DCT 225

Testing and Operation of Delaware's First Permanently Instrumented "Smart" Bridge

By

HARRY W. SHENTON III
DOUGLAS NATHANIEL CHARLES

Center for Innovative Bridge Engineering
University of Delaware

January 2012

Delaware Center for Transportation
University of Delaware
355 DuPont Hall
Newark, Delaware 19716
(302) 831-1446

The Delaware Center for Transportation is a university-wide multi-disciplinary research unit reporting to the Chair of the Department of Civil and Environmental Engineering, and is co-sponsored by the University of Delaware and the Delaware Department of Transportation.

DCT Staff

Ardeshir Faghri
Director

Jerome Lewis
Associate Director

Ellen Pletz
Assistant to the Director

Earl "Rusty" Lee
T^2 Program Coordinator

Matheu Carter
T^2 Engineer

Sandra Wolfe
Event Coordinator

DCT Policy Council

Natalie Barnhart, Co-Chair
Chief Engineer, Delaware Department of Transportation

Babatunde Ogunnaike, Co-Chair
Dean, College of Engineering

Delaware General Assembly Member
Chair, Senate Highways & Transportation Committee

Delaware General Assembly Member
Chair, House of Representatives Transportation/Land Use & Infrastructure Committee

Ajay Prasad
Professor, Department of Mechanical Engineering

Harry Shenton
Chair, Civil and Environmental Engineering

Michael Strange
Director of Planning, Delaware Department of Transportation

Ralph Reeb
Planning Division, Delaware Department of Transportation

Stephen Kingsberry
Executive Director, Delaware Transit Corporation

Shannon Marchman
Representative of the Director of the Delaware Development Office

James Johnson
Executive Director, Delaware River & Bay Authority

Holly Rybinski
Project Manager-Transportation, AECOM

Delaware Center for Transportation
University of Delaware
Newark, DE 19716
(302) 831-1446

TESTING AND OPERATION OF DELAWARE'S FIRST PERMANENTLY INSTRUMENTED "SMART" BRIDGE

By

Harry W. Shenton III

Douglas Nathaniel Charles

January 25, 2012

EXECUTIVE SUMMARY

In 2005 a research project was initiated at the University of Delaware Center for Innovative Bridge Engineering (CIBrE) to permanently instrument a typical highway bridge, so that quantitative data could be collected about the long-term performance of the bridge. The bridge selected to be Delaware's first "Smart Bridge" was bridge 1-821, which carries I-495 north over Edgemoor road in Wilmington, Delaware. By the completion of that project, along with a number of other tasks, the monitoring system had been designed and significant progress was made on installation of the system. The monitoring system was operational but additional work needed to be completed before it was fully operational. The results of that work are reported in Delaware Center for Transportation Report DCT 194. Presented in this report are the results of a follow-up study to the original project that completed the installation of the monitoring system and made it operational.

Bridge 1-821 is 243 ft long, a slab-on-steel girder design, and consists of a 3 span continuous section and a simply supported span. The bridge is just under 70 ft wide and has three travel lanes, an exit lane, and a shoulder. It experiences a high level of car and truck traffic. The monitoring system installed on the bridge consisted of 61 sensors to measure strains, accelerations, temperature, and deflection at various locations on the bridge. A datalogger was placed on-site to collect "monitor" and "event" data 24/7. The logger is accessed remotely using the internet, through either a DSL modem or a cellular modem, both of which are at the site.

Shortly after the monitoring system was installed in 2007, a problem was discovered with the bridge coating system and the owner (DelDOT) was forced to

have the bridge repainted. As a result, all of the sensors, cables, and junction boxes had to be removed, and then reinstalled after the painting was complete. A limited window of time was provided to reinstall the equipment from the painters scaffold before it was taken down; however, the time allotted was insufficient for the significant amount of work needed to be done. The reinstall could not be completed until months later, with the assistance of a DelDOT under bridge inspection vehicle. Removal and reinstallation of the system resulted in a number of problems that needed to be addressed to bring the system back to operational status.

Chapter 2 provides background on the project and the status of the monitoring system at the start of this project. Bridge 1-821 was selected in 2003 to be the first smart bridge based on a number of criteria, such as commonality of the design, ADT, ease of access, and proximity to the University of Delaware. A detailed description of the bridge is presented. The work conducted by Reader (2007) and Natale (2008) on the project is summarized.

At the end of the initial phase of the reinstall in 2008, 14 of the 30 resistive foil strain gages were not operational. The cause of the problems was likely the hasty effort to reinstall the equipment before the painters scaffold was removed. Chapter 3 describes how the problems were diagnosed and repairs were made. The problems were found to be with faulting wiring at the bridge completion module, or shorts or incorrect connections at the junction boxes. A bucket truck or an Under Bridge Inspection Vehicle (UBIV) was needed to gain access to the sensors to make the repairs. DelDOT provided the vehicles and the maintenance of traffic when the UBIV was used, to make the repairs. The work, however, had to be scheduled during times when the vehicles and crews were not needed for required inspections or

maintenance. As a result, the sensor repairs were completed over a period of eight days (four separate, two day periods), between August of 2009 and June of 2010. The repairs consisted of resoldering the connections properly at the bridge completion module or correcting the connections at the junction boxes. The chapter ends with a discussion of trouble shooting the wiring at the main enclosure.

Two different types of data are collected by the monitoring system – "monitor" and "event" data. Monitor data is a ten second average taken every hour, on the hour, of all of the sensors. This data can be used to study the long-term performance of the bridge due to environmental effects and sustain load. Event data records are automatically triggered when a heavy vehicle crosses the bridge, based on a user specified trigger. This information can be used to study the response of the bridge to the site-specific traffic. Chapter 4 describes the work carried out to develop and enhance the datalogger program for capturing the monitor and event data. The program logic to capture the event data that was developed in the earlier project was revised to use a trigger value that is relative to the most recent monitor reading for the trigger gages. This was necessary to account for the variation in sensor output due to thermal effects. The program logic to read the monitor data that was developed in the earlier project was also corrected so that only one average is taken and recorded once an hour (the earlier program incorrectly recorded the reading four times and hour, which was unnecessary and used four times the storage).

With a tremendous amount of monitor and event data being collected, there needs to be a way to process the data automatically and quickly for easy interpretation. Chapter 4 describes the Matlab programs that were developed for post-processing the monitor and event data. For the event date, this includes (1) a program

to parse individual events into separate data files and store then in a logical manner on disk, (2) a program that scans each individual event file and calculates the maximum, minimum, average, and standard deviation for every sensor and writes the results to a summary file, and (3) a program for plotting individual event records. For the monitor data, this includes a program to gather data that has been downloaded into a number of different files, combine it into one single master file, and then create time history plots of the monitor data from the master file.

Chapter 5 describes operation of the monitoring system, which includes downloading the event and monitor data tables and post-processing the data. Instructions are given for how to download the event data table in PC9000, and then convert it to individual files using the "Convert Data Files" command. The monitor data table can be downloaded in a similar fashion, or, alternatively using another Campbell Scientific program called RTDAQ. Step-by-step instructions are given for sorting the event data files, calculating the statistics, and graphing individual event files. Instructions are also given for combining monitor data into a single month record and plotting the results.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

Chapter 1

INTRODUCTION

1.1 Motivation

As many of the nation's bridges reach the end of their design service life, the reliability and safety of these bridges have become an increasing concern in the United States. Aging infrastructure facilities are currently deteriorating at a rapid rate, faster than they are being repaired, rehabilitated, or rebuilt (Maalej, et al, 2002). In the wake of the collapse of the Interstate 35W Bridge in Minneapolis, Minnesota in 2007, public awareness peaked with regards to the safety of US infrastructure. This attention has led to an abundance of research in the area of structural health monitoring.

Health monitoring is a method which uses strategically placed sensors to quantify bridge performance and recognize possible warning signs of problems before they become hazards to the public. Health monitoring can be used to assess the deterioration of a bridge due to service conditions or following unforeseen loading conditions (Land, et al, 2003). Most inspection techniques deal primarily with qualitative things such as visually identifying cracks and rusting of girders and connections. In order to better understand the way bridges deteriorate and become less serviceable, more quantitative data must be gathered. This was the main motivation behind the Federal Highway Administration (FHWA) developing the Long-Term Bridge Performance Program (LTBPP), which was enacted by the U.S. congress in 2005. The goal of this 20 year research effort is to compile a

comprehensive database of quantitative information from a representative sample of bridges nationwide, considering every element of a bridge. By analyzing all physical and functional variables that affect bridge performance, this program hopes to provide detailed and timely pictures of bridge health, as well as aiding to develop better bridge management tools (Federal Highway Administration (FHWA)).

1.2 History

The Federal Highway Act of 1956 led to an explosion of bridge and highway construction in the United States that continued for almost two decades. Fueled by the federal promise to provide 90% of construction costs for all highway projects, states were able to build and improve infrastructure nationwide for a fraction of the actual cost. This program had an incredible impact on the construction industry, as these were the busiest years in U.S. history in terms of bridge construction. Figure 1.1 below is evidence of this influence. The bridges built during this time period were completed with the main objectives being low cost and fast construction. Unfortunately not much thought was put into the future safety and serviceability of these structures. Most of the bridges built during this time period are now between 40 and 50 years old, and have experienced an ever increasing level of traffic and truck loads, that could have never been foreseen at the time of construction.

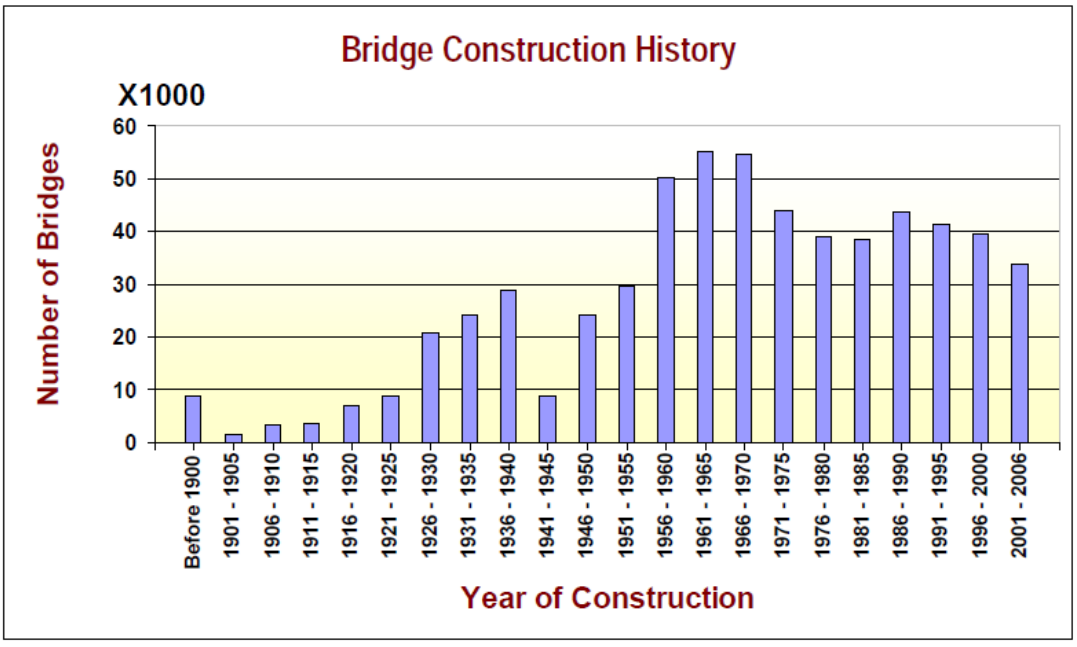


Figure 1.1 Yearly bridge construction history histogram (Federal Highway Administration (FHWA))

The Silver Bridge, which crossed the Ohio River between Ohio and West Virginia, collapsed in December of 1967 due to fatigue and killed 46 people, injuring 9 more. This disaster was the first major structural failure since the wind induced failure of the Tacoma Narrows Bridge in 1940, and prompted the United States government to intervene and re-evaluate bridge safety regulations. Several years later in 1970, the National Bridge Inspection Program was created, requiring each state to evaluate their bridges every two years and submit results to the FHWA. These results were then combined to create the National Bridge Inventory (NBI) database. The following year the National Bridge Inspection Standards (NBIS) were established, setting a uniform, national standard for bridge inspection and safety evaluation (Phares, et al, 2005).

Evaluation techniques to date have been primarily based on subjective visual inspections of a structure. In recent years, however, more accurate methods of inspection have been developed. Localized testing techniques such as eddy currents, ultrasonics, acoustic based testing, and strain and corrosion monitoring, together with visual inspections can produce a much more detailed picture of structural integrity. The main problem with these localized methods is that they require an idea of where a problem may occur before an assessment can be completed. Considering the large number of national bridges, coupled with the increasing complexity of these structures, it is not economically feasible to identify and test all potential damage locations (Phares, et al, 2005).

One of the most important things for the future of health monitoring of bridge structures is also a simple one: an accumulation of data. Health monitoring has been used for years to monitor large signature bridges, but has rarely been implemented for common highway bridges. The more data that can be accumulated for a typical slab on girder steel bridge, such as bridge1-821, the wider the potential applications of the results may be.

1.3 Significance

Highway bridges are vital components of the nation's transportation network and provide critical links that make nationwide commerce and mobility possible. This transportation system has become so integrated into the daily lives of Americans that it is sometimes taken for granted, like clean tap water or electricity (Federal Highway Administration (FHWA)). To combat this complacent public attitude, it is important that the engineering community develop new ways to highlight how serious the infrastructure problem in our country has become.

As of December 31, 2010 the NBI included a total of approximately 605,000 bridges; 11.5% of these bridges were classified as structurally deficient, while another 12.8% were listed as functionally obsolete (Federal Highway Administration (FHWA)). Neither distinction implicitly means a lack of safety, but these numbers do imply an overwhelming number of defects in U.S infrastructure. With limited funds dedicated each year to bridge rehabilitation, identifying bridges with considerable structural flaws is critical to the safety of the American public.

One way of accomplishing this task is to gather more quantitative data about our bridges so that the limited amount of resources can be spent where they are needed most. Health monitoring can help bridge owners to determine how much useful life is left in a structure, plan and design effective retrofit strategies, program long term maintenance, reduce life cycle costs through preventative maintenance, and evaluate bridge integrity after a catastrophic event (Land, et al, 2003). It is believed that data accumulated from health monitoring systems will not only help owners extend the life of bridges currently in use, but also help designers of future projects be more efficient by avoiding problems observed in current structures (Natale, 2008). With the accumulation of data and careful observation of long-term trends of monitored structures it may be possible to avoid, or at the very least delay, the next major bridge catastrophe.

1.4 Thesis Outline

The remainder of this thesis will be organized in the following manner:

Chapter 2 – Background describes the bridge selected for monitoring, as well as the acquisition system used for this project. The work done on the project by previous researchers is also presented in detail.

Chapter 3 – System Repairs discusses the bridge completion modules used for foil strain gages on the bridge and a summary of repairs performed at the bridge site. A troubleshooting guide to identifying and fixing wiring issues is also included.

Chapter 4 – Data Processing and Code Development provides information on previous data acquisition codes and how current code has improved upon them. This chapter also goes into detail about the format of event and monitoring data and how their respective programs deal with the differences in formats.

Chapter 5 – Operation of the Monitoring System includes a comprehensive step by step description of the download and data analysis process.

Chapter 6 – Current System Status and Future Development discusses the current status of various aspects of the monitoring project and provides several ways to improve the project moving forward.

Appendix includes all the MATLAB and PC9000 programs discussed in the thesis, as well as additional graphs produced using these programs. It also includes the description of a semi-controlled load test conducted on the bridge

Bibliography lists the references used throughout the project.

Chapter 2

BACKGROUND

Work on this instrumentation project began in 2005 with Reader (Reader, 2007) investigating different acquisition systems and gage types. Her work concluded in the spring of 2007 and was picked up by Natale (Natale, 2008). He continued working on the project until he graduated in the spring of 2008 and was then continued by the author in the fall of 2008. To best understand the most recent additions to this project, it is helpful to first cover the steps taken by the previous researchers.

2.1 Bridge Selection and Description

In choosing which bridge would become Delaware's first permanently instrumented bridge, several basic criteria were required to be met. The first was that the bridge had to be a common type of bridge, so that any results could have a wide range of application. Some other criteria included being close in proximity to the University of Delaware, high average daily traffic, high average daily truck traffic, and easy accessibility to the bridge. During the summer of 2003, Lynch (2003) conducted a study of approximately 180 bridges in New Castle County to determine the best option for the project; the result of this study was Bridge 1-821, located in Wilmington, Delaware, was selected as the best candidate for Delaware's first permanently instrumented bridge (Lynch, 2003).

Bridge 1-821 is located on I-495 about 15 miles north of the University of Delaware campus in Newark, Delaware. The bridge carries northbound traffic over

Edgemoor Road in Wilmington, Delaware. The bridge is 69’-6” wide and includes three travel lanes, one exit lane and one emergency lane. The original construction of the bridge included 7 girders to carry just two lanes of traffic and an exit lane, but it was later expanded by two girders (G8 and G9 in Figure 2.1 below) to add one additional lane and an emergency lane. The current configuration includes 9 girders spaced at 8’-0” on center, supporting a composite concrete deck slab. The entire bridge spans a total of 243 feet and has four spans, three continuous and one simply supported. The continuous spans, two, three and four, are the spans that are instrumented and have a total length of 200 feet. Spans two and four are 61’-3 ¾ ” in length, while span three is the longest at 76’-7 5/8 ”. The continuous portion is supported by three sets of concrete piers and the north abutment at the end of span four.

Four different girder cross sections are present along the length of the instrumented spans. Because the bridge is continuous, the cross sections change over each pier to add additional strength in the negative bending regions along the girder. The original seven girders have three varying cross sections. The first of these cross sections is a W36x135 with an 8” composite slab. The second cross section is a W36x150 with an 8” composite slab. The third cross section detail is a W36x150 with an 8” composite slab; the same as two but with an additional 0.668”x10” cover plate on the top and bottom flanges. The two new girders, G8 and G9, have only two different cross sections along their length. The first is the same as cross section 2 from the original girders, W36x150 with an 8” composite slab. The second cross section is used in negative bending regions where the girders cross piers and is a W36x210 with an 8” composite slab. All connecting diaphragms are MC 18x42.7 cross sections.

Drawings of these cross section details can be seen in Figures 2.2-2.6. Table 2.1 breaks down the location of each of these cross sections and their lengths, measured from pier one at the south end of span two in Figure 2.1.

Table 2.1 Summary of cross sections along the length of the bridge (Reader 2007)

Girders 1 - 7 X-Section	Distance (ft)	Cross Section #
Span 2	0.0' - 44.3'	1
	44.3' - 52.3'	2
	52.3' - 61.3'	3
Span 3	61.3' - 70.3'	3
	70.3' - 78.3'	2
	78.3' - 120.94'	1
	120.94' - 128.94'	2
	128.94' - 137.94'	3
Span 4	137.94' - 146.94'	3
	146.94' - 154.94'	2
	154.94' - 199.24'	1

Girders 8 – 9 X-Section	Distance (ft)	Cross Section #
Span 2	0.0' - 44.3'	2
	44.3' - 61.3'	4
Span 3	61.3' - 78.3'	4
	78.3' - 120.94'	2
	120.94' - 137.94'	4
Span 4	137.94' - 154.94'	4
	154.94' - 199.24'	2

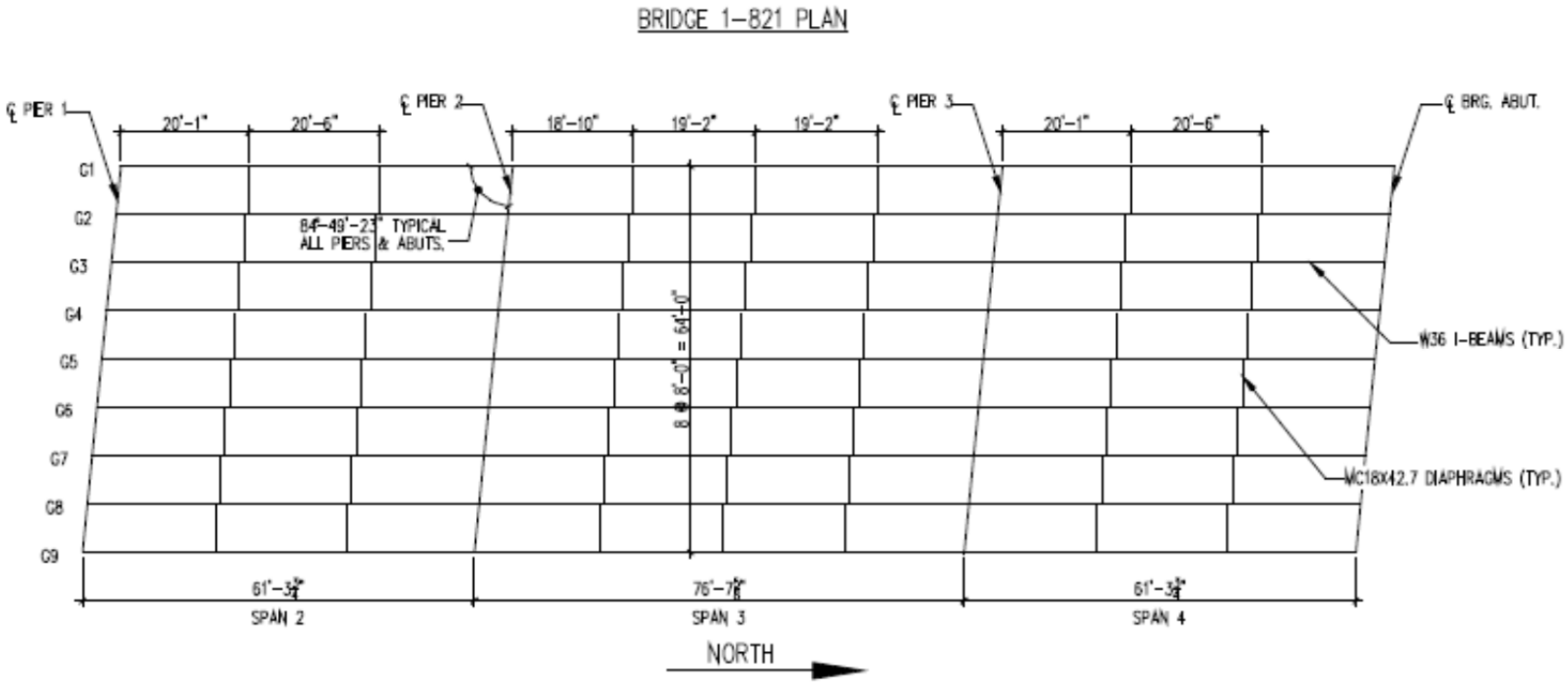


Figure 2.1 AutoCAD drawing of Bridge 1-821 (Reader 2007)

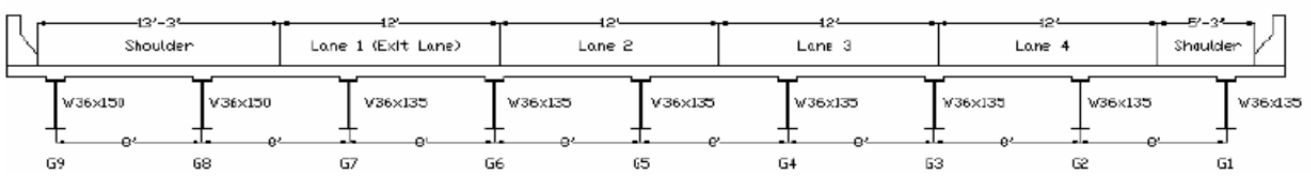


Figure 2.2 Cross section of Bridge 1-821 (Rakowski 2008)

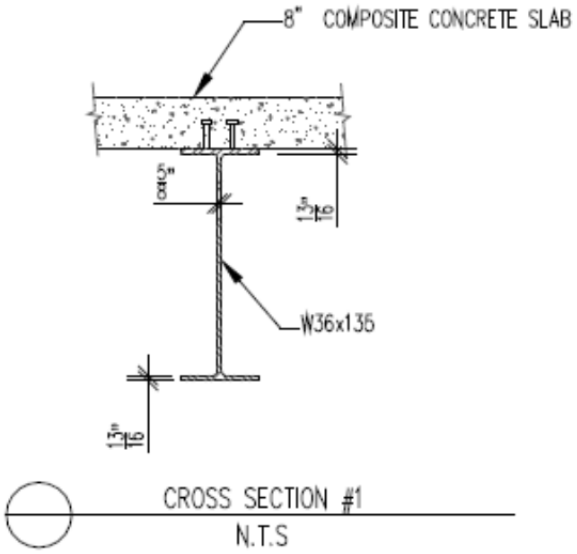


Figure 2.3 Cross section #1 detail (Reader 2007)

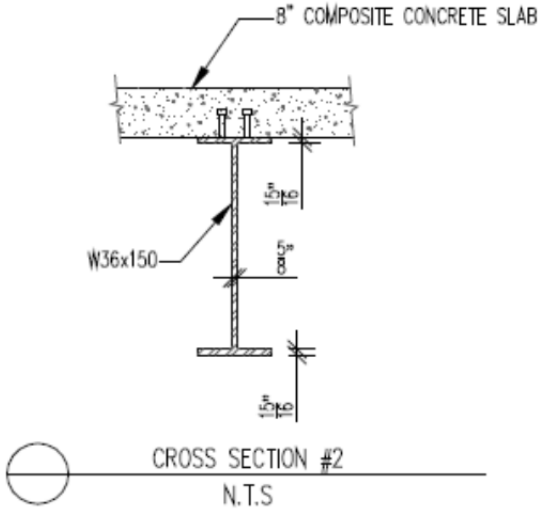


Figure 2.4 Cross section #2 detail (Reader 2007)

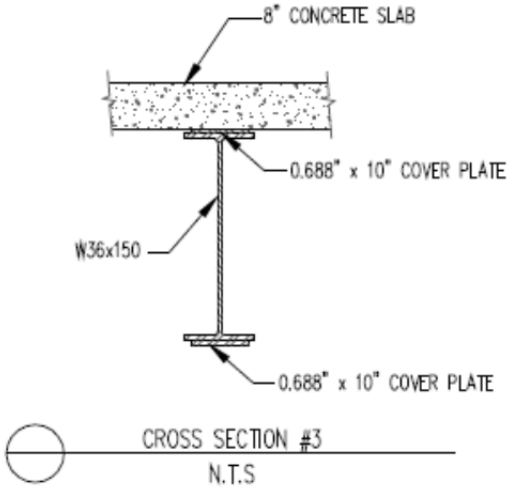


Figure 2.5 Cross section #3 detail (Reader 2007)

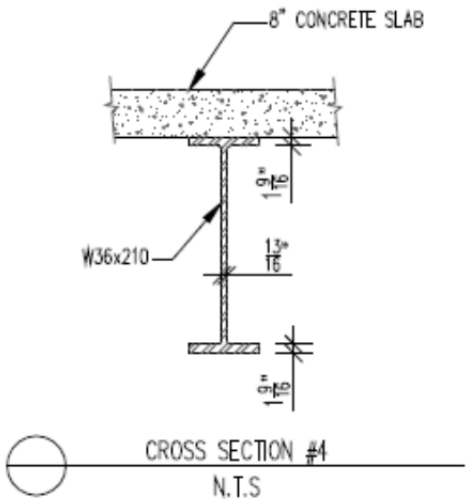


Figure 2.6 Cross section #4 detail (Reader 2007)

2.2 Prior Project Status

2.2.1 Reader (2007)

Reader conducted the original planning and instrumentation of the monitoring system in 2006 and 2007. This included reviewing and selecting the gages to be used and installing the original monitoring system on the structure. The final design included six different types of sensors, with a total of 58 gages. These gages include 31 foil strain gages, six accelerometers, five thermocouples, four vibrating wire strain gages, nine string pots, and three resistance temperature detectors (RTD). Reader researched and selected all necessary hardware to build and operate the monitoring system. Specifications for each type of gage and corresponding hardware can be found in Reader, 2007.

The data acquisition system selected for bridge 1-821 is a Campbell Scientific CR9000X Measurement control system. This data acquisition system was

selected because it is capable of supporting a large number of channels and is intended for applications requiring rapid scan rate. The CR9000X has an internal memory of 128 MB as well as an external PC card memory option for additional memory storage. The system has nine available I/O modules that can be configured with input or excitation boards based on the necessary application (Reader, 2007). The modules used for this particular project are the CR9050 5 Volt Analog Input Module and the CR9060 Excitation Module. Data acquisition programs for the CR9000X are completed in the BASIC programming language and can be generated using the PC9000 support software. Communication with the CR9000X can be done on site through a direct serial port connection or remotely over the internet using an ethernet connection (Reader, 2007).

Reader was responsible for developing a grid system and nomenclature to identify gages and their locations on the structure. The nomenclature system uses up to four characters to give the necessary information for identifying a gage. The first character denotes the first letter of the type of gage you are referencing. For example "S" represents foil strain gage and "T" represents thermocouples. The next two characters are two numbers that indicate the grid location of the gage on the bridge. The first number identifies which span the gage is on, while the second number describes the girder location. The girders are numbered one through nine starting with the west side of the bridge and the sections are numbered one through seven starting from the south end of the bridge. The grid system uses the spans running north-south as an x-axis and the girders running east-west as a y-axis (Reader, 2007). The fourth character is a letter that gives further information about the gage location. For strain gages the fourth letter will be a "T" or "B" and designates whether the gage is on the

top or bottom flange of the girder. For accelerometers the fourth letter will be either a "X" or "Y" and indicates the wiring and output for accelerations being measured in the lateral and vertical direction. An example of the nomenclature is the trigger gage, S35B; the letter S describes this as a foil strain gage, the numbers identify the gage location as being on span 3 on girder number 5, and the final letter B details the gage is on the bottom flange.

