>1.64</td>
<td>2.06</td>
<td>3.07</td>
</tr>
<tr>
<td>Model 3</td>
<td>10^4</td>
<td>10^6</td>
<td>10^4</td>
<td>10^8</td>
<td>10^6</td>
<td>1.60</td>
<td>2.03</td>
<td>3.01</td>
</tr>
</tbody>
</table>

The results of the unconstrained optimization step were assessed for physical meaning before applying the constraint conditions to rank the optimum models. Sensitivity curves were created from the design points during the DOE. The sensitivity analysis showed that the first
bending mode was the most sensitive to the BCs being updated, followed by the 3rd bending mode. The 1st bending mode’s relative amplitude was sensitive to the longitudinal stiffness of Pier 4. If the longitudinal stiffness is higher than $10^4$ at Pier 4, the relative amplitude of the mode shape of the 1st bending mode was comparable with the field results; below this threshold, the relative amplitude was incomparable. The magnitude of the natural frequency of the 1st bending mode was the most sensitive to the longitudinal stiffness of the end piers (Piers 3 and 6). Increased longitudinal stiffness at the end piers resulted in proportional increase in natural frequency if other factors are held constant. The natural frequency of the 3rd bending mode varied in a similar pattern as the 1st bending mode. In addition, the relative amplitude of the mode shape of the 3rd bending mode was sensitive to rotational stiffness at all supports. Increased rotational stiffness about the longitudinal axis resulted in torsional behavior introduced into the 3rd bending mode. The 2nd bending mode was consistent in both natural frequency and relative amplitude of the mode shape for a wide range of updating parameters.

3.5.5 Ranking algorithm

The constraint conditions were applied to the optimum model class to rank the models. Figure 57 is a flow chart of the ranking algorithm. When a set of optimum model class is determined from the initial optimization procedure, the hard constraint will be evaluated in terms of a pre-defined tolerance value. If a model does not satisfy the hard constraint, the model will be eliminated from the candidates. The models satisfying the hard constraint will be considered as plausible models. Finally, the model classes satisfying the soft constraint will be ranked as three tiers of optimum models, ‘most likely’, ‘likely’, and ‘less likely’. In the model ranking step, the temperature induced tilts was considered as a ‘hard’ constraint given the higher reliability of the diurnal tilt behavior relative to the tilt under service load. The initial tilt profile under dead load was
considered a ‘soft’ constraint given that the initial tilt under service load is influenced by uncertainty factors such as camber and the quality of instrumentation.

![Flow chart of ranking algorithm.](image)

Based on the ranking algorithm, the optimum stiffness parameters for modal properties are summarized in Table 8. Positive tilts are shown in bracket; otherwise, tilts are negative. The tilt of piers are represented by tilt $x$ at a given Pier, ‘$x$’. Using a hard constraint tolerance, $|tol| \leq 0.5\Psi$ for the temperature-induced tilt (tilt$T$) at the piers, the sample optimum model class is reduced to Models 2 and 3 from Table 7. Using a soft constraint tolerance, $0^\circ \leq Tol \leq 0.1^\circ$, Models 2 and 3 are ranked according to their proximity to the field recorded tilt of $0.05^\circ$. A lower rank implies greater confidence in the plausibility of the model. A rank of $W$ implies that the model does not satisfy the hard constraint. Therefore, the most plausible model was successfully ranked
based on the hard and soft constraints. Still, the updated model based on BCs alone resulted in the first three natural frequencies with errors more than 5%.

Table 8 Results of updated stiffness at the bearings.

<table>
<thead>
<tr>
<th>Optimum model</th>
<th>tilt3</th>
<th>tilt4</th>
<th>tilt5</th>
<th>tilt6</th>
<th>tiltT</th>
<th>f1</th>
<th>f2</th>
<th>f3</th>
<th>Rank</th>
</tr>
</thead>
<tbody>
<tr>
<td>Model 1</td>
<td>-0.82</td>
<td>-0.89</td>
<td>0.85</td>
<td>0.75</td>
<td>0.08</td>
<td>1.52</td>
<td>2.06</td>
<td>2.93</td>
<td>W</td>
</tr>
<tr>
<td>Model 2</td>
<td>-0.43</td>
<td>-0.30</td>
<td>-0.23</td>
<td>-0.24</td>
<td>-0.01</td>
<td>1.64</td>
<td>2.06</td>
<td>3.07</td>
<td>2</td>
</tr>
<tr>
<td>Model 3</td>
<td>-0.50</td>
<td>-0.10</td>
<td>-0.10</td>
<td>-0.30</td>
<td>-0.05</td>
<td>1.60</td>
<td>2.03</td>
<td>3.01</td>
<td>1</td>
</tr>
<tr>
<td>Field</td>
<td>-0.58</td>
<td>0.00</td>
<td>-0.05</td>
<td>-0.25</td>
<td>-0.05</td>
<td>1.57</td>
<td>2.01</td>
<td>2.87</td>
<td>N/A</td>
</tr>
</tbody>
</table>

3.5.6 Updating Material Properties
The FE model was further refined globally based on the sensitivity of the output parameters to the material properties. Given the uniform effect of the material properties on the modal properties as shown in Figure 53, it was determined that the further refinement of the optimum model could be achieved by updating the material properties. The input parameters are the Young’s modulus, concrete density, and Poisson ratio for the first three modes. The target output parameters in the objective functions are the natural frequencies and temperature-induced tilts. The Young’s modulus is reduced by 5% in the updated model. The concrete density was increased by 2.7%. These modifications resulted in closely matching natural frequencies but negligible effects on the tilts. The reduced Young’s modulus can result from the effects of deterioration and fatigue. The increased concrete density is a compensation for reinforcement which will tend to increase the effective density of the reinforced concrete. These input parameter updates are consistent with model updating practice. In general, for model updating of
civil structures, Zhang et al. (2001) allowed variation of up to 150% for uncertain parameters. Jaishi and Ren (2007) also allowed parameter variations up to 45%. In the final optimum model, the natural frequencies of the updated baseline and of the field measurement matched closely for all three modes within 2% error bound. Table 9 and Figure 58 summarize the optimum results of the updating procedure to establish the baseline global material properties.

Table 9 Updated material properties for Baseline model

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Design specs</th>
<th>Updated baseline</th>
</tr>
</thead>
<tbody>
<tr>
<td>$E = f (f_c')$ (ksi)</td>
<td>3834 ksi ($f_c' = 4$ ksi)</td>
<td>3737 ksi ($f_c' = 3.8$ ksi)</td>
</tr>
<tr>
<td>Density (lb/ft$^3$)</td>
<td>150</td>
<td>152</td>
</tr>
<tr>
<td>Poisson ratio</td>
<td>0.18</td>
<td>0.18</td>
</tr>
</tbody>
</table>

Figure 58 Comparison of natural frequencies (Hz).
3.6 Application on causes of torsional cracks

Significant torsional cracks were observed in Piers 4 and 5 prior to installation of the monitoring system. Virkler and DeWolf (2004) hypothesized that inadequate room for temperature expansion at Piers 3 and 6 induced superstructure rotations that put the piers in torsion. In addition, a permanent out-of-plane tilt ranging from 0.001° to 0.005° was recorded at the piers after the winter of 2003. The baseline FE model was used for combined static and thermal analyses to diagnose the torsional cracks in order to verify the hypothesis. The validity of the FE model’s simulations was monitored by performing modal analysis on the pre-loaded bridge and comparing the extracted modal parameters with the field results.

Curved bridges are designed with torsion to withstand significant rotational deformations by connecting the substructure and superstructure by high-strength multi-rotational bearings such as pot, disk and spherical bearings. The existence of torsional cracks can be caused by damaged rotational bearings which can become locked under high load. Such locked bearings would restrict free rotation of the deck under thermal expansion, and induce torsion in the piers. Another likely cause of torsional cracks in the piers is ultimate shear loading at the bearings. Figure 59 shows the illustration of the ultimate shear loading effects. \( F_{\text{ext}} \) and \( F_{\text{int}} \) are the shear loadings of the external bearing and the internal bearing, respectively. If shear loading at a given bearing support is offset from the shear center, it could introduce torsion into the pier. In addition, the pre-stressing in curved bridges introduces radial forces that create additional stresses and deformations if neglected in the design of restraints.
A static analysis of the bridge under self-weight revealed that bearing locking was a likely cause of the torsional cracks. The undamaged model assumed functional rotational bearings at the Piers 4 and 5, hence there was no torsion transferred to the Piers. However, given the fixed longitudinal translation at Pier 4, significant shear forces are transferred from the superstructure to the Pier 4 through the pot bearings.

