SANDIA REPORT

SAND2003-8589
Unlimited Release
Printed September 2003

Smart Sensor Technology for Joint Test Assembly Flights

K. D. Marx, R. C. Armstrong, N. M. Berry, R. L. Bierbaum, T. J. Deyle,
J. L. Dimkoff, A. B. Doser, C. M. Pancerella, D. A. Sheaffer, and E. J. Walsh

Prepared by
Sandia National Laboratories
Albuquerque, New Mexico 87185 and Livermore, California 94550

Sandia is a multiprogram laboratory operated by Sandia Corporation,
a Lockheed Martin Company, for the United States Department of Energy's
National Nuclear Security Administration under Contract DE-AC04-94AL85000.

Sandia National Laboratories

SAND2003-8589
Unlimited Release
Printed September 2003

Smart Sensor Technology for Joint Test Assembly Flights

K. D. Marx*, R. C. Armstrong*, N. M. Berry*, R. L. Bierbaum*, T. J. Deyle†, J. L. Dimkoff*, A. B. Doser‡, C. M. Pancerella*, D. A. Sheaffer*, and E. J. Walsh*

Sandia National Laboratories
***Livermore, California**
and
‡Albuquerque, New Mexico

†University of Nebraska-Lincoln (Omaha Campus)

Abstract

The world relies on sensors to perform a variety of tasks from the mundane to sophisticated. Currently, processors associated with these sensors are sufficient only to handle rudimentary logic tasks. Though multiple sensors are often present in such devices, there is insufficient processing power for situational understanding. Until recently, no processors that met the electrical power constraints for embedded systems were powerful enough to perform sophisticated computations. Sandia performs many expensive tests using sensor arrays. Improving the efficacy, reliability and information content resulting from these sensor arrays is of critical importance. With the advent of powerful commodity processors for embedded use, a new opportunity to do just that has presented itself.

This report describes work completed under Laboratory-Directed Research and Development (LDRD) Project 26514, Task 1. The goal of the project was to demonstrate the feasibility of using embedded processors to increase the amount of useable information derived from sensor arrays while improving the believability of the data. The focus was on a system of importance to Sandia: Joint Test Assemblies for ICBM warheads. Topics discussed include: (1) two electromechanical systems to provide data, (2) sensors used to monitor those systems, (3) the processors that provide decision-making capability and data manipulation, (4) the use of artificial intelligence and other decision-making software, and (5) a computer model for the training of artificial intelligence software.

Contents

Figures

Executive Summary

The goal of this work was to show that intelligent embedded processors could be used to advantage to direct the flow of information from sensor arrays such as those used in Joint Test Assembly (JTA) flights. At the beginning of the project, it was decided that it would be too difficult and too intrusive to try to base the work on a real JTA system. Instead, two simple electromechanical systems instrumented with sensors were used as test beds to mimic JTA instrumentation.

The heart of a smart sensor system is the computer processor. In each of the two systems studied, the processor was chosen for characteristics that would make it potentially flyable on a JTA flight. These characteristics include small size, light weight, and an absence of moving parts such as rotating hard drives or fans. Each processor was equipped with a data acquisition (DAQ) board to acquire signals from the sensor arrays. No attempt was made to transmit signals over a wireless network. Instead, transmission was accomplished through Ethernet connections to desktop computers.

The first electromechanical system that was employed consisted of an aluminum bar mounted on a vibration shaker. The bar was instrumented with two strain gauges, temperature sensors, and a tachometer to monitor the frequency of vibration. This system was named the JTA Mockup. The sensor output was connected through the DAQ board to a 170 MHz PC104 processor with 130 MByte RAM and a 1 GByte flash disk. A version of the Linux operating system was installed on the processor. Data was taken from four sensors at the rate of 5120 samples/s/sensor (i.e., 20480 total samples/s).

The capability for recognizing different situations (and hence providing the opportunity to react to them) was demonstrated by taking fast Fourier transforms (FFTs) of the two streams of strain gauge data. This made it possible to determine the state of the JTA Mockup system and to detect the presence of obstacles impeding the motion of the two arms of the aluminum bar. Performing an FFT on two 512-sample blocks of data and sending a message to the desktop computer was done approximately 90 times/minute (1 ½ times/s). This represents a quantitative milestone for the project—demonstrating that a flyable processor running a full-up operating system could perform FFTs and report results at this frequency. No attempt was made to optimize this procedure; it is expected that the frequency of reports could be increased.

At this point, the potential for the use of artificial intelligence (AI) as part of a decision-making process was explored. The idea was that in a real JTA smart sensor system, AI modules could learn to (e.g.) determine system operating states, detect fundamental changes in sensor output, detect equipment failures and then take action to switch to different sensors, or to diagnose interesting features of the data and of the failures. The initial attempt to introduce AI involved the use of a type of neural network (NN) known as a self-organizing map (SOM). One of the key attributes of the SOM is that the various system states do not need to be defined *a priori*. Data recorded from the sensors were used to train a SOM, and the SOM then categorized the data. This

categorization was closely related to the determination of operating states referred to in connection with the use of the FFT results described above.

The training of the SOM was successful. When run on a desktop computer, the SOM was capable of recognizing operating states and reporting the results. It was recognized that the JTA Mockup did not represent a really good application of the SOM, as the amount of data (four sensors) was limited, and this type of AI is most useful when used to analyze more complex situations. However, the goal of the project was proof-of-principle, and this was achieved up to the point of training the SOM and running it against sensor data. What remained was to install the SOM on the JTA Mockup computer processor and to demonstrate that such an AI module could run on a flyable system. This effort was unsuccessful. The problem was that the SOM was written in the MATLAB language, with the intent that an advertised MATLAB feature could be used to convert the MATLAB code to C language code and run on the JTA Mockup processor. It was found that this conversion was not possible due to unanticipated idiosyncrasies of the MATLAB-produced C code. The project did not have the resources to install MATLAB on the processor or to rewrite the C code, so this avenue had to be abandoned.

It was then recognized that the computer processor described above was relatively slow and limited in memory and in flash disk space compared to processors currently (in 2003) available. It was also desired to make another attempt to demonstrate the utility of AI in smart sensor systems. So an 800 MHz PC104 processor with 500 MByte RAM and a 2 GByte flash disk was purchased. This processor was embedded in a new electromechanical system referred to as the geophone grid (GG) system. The GG system consisted of nine geophones arranged in a 3 x 3 grid on a floor. The geophones are the sensors in this system, and they generate electrical signals in response to floor vibrations. The idea was to use the GG system to detect and locate the occurrence of footsteps within the grid. Data was taken from the nine sensors at the rate of 10,000 samples/s/sensor (i.e. 90,000 total samples/s).

The analysis of the geophone data was done through the use of a neural net approach known as backpropagation (BPNN). This is a simpler type of NN than a SOM. A number of experiments were run with the GG/BPNN system which involved training BPNNs with data from both single geophones and from the 3 x 3 grid. The trained BPNNs were used to detect the presence of footsteps in the single geophone case, and to determine the location of footsteps in the 3 x 3 grid. The BPNN proved very capable in detecting a footstep via a single geophone sensor. However, the determination of location in the nine-sensor grid was somewhat limited due to lack of experimentation with preprocessing schemes and due to the lengthy required training times. Additionally, due to lack of time, the nine-sensor BPNN was run only on a desktop computer. However, the objective of demonstrating that the single-sensor AI system could be run on the PC104 processor was a complete success. Furthermore, there appears to be no reason that the nine-sensor BPNN would not have worked just as well on the embedded processor as on the desktop computer.

In summary, this project has explored the potential that embedded commodity processors have for enabling smart sensor systems. The principle has been demonstrated, and quantitative information has been made available from which to proceed with further development.

I. Introduction

The aim of this project was to demonstrate the utility of employing computer processors in an instrumented system to control the flow of sensor data. The idea is to embed one or more processors in an instrumented system and use software running on the processor to make intelligent decisions. The nature of these decisions is (1) to determine whether the data taken from sensors is accurate and reliable, and (2) in the presence of limited transmitter bandwidth, to decide which data to transmit. Although the primary focus of this work is on Joint Test Assembly (JTA) systems, the ideas are extendable to other systems that employ sensors.

For example, suppose that a particular subsystem is instrumented with two accelerometers, but that due to bandwidth restrictions, data from only one of these can be transmitted. Suppose that an intelligent system (the embedded processor and appropriate software) is capable of recognizing that data from the selected accelerometer is no longer reliable. This may be as simple as noting that there is no signal present above some minimum noise level. However, it could encompass a more complex situation in which the software is capable of recognizing that even though there is a strong signal from the accelerometer, the spectrum of the signal indicates that the accelerometer is no longer mounted securely. Given this information, the software could direct the instrumentation system to switch from the first accelerometer to the second. One can envision more complex situations where the software detects a problem in a critical subsystem, and switches transmitter resources from uninteresting data elsewhere to a previously dormant array of sensors designed to diagnose such problems.

The following approach was taken in this work:

(1) A simple electromechanical system was constructed and instrumented (see Section II). The purpose of the system was to provide some JTA-like data for processing. It has been given the name "JTA Mockup."

