95-FR2

INTERNATIONAL ELECTRONIC MACHINES CORPORATION

Real Time Hunting Mitigation

Sensor Systems to Detect Hazardous Conditions in High Speed Rail Operations

Phase I Report

Zahid F. Mian, President
D. Richard Arthur, V.P.
International Electronic Machines Corp.

July, 1996

60 Fourth Avenue, Albany, New York 12202-1924
(518) 449-5504 FAX (518) 449-5567

ACKNOWLEDGMENTS

The authors wish to acknowledge Dr. Herb Wienstock of The Volpe Center, Dr. George List of Rensselaer Polytechnic Institute, Mr. Stan Canjea of New Jersey Transit, Mr. Steve Shapiro of The Budd Corporation, Mr. Don Bolt of TTX Corporation, and Mr. Bob McCowan of the FRA for their support in this project.

Table of Contents

Executive Summary

With the advances in train speeds coupled with lighter weight cars, the problem of truck hunting is expected to grow. Even though modern truck designs are performing excellently in raising the critical speed where hunting occurs, no truck design can eliminate the tendency to hunt altogether. Also, as components wear, the critical speeds drop; often in unpredictable ways. In order to ensure safe high speed rail operations, a new technology is required to continuously monitor the performance of the train cars to alert the engineer if hunting is occurring so appropriate remedial action can be taken. In order to be effective, such a system must be able to differentiate between hunting and other vibration patterns which may not be a cause for alarm to limit the incidences of false alarms.

Using advanced digital signal processing, IEM analyzed truck vibration data and identified the specific vibration signature of hunting. This signature clearly differentiates hunting from signals generated by such transient phenomena as switch points and rough track. This approach has been confirmed through three distinct methods: simulation, analysis of hunting data developed by the TTX Corporation, and the analysis of actual truck vibration data gathered by a prototype system developed by IEM used in six test runs on AMTRAK cars operating in the NE Corridor.

The most significant research finding is a precise frequency spectra corresponding to the onset of hunting. This frequency spectra appears to be constant at all speeds and amplitudes. This means that the onset of hunting can be identified at speeds and amplitudes well below levels associated with safety or ride quality concerns. Once hunting has been identified, other signal signatures such as peak amplitudes of the root mean square (RMS) values of the lateral oscillations, slope of the RMS values, and duration of the oscillation pattern can be used to determine the significance and severity of the hunting event so that appropriate warnings can be generated and remedial action can be taken in a useful time frame.

The second significant finding of the research program is the characteristic signature of a well-tuned truck to a lateral disturbance. This algorithm, based on a polynomial function representation of the RMS-peak-amplitudes of a transient response, proved to be very reliable given the limited amount of data available. This signature can be used to satisfy the fleet screening function called for in the original solicitation. Trucks which show a slower than normal recovery rate can be tagged for inspection. Furthermore, the same analytic approach which generated this signature can be used to identify additional signatures related to a broad assortment of truck defects including: worn wheels, flat spots, bearing defects, and worn dampers. Also, by interfacing a geographic database with the hunting detection system, it will be possible to classify track conditions based on their impact on ride quality and hunting. Additional research incorporating an expanded database from a variety of trucks in field service would enable us to more specifically identify such signatures. This technique has the potential to provide a valuable new ability to develop cost-effective fleet maintenance practices.

In sum, the feasibility of using digital spectrum analysis techniques to determine the onset of hunting has been demonstrated. The approach has been shown to be able to provide reliable information about the propensity of a specific truck to hunt by comparing the response of different trucks to known lateral disturbances at a constant speed. Finally, the robustness of the approach is such that it has a broad potential to identify a wide variety of truck defects along with the ability to classify track conditions relative to impacts on ride quality and truck performance. Additional research is required to acquire additional data from a broad variety of truck in service to further demonstrate the utility of the approach.