Five gages are labeled independently from the grid system nomenclature and these are labeled SCOU, TCOU, RCOU, SDUM, and TD35. SCOU is a strain gage and RCOU and TCOU are temperature gages, all of which are located on a steel coupon attached to one of the girders. SDUM is a dummy strain gage which is used to assess signal noise in the system. TD35 is a temperature gage located on the deck of the bridge (Reader, 2007).

A finite element model (FEM) of the bridge was also developed to compare to initial data collected from the bridge. The FEM was calibrated based on static diagnostic and dynamic tests performed by undergraduate researchers, Burrell and Wherum (Burrell, 2004, Wehrum, 2006). Data from field tests and the FEM were used to compare transverse load distribution between girders. The natural frequencies of the bridge were also calculated and compared to the values developed from the FEM model. In both cases the values predicted from the FEM were fairly close to the values observed from the field data. Reader also developed the initial data collection program using the software program PC9000 and provided a starting point for Natale. Additional information on both the CR9000X data acquisition system and the gages used for the monitoring system can be found in Reader (Reader, 2007).

2.2.2 Natale (2008)

One of the factors considered in the initial selection of Bridge 1-821 for this monitoring project was the condition of the paint on the superstructure. The Delaware Department of Transportation (DelDOT) originally believed that Bridge 1-821 would only need minor touch up paint jobs after the system was first installed in 2007. In the summer of 2007, however, DelDOT decided to spot paint and power wash the bridge to complete a bearing replacement from the winter of 2006. Once the power washing began, large sections of paint were being blasted off the girders. Upon inspection, several layers of old paint were found on the superstructure of the bridge. DelDOT then decided the best course of action was to sandblast all the paint off the entire bridge, add one layer of primer, and add two new layers of paint to the entire structure. The result of this decision was that all sensors, wires, wire trays and both junction boxes had to be removed from the bridge.

Marinis Brothers Incorporated were contracted to begin the sandblasting and painting operations in September of 2007. Marinis offered the use of their scaffolding system to aid in the removal of the monitoring instruments from the structure. This significantly reduced the amount of time for the removal process, as a boom lift would have been required otherwise. September 11-12, 2007 the accelerometers, string pots, and all bridge completion modules were removed and salvaged. The vibrating wire strain gages, foil strain gages, and the resistance temperature detectors could not be salvaged because they were all welded to the girders. The major drawback of removing the system so quickly was that not all the wire configurations and locations were recorded, which made re-installation a much more time consuming process.

While the system was removed from the bridge, the design of the entire monitoring system was re-evaluated. All of the salvaged equipment was tested to make sure there was no damage from the removal process. At this time it was decided that the string pot configurations were not functioning effectively. The goal of the string pots is to determine vertical deflection and beam rotation during events on the bridge. The setup included a cable stretched from beam to beam, providing a datum line to compare to deflections and rotations. It was found that the tensile strength in the cable wasn't sufficient to remain stationary, however, and no deflections were recorded during daily events on the bridge. Several changes were attempted, but with no success. The cable was first tightened, resulting in the cable snapping under the increased tensile strain. A larger diameter cable, capable of higher tension without snapping, was then tried but it still did not perform the way it should. As of now the string pot configuration is still not operational.

Once the sandblasting and painting was completed, Marinis again offered the scaffolding to aid in the re-installation of the monitoring system. The re-installation began on Wednesday, November 7, 2007 but was not completed by Friday, November 16, 2007 when Marinis was forced to remove the scaffolding due to time constraints. The rest of the installation waited until April 2008, when a bucket truck was provided by DelDOT. This truck didn't have the necessary reach to help install gages onto all the girders, however, and an under bridge inspection vehicle (UBIV) was reserved in late April to install the remaining fifteen strain gages, 2 vibrating wire strain gages, 5 accelerometers, and the four temperature sensors. As mentioned before, the wiring from the foil strain gages to the bridge completion modules was not accurately recorded for all gages during the removal process in

September 2007. This resulted in not all gages working properly and left with many rewiring issues to manage during the fall 2008 and spring of 2009.

Chapter 3

SYSTEM REPAIRS

As of the fall of 2008, only 16 of the total 30 foil strain gages were properly wired and operating as they should. Figure 3.1 shows the gages that were identified as not working properly in red circles. The green boxes identified in the figure show the gages that were used as wiring templates to correct the issues with the other gages.

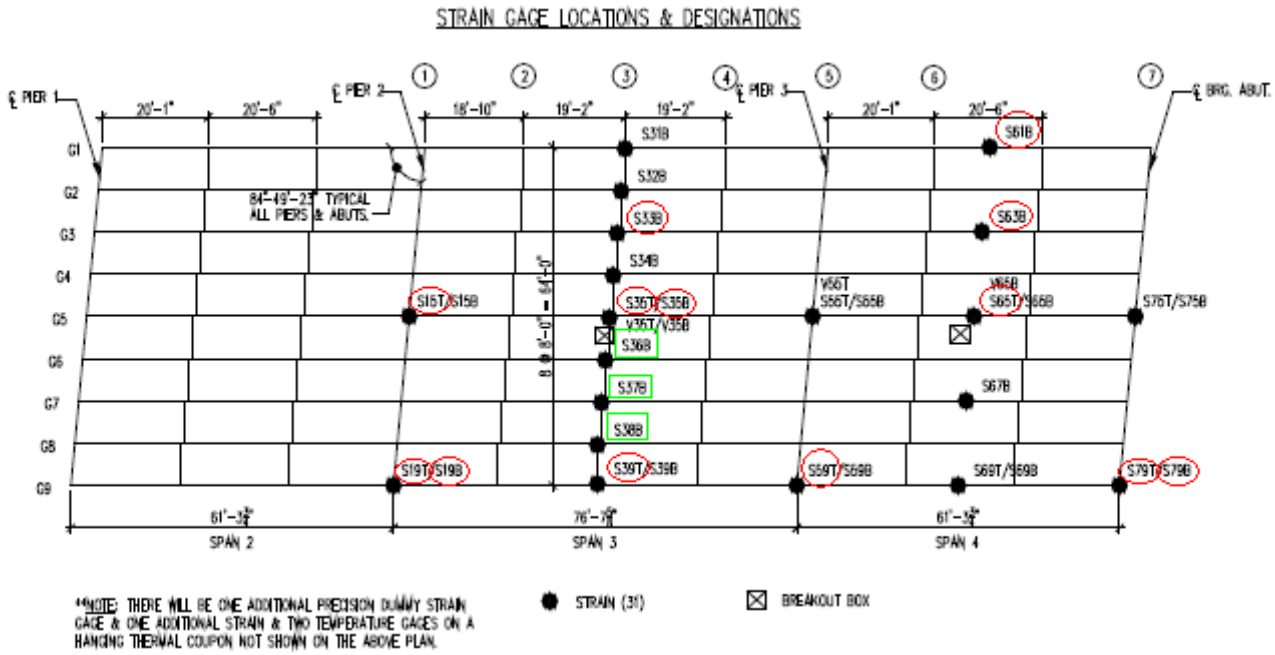


Figure 3.1 Problem gage locations

The gages that weren't working properly were identified a few different ways. The most direct way was to look at the data from the CR9000 data logger and see which gages were recording "NAN" (Not-A-Number) instead of strain values. Others were identified because they were recording atypical data during events or did not register any peaks at all. Finally the resistance, in Ohms, was checked for all the gages and some of the values recorded indicated wiring issues were present in several more gages. Most of the other 14 gages had problems with their wiring to the bridge completion modules. Some were wired to the wrong tabs on the module, so the bridge circuit was not complete and therefore the gage did not behave as it should. Some gages had running solders between adjacent tabs which needed to be cleaned off and re-soldered, while others simply had loose wires no longer attached to the module at all. As an example of the condition of the bridge completion modules in the field, the module for gage S37B is shown in Figure 3.2 below.

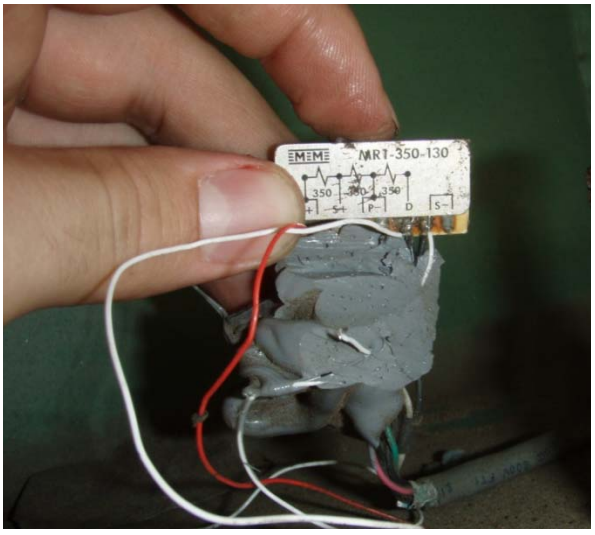


Figure 3.2 Example of bridge completion module in the field

3.1 Bridge Completion Module Background

The foil strain gages used for the monitoring system are quarter bridge strain gages and thus required bridge completion modules to complete the circuit. The MR1-350-130 Vishay Micromeasurements bridge completion module accomplishes this task by completing the full bridge circuit. The module has a resistance of 350 ohms between each node and also provides a 350 ohm dummy gage for individual shunt calibrations (Reader 2007). One of these modules is present at every site where a foil gage has been spot welded to a girder. The modules are attached to the strain gages by soldering the leads from the gage to specific tabs on the module. The foil strain gages have three output wires that should be soldered to the P+, D, and S- tabs on the completion module. There are six other output wires that connect the completion module to the junction box, which then is connected further down the circuit to the data acquisition system in the main enclosure. There are nine wires connected to eight tabs of the bridge completion module, so two wires need to be connected to the same tab. This can occur at the D tab or S-, as they actually represent the same place in the full bridge circuit. A visual interpretation of a full bridge circuit can be seen in Figure 3.3.

The CAD drawing below shows each of the nodes of the full bridge circuit and the significance of each wire connected to the nodes. The six output wires that complete the full bridge circuit include two shunt wires, low and high sense wires, a grounding wire, and an excitation wire. You can see from the figure below why the S- and D can be considered to be on the same tab, because they are essentially sharing one spot in the circuit. As mentioned above, the strain gage should be soldered to the P+, D, and S- tabs on the completion module, which can also be seen by the red, white, and white wire designations adjacent to the identifying circle in the figure

below. The strain gages are called quarter bridge strain gages for the simple reason that they represent one quarter of the full bridge circuit shown below in Figure 3.3.

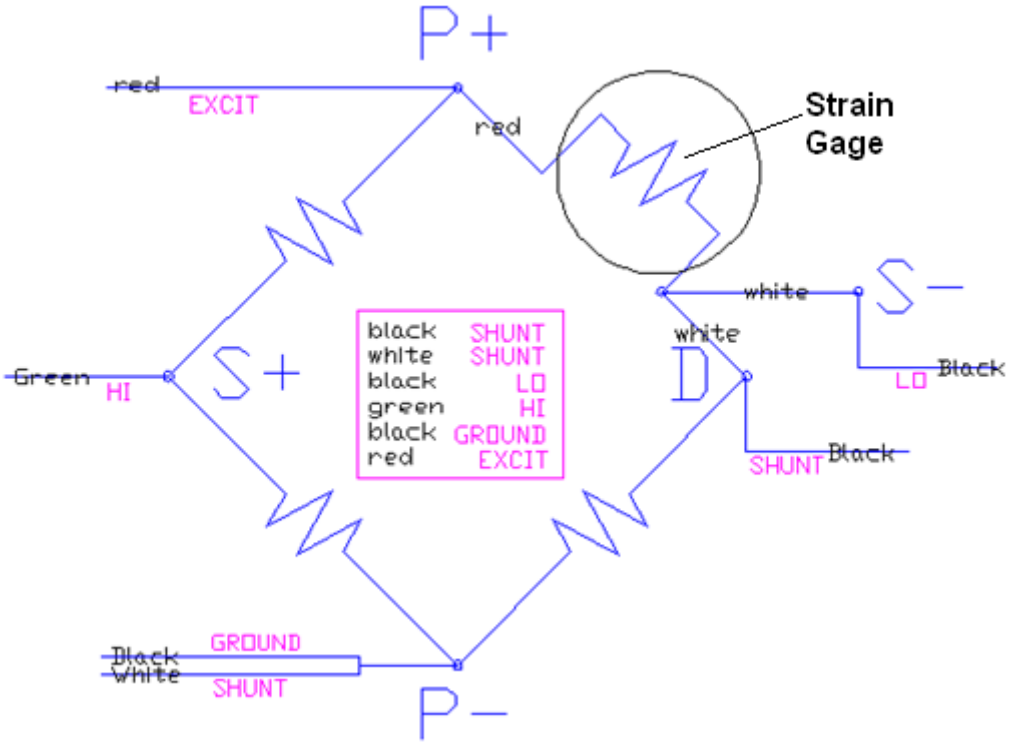


Figure 3.3 CAD drawing of a full bridge circuit

Figure 3.4 is a CAD drawing of the wiring for gage S69B and its connection to the bridge completion module. This is the exact same conceptual wiring setup as seen in Figure 3.3, except Figure 3.4 is a visual representation of how a gage is properly wired. The six output wires are in a grey multi-conductor cable connecting each gage to one of the two junction boxes. Another grey multi-conductor cable connects the junction boxes to the main enclosure where the CR9000 is housed.

Pictures of the main enclosure and the south junction box are both shown in figures in the next section.

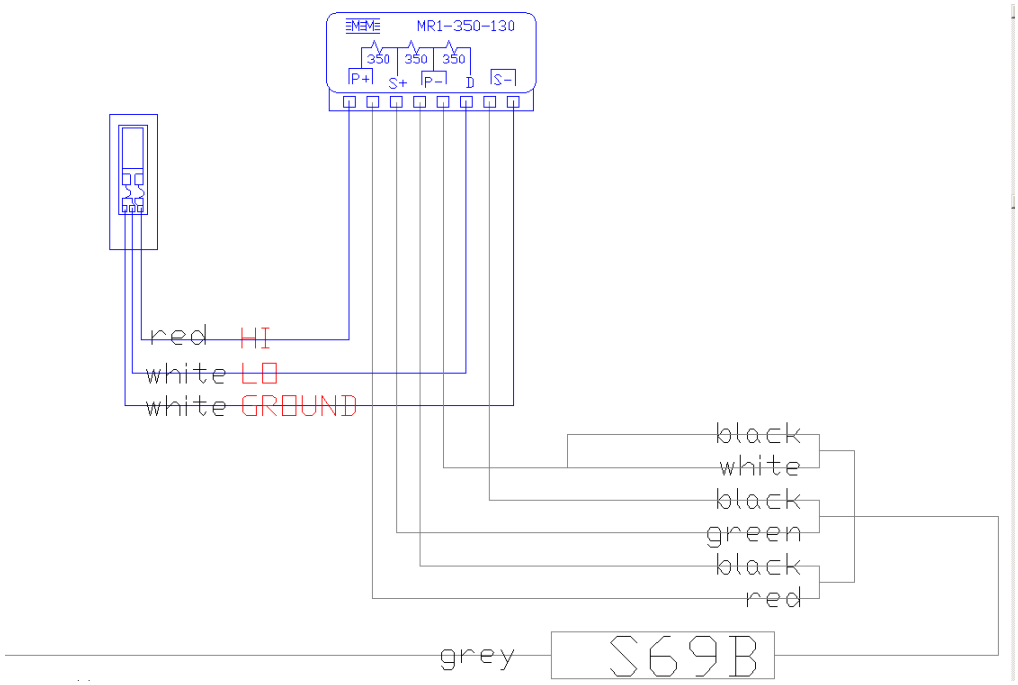


Figure 3.4 CAD drawing of strain gage S69B wired to a bridge completion module

3.2 Site Repairs

3.2.1 August 11-12, 2009

Work commenced in August of 2009 to repair sensors that were not operational. Since the bottom of the girders is about 40 feet above the ground level, assistance was necessary to reach the gages for repairs. DelDOT was able to provide a 50’ bucket truck typically used for inspections to aid in this task, and repairs began on

August 11, 2009. The boom on the truck used, seen in Figure 3.5, was not long enough to reach all the gages or the north junction box, but was sufficient to check the south junction box and 8 of the 14 problem gages.

Figure 3.5 DelDOT 50' Boom Bucket Truck

The first task was to record the wiring setup in the main enclosure and verify that it was consistent with previous records. Next the bucket truck was used to access the south junction box, where the wiring setup was again recorded and matched up with the wiring from the main enclosure. Changes in the junction box are easier to implement than in the main enclosure because they are simply wrapped around terminal strips rather than being plugged into port sockets at different levels. A picture of the main enclosure can be seen in Figure 3.6 and the south junction box can be seen

in Figure 3.7. Several discrepancies were found between the junction box and the main enclosure, so some minor changes were made in the south junction box to correct them.

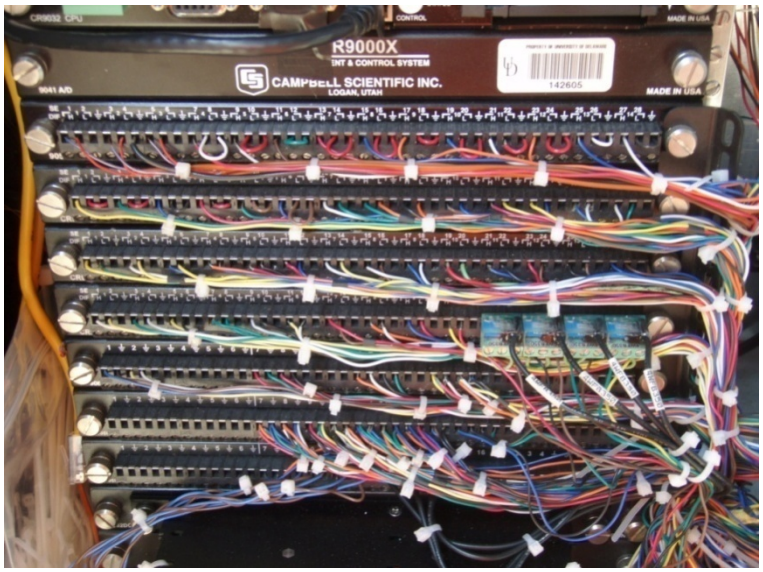


Figure 3.6 Wiring in the main enclosure

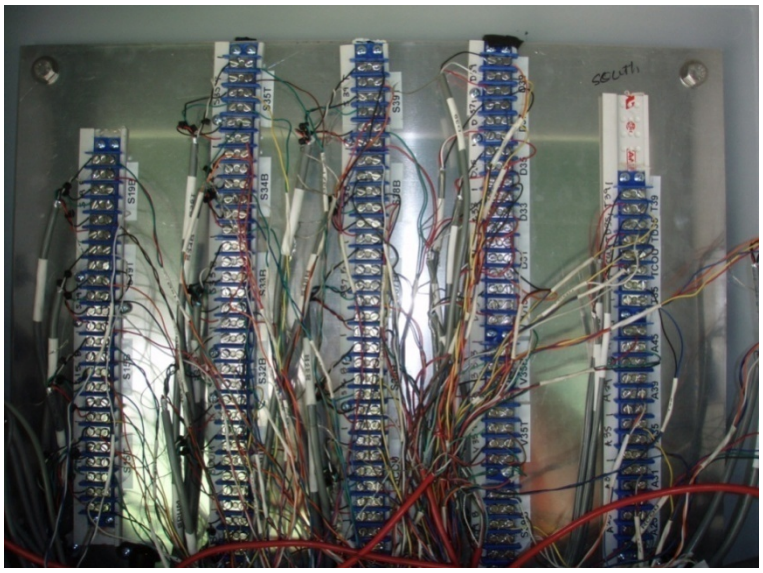


Figure 3.7 Wiring in the south junction box

Before attempting to repair the gages that were not working, we first needed a template for the correct wiring of a bridge completion module. So the next step was to record the wiring of three gages known to be working, S36B, S37B and S38B, and come up with acceptable wiring configurations for the modules. Then the wiring for the gages that were either not working or had unexpected resistances were recorded and compared to the wiring guidelines developed from the working gages. Necessary changes were identified and completed in the afternoon of August 11 and the morning of August 12.The limited range of the bucket truck only allowed for following gages to be looked at and fixed: S31B, S32B, S33B, S34B, S35B, S35T, S39B, and S39T.

There are a few common issues that were identified with the modules of these problem gages. The first and most frequent issue was that wires were simply soldered to incorrect tabs on the module, which resulted in a circuit that was

prematurely closed or not closed at all. Another prevalent issue was loose wires that would no longer have contact with the correct tabs on their module, again preventing the circuit from closing. The final issue was running solders between adjacent tabs on the completion module. This also prematurely closed the bridge circuit, by causing the equivalent effect of two wires on the same tab. This was fixed by cleaning the old solder off with a knife or soldering iron, and re-soldering tabs to ensure they were separately bonded.

Another possible cause for unexpected gage behavior was wiring mistakes elsewhere on the bridge. This was more difficult to identify then simple module errors and was worked into the troubleshooting process. This process began by recording the completion module wiring and considering potential changes. After making the necessary changes, the resistances were then checked again to see if the repairs were effective. If the bridge completion module was correctly wired and an unexpected resistance was still observed, then changes needed to be made to wiring elsewhere on the bridge. The four possible areas for wiring changes include: at the completion module, at the input to the junction box, at the output to the junction box, and at the main enclosure. Wiring changes were made in different places for different gages, depending on what issues were identified and where the easiest place for necessary repairs happened to be.

3.2.2 September 29-30, 2009

At the end of September 2009 DelDOT provided another two days with a different inspection truck. This time it was a UBIV truck that would park on the bridge deck and swing its boom and bucket under the bridge, allowing for a much wider range of access to the girders and gages. The problem with the UBIV truck is that it

required both lane and shoulder closings for about a mile approaching the truck. This disrupted traffic on I-495, but more importantly it didn't allow for any flexibility with regard to moving the truck from one side of the bridge to another. So on September 29-30 the right lane and shoulder were closed for work on gages on the east side of the bridge.

The first thing that was done on the 29th was recording the wiring in the north junction box and verifying its wiring with the wiring in the main enclosure. Several loose wires were discovered and re-attached in the north junction box before continuing with troubleshooting elsewhere. The same wiring templates that were developed in August were again used for comparing to the problem gages. A total of 11 gages were worked on during this trip, they included: S55B, S55T, S59B, S59T, S65B, S65T, S67B, S69B, S69T, S79B, and S79T. S79B and S79T can be reached by walking up the north abutment, so the truck wasn't necessary for these repairs. Not all of these gages were fixed on this trip, however, because the not all the problems could be identified and several gages that appeared to be fixed again gave unexpected readings.

3.2.3 December 8, December 10 2008

Several months later DelDOT again provided the UBIV truck, but this time the left lane and shoulder were closed for repairs to gages on the west side of the bridge. These gages were S61B, S63B, and S65T, which had been worked on before. Again the north junction box was checked to make sure nothing had changed and would indicate that gages may be working incorrectly; again nothing was found to suggest that S65T should be recording incorrect resistances. S61B and S63B were both repaired by re-soldering the gage wires to the bridge completion module in the

correct configurations. Several different combinations were attempted with S65T, but with no success.

Rain prevented the UBIV from being available on December 9, so the right lane closure was postponed until December 10. The same UBIV was used for the 10^{th} as the 8^{th}. S39T was found to have a loose wire on its bridge completion module, so that was re-soldered and the gage began working properly again. After exhausting all available troubleshooting options with gage S65T, it was concluded that the bridge completion module may have been damaged and was affecting the measured resistance. The solution to this problem was to remove the old module and order a new one to be installed at a later date. This replacement was done during the June trip to the bridge.

A semi-controlled load test using the closed lanes and the UBIV truck was also conducted during the December dates with the help of DelDOT driver Rick Moore. The details regarding how the load test was conducted, along with results, can be found in Appendix C.

3.2.4 June 7-8, 2010

The UBIV was again reserved and used for June 7 and 8 of 2010. The goal of this trip was to replace the damaged bridge completion module for strain gage S65T and to replace the protective boxes for each of the strain gages worked on during previous trips. The repair to S65T was completed on June 7 and afterwards it was verified that the gage was again working properly. The rest of the time at the bridge was spent caulking protective cases over gages and making sure their wiring was secured so that future wiring issues could be avoided. June 7 was spent with a right

lane closure, so the gages on the east side of the bridge were closed up. There was a left lane closure on June 8 to caulk the gages on the west side of the bridge.

3.3 Troubleshooting CR9000 Wiring Issues in Main Enclosure

Before testing wires and making any changes to the system in the main enclosure, the "as built" conditions should always be accurately recorded. This allows the user to have a wiring plan to go back to should any potential changes cause additional problems in the monitoring system. It is also important to always write a detailed description of what changes are being made and where in the system. Noting why changes were made is also encouraged when it is possible for future troubleshooting issues.

To check the wiring status of a gage, the user must have a multimeter capable of reading resistance in ohms. When taking readings with the multimeter the user must be careful not to touch the leads with their hands, because the human body has a resistance associated with it. This means that a hand on one or both of the leads could still produce an effective resistance reading on the multimeter even if the wire circuit is open and should not have a finite resistance value.

Once the wiring in the main enclosure is accurately recorded and the user is ready to take readings, the CR9000 should be powered off, because resistances will vary significantly if it is not. The six wires corresponding to the gage to be checked should all be isolated and removed from their slots in the CR9000 circuit board. It is very important that the excitation wires are disconnected from their slot before any readings are taken. The excitation slot has a metal bar that runs along the inside of the module, essentially connecting the excitation wires for every gage in the excitation ports. Therefore if the excitation wires are not removed, the user will likely be reading

the effective resistance for all gages connected to the excitation ports, not the gage of interest.

After the wires are disconnected the user must identify the location of P- and S- (or D), by finding wires that connect to the same location as shown previously in Figure 3.3. This can be done by reading the resistance between different combinations of wires until the correct resistance is found. Table 3.1 provides a list of what resistances should be observed between certain wire combinations. Once the same wire locations have been identified, the user should then find the wires opposite P- and S- by again testing combinations until 356 ohms resistance is found. One set of wires producing 356 ohms is the excitation wire pair (P+ and P-) and the other set of wires producing 356 ohms is the analog wire pair (S+ and S-). By using Table 3.1 and the picture of a working module and gage from Figure 3.3, the user should be able to construct a representation of the circuit and identify potential issues. If values other than those found in Table 3.1 are observed, then an unknown short has occurred somewhere in the circuit. It is possible to reconfigure wires in the main enclosure to fix these shorts, but if that doesn't work then the problem could be in the junction box or more likely directly at the gage site. Gages with identified shorts should be marked in field notes and checked more thoroughly with the UBIV if possible.

Table 3.1 Resistance output for wire combinations

Name	Wire connection	Resistance (Ω)
Excitation	S+ and (S- or D)	356
Analog	P+ and P-	356
Shunt	P- and (S- or D)	240
Same Location	(P- and P-) or (S- and S-)	~7-9
Unknown Short		Anything else

The wiring chart has designated certain color wires to be excitation or analog wires, but they can be changed if necessary. As long as two different sets of wires are clearly identified as having an effective resistance of approximately 356 ohms, then a correct wiring combination has been determined. The tables below list all the gages for the monitoring system, whether they are currently reading data, and whether their wiring has been verified. For the strain gages, Table 3.2 includes a column for gage status and wiring verification. Gage status is described as 'working', 'irregular', or 'questionable'. Working is for gages that currently have no problems, irregular is for gages that work properly sometimes, and questionable describes gages that are recording data but due to wiring issues are not necessarily reliable. The wiring verification column describes gages as either 'OK' or 'Recheck', depending on the current gage status and what previous attempts to repair have yielded.

Table 3.3 contains the list and status of the accelerometers used in the monitoring system. Table 3.4 lists the status of the vibrating wire strain gages, Table 3.5 lists the status of the temperature gages, and Table 3.6 lists the displacement gages. The displacement gages are currently not operational, but they still record number values in the monitoring and event data.

Table 3.2 Foil strain gage status table

Strain Gage	Gage Status	Wiring Verified
S15T	Working	OK
S15B	Working	OK
S19T	Working	OK
S19B	Working	OK
S31B	Working	OK
S32B	Working	OK
S33B	Working	OK
S34B	Working	OK
S35T	Working	OK
S35B	Working	OK
S36B	Working	OK
S37B	Working	OK
S38B	Irregular	Recheck
S39T	Irregular	Recheck
S39B	Working	OK
S55T	Working	OK
S55B	Working	OK
S59T	Working	OK
S59B	Working	OK
S65T	Questionable	Recheck
S65B	Working	OK
S69T	Working	OK
S69B	Working	OK
S75T	Working	OK
S75B	Working	OK
S79T	Questionable	Recheck
S79B	Questionable	Recheck

Table 3.3 Accelerometer status table

Accelerometer	Gage Status
A25	Working
A31	Working
A35x	Working
A35y	Working
A39	Working
A45	Working
A65x	Working
A65y	Working

Table 3.4 Vibrating wire strain gage status

VBW strain gage	Gage Status
V55	Working
V65	Working
V35T	Working
V35B	Working

Table 3.5 Temperature gages

Temperature Gage	Gage Status
TD35	Not Working
T65	Irregular
TCOU	Not Working
R35	Working
RCOU	Working
R39	Working

Table 3.6 Displacement Gages

Displacement Gage	Gage Status
D15	Not Working
D31	Not Working
D33	Not Working
D35	Not Working
D37	Not Working
D39	Not Working
D55	Not Working
D65	Not Working
D75	Not Working

Chapter 4

DATA PROCESSING AND CODE DEVELOPMENT

Once the monitoring system was in place and its components were repaired to a functional level, the next task became data processing. The first step in this process was making sure the data being recorded to the data logger was actually the data desired for recording. Once that was established, it was necessary to develop a method for converting raw data to usable tables and graphs. Two different software programs were used to complete these tasks, PC9000 and MATLAB.