(2) A small, flyable computer processor was acquired and an appropriate operating system was installed on it (Section III). Data acquisition from the JTA Mockup to the processor was enabled through appropriate hardware and software.

(3) Software was written to process the data and make decisions based on its characteristics (Section IV). At first this software consisted of simple C code with conditional branches. Then a more sophisticated approach was implemented which utilized a self-organizing map (SOM), a type of artificial intelligence (AI) software (Section V). Laboratory data from the JTA Mockup was used to train the SOM to recognize various states of the system. (Note: Due to unforeseen circumstances, it was not possible to take the final step of running the SOM on the JTA Mockup processor. This led to Item (6) below.)

(4) A computer model of the JTA Mockup and the sensor system was developed (Section VI). The purpose of this model was to demonstrate that if a sufficiently accurate model

could be constructed, it could be used to train the SOM, thereby avoiding the necessity for extensive laboratory testing to perform this function.

(5) A second, more powerful, but still flyable processor was acquired and configured for smart sensor applications (Section VIII).

(6) A second type of AI module, a backpropagation neural net (BPNN), was tested on a different electromechanical system using the newly acquired processor (Sections VII, IX-XII).

II. Electromechanical System 1 (JTA Mockup)

The JTA Mockup is shown in cartoon form in Figure 1. A photograph is shown in Figure 2.

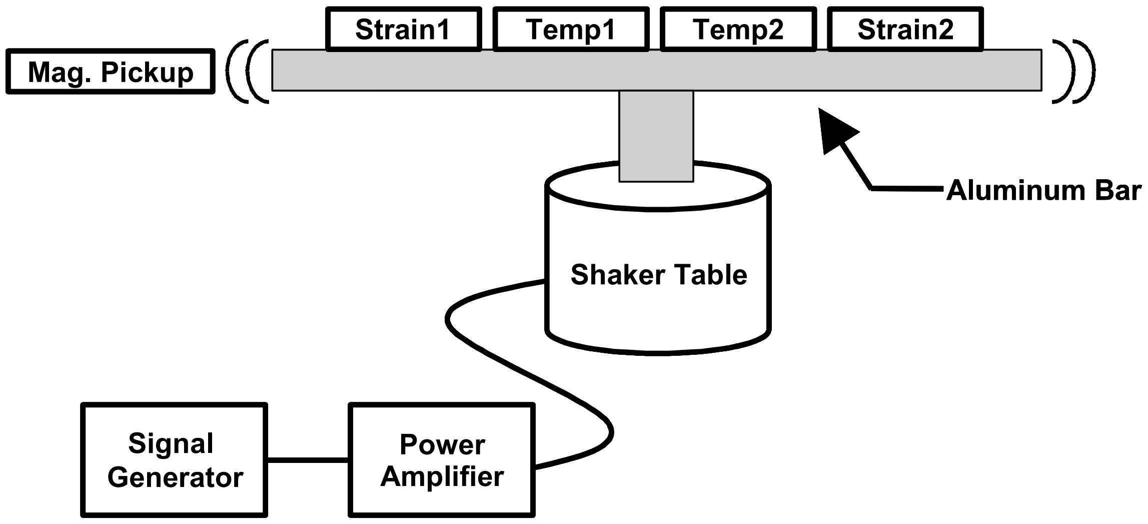


Figure 1. Simplified sketch of the JTA Mockup (not to scale).

The system consists of an instrumented aluminum bar mounted on a small shaker table. The dimensions of the bar are 0.5" x 1" x 24". The shaker table is driven by an audio power amplifier connected to a signal generator.[1] In the present work, the signal generator frequency has ranged from 40 Hz to 60 Hz, with the nominal operating frequency set at 50 Hz. The signal amplitude and amplifier gain have been adjusted for maximum shaker displacement consistent with operational safety and limited noise.

[1] The vibration shaker used was an Unholtz Dickie Model 5PM. The signal generator was a Tektronix SC 502, and the power amplifier was a Mackie FR Series M-1200.

As indicated in Figure 1, the sensors employed on the JTA Mockup consist of two strain gauges, two temperature sensors, and a magnetic pickup coil[2]. There are two separate Wheatstone bridge circuits of which each strain gauge forms one leg. Precision potentiometers permit the balancing of the bridges. The bridge circuits and temperature sensors are powered by 9 V and 5 V taps on a DC power supply.

The outputs from the sensors are sent to a signal conditioning board, which buffers all the signals, filters the strain gauge signals, and converts the magnetic pickup coil output to a voltage proportional to frequency (i.e., a tachometer signal). The output from the signal conditioning board is sent to the data acquisition board on the processor (see below).

It was intended that data from only one temperature sensor will be processed. Hence, a multiplexer was used to switch from one temperature sensor to the other. The

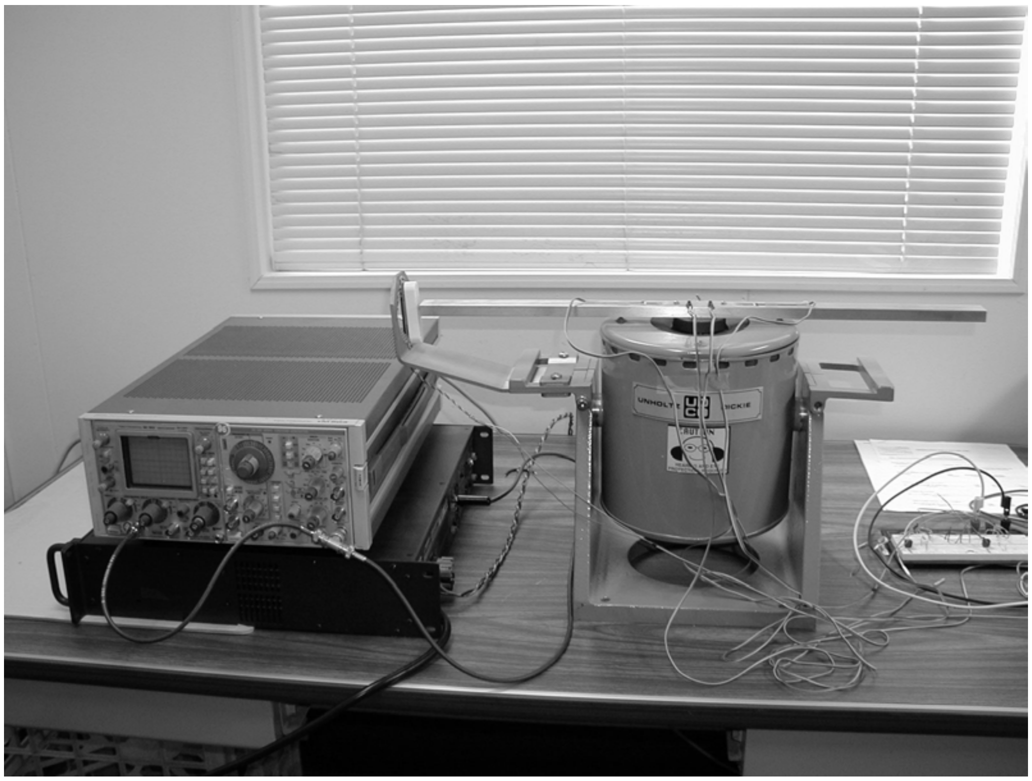


Figure 2. Photograph of the JTA Mockup system.

[2] The strain gauges were Micro-Measurements WK-06-125TQ-10C gauges, and the temperature sensors were National Semiconductor LM35 Precision Centigrade Temperature Sensors.

purpose of this arrangement was to be able to use feedback from the processor to do this switching when the sensor in current use failed. However, this feature was never implemented.

III. Embedded Processor 1

The computer processor used in the JTA Mockup system was a 170 MHz PC104 processor with 130 MByte RAM and a 1 GByte flash disk. The primary requirements for this application were (1) low power, and (2) the capability to employ a flash disk rather than a hard drive. The implementation of a hard drive would probably be impossible for a system embedded in a JTA flight because of its size and lack of ruggedness due to its inclusion of moving parts.

A version of the Linux operating system was installed on the processor. Signal output from the sensors was acquired by the processor via a data acquisition (DAQ) board which was included as part of the system.[3] Data was taken from four sensors at the rate of 5120 samples/s/sensor (i.e., 20480 total samples/s).