An approximate estimation of the ultimate torsional torque loading, $T$, for the circular pier is given by:

$$T = \frac{\pi \tau c^3}{2}$$  \hspace{1cm} (9)

where, $c$ is the radius of pier column and $\tau$ is the shear capacity of the pier cap. An approximate estimation of the ultimate torsional torque loading showed that a torque of 5,500 kip-ft is required to induce shear cracks. Given the results of loading in Table 10, it is shown that shear effects are not enough to induce the torsional cracks. The extreme temperature variations observed in the field were simulated on the FE model to compare their effects with service loads. The results showed insignificant torsional loading on the bridge. Finally, the bearings at Piers 4 and 5 were alternatively locked to simulated locked pot bearings. The results for the locked bearing at Pier 4, shown in Table 10, were significantly increased torque induced at the bearings.
A modal analysis of the model under locked bearing showed a slight reduction in the first natural frequency but no effects in the higher frequencies. Given the high torque induced by a single locked bearing, it is reasonable to assume that partial locking can accelerate torsional cracks induced by ambient and gravity load effects. Hence it is likely that one or more of the bearings at Pier 4 and/or Pier 5 become locked under high rotational deformation and induce torsional loading in the piers.

Table 10 Forces and moments under service and locked bearing

<table>
<thead>
<tr>
<th>Pier</th>
<th>Transverse shear (kip)</th>
<th>Longitudinal shear (kip)</th>
<th>Torque (kip-ft.)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Unlocked</td>
<td>Locked</td>
<td>Unlocked</td>
</tr>
<tr>
<td>Pier 4</td>
<td>10</td>
<td>11</td>
<td>-3</td>
</tr>
<tr>
<td>Pier 5</td>
<td>-3.3</td>
<td>-3.4</td>
<td>11</td>
</tr>
</tbody>
</table>

3.7 Conclusion
Multi-scale data approach is developed to improve accuracy and ensure physical connectivity for FE model updating of an in-service highway bridge. This approach was implemented on a skewed highway bridge monitored with a sparse array of accelerometers, tilt-meters and temperature transducers. The static and dynamic properties of the bridge were used to iteratively derive a baseline model of the bridge. An initial optimum model class was created through an optimization procedure using the uncertain properties of the bridge and the modal properties obtained from extensive field monitoring. A ranking algorithm based on static tilt data obtained from field monitoring was used to assess the plausibility of the initial optimum model classes. The close match and correlation between field results and model results demonstrated the strong
potential of the multi-scale data for updating FE models of civil structures. Finally, the different hypotheses of the cause of torsional cracks observed in the field were tested by simulation on the updated model. Throughout these processes, modal properties of the bridge were tracked in order to maintain physical connectivity between the model results and field results. It was demonstrated that FEM simulation and experimental modal analysis can serve as mutual validation tools.
4 Chapter 4 Concluding Remarks

Computer-aided SHM tools were developed in this research. Hybrid methods involving experimental techniques and numerical techniques were combined. An autonomous framework for SHM was demonstrated on a wireless platform using a graphical user interface for continuous monitoring of structures. Real time information of structural state was provided in a user-friendly platform. More importantly, the problem of data inundation during SHM was mitigated by providing routines to extract a summary of structural state. Finally, a proposal for model selection during numerical FE updating of civil structures was demonstrated on an in-service highway bridge. The evidence-based approach for model selection enables the user of complex numerical models to retain physical meaning in their results.
5 References


[26] National Instruments (2010), A Bridge Health Monitoring System Based on NI Hardware and Software.


% fclose(comy); fclose('all'); clear all; close all; clc

function varargout = ConImote2(varargin)

% CONIMOTE2 MATLAB code for ConImote2.fig
% CONIMOTE2, by itself, creates a new CONIMOTE2 or raises the existing
% singleton*.
% H = CONIMOTE2 returns the handle to a new CONIMOTE2 or the handle to
% the existing singleton*.
% CONIMOTE2('CALLBACK',hObject,eventData,handles,...) calls the local
% function named CALLBACK in CONIMOTE2.M with the given input arguments.
% CONIMOTE2('Property','Value',...) creates a new CONIMOTE2 or raises the
% existing singleton*. Starting from the left, property value pairs are
% applied to the GUI before ConImote2_OpeningFcn gets called. An
% unrecognized property name or invalid value makes property application
% stop. All inputs are passed to ConImote2_OpeningFcn via varargin.
% *See GUI Options on GUIDE's Tools menu. Choose "GUI allows only conimote2
% instance to run (singleton)".
%
% See also: GUIDE, GUIDATA, GUIHANDLES

% Edit the above text to modify the response to help ConImote2

% Last Modified by GUIDE v2.5 14-Apr-2012 21:12:37

% Begin initialization code - DO NOT EDIT

gui_Singleton = 1;
gui_State = struct('gui_Name', mfilename, ...        'gui_Singleton', gui_Singleton, ...
                      'gui_OpeningFcn', @ConImote2_OpeningFcn, ...
                      'gui_OutputFcn', @ConImote2_OutputFcn, ...
                      'gui_LayoutFcn', [], ...,        'gui_Callback', []);
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end

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

% --- Executes just before ConImote2 is made visible.

function ConImote2_OpeningFcn(hObject, eventdata, handles, varargin)
% This function has no output args, see OutputFcn.
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% varargin   command line arguments to ConImote2 (see VARARGIN)

% Choose default command line output for ConImote2
handles.output = hObject;

% Update handles structure
guidata(hObject, handles);

% UIWAIT makes ConImote2 wait for user response (see UIRESUME)
% uiwait(handles.figure1);
axes(handles.heading)
imshow('lablogo.png')

axes(handles.axes1)
imshow('start.png')

axes(handles.axes13)
imshow('funknown.png')

axes(handles.axes14)
imshow('funknown.png')

axes(handles.axes15)
imshow('funknown.png')

axes(handles.axes17)
imshow('funknown.png')

t = timer('TimerFcn', @(x,y)disp('Starting'), 'StartDelay', 3);
start(t)
wait(t)
delete(t)

% --- Outputs from this function are returned to the command line.
function varargout = ConImote2_OutputFcn(hObject, eventdata, handles)
% varargout  cell array for returning output args (see VARARGOUT);
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

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

% --- Executes on button press in pushbutton1.
function pushbutton1_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

%***************GUI/Command line interaction*************************
handles.output = hObject;

% Update handles structure
guidata(hObject, handles);

%textInEditor = get(handles.edit1,'string')   % Read user input text in text editor and enter in text output
%set(handles.text1,'string',textInEditor)

%set(handles.popupmenul,'Value', 4)  
% ComValue = get(handles.popupmenul,'Value');
% set(handles.text1,'string',ComValue)

%%% Syntax for getting Pop-up menu options
% index = get(hObject,'Value');    % index location of pop-up menu
% strlist = get(hObject,'String'); % Get all choices
%strlist(index);                   % Get chosen's name

%fclose(comy); delete(comy);

% Close existing figures and command prompts
dos('TASKKILL /IM cmd.exe');

%Clear all ports somehow open;
newobjs = instrfind;
if length(newobjs) == 0
    size(newobjs);
else
    fclose(newobjs);
    delete(newobjs);
    clear newobjs
end
delete(instrfindall);

% Now: Creating data port
daportIndex = get(handles.popupmenul,'Value');
daportList  = get(handles.popupmenul,'string');
%daport      = char(daportList(daportIndex,:))
daport      = strtrim(char(daportList(daportIndex,:)))
%sez = size(daport);

% Text file for data
ofilename = 'outfile.txt';
%ofilename  = char(get(handles.edit1,'string'));

eval(['batmsg = strcat(''autocomm -n -o ',ofilename,' ',daport,'''');']); %creates a string variable "batmsg".
batfid = fopen('gui.bat','w'); %gui.bat is created by us, 'w' means it can be written to. also, '.bat' is a batch file which can be implemented in dos
fwrite(batfid, batmsg); fclose(batfid); pause(0.1);
%dos('start "" /min gui.bat'); %dos(from windows) command http://www.computerhope.com/starthlp.htm

%%% data port Initialization
dos('start "" /min gui.bat')

%%%**********Major Troubleshooting proble************
%%% Need to wait for dataport to finish execution by adding a timer which
%%% waits Magically solve 'Serial Port initialization error problem

t = timer('TimerFcn',@(x,y)disp('DataPort ready!'),'StartDelay',1);
start(t)
wait(t)
delete(t)

% Now: Creating debug port
debugIndex = get(handles.popupmenu3,'Value');
debugList = get(handles.popupmenu3,'string');
debugport = strtrim(char(debugList(debugIndex,:)))%Oh God I spent a week on this, white spaces after COM port name;
%eval(['debugport = strcat('' ',debugport,''');'])
%debugsize1 = size(debugport);

% Get Node IDS
% rsnodes = char(get(handles.edit2,'string')) % Make this multiple nodes in future

% NodeIDS from Table uitable1
thedata = get(handles.uitable1,'data');
% size(thedata)
[row,col,v] = find(thedata);
nodeids = unique(v);
lengthnids = length(nodeids);