4.1 PC9000 Code

PC9000 is the software that was primarily used for communicating with the Campbell Scientific CR9000X data acquisition system. The interface for this software makes it possible to view data in real-time, download the logger files that are being recorded, and to develop new code to manipulate the way the CR9000 records data. The CR9000 is continuously logging data but only records this data to tables when the PC9000 code tells it to. The two types of data tables being used for this monitoring project are "monitoring" data and "event" data.

4.1.1 Past Work

4.1.1.1 Event Data

The event data is meant to be taken when a certain threshold is reached for a "trigger" gage, signaling to the CR9000 data logger that an event has occurred. This

data can then be used to monitor truck traffic trends over the course of time, as well as using individual events for dynamic analysis. Before writing the code to capture these events, however, certain parameters had to be calculated as a starting point. It was decided that a scan rate of 100 samples per second was required to accurately represent data trends. Considering the length of the bridge and speed limit that passing traffic should be observing, it was calculated that it would take about 3 seconds for a truck to cross Bridge 1-821. An extra second was added to make sure that all event peaks were captured, resulting in each event being represented by a 4-second long data file.

Reader developed the first PC9000 code and Natale used that code as a guide and wrote new algorithms for the CR9000. Natale's event code required three input parameters in the event call table: number of pre-trigger records, a trigger threshold, and the number of post trigger records. An example of this event call table code can be seen below:

```
DataEvent(200, S35B>=100,200)
```

The code above tells the CR9000 that once the strain gage S35B exceeds a value of 100 an event has occurred and data must be written into the event data table. Since the logger is always scanning data, it has a back log of data for each gage. The first number tells the logger to take 200 records before the trigger and the last number tells it to take 200 records after the trigger. So including the initial trigger data point, each event contains a total of 401 records. Once an event is triggered, data is taken for the nine displacement gages, eight accelerometers, and twenty-seven foil strain gages.

These events are then stored sequentially in the event data table until the user downloads and processes them.

This code was simple and easy to use but it was soon found to have significant limitations. The biggest problem was with the temperature induced strain of the gages and the fixed trigger threshold. After trying several different trigger values and watching how many events were being recorded, it was found that 100 microstrain was an appropriate threshold to get a consistent number of events. The problem was then identified over a period of time by watching the number of events being recorded while PC9000 code with the same trigger threshold was implemented. It was observed that sometimes there may be 300 events in a day, while other times there may be only 5 events in a week. Discounting extremely irregular truck traffic patterns, the problem was identified to be temperature induced strain in the trigger gage. As the temperature increased some days and the trigger gage expanded, the strain in the gage also increased, while the trigger threshold remained the same. So if the trigger gage experienced 50 microstrain due to a temperature increase, then a truck causing a strain of only 50 microstrain to the trigger gage could trigger an event to be recorded. This explains why some days saw a large increase in event activity and while others saw none. If the rational were reversed and a temperature decrease caused a negative strain in the trigger gage of 50 microstrain, then a truck capable of causing 150 microstrain in the trigger gage would be necessary to trigger an event. This issue was dealt with by developing PC9000 code with a “floating” event trigger that incorporates the monitoring data discussed in the next section.

4.1.1.2 Monitoring Data

The other type of data table that is captured and recorded by the PC9000 code is monitor data. This data will be primarily used to study long term trends on the bridge. Monitoring data is recorded every hour and is collected using a slow scan rate. Data is collected over about a ten second period and averaged for every gage for each monitoring record. Each record is placed sequentially into a monitoring data table on the PC card in the CR9000. This data is downloaded in the same manner as the event data, but all the monitoring records are contained in a single file rather than the individual files used for the event data. Each record contains data from the temperature gages, accelerometers, foil strain gages, and vibrating wire gages.

Natale also developed the first PC9000 code to collect monitoring data, but this first version had some issues as well. The main issue was that rather than recording 1 average per hour for each gage, the logger would record 4 averages for each gage every hour. These 4 averages were all taken in quick succession and did not provide any more useful data then 1 monitoring record an hour. This was also a large waste of storage space on the PC card where the data tables were being saved, as 4 times the memory was necessary every hour to save 4 monitoring records as opposed to the 1 that was desired. This was a serious problem early in the project, because there was not much memory available on the PC card and a bulk of that was dedicated to unnecessary records.

4.1.2 Current Work

4.1.2.1 Event Code

Before solving the event trigger problem, it was first necessary to get a better grasp of the correlation between temperature changes and the corresponding

strains associated with them. It was decided that the best way to do this was to collect a great deal of monitoring data and observe the trend between temperature readings and strain in the trigger gage S35B. Collecting a significant amount of monitoring data proved difficult, however, because the memory on the PC card was auto-allocated by the PC9000 code and was therefore dominated by event data and needless monitoring records. The event data receives precedence over monitoring data for memory allocation because the scan rate for the event data is much faster. To compensate for this a theoretically unattainable trigger threshold of 100,000 microstrain was set to ensure that event data wouldn't be recorded, thus freeing memory to record more monitoring data. This code was used for a about a month and a half in November to December. The data collected over this time period was then used to create plots of both temperature and the strain of S35B. Although it was already known that there was considerable variation in temperature from day to night, this process quantified that knowledge and proved that the temperature fluctuations were quite significant.

There were several different ideas considered to solve the problem caused by temperature variations and their effects on the trigger gage and thus the event data. The first idea was to create a 'floating baseline' that would average monitoring data from the trigger gage over a 4-6 hour period and shift the trigger threshold with it. The month of monitoring data collected indicated this would be a poor solution, however, as there was still significant fluctuations in strain in some 4-6 hour time periods, mainly when the sun was coming up or going down. The next solution that was considered was to write new PC9000 code that would change the trigger threshold depending on the last temperature that was recorded in monitoring data table. This would require setting up a table of temperature ranges and programming the data

logger to record at different thresholds depending on what temperature range was identified. This also proved to be a flawed solution, as it required much more data to develop necessary correlations between temperature changes and strains caused by them. After analyzing the monitoring data collected, it was also evident that the temperature gages were not reliable enough to make this a good solution. For several long stretches of time during this month or so of data, the temperature gages simply recorded 'NAN,' meaning the gage was not working properly.

The final solution that was eventually implemented combined the first two ideas. This solution involved writing PC9000 code, with the aid of Campbell Scientific, that would refer to only the last hour's monitoring data for S35B rather than 4-6 hours. It was determined that the temperature drift during one hour was not too significant, and therefore provided much more accurate estimations than the baseline established by the larger window of 4-6 hours. A few different strain values were used for a period of several weeks and the number of events they logged was recorded. It was concluded at the end of this period that 110 microstrain was an appropriate threshold for the trigger gage. So once the strain value from the monitoring table is obtained, 110 microstrain is then added to it and this becomes the new trigger threshold for event data for the next hour. This means that over the course of a day, it is likely that 24 different trigger values have been used by the data logger. The data event table call as it is used in the current PC9000 code can be seen below:

```
DataEvent (200,(S35B>= TrigVar OR S36B>= TrigVar OR S34B >=
TrigVar),True,200)
```

Because gages sometimes go offline and will record 'NAN' for several hours, additional logic was included to make the event data code more robust. This logic first checks the monitoring table for a number value for S35B in the last hour's record. If a number is not present, then the code then checks the adjacent gage, S36B, for a number value in the record. If a number is present, then the data logger uses S36B as the trigger gage for that hour. If the last hour of data happens to also have a 'NAN' for S36B, then the logger checks S34B in a similar process. With this PC9000 code implemented the logger is capable of accurately recording appropriate events at all times except the extremely unlikely case where gages S35B,S34B, and S36B have all stopped working during the same hour. In the event that all three gages do stop working however, logic has been included in the TrigVar definition that assigns an arbitrary value of 1111 microstrain so that the program will not crash and continue to at least record monitoring data for the hour. The program would then recheck the trigger gages the next hour to reestablish an appropriate trigger for an event. The logic that performs these checks can be seen below:

```
If ReadAVG = 101 Then
      If S35BAvg <>0 Then
        TrigVar = S35BAvg + 110
        ElseIf S36BAvg<> 0 Then
            TrigVar = S36BAvg + 110
        ElseIf S34BAvg <>  0 Then
             TrigVar = S34BAvg + 110
             ElseIf S35BAvg  = NAN Then
                 TrigVar = 1111
             EndIf
   ReadAVG = 0
EndIf
```

4.1.2.2 Monitoring Code

The problem with the monitoring data was also solved by implementing new PC9000 code to alter the way data was recorded. With the help of Campbell Scientific, the code was analyzed in detail and it was discovered that a secondary file created by the PC9000 code was governing the monitoring code and causing 4 records an hour to be recorded. Once this secondary code was fixed the monitoring data began being recorded as it was intended to. A larger memory card was also purchased, increasing PC card memory from 256 MB to 2 GB. This increase in memory allows for both types of data to be recorded for longer periods of time without worrying about the preference of memory auto-allocation.

4.2 Data Formatting and Processing

4.2.1 Event Data

The format and download procedure for the event data required a bit more consideration and several different programs for processing and analyzing the data. The event data table is downloaded from the PC card by going to the 'logger files' window and highlighting 'EVENT.dat' and clicking retrieve. The user is then prompted to set a name and directory path for the event data file, after which the download begins. Once this download is finished, the user is left with a data table in binary that contains all of the events for that particular download. The data can be converted to ASCII format for further processing by using the file converter included in the PC9000 software. The user must select an input binary file pathname and output ASCII file pathname. Another one of the choices in the file conversion window is 'Process file markers' and by selecting this option before converting, the data will also be separated into individual event files. Each of these event files is then saved in the

output folder with the same name as the ASCII output file, but numbered in ascending order with the format ‘filename.000.’

4.2.1.1 Event Sorting Program

The number of event files in each download also depends on the amount of time the program has run between downloads. There is not a linear correlation between time and events as there is with the monitoring data files, but the more time you leave the program running, the more likely it is that more events will occur. This lead to the first issue that had to be dealt with concerning the event data processing, which was organizing events in a logical way to be used later. If several days pass between downloads, then it is possible for events from multiple days to be saved under one filename, matching the day of the download. This becomes problematic in processing because you may end up having events from one day saved under two separate filenames, therefore splitting up the data from that day. To deal with this issue a MATLAB program was written to create folders for events from each day and to sort the events into their proper folders for further processing. This program takes in the filename of the download and scans the first line for each event file individually, comparing the year, month, and day of the event to folders already created. If a folder already exists for the year, month, or day in question, the event is simply moved into that corresponding folder. If the proper folder does not exist, then the program creates the necessary folders and then moves the file. This MATLAB program can be used for any number of event files with the same filename, and results in each event being sorted into the appropriate folder for when that event occurred. This MATLAB program is referenced as “Event Sorting Program” and can be seen in Appendix A.2.

Several screen shots of the folders created and organized by this program can be seen in Figure 4.1-Figure 4.4. The folders show the organization of events that occurred on 2/13/2010, starting with the year in Figure 4.1. Although the filenames of the events in Figure 4.4 say 2/15/2010, that was just the download date of the event data. The events themselves occurred on February 13, 2010.

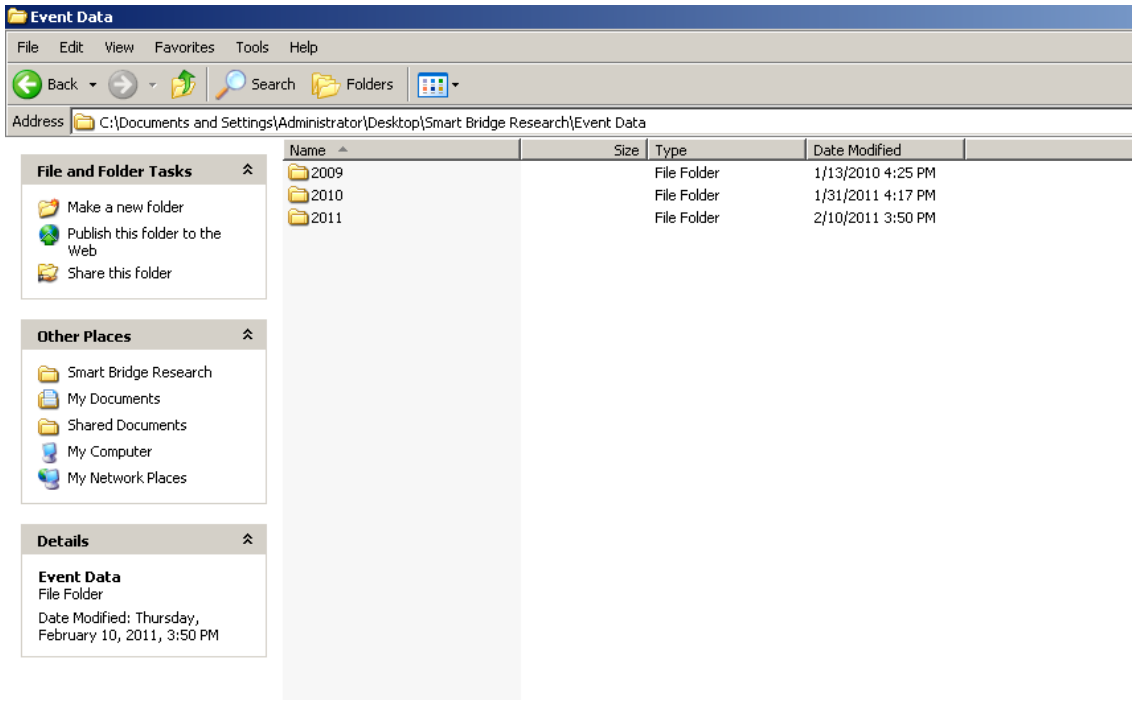


Figure 4.1 Year folders created by event sorting program

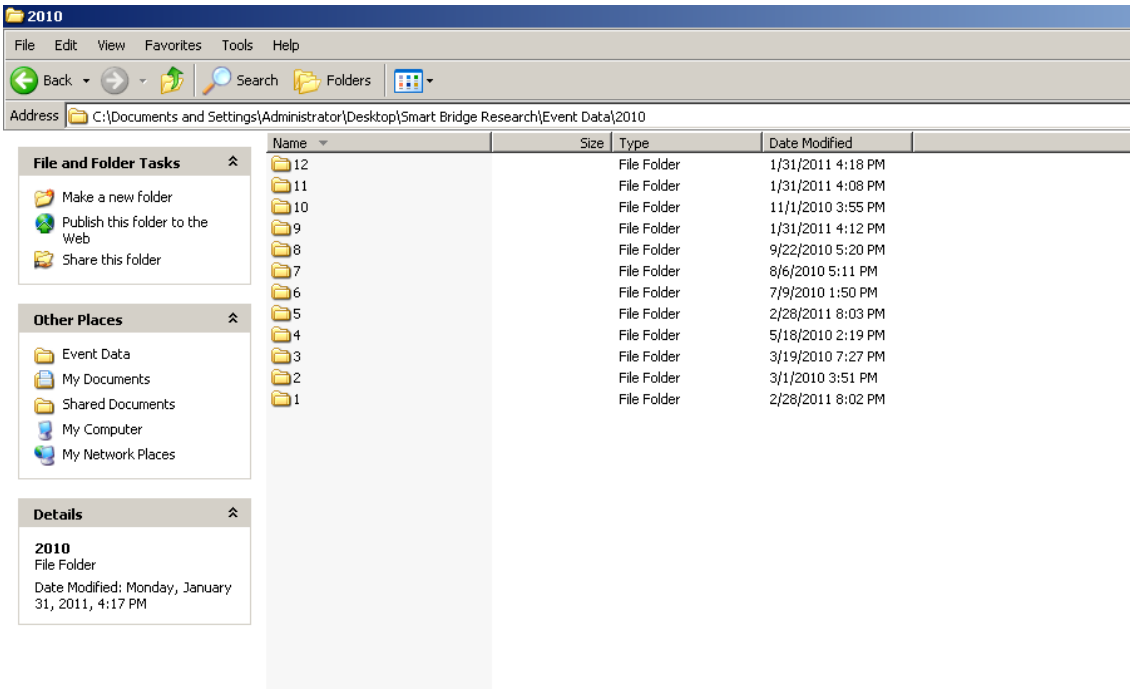


Figure 4.2 Month folders created by event sorting program

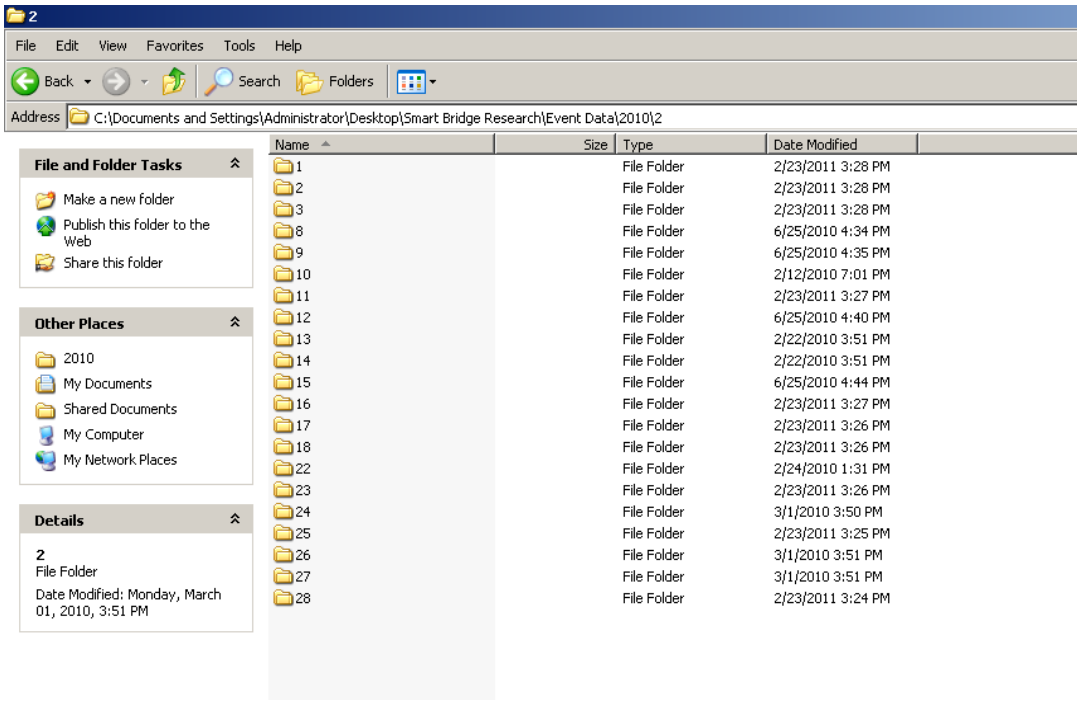


Figure 4.3 Day folders created by event sorting program

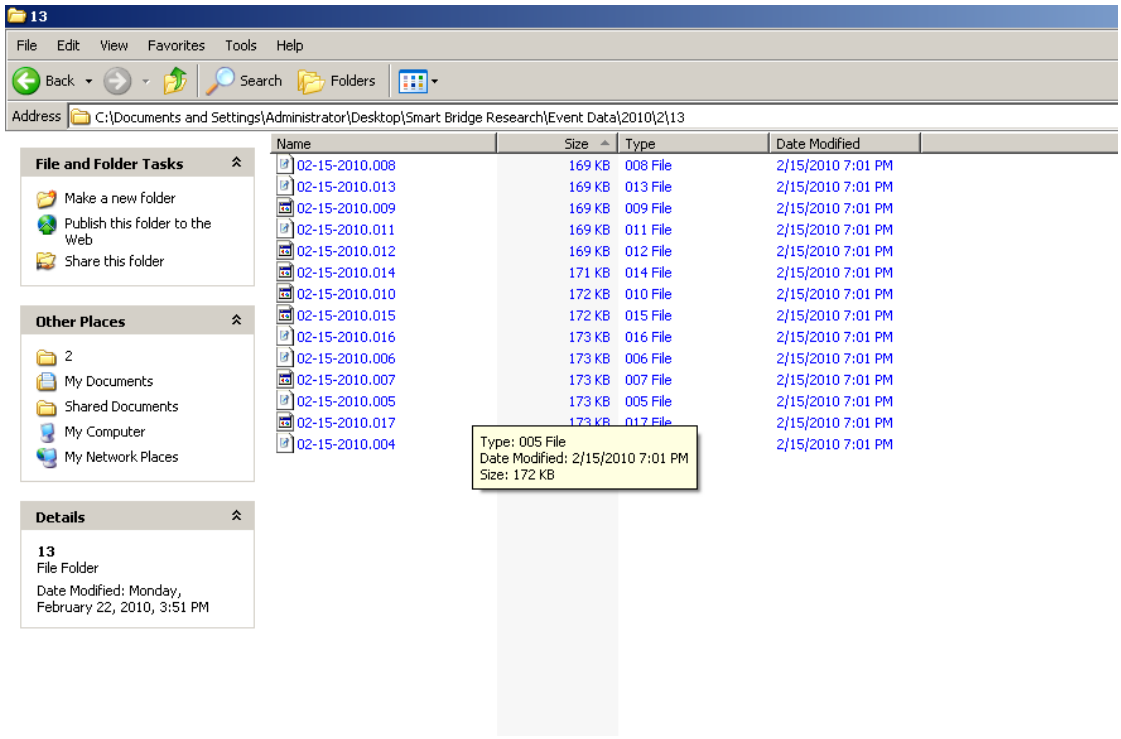


Figure 4.4 Events organized by the event sorting program

4.2.1.2 Event Processing Program

Once the event files are sorted into their proper folders, they are then ready to be passed through the next MATLAB program labeled as "Event Data Extraction Program" which can be seen in Appendix A.1. The user inputs a path to the folder that they would like to be processed and runs it in MATLAB. The program has a built in counter that numbers the files in the folder in question and uses this as a counter to 'walk' through the files. The program skips over the first four lines of data which contains header information about gages and their locations. Since the date and time is obviously different for every line of data recorded during an event, the program has a large section devoted to taking in this varying date and time information properly and storing it. The logic in this part of the program counts the characters until

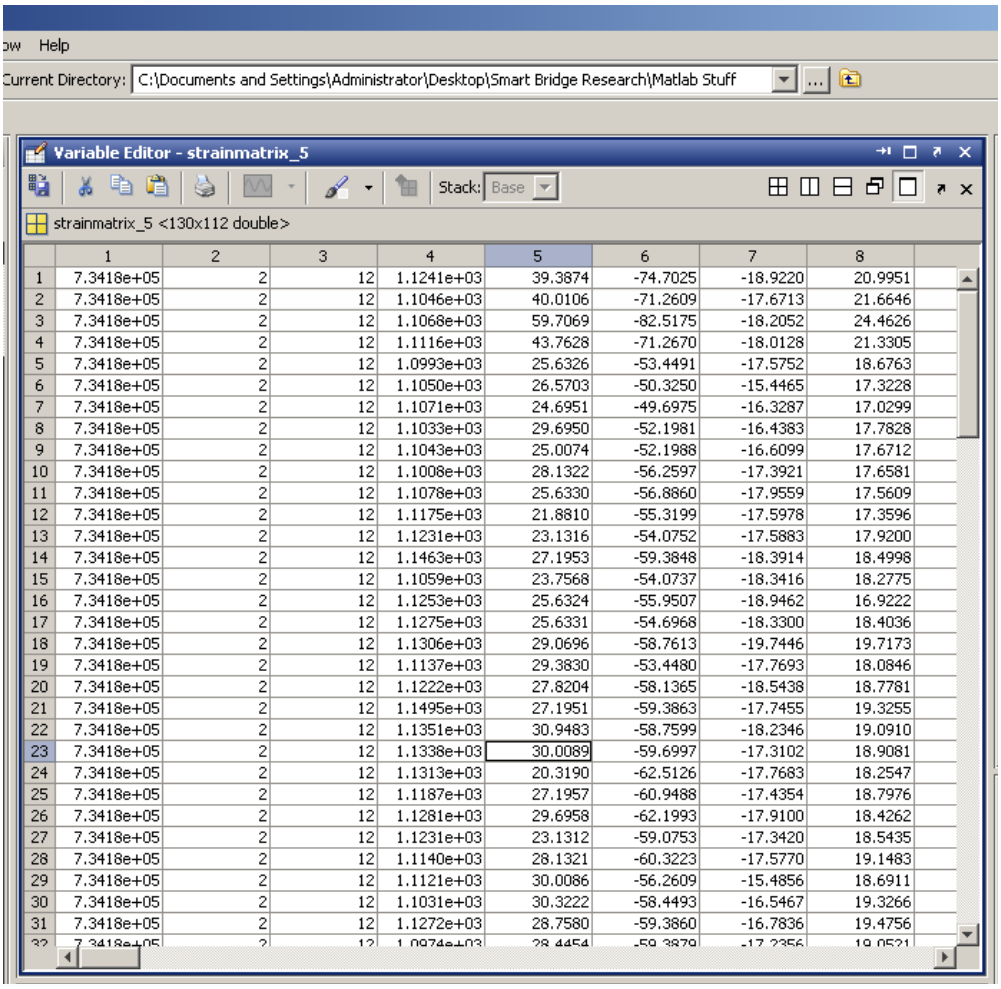
the first comma of each line is reached, signifying that the date and time information is done. Depending on how many characters it takes to get to the first comma, the program has an appropriate way to save the year, month, day, hour, minute, and second data for each timestamp individually. This data is then saved under a single variable name and converted to a single number using the 'Datenum' function in MATLAB. This number increases as the date and time put into it is "increased" so it is also stored and can later be used for graphing purposes, e.g., for plotting the weekly and monthly breakdown of events. This number can also be converted back to date and time information if need be.

Once the program has scanned over the date and time, it reads event data line by line and puts it into a matrix. The data is read in with each row corresponding to a different gage, because MATLAB is capable of reading in an infinite number of rows but the number of columns must be specified. This way if for some reason an event being processed isn't the total 401 lines of data, the program will not crash and continue to the next file. After all the lines of data are read in, the matrix is then transposed so each column corresponds to a different gage. Next the program calculates the maximum, minimum, average, and standard deviation for each gage during the event. This is done by column, so each gage has a line of code conducting the calculations for that particular column. For the gages that are not working during the event, there is a dummy vector, represented by zeros for each of the values calculated for the other gages. In the program this vector looks like this: [0 0 0 0]. This dummy vector is then used in the place of the maximum, minimum, average, and standard deviation in the final strain matrix. This is another way that the program was made more robust. The dummy vector serves as a placeholder in the final matrix, so

that the columns of data stay in the correct order. Once all the values are calculated for each gage, the program places all of these values in a single strain matrix for easy viewing and continues on to the next event. This is done for each event in the folder being processed, so each row of the final strain matrix represents one event. Therefore the number of events in the folder being processed and the number of rows in the final strain matrix should be the same.

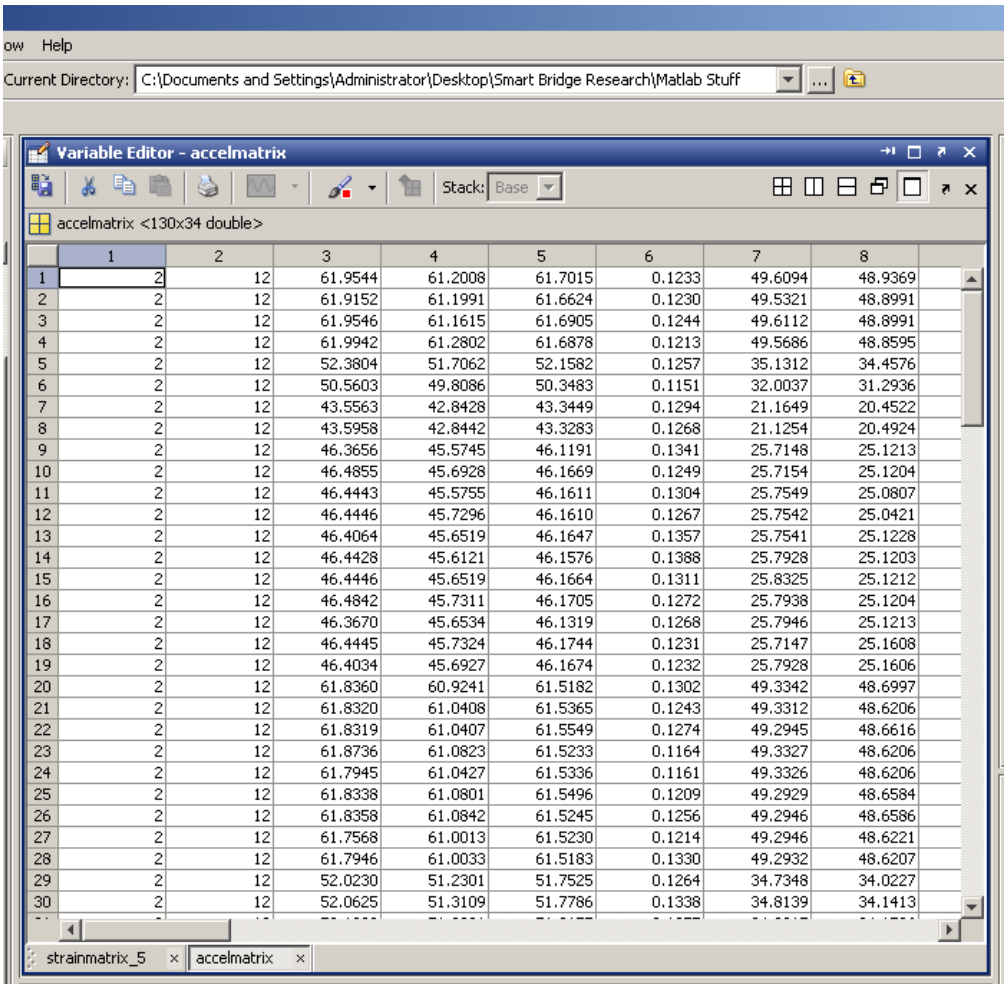
Additional data was included in this final strain matrix to make further processing easier. The first column holds the 'Datenum' for each event, so it can be graphed appropriately in reference to other events. The next two columns are the month and day of the event respectively. The fourth column is the maximum strain value that the trigger gage S35B recorded during each event. This provides a quick reference that makes comparing the magnitudes of events very easy.