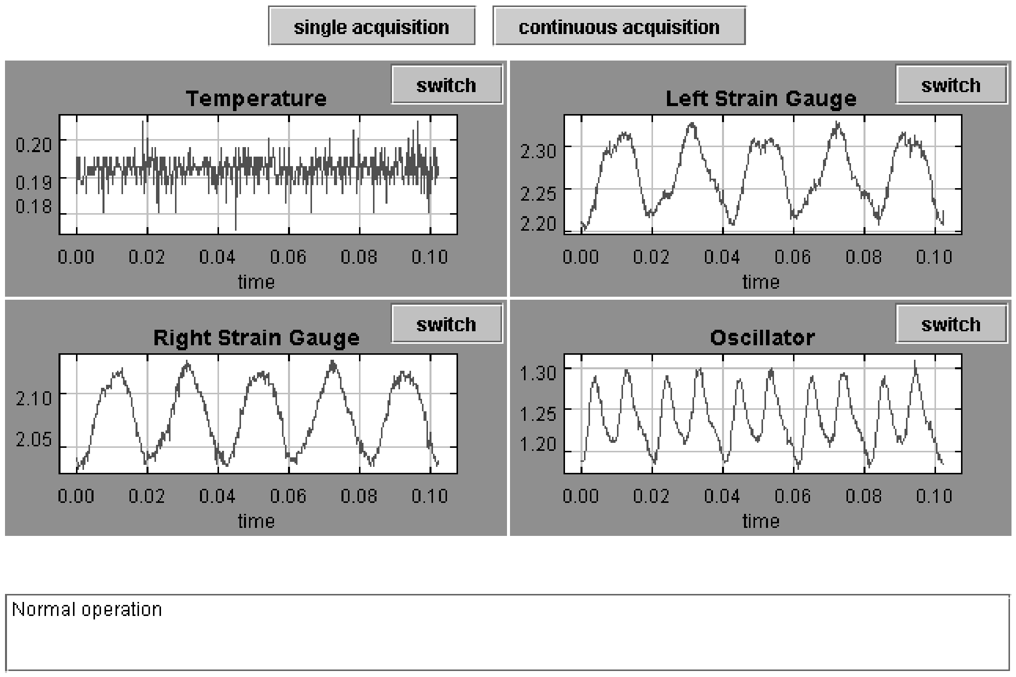
IV. Simple Operating State Recognition System

In this section, various operating states of the JTA Mockup system will be described, and output from the sensors will be shown. A simple computer program written in C and using conditional branches was used to detect the operating states. The program operates by interrogating the signals from the sensors to detect five different operating states, as follows: (1) Equipment not operating (power supply not on). (2) No power to shaker (no significant power delivered from the audio amplifier to the shaker, as determined from the power in the strain gauge data). (3) Normal operation (significant power delivered from the audio amplifier to the shaker). (4) Obstacle impeding the motion of the right arm of the aluminum bar (significant power in the higher harmonics of the strain gauge signals, and more such power in the signal from the strain gauge on the right than in that from the strain gauge on the left). (5) Obstacle impeding the motion of the left arm of the aluminum bar (using the converse of (4)).

Note that detection of states (2)-(5) involve taking a fast Fourier transform of the data from both strain gauges and making decisions based on the characteristics of the power spectral density obtained from the transforms. Performing an FFT on two 512-sample blocks of data and sending a message to the desktop computer was done approximately 90 times/minute (1 ½ times/s). This is one example of the type of sophisticated data processing and decision-making that is possible with a computer embedded in a sensor system.

[3] This processor will not be described further here, as the second processor used on the project is more powerful and more suitable for this application. See Figure 8 for a photograph of the second processor.

A graphical user interface (GUI) was written to display the sensor output and the detected operating state. The program and the GUI server code run on the embedded processor. The GUI returns the identification of the operating state to a desktop computer connected to the processor through an Ethernet connection to the Sandia SON network. An illustration of the GUI is given in Figure 3. In the case shown, the JTA Mockup system was in its normal operating state, running at 50 Hz. Note that the temperature sensor data and the tachometer data (denoted “Oscillator” in the figure) are time-varying signals with slowly varying DC components. The noise in the temperature data and the oscillatory signal in the tachometer data are averaged out so that only the slowly varying components remain when the data is processed. Also, no attempt was made in the GUI to convert the signals to true temperature, strain, or frequency.

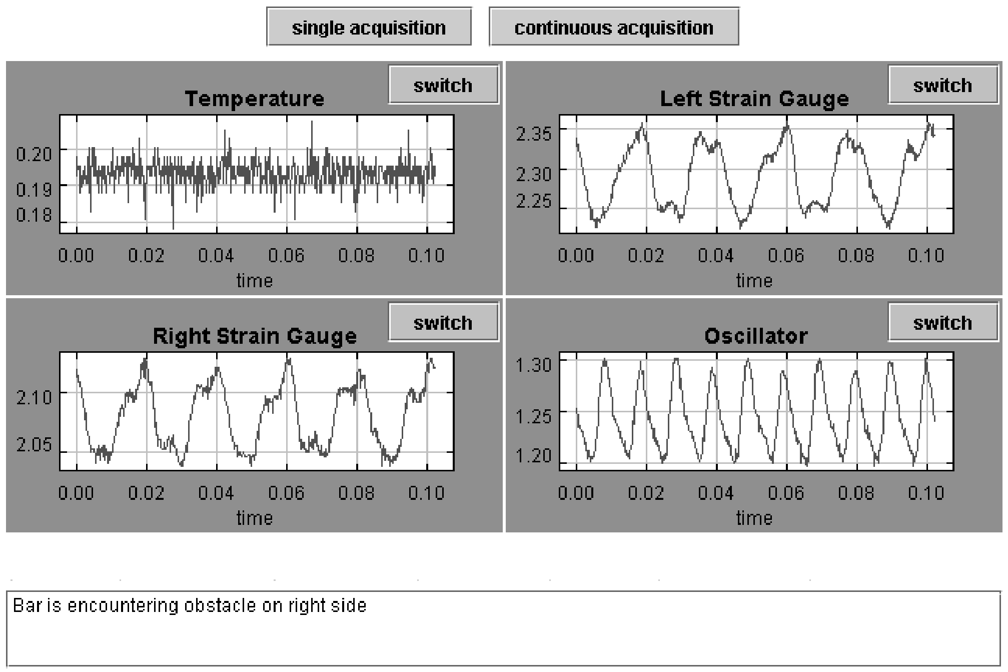


Normal operation

Figure 3. Output from the JTA Mockup GUI. As indicated in the lower left of the figure, the computer code resident on the embedded processor has detected that the system is in its normal operating state.

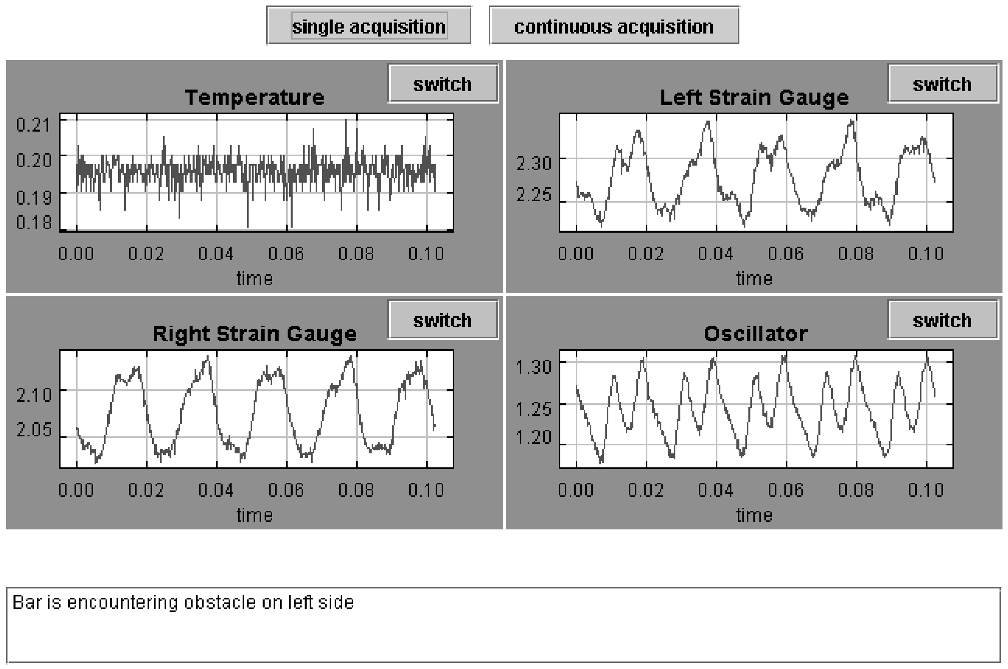
Examples of the cases in which obstacles impeded the vibration of the aluminum bar are given in Figures 4 and 5.

Recall from the discussion above that the detection of obstacles depends on a determination of the power in the higher harmonics of the signal relative to the fundamental. It can be seen that there is a significant amount of power in higher



```
Bar is encountering obstacle on right side
```

Figure 4. Output from the JTA Mockup GUI showing the response to the situation in which an obstacle is impeding the motion of the right side of the bar.



```
Bar is encountering obstacle on left side
```

Figure 5. Output from the JTA Mockup GUI showing the response to the situation in which an obstacle is impeding the motion of the left side of the bar.

harmonics in the “Normal operation” state shown in Figure 3. It is believed that this is due to the close proximity of the embedded processor power supply to the sensor signal lines going into the DAQ board. In any case, it is worth noting that it would not be not an easy matter for a human observer to quickly distinguish between “Normal operation” and “Bar is encountering obstacle…” in Figures 4 and 5. This illustrates the potential for the use of embedded processors in smart sensor applications.

V. Use of a Self-Organizing Map to Classify the Data

In this section, the introduction of artificial intelligence into the JTA Mockup system will be described. As noted above, the data available from the JTA testbed included measurements from two strain gauges, one temperature sensor and a tachometer. In order to classify the data, a method was developed, based partially on self-organizing maps (SOMs). See, e.g., Kohonen (2001). Since they have the advantage of compressing several dimensions of data into two or three dimensions, while still preserving the distance relationships present in the original data set, SOMs were useful in this work. A SOM is a variant of a neural network, but works on the principle of “unsupervised training”, meaning that it trains itself with no particular regard to what the output should be, allowing the map to explore the relationships among the data points free from any biases. After a map is produced, the programmer can designate any labels to the output that he or she wishes.