I Problem / Opportunity Identification

A The Problem of Hunting

One of the critical phenomena that potentially limit the operating speeds of high speed rail vehicles is "truck hunting." Truck hunting is a high frequency pronounced yaw/oscillation of the truck that can occur as a function of: speed; weight distribution; the mechanical condition of the car, truck, and wheel; track geometry and condition; environmental factors such as wind, precipitation and temperature; and dynamic factors such as the adhesion between the wheel and the rail. When hunting occurs, it can result in violent flange/rail contact leading rapidly to unacceptable levels of lateral stress which can lead to wheel climb and rail overturning, deteriorated ride quality as well as wear and damage to the track, wheel, and truck components. As speeds increase, both the potential for hunting and the potential violence of the hunting phenomenon increase.

Although advances in truck design incorporated into such vehicles as the X-2000 and the ICE train have lead to significant reductions in hunting propensities along with higher critical speeds below which hunting is not likely, the concern about hunting is still real both for the higher speeds of the more advanced trains and even for the older truck designs which operate at lower speeds. Additionally as equipment ages, wear conditions can significantly change the operational characteristics of the truck leading to lower critical speeds than those specified for the original equipment.

As we can see from the following chart taken from USDOT's Annual Accident/Incident Bulletin, there were 644 accidents due to wheel rail interactions which include hunting as well as various other phenomena with a total cost of more than $38 million between 1984 and 1991.

Accidents Due to the Interaction of Lateral/Vertical Forces
Number and Cost Per Year
1984 - 1991

Year	1984	1985	1986	1987	1988	1989	1990	1991
Number	102	100	67	53	74	80	80	88
Cost	4,094,051	5,175,329	4,125,643	3,246,979	3,932,665	6,556,014	6,326,011	4,712,688

In addition to the overriding safety and ride quality issues, the high wheel/rail lateral forces associated with hunting produce expensive accelerated wear rates on the wheels, rails, and various components of the truck including the center plate and bolster.

B The Definition of Hunting

Hunting is a natural and inevitable tendency for a truck with tapered wheels and axle sets to oscillate as a way to dissipate lateral shocks. Although proper wheel profile maintenance, rail profile

maintenance, and truck maintenance can mitigate the impact and severity of hunting, no truck design can completely eliminate the phenomenon altogether at all speeds. Francis Dean defines hunting as:

> "Hunting is a combination of yaw and lateral displacement which exists, to some degree, at every speed. As speed increases, the frequency and amplitude of this motion increases, until at the critical speed, a divergent oscillation occurs. This oscillation is limited by flange contact.
>
> "The critical speed of hunting can be controlled chiefly by three parameters:
> - primary stiffness in the lateral and longitudinal directions, and
> - secondary yaw stiffness." [1]

The standard working definition of hunting was, like so many railroad products, services, and concepts, developed by the American Association of Railroads (AAR) for application in freight operations. The AAR defines truck hunting as "six or more consecutive oscillations having a peak acceleration in excess of 0.8 g peak-to-peak at a frequency of between 1 and 10 Hz."[2].

There are a number of problems with this definition for our purposes. The greatest problem with the definition is that it is static while hunting describes a dynamic phenomenon. What this means is that a truck may experience hunting in the classic sense in response to a transient stimulus such as a passing train or a switch point. The truck may rotate for a period and then quickly diminish. This is not hunting. At the same time, a truck which is oscillating back and forth may not reach the 0.8 g threshold and thus not be considered to be hunting by this definition. Meanwhile it continues to hunt with the resulting wear and tear on mechanical and track components coupled with a loss in ride quality. Depending on the track condition and other factors, this type of hunting has the potential for degrading into a dangerous condition. Obviously, to the extent that the hunting activity poses a safety hazard or is degrading ride quality, the engineer needs to be aware of such oscillations even if they do not reach the standard AAR definition of hunting.

As will be shown below, IEM has developed a definition of hunting based on the unique signature of the hunting phenomenon. This signature is constant relative to both speed and acceleration levels.