%Initialize
nodestrings = {num2str(nodeids(1,1))};
nodeChar(1,:) = [char(nodestrings(1,1)), ' '];
%size(nodeChar)
NodeID = nodeChar(1,:);
%size(NodeID)

for i = 2:length(nodeids)
    nodestrings = {num2str(nodeids(i,1))} %num2str(nodeids(i,:))
    nodeChar = [char(nodestrings), ' ']

NodeID = [NodeID,nodeChar]

% Finally, NodeID
%size(NodeID)
rsnodes = strtrim(NodeID)
%size(rsnodes)

%******************************************************************************
imotedebuglog = fopen('imotedebug.txt','w'); %Initialize debugging file % Permission 'a'
append, 'w' discard existing
% fidlog = fopen('guilogfile.txt','a');
%         timestamp = datestr(now,0);
%         logmsg= sprintf('
%s %s
', timestamp,'ListNodes');
%         fprintf(fidlog,'%s
',rmsg);
%******************************************************************************

%WAKE UP SNOOZE-ALARM ENABLED LEAF NODE
y = instrfind;
if isempty(y)
    else
        fclose(y);
        delete(y);
    end
% Calc sheet
wakeClock =1;
wakeline =1;
wakeCallsize = num2str(length(nodeids));

%wakefinish = 'BluSH>Successfully woke up 1 node(s): 21'; % Have to code this depending on Number of Leaf Nodes
eval(['wakefinish = strcat(''BluSH>Successfully woke up 1 node(s): ','rsmnodes,'''');']);
eval(['wakedcallednodes = strcat(''BluSH>Successfully woke up ','wakeCallsize,' node(s): ','rsmnodes,'''');']);

wakeupAll    = '- All nodes awake';
notAwake = 'Bad command';
oneNodenotWake = 'Failed to wake up any nodes.';

wakeCount = 0;
wakeMax   = 20;  % Change this after experimenting with cycles

% CREATE SERIAL PORTS
comy = serial(debugport);
set(comy,'BaudRate', 115200, 'DataBits', 8, 'FlowControl', 'none', 'Parity', 'none', 'StopBits', 1, 'Terminator', 'CR/LF','Timeout',15);  % 5 seconds Timeout was inadequate, 10 Ok, but give room because of 10 seconds interval for sleep/wake cycle
fopen(comy);
eval(['setrsnodes = strcat('WakeUp ',rsnodes,''');']); fprintf(comy, setrsnodes); pause(0.01);

t = timer('TimerFcn',@(x,y)disp('All is well!'),'StartDelay', 1);
start(t)
wait(t)
delete(t)

% WAKE UP SNOOZE-ALARM ENABLED LEAF NODE
while wakeClock
    waketext = char(fgetl(comy))
    % set(handles.debugimote2,'string',waketext);
    % fprintf(imotedebuglog,'%s
',waketext); %fprintf(fidlog,'%s
',rmsg);
    if strncmp(char(waketext),char(wakefinish),length(wakefinish));
        wakeClock =0
        wakeFeedback = 'Sleeping beauty wakes up, Yay!!! '
        % set(handles.debugimote2,'string',wakeFeedback);
        % fprintf(imotedebuglog,'%s
',wakeFeedback);
    elseif strncmp(char(waketext),char(wakeCallednodes),length(wakeCallednodes));
        wakeClock =0
        wakeFeedbackAll = 'All requested nodes awake '
        % set(handles.debugimote2,'string',wakeFeedbackAll);
        % fprintf(imotedebuglog,'%s
',wakeFeedbackAll);
    elseif strncmp(char(waketext),char(wakeAll),length(wakeAll));
        wakeClock =0
        wakeFeedbackAll = 'All nodes awake(Lots of them) '
        % set(handles.debugimote2,'string',wakeFeedbackAll);
        % fprintf(imotedebuglog,'%s
',wakeFeedbackAll);
    elseif (wakeCount > wakeMax) || (strncmp(char(waketext),char(notAwake),length(notAwake))) || (strncmp(char(waketext),char(oneNodenotWake),length(oneNodenotWake)));
        wakeClock =0
        wakeFeedbackCount = 'Too many attempts at WakeUp!!! '  % This can be further refined
        % set(handles.debugimote2,'string',wakeFeedbackCount);
        % fprintf(imotedebuglog,'%s
',wakeFeedbackCount);
        dos('TASKKILL /IM cmd.exe');
    end
end

t = timer('TimerFcn',@(x,y)disp('All is well!'),'StartDelay', 15);
start(t)
wait(t)
delete(t)

%Clear all ports somehow open;
% newobjs = instrfind;
% if length(newobjs) == 0
%    size(newobjs);
% else
%    fclose(newobjs);
%    delete(newobjs);
%    clear newobjs
% end
% delete(instrfindall);
fclose all
set(handles.debugimote2,'string',waketext);
fprintf(imotedebuglog,'%s
',waketext);
fclose(comy); delete(comy); clear comy;
ConImote2('pushbutton1_Callback',hObject,eventdata,guidata(hObject))

else
  wakeClock =1;
  %rsline = rsline +1;
end
  wakeCount = wakeCount +1;
end

fclose(comy); delete(comy); clear comy;

%************************************************************
% SetRSNodes

comy = serial(debugport);
%comy = serial('COM4');
set(comy,'BaudRate', 115200, 'DataBits', 8, 'FlowControl', 'none', 'Parity', 'none', 'StopBits', 1, 'Terminator', 'CR/LF', 'Timeout', 1);
fopen(comy);
%setrsnodes = 'SetRSNodes 21';
eval(['setrsnodes = strcat(''SetRSNodes '',rsnodes,'''');']); fprintf(comy, setrsnodes);
pause(0.01);
fprintf(imotedebuglog,'%s
',setrsnodes);
fclose(comy); delete(comy); clear comy;

%SetRSParameters and Clear meta data

% A. Sensor channels x,y,z,external device in channel 4

statusVal1 = get(handles.checkbox2,'Value');
maxVal1 = get(handles.checkbox2,'Max');
statusVal2 = get(handles.checkbox3,'Value');
maxVal2 = get(handles.checkbox3,'Max');
statusVal3 = get(handles.checkbox4,'Value');
maxVal3 = get(handles.checkbox4,'Max');
statusVal4 = get(handles.checkbox5,'Value');
maxVal4 = get(handles.checkbox5,'Max');

chStatus = [statusVal1 statusVal2 statusVal3 statusVal4];
chMax = [maxVal1 maxVal2 maxVal3 maxVal4];

%checkboxes ={'checkbox2','checkbox3','checkbox4','checkbox5'};
index =1;
chCount = 1;
for i = 1:1:4
if chStatus(1,i) == chMax(1,i)
    sensorCh(index) = num2str(i)
    index = index +1
end
end

sensorCh;

%size(sensorCh) %Check status of channels.

%B Data Samples (1000 works, why doesn't lower samples work? ask about this
dataSamples = char(get(handles.edit3,'string'));

%C Sampling rate of data in frequency units, 'Hz'

rateIndex  = get(handles.popupmenu2,'Value');
rateList   = get(handles.popupmenu2,'string');
rateSample = char(rateList(rateIndex,:));

%D Synchronized sensing or not (Synch =1, Non-synch = 0) %In future i'd
%like to make the default conimote2 synch status, requires using 'Min',not 'Max'.

if get(handles.radiobutton1,'Value') == get(handles.radiobutton1,'Max')
    synchStatus = '1';
else
    synchStatus = '0';
end

%E RemoteSensing Data Block location (Put data in 1st block = clearmetaData
%= 1, find nex available block =0

if get(handles.radiobutton2,'Value') == get(handles.radiobutton2,'Max')
    clearmetaData = '1';
else
    clearmetaData = '0';
end

%F Nfft for finding resolution of spectral frequencies

nfftIndex  = get(handles.nfft,'Value');
nfftList   = get(handles.nfft,'string');
Nfft = char(nfftList(nfftIndex,:));
Nfft = str2num(Nfft);

%Debugging SetRSParameters
comy = serial(debugport);
set(comy,'BaudRate', 115200, 'DataBits', 8, 'FlowControl', 'none', 'Parity', 'none',
'StopBits', 1,'Terminator', 'CR/LF','Timeout',1);
 fopen(comy);
 setrsparameters = ['SetRSParameters ', sensorCh,' ', dataSamples,' ', rateSample,' ',
clearmetaData]
fprintf(comy, setrsparameters); pause(0.01);
fprintf(imotedebuglog,'%s\n',setrsparameters);
fclose(comy); delete(comy); clear comy;