The aim was to use SOMs to determine whether the JTA Mockup system was performing properly, and if not, to pinpoint what the problem in the assembly might be. In order to use a SOM, a number of data files must be presented to the network in order to train to a solution. Ideally, the training data should include many examples of all the anomalies one wishes the network to classify. In addition to normal function, the training data provided the following examples of anomalies: no power to shaker, dead temperature sensor, increase in system frequency, decrease in system frequency, shaker obstruction. In this work, the training data that was available was divided into files of 512 time samples each. With respect to the strain gauges, the critical information was contained in the frequency content of the data. Keeping this in mind, an FFT was taken of the strain gauge data before it was handed off to the SOM for training. In addition, the DC component of the frequency data was irrelevant and was subtracted. Only the first 25 frequency bins were used for each strain gauge, as the higher frequency data had been found experimentally to contain no useful information. In the case of the temperature and tachometer measurements, an average value was taken for each parameter over the course of 512 time samples. Thus each training file presented to the SOM network consisted of 25 frequency bins for each of the two strain gauges, the average temperature, and the average tachometer reading, for a total of 52 parameters.

The software used for training the SOM was available, without licensing restrictions, from Helsinki University of Technology. The SOM package was written for MATLAB, and was free to download from the Helsinki University of Technology internet site. Five hundred data files, containing various anomalies, were available for the training and testing of the algorithm. One fifth of the files were selected for training purposes, and were presented to the SOM software. SOMs use unsupervised training techniques, which means that they create a mapping without any guidance of the user desired outcome. The dimensionality of the SOM was selected based on the eigenvalues of the training data. In this case, the SOM was a 12x8-dimensional matrix. The output of the SOM does not contain any information regarding the classification of the data. In order to place membership on the data, postprocessing is required. The classic K-means clustering algorithm was used for this purpose. It was assumed the data consisted of 7 classes. Initial centroids were chosen at random from the map coordinates.

Upon first trial of this method, it was found the algorithm produced mixed results. While it correctly identified the cases where there was no power, dead temperature sensor, and obstruction, it incorrectly identified files with frequency changes as “normal”. Frequency corrupted files can be identified by a shift in a high frequency component. Since the magnitude of the component was small compared to the main, low frequency spike, the SOM tended to ignore the high frequency component when it created its mapping. However, the frequency shift is predictable in these files. The SOM need only be forced to pay attention to the high frequency spike. This outcome was achieved by boosting the FFT magnitude at the location of the high frequency spike. This scheme produced the desired results.

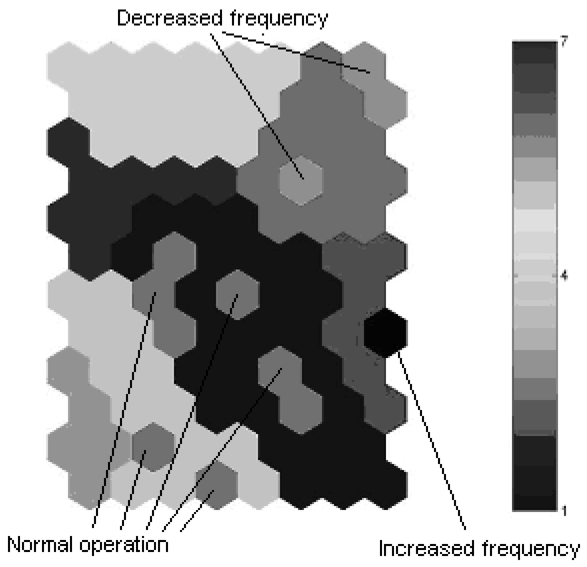


Figure 6. Partial output of the clustered SOM.

Figure 6 represents the partial output of the clustered SOM. The classes are coded with different shading in the large regions, and locations of test files are superimposed as individual cells. In order to reduce confusion with too many shades included in the plot, only half the test cases are plotted here: normal operational files, increased frequency, decreased frequency. Since a SOM compacts a feature space into fewer dimensions, it is necessarily a many-to-one mapping. There is no way to discern which files produced a given outcome, as one shaded cell on the map represents the output of several files. Note how the frequency shifted files are clearly delineated from each other, as well as the normal files. Also note that the normal files fall into two separate classes. The artifact is not seen as undesirable, however, as no anomalies fall within the two normal classes, and the user could always choose to do an off-line merge of the normal classes.

Figure 7 also represents the output of the clustered SOM, but in this case, different anomalies are plotted: no power to system, dead temperature sensor, obstruction. Not only do these anomalies all fall into separate classes, but a comparison with the previous figure demonstrates that they are separable from the other test cases as well.

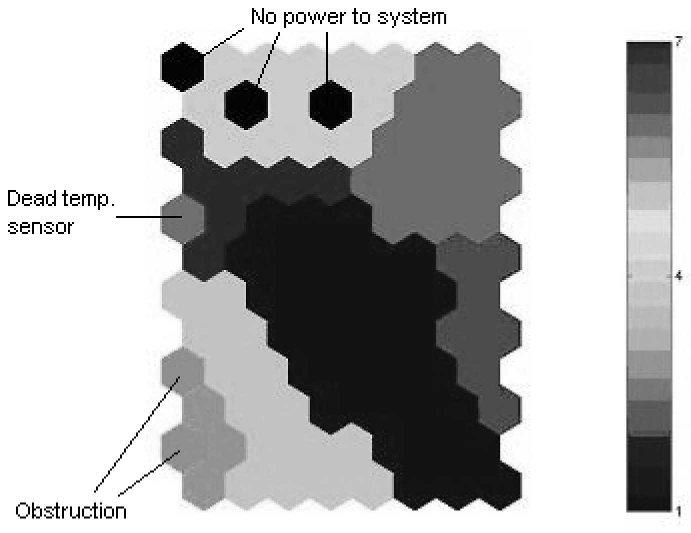


Figure 7. Partial output of the clustered SOM with different anomalies.

The graphing displayed above was done for ease of viewing. In the implemented algorithm, the output is a single integer number, between 1 and 7, which designates the class membership of a given test file. This format of output was better suited for near real-time applications

Unlike a rules-based approach, the hybrid SOM presented here has the advantage that it can be retrained as new anomalies or significant changes in the system are observed, without the costly expense of creating new rules. In this application, the training of the SOM required less than 10 seconds of off-line training on a Pentium 4 laptop.

What remained in the implementation of the SOM was to install it on the JTA Mockup computer processor and demonstrate that such an AI module could run on a flyable system. This effort was unsuccessful. The problem was that the SOM was written in the MATLAB language, with the intent that an advertised MATLAB feature could be used to convert the MATLAB code to C language code and run on the JTA Mockup processor. It was found that this conversion was not possible due to unanticipated idiosyncrasies of the MATLAB-produced C code. The project did not have the resources to install MATLAB on the processor or to rewrite the C code, so this avenue had to be abandoned.

VI. Computer model of the JTA Mockup system

One of the goals of this project was to demonstrate that AI modules in a smart sensor system could be trained with data from a computer model. The idea was to develop a computer model of the electromechanical system (JTA Mockup in this case) and the sensor system itself. The use of output from such a model for training is clearly limited by the accuracy of the model and any failure to include important phenomena.

Such a model of the JTA Mockup was developed using the PSpice electrical simulation code. Large quantities of data were taken to characterize the sensors in order to build their respective models. The temperature sensor and tachometer models were strictly behavioral models; they effectively modeled sensor output without including underlying physics principles. Due to the limited variation in the output signal for these two sensors, constructing a behavioral model was straightforward.

The strain gauge sensors generate more complex and varied waveforms. Discrimination of these waveforms can yield information on the location and composition of the obstacle. Information could be extracted from the signal spectrum (both amplitude and phase), the DC offset, and the amplitude of the signal. By comparing these attributes of the two strain gauges, it was demonstrated that the location of obstructions could be determined. In addition to determining on which arm obstructions were located (left or right), the side of the bar (top, bottom, or side) could be identified as well. Additionally, the displacement along the bar could be determined, if only with limited accuracy. It was also possible to determine the existence of two obstructions under certain conditions, but location information is not discernable with only two sensors.

It was also noted that the information as to the mass and hardness of the obstruction could be found. Harder objects tend to generate more high harmonics than soft objects. Further investigation of this was not pursued. Instead, a consistent hard obstruction was used to eliminate this variable.

Building a behavioral model to cover all the above operating conditions was a formidable challenge. It was decided to limit the strain sensor model to the generation of a generic (left or right) obstruction due to limited time. This shortage of time and resources ultimately prevented a demonstration of the utility of the use of model output for training.

VII. Electromechanical System 2 (Geophone Grid)

At this point, it was recognized that the computer processor described above was relatively slow and limited both in memory and in flash disk space compared to processors currently (in 2003) available. It was also desired to make another attempt to demonstrate the utility of AI in smart sensor systems. The following sections describe a phase of the project in which these two aspects were addressed by embedding a new processor in a new electromechanical system referred to as the geophone grid (GG) system.