The program also processes the acceleration data in a similar fashion as described above. A resulting final acceleration matrix is the output, with the first two columns corresponding to the month and day of the event respectively. A screen shot of what the final strain matrix and the final acceleration matrix look like in MATLAB can be seen below in Figure 4.5 and Figure 4.6.



	1	2	3	4	5	6	7	8
1	7.3418e+05	2	12	1.1241e+03	39.3874	-74.7025	-18.9220	20.9951
2	7.3418e+05	2	12	1.1046e+03	40.0106	-71.2609	-17.6713	21.6646
3	7.3418e+05	2	12	1.1068e+03	59.7069	-82.5175	-18.2052	24.4626
4	7.3418e+05	2	12	1.1116e+03	43.7628	-71.2670	-18.0128	21.3305
5	7.3418e+05	2	12	1.0993e+03	25.6326	-53.4491	-17.5752	18.6763
6	7.3418e+05	2	12	1.1050e+03	26.5703	-50.3250	-15.4465	17.3228
7	7.3418e+05	2	12	1.1071e+03	24.6951	-49.6975	-16.3287	17.0299
8	7.3418e+05	2	12	1.1033e+03	29.6950	-52.1981	-16.4383	17.7828
9	7.3418e+05	2	12	1.1043e+03	25.0074	-52.1988	-16.6099	17.6712
10	7.3418e+05	2	12	1.1008e+03	28.1322	-56.2597	-17.3921	17.6581
11	7.3418e+05	2	12	1.1078e+03	25.6330	-56.8860	-17.9559	17.5609
12	7.3418e+05	2	12	1.1175e+03	21.8810	-55.3199	-17.5978	17.3596
13	7.3418e+05	2	12	1.1231e+03	23.1316	-54.0752	-17.5883	17.9200
14	7.3418e+05	2	12	1.1463e+03	27.1953	-59.3848	-18.3914	18.4998
15	7.3418e+05	2	12	1.1059e+03	23.7568	-54.0737	-18.3416	18.2775
16	7.3418e+05	2	12	1.1253e+03	25.6324	-55.9507	-18.9462	16.9222
17	7.3418e+05	2	12	1.1275e+03	25.6331	-54.6968	-18.3300	18.4036
18	7.3418e+05	2	12	1.1306e+03	29.0696	-58.7613	-19.7446	19.7173
19	7.3418e+05	2	12	1.1137e+03	29.3830	-53.4480	-17.7693	18.0846
20	7.3418e+05	2	12	1.1222e+03	27.8204	-58.1365	-18.5438	18.7781
21	7.3418e+05	2	12	1.1495e+03	27.1951	-59.3863	-17.7455	19.3255
22	7.3418e+05	2	12	1.1351e+03	30.9483	-58.7599	-18.2346	19.0910
23	7.3418e+05	2	12	1.1338e+03	30.0089	-59.6997	-17.3102	18.9081
24	7.3418e+05	2	12	1.1313e+03	20.3190	-62.5126	-17.7683	18.2547
25	7.3418e+05	2	12	1.1187e+03	27.1957	-60.9488	-17.4354	18.7976
26	7.3418e+05	2	12	1.1281e+03	29.6958	-62.1993	-17.9100	18.4262
27	7.3418e+05	2	12	1.1231e+03	23.1312	-59.0753	-17.3420	18.5435
28	7.3418e+05	2	12	1.1140e+03	28.1321	-60.3223	-17.5770	19.1483
29	7.3418e+05	2	12	1.1121e+03	30.0086	-56.2609	-15.4856	18.6911
30	7.3418e+05	2	12	1.1031e+03	30.3222	-58.4493	-16.5467	19.3266
31	7.3418e+05	2	12	1.1272e+03	28.7580	-59.3860	-16.7836	19.4756
32	7.3418e+05	2	12	1.0974e+03	28.4454	-59.3879	-17.2356	19.0521

Figure 4.5 Screen shot of final strain matrix in MATLAB



	1	2	3	4	5	6	7	8
1	2	12	61.9544	61.2008	61.7015	0.1233	49.6094	48.9369
2	2	12	61.9152	61.1991	61.6624	0.1230	49.5321	48.8991
3	2	12	61.9546	61.1615	61.6905	0.1244	49.6112	48.8991
4	2	12	61.9942	61.2802	61.6878	0.1213	49.5686	48.8595
5	2	12	52.3804	51.7062	52.1582	0.1257	35.1312	34.4576
6	2	12	50.5603	49.8086	50.3483	0.1151	32.0037	31.2936
7	2	12	43.5563	42.8428	43.3449	0.1294	21.1649	20.4522
8	2	12	43.5958	42.8442	43.3283	0.1268	21.1254	20.4924
9	2	12	46.3656	45.5745	46.1191	0.1341	25.7148	25.1213
10	2	12	46.4855	45.6928	46.1669	0.1249	25.7154	25.1204
11	2	12	46.4443	45.5755	46.1611	0.1304	25.7549	25.0807
12	2	12	46.4446	45.7296	46.1610	0.1267	25.7542	25.0421
13	2	12	46.4064	45.6519	46.1647	0.1357	25.7541	25.1228
14	2	12	46.4428	45.6121	46.1576	0.1388	25.7928	25.1203
15	2	12	46.4446	45.6519	46.1664	0.1311	25.8325	25.1212
16	2	12	46.4842	45.7311	46.1705	0.1272	25.7938	25.1204
17	2	12	46.3670	45.6534	46.1319	0.1268	25.7946	25.1213
18	2	12	46.4445	45.7324	46.1744	0.1231	25.7147	25.1608
19	2	12	46.4034	45.6927	46.1674	0.1232	25.7928	25.1606
20	2	12	61.8360	60.9241	61.5182	0.1302	49.3342	48.6997
21	2	12	61.8320	61.0408	61.5365	0.1243	49.3312	48.6206
22	2	12	61.8319	61.0407	61.5549	0.1274	49.2945	48.6616
23	2	12	61.8736	61.0823	61.5233	0.1164	49.3327	48.6206
24	2	12	61.7945	61.0427	61.5336	0.1161	49.3326	48.6206
25	2	12	61.8338	61.0801	61.5496	0.1209	49.2929	48.6584
26	2	12	61.8358	61.0842	61.5245	0.1256	49.2946	48.6586
27	2	12	61.7568	61.0013	61.5230	0.1214	49.2946	48.6221
28	2	12	61.7946	61.0033	61.5183	0.1330	49.2932	48.6207
29	2	12	52.0230	51.2301	51.7525	0.1264	34.7348	34.0227
30	2	12	52.0625	51.3109	51.7786	0.1338	34.8139	34.1413

Figure 4.6 Screen shot of final acceleration matrix in MATLAB

4.2.1.3 Event Graphing Program

Once the final strain matrix is complete and the general magnitude of events has been compared, the user may want to take a more in depth look at a particular event. For this purpose another program referred to as “Event Graphing Program” can be used. This program, seen in Appendix A.3, is essentially the same as the event processing program described above in 4.2.1.2. In the event processing program, each variable is erased after it is put into the final strain matrix, by a ‘clear

all' command. This clearing is necessary to avoid errors within the program as it goes from file to file, however it makes looking at an individual event file impossible. To get around this problem, the event graphing program was created. In this program there is a user input 'stopfile' feature that literally stops the program when a certain file has been reached and processed, therefore preventing the variables and the event's data from being cleared. Once the event processor program has been used and the desired events to be analyzed further have been identified, the user opens the graphing program and inputs the necessary 'stopfile' value for the file in question.

Several lines of code at the end of this program deal with graphing the events and labeling their axes appropriately. A reference list of the gages and their corresponding column location in the strain matrix is also provided to make producing multiple graphs from one event easier. An example of a graph produced from this program can be seen below in Figure 4.7.

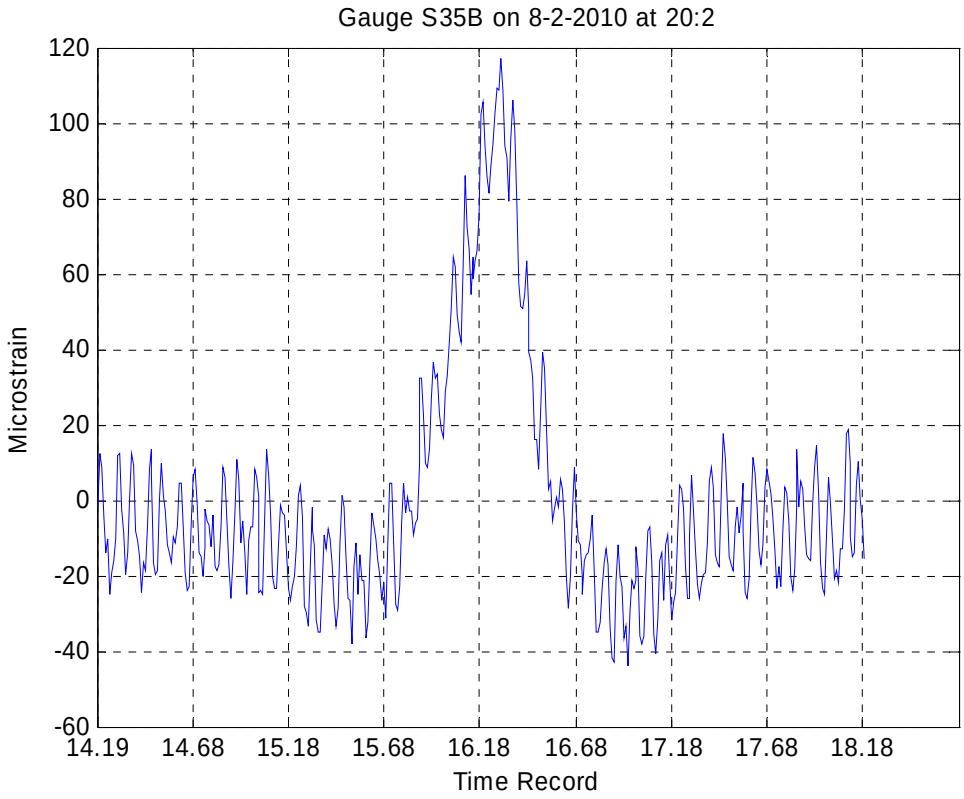


Figure 4.7 Output from the event graphing program

4.2.2 Monitoring Data

As indicated above, every hour that the PC9000 code is running, monitoring data is being logged and appended to the monitoring table on the PC card. It's obvious then that the longer the user waits between downloading data, the longer the monitoring file will become. This is not a problem in and of itself, but it does become an issue when it comes to processing these monitoring records. An individual monitoring record of say 24 hours is really only useful when it can be combined with the monitoring records taken before and after those 24 hours. It would be unreasonably difficult to try to compare temperature and strain trends over a month's

time by using 30 different monitoring records corresponding to different days in that month. This means that when it came time to process the monitoring data, it was crucial to find a way to combine all the monitoring records into a single file that can then be used to view a week, month, or year's worth of data together.

In order to fulfill this need, a MATLAB program was written that reads in monitoring files one at a time and appends them to a saved matrix file. This program is referred to as "Monitoring Processor Program" (Appendix A.4) and it requires the user to input both the filename of the monitoring data, as well as the number of files in the data file. This is necessary because the monitoring data can be any number of lines, and the MATLAB program needs to have an end point for the 'for' loop that is used to read in the data line by line. The file is first read into a separate matrix and the date and time data is converted to a single number using the MATLAB command 'datenum.' 'Datenum' converts an entire date and time record to a single number; this number will increase as the date and time data advances forward in time. This value is then stored in the first column of the data matrix and appended to the larger monitoring data matrix that contains previous files. The 'datenum' in the first column allows the user to sort the matrix in descending order, which automatically orders the data from oldest to newest. This is useful in case there are gaps in the data or if the monitoring data files are downloaded out of order.

With the data now in the correct chronological order, it is now ready to be graphed. Several lines of code are included to create graphs of individual gages over a period of time, specified by the number of records the user inputs. A list of gage's and their corresponding column number in the data matrix is included in this MATLAB

program for a quick reference when creating these graphs. An example of one of the graphs produced by this program can be seen in Figure 4.8 below.

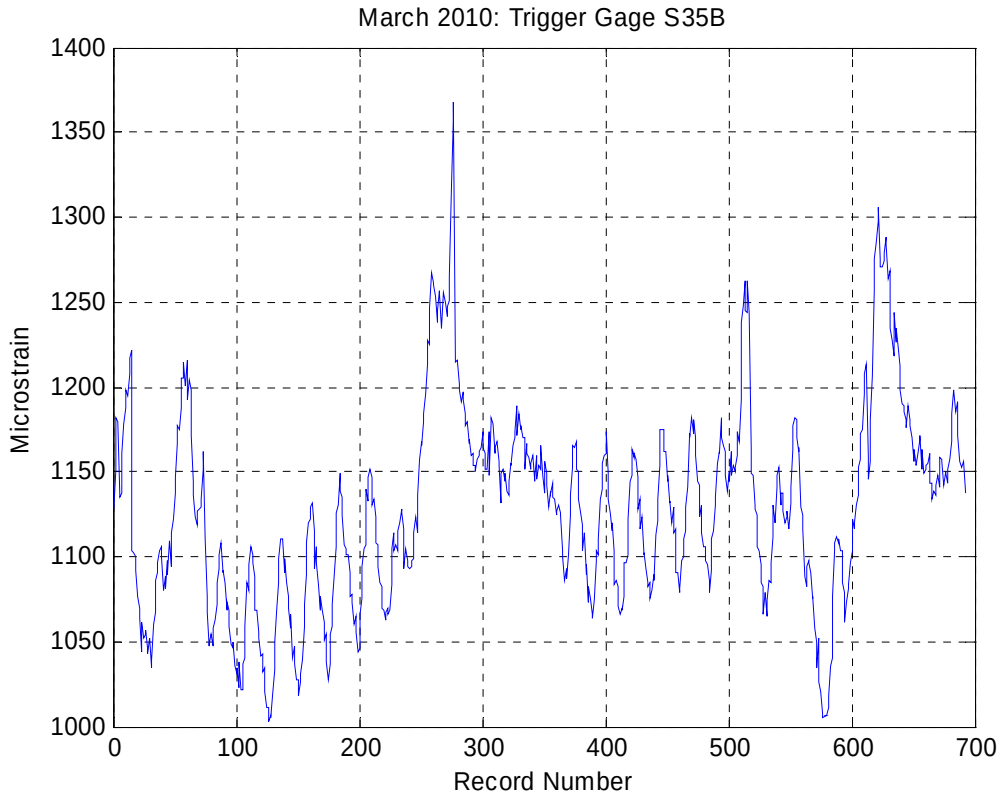


Figure 4.8 Output graph from the monitoring processor program

Chapter 5

OPERATION OF THE MONITORING SYSTEM

Presented in this chapter are instructions for how to download, process, and graph the event and monitor data. The MATLAB codes developed to complete these tasks were described in Chapter 4 and are included in their entirety in Appendix A.

5.1 Descriptions of Using the Software

5.1.1 Downloading Event Data

The process for analyzing event data begins by first downloading the data, and then converting the data to the proper format for the MATLAB programs to be effective. The following list is a step by step description of the download process for event data:

1. Open PC9000, click the 'RealTime' tab and select 'Field Monitor.'

2. Click the 'logger' box on the left side of the screen, which brings up Figure 5.1.

3. Select 'event' to monitor the event data table and see if the memory allocation for the event data is close to being full. (If

there is still space in the table, wait several days until it is closer to being full.)

4. Select the 'Tools' tab and scroll down to select 'Logger Files.' This is a window where the user can view all the data and program files on the CPU and PC card (shown in Figure 5.2.)

5. Highlight the 'EVENT.DAT' file on the PC card and then click the 'Retrieve' button.

6. User will be prompted with the window shown in Figure 5.3. Select to save the event file in 'Scratch Folder' at the top, then input the month, day, and year (in the format "mm-dd-yy") of the download as the filename.

7. Click 'Save', shown on the right side of Figure 5.3, and download will begin. (It may require several attempts to download the entire event table due to connection issues. For reference, a table size of 150,000 records takes about 45 minutes and will be about 27 mega bytes. A 'file complete' response will occur if no errors occurred.)

8. Next select 'File' on the PC9000 menu bar and scroll down to 'Convert Data Files.' This window is shown in Figure 5.4.

9. Click the question mark next to ‘Input Binary File’ line in this window and select the recently downloaded event data table from the ‘Scratch Folder.’

10. Click the question mark next to ‘Output ASCII File’ line and enter the filename as the month, day, and year in the following format, “mm-dd-yyyy.” (This is different then the input binary filename to avoid errors in later data processing.)

11. Select the ‘Process File Marks’ option shown in Figure 5.4 before converting. (The ‘Time Stamps’ and ‘Add Record Number’ options should already be selected, but always make sure they are also checked.)

12. Click ‘Convert’ and the conversion will begin.

13. Upon completion, a window will pop up with a list of all the event files that were converted. These are in the format “mm-dd-yyyy.###” where the ‘###’ numbers begin with ‘000’ and increase to whatever number of events were in the original binary event data table. A sample of this window is shown in Figure 5.5.

14. Raw data for individual events can then be viewed by selecting them in the ‘Scratch Folder.’

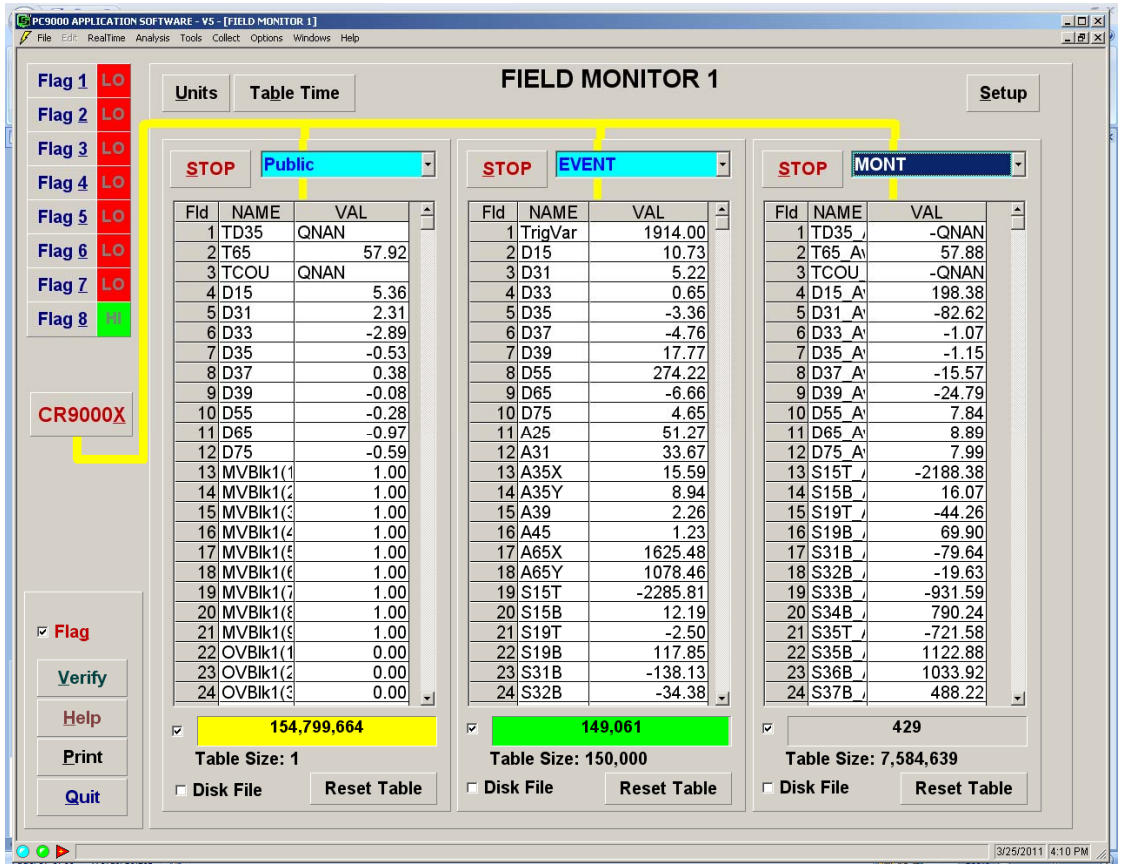


Figure 5.1 Screen shot of field monitor in PC9000

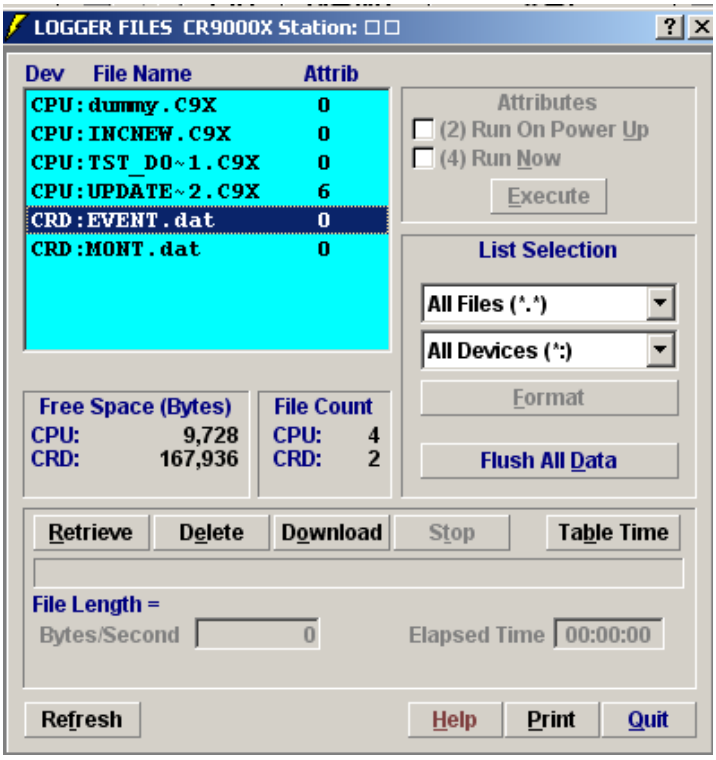


Figure 5.2 Logger files window in PC9000

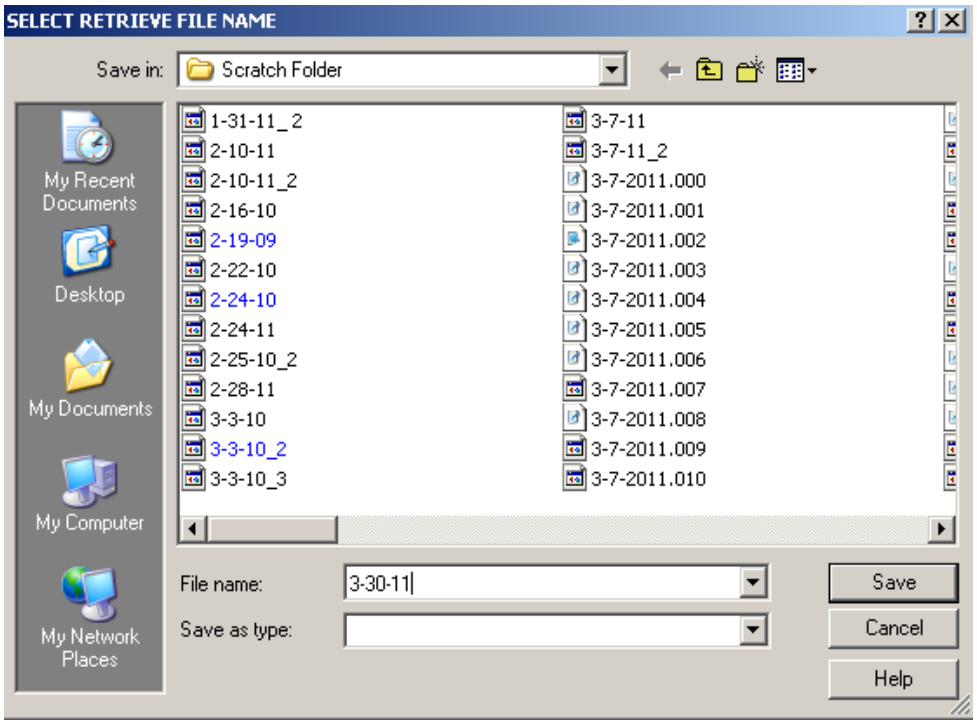


Figure 5.3 Retrieve file prompt window

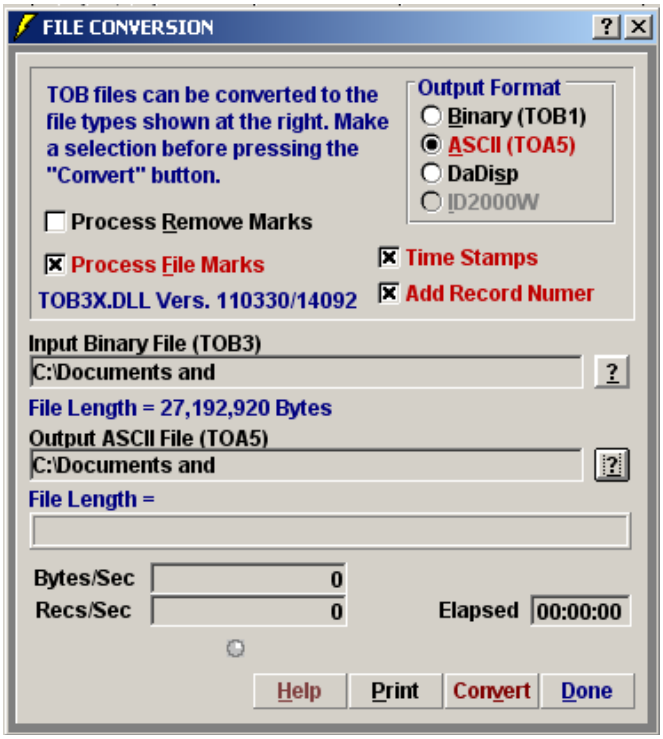


Figure 5.4 File conversion window for event data

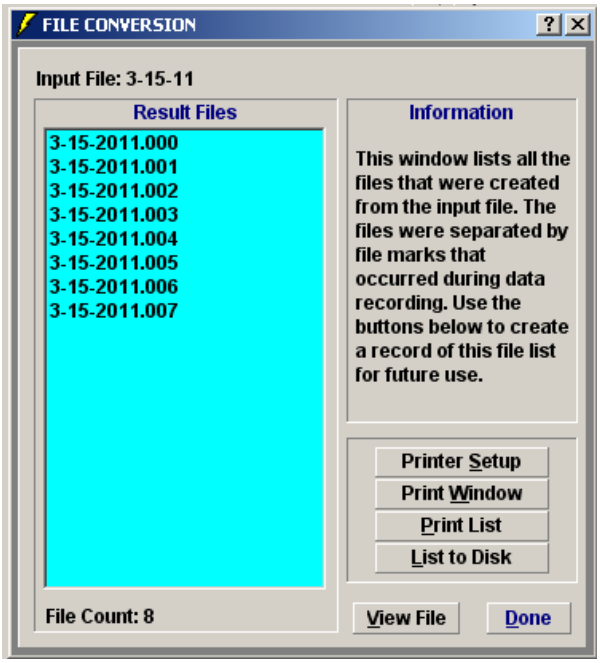


Figure 5.5 List of converted files

5.1.2 Downloading Monitoring Data

Monitoring data can be downloaded using the PC9000 software by following the steps outlined above in section 5.1.1. The only major difference would be in step 5, where 'MONT.DAT' should be selected from the PC card for retrieval rather than 'EVENT.DAT.' There is a simpler, more direct way to download the monitoring data with another software program called RTDAQ. The process for downloading monitoring data using RTDAQ is described in the following list:

1. Open RTDAQ software.

2. Select the 'Collect Data' tab on the top of the home screen. RTDAQ will communicate and connect to the logger, resulting in the window shown in Figure 5.6.

3. Next highlight ‘MONT’ in the table collection section.

4. Click the ‘Change File Name’ button at the bottom of the window.

5. Input the new filename as the date of the download in the format “mm-dd-yyyy.”

6. Click ‘File Fomat’ in the ‘Collections Options’ section (shown in Figure 5.6) and scroll down to ‘ASCII Table Data (TOA5).’

7. Check that ‘Include Timestamp’ and “Include Record Number’ are both selected in the record information section of Figure 5.6.

8. Select the box next to ‘MONT’ in the table collection section and click start collection to download data.

9. These tables are much smaller than the event data tables, so this typically does not take long. RTDAQ also has had a more reliable connection than PC9000, so there shouldn’t be issues with the port closing.

RTDAQ is not used for downloading all data currently because there have been problems thus far finding a way to download the event data into the correct format. Up to this point when RTDAQ has been used to download event data, the resulting data file has been in one very large data table, with all the events listed in sequential order. Since all of the MATLAB programs have been written for data separated into individual events, the PC9000 software should still be used until a more

reliable method of downloading and separating event data files is clearly defined within RTDAQ.