%StartRemoteSensing
comy = serial(debugport);
set(comy,'BaudRate', 115200, 'DataBits', 8, 'FlowControl', 'none', 'Parity', 'none', 'StopBits', 1, 'Terminator', 'CR/LF', 'Timeout',700);
fopen(comy);
startrs = ['StartRemoteSensing ',clearmetaData]; fprintf(comy, startrs); pause(0.01);
fprintf(imotedebuglog,'%s
',startrs);

%Implementation of shmData;

shmclock =1;
rsline =1;
%rsfinish = '- Responsive nodes are 21';
eval(['rsfinish = strcat(''- Responsive nodes are ',rsnodes,'']);']);
badRstext = 'Bad command';
while shmclock
    rstext = char(fgetl(comy))
    %rstext = strtrim(rstext)
    %waketext = char(fgetl(comy))
    %set(handles.debugimote2,'string',rstext);
    %fprintf(imotedebuglog,'%s
',rstext); %fprintf(fidlog,'%s
',rmsg);
    ifstrncmp(char(rstext),char(rsfinish),length(rsfinish));
        shmclock =0
    else
        shmclock =1;
        %rsline = rsline +1;
    end
end

set(handles.debugimote2,'string',rstext);
fprintf(imotedebuglog,'%s
',rstext); %fprintf(fidlog,'%s
',rmsg);

fprintf(comy, '\n'); pause(0.01); %Equivalent to pressing enter in the cygwin bash shell!
fprintf(comy, 'RetrieveData -1 21 ');
eval(['setrsnodes = strcat(''RetrieveData -1 ',rsnodes,'''');']); fprintf(comy, setrsnodes); pause(0.01);

% Give imote enough time to retrieve and write output into directory
retrieveclock =1;
retrieveline =1;
retrievefinish = '- Finished writing output.';
deviceremoved = 'Device Removed;';
while retrieveclock
    retrievetext = char(fgetl(comy))

    set(handles.debugimote2,'string',retrievetext);
    fprintf(imotedebuglog,'%s
',retrievetext); %fprintf(fidlog,'%s
',rmsg);

    if strncmp(char(retrievetext),char(retrievefinish),length(retrievefinish));
        retrieveclock =0
    elseif strncmp(char(retrievetext),char(deviceremoved),length(deviceremoved));
        retrieveclock =0

        set(handles.debugimote2,'string','Device Removed;');

        imageArray =imread('warning.jpg');
        % Switch active axes to the conimote2 you made for the image.
        axes(handles.axes1);
        % Put the image array into the axes so it will appear on the GUI
        imshow(imageArray);
        newobjs = instrfind;

        if length(newobjs) == 0
            size(newobjs);
        else
            fclose(newobjs);
            delete(newobjs);
            clear newobjs
        end
        delete(instrfindall);

        dos('TASKKILL /IM cmd.exe');
        delete('gui.bat')

        pause(1);

        t = timer('TimerFcn',@(x,y)disp('All is not well!'),'StartDelay', 15);
        start(t)
        wait(t)
        delete(t)

    ConImote2('pushbutton1_Callback',hObject,eventdata,guidata(hObject)) % Found it✓ from call back syntax in GUIDE figure editor
        %healthcounter = healthcounter +1;
    else
        retrieveclock =1;
        %retrieveline = retrieveline +1;
    end
end

fclose(comy); delete(comy); clear comy;
dogshow = 'fun all the time !!! '

%% tester
x1 = 1: 10;
x2 = sin(x1);
% axes(handles.axes4);
% plot(x1,x2)

% **************************CLOSE DOS PROMPT ******************************
dos('TASKKILL /IM cmd.exe');
delete('gui.bat')
%dos('TASKKILL /IM cmd.exe');

% Post Processing: Feedback and Plotting *******************************

debugimote2contents = fileread('imotedebug.txt');
set(handles.debugScroll,'string',debugimote2contents);
% outputtextprocessing
% set(hObject,'string','File name')
%Simply calling ff to get last test data and plot it.
pris = dir('outfile*'); % depends on what I call the filename in imote2comm
lastestfile = pris(end); % gives most recent output file
prisdata = importdata(lastestfile.name);
realdata = prisdata.data;

% Data Calibration
offset = 14270.332; % Values from UIUC calibration test
scale  = 7656.57;

samplesize  = str2num(dataSamples);
channelSize  = (length(realdata))/samplesize;
[dataLength,channelNumber]=size(realdata);
% Create matrix of calibrated data for different sensors and their
% subsequent channels
sizeSeries = 1;
for i = 1:1: channelSize
    for j = 2:1:channelNumber
        cleandata(:,sizeSeries)= realdata(samplesize*(i-1)+1:samplesize*i,j);
        sizeSeries = sizeSeries+1;
    end
end
% cleandata = realdata(1:1000,2:end); % To change based on samples requested
 calibrated_data = (cleandata - offset)/scale;

% time array
% fs = 100 % 25, 50, 100, 280;
% Modal Analysis
% post processing

% Call preprocessing
% Input Parameters
%sensors = (channelNumber-1)*channelSize; % To be put in GUI or calculated based on number of Degrees of freedom
%sensors = dataLength/samplesize; % This is necessary compared to algorithm above because sometime data is not retrieved
% input loading
%load av_11_03.mat
% load lumpcolumn;
% data = LumpColumn(:,2:5);

fs0 = str2num(rateSample);

if fs0 == 280 % Corrected April 10th
   fs = 256;
   dt = 1/fs;
   data = calibrated_data;
   data = resample(data,fs,fs0);
   time = 0:dt:dt*(length(data)-1);
elseif fs0 == 100
   fs = 64;
   dt = 1/fs;
   data = calibrated_data;
   data = resample(data,fs,fs0);
   time = 0:dt:dt*(length(data)-1);
elseif fs0 == 50
   fs = 32;
   dt = 1/fs;
   data = calibrated_data;
   data = resample(data,fs,fs0);
   time = 0:dt:dt*(length(data)-1);
% elseif fs0 == 25
%   fs = 16;
%   dt = 1/fs;
%   data = calibrated_data;
%   data = resample(data,fs,fs0);
%   time = 0:dt:dt*(length(data)-1);
else
   fs = 16;
   dt = 1/fs;
   data = calibrated_data;
   data = resample(data,fs,fs0);
   time = 0:dt:dt*(length(data)-1);
end
%data = resample(data,256,fs); % This should be automated % Done April 10th

%data = data(:,2:2:end);
[sthing1,sensors] = size(data);
%fs = 256; % Correct this - Correct April 10th.
% Take moment to close dos prompt
%dos('TASKKILL /IM cmd.exe');

%figure;plot(time,data);grid

% fs = 512; %512 and 1024, 1024 and 1024 % sampling frequency and Nfft
% dt = 1/fs;
% time = 0:dt:dt*(length(data)-1);
% Nfft =1024;
delta_f = fs/Nfft; % Frequency resolution needed for finding modal freq.

% ****************** Normal Frequency behavior **********************
% Healthy modal frequencies % USER INPUT
% mode1 = 1.5:delta_F:2.5;
% mode2 = 12:delta_F:14;
% mode3 = 33.5:delta_F:35.5;
% mode4 = 0:delta_F:50*delta_F;

% ******** Mode Buttons ********************************************
if get(handles.mode1check,'Value') == get(handles.mode1check,'Max')
    model1Button = 1;
else
    model1Button = 0;
end

if get(handles.mode2check,'Value') == get(handles.mode2check,'Max')
    model2Button = 1;
else
    model2Button = 0;
end

if get(handles.mode3check,'Value') == get(handles.mode3check,'Max')
    model3Button = 1;
else
    model3Button = 0;
end

if get(handles.mode4check,'Value') == get(handles.mode4check,'Max')
    model4Button = 1;
else
    model4Button = 0;
end

healthydata = get(handles.uitable2,'data');
mode11 = healthydata(1,1);
mode12 = healthydata(1,2);
mode21 = healthydata(2,1);
mode22 = healthydata(2,2);
mode31 = healthydata(3,1);
mode32 = healthydata(3,2);
mode41 = healthydata(4,1);
mode42 = healthydata(4,2);

% Healthy modal frequencies % USER INPUT
mode1 = mode11:delta_F:mode12;
mode2 = mode21:delta_F:mode22;
mode3 = mode31:delta_F:mode32;
mode4 = mode41:delta_F:mode42;

healthy_modes = [mode1,mode2,mode3,mode4];

numNodes = channelSize;
%Healthy mode shape
healthyModes = get(handles.uitable3,'data');
healthyModes = healthyModes(:,1:numNodes);
%Healthy mode shape Amplitudes
healthyModes1 = healthyModes(1,:)
healthyModes2 = healthyModes(3,:)
healthyModes3 = healthyModes(5,:)
healthyModes4 = healthyModes(7,:)
%Healthy mode shape Phases
healthyPhase1 = healthyModes(2,:)
healthyPhase2 = healthyModes(4,:)
healthyPhase3 = healthyModes(6,:)
healthyPhase4 = healthyModes(8,:)