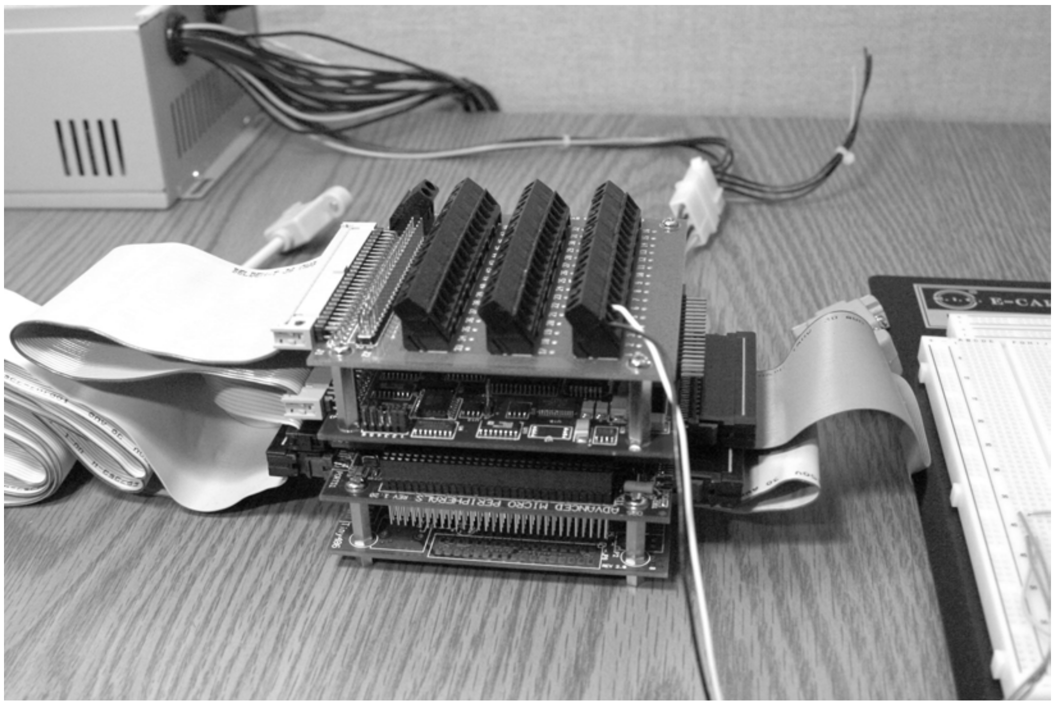


Figure 8. Photograph of the processor board assembly. The top board is a breakout board to route the sensor signals to the DAQ board. The remaining three boards are (from second to bottom) the DAQ board, the processor board containing the Crusoe chip, and a power base board. Only one sensor connection is made in the figure (see the two wires connected to the breakout board). In the actual GG configuration, nine connections were made.

VIII. Embedded Processor 2

The computer processor used in the GG system is a Tiny886ULP (Ultra Low Power) PC/104+ Crusoe Computer from Advanced Microperipherals Ltd. The Crusoe chip used contained 500 MByte of memory and used a 2 GByte flash disk. Its CPU speed was 800 MHz. A version of the GNU/Linux operating system was configured on a separate Linux computer and then copied to the flash disk. A 200 kHz data acquisition (DAQ) board from Diamond Systems Corporation was included as part of the system. ld was set at from 500 to 1500. The processor board assembly is shown in Figure 8.

As an aside, it should be noted that the DAQ board has some features that were not utilized in this work, but may offer interesting opportunities for the implementation of embedded processors in the future. The board provides for digital inputs to reconfigure the board "on the fly". One can change the gain of each analog input channel as well as the sensor sampling scheme (i.e., the channels to be sampled and the sampling rate). These capabilities offer a more robust set of responses to the decision-making algorithm.

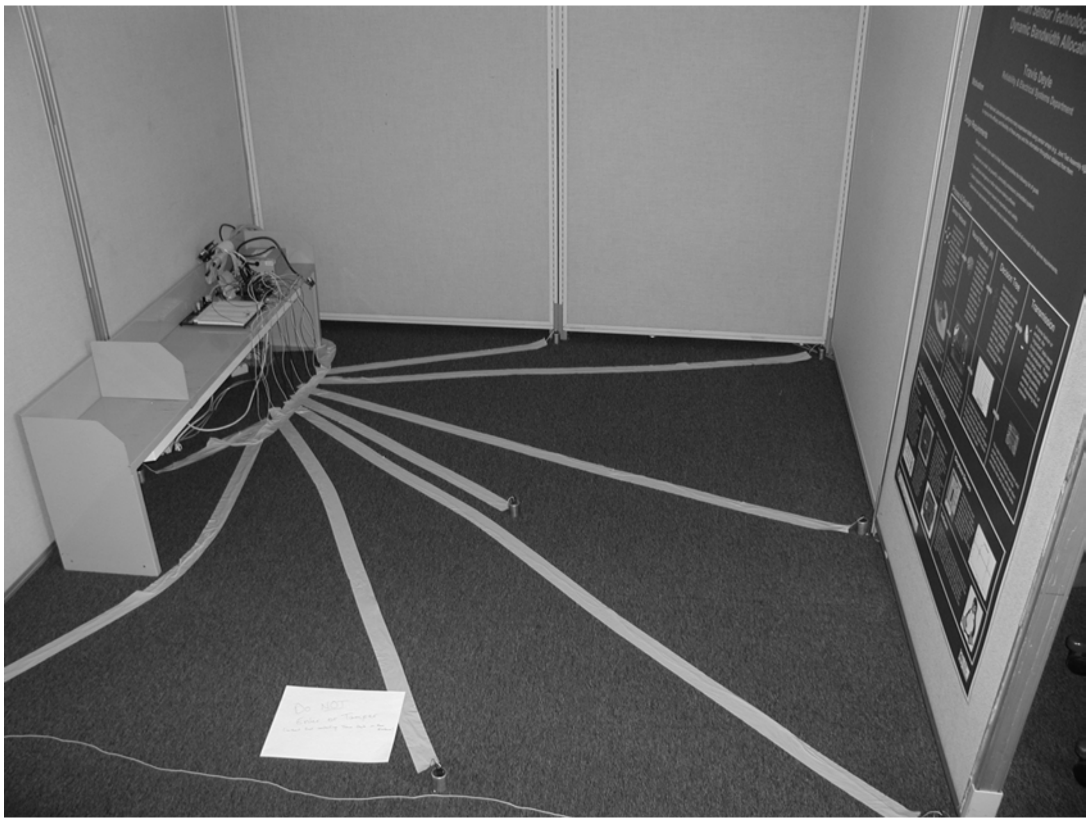


Figure 9. Geophone sensor array. The geophones are laid out in a 3 x 3 grid.

IX. Sensor Array for Backpropagation Neural Net Data

The sensor array used to take data for the neural nets consists of 9 Model MD-81 Geophones manufactured by Geosource Inc. The geophones were arranged in a 3 by 3 square array with a nearest-neighbor separation of from 39 to 47 inches (see Figure 9). Each geophone was connected to the DAQ board so that signals could be read into the processor.

The geophones were excited by stepping on the floor at arbitrary locations within the 3 by 3 grid. Typical data from one geophone is shown in Figure 10. When excited by a footstep, the signals from the geophones rang at frequencies of the order of 25-100 Hz.

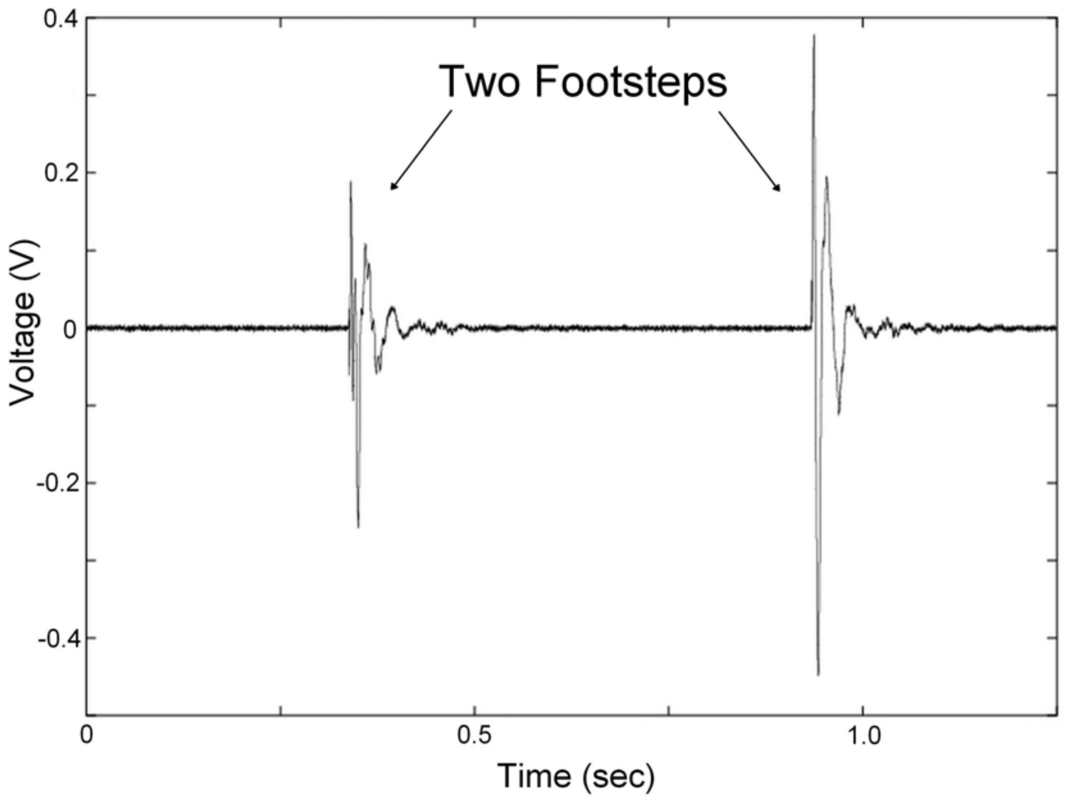


Figure 10. Typical data from a single geophone.