C New Approaches to Hunting Mitigation

Most of the work directed toward the goal of eliminating hunting as a problem has focussed on truck design. Specifically, the addition of snubbers and shock absorbers, the tightening of lateral clearances, the fine tuning of wheel/rail interfaces, and the addition of steering arms on existing truck designs have proven effective in increasing the "critical speed" at which hunting may be expected. Additionally the development of the radial or "steerable" truck, where each of the two axles can independently negotiate a curved section of rail, has had notable success. However, given the impracticality of retrofitting an existing inventory of vehicles with new trucks, the question arises as to the feasibility of developing ways to mitigate hunting as it occurs.

Several factors tend to support the thesis that such a mitigation strategy is feasible.

First, it is well known that for a given set of physical factors such as truck design and condition and track geometry and condition, the primary variable impacting on the propensity to hunt is speed. Observations taken by Dr. Hans True of the Department of Applied Mathematical Physics at the Technical University of Denmark, show that the onset of hunting does take place very rapidly within a very tight speed band. He states, "The jump in amplitude of the oscillation of the lateral acceleration is so sudden that it easily can be related to the speed of the vehicle at that instant within say 3 km/h." Adds, Dr. George List of Rensselaer Polytechnic Institute, "For any given transient (e.g. switch point impact), higher speeds lead to higher energy input per occurrence leading to a greater amount of energy to dissipate leading to a greater propensity to hunt."

Second, the development of slip/slide technology has enabled locomotives to relate the power assigned to a given motive wheelset with a real time measurement of the adhesion of that wheelset. In other words, if during acceleration, a wheelset begins to slip, power is immediately cut until the slippage terminates. At that point power is restored. This system ensures that all wheelsets are providing a balanced and full tractive effort in spite of different wheel/rail adhesion characteristics.

Third, new technologies such as dynamic suspension and top of rail lubrication may make it possible to stop hunting immediately following its onset without disrupting train speeds.

To take advantage of these factors it will first be necessary to develop a system which detects the onset of hunting in real time and conveys that information effectively to the engineer and/or an automated system so that corrective action can be taken in a useful time frame. A second goal is the development of a set of standard operating procedures and/or measurement techniques to provide maintenance personnel with cost-effective approaches to detecting the conditions that can result in a car having an unacceptable propensity to hunt.

Benefits of such a system to the nation's transportation system would be manifold:

i Improved safety to high speed rail operations.
ii Improved ride quality to high speed rail operations.
iii Highly cost-effective increase in the upper limit of operating speeds.
iv Improved fleet diagnostic techniques.

To the extent that uncertainty creates risk and risk results in caution, having a system to eliminate the uncertainty regarding the onset of hunting will enable the railroad to downgrade the impact that the fear of hunting currently has on speed limits.

II Technical Objectives

There were two technical objectives to the Phase I research program. The first of these was to demonstrate the feasibility of the development of a system to identify the onset of hunting in real time and communicate that information to the train engineer so appropriate remediation could be applied. The second was to identify a series of tests and/or procedures which could be implemented to detect trucks with a high risk of hunting.

A Real Time Hunting Mitigation

In simplest terms, this project can be summed up as a rule and a feedback system. The rule is: If yaw oscillation is prolonged, then slow the train down until it drops to some level Y. The feedback system consists of a set of devices which monitors yaw oscillation (frequency, amplitude, and duration) and when it exceeds an allowable limit X, it communicates that information to the train operator. To implement the project in these terms, we need to identify X and Y along with the specifications for gathering the data and communicating it to the operator.

However, life is never that simple. In some cases an isolated lateral shock from a track disturbance may not be remedied by a change in the train speed. At the same time, a series or pattern of lateral oscillations may indicate hunting which can be remedied by a change in train speed. So at the least, we need to differentiate between those lateral oscillations which can and those which cannot be remedied through available real time actions.

Additionally, to provide the optimal value for the system, it would be desirable to intervene as early as possible during the hunting phenomenon to minimize disturbances to ride quality, wear and tear on components, and the possible development of an unsafe condition while at the same time minimizing the disruption to the routine operation of the train.