%*********************START CHANNEL************************
for ii =1:sensors
    [plotcsdxy(:,ii), plotFxy]= cpsd(data(:,ii),data(:,ii),[],[],Nfft,fs); %CSD of the
    output and input.
    [plotcsdxyb(:,ii),Fxyb]= cpsd(data(:,ii),data(:,1),[],[],Nfft,fs);
    plotfrf_x = plotcsdxy./plotcsdxyb;
    %figure;plot(Fxy, abs(frf_xy(:,ii)));grid;xlim([0 200]);
    %figure;plot(Fxy, angle(frf_xy(:,ii)));grid;xlim([0 200]);
end
%**********************END CHANNEL**************************

% numNodes = channelSize;
numChannels = channelNumber-1;
channelID = 1;
%starTer = 1;
for i =1:1:numChannels % this is the number of channels for node;
    numSensors =1; % to create the CPSD matrix(size is number of channels per node x

number of nodes
    for j = 1: numChannels: numChannels*(numNodes -1) + i
        [csdxx(:,numSensors),Fxy] = cpsd(data(:,j),data(:,j),[],Nfft/2,Nfft,fs); %[csdxx(:,ii),Fxx]= cpsd(data(:,ii),data(:,ii),[],[],[],[],fs);
        [csdxy(:,numSensors),Fxy] = cpsd(data(:,j),data(:,i),[],Nfft/2,Nfft,fs);
        numSensors = numSensors +1;
    end
    csdxxMatrix(:,:,i) = csdxx(:,:,i);
    csdxyMatrix(:,:,i) = csdxy(:,:,i);
    %channelID = channelID + numChannels;
end
%figure;plot(Fxy, csdxx);grid;xlim([.5 fs/2]);grid on
%figure;plot(Fxy, csdxxMatrix(:,:,1));grid;xlim([.5 fs/2]);grid on

%*******************************************************************************

x = Fxy;
compared_freq =[]; % Initialize common frequency vector;
for n_Ch =1:1:numChannels
    n=1;
    for j = 1:1:numNodes % This has to be changed depending on number of sensors
        y = abs(csdxxMatrix(:,j,n_Ch));
        for i=2:1:(length(y)-1) %for i=2:1:(length(y)-1)/2
            if y(i-1) < y(i) && y(i+1) < y(i)
                z(n,j) = find(y==y(i)); % indices of 'peak' magnitudes
                n=n+1;
            % else
            %     n=n;
            end
        end
        n=1;
    end
    size_z = size(z)
end

% Find indices that are common to all sensors
    c= z(:,1);

% % Extracted modal frequencies based on "peak" algorithm
for i = 2:1:numNodes
    c = intersect(c,z(:,i));
end
modal_frequencies = Fxy(c);
% mag_modes = abs(plotfrf_xy); %To be restored
% mode_shapes = mag_modes(c,:); %To be restored
% *************Mode Shape Check******************************
%Mode Shapes and Frequencies
mag_modes = abs(csdxxMatrix(:,:,n_Ch));
phase_modes = angle(csdxyMatrix(:,:,n_Ch));
phase_modes = mod(round(phase_modes),2);

% *********************************
%Compare healthy modal frequencies and extracted modal frequencies
compared_freq= intersect(modal_frequencies,healthy_modes)
compared_freq1 = intersect(compared_freq,model1)
compared_freq2 = intersect(compared_freq,model2);
compared_freq3 = intersect(compared_freq,model3);
compared_freq4 = intersect(compared_freq,model4);

mode_shapes1(1,1) = 0;
if length(compared_freq1) ~= 0
    for numCompFreq1= 1:1:length(compared_freq1)
        fIndex1= find(compared_freq1(numCompFreq1) == Fxy);
        mode_shapesInit1 = mag_modes(fIndex1,:); %mode_shapesInit
        for locate1 = 2:1:numNodes
            if(mode_shapesInit1(locate1) >= mode_shapesInit1(locate1-1))
                mode_shapes1(1,locate1) = 1;
            else
                mode_shapes1(1,locate1) = 0;
            end
        end
        % mode_shapes1
        phase_shapes1 = phase_modes(fIndex1,:)
        if (min(mode_shapes1 == healthyModes1)) == 1 && (min(phase_shapes1 == healthyPhase1))==1
            healthStatus1(numCompFreq1) = 1;
        else
            healthStatus1(numCompFreq1) = 0;
        end
    end
    if sum(healthStatus1) >= 1
        prefFreq_index1 = find(max(mode_shapesInit1) == mag_modes);
        normalFreq1_index = max(prefFreq_index1);
        normalFreq1 = Fxy(normalFreq1_index)
        healthModel1 = 1;
    else
        healthModel1 = 0;
        normalFreq1 = 0
    end
else
    healthMode1 = 0;
    normalFreq1 = 0
end

mode_shapes2(1,1) = 0;

if length(compared_freq2) ~= 0
    for numCompFreq2 = 1:1:length(compared_freq2)
        fIndex2 = find(compared_freq2(numCompFreq2) == Fxy);
        mode_shapesInit2 = mag_modes(fIndex2,:);

        for locate2 = 2:1:numNodes
            if(mode_shapesInit2(locate2) >= mode_shapesInit2(locate2-1))
                mode_shapes2(1,locate2) = 1;
            else
                mode_shapes2(1,locate2) = 0;
            end
        end
    end

    phase_shapes2 = phase_modes(fIndex2,:);

    if (min(mode_shapes2 == healthyModes2)) == 1 && (min(phase_shapes2 == healthyPhase2)) == 1
        healthStatus2(numCompFreq2) = 1;
    else
        healthStatus2(numCompFreq2) = 0;
    end
end

if sum(healthStatus2) >= 1
    preFreq_index2 = find(max(mode_shapesInit2) == mag_modes);
    normalFreq2_index = max(preFreq_index2);
    normalFreq2 = Fxy(normalFreq2_index);
    healthMode2 = 1;
else
    healthMode2 = 0;
    normalFreq2 = 0
end

else
    healthMode2 = 0;
    normalFreq2 = 0
end

mode_shapes3(1,1) = 0;
if length(compared_freq3) ~= 0
    for numCompFreq3 = 1:1:length(compared_freq3)
        fIndex3 = find(compared_freq3(numCompFreq3) == Fxy);
        mode_shapesInit3 = mag_modes(fIndex3,:);
for locate3 = 2:1:numNodes
    if (mode_shapesInit3(locate3) >= mode_shapesInit3(locate3-1))
        mode_shapes3(1,locate3) = 1;
    else
        mode_shapes3(1,locate3) = 0;
    end
end

phase_shapes3 = phase_modes(fIndex3,:);

if (min(mode_shapes3 == healthyModes3)) == 1 && (min(phase_shapes3 == healthyPhase3)) == 1
    healthStatus3(numCompFreq3) = 1;
else
    healthStatus3(numCompFreq3) = 0;
end

if sum(healthStatus3) >= 1
    preFreq_index3 = find(max(mode_shapesInit3) == mag_modes);
    normalFreq3_index = max(preFreq_index3);
    normalFreq3 = Fxy(normalFreq3_index);
    healthMode3 = 1;
else
    healthMode3 = 0;
    normalFreq3 = 0
end

else
    healthMode3 = 0;
    normalFreq3 = 0
end

mode_shapes4(1,1) = 0;

if length(compared_freq4) ~= 0
	for numCompFreq4 = 1:1:length(compared_freq4)
	    fIndex4 = find(compared_freq4(numCompFreq4) == Fxy);
	    mode_shapesInit4 = mag_modes(fIndex4,:);
	    for locate4 = 2:1:numNodes
	        if (mode_shapesInit4(locate4) >= mode_shapesInit4(locate4-1))
	            mode_shapes4(1,locate4) = 1;
	        else
	            mode_shapes4(1,locate4) = 0;
	        end
	    end
	    phase_shapes4 = phase_modes(fIndex4,:);
end

if (min(mode_shapes4 == healthyModes4)) == 1 && (min(phase_shapes4 == healthyPhase4)) == 1
    healthStatus4(numCompFreq4) = 1;
else
    healthStatus4(numCompFreq4) = 0;
end

if sum(healthStatus4) >= 1
    preFreq_index4 = find(max(mode_shapesInit4) == mag_modes);
    normalFreq4_index = max(preFreq_index4);
    normalFreq4 = Fxy(normalFreq4_index);
    healthMode4 = 1;
else
    healthMode4 = 0;
    normalFreq4 = 0
end

else
    healthMode4 = 0;
    normalFreq4 = 0
end

% % %
% % % for numCompFreq= 1:1:length(compared_freq1)
% % %    %a = find(compared
% % %    fIndex(numCompFreq) = find(compared_freq1(numCompFreq) == Fxy);
% % %    mode_shapes = mag_modes(fIndex,:);
% % %    phase_shapes = phase_modes(fIndex,:);
% % % end
% % %
% % %
% % %
healthChannels(n_Ch,:) = [healthMode1,healthMode2,healthMode3,healthMode4]
healthFreq(n_Ch,:) = [normalFreq1,normalFreq2,normalFreq3,normalFreq4]
end