X. Backpropagation Neural Net Procedures

The BPNNs used in this work were enabled through the use of routines in a library obtained from the website http://ieee.uow.edu.au/~daniel/software/libneural/. This library contains code for creating a three-layer backpropagation neural network. The general features and use of such a neural net is described in Eberhart and Dobbins

(1990) and in Welstead (1994). This material will not be repeated here; the following sections offer some of the details of our particular application to the identification of footsteps within the sensor grid described above. It should be noted that extensive coding in C and C++ was required to adapt the library routines to the present problem.

XI. Single-Channel Backpropagation Neural Net

The use of a backpropagation neural net to process input from just one sensor of the nine-sensor array will be discussed first. The single-channel BPNN is shown in Figure 11. This BPNN has one input, corresponding to the single channel of geophone

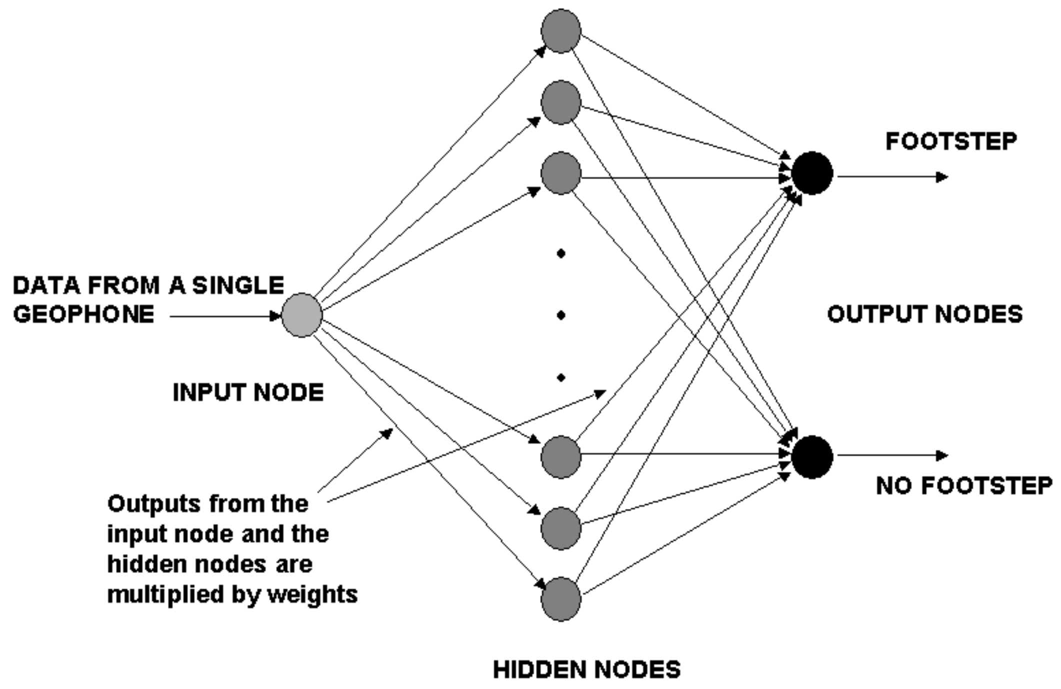

FOOTSTEP
DATA FROM A SINGLE GEOPHONE
OUTPUT NODES
INPUT NODE
Outputs from the input node and the hidden nodes are multiplied by weights
NO FOOTSTEP
HIDDEN NODES

Figure 11. Single-channel backpropagation neural net. There are 30 hidden nodes in this BPNN. The nine-channel BPNN contains 9 input nodes, 60 hidden nodes and 5 output nodes.

data. It has two outputs, corresponding to detection of a footstep or of no footstep. These quantities are determined by the configuration being studied, i.e., the single sensor with one of two outcomes. It remains to configure the hidden nodes. In general, the number of layers of hidden nodes in a neural net and the number of nodes in each layer is usually chosen to optimize the accuracy of the net with a view to limiting the amount of time required to train the net and to run it. In this project, a single layer with 30 hidden nodes was used in the single-channel BPNN.

The first step is to obtain a data file from a sensor. This is a single-column file with the raw-signal data acquired from the DAQ board. Each line of the file is a sample of geophone data at a single point in time. Coming from the DAQ board, it is not in any particular units; it was not necessary to convert it to actual voltage for our purposes. The values of the signal ranged from about -5000 to +5000 in arbitrary units. A program converts this file to a new file, referred to as a training file, in which a value of 0 or 1 was appended to each data point, according to whether that data point was classified as representing a footstep or not. The means of classification are discussed in detail below. It is important to note that the inputs to neural nets in general must usually be normalized in order that the nets work properly (Eberhart and Dobbins, 1990). In the present case, the absolute values of the geophone output (between 0 and 5000 in terms of raw values from the DAQ board) were normalized to between 0 and 1.

The classification process was complicated by the fact that the footstep waveforms have zero-crossings in them due to the ringing of the geophone output. Hence the state of "no footstep" could be viewed as occurring intermittently while a footstep was actually being recorded. Two different means of classifying the state of "footstep" or "no footstep" were explored. The first used a threshold value to classify each data point in the waveform (i.e., the presence of a footstep was attributed to signals with absolute values greater than some threshold which was set at 500 units). This meant that the zero-crossings that occurred during a footstep signal were actually classified during training as "no footstep". While making the training faster to accomplish, it seemed to belie the objective of using AI to discriminate complex signal behavior by incorporating the use of a predetermined threshold. Due to the time limitations of the project, this was the predominant method of training used.

The second training approach examined used an enveloping process to define the state, where the entire duration of the footstep (irrespective of whether it was positive, negative, or zero) was categorized as "footstep". As expected, this approach required much more training time because essentially contradictory information was being given vis-à-vis what constitutes a footstep. However, this approach was also effective in detecting footsteps.

Once the training file was developed using one of the approaches above, the BPNN was then trained. The output from this training procedure is a file containing the weights for the BPNN. The term "weights" refers to the factors by which the output from the input node and the outputs from the hidden nodes are multiplied to obtain the inputs to the downstream nodes (see Figure 11).

The training procedure to generate the weights file is a lengthy process. It typically involves many iterations of passing the training data through the BPNN. However, this iterative procedure need be done only once. Training is carried out on a fast desktop computer (it could be done on a supercomputer if necessary) to expedite the computation.