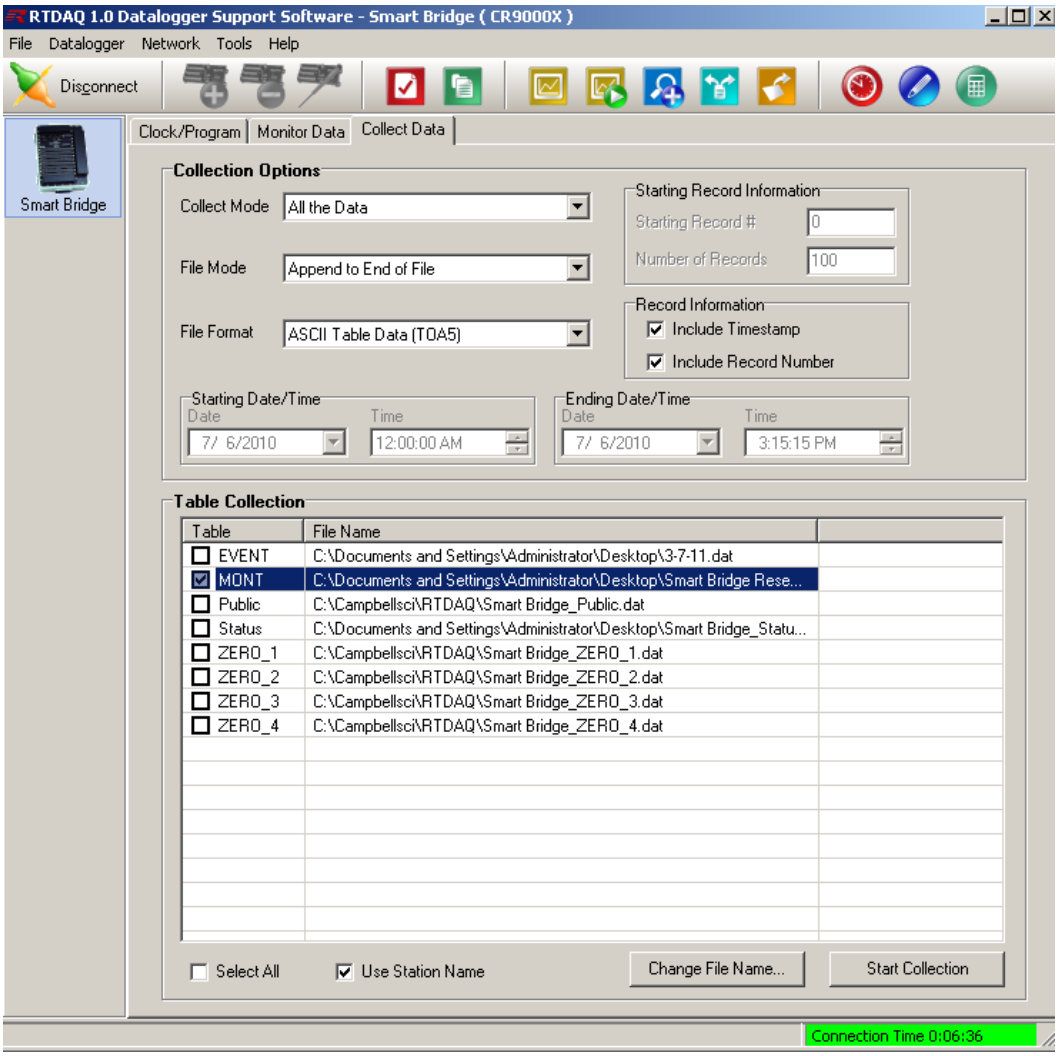


Figure 5.6 RTDAQ data collection window

5.1.3 Clearing Tables and Restarting PC9000 Program

Once the event and monitoring data tables have both been downloaded, the user will next want to erase the tables on the data logger and begin to record new data. This has typically been done in PC9000, although it is also possible to reset the program through the RTDAQ software. The following list describes the process of clearing tables and restarting the program using the PC9000 software:

1. Open PC9000 and establish a connection with the data logger

2. Select the 'Tools' tab at the top of the screen and scroll down to 'Logger Files.' (Here the user can manipulate the PC9000 programs that are run by the logger, which in turn dictate how data is recorded.)

3. Identify what program is running on the data logger CPU by looking at the 'Attribute' column in the logger file window, shown below in Figure 5.7. (The program currently running gets an attribute number of '4.' An additional attribute value of '2' is given to the program selected to run in case of a power supply loss or other system shut down.)

4. Next highlight the program currently running on the data logger and de-select both the corresponding attributes in the upper right corner of the logger files window shown in Figure 5.7.

5. Click the 'Execute' button and PC9000 will stop the program.(This may take several minutes.)

6. When all of the attribute values are set to ‘0’ the user can then clear the data by clicking the ‘Flush All Data’ button on the right side of the window. The user will then be asked if they are sure they would like to delete all data files, by clicking ‘yes’ the data will be erased from the logger. (Again this may take a few minutes.)

7. Once the data is cleared, the user can highlight the appropriate program to run and re-select the boxes next to the attributes in the top right of the window.

8. Click ‘Execute’ and the PC9000 will load the highlighted program and begin using it to record new data immediately.

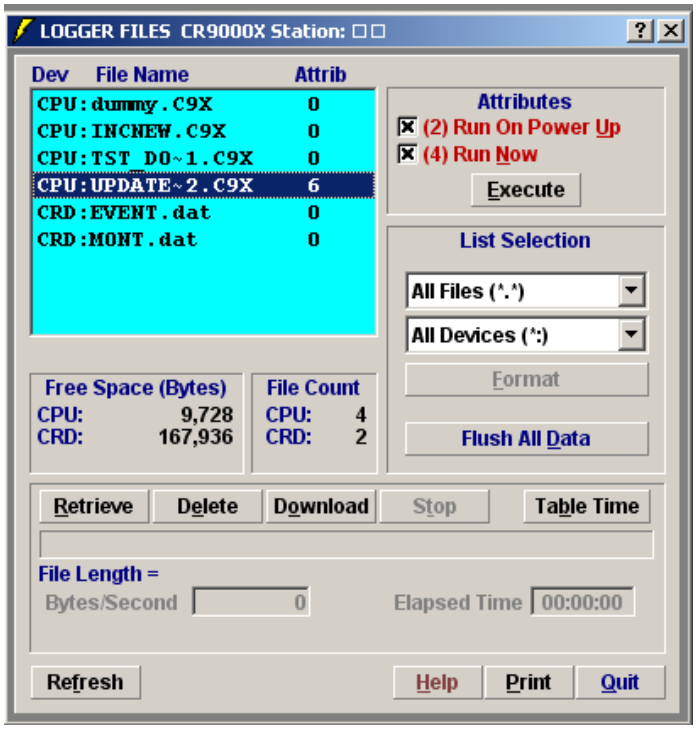


Figure 5.7 Logger files window

5.2 Data Processing

Although most of the gages have been repaired, several gages still stop working periodically. These gages sporadically record “NAN” rather than an actual number, which makes processing the data much more difficult and extremely tedious. Because MATLAB uses matrices to store data, the data is taken in line by line and stored into a matrix. The issue that develops from this is that MATLAB can store either numbers or characters in a matrix, but not both in the same matrix. Steps were taken to make the MATLAB programs more durable, but there was only so much of the process that could be automated. The steps described below will process the files correctly, provided there is not an unexpected “NAN” in the data file.

5.2.1 Event Data

5.2.1.1 Event Sorting Program

Once the raw event data is downloaded into the scratch folder, the user will then want to place them in the appropriate folders for further processing. To do this the user will use the "Event Sorting Program" described in detail in section 4.2.1.1 and shown in its entirety in Appendix A.2. A summary of how to use this program can be found in the following list:

1. Open MATLAB software.
2. Select the 'File' tab and scroll down to 'Open.' Select and open the m-file for the program named "Event Sorting Program."
3. Input the name (it will be the date of the file downloads) of the events to be sorted in the 'counter' variable definition and each of the 3 'filename' variable definitions in the first 'for' loop. It is important not to change anything else in these lines besides the date of the event, in the form 'yyyy-mm-dd." (The other parts of the definitions are crucial to the program for either counting the files or the working its way through them one by one.)
4. Save and run the program or hit 'F5' as a shortcut.
5. The message 'Program has completed' will appear if no errors occurred.

6. Check the ‘Event Data’ Folder and make sure actual date that the events occurred correspond to the date of the folders they are in.

7. Check the size of the event data files sorted in these folders and make note of any files that are smaller than 170 KB. Check the time or record numbers of these marked files and verify they are indeed 401 records long. If they are not the correct length, delete them as they will cause problems in the next step of the analysis process.

5.2.1.2 Event Data Extraction Program

Once these events are properly sorted, the user can continue with further processing using the “Event Data Extraction Program” program described in section 4.2.2.2 and shown in Appendix A.1. The following list summarizes how to use this program and points out some important things to remember when troubleshooting:

1. Open MATLAB software.

2. Select the ‘File’ tab and scroll down to ‘Open.’ Select and open the m-file for the program named “Event Data Extraction Program.”

3. Input the name of the event file to be analyzed in the two ‘pathname’ variable definitions. There is one inside and one outside of the ‘for’ loop. These dates will be of the form “yyyy-mm-dd.”

4. Save and run the program or hit ‘F5’ as a shortcut.

5. The message ‘Program has completed’ will appear if no errors occurred.

6. If errors occur, the most likely area for a problem is in ‘sscanf’ line of the program, where data is read into a matrix line by line. If one or more of the gages is recording ‘NAN’ where the program expects a number to be or a gage is recording a number instead of an expected ‘NAN,’ the program will crash. Identify which event the error occurred from and look at this event in more detail to see what gages may be a problem. Match the data to be read in that event with the values in the ‘sscanf’ line and the program should run. Numbers should be denoted by ‘%f’ while ‘NAN’ should be read in with ‘%*5c’ in this line of code.

7. If step 6 is taken and another different error occurs, it is likely because the user forgot to assign the ‘dummy’ gage variable to one or more gages that are recording ‘NAN.’ This is done in the section of the program that takes the maximum, minimum, mean, and standard deviation for each gage during an event. Check that all gages that are recording ‘NAN’ are assigned a ‘dummy’ value and attempt to run the program again.

8. Once the program runs without errors, select ‘Smartdata.mat’ in the ‘Current Directory’ box on the upper left side of the MATLAB home window, shown below in Figure 5.8.

9. This will move the ‘strainmatrix’ and ‘accelmatrix’ variables into the ‘MAT File’ window, shown in the bottom left of Figure 5.8. Double click each of these variables individually.

10. Then the variables will be moved into the ‘Workspace’ box in the upper right hand corner of Figure 5.8. These matrices can then be selected and opened in the variable editor (shown below in the middle of Figure 5.8) to view the max., min., mean, and STD for each individual event together in one matrix.

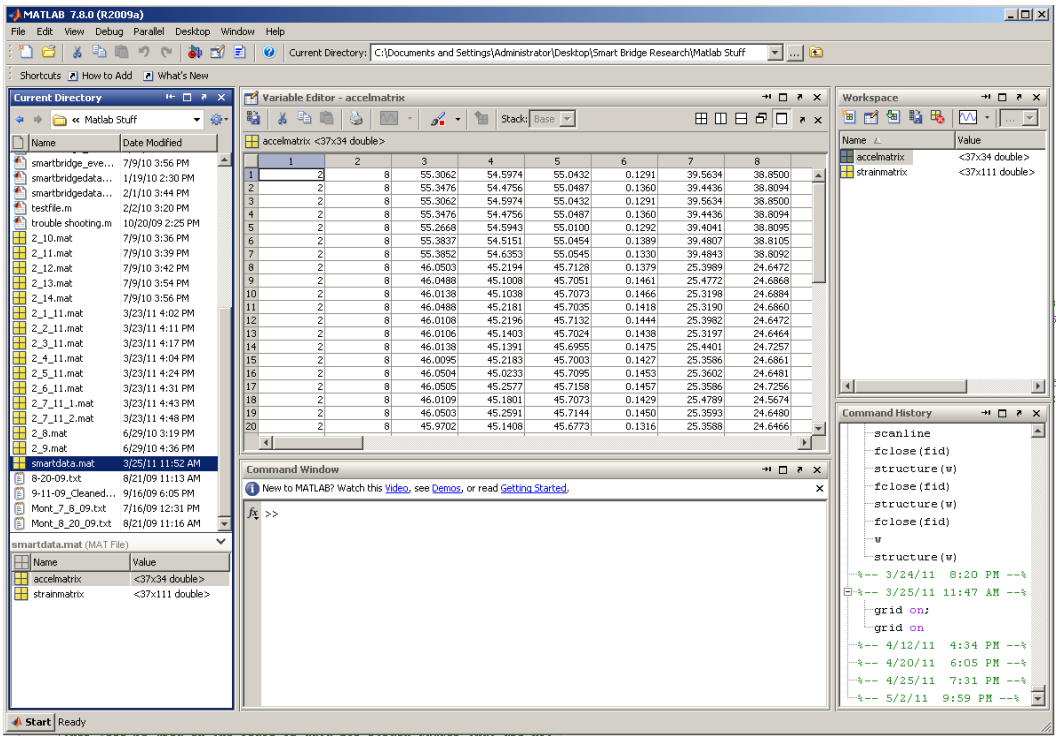


Figure 5.8 MATLAB home screen shot

5.2.1.3 Event Graphing Program

After using the data extraction program described above, the user may be interested in looking at the all data from an individual event, rather then just the calculated max., min., mean, and STD values found in the final strain matrix or final acceleration matrix. For this task the user should use the MATLAB program named "Event Graphing Program." This program is described in 4.2.1.3 and is also shown in Appendix A.3. This program should be used after the user has already run the data extraction program and identified events of interest. The following list contains a description of how to use this program:

1. Open MATLAB software.

2. Select the ‘File’ tab and scroll down to ‘Open.’ Select and open the m-file for the program named “Event Graphing Program.”

3. Input the name of the event file to be analyzed in the two ‘Pathname’ variable definitions. There is one inside and one outside of the ‘For’ loop. These dates will be of the form “yyyy-mm-dd.”

4. Input the number of the event of interest in the two ‘stopfile’ variable definitions. There is one inside and one outside of the ‘For’ loop. For example if there were 30 events on a given day and the user would like to look at the 20^{th} event, ‘stopfile’ would be set to 20. (‘Stopfile’ stops the program before it clears the data for the event of interest, thus saving it for further inspection.)

5. Save and run the program or hit ‘F5’ as a shortcut.

6. The message ‘Program has completed’ will appear if no errors occurred.

7. There should be no errors because this program uses similar code to the data extraction program, but if an error does occur, follow steps 6-7 above in section 5.2.1.2 to identify the problem.

8. Once the program runs without errors, the user will have all of the data for all of the gages for the event specified by the 'stopfile' variable.

9. The user can then create graphs of the event for any gage they would like. A list of the gage names and their corresponding column in the final strain matrix is included in this program for quick reference. (A graph produced with this program is shown in Figure 4.7.)

5.2.1.4 Weekly Event Breakdown

Since a large amount of event data is being recorded by the data logger every month, it made sense to break down the event data into week long increments, which could later be combined for the monthly reports. These weekly event graphs can be produced by following the instructions found in the following list:

1. Run the "Event Data Extraction Program" by following steps 1-8 in section 5.2.1.2 above.

2. When the 'smartdata.mat' variable is highlighted right click and select 'rename' or hit 'F2' as a shortcut.

3. Give the 'smartdata.mat' variable a new, unique name.

4. Process the events for every day in the week (or longer period of time) of interest in a similar fashion, each time giving the new 'smartdata.mat' variable produced a unique name.

5. Once each event of interest has been saved under different names, the user should move each of the events into the MATLAB workspace in the same way as described in steps 9-10 in section 5.2.1.2 above.

6. After all the events of interest are present in the workspace, the user can combine all these events into one matrix with one variable name. An example of a MATLAB command to complete this task is: week = [name_1; name_2; name_3; etc..]

7. To ensure the events are in the proper chronological order before graphing, the user should use the 'sortrow' command in MATLAB. Since the 'datenum' value is included as the first column of the final strain matrix and that value is always increasing as records move forward through time, this will always make sure the data is in order from earliest to most recent. An example of this command is: week=sortrows(week,1).

8. After all the events are combined in the proper order in one matrix, the user can then plot the results of time (again using the 'datenum' column) versus any gage value from the final strain matrix (max, mean, etc.). An example of a plot created using this method is shown in Figure 5.9 below. (Further manipulation of the x-axis was necessary to include the day name rather than a 'datenum' value)

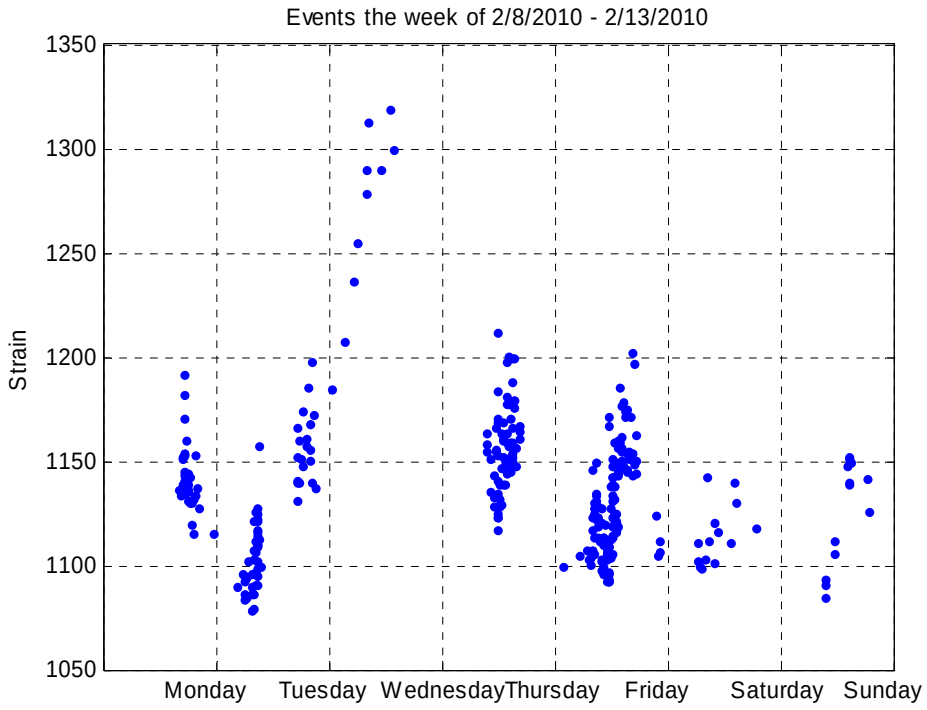


Figure 5.9 Maximum strains for trigger gage S35B for events the week of 2/8/2010 through 2/13/2010

5.2.2 Monitoring Data

5.2.2.1 Monitoring Data Processor Program

The monitoring data isn't sorted using a MATLAB program, because the format of the data makes it much more difficult to automate this process. To look at monitoring data, the user will use the MATLAB program called "Monitor Processor Program" discussed previously in section 4.2.2 and also found in Appendix A.4. The following list is a description of how to use this program:

1. Open MATLAB software.

2. Select the ‘File’ tab and scroll down to ‘Open.’ Select and open the m-file for the program named “Monitor Processor Program.”

3. Input the name of the monitoring data file to be processed in the ‘filename’ variable definition, in the form ‘mm-dd-yyyy.’

4. Input the number of records in the monitoring data file to be processed in the ‘total_record_num’ variable definition. This value can be found by opening the data file to be processed in another program such as Microsoft Excel or Notepad and looking at the last value in the record number column.

5. Input the name the processed data should be saved under in the ‘smartdata’ variable definition line of the program.

6. Save and run the program or hit ‘F5’ as a shortcut.

7. The message ‘Program has completed’ will appear if no errors occurred.

8. If an error occurs, it is likely due to a mismatched input in the ‘sscanf’ line of code. As described before, if one or more of the gages is either recording ‘NAN’ where the program expects a number to be or the gage is recording a number instead of an expected ‘NAN’ the program will crash. Identify which monitoring record contains the error and see what gages may be

a problem. Match the data to be read with this 'sscanf' line and the program should run. Again numbers should be denoted by '%f' while 'NAN' should be read in with '%*5c.'

9. As before with the event data, this program produces a 'smartdata' matrix which will be saved under the name the user specified in step 5 above.

10. Move this matrix to the MATLAB workspace by following the instruction in steps 8-10 in section 5.2.1.2 above.

11. The matrix for the event just processed can then be viewed in the variable editor in MATLAB.

5.2.2.2 Monthly Monitoring Data Breakdown

One of the long term goals of the project is to have data processing programs set up to conduct monthly reports detailing both monitoring data and event data from the past month. By using the monitoring processor program as described above for several consecutive data files, it is possible to combine these individual files into one matrix, representing one month. The following list describes how the user could create such a matrix:

1. Run the "Monitor Processor Program" by following steps 1-9 in section 5.2.2.1 above.

2. Once the user has all the monitoring files in the MATLAB workspace, the files can be combined into one matrix with one variable name. An example of MATLAB code to complete this task is: month = [name_1; name_2; name_3; etc..].

3. Again the user will want to sort the data into the proper chronological order, by using the 'sortrow' command in MATLAB. Building of the example above, the code to do this would look like: month = sortrow(month,1).

4. Because it is possible for monitoring files to overlap, if the user forgets to flush the data after a download for instance, this data must also be checked for duplicates. To do this the MATLAB command 'unique' is utilized. Unique checks each entry against all other entries in the matrix and erases duplicates if found. Again building off the previous example, this command would be used as follows: month = unique(month,'rows').

5. After the monitoring data is all in the correct order in one matrix, the user can then plot data from a gage for every monitoring record in the matrix, thus producing a graph of that gage over the month processed. An example of a graph resulting from this process can be seen in Figure 5.10 below.

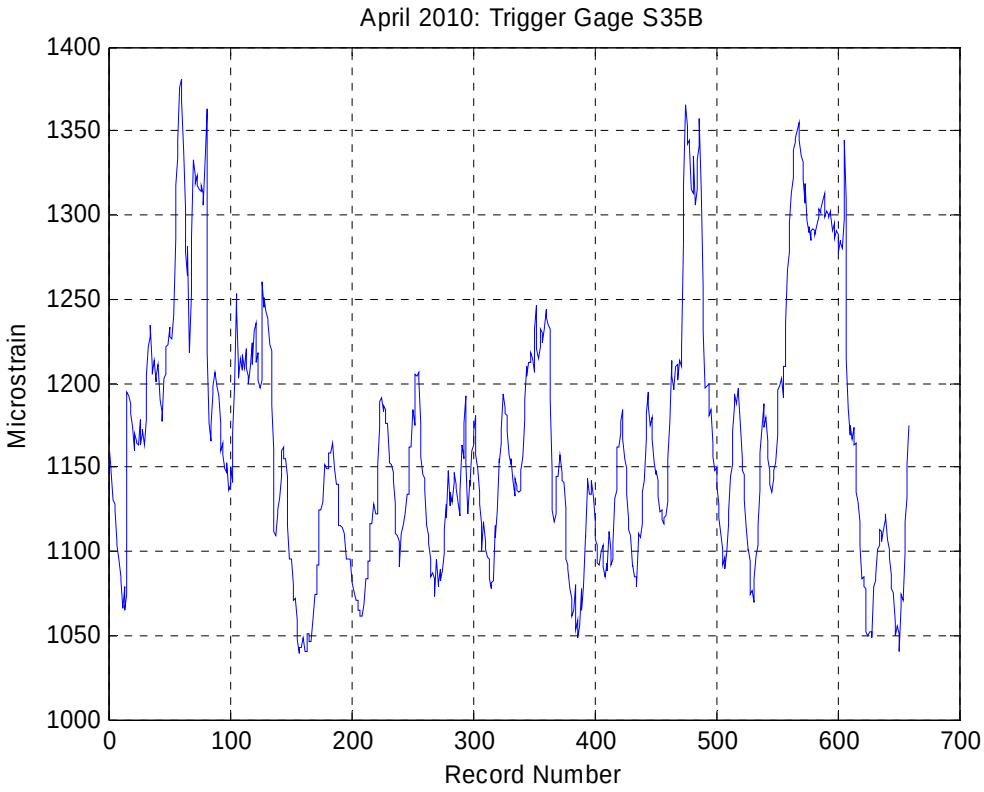


Figure 5.10 Monitoring data for trigger gage S35B for April 2010

Chapter 6

CURRENT SYSTEM STATUS AND FUTURE DEVELOPMENT

Of the thirty strain gages on the bridge, two gages are still periodically having issues. S39T and S38B are both wired correctly to their bridge completion modules, but will sometimes record "NAN" for a period of time and then come back online. There are also several gages that are currently recording data, but still have some wiring issues that were identified by measuring resistances. These gages can be seen in Table 3.4 and are defined as questionable. These issues make processing the event and monitoring data very tedious, as the MATLAB programs must have the correct description of the data they are inputting or they will crash. If the programs believe they are supposed to scan in 46 floating point numbers and even one of those gages malfunctions and records "NAN" for just one line of data out of 401 lines, the program will crash. For this reason one of the most important things to deal with moving forward is the reliability of the strain gages being used for the event data.

Similarly the temperature gages, which are only being used for the monitoring data, have been unreliable at best. Two of the temperature gages, TD35 and TCOU, are not functioning at all meaning they constantly record "NAN." The temperature gage T65, however, goes back and forth between periods of malfunctioning and recording correct temperature data. This gage periodically working and then not working makes processing the monitoring data difficult, for the same reasons mentioned above with the event data. Therefore these temperature gages should be repaired or replaced, or simply removed from the monitoring system if necessary. If the data was consistently of the same format, then processing programs

could be written to automate things and make data analysis much less time consuming for the user. It is critical to the future of this project that the sorting and processing of data be done with minimal human interaction. This will allow the user to dedicate more time to actually analyzing data and recognizing trends and comparisons, rather then simply organizing data and creating graphs.

Another aspect of the project which must be improved moving forward is developing an operational design set up for the string pots. Once a stationary datum can reliably be established, the displacement and rotation of girders can be coupled with event data to get a more comprehensive view of bridge behavior. It will be extremely helpful in identifying future problems with the bridge if there are accurate records of what displacements and rotations are associated with events of a certain magnitude.

As was discussed in Chapter 5, both PC9000 and RTDAQ are currently being used to download data from the logger. PC9000 uses a DSL modem and ethernet connection and was originally the only program used for this task. Slow transfer rates and an expensive DSL internet fees were motivation to look for an alternate program to use for downloading and monitoring data. RTDAQ was selected because it is capable of faster transfer rates and is able to use a dynamic IP address, associated with a cellular modem connection. A Raven Airlink Cellular Modem was installed at the site and an internet account was set up for about half the price of the previous DSL connection. RTDAQ downloads the monitoring data without issue, but the file markers separating the event data into individual data files has proved to be a problem thus far in the downloading process. This is why PC9000 and RTDAQ are both included in section 5.1. There is a way to process file markers in RTDAQ, by

downloading the event data in binary format and using the 'card converter' tool to convert to ASCII format and process the file markers into individual even data files. This process has not worked, however, and therefore more time must be spent refining the download process in RTDAQ so that the event data can be downloaded in the correct format. This will allow RTDAQ to handle all downloads from the logger and thus will eliminate the need for the PC9000 software and the more expensive DSL connection.

While the MATLAB programs that have been developed to this point serve their purposes well, more work must be done to automatically process event data into weekly groups and monitoring data into monthly data files. These can then be combined into monthly reports that will show monitoring trends through the month, as well as highlighting significant events. This is possible with the current programs, but it would require an inordinate amount of time for each report.

BIBLIOGRAPHY

Burrell, G. (2004). *Delaware's Smart Bridge Diagnostic Test.* Newark, DE: University of Delaware.

Federal Highway Administration (FHWA). (n.d.). *LTBP: Long-Term Bridge Performance Program*. Retrieved April 20, 2010, from http://www.fhwa.dot.gov/research/tfhrc/programs/infrastructure/structures/ltbp/

Land, R., Murugesh, G., & Raghavendrachar, M. (2003). "Assessing Bridge Performance - When, What, and How". *Technical Memorandum of Public Works Research Institute* , 213-228.

Lynch, M. (2003). *Delaware's First "Smart" Bridge.* Wilmington, DE: University of Delaware.

Maalej, M., Karasaridis, A., Pantazopoulou, S., & Hatzinakos, D. (2002, July 23). Structural Health Monitoring of Smart Structures. *Smart Materials and Strucutres* .

Natale, R. (2008). *Further Development of Delaware's First Permanantly Instrumented Bridge.* Newark, DE: University of Delaware.

Phares, B. M., Wipf, T. J., Greimann, L. F., & Lee, Y.-S. (2005). *Health Monitoring of Bridge Structures and Components Using Smart Structure Technology Volume 1.* Center for Transportation Research and Education.Iowa State University.

Rakowski, M. D. (2008). *Bridge Evaluation Using In-Service and Weigh-In-Motion Data.* Newark,DE: University of Delaware.

Reader, N. (2007). *Delaware's First Long Term Instrumented Bridge: A Prototypical Instrumentation & Installation Plan.* Newark, DE: University of Delaware.

Wehrum, K. (2006). *Dynamic Analysis of Delaware's Smart Bridge.* Newark, DE: University of Delaware.