% healthFreq
% healthChannels
% % %Mode properties post-processing

statusMode1 = healthChannels(:,1);
statusMode2 = healthChannels(:,2);
statusMode3 = healthChannels(:,3);
statusMode4 = healthChannels(:,4);

statusFreq1 = healthFreq(:,1);
statusFreq2 = healthFreq(:,2);
statusFreq3 = healthFreq(:,3);
statusFreq4 = healthFreq(:,4);

% Steps for searching for the healthy natural frequency for different channels
statusMet1 = 1;
counterF1 = 1;
while statusMet1 && counterF1 <= numChannels
    if (statusModel(counterF1) == 1)
        frequency1 = statusFreq1(counterF1);
        % fprintf(histogram1,'%d
',frequency1);
counterF1= counterF1+1;
    statusMet1 = 0;
    else
        counterF1= counterF1+1;
        frequency1 =[];
    end
end

%%% ******************END******************

% Steps for searching for the healthy natural frequency for different channels
statusMet2 = 1;
counterF2 = 1;
while statusMet2 && counterF2 <= numChannels
    if (statusMode2(counterF2) == 1)
        frequency2 = statusFreq2(counterF2);
        % fprintf(histogram1,'%d
',frequency1);
counterF2= counterF2+1;
    statusMet2 = 0;
    else
        counterF2= counterF2+1;
        frequency2 =[];
    end
end

%%% ******************END******************

% Steps for searching for the healthy natural frequency for different channels
statusMet3 = 1;
counterF3 = 1;
while statusMet3 && counterF3 <= numChannels
    if (statusMode3(counterF3) == 1)
        frequency3 = statusFreq3(counterF3);
        % fprintf(histogram1,'%d
',frequency1);
counterF3= counterF3+1;
    statusMet3 = 0;
    else
        counterF3= counterF3+1;
        frequency3 =[];
    end
end

%%% ******************END******************
% Steps for searching for the healthy natural frequency for different channels
statusMet4 = 1;
counterF4 = 1;
while statusMet4 && counterF4 <= numChannels
    if (statusMode4(counterF4) == 1)
        frequency4 = statusFreq4(counterF4);
        % fprintf(histogram1,'%d
',frequency1);
        counterF4 = counterF4+1;
        statusMet4 = 0;
    else
        counterF4 = counterF4+1;
        frequency4 = [];
    end
end

%%% ******************END******************

%Plots
axes(handles.axes4);
hold off
for i = 1:1:numChannels
    plot(Fxy, csdxxMatrix(:,i));grid;xlim([1 fs/2]);grid on;%legend('show')
    hold all
    % plot(Fxy, abs(csdxx));grid;xlim([0 fs/2]);grid on; legend('show')
end
%legend('show')
%plot(plotFxy, abs(plotcsdxx));grid;xlim([0 fs/2]);

axes(handles.axes5);
plot(time, data(:,1:end));grid on; %legend('show')
%figure;plot(Fxy, abs(csdxyb));grid;xlim([0 200]);
% figure;plot(Fxy, db(abs(csdxx)));grid;xlim([0 200]);
% figure;plot(Fxy, angle(csdxyb));grid;xlim([0 200]);

% **********************************************************************
% Compare extracted "peak" frequencies for all locations to ensure
% consistency of "peaking" since it seems there are some 'insignificant'
% modes that are appearing in the algorithm.

%******PLOT FREQUENCY STATISTICS**************************************

histogram1 = fopen('histogram1.txt','a');
histogram2 = fopen('histogram2.txt','a');
histogram3 = fopen('histogram3.txt','a');
histogram4 = fopen('histogram4.txt','a');
% This part should be troubleshoot for closely spaced frequency system
% where there might be multiple frequencies in the range

if statusMet1 == 0
    fprintf(histogram1,'%d\n',frequency1);
    set(handles.text6,'string',num2str(frequency1));
    status_freq1 =1;
    axes(handles.axes13);
    imshow('fgood.png')
else
    status_freq1 =0;
    set(handles.text6,'string',num2str(frequency1));
end

if statusMet2 == 0
    fprintf(histogram2,'%d\n',frequency2);
    set(handles.text7,'string',num2str(frequency2));
    axes(handles.axes14);
    imshow('fgood.png')
else
    status_freq2 =0;
    set(handles.text7,'string',num2str(frequency2));
end

if statusMet3 == 0
    fprintf(histogram3,'%d\n',frequency3);
    set(handles.text8,'string',num2str(frequency3));
    axes(handles.axes15);
    imshow('fgood.png')
else
    status_freq3 =0;
    set(handles.text8,'string',num2str(frequency3));
end

if statusMet4 == 0
    fprintf(histogram4,'%d\n',frequency4);
    set(handles.text9,'string',num2str(frequency4));
    axes(handles.axes17);
    imshow('fgood.png')
else
    status_freq4 =0;
    set(handles.text9,'string',num2str(frequency4));
end

%SUMMARY of frequency status
status_freq = [status_freq1, status_freq2, status_freq3, status_freq4]; % used for health check
sumStatusfreq = status_freq1 + status_freq2 + status_freq3 + status_freq4;
% Actual Plot of frequency time history

freqhistory1 = importdata('histogram1.txt');
freqhistory2 = importdata('histogram2.txt');
freqhistory3 = importdata('histogram3.txt');
freqhistory4 = importdata('histogram4.txt');

meanM1 = mean(freqhistory1);
meanM2 = mean(freqhistory2);
meanM3 = mean(freqhistory3);
meanM4 = mean(freqhistory4);

meanModes = [meanM1, meanM2, meanM3, meanM4];

stdM1 = std(freqhistory1);
stdM2 = std(freqhistory2);
stdM3 = std(freqhistory3);
stdM4 = std(freqhistory4);

stdModes = [stdM1, stdM2, stdM3, stdM4];

axes(handles.axes7);
hold off
bar(meanModes, 'g'); hold on
errorbar(meanModes, stdModes, 'r', 'linestyle', 'none');

axes(handles.axes8);
hold off
plot(1:1:length(freqhistory1), freqhistory1, '-ro', 'MarkerFaceColor', 'r', 'MarkerSize', 2);
hold on
plot(1:1:length(freqhistory2), freqhistory2, '-go', 'MarkerFaceColor', 'g', 'MarkerSize', 2);
hold on
plot(1:1:length(freqhistory3), freqhistory3, '-bo', 'MarkerFaceColor', 'b', 'MarkerSize', 2);
hold on
plot(1:1:length(freqhistory4), freqhistory4, '-ko', 'MarkerFaceColor', 'y', 'MarkerSize', 2);

legend('f1', 'f2', 'f3', 'f4', 0) % This should be corrected depending on size of frequency histories being studied

% Setting Health Status criteria

% Initialize
healthStatus = 0;

% Check if frequency buttons have been clicked
sumButtons = modelButton + mode2Button + mode3Button + mode4Button;
statusButtons = [mode1Button, mode2Button, mode3Button, mode4Button]; % used to check health status

if sumButtons == 0
    strictCheck = 0;
else
    strictCheck = 1;
end

% Summary of buttons pressed

% Criteria Check

if strictCheck == 1
    if min(statusButtons <= status_freq) == 1
        healthStatus = 1;
    else
        healthStatus = 0;
    end
end

if healthStatus == 0         %&& healthcounter < 6
    scReamNow1 = 'There is a strict mode check requirement and it was not met!'
    set(handles.debugimote2, 'string', scReamNow1);

    imageArray = imread('warning.jpg');
    axes(handles.axes1);
    % Put the image array into the axes so it will appear on the GUI
    imshow(imageArray);

    %ConImote2('axes1_CreateFcn', hObject, eventdata, guidata(hObject)) % Turn on alarm

    sign

    t = timer('TimerFcn', @(x,y) disp('All is not well!'), 'StartDelay', 15);
    start(t)
    wait(t)
    delete(t)

    % Signal Stopping
    % if kofi == 1
    %     scream = 'we scream icecream lol'
    % else
    if get(handles.stop, 'userdata') == 1
        % Set the cancel pushbutton user data to default
        set(handles.stop, 'userdata', 0);