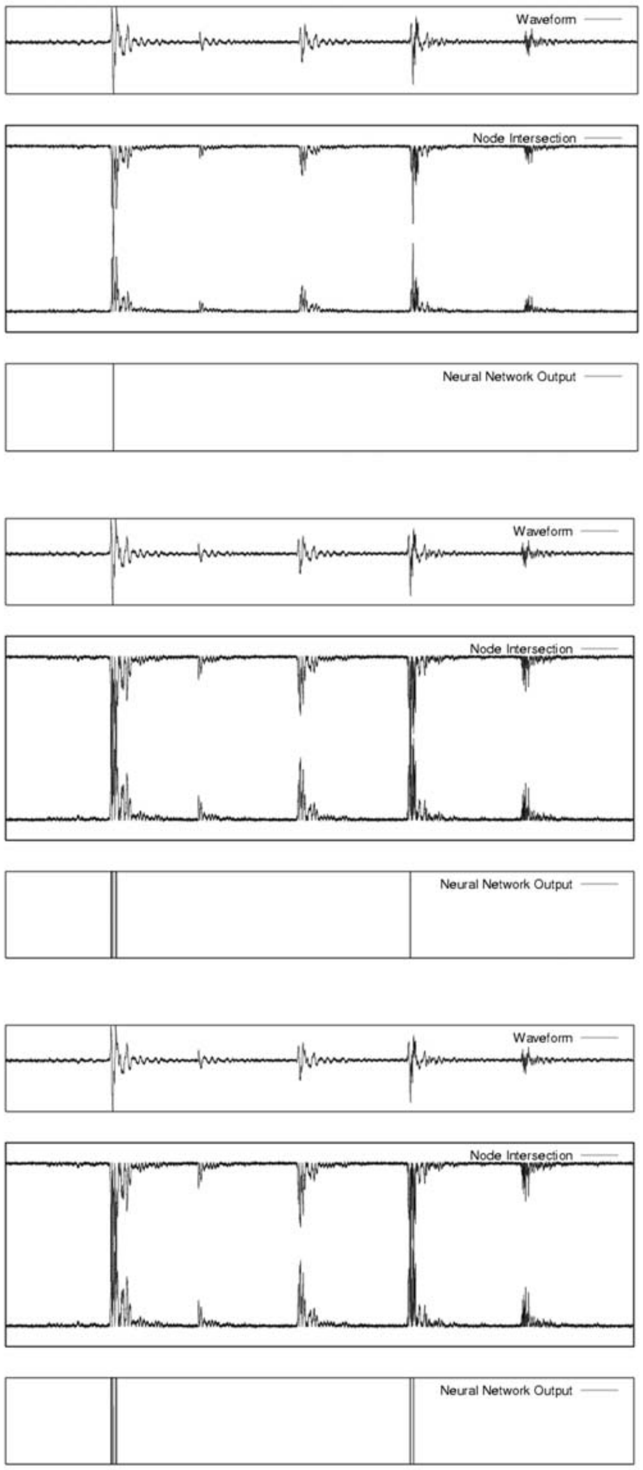
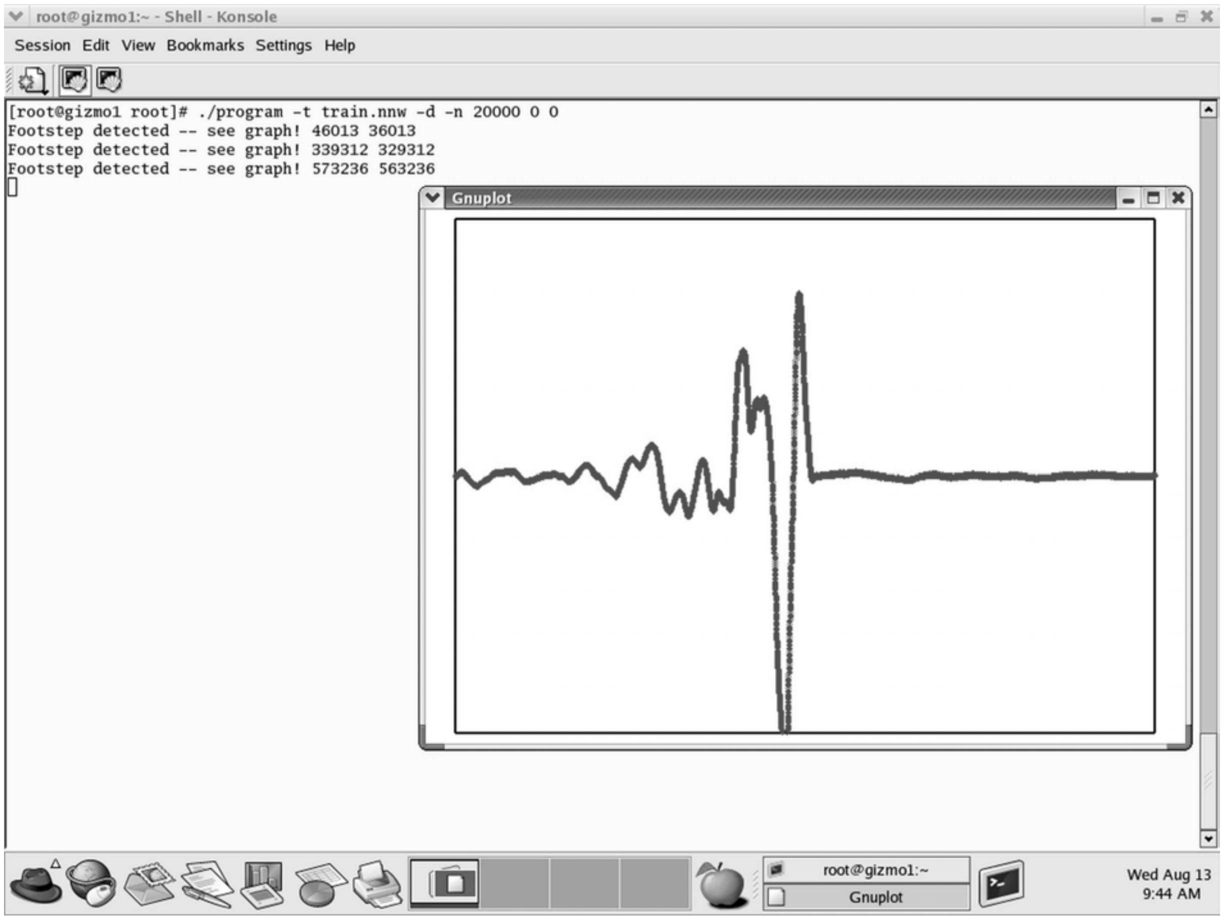


Figure 12. Results of training the single-sensor BPNN 3 times (top 3 plots), 7 times (middle 3 plots), and 10 times (bottom 3 plots). Note the progressive improvement in the ability of the neural net to correctly classify footsteps.

Figure 12 shows the results of training the BPNN 3 times, 7 times, and 10 times, with data from a single geophone. The first ("threshold") method of training described above was used to classify the states in this case. In each set of three plots, the first plot shows the data used for training (the same for all three training runs). The second plot shows the output from the two output nodes. The upper trace in this plot shows the "No Footstep" results, and the lower trace shows the "Footstep" results (see Figure 11). The algorithm specifies that a footstep has occurred when the two traces intersect. These occurrences are shown in the third plot of each set. Note the way in which the number of intersections, i.e., the detection of a footstep increases with the number of training runs. This demonstrates that the ability of the BPNN to obtain the desired result improves as the number of training runs increases.


root@gizmo1:~ - Shell - Konsole
Session Edit View Bookmarks Settings Help
Gnuplot
root@gizmo1:~
Gnuplot
Wed Aug 13
9:44 AM

```
[root@gizmo1 root]# ./program -t train.nnw -d -n 20000 0 0
Footstep detected -- see graph! 46013 36013
Footstep detected -- see graph! 339312 329312
Footstep detected -- see graph! 573236 563236
```

Figure 13. Screenshot made while running the single-channel neural net on the embedded processor. Note: Time runs backwards on the plot. I.e., the earliest time is on the right. The text in the upper left part of the screen flags those occasions when the net has detected a footstep.

Once the weights file is created, it is copied to the embedded processor. There, the BPNN is run on live data from one of the geophone sensors. An example of this is shown in Figure 13. Note that the data were sent via an Ethernet connection from the embedded processor to a "remote" display, emulating the transmission of JTA data to a ground station.

This simple example illustrated several important concepts for the project:

- Real-time sensor data streams can be captured and analyzed "on-the-fly" using AI with an embedded processor.
- One can determine states using AI and make decisions based upon those states—in this case, data were sent to and viewed on the remote display only when the state of "footstep" was detected.
- The notion of only sending data when it is deemed "interesting" was demonstrated.

XII. Nine-Channel Neural Net

Now, the detection of a single footstep based on a geophone signal exceeding some threshold is an admittedly trivial example of the application of artificial intelligence. The same thing could be accomplished in much less time with a simpler computer code and an empirically adjusted threshold. However, the BPNN example described above accomplishes the goal of demonstrating that such software can be easily run on an embedded processor. It also provides useful guidance as to how to proceed with more complex examples. Such an example will now be presented.

Recall the geophone grid shown in Figure 9. Typical data from all nine sensors are shown in Figure 14. Data was taken from the nine sensors at the rate of 10,000 samples/s/sensor (i.e. 90,000 total samples/s). In this full GG case, the BPNN contains 9 input nodes corresponding to the 9 sensors. It has 5 output nodes corresponding to the cases of footsteps in any of the 4 quadrants formed by the 3 x 3 grid of sensors, and to the case of no footstep. Again, a single layer of hidden nodes was used. For the nine-sensor case, this layer contained 60 nodes. The training of the BPNN was similar to the single-sensor case, except that training was done with data taken from footsteps made at 36 different locations within the grid.