APPENDIX A: MATLAB AND PC9000 PROGRAMS

A.1 MATLAB Event Data Extraction Program

```
%%Event Data Extraction Program
%This program takes in raw event data and places data into a matrix.
%The matrix is manipulated to return the max, min, avg, and std
deviation
%for each strain gage and accelerometer.
%The program outputs the calculated parameters in two matrices.
%The program skips over gages that are specified with %*5c in the
scanline
%portion of the program. This should be done for all
%gages that return a value of "NaN".

clear all;
%Setting the pathname for the folder or directory to be processed
%Be sure to include a '\' after the final year, month, day you would
like
%to specify
pathname='C:\Documents and Settings\Administrator\Desktop\Smart
Bridge Research\Event Data\2011\2\8\';

%Counter gives the number of files in the folder of pathname
counter=length(dir(sprintf('%s',pathname)));
%subtracting two empty files from count, leaving actual number of
files
numberoffiles=counter-2;
%Creating a structure to organize the files for analysis
structure=dir(sprintf('%s',pathname));

%define size of strain and acceleration matrix to save data
strainmatrix_8=zeros(numberoffiles,111);
accelmatrix=zeros(numberoffiles,34);


%saving strain and acceleration matrices created
save smartdata strainmatrix_8 accelmatrix

for w=3:counter

    %Saving the proper index for each event's placement in the matrix
    matindex=w-2;
    %Resetting the pathname and structure inside of the loop,
    %needed because 'clear all' at the end resets all these variables
    pathname='C:\Documents and Settings\Administrator\Desktop\Smart
Bridge Research\Event Data\2011\2\8\';
    structure=dir(sprintf('%s',pathname));
    %Setting 'f' to each individual file name
    %Changes each time through the loop as 'w' changes
    f=structure(w).name;

 %setting the filename and opening the file for processing
filename=sprintf('%s%s',pathname,f);
```

```
fid=fopen(filename);

 %skip over top 4 rows of header information
    for j=1:4
        extraline=fgetl(fid);
    end

    %scan in data collected by gages

    for i=1:401
        scanline=fgetl(fid);
        if scanline < 0, break, end;
        %k represents the length of the date and time information
        %This includes the ',' so the next character is our first
column of
        %data
        for k=1:length(scanline)
            if scanline(k)==','
                break
            end
        end

        %These logic checks ensure that the data and time information
is
        %saved properly
        %Because the number of digits in the second columns change
if k==25
     year=str2num(scanline(2:5));
     month=str2num(scanline(7:8));
     day=str2num(scanline(10:11));
     hour=str2num(scanline(13:14));
     minute=str2num(scanline(16:17));
     second=str2num(scanline(19:20))+str2num(scanline(22:23))/100;

 else if k==24
     year=str2num(scanline(2:5));
     month=str2num(scanline(7:8));
     day=str2num(scanline(10:11));
     hour=str2num(scanline(13:14));
     minute=str2num(scanline(16:17));
     second=str2num(scanline(19:20))+str2num(scanline(22))/100;

 else if k==23
     year=str2num(scanline(2:5));
     month=str2num(scanline(7:8));
     day=str2num(scanline(10:11));
     hour=str2num(scanline(13:14));
     minute=str2num(scanline(16:17));
     second=str2num(scanline(19:20));

 else if k==22
```

```
      year=str2num(scanline(2:5));
      month=str2num(scanline(7:8));
      day=str2num(scanline(10:11));
      hour=str2num(scanline(13:14));
      minute=str2num(scanline(16:17));
      second=0;

      end
      end
      end
end

%Creating a single number to represent the date and time information
       date(i)=datenum(year,month,day,hour,minute,second);

%Reading the data in line by line using scanf. The scan starts the
%character after the date data is finished
%'%f's need to be changed to '%*5c' if a gauge stops reading
correctly and
%is only recording '"NAN"'
% % The numbers list above/below the scanning lines corresponds to
the
% gages so they can be properly lined up when incorrectly working

% % % %  Use this code to import data when a gage is not working
properly
% % %                                              1  2  3  4  5  6  7  8
9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31
32  33 34 35 36 37 38 39 40 41 42 43 44 45 46
a(:,i)=sscanf(scanline(k+1:length(scanline)),'%f,%f,%f,%f,%f,%f,%f,%f
,%f,%f,%f,%f,%f,%f,%f,%f,%f,%f,%f,%f,%f,%f,%f,%f,%f,%f,%f,%f,%f,%f,%f
,%f,%*5c,%f,%f,%f,%f,%f,%f,%f,%f,%f,%f,%f,%f',[45,inf]);

% % % Use this to import data when all gages are working
% %
a(:,i)=sscanf(scanline(k+1:length(scanline)),'%f,%f,%f,%f,%f,%f,%f,%f
,%f,%f,%f,%f,%f,%f,%f,%f,%f,%f,%f,%f,%f,%f,%f,%f,%f,%f,%f,%f,%f,%f,%f
,%f,%f,%f,%f,%f,%f,%f,%f,%f,%f,%f,%f,%f,%f',[46,inf]);
% % %    %                                              1  2  3  4  5  6
7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29
30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46
    end

%Transposing the matrix so the data is organized by column rather
then by
%row
a=a';

%%This was an attempt to scan the data matrix and fill in -1 for
every
%%"NAN" in the data. This logic was not functional,however.
```

```
% % %
% % % for col_count=1:length(a)
% % %     for row_count=1:45
% % %         if(a(row_count,col_count))=='"NAN"'
% % %             a(row_count,col_count)==-1
% % %
% % %         end
% % %     end
% % % end
load smartdata

    %Creating a dummy vector of zeros
    %This will be used in the place of data for strain gauges that
are not
    %working properly
    %Use the dummy vector for any gauges that have '%*5c' in the
sscanf
    %call above

    dummy=[ 0 0 0 0 ];

     %find max, min, average, and standard deviation for string pots,
excluding D65 (returns 'NaN')
    %spring pot setup was not operational, thus data was not valid
    %code can be uncommented when string pots are returning vaild
data
    %displacement matrix must be created at begining of program to
save all data
    %d15=[max(a(:,2)) min(a(:,2)) mean(a(:,2)) std(a(:,2),1)];
    %d31=[max(a(:,3)) min(a(:,3)) mean(a(:,3)) std(a(:,3),1)];
    %d33=[max(a(:,4)) min(a(:,4)) mean(a(:,4)) std(a(:,4),1)];
    %d35=[max(a(:,5)) min(a(:,5)) mean(a(:,5)) std(a(:,5),1)];
    %d37=[max(a(:,6)) min(a(:,6)) mean(a(:,6)) std(a(:,6),1)];
    %d39=[max(a(:,7)) min(a(:,7)) mean(a(:,7)) std(a(:,7),1)];
    %d55=[max(a(:,8)) min(a(:,8)) mean(a(:,8)) std(a(:,8),1)];
    %d65=[max(a(:,9)) min(a(:,9)) mean(a(:,9)) std(a(:,9),1)];
    %d75=[max(a(:,10)) min(a(:,10)) mean(a(:,10)) std(a(:,10),1)];

    %find max, min, average, and standard deviation for
acccelerometers
    a25=[max(a(:,11)) min(a(:,11)) mean(a(:,11)) std(a(:,11),1)];
    a31=[max(a(:,12)) min(a(:,12)) mean(a(:,12)) std(a(:,12),1)];
    a35x=[max(a(:,13)) min(a(:,13)) mean(a(:,13)) std(a(:,13),1)];
    a35y=[max(a(:,14)) min(a(:,14)) mean(a(:,14)) std(a(:,14),1)];
    a39=[max(a(:,15)) min(a(:,15)) mean(a(:,15)) std(a(:,15),1)];
    a45=[max(a(:,16)) min(a(:,16)) mean(a(:,16)) std(a(:,16),1)];
    a65x=[max(a(:,17)) min(a(:,17)) mean(a(:,17)) std(a(:,17),1)];
    a65y=[max(a(:,18)) min(a(:,18)) mean(a(:,18)) std(a(:,18),1)];

% % %     Use this list of assignments to define values when all
gages are working
```

```
% % %     %find max, min, average, and standard deviation for foil
gages
% % %     %Note: s39T, s79T, and s79b are not included in this
program because they were not fucntioning
% % %     s15t=[max(a(:,19)) min(a(:,19)) mean(a(:,19))
std(a(:,19),1)];
% % %     s15b=[max(a(:,20)) min(a(:,20)) mean(a(:,20))
std(a(:,20),1)];
% % %     s19t=[max(a(:,21)) min(a(:,21)) mean(a(:,21))
std(a(:,21),1)];
% % %     s19b=[max(a(:,22)) min(a(:,22)) mean(a(:,22))
std(a(:,22),1)];
% % %     s31b=[max(a(:,23)) min(a(:,23)) mean(a(:,23))
std(a(:,23),1)];
% % %     s32b=[max(a(:,24)) min(a(:,24)) mean(a(:,24))
std(a(:,24),1)];
% % %     s33b=[max(a(:,25)) min(a(:,25)) mean(a(:,25))
std(a(:,25),1)];
% % %     s34b=[max(a(:,26)) min(a(:,26)) mean(a(:,26))
std(a(:,26),1)];
% % %     s35t=[max(a(:,27)) min(a(:,27)) mean(a(:,27))
std(a(:,27),1)];
% % %     s35b=[max(a(:,28)) min(a(:,28)) mean(a(:,28))
std(a(:,28),1)];
% % %     s36b=[max(a(:,29)) min(a(:,29)) mean(a(:,29))
std(a(:,29),1)];
% % %     s37b=[max(a(:,30)) min(a(:,30)) mean(a(:,30))
std(a(:,30),1)];
% % %     s38b=[max(a(:,31)) min(a(:,31)) mean(a(:,31))
std(a(:,31),1)];
% % %     s39t=[max(a(:,32)) min(a(:,32)) mean(a(:,32))
std(a(:,32),1)];
% % %     s39b=[max(a(:,33)) min(a(:,33)) mean(a(:,33))
std(a(:,33),1)];
% % %     s55t=[max(a(:,34)) min(a(:,34)) mean(a(:,34))
std(a(:,34),1)];
% % %     s55b=[max(a(:,35)) min(a(:,35)) mean(a(:,35))
std(a(:,35),1)];
% % %     s59t=[max(a(:,36)) min(a(:,36)) mean(a(:,36))
std(a(:,36),1)];
% % %     s59b=[max(a(:,37)) min(a(:,37)) mean(a(:,37))
std(a(:,37),1)];
% % %     s65t=[max(a(:,38)) min(a(:,38)) mean(a(:,38))
std(a(:,38),1)];
% % %     s65b=[max(a(:,39)) min(a(:,39)) mean(a(:,39))
std(a(:,39),1)];
% % %     s69t=[max(a(:,40)) min(a(:,40)) mean(a(:,40))
std(a(:,40),1)];
% % %     s69b=[max(a(:,41)) min(a(:,41)) mean(a(:,41))
std(a(:,41),1)];
```

```
% % %     s75t=[max(a(:,42)) min(a(:,42)) mean(a(:,42))
std(a(:,42),1)];
% % %     s75b=[max(a(:,43)) min(a(:,43)) mean(a(:,43))
std(a(:,43),1)];
% % %     s79t=[max(a(:,44)) min(a(:,44)) mean(a(:,44))
std(a(:,44),1)];
% % %     s79b=[max(a(:,45)) min(a(:,45)) mean(a(:,45))
std(a(:,45),1)];

% % %     s15t=[max(a(:,19)) min(a(:,19)) mean(a(:,19))
std(a(:,19),1)];
% % %     s15b=[max(a(:,20)) min(a(:,20)) mean(a(:,20))
std(a(:,20),1)];
% % %     s19t=[max(a(:,21)) min(a(:,21)) mean(a(:,21))
std(a(:,21),1)];
% % %     s19b=[max(a(:,22)) min(a(:,22)) mean(a(:,22))
std(a(:,22),1)];
% % %     s31b=[max(a(:,23)) min(a(:,23)) mean(a(:,23))
std(a(:,23),1)];
% % %     s32b=[max(a(:,24)) min(a(:,24)) mean(a(:,24))
std(a(:,24),1)];
% % %     s33b=[max(a(:,25)) min(a(:,25)) mean(a(:,25))
std(a(:,25),1)];
% % %     s34b=[max(a(:,26)) min(a(:,26)) mean(a(:,26))
std(a(:,26),1)];
% % %     s35t=[max(a(:,27)) min(a(:,27)) mean(a(:,27))
std(a(:,27),1)];
% % %     s35b=[max(a(:,28)) min(a(:,28)) mean(a(:,28))
std(a(:,28),1)];
% % %     s36b=[max(a(:,29)) min(a(:,29)) mean(a(:,29))
std(a(:,29),1)];
% % %     s37b=[max(a(:,30)) min(a(:,30)) mean(a(:,30))
std(a(:,30),1)];
% % %     s38b=dummy;
% % %     s39t=dummy;
% % %     s39b=[max(a(:,31)) min(a(:,31)) mean(a(:,31))
std(a(:,31),1)];
% % %     s55t=[max(a(:,32)) min(a(:,32)) mean(a(:,32))
std(a(:,32),1)];
% % %     s55b=[max(a(:,33)) min(a(:,33)) mean(a(:,33))
std(a(:,33),1)];
% % %     s59t=[max(a(:,34)) min(a(:,34)) mean(a(:,34))
std(a(:,34),1)];
% % %     s59b=[max(a(:,35)) min(a(:,35)) mean(a(:,35))
std(a(:,35),1)];
% % %     s65t=[max(a(:,36)) min(a(:,36)) mean(a(:,36))
std(a(:,36),1)];
% % %     s65b=[max(a(:,37)) min(a(:,37)) mean(a(:,37))
std(a(:,37),1)];
% % %     s69t=[max(a(:,38)) min(a(:,38)) mean(a(:,38))
std(a(:,38),1)];
```

```
% % %     s69b=[max(a(:,39)) min(a(:,39)) mean(a(:,39))
std(a(:,39),1)];
% % %     s75t=[max(a(:,40)) min(a(:,40)) mean(a(:,40))
std(a(:,40),1)];
% % %     s75b=[max(a(:,41)) min(a(:,41)) mean(a(:,41))
std(a(:,41),1)];
% % %     s79t=[max(a(:,42)) min(a(:,42)) mean(a(:,42))
std(a(:,42),1)];
% % %     s79b=[max(a(:,43)) min(a(:,43)) mean(a(:,43))
std(a(:,43),1)];


% % % % %  Use this list of assignments when one of the middle gages,
% % % % %  most commonly S38B and S39T go in and out of functioning

    s15t=[max(a(:,19)) min(a(:,19)) mean(a(:,19)) std(a(:,19),1)];
    s15b=[max(a(:,20)) min(a(:,20)) mean(a(:,20)) std(a(:,20),1)];
    s19t=[max(a(:,21)) min(a(:,21)) mean(a(:,21)) std(a(:,21),1)];
    s19b=[max(a(:,22)) min(a(:,22)) mean(a(:,22)) std(a(:,22),1)];
    s31b=[max(a(:,23)) min(a(:,23)) mean(a(:,23)) std(a(:,23),1)];
    s32b=[max(a(:,24)) min(a(:,24)) mean(a(:,24)) std(a(:,24),1)];
    s33b=[max(a(:,25)) min(a(:,25)) mean(a(:,25)) std(a(:,25),1)];
    s34b=[max(a(:,26)) min(a(:,26)) mean(a(:,26)) std(a(:,26),1)];
    s35t=[max(a(:,27)) min(a(:,27)) mean(a(:,27)) std(a(:,27),1)];
    s35b=[max(a(:,28)) min(a(:,28)) mean(a(:,28)) std(a(:,28),1)];
    s36b=[max(a(:,29)) min(a(:,29)) mean(a(:,29)) std(a(:,29),1)];
    s37b=[max(a(:,30)) min(a(:,30)) mean(a(:,30)) std(a(:,30),1)];
    s38b=[max(a(:,31)) min(a(:,31)) mean(a(:,31)) std(a(:,31),1)];
    s39t=dummy;
    s39b=[max(a(:,32)) min(a(:,32)) mean(a(:,32)) std(a(:,32),1)];
    s55t=[max(a(:,33)) min(a(:,33)) mean(a(:,33)) std(a(:,33),1)];
    s55b=[max(a(:,34)) min(a(:,34)) mean(a(:,34)) std(a(:,34),1)];
    s59t=[max(a(:,35)) min(a(:,35)) mean(a(:,35)) std(a(:,35),1)];
    s59b=[max(a(:,36)) min(a(:,36)) mean(a(:,36)) std(a(:,36),1)];
    s65t=[max(a(:,37)) min(a(:,37)) mean(a(:,37)) std(a(:,37),1)];
    s65b=[max(a(:,38)) min(a(:,38)) mean(a(:,38)) std(a(:,38),1)];
    s69t=[max(a(:,39)) min(a(:,39)) mean(a(:,39)) std(a(:,39),1)];
    s69b=[max(a(:,40)) min(a(:,40)) mean(a(:,40)) std(a(:,40),1)];
    s75t=[max(a(:,41)) min(a(:,41)) mean(a(:,41)) std(a(:,41),1)];
    s75b=[max(a(:,42)) min(a(:,42)) mean(a(:,42)) std(a(:,42),1)];
    s79t=[max(a(:,43)) min(a(:,43)) mean(a(:,43)) std(a(:,43),1)];
    s79b=[max(a(:,44)) min(a(:,44)) mean(a(:,44)) std(a(:,44),1)];

    %Organizing the max/min/avg/std calculations for each gauge into
two
    %matrices. One for strain gauges and one for acceleration gauges
    finalrowstrain=[date(i) month day s35b(1) s15t s15b s19t s19b
s31b s32b s33b s34b s35t s35b s36b s37b s38b s39t s39b s55t s55b s59t
s59b s65t s65b s69t s69b s75t s75b s79t s79b];
    finalrowaccel=[month day a25 a31 a35x a35y a39 a45 a65x a65y];
```

```
    strainmatrix_8(matindex,:)=finalrowstrain;
    accelmatrix(matindex,:)=finalrowaccel;

    %saving and closing the file Identifier.
    %also clearing variables for the next pass
    save smartdata strainmatrix_8 accelmatrix
    fclose(fid);

    clear all;

end

%Reports that the program has finished processing
fprintf('Program has completed');
```

A.2 MATLAB Event Data Sorting Program

```
%%Event Sorting Program

%This program takes in a filename from the user and sorts each event
file
%with that name into a folder corresponding to the year, month, and
day
%that each event occurred.

% Creating a counter that represents the number of files to be
processed by the program

counter=dir('C:\Documents and Settings\Administrator\Desktop\Smart
Bridge Research\Scratch Folder\2-24-2011.*');

%Opening the file for read access
% If 'fid' errors occur, it may be necessary to change the file
numbers
% below manually. It will crash if files have been deleted and there
isn't
% a file for w+1 to read

%    for w=70:70+(length(counter)-1)
for w=0:(length(counter)-1)

    clear fid;
    if w<10
        filename=['C:\Documents and
Settings\Administrator\Desktop\Smart Bridge Research\Scratch
Folder\2-24-2011.00' int2str(w)];
        fid=fopen(filename);

    elseif (w>=10) && (w<100)
```

```
        filename=['C:\Documents and
Settings\Administrator\Desktop\Smart Bridge Research\Scratch
Folder\2-24-2011.0' int2str(w)];
        fid=fopen(filename);

    elseif w>=100
        filename=['C:\Documents and
Settings\Administrator\Desktop\Smart Bridge Research\Scratch
Folder\2-24-2011.' int2str(w)];
        fid=fopen(filename);

    end

filelocation = 'C:\Documents and Settings\Administrator\Desktop\Smart
Bridge Research\Event Data\';
%Skippin first four lines of header information
    for j=1:4
        extraline=fgetl(fid);
    end

 %Parsing out the time and date information individually for each
line
 %The k=25,24,23 statements are to properly account for date/times of
different lengths
 %These will be used later for sorting events in their correct
folders
 scanline=fgetl(fid);
  if scanline < 0, break, end;
  for k=1:length(scanline)
      if scanline(k)==','
           break
      end
  end

 if k==25
     year=str2num(scanline(2:5));
     month=str2num(scanline(7:8));
     day=str2num(scanline(10:11));
     hour=str2num(scanline(13:14));
     minute=str2num(scanline(16:17));
     second=str2num(scanline(19:20))+str2num(scanline(22:23))/100;

 else if k==24
     year=str2num(scanline(2:5));
     month=str2num(scanline(7:8));
     day=str2num(scanline(10:11));
     hour=str2num(scanline(13:14));
     minute=str2num(scanline(16:17));
     second=str2num(scanline(19:20))+str2num(scanline(22))/100;

 else if k==23
```

```
     year=str2num(scanline(2:5));
     month=str2num(scanline(7:8));
     day=str2num(scanline(10:11));
     hour=str2num(scanline(13:14));
     minute=str2num(scanline(16:17));
     second=str2num(scanline(19:20));

     end
     end
 end

%Checking if a folder exists for the year of the event being sorted
%If the folder does not exist, it will create one
if isdir(sprintf('%s%i',filelocation,year))==0

 mkdir(filelocation, num2str(year));

end

 %Checking if a folder exists for the month of the event being sorted
%If the folder does not exist, it will create one
if isdir(sprintf('%s%i\\%i',filelocation,year,month))==0

    mkdir(sprintf('%s%i\\',filelocation,year), num2str(month));

    end

%Checking if a folder exists for the day of the event being sorted
%If the folder does not exist, it will create one
if isdir(sprintf('%s%i\\%i\\%i',filelocation,year,month,day))==0

    mkdir(sprintf('%s%i\\%i',filelocation,year,month), num2str(day));

end

 %Setting 'eventfolder' as the proper pathname for the event being
sorted
eventfolder=(sprintf('%s%i\\%i\\%i',filelocation,year,month,day));
fid;

%Closing the fid assciated with the open file
[m]=fclose('all');


%Moving the event data file from the scratch folder to proper date
folder

%It is also possible to extend this 'movefile' command and also
rename the
```

```
%file as it is moved to its new folder
[a,b,c]=movefile(filename,eventfolder);

end

%Reports that the program has finished processing
 fprintf('Program has completed');
```

A.3 Event Graphing Program

```
%%Event Graphing Program

%This program evaluates event data from a folder the user inputs as
the
%pathname below. This program should be used after the "Event Data
%Extraction Program" and an event of interest has been identified.
The
%number of the event of interest should also be input by the user as
the
%'Stopfile' variable below.

clear all;

%Setting the pathname for the folder or directory to be processed
pathname='C:\Documents and Settings\Administrator\Desktop\Smart
Bridge Research\Event Data\2010\2\8\';

%Counter gives the number of files in the folder of pathname
counter=length(dir(sprintf('%s',pathname)));
%subtracting two empty files from count, leaving actual number of
files
numberoffiles=counter-2;
%Creating a structure to organize the files for analysis
structure=dir(sprintf('%s',pathname));

%define size of strain and acceleration matrix to save data
strainmatrix=zeros(numberoffiles,111);
accelmatrix=zeros(numberoffiles,34);

%saving strain and acceleration matrices created
save smartdata strainmatrix accelmatrix

% Setting a variable for the event file that you would like to use to
graph
stopfile=35;
% You need to add '2' to get the correct file
stopfile=stopfile+2;

%Note:You must also change the 'stopfile' value inside of the for
loop below
```

```
for w=3:stopfile

    stopfile=35;
    stopfile=stopfile+2;
    if w~=stopfile
    clear a;
    end
    %Saving the proper index for each event's placement in the matrix
    matindex=w-2;
    %Resetting the pathname and structure inside of the loop,
    %needed because 'clear all' at the end resets all these variables
    pathname='C:\Documents and Settings\Administrator\Desktop\Smart
Bridge Research\Event Data\2010\2\8\';
    structure=dir(sprintf('%s',pathname));
    %Setting 'f' to each individual file name
    %Changes each time through the loop as 'w' changes
    f=structure(w).name;

 %setting the filename and opening the file for processing
filename=sprintf('%s%s',pathname,f);
fid=fopen(filename);

 %skip over top 4 rows of header information
    for j=1:4
        extraline=fgetl(fid);
    end

    %scan in data collected by gages
    % %*5c skips over the "NAN" columns: S65T

    for i=1:401
        scanline=fgetl(fid);
        if scanline < 0, break, end;
        %k represents the length of the date and time information
        %This includes the ',' so the next character is our first
column of
        %data
        for k=1:length(scanline)
            if scanline(k)==','
                break
            end
        end

        %These logic checks ensure that the data and time information
is
        %saved properly
        %Because the number of digits in the second columns change
if k==25
     year=str2num(scanline(2:5));
     month=str2num(scanline(7:8));
```

```
        day=str2num(scanline(10:11));
        hour=str2num(scanline(13:14));
        minute=str2num(scanline(16:17));
        second=str2num(scanline(19:20))+str2num(scanline(22:23))/100;

    else if k==24
        year=str2num(scanline(2:5));
        month=str2num(scanline(7:8));
        day=str2num(scanline(10:11));
        hour=str2num(scanline(13:14));
        minute=str2num(scanline(16:17));
        second=str2num(scanline(19:20))+str2num(scanline(22))/100;

    else if k==23
        year=str2num(scanline(2:5));
        month=str2num(scanline(7:8));
        day=str2num(scanline(10:11));
        hour=str2num(scanline(13:14));
        minute=str2num(scanline(16:17));
        second=str2num(scanline(19:20));

        end
        end
end

    %Creating a single number to represent the date and time information
          date(i)=datenum(year,month,day,hour,minute,second);
          time(i)=second;

%Reading the data in line by line.
%'%f's need to be changed to '%*5c' if a gauge stops reading
correctly and
%is only recording '"NAN"'
%                                                 1  2  3  4  5  6  7  8
9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31
32 33 34 35 36 37   38 39 40 41 42 43 44 45 46
a(:,i)=sscanf(scanline(k+1:length(scanline)),'%f,%f,%f,%f,%f,%f,%f,%f
,%f,%f,%f,%f,%f,%f,%f,%f,%f,%f,%f,%f,%f,%f,%f,%f,%f,%f,%f,%f,%f,%f,%f
,%f,%f,%f,%f,%f,%f,%*5c,%f,%f,%f,%f,%f,%f,%f',[44,inf]);

    end

%Transposing the matrix so the data is organized by column rather
then by
%row
a=a';
load smartdata

    %Creating a dummy vector of zeros
    %This will be used in the place of data for strain gauges that
are not
```

```
    %working properly
    %Use the dummy vector for any gauges that have '%*5c' in the
sscanf
    %call above

    dummy=[ 0 0 0 0 ];

     %find max, min, average, and standard deviation for string pots,
excluding D65 (returns 'NaN')
    %spring pot setup was not operational, thus data was not valid
    %code can be uncommented when string pots are returning vaild
data
    %displacement matrix must be created at begining of program to
save all data
    %d15=[max(a(:,2)) min(a(:,2)) mean(a(:,2)) std(a(:,2),1)];
    %d31=[max(a(:,3)) min(a(:,3)) mean(a(:,3)) std(a(:,3),1)];
    %d33=[max(a(:,4)) min(a(:,4)) mean(a(:,4)) std(a(:,4),1)];
    %d35=[max(a(:,5)) min(a(:,5)) mean(a(:,5)) std(a(:,5),1)];
    %d37=[max(a(:,6)) min(a(:,6)) mean(a(:,6)) std(a(:,6),1)];
    %d39=[max(a(:,7)) min(a(:,7)) mean(a(:,7)) std(a(:,7),1)];
    %d55=[max(a(:,8)) min(a(:,8)) mean(a(:,8)) std(a(:,8),1)];
    %d65=[max(a(:,9)) min(a(:,9)) mean(a(:,9)) std(a(:,9),1)];
    %d75=[max(a(:,10)) min(a(:,10)) mean(a(:,10)) std(a(:,10),1)];

    %find max, min, average, and standard deviation for
acccelerometers
    a25=[max(a(:,11)) min(a(:,11)) mean(a(:,11)) std(a(:,11),1)];
    a31=[max(a(:,12)) min(a(:,12)) mean(a(:,12)) std(a(:,12),1)];
    a35x=[max(a(:,13)) min(a(:,13)) mean(a(:,13)) std(a(:,13),1)];
    a35y=[max(a(:,14)) min(a(:,14)) mean(a(:,14)) std(a(:,14),1)];
    a39=[max(a(:,15)) min(a(:,15)) mean(a(:,15)) std(a(:,15),1)];
    a45=[max(a(:,16)) min(a(:,16)) mean(a(:,16)) std(a(:,16),1)];
    a65x=[max(a(:,17)) min(a(:,17)) mean(a(:,17)) std(a(:,17),1)];
    a65y=[max(a(:,18)) min(a(:,18)) mean(a(:,18)) std(a(:,18),1)];

    %find max, min, average, and standard deviation for foil gages
    %Note: s39T, s79T, and s79b are not included in this program
because they were not fucntioning
    s15t=[max(a(:,19)) min(a(:,19)) mean(a(:,19)) std(a(:,19),1)];
    s15b=[max(a(:,20)) min(a(:,20)) mean(a(:,20)) std(a(:,20),1)];
    s19t=[max(a(:,21)) min(a(:,21)) mean(a(:,21)) std(a(:,21),1)];
    s19b=[max(a(:,22)) min(a(:,22)) mean(a(:,22)) std(a(:,22),1)];
    s31b=[max(a(:,23)) min(a(:,23)) mean(a(:,23)) std(a(:,23),1)];
    s32b=[max(a(:,24)) min(a(:,24)) mean(a(:,24)) std(a(:,24),1)];
    s33b=[max(a(:,25)) min(a(:,25)) mean(a(:,25)) std(a(:,25),1)];
    s34b=[max(a(:,26)) min(a(:,26)) mean(a(:,26)) std(a(:,26),1)];
    s35t=[max(a(:,27)) min(a(:,27)) mean(a(:,27)) std(a(:,27),1)];
    s35b=[max(a(:,28)) min(a(:,28)) mean(a(:,28)) std(a(:,28),1)];
    s36b=[max(a(:,29)) min(a(:,29)) mean(a(:,29)) std(a(:,29),1)];
    s37b=[max(a(:,30)) min(a(:,30)) mean(a(:,30)) std(a(:,30),1)];
    s38b=[max(a(:,31)) min(a(:,31)) mean(a(:,31)) std(a(:,31),1)];
    s39t=[max(a(:,32)) min(a(:,32)) mean(a(:,32)) std(a(:,32),1)];
```

```
    s39b=[max(a(:,33)) min(a(:,33)) mean(a(:,33)) std(a(:,33),1)];
    s55t=[max(a(:,34)) min(a(:,34)) mean(a(:,34)) std(a(:,34),1)];
    s55b=[max(a(:,35)) min(a(:,35)) mean(a(:,35)) std(a(:,35),1)];
    s59t=[max(a(:,36)) min(a(:,36)) mean(a(:,36)) std(a(:,36),1)];
    s59b=[max(a(:,37)) min(a(:,37)) mean(a(:,37)) std(a(:,37),1)];
    s65t=dummy;
    s65b=[max(a(:,38)) min(a(:,38)) mean(a(:,38)) std(a(:,38),1)];
    s69t=[max(a(:,39)) min(a(:,39)) mean(a(:,39)) std(a(:,39),1)];
    s69b=[max(a(:,40)) min(a(:,40)) mean(a(:,40)) std(a(:,40),1)];
    s75t=[max(a(:,41)) min(a(:,41)) mean(a(:,41)) std(a(:,41),1)];
    s75b=[max(a(:,42)) min(a(:,42)) mean(a(:,42)) std(a(:,42),1)];
    s79t=[max(a(:,43)) min(a(:,43)) mean(a(:,43)) std(a(:,43),1)];
    s79b=[max(a(:,44)) min(a(:,44)) mean(a(:,44)) std(a(:,44),1)];

    %Organizing the max/min/avg/std calculations for each gauge into
two
    %matrices. One for strain gauges and one for acceleration gauges
    finalrowstrain=[month day s35b(1) s15t s15b s19t s19b s31b s32b
s33b s34b s35t s35b s36b s37b s38b s39t s39b s55t s55b s59t s59b s65t
s65b s69t s69b s75t s75b s79t s79b];
    finalrowaccel=[month day a25 a31 a35x a35y a39 a45 a65x a65y];

    strainmatrix(matindex,:)=finalrowstrain;
    accelmatrix(matindex,:)=finalrowaccel;

    %saving and closing the file Identifier.
    %also clearing variables for the next pass
    save smartdata strainmatrix accelmatrix
    fclose(fid);
%     w
%     stopfile
     if w ~= stopfile
      clear all;
     end

end

%Reports that the program has finished processing
fprintf('Program has completed');

%The following is a list of gauge names and their corresponding
column
%number in the matrix 'a' for graphing purposes

% Gauge Name ----- Column Number
% __________       _____________
% Record #               1
% D15                    2
% D31                    3
% D33                    4
% D35                    5
```

```
% D37                     6
% D39                     7
% D55                     8
% D65                     9
% D75                    10
% A25                    11
% A31                    12
% A35x                   13
% A35y                   14
% A39                    15
% A45                    16
% A65x                   17
% A65y                   18
% S15T                   19
% S15B                   20
% S19T                   21
% S19B                   22
% S31B                   23
% S32B                   24
% S33B                   25
% S34B                   26
% S35T                   27
% S35B                   28
% S36B                   29
% S37B                   30
% S38B                   31
% S39T                   32
% S39B                   33
% S55T                   34
% S55B                   35
% S59T                   36
% S59B                   37
% S65T                   38
% S65B                   39
% S69T                   40
% S69B                   41
% S75T                   42
% S75B                   43
% S79T                   44
% S79B                   45

%Setting up a graphing function
temp=length(a(:,1));
x=1:1:temp;
plot(x,a(:,29)-mean(a(:,29)));

%Change the gauge name in the title as you change it in the plot
command
title(sprintf('Gauge S35B on %i-%i-%i at %i:%i',day, month,
year,hour, minute))
xlabel('Time Record');
ylabel('Microstrain');
```

```
grid on;

%Setting the x axis to the number of points we want
%Also changing the label on the x axis to time rather than record
number
set(gca,'xtick', [1 50 100 150 200 250 300 350 400]);
set(gca, 'xticklabel', [time(1) time(50) time(100) time(150)
time(200) time(250) time(300) time(350) time(400)]);
```