        % Update handles structure if you want to update the data in your Gui
        % guidata(hObject, handles);
        return;
    else
ConImote2('pushbutton1_Callback', hObject, eventdata, guidata(hObject)) % Found it from callback syntax in GUIDE figure editor
end
    healthcounter = healthcounter + 1;
    end
else
    scReamNow1 = 'Yay, there is a strict mode check requirement and it was met!'
    set(handles.debugimote2, 'string', scReamNow1);
    % Read in image
    imageArray = imread('greenlight.jpg');
    % Switch active axes to the conimote2 you made for the image.
    axes(handles.axes1);
    % Put the image array into the axes so it will appear on the GUI
    imshow(imageArray);

    health_status = 'All is well'    % start timer in Matlab
    t = timer('TimerFcn', @(x, y) disp('All is well!'), 'StartDelay', 60);
    start(t)
    wait(t)
    delete(t)

    if get(handles.stop, 'userdata') == 1
        % Set the cancel pushbutton user data to default
        set(handles.stop, 'userdata', 0);

        % Update handles structure if you want to update the data in your GUI
        guidata(hObject, handles);
        return;
    else

        ConImote2('pushbutton1_Callback', hObject, eventdata, guidata(hObject))
    end
end

else %
    if max(healthFreq) <= 0        %& healthcounter < 6
        % scReamNow2 = 'aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa'
        scReamNow2 = 'No mode check requirement, no modes are normal!'
        set(handles.debugimote2, 'string', scReamNow2);

        imageArray = imread('warning.jpg');
        % Switch active axes to the conimote2 you made for the image.
        axes(handles.axes1);
        % Put the image array into the axes so it will appear on the GUI
        imshow(imageArray);

        % ConImote2('axes1_CreateFcn', hObject, eventdata, guidata(hObject)) % Turn on alarm
        sign
t = timer('TimerFcn',@(x,y)disp('All is not well!'),'StartDelay', 15);
start(t)
wait(t)
delete(t)

% Signal Stopping
% if kofi ==1
%   scream = 'we scream icecream lol'
% else
if get(handles.stop, 'userdata') == 1
    % Set the cancel pushbutton user data to default
    set(handles.stop, 'userdata', 0);

    % Update handles structure if you want to update the data in your Gui
    % guidata(hObject, handles);
    return;
else
    ConImote2('pushbutton1_Callback',hObject,eventdata,guidata(hObject)) % Found it from call back syntax in GUIDE figure editor
    %healthcounter = healthcounter +1;
    % end
    else
    %ConImote2('axes2_CreateFcn',hObject,eventdata,guidata(hObject)) % Turn on green light
    scReamNow2 = 'Yay, no strict mode check requirement, some or all modes are normal!'    
    set(handles.debugimote2,'string',scReamNow2);

    % Read in image
    imageArray =imread('greenlight.jpg');
    % Switch active axes to the conimote2 you made for the image.
    axes(handles.axes1);
    % Put the image array into the axes so it will appear on the GUI
    imshow(imageArray);

health_status = 'All is well'    %start timer in Matlab
    t = timer('TimerFcn',@(x,y)disp('All is well!'),'StartDelay', 60);
    start(t)
    wait(t)
    delete(t)

    if get(handles.stop, 'userdata') == 1
        % Set the cancel pushbutton user data to default
        set(handles.stop, 'userdata', 0);

        % Update handles structure if you want to update the data in your Gui
        % guidata(hObject, handles);
        return;
else

    ConImote2('pushbutton1_Callback',hObject,eventdata,guidata(hObject))

end
end
end

%**************************************************************************
%**************************************************************************
%**************************************************************************
%***************************************END END END OF COMMENT OUT

% --- Executes on button press in checkbox1.
function checkbox1_Callback(hObject, eventdata, handles)
% hObject    handle to checkbox1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hint: get(hObject,'Value') returns toggle state of checkbox1

% --- Executes on slider movement.
function slider1_Callback(hObject, eventdata, handles)
% hObject    handle to slider1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'Value') returns position of slider
%        get(hObject,'Min') and get(hObject,'Max') to determine range of slider

% --- Executes during object creation, after setting all properties.
function slider1_CreateFcn(hObject, eventdata, handles)
% hObject    handle to slider1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: slider controls usually have a light gray background.
if isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor', [.9 .9 .9]);
end

function edit1_Callback(hObject, eventdata, handles)
% hObject    handle to edit1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% Hints: get(hObject,'String') returns contents of edit1 as text
%        str2double(get(hObject,'String')) returns contents of edit1 as a double

% --- Executes during object creation, after setting all properties.
function edit1_CreateFcn(hObject, eventdata, handles)

% hObject    handle to edit1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

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

% --- Executes on button press in togglebutton1.
function togglebutton1_Callback(hObject, eventdata, handles)

% hObject    handle to togglebutton1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hint: get(hObject,'Value') returns toggle state of togglebutton1

% --- Executes on selection change in listbox1.
function listbox1_Callback(hObject, eventdata, handles)

% hObject    handle to listbox1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: contents = cellstr(get(hObject,'String')) returns listbox1 contents as cell array
%        contents{get(hObject,'Value')} returns selected item from listbox1

% --- Executes during object creation, after setting all properties.
function listbox1_CreateFcn(hObject, eventdata, handles)

% hObject    handle to listbox1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: listbox controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end
% --- Executes on selection change in popupmenu1.
function popupmenu1_Callback(hObject, eventdata, handles)
% hObject    handle to popupmenu1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: contents = cellstr(get(hObject,'String')) returns popupmenu1 contents as cell array
%        contents{get(hObject,'Value')} returns selected item from popupmenu1

% --- Executes on selection change in popupmenu2.
function popupmenu2_Callback(hObject, eventdata, handles)
% hObject    handle to popupmenu2 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: contents = cellstr(get(hObject,'String')) returns popupmenu2 contents as cell array
%        contents{get(hObject,'Value')} returns selected item from popupmenu2

% --- Executes during object creation, after setting all properties.
function popupmenu1_CreateFcn(hObject, eventdata, handles)
% hObject    handle to popupmenu1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

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

set(hObject,'string','COM1|COM2|COM3|COM4|COM5|COM6|COM7|COM8|COM9|COM10|COM11|COM12|COM13|COM14|COM15|COM16')  % set options for drop-down menu

% --- Executes during object creation, after setting all properties.
function popupmenu2_CreateFcn(hObject, eventdata, handles)
% hObject    handle to popupmenu2 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

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

end
set(hObject,'string','25|50|100|280|')

% --- Executes during object creation, after setting all properties.
function text1_CreateFcn(hObject, eventdata, handles)
    % hObject    handle to text1 (see GCBO)
    % eventdata  reserved - to be defined in a future version of MATLAB
    % handles    empty - handles not created until after all CreateFcns called
    set(hObject,'string','File name')

% --- Executes on selection change in popupmenu3.
function popupmenu3_Callback(hObject, eventdata, handles)
    % hObject    handle to popupmenu3 (see GCBO)
    % eventdata  reserved - to be defined in a future version of MATLAB
    % handles    structure with handles and user data (see GUIDATA)
    % Hints: contents = cellstr(get(hObject,'String')) returns popupmenu3 contents as cell array
    %        contents{get(hObject,'Value')} returns selected item from popupmenu3

% --- Executes during object creation, after setting all properties.
function popupmenu3_CreateFcn(hObject, eventdata, handles)
    % hObject    handle to popupmenu3 (see GCBO)
    % eventdata  reserved - to be defined in a future version of MATLAB
    % handles    empty - handles not created until after all CreateFcns called
    % Hint: popupmenu controls usually have a white background on Windows.
    %       See ISPC and COMPUTER.
    if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor'))
        set(hObject,'BackgroundColor','white');
    end
    set(hObject,'string','COM1|COM2|COM3|COM4|COM5|COM6|COM7|COM8|COM9|COM10|COM11|COM12|COM13|COM14|COM15|COM16') % set options for drop-down menu

function edit2_Callback(hObject, eventdata, handles)
    % hObject    handle to edit2 (see GCBO)
    % eventdata  reserved - to be defined in a future version of MATLAB
    % handles    structure with handles and user data (see GUIDATA)
    % Hints: get(hObject,'String') returns contents of edit2 as text
    %        str2double(get(hObject,'String')) returns contents of edit2 as a double

% --- Executes during object creation, after setting all properties.
function edit2_CreateFcn(hObject, eventdata, handles)
  % hObject    handle to edit2 (see GCBO)
  % eventdata  reserved - to be defined in a future version of MATLAB
  % handles    empty - handles not created until after all CreateFcns called

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

% --- Executes on button press in checkbox4.
function checkbox4_Callback(hObject, eventdata, handles)
  % hObject    handle to checkbox4 (see GCBO)
  % eventdata  reserved - to be defined in a future version of MATLAB
  % handles    structure with handles and user data (see GUIDATA)

  % Hint: get(hObject,'Value') returns toggle state of checkbox4

% --- Executes on button press in checkbox3.
function checkbox3_Callback(hObject, eventdata, handles)
  % hObject    handle to checkbox3 (see GCBO)
  % eventdata  reserved - to be defined in a future version of MATLAB
  % handles    structure with handles and user data (see GUIDATA)