An example of nine-sensor BPNN output is shown in Figure 15. This example is for a footstep in quadrant 3 (the quadrant encompassed by Geophones 3, 4, 6, and 7). The scale of the neural net output in Figure 15 is from 0 to 4, and the output appears in unit increments (0, 1, 2, 3, 4). For a footstep in Quadrant 3, the output should read only 3 during the footstep. Note that some false Quadrant 2 and 4 readings appear in the figure. These false readings are believed due to the following: (1) complications involved in the definition of what constitutes a footstep (see discussion in Section XI), (2) lack of sufficient preprocessing of the complex geophone signals to enable the BPNN to

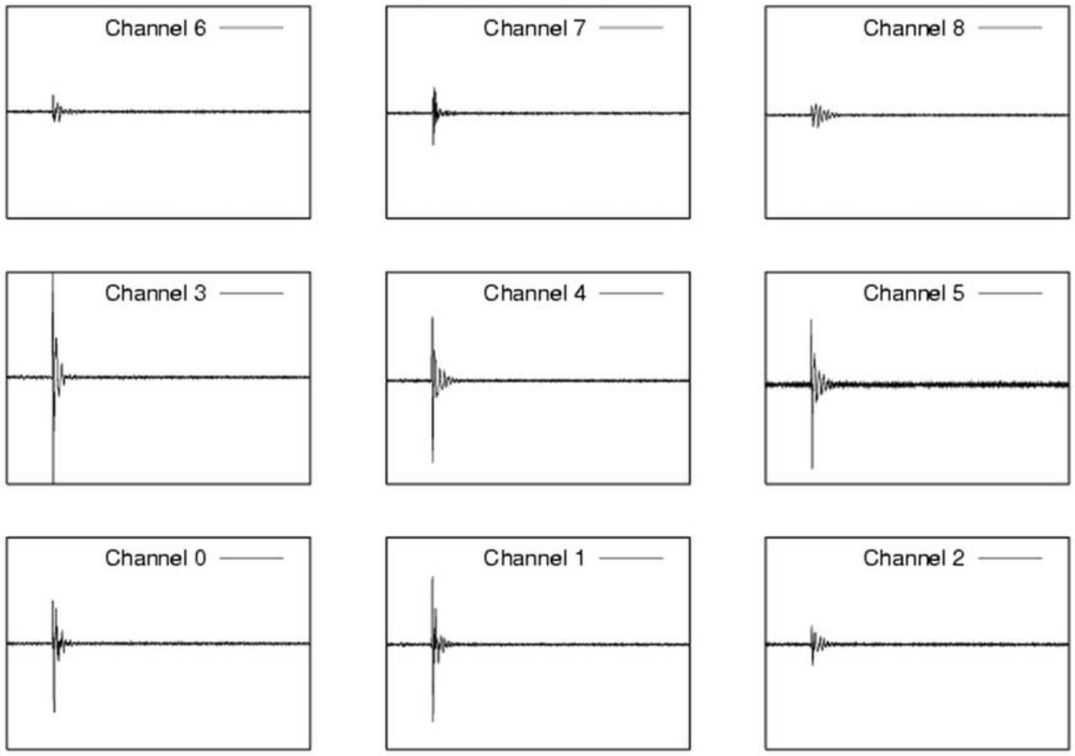


Figure 14. Typical data from the nine-geophone grid. The footstep was in the center of the first quadrant (the quadrant encompassed by Geophones 0, 1, 3, and 4).

converge more robustly to an acceptable solution, and (3) insufficient training time. The training challenge is exacerbated in the nine-sensor case because footstep location is now to be determined by the relative magnitudes of nine data points, each of which is a sample from a ringing waveform. If time had allowed, the use of low-pass filtering on the footstep waveforms would have been explored in order to smooth out the ringing and perhaps make the footstep definition/location and subsequent training more straightforward.

XIII. Conclusions

During the course of this LDRD, numerous technical opportunities and challenges were explored through the use of two sensor arrays. These and some ideas for future work are highlighted below.

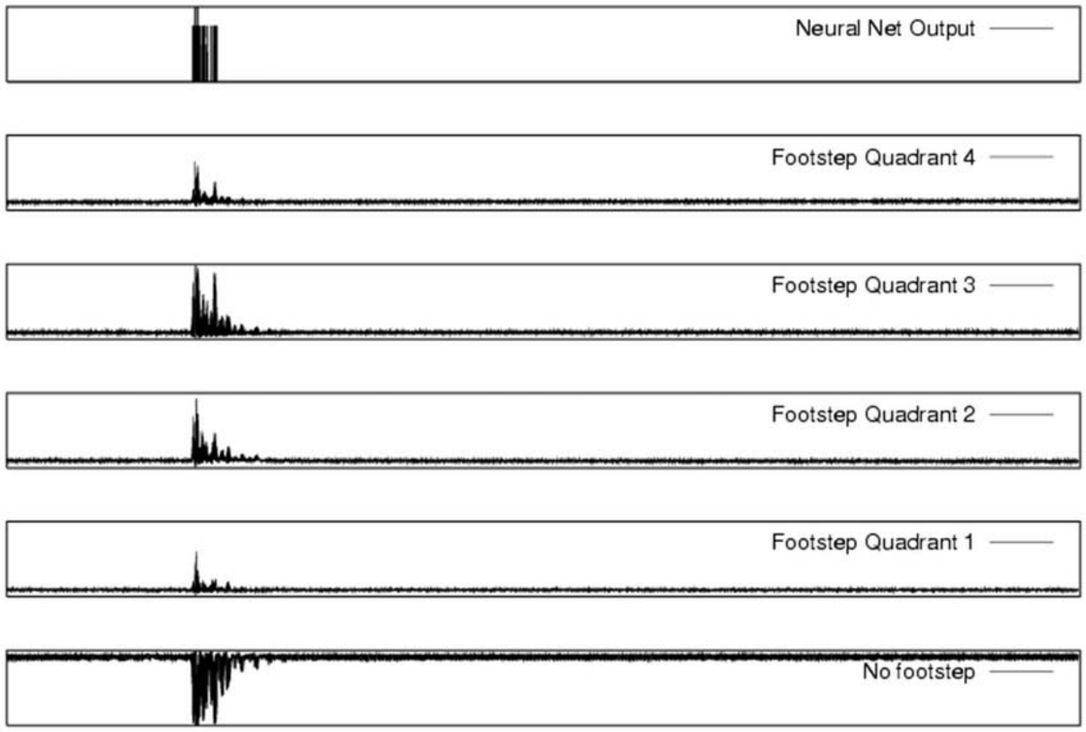


Figure 15. Nine-sensor backpropagation neural net output.

Technical Opportunities

- The use of real-time AI and subsequent decision-making was clearly demonstrated.
- Commodity embedded processors that meet the stringent volume and power constraints for JTAs appear to have the potential to support these applications.
- Two different sensor systems (each generating real-time data streams with bandwidth commensurate to that needed for JTAs) were successfully developed.

Technical Challenges

- Use of either SOMs or BPNNs require generation of training data that is complete in the sense of encompassing all possible normal and anomalous behaviors that might be observed.
- Training of the neural network for the Geophone Grid was problematic due to the nature of the ringing signal. Preprocessing of signals may be necessary to ensure unambiguous state determination.
- One of the major technical hurdles in this project for both sensor systems was installation of the operating system on the embedded processor. This involved extracting the needed elements from a robust Linux kernel, recompiling, loading, and

making it bootable on the embedded processor. Although conceptually straightforward, it proved to be a time-consuming task. This should become easier as processors with increased capability become available.

Future work

- Utilize more JTA-like sensors and signals, with attention to JTA bandwidths.
- Work to overcome the challenges presented by both types of artificial intelligence modules and procedures.
- In the actual application of the type of smart sensor system described here, risks will be incurred—specifically, there will be the risk of making incorrect decisions and sending the wrong data. This implies that before entrusting JTA data to such a system, a careful risk analysis should be carried out.

Acknowledgments

We would like to thank the following for useful discussions and for their support during the completion of this project: P. E. Nielan, K. E. Washington, D. R. Henson, K. R. Hughes, E. P. Parker, J. S. Wagner, P. W. Dean, M. L. Koszykowski, D. E. Rhodes, and S. R. Parker. Many thanks go to J. F. Stamps for providing a crucial fix to the signal conditioning board.

References

Eberhart, R. C., and Dobbins, R. W., "Neural Network PC Tools: A Practical Guide," Academic Press, San Diego (1990).

Kohonen, T., "Self-Organizing Maps," Springer, Berlin (2001).

Welstead, S. T., "Neural Network and Fuzzy Logic Applications in C/C++," Wiley, New York (1994).

Distribution

1	T. J. Deyle 6241 S. 170th St. Omaha, NE 68135	
1	MS 0323	LDRD Office (D. L. Chavez, 1011)
1	MS 1188	J. S. Wagner, 15311
1	MS 1188	A. B. Doser, 15311
1	MS 1188	E. P. Parker, 15311
1	MS 9003	K. E. Washington, 8900
1	MS 9003	J. L. Handrock, 8960
1	MS 9007	D. R. Henson, 8200
1	MS 9012	P. E. Nielan, 8964
1	MS 9012	C. M. Pancerella, 8964
1	MS 9012	E. J. Walsh, 8964
1	MS 9013	R. E. Oetken, 8231
1	MS 9013	D. A. Sheaffer, 8231
1	MS 9036	W. G. Wilson, 8230
1	MS 9102	A. L. Hull, 8233
1	MS 9102	P. Y. Yoon, 8235
1	MS 9103	J. F. Stamps, 8111
1	MS 9202	K. R. Hughes, 8205
1	MS 9202	R. L. Bierbaum, 8205
1	MS 9202	S. L. Brandon, 8205
1	MS 9202	J. L. Dimkoff, 8205
10	MS 9202	K. D. Marx, 8205
1	MS 9915	M. L. Koszykowski, 8961
1	MS 9915	R. C. Armstrong, 8961
1	MS 9915	N. M. Berry, 8964
3	MS 9018	Central Technical Files, 8945-1
1	MS 0899	Technical Library, 9616
1	MS 9021	Classification Office, 8511 for Technical Library, MS 0899, 9616 DOE/OSTI via URL