A.4 Monitoring Data Processor Program

```
%%Monitoring Data Processor Program

%This program reads a monitoring data file that is specified by the
user in
%the filename line below. The user must also provide the total number
of
%records in the file to be read.
%To combine multiple files from one month into one matrix, each file
must
%be saved as a separate 'smartdata' file and then combined after all
files
%have been processed. Several examples of this process can be seen at
the
%bottom of this program.

% Enter Filename
filename='C:\Documents and Settings\Administrator\Desktop\Smart
Bridge Research\Monitoring Data\8-25-2010';
% Enter number of records in the file being processed
total_record_num=22;

fid=fopen(filename);

% % % creating date and time vectors of the proper length
year=zeros(total_record_num,1);
month=zeros(total_record_num,1);
day=zeros(total_record_num,1);
hour=zeros(total_record_num,1);
minute=zeros(total_record_num,1);
second=zeros(total_record_num,1);
date=zeros(total_record_num,1);

%define size of strain and acceleration matrix to save data
% mont_data=mont_data;
% JULY_1=zeros(total_record_num,63);
AUGUST_13=zeros(total_record_num,63);

%saving strain and acceleration matrices created
% save smartdata mont_data mont_data_1
```

```
% Skips 4 lines of header information
 for j=1:4

        extraline=fgetl(fid);
 end

 clear a;
for i=1:total_record_num
    scanline=fgetl(fid);
        if scanline < 0, break, end;
        %k represents the length of the date and time information
        %This includes the ',' so the next character is our first
column of
        %data
        for k=1:length(scanline)
            if scanline(k)==','
                break
            end
        end

        %These logic checks ensure that the data and time information
is
        %saved properly
        %Because the number of digits in the second columns change

if k==25
     year(i)=str2num(scanline(2:5));
     month(i)=str2num(scanline(7:8));
     day(i)=str2num(scanline(10:11));
     hour(i)=str2num(scanline(13:14));
     minute(i)=str2num(scanline(16:17));
     second(i)=str2num(scanline(19:20))+str2num(scanline(22:23))/100;

 else if k==24
     year(i)=str2num(scanline(2:5));
     month(i)=str2num(scanline(7:8));
     day(i)=str2num(scanline(10:11));
     hour(i)=str2num(scanline(13:14));
     minute(i)=str2num(scanline(16:17));
     second(i)=str2num(scanline(19:20))+str2num(scanline(22))/100;

 else if k==23
     year(i)=str2num(scanline(2:5));
     month(i)=str2num(scanline(7:8));
     day(i)=str2num(scanline(10:11));
     hour(i)=str2num(scanline(13:14));
     minute(i)=str2num(scanline(16:17));
     second(i)=str2num(scanline(19:20));

 else if k==22
```

```
        year(i)=str2num(scanline(2:5));
        month(i)=str2num(scanline(7:8));
        day(i)=str2num(scanline(10:11));
        hour(i)=str2num(scanline(13:14));
        minute(i)=str2num(scanline(16:17));
        second(i)=0;

        end
        end
        end
end

% % % Creating a single number to represent the date and time
information

date(i)=datenum(year(i),month(i),day(i),hour(i),minute(i),second(i));
% %
% %
% %
% % % % % Reading the data in line by line.
% % % % % '%f's need to be changed to '%*5c' if a gauge stops reading
correctly and
% % % % % is only recording '"NAN"'
% % % % %                                                  1    2    3    4
5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27
28 29 30 31 32   33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49
50 51 52 53 54 55 56 57 58 59 60 61 62 63 64 65
% %
a(:,i)=sscanf(scanline(k+1:length(scanline)),'%f,%*5c,%*5c,%*5c,%f,%f
,%f,%f,%f,%f,%f,%f,%f,%f,%f,%f,%f,%f,%f,%f,%f,%f,%f,%f,%f,%f,%f,%f,%f
,%f,%f,%f,%*5c,%f,%f,%f,%f,%f,%f,%f,%f,%f,%f,%f,%f,%f,%f,%f,%f,%f,%f,
%f,%f,%f,%f,%f,%f,%f,%f,%f,%f,%f,%f,%f,%f,%f',[61,inf]);
% % % % %                                                  1    2    3    4
5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27
28 29 30 31 32   33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49
50 51 52 53 54 55 56 57 58 59 60 61 62 63 64 65       43 44 45 46 47
48 49 50 51 52 53
% % %                                                1    2    3    4  5
d  d  d  d  d  d  d  d  d  s  s  s  s  s  s  s  s  s  s  s  s  s  s
s  s  s  s  s    s  s  s  s  s  s  s  s  s  s  r  r  r  v  v  v  v sg
sg sg sg nr nr nr nr rs rs rs rs  w  w  w  w       41 42 43 44

% % % Reading the data in line by line.
% % % '%f's need to be changed to '%*5c' if a gauge stops reading
correctly and
% % % is only recording '"NAN"'
% % %                                                  1    2    3    4  5  6
7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29
30 31 32   33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51
52 53 54 55 56 57 58 59 60 61 62 63 64 65
a(:,i)=sscanf(scanline(k+1:length(scanline)),'%f,%*5c,%*5c,%*5c,%f,%f
,%f,%f,%f,%f,%f,%f,%f,%f,%f,%f,%f,%f,%f,%f,%f,%f,%f,%f,%f,%f,%f,%f,%f
```

```
,%f,%f,%f,%f,%f,%f,%f,%f,%f,%f,%f,%f,%f,%f,%f,%f,%f,%f,%f,%f,%f,%f,%f,%f
,%f,%f,%f,%f,%f,%f,%f,%f,%f,%f,%f,%f,%f','',[62,inf]);
% % %                                                1    2    3    4  5  6
7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29
30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52
53 54 55 56 57 58 59 60 61 62 63 64 65       43 44 45 46 47 48 49 50
51 52 53
%                                             1    2    3    4  5   d  d
d  d  d  d  d  d  d  s  s  s  s  s  s  s  s  s  s  s  s  s  s  s  s
s  s  s    s  s  s  s  s  s  s  s  s  s  s  r  r  r  v  v  v  v sg sg sg
sg nr nr nr nr rs rs rs rs  w  w  w  w       41 42 43 44

end

% % %Transposing the matrix so the data is organized by column rather
then by
% % %row
a=a';

% load smartdata
AUGUST_13=[date a];

% mont_data=[mont_data 1 mont_data_2];

save smartdata august
fclose(fid);
fprintf('Program has completed');

% %
august=[AUGUST_1;AUGUST_2;AUGUST_3;AUGUST_4;AUGUST_5;AUGUST_6;AUGUST_
7;AUGUST_8;AUGUST_9;AUGUST_10;AUGUST_11;AUGUST_12;AUGUST_13;AUGUST_14
;AUGUST_15;AUGUST_13];
% % august=sortrows(august,1);
% % august=unique(august,'rows');
% % plot(1:length(august),august(:,21));
% % xlabel('Record Number');
% % % % plot(august(:,1),august(:,21))
% % % % xlabel('Datenum')
% % ylabel('Strain');
% % title('August 2010: Trigger Gage S35B');
% % grid on;

%july=[JULY_1;JULY_2;JULY_3;JULY_4;JULY_5;JULY_6;JULY_7;JULY_8;JULY_9
;];
% % july=sortrows(july,1);
% % july=unique(july,'rows');
% % plot(1:length(july),july(:,21));
% % xlabel('Record Number');
% % % % plot(july(:,1),july(:,21))
% % % % xlabel('Datenum')
% % ylabel('Strain');
```

```
% % title('July 2010: Trigger Gage S35B');
% % grid on;

%june=[JUNE_1;JUNE_2;JUNE_3;JUNE_4;JUNE_5;JUNE_6;JUNE_7;JUNE_8;JUNE_9
;JUNE_10;JUNE_11];
% % june=sortrows(june,1);
% % june=unique(june,'rows');
% % plot(1:length(june),june(:,21));
% % xlabel('Record Number');
% % % % plot(june(:,1),june(:,21))
% % % % xlabel('Datenum')
% % ylabel('Strain');
% % title('June 2010: Trigger Gage S35B');
% % grid on;

% % may=[MAY_1;MAY_2;MAY_3;MAY_4;MAY_5;MAY_6;MAY_7;MAY_8;MAY_9;];
% % m=may(2:595,:);
% % may=m;
% % may=sortrows(may,1);
% % may=unique(may,'rows');
% % plot(1:length(may),may(:,21));
% % xlabel('Record Number');
% % % % plot(may(:,1),may(:,21))
% % % % xlabel('Datenum')
% % ylabel('Strain');
% % title('May 2010: Trigger Gage S35B');
% % grid on;

%april=[APRIL_1;APRIL_2;APRIL_5;APRIL_12;APRIL_13;APRIL_19;APRIL_20;A
PRIL_21;APRIL_27;APRIL_29;];
% april=sortrows(april,1);
% april=unique(april,'rows');
% plot(april(:,1),april(:,24))
% xlabel('Datenum')
% ylabel('Strain')
% title('S35B Mont. Data  4-1 to 4-30')

% Had to remove a row of zeros from the final matrix 'march'
%march=[MARCH_1;MARCH_2;MARCH_3;MARCH_4;MARCH_5;MARCH_6;MARCH_7;MARCH
_8;
%     MARCH_9;MARCH_10;MARCH_11;MARCH_12];
% march=sortrows(march,1);
% march=unique(march,'rows');
% plot(march(:,1),march(:,24))
% xlabel('Datenum')
% ylabel('Strain')
% title('S35B Mont. Data  3-1 to 3-31')
```

A.5 PC9000 Data Generating Program

```
'Modified 2-2-2010 to provide 1 averaged output of
vibrating wire measurements, every hour. This
'replaces the previous method of taking 4-5 samples on
top of every hour.
'NOTE: The one measurement average is computed over a 10
1/2 second interval.

Const PERIOD = 10
Const P_UNITS = 1
Const INTERVAL1 = 10
Const UNITS1 = 1
Const INTERVAL2 = 2100
Const UNITS2 = 1
Const TRNG1 = 17
Const TTYPE1 = 0
Const TREP1 = 3
Const TSETL1 = 30
Const TINT1 = 40
Const TMULT1 = 1.8
Const TOSET1 = 32
Public TBlk1(TREP1)
Units TBlk1 = Deg_F
Const VRNG1 = 0
Const VREP1 = 9
Const VSETL1 = 30
Const VINT1 = 40
Const VMULT1 = 1
Const VOSET1 = 0
Public VBlk1(VREP1)
Public MVBlk1(VREP1)
Public OVBlk1(VREP1)
Units VBlk1 = mVolts
Dim VBlk1Zero(VREP1)
Const BRNG1 = 0
Const BREP1 = 8
Const BEXCIT1 = 5000
Const BSETL1 = 30
Const BINT1 = 40
Const BMULT1 = 5
Const BOSET1 = 0
Public BBlk1(BREP1)
```

```
Dim OBBlk1(BREP1)
Dim BBlk1ZeroMv(BREP1)
Units BBlk1ZeroMv = mVperV
Units BBlk1 = mVoltPVolt
Const BRNG2 = 16
Const BREP2 = 27
Const BEXCIT2 = 5000
Const BSETL2 = 30
Const BINT2 = 40
Const BGF2 = 2
Const BCODE2 = -1
Const BMULT2 = 1
Const BOSET2 = 0
Public BBlk2(BREP2)
Dim GBBlk2(BREP2)
Dim BBlk2ZeroMv(BREP2)
Dim BBlk2ZeroUs(BREP2)
Units BBlk2ZeroMv = mVperV
Units BBlk2ZeroUs = uStrain
Units BBlk2 = uStrain
Const BRNG3 = 1
Const BREP3 = 2
Const BEXCIT3 = 5000
Const BSETL3 = 30
Const BINT3 = 40
Const BGF3 = 2
Const BCODE3 = -1
Const BMULT3 = 1
Const BOSET3 = 0
Public BBlk3(BREP3)
Dim GBBlk3(BREP3)
Dim BBlk3ZeroMv(BREP3)
Dim BBlk3ZeroUs(BREP3)
Units BBlk3ZeroMv = mVperV
Units BBlk3ZeroUs = uStrain
Units BBlk3 = uStrain
Const BRNG4 = 0
Const BREP4 = 3
Const BEXCIT4 = 5000
Const BSETL4 = 30
Const BINT4 = 40
Const BGF4 = 2
Const BCODE4 = -1
Const BMULT4 = 1
```

```
Const BOSET4 = 0
Public BBlk4(BREP4)
Units BBlk4 = uStrain
const CR9060_slot = 10
const CR9052_slot = 11
const CR9050_slot = 7
const T_vbw_subscan = 60
const N_vbw_subscan = 35000
const T_vbw_scan = T_vbw_subscan*N_vbw_subscan
const N_vbw_scan = 5
const vbw1_Vex_02p = 5000
const vbw1_f_lo = 1400
const vbw1_f_hi = 3500
const vbw1_range = mV200
const vbw1_min_amp   =  0.1
const vbw1_max_amp   = 40.0
const vbw1_min_smnr  = 1.5
const vbw1_min_t_res = 1000
const vbw1_max_t_res = 4000
const vbw2_Vex_02p   = 5000
const vbw2_f_lo      = 1400
const vbw2_f_hi      = 3500
const vbw2_range     = mV200
const vbw2_min_amp   =  0.1
const vbw2_max_amp   = 40.0
const vbw2_min_smnr  = 1.5
const vbw2_min_t_res = 1000
const vbw2_max_t_res = 4000
const vbw3_Vex_02p   = 5000
const vbw3_f_lo      = 1400
const vbw3_f_hi      = 3500
const vbw3_range     = mV200
const vbw3_min_amp   =  0.1
const vbw3_max_amp   = 40.0
const vbw3_min_smnr  = 1.5
const vbw3_min_t_res = 1000
const vbw3_max_t_res = 4000
const vbw4_Vex_02p   = 5000
const vbw4_f_lo      = 1400
const vbw4_f_hi      = 3500
const vbw4_range     = mV200
const vbw4_min_amp   =  0.1
const vbw4_max_amp   = 40.0
const vbw4_min_smnr  = 1.5
```

```
const vbw4_min_t_res = 1000
const vbw4_max_t_res = 4000
Const ts_len = 4096
'--------------------------------Added  222010-----------
------------------------------------------------------------
--
Public TrigVar    'added 2-1-2010 to facilitate trigger
threshold changes.
Public VWScanCount 'Added to control the number of
averages in the Mont data table.
Public S35BAvg, S36BAvg, S34BAvg, ReadAVG,CNT_AVG
Public rTime (9)
Alias rTime(1) = Year          'assign the alias Year to
rTime(1)
Alias rTime(2) = Month         'assign the alias Month to
rTime(2)
Alias rTime(3) = DOM           'assign the alias Day to
rTime(3)
Alias rTime(4) = Hour          'assign the alias Hour to
rTime(4)
Alias rTime(5) = Minute        'assign the alias Minute
to rTime(5)
Alias rTime(6) = Second        'assign the alias Second
to rTime(6)
Alias rTime(7) = uSecond       'assign the alias uSecond
to rTime(7)
Alias rTime(8) = WeekDay       'assign the alias WeekDay
to rTime(8)
Alias rTime(9) = Day_of_Year   'assign the alias
Day_of_Year to rTime(9)
'------------------------------------------------------------
------------------------------------------------------------
--
Public vbw_f_peak (4)
units  vbw_f_peak = Hz
public vbw_sig_str (4)
units  vbw_sig_str = mV rms
public vbw_smnr (4)
units vbw_smnr = no_units
public vbw_t_res (4)
Units  vbw_t_res = Ohms
public vbw_warning (4)
units  vbw_warning = no_units
public battv
```

```
units battv = V
Alias TBlk1(1) = TD35
Alias TBlk1(2) = T65
Alias TBlk1(3) = TCOU
Alias VBlk1(1) = D15
Alias VBlk1(2) = D31
Alias VBlk1(3) = D33
Alias VBlk1(4) = D35
Alias VBlk1(5) = D37
Alias VBlk1(6) = D39
Alias VBlk1(7) = D55
Alias VBlk1(8) = D65
Alias VBlk1(9) = D75
Alias BBlk1(1) = A25
Alias BBlk1(2) = A31
Alias BBlk1(3) = A35X
Alias BBlk1(4) = A35Y
Alias BBlk1(5) = A39
Alias BBlk1(6) = A45
Alias BBlk1(7) = A65X
Alias BBlk1(8) = A65Y
Alias BBlk2(1) = S15T
Alias BBlk2(2) = S15B
Alias BBlk2(3) = S19T
Alias BBlk2(4) = S19B
Alias BBlk2(5) = S31B
Alias BBlk2(6) = S32B
Alias BBlk2(7) = S33B
Alias BBlk2(8) = S34B
Alias BBlk2(9) = S35T
Alias BBlk2(10) = S35B
Alias BBlk2(11) = S36B
Alias BBlk2(12) = S37B
Alias BBlk2(13) = S38B
Alias BBlk2(14) = S39T
Alias BBlk2(15) = S39B
Alias BBlk2(16) = S55T
Alias BBlk2(17) = S55B
Alias BBlk2(18) = S59T
Alias BBlk2(19) = S59B
Alias BBlk2(20) = S65T
Alias BBlk2(21) = S65B
Alias BBlk2(22) = S69T
Alias BBlk2(23) = S69B
```

```
Alias BBlk2(24) = S75T
Alias BBlk2(25) = S75B
Alias BBlk2(26) = S79T
Alias BBlk2(27) = S79B
Alias BBlk3(1) = SDUM
Alias BBlk3(2) = SCOU
Alias BBlk4(1) = R35
Alias BBlk4(2) = RCOU
Alias BBlk4(3) = R39
Alias vbw_f_peak (1) = V55
Alias vbw_f_peak (2) = V65
Alias vbw_f_peak (3) = V35T
Alias vbw_f_peak (4) = V35B
Public Flag(8)
Dim I
Dim Count
Dim TRef(4)
'-----------------------------------------------Added 2-
2-2010 ------------------------------------------------
-
DataTable(EVENT,True,150000)
DataInterval(0,INTERVAL1,UNITS1,500)
DataEvent (200,(S35B>= TrigVar OR S36B>= TrigVar OR S34B
>= TrigVar),True,200)
'------------------------------------------------------
------------------------------------------------------
--
CardOut(1,150000)
Sample (1,TrigVar,FP2)
Sample (VREP1,VBlk1(),IEEE4)
Sample (BREP1,BBlk1(),IEEE4)
Sample (BREP2,BBlk2(),IEEE4)
EndTable
DataTable(MONT,VWScanCount=5,-1)
'DataInterval(0,INTERVAL2,UNITS2,1000)   DO NOT NEED when
using trigger variable in table statement
CardOut(1,-1)
Average (TREP1,TBlk1(),FP2,0)
Average (VREP1,VBlk1(),IEEE4,0)
Average (BREP2,BBlk2(),IEEE4,0)
Average (BREP3,BBlk3(),IEEE4,0)
Average (BREP4,BBlk4(),IEEE4,0)
Average (4, vbw_f_peak  (1), ieee4,0)
Average (4, vbw_sig_str (1), ieee4,0)
```

```
Average (4, vbw_smnr    (1), ieee4,0)
Average (4, vbw_t_res   (1), ieee4,0)
Average (4, vbw_warning (1), ieee4,0)
EndTable
DataTable(ZERO_1,Count>99,100)
Average(BREP2,BBlk2ZeroMv(),IEEE4,False)
Average(BREP2,BBlk2ZeroUs(),IEEE4,False)
EndTable
DataTable(ZERO_2,Count>99,100)
Average(BREP1,BBlk1ZeroMv(),IEEE4,False)
EndTable
DataTable(ZERO_3,Count>99,100)
Average(VREP1,VBlk1Zero(),IEEE4,False)
EndTable
DataTable(ZERO_4,Count>99,100)
Average(BREP3,BBlk3ZeroMv(),IEEE4,False)
Average(BREP3,BBlk3ZeroUs(),IEEE4,False)
EndTable
Sub Zero1
Count = 0
Scan(PERIOD,P_UNITS,100,100)
BrFull(BBlk2ZeroMv(),BREP2,BRNG2,5,7,8,15,3,BEXCIT2,False
,True,BSETL2,BINT2,BMULT2,BOSET2)
StrainCalc(BBlk2ZeroUs(),BREP2,BBlk2ZeroMv(),0,BCODE2,GBB
lk2(),0)
Count = Count + 1
CallTable ZERO_1
Next Scan
For I = 1 To BREP2
BBlk2ZeroMv(I) = ZERO_1.BBlk2ZeroMv_Avg(I,1)
Next I
Flag(1) = False
EndSub
Sub Zero2
Count = 0
Scan(PERIOD,P_UNITS,100,100)
BrFull(BBlk1ZeroMv(),BREP1,BRNG1,4,13,8,7,1,BEXCIT1,False
,False,BSETL1,BINT1,BMULT1,0)
Count = Count + 1
CallTable ZERO_2
Next Scan
For I = 1 To BREP1
OBBlk1(I) = -ZERO_2.BBlk1ZeroMv_Avg(I,1)
Next I
```

```
Flag(2) = False
EndSub
Sub Zero3
Count = 0
Scan(PERIOD,P_UNITS,100,100)
VoltDiff(VBlk1Zero(),VREP1,VRNG1,4,4,True,VSETL1,VINT1,MV
Blk1(),0)
Count = Count + 1
CallTable ZERO_3
Next Scan
For I = 1 To VREP1
OVBlk1(I) = -ZERO_3.VBlk1Zero_Avg(I,1)
Next I
Flag(3) = False
EndSub
Sub Zero4
Count = 0
Scan(PERIOD,P_UNITS,100,100)
BrFull(BBlk3ZeroMv(),BREP3,BRNG3,7,6,9,14,2,BEXCIT3,False
,True,BSETL3,BINT3,BMULT3,BOSET3)
StrainCalc(BBlk3ZeroUs(),BREP3,BBlk3ZeroMv(),0,BCODE3,GBB
lk3(),0)
Count = Count + 1
CallTable ZERO_4
Next Scan
For I = 1 To BREP3
BBlk3ZeroMv(I) = ZERO_4.BBlk3ZeroMv_Avg(I,1)
Next I
Flag(4) = False
EndSub
BeginProg
TrigVar = 1110        '*****2-2-2010******set trigger
threshold variable **********************
RealTime(rtime)
For I = 1 To VREP1
MVBlk1(I) = VMULT1
OVBlk1(I) = VOSET1
Next I
For I = 1 To BREP1
OBBlk1(I) = BOSET1
Next I
For I = 1 To BREP2
GBBlk2(I) = BGF2
Next I
```

```
For I = 1 To BREP3
GBBlk3(I) = BGF3
Next I
MVBlk1(1) = 1
MVBlk1(2) = 1
MVBlk1(3) = 1
MVBlk1(4) = 1
MVBlk1(5) = 1
MVBlk1(6) = 1
MVBlk1(7) = 1
MVBlk1(8) = 1
MVBlk1(9) = 1
While (True)
Scan (T_vbw_scan, uSec, 20, N_vbw_scan)
     RealTime (rTime)
include "cpu:incnew.c9x"
If Status.MeasureTime(1,1) >= 20 Then
Flag(8) = True
ElseIf Status.MeasureTime(1,1) > 30 Then
Flag(8) = False
EndIf
BrHalf (vbw_t_res (1), 1, mV5000, CR9050_slot, 21,
CR9060_slot, 12, 1, 5000, False, 1000, 16667, 1.0, 0.0)
vbw_t_res (1) =  vbw_t_res (1) * 350/(1 - vbw_t_res (1))
BrHalf (vbw_t_res (2), 1, mV5000, CR9050_slot, 23,
CR9060_slot, 13, 1, 5000, False, 1000, 16667, 1.0, 0.0)
vbw_t_res (2) =  vbw_t_res (2) * 10e3/(1 - vbw_t_res (2))
BrHalf (vbw_t_res (3), 1, mV5000, CR9050_slot, 25,
CR9060_slot, 14, 1, 5000, False, 1000, 16667, 1.0, 0.0)
vbw_t_res (3) =  vbw_t_res (3) * 10e3/(1 - vbw_t_res (3))
BrHalf (vbw_t_res (4), 1, mV5000, CR9050_slot, 27,
CR9060_slot, 15, 1, 5000, False, 1000, 16667, 1.0, 0.0)
vbw_t_res (4) =  vbw_t_res (4) * 10e3/(1 - vbw_t_res (4))
ModuleTemp(TRef(),1,4,20)
TCDiff(TBlk1(),TREP1,TRNG1,4,1,TTYPE1,TRef(1),True,TSETL1
,TINT1,TMULT1,TOSET1)
BrFull(BBlk2(),BREP2,BRNG2,5,7,8,15,3,BEXCIT2,False,True,
BSETL2,BINT2,BMULT2,BOSET2)
StrainCalc(BBlk2(),BREP2,BBlk2(),BBlk2ZeroMv(),BCODE2,BGF
2,0)
BrFull(BBlk3(),BREP3,BRNG3,7,6,9,14,2,BEXCIT3,False,True,
BSETL3,BINT3,BMULT3,BOSET3)
StrainCalc(BBlk3(),BREP3,BBlk3(),BBlk3ZeroMv(),BCODE3,BGF
3,0)
```

```
BrFull(BBlk4(),BREP4,BRNG4,7,8,9,15,1,BEXCIT4,False,False
,BSETL4,BINT4,BMULT4,BOSET4)
StrainCalc(BBlk4(),BREP4,BBlk4(),0,BCODE4,BGF4,0)
VoltDiff(VBlk1(),VREP1,VRNG1,4,4,True,VSETL1,VINT1,MVBlk1
(),OVBlk1())
If Flag(3) Then Zero3
If Flag(4) Then Zero4
VWScanCount=VWScanCount+1 '------ 2-2-2010 ------ counter
for Monitor Table (MONT)
vbw_warning (1) = 0
If (vbw_sig_str (1) <  vbw1_min_amp)   Then vbw_warning
(1) =  1 OR vbw_warning (1)
if (vbw_sig_str (1) >  vbw1_max_amp)   then vbw_warning
(1) =  2 or vbw_warning (1)
if (vbw_f_peak  (1) =  vbw1_f_lo)      then vbw_warning
(1) =  4 or vbw_warning (1)
if (vbw_f_peak  (1) =  vbw1_f_hi)      then vbw_warning
(1) =  8 or vbw_warning (1)
if (vbw_smnr    (1) <  vbw1_min_smnr)  then vbw_warning
(1) = 16 or vbw_warning (1)
if (vbw_f_peak  (1) =  nan)            then vbw_warning
(1) = 32 or vbw_warning (1)
if ((vbw_t_res (1) < vbw1_min_t_res) or (vbw_t_res (1) >
vbw1_max_t_res) or (vbw_t_res (1) = nan)) then
vbw_warning (1) = 64 OR vbw_warning (1)
endif
vbw_warning (2) = 0
if (vbw_sig_str (2) <  vbw2_min_amp)   then vbw_warning
(2) =  1 or vbw_warning (2)
If (vbw_sig_str (2) >  vbw2_max_amp)   Then vbw_warning
(2) =  2 OR vbw_warning (2)
if (vbw_f_peak  (2) =  vbw2_f_lo)      then vbw_warning
(2) =  4 or vbw_warning (2)
If (vbw_f_peak  (2) =  vbw2_f_hi)      Then vbw_warning
(2) =  8 OR vbw_warning (2)
if (vbw_smnr    (2) <  vbw2_min_smnr)  then vbw_warning
(2) = 16 or vbw_warning (2)
If (vbw_f_peak  (2) =  nan)            Then vbw_warning
(2) = 32 OR vbw_warning (2)
if ((vbw_t_res (2) < vbw2_min_t_res) or (vbw_t_res (2) >
vbw2_max_t_res) or (vbw_t_res (2) = nan)) then
vbw_warning (2) = 64 OR vbw_warning (2)
EndIf
vbw_warning (3) = 0
```

```
if (vbw_sig_str (3) <  vbw3_min_amp)   then vbw_warning
(3) =  1 or vbw_warning (3)
if (vbw_sig_str (3) >  vbw3_max_amp)   then vbw_warning
(3) =  2 or vbw_warning (3)
if (vbw_f_peak  (3) =  vbw3_f_lo)      then vbw_warning
(3) =  4 or vbw_warning (3)
if (vbw_f_peak  (3) =  vbw3_f_hi)      then vbw_warning
(3) =  8 or vbw_warning (3)
if (vbw_smnr    (3) <  vbw3_min_smnr)  then vbw_warning
(3) = 16 or vbw_warning (3)
if (vbw_f_peak  (3) =  nan)            then vbw_warning
(3) = 32 or vbw_warning (3)
if ((vbw_t_res (3) < vbw3_min_t_res) or (vbw_t_res (3) >
vbw3_max_t_res) or (vbw_t_res (3) = nan)) then
vbw_warning (3) = 64 or vbw_warning (3)
endif
vbw_warning (4) = 0
if (vbw_sig_str (4) <  vbw4_min_amp)   then vbw_warning
(4) =  1 or vbw_warning (4)
if (vbw_sig_str (4) >  vbw4_max_amp)   then vbw_warning
(4) =  2 or vbw_warning (4)
if (vbw_f_peak  (4) =  vbw4_f_lo)      then vbw_warning
(4) =  4 or vbw_warning (4)
if (vbw_f_peak  (4) =  vbw4_f_hi)      then vbw_warning
(4) =  8 or vbw_warning (4)
if (vbw_smnr    (4) <  vbw4_min_smnr)  then vbw_warning
(4) = 16 or vbw_warning (4)
if (vbw_f_peak  (4) =  nan)            then vbw_warning
(4) = 32 or vbw_warning (4)
if ((vbw_t_res (4) < vbw4_min_t_res) or (vbw_t_res (4) >
vbw4_max_t_res) or (vbw_t_res (4) = nan)) then
vbw_warning (4) = 64 or vbw_warning (4)
EndIf
CallTable MONT

'---------------------------------------Added 2-2-2010---
-------------------------------------------------------
--
If VWScanCount=5 Then    'loop through VWscan loop 5
times before taking 1 average in the Mont table
    VWScanCount = 0
    ReadAVG = 101
    CNT_AVG = CNT_AVG + 1
 EndIf
```

```
'-------------------------------------------------------
-------------------------------------------------------
--
NextScan
Scan(PERIOD,P_UNITS,10,360000)
     RealTime (rTime)
If Status.MeasureTime(1,1) >= 20 Then
Flag(8) = True
ElseIf Status.MeasureTime(1,1) > 30 Then
Flag(8) = False
EndIf
'----------------------------------Added 2-2-2010--------
-------------------------------------------------------
S35BAvg = Mont.S35B_Avg(1,1)
S34BAvg = Mont.S34B_Avg(1,1)
S36BAvg = Mont.S36B_Avg(1,1)
'-------------------------------------------------------
-------------------------------------------------------
--
VoltDiff(VBlk1(),VREP1,VRNG1,4,4,True,VSETL1,VINT1,MVBlk1
(),OVBlk1())
BrFull(BBlk1(1),2,BRNG1,4,13,8,7,1,BEXCIT1,False,False,BS
ETL1,BINT1,BMULT1,OBBlk1(1))
BrFull(BBlk1(3),2,BRNG1,5,1,8,1,2,BEXCIT1,False,False,BSE
TL1,BINT1,BMULT1,OBBlk1(3))
BrFull(BBlk1(5),2,BRNG1,5,3,8,11,1,BEXCIT1,False,False,BS
ETL1,BINT1,BMULT1,OBBlk1(5))
BrFull(BBlk1(7),2,BRNG1,5,5,8,2,2,BEXCIT1,False,False,BSE
TL1,BINT1,BMULT1,OBBlk1(7))
BrFull(BBlk2(),BREP2,BRNG2,5,7,8,15,3,BEXCIT2,False,True,
BSETL2,BINT2,BMULT2,BOSET2)
StrainCalc(BBlk2(),BREP2,BBlk2(),BBlk2ZeroMv(),BCODE2,BGF
2,0)
'-------------------Added TrigVar code for Table Event
2-2-2010-----------------------------------
If ReadAVG = 101 Then
       If S35BAvg <>0 Then
          TrigVar = S35BAvg + 110
          ElseIf S36BAvg<> 0 Then
               TrigVar = S36BAvg + 110
          ElseIf S34BAvg <>  0 Then
                TrigVar = S34BAvg + 110
                ElseIf S35BAvg  = NAN Then
                     TrigVar = 1111
```

```
                EndIf
    ReadAVG = 0
EndIf
'-------------------------------------------------------------
-------------------------------------------------------------
--
If Flag(3) Then Zero3
If Flag(2) Then Zero2
If Flag(1) Then Zero1
If Flag(4) Then Zero4
CallTable EVENT
Next Scan
Wend
SlowSequence
Dim CountSlow
Dim TripVolt
Dim CountAvg
Scan(1,Sec,0,0)
Calibrate
BiasComp
Battery(TripVolt,0)
CountSlow = CountSlow + 1
If CountSlow >= 60 Then
CountSlow = 0
AvgRun(TripVolt,1,TripVolt,10)
CountAvg = CountAvg + 1
If CountAvg > 9 Then
CountAvg = 0
If TripVolt < 11.5 Then
PowerOff(0,1,Hr)
EndIf
EndIf
EndIf
Next Scan
EndProg
```