  % Hint: get(hObject,'Value') returns toggle state of checkbox3

% --- Executes on button press in checkbox2.
function checkbox2_Callback(hObject, eventdata, handles)
  % hObject    handle to checkbox2 (see GCBO)
  % eventdata  reserved - to be defined in a future version of MATLAB
  % handles    structure with handles and user data (see GUIDATA)

  % Hint: get(hObject,'Value') returns toggle state of checkbox2

% --- Executes on button press in checkbox5.
function checkbox5_Callback(hObject, eventdata, handles)
  % hObject    handle to checkbox5 (see GCBO)
  % eventdata  reserved - to be defined in a future version of MATLAB
  % handles    structure with handles and user data (see GUIDATA)

  % Hint: get(hObject,'Value') returns toggle state of checkbox5

% --- Executes on button press in radiobutton1.
function radiobutton1_Callback(hObject, eventdata, handles)
% hObject    handle to radiobutton1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hint: get(hObject,'Value') returns toggle state of radiobutton1

% --- Executes on button press in radiobutton2.
function radiobutton2_Callback(hObject, eventdata, handles)
% hObject    handle to radiobutton2 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hint: get(hObject,'Value') returns toggle state of radiobutton2

function edit3_Callback(hObject, eventdata, handles)
% hObject    handle to edit3 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of edit3 as text
%        str2double(get(hObject,'String')) returns contents of edit3 as a double

% --- Executes during object creation, after setting all properties.
function edit3_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edit3 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

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

% --- Executes during object creation, after setting all properties.
function axes1_CreateFcn(hObject, eventdata, handles)
% hObject    handle to axes1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called
%axis off;
% Hint: place code in OpeningFcn to populate axes1
% axes(hObject)
% imshow('start.png')
% imshow(imageArray);
function axes2_CreateFcn(hObject, eventdata, handles)
    % hObject    handle to axes2 (see GCBO)
    % eventdata  reserved - to be defined in a future version of MATLAB
    % handles    empty - handles not created until after all CreateFcns called

    % Hint: place code in OpeningFcn to populate axes2

    % Read in image
    imageArray = imread('greenlight.jpg');
    % Switch active axes to the conimote2 you made for the image.
    axes(handles.axes2);
    % Put the image array into the axes so it will appear on the GUI
    imshow(imageArray);
    axis off;

function uitable1_CreateFcn(hObject, eventdata, handles)
    % hObject    handle to uitable1 (see GCBO)
    % eventdata  reserved - to be defined in a future version of MATLAB
    % handles    empty - handles not created until after all CreateFcns called
    dat = zeros(99,1);

function uitable1_CellSelectionCallback(hObject, eventdata, handles)
    % hObject    handle to uitable1 (see GCBO)
    % eventdata  structure with the following fields (see UITABLE)
    %   Indices: row and column indices of the cell(s) currently selected
    %   handles    structure with handles and user data (see GUIDATA)

function uitable1_CellEditCallback(hObject, eventdata, handles)
    % hObject    handle to uitable1 (see GCBO)
    % eventdata  structure with the following fields (see UITABLE)
    %   Indices: row and column indices of the cell(s) edited
    %   PreviousData: previous data for the cell(s) edited
    %   EditData: string(s) entered by the user
    %   NewData: EditData or its converted form set on the Data property. Empty if Data was not changed
    %   Error: error string when failed to convert EditData to appropriate value for Data
    %   handles    structure with handles and user data (see GUIDATA)

function axes4_CreateFcn(hObject, eventdata, handles)
    % hObject    handle to axes4 (see GCBO)
    % eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: place code in OpeningFcn to populate axes4

% --- Executes during object creation, after setting all properties.
function debugimote2_CreateFcn(hObject, eventdata, handles)
% hObject    handle to debugimote2 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

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

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

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

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

% --- Executes on selection change in nfft.
function nfft_Callback(hObject, eventdata, handles)
% hObject    handle to nfft (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: contents = cellstr(get(hObject,'String')) returns nfft contents as cell array
%        contents{get(hObject,'Value')} returns selected item from nfft

% --- Executes during object creation, after setting all properties.
function nfft_CreateFcn(hObject, eventdata, handles)
% hObject    handle to nfft (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called
% Hint: popupmenu controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'), get
(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end
set(hObject,'string','64|128|256|512|1024|2048|4096|8192')  % set options for drop-down menu

% --- Executes on button press in cleanup.
function cleanup_Callback(hObject, eventdata, handles)
    % hObject    handle to cleanup (see GCBO)
    % eventdata  reserved - to be defined in a future version of MATLAB
    % handles    structure with handles and user data (see GUIDATA)
cleanUpfiles = dir('outfile*');
while length(cleanUpfiles) > 10
    cleanOld = cleanUpfiles(1);
    delete(cleanOld.name);
    cleanUpfiles = dir('outfile*');
end
set(handles.debugimote2,'string','Old output files deleted!');

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

    % %Reset Gateway Node
    % COM_ports = instrfind;
    % size_ports = max(size(COM_ports));
    % resetGateway = ['LocalCommand Reset'];
    %
    % comy = serial(debugport);
    % baud_Rate = comy.Baudrate;
    %
    % while baud_Rate ~= 9600
    %     fprintf(comy, resetGateway); pause(3);
    % end
    % fclose(comy); delete(comy);  clear comy;
%
    % Close and delete open serial ports
    % fclose(COM_ports);
    % delete(COM_ports);

    set(handles.stop, 'userdata',1);

set(handles.debugimote2,'string','Program Stopped!');
% kofi = 1
% Close dos prompt
% dos('TASKKILL /IM cmd.exe');

% --- Executes during object creation, after setting all properties.
function heading_CreateFcn(hObject, eventdata, handles)
    % hObject handle to heading (see GCBO)
    % eventdata reserved - to be defined in a future version of MATLAB
    % handles empty - handles not created until after all CreateFcns called
    axes(hObject)
    imshow('lablogo.png')
    % Hint: place code in OpeningFcn to populate heading

% --- Executes during object creation, after setting all properties.
function uitable2_CreateFcn(hObject, eventdata, handles)
    % hObject handle to uitable2 (see GCBO)
    % eventdata reserved - to be defined in a future version of MATLAB
    % handles empty - handles not created until after all CreateFcns called
dat2 = zeros(4,2);

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

% --- Executes on button press in modelcheck.
function modelcheck_Callback(hObject, eventdata, handles)
    % hObject handle to modelcheck (see GCBO)
    % eventdata reserved - to be defined in a future version of MATLAB
    % handles structure with handles and user data (see GUIDATA)
    % Hint: get(hObject,'Value') returns toggle state of modelcheck

% --- Executes on button press in mode2check.
function mode2check_Callback(hObject, eventdata, handles)
    % hObject handle to mode2check (see GCBO)
    % eventdata reserved - to be defined in a future version of MATLAB
    % handles structure with handles and user data (see GUIDATA)
    % Hint: get(hObject,'Value') returns toggle state of mode2check

% --- Executes on button press in mode3check.
function mode3check_Callback(hObject, eventdata, handles)
    % hObject handle to mode3check (see GCBO)
    % eventdata reserved - to be defined in a future version of MATLAB
    % handles structure with handles and user data (see GUIDATA)
% Hint: get(hObject,'Value') returns toggle state of mode3check

% --- Executes on button press in mode4check.
function mode4check_Callback(hObject, eventdata, handles)

% hObject    handle to mode4check (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hint: get(hObject,'Value') returns toggle state of mode4check

% --- Executes during object creation, after setting all properties.
function text6_CreateFcn(hObject, eventdata, handles)

% hObject    handle to text6 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% --- Executes during object creation, after setting all properties.
function axes13_CreateFcn(hObject, eventdata, handles)

% hObject    handle to axes13 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called
% axes(handles.axes13)
% imshow('funknown.png')
% Hint: place code in OpeningFcn to populate axes13

% --- Executes during object creation, after setting all properties.
function axes14_CreateFcn(hObject, eventdata, handles)

% hObject    handle to axes13 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called
% axes(hObject)
% imshow('funknown.png')
% Hint: place code in OpeningFcn to populate axes13

% --- Executes during object creation, after setting all properties.
function axes15_CreateFcn(hObject, eventdata, handles)

% hObject    handle to axes13 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called
% axes(hObject)
% imshow('funknown.png')
% Hint: place code in OpeningFcn to populate axes13

% --- Executes during object creation, after setting all properties.
function axes17_CreateFcn(hObject, eventdata, handles)

% hObject    handle to axes17 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles empty - handles not created until after all CreateFcns called
% axes(hObject)
% imshow('funknown.png')
% Hint: place code in OpeningFcn to populate axes

% t = timer('TimerFcn', @(x,y)disp('Starting'), 'StartDelay', 5);
% start(t)
% wait(t)
% delete(t)