APPENDIX B: MONTHLY BREAKDOWN GRAPHS

B.1 Monthly Breakdowns

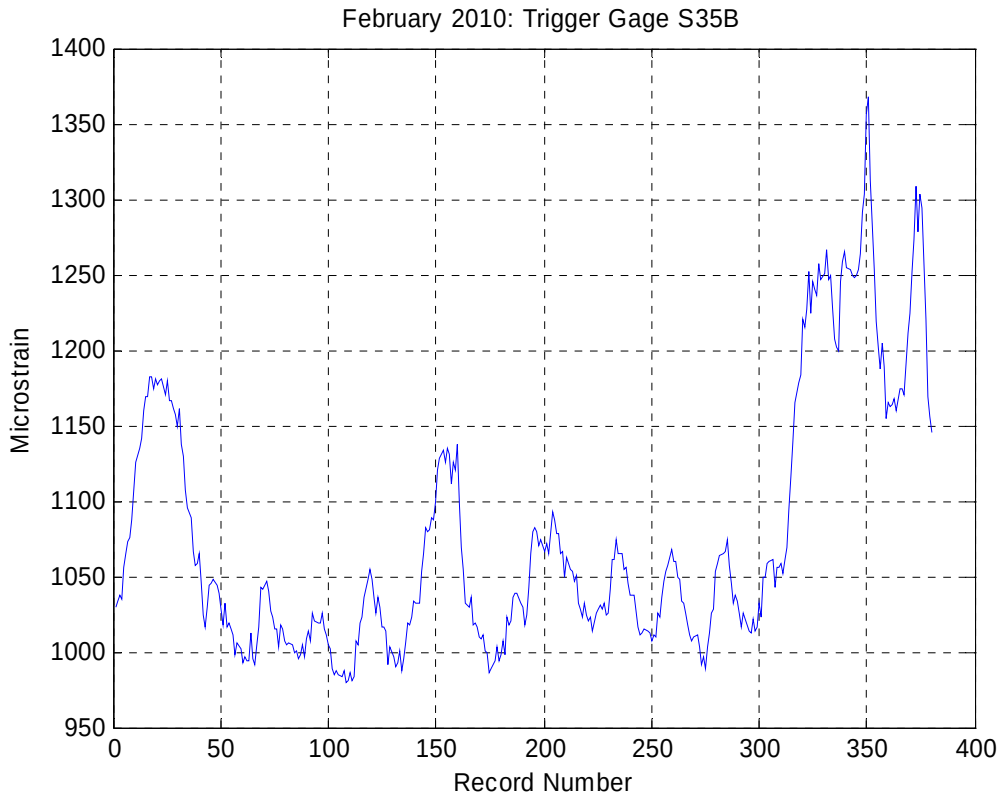


Figure B.1 Monitoring data for S35B for February 2010

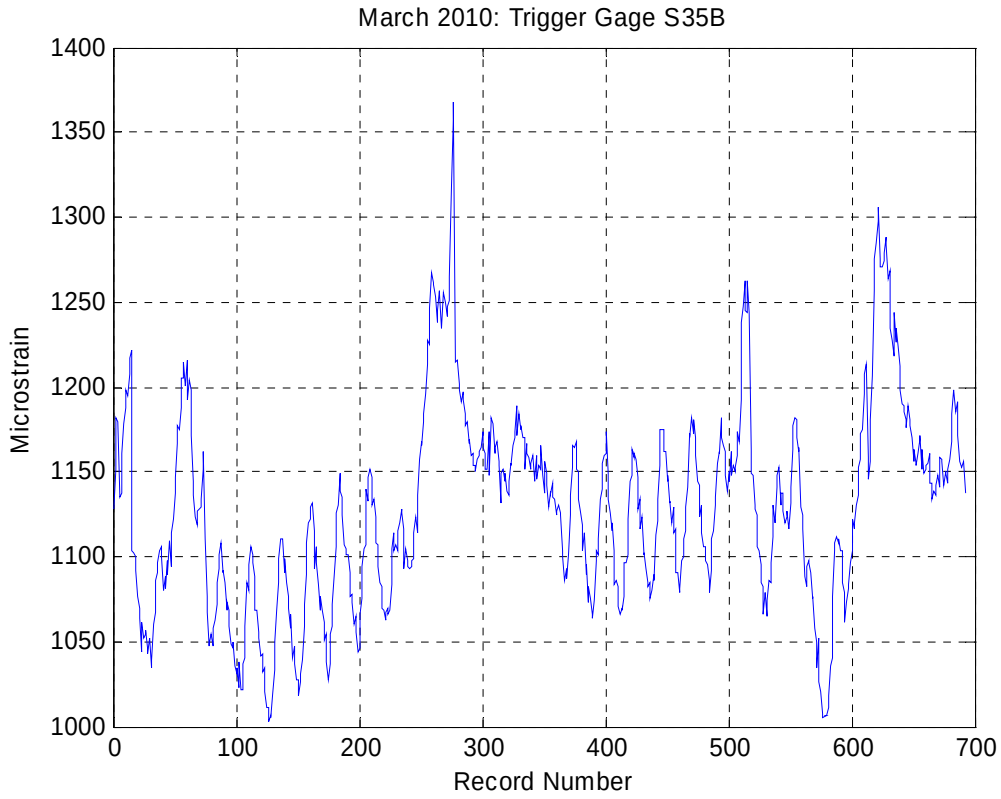


Figure B.2 Monitoring data for S35B for March 2010

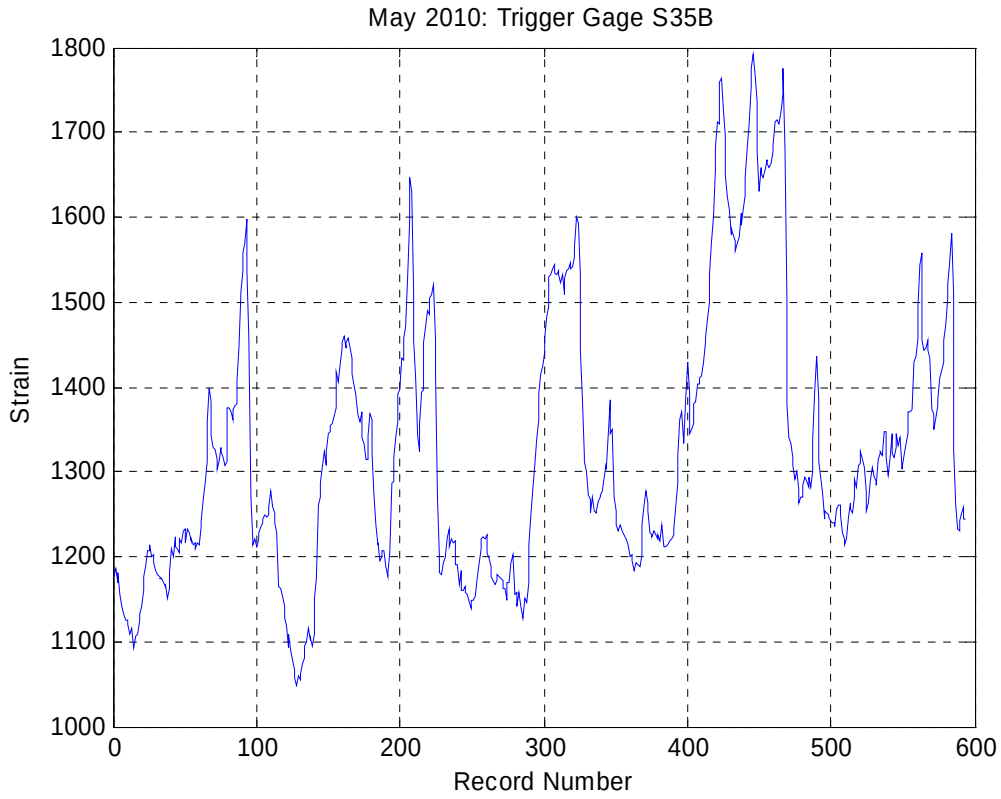


Figure B.3 Monitoring data for S35B for May 2010

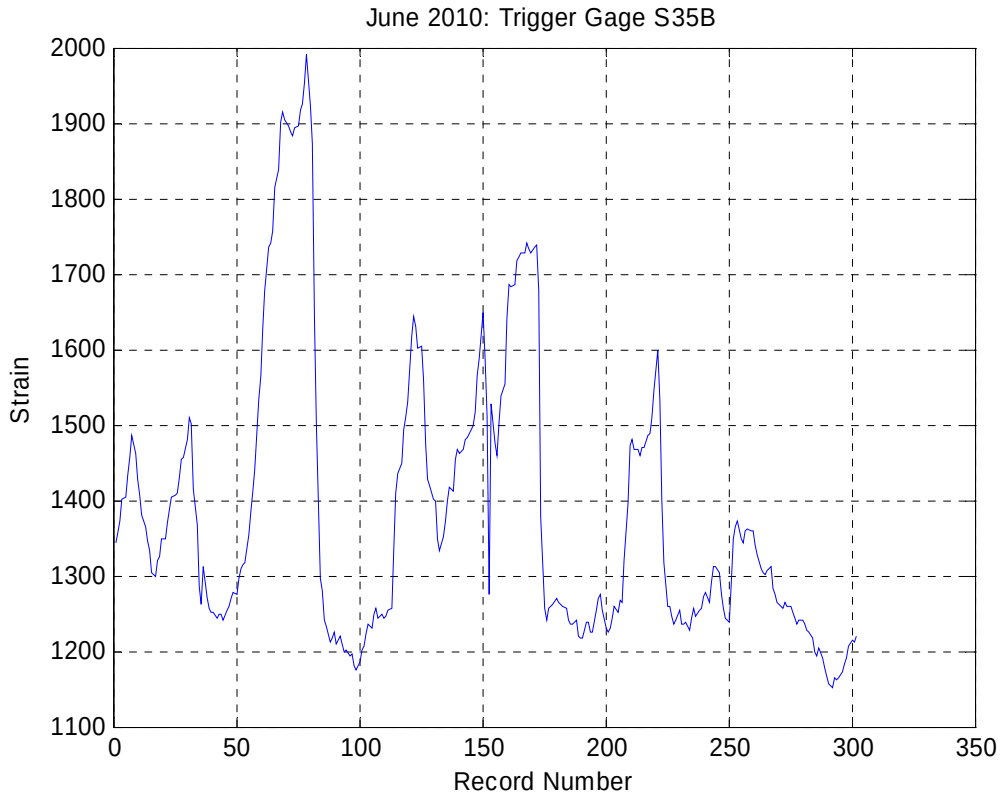


Figure B.4 Monitoring data for S35B for June 2010

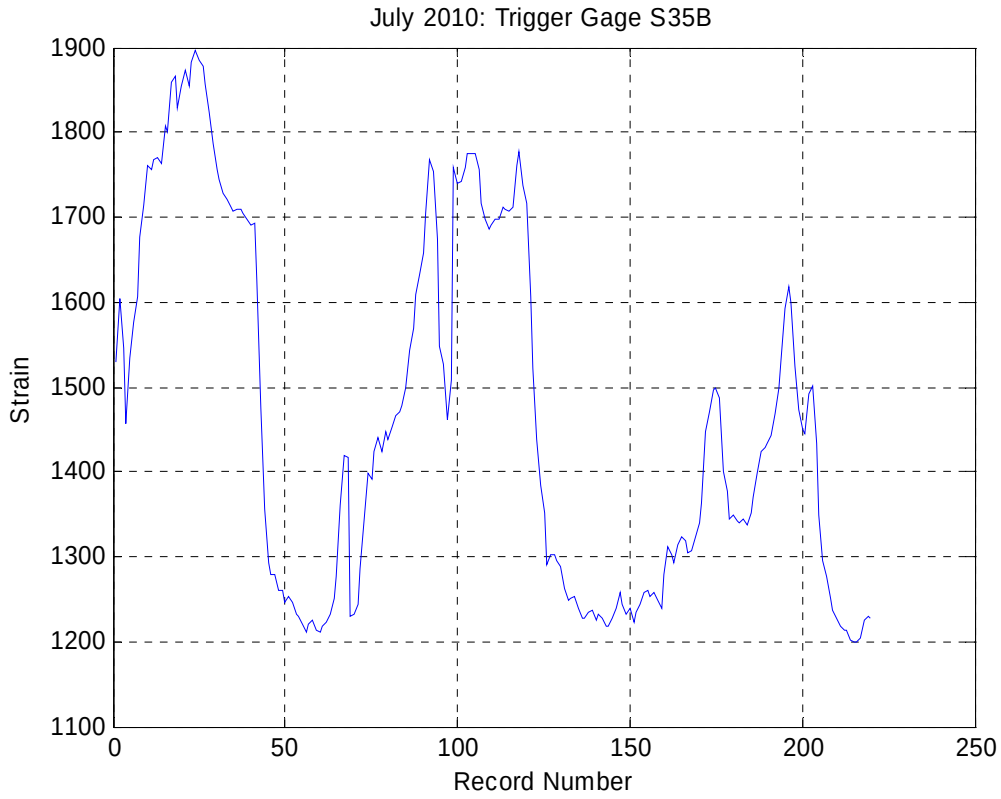


Figure B.5 Monitoring data for S35B for July 2010

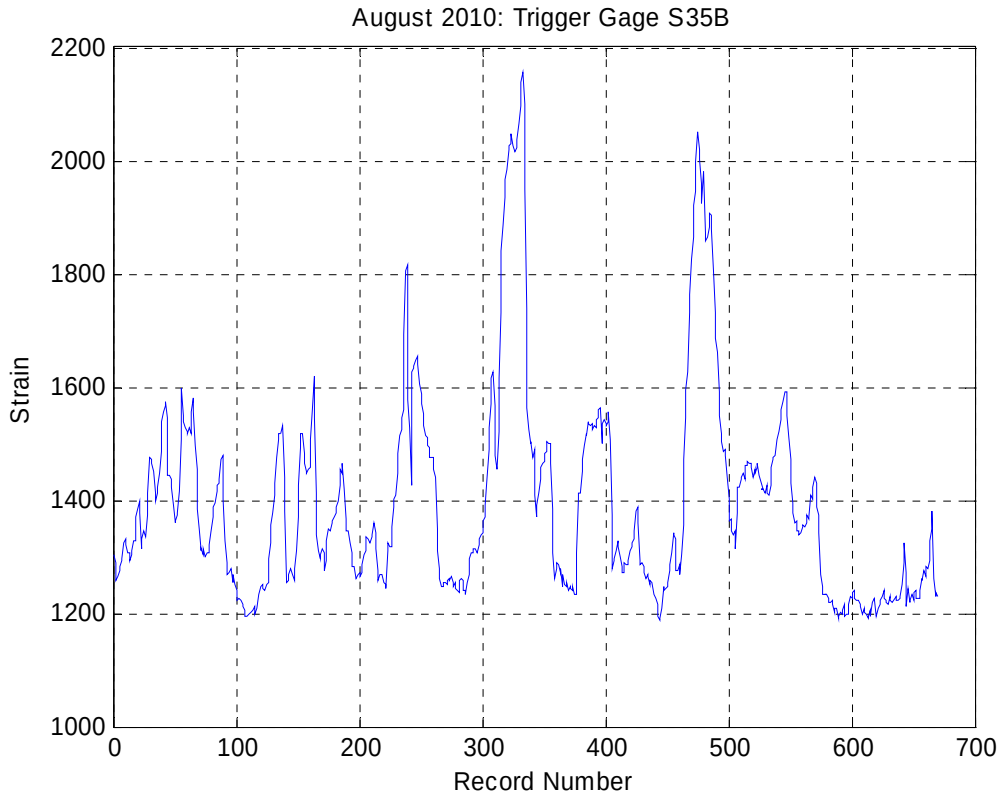


Figure B.6 Monitoring data for S35B for August 2010

APPENDIX C: LOAD TEST

C.1 Load Test

In order to get a better perspective of the magnitude of the events being recorded, a semi-controlled load test was conducted with the DelDOT UBIV truck on December 8 and December 10. During these tests two lanes of the bridge were shut down at a time, the shoulder and the adjacent lane of traffic. The west (left) side of the bridge was tested on December 8 and the east (right) side of the bridge was tested on December 10. Since the entire bridge was not closed for the test, it cannot be called a truly controlled load test. The tests did give a nice reference point, however, to compare the strain caused by the 66,000 pound UBIV to the daily events recorded by the data logger.

Rather than write an entirely new PC9000 program to use for the load test, the normal code was altered to meet the needs for the load test. Prior to going out to the bridge on December 8 the PC9000 code was modified so the event data table could be manually triggered, rather than waiting until the trigger gage reached a predetermined threshold. This meant that instead of simply taking 401 points for an event, data could be recorded to the event table as long as was desired. The differences between the two PC9000 event table code calls can be seen below:

Original trigger gage code for event table

```
DataTable(EVENT,True,-1)
DataEvent (200,S35B >=100,True,200)
```

Load Test code for event table

```
DataTable(Event,TriggerVarTab,5000)
If Flag(7) = -1 Then
TriggerVarTab= -1
Else
TriggerVarTab = 0
EndIf
```

The original code call tells the logger that an event has occurred when a strain of greater than 100 microstrain is logged for gage S35B. The new code, however, allows for a user to control an event stopping and starting by manually changing the "Flag 7" control in the PC9000 program from 'true' to 'untrue.' This flag control could have been done from a remote site using the internet connection associated with the data logger as well, but for the load test a laptop equipped with PC9000 software was directly connected to logger at the bridge.

Rick Moore was operating the UBIV during the test and communicated by cell phone when the truck was approaching the bridge to begin recording data. The UBIV was driven at about 10 miles per hour in the closed traffic lane, resulting in about 20 seconds of data for each pass of the test. Three passes were done in the left lanes on December 8 and three passes were conducted in the right lanes on December 10 when the right side of the bridge was closed. Data from one of the right lane load tests can be seen below in Figure C.1. The two axels of the truck can clearly be identified by the peak strains seen between record 1000 and record 1200. This was the best pass of data, as it resulted in the clearest peaks and had the least amount of background noise from other traffic on the bridge.

Delaware Center for Transportation University of Delaware Newark, Delaware 19716

AN EQUAL OPPORTUNITY/AFFIRMATIVE ACTION EMPLOYER

To the extent permitted by applicable State and Federal laws, the University of Delaware is committed to assuring equal opportunity to all persons and does not discriminate on the basis of race, creed, color, sex, age, religion, national origin, veteran or handicapped status, or gender identity and expression, or sexual orientation in its educational programs, activities, admissions, or employment practices as required by Title IX of the Educational Amendments of 1972, Section 504 of the Rehabilitation Act of 1973, Title VII of the Civil Rights Act of 1964, and other applicable statutes. The University of Delaware has designated Karen Mancini, Director of the Office of Disabilities Support Services, as its ADA/Section 504 Coordinator under Federal law. Inquiries concerning Americans with Disabilities Act compliance, Section 504 compliance, campus accessibility, and related issues should be referred to Karen Mancini (302-831-4643) in the Office of Disabilities Support Services. Inquiries concerning Title VII and Title IX compliance and related issues should be referred to the Director of the Office of Equity and Inclusion, Becki Fogerty (302-831-8063).

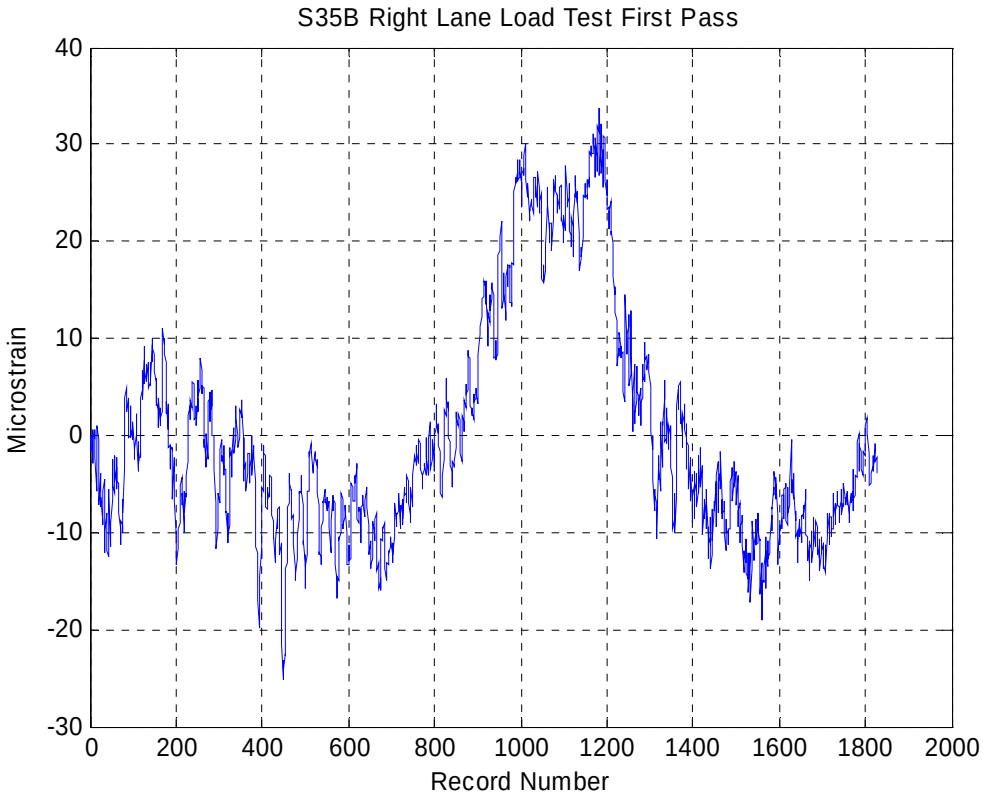


Figure C.1 First pass of right lane load test for strain gage S35B