To achieve this latter goal will require developing a new level of understanding of the hunting phenomenon which in turn will require answering five questions - our Technical Objectives:

i What is hunting?
ii What is the earliest reliable "leading indicator" of hunting?
iii How can we detect this "leading indicator?"
iv How can we ensure the "survivability and life cycle cost effectiveness" of the proposed hunting detection equipment?
v What is the remediation strategy?

B Hunting Diagnostics

The second objective of the SBIR is to develop a sensor system to "measure and detect mechanical conditions which are conducive to hunting, prior to operating the vehicle in a speed regime where hunting might actually occur." The outcome of such a system would be to screen fleets so that "operating fleets of cars had no vehicles unreasonably susceptible to truck hunting."

There are two factors that can cause a truck to hunt at a lower speed than that specified by the manufacturer. The first is a change in wheel/rail interaction. The second is the loss or breakage of a damping element such as a side frame damper.

1 How can these conditions be identified?

All RR truck manufacturers provide specific inspection and maintenance requirements to ensure that their trucks do not hunt below certain specified speeds. However, other factors that are out of the control of the truck manufacturers may alter the wheel/rail interaction and result in hunting at lower speeds than those predicted by the manufacturer. The most obvious of these is wheel and rail profile. At this point, there are several firms offering automated rail profile measurement systems so this proposal will not attempt to address the track side of the hunting equation.

a Identification of Changes in Wheel Profile

A great deal of work has been done on the effect of wheel wear on hunting. As the wheel wears, the taper changes and the wheel/rail contact point changes leading to an increase in the effective conicity of the wheel. One of the reasons that AMTRAK has chosen a relatively flat one in forty taper has specifically been for its resistance to hunting. Therefore, it is the feeling of many leading experts that monitoring wheel wear is perhaps the single most important screening that can be performed to ensure that no trucks have a preventable propensity to hunt.

In a wheel wear monitoring program, there are two questions that need to be addressed regarding changes in wheel profile. The first is

i What wheel wear patterns are associated with hunting for a particular car?
ii How can this wear pattern be identified?

The answer to the first question can be largely generated through modeling. It can be verified through the high speed acceptance testing that will take place at the Association of American Railroads' Transportation Test Center (TTC) prior to placing the high speed trains in service. This analysis will focus on identifying the specific wear profiles that will result in a propensity to hunt. It will attempt to quantify changes in critical speed (the speed at which hunting may occur) with changes in profile for a given truck.

Once the critical profiles have been identified, there are two approaches to screening the fleet to detect wheels that have worn to the critical profile. The first approach is the adoption of one of the automated wheel inspection technologies currently under development.

While there are a number of automated wheel profile measurement systems under development, none has achieved the level of reliability and market acceptance of the track profiling devices. In application, the device could be installed on a track segment approaching a yard, station, or service facility and could identify the type of car or locomotive passing over the inspection by using strain gauges to assess the weight of the vehicle and using a look-up table to relate to car/locomotive type. It would then take a profile of the wheel and compare that profile with the tolerable limits for that type of vehicle.

In the absence of automated systems, it may be necessary to augment existing wheel measurement practices to incorporate condemnation or defect wear limits related specifically to wheel/rail

interaction as opposed to the existing limits which focus exclusively on the structural integrity of the wheel. For example, virtually all existing wheel inspection practices focus on flange thickness, flange height, and rim thickness. Some include the use of a pi tape or a reference groove to measure wheel to wheel diameters. However, as reported in "Analysis of Wheel/Rail Interaction as Related to the Derailment of Train 278 on December 16, 1992" prepared by Zeta-Tech Associates, Inc. for the Bay Area Rapid Transit District, "The angle of the wear on the wheel flange does directly impact the potential for wheel climb, and it is also related to the Lateral and Vertical wheel rail forces."

They continue, "It should be noted here that wheel flange angle may not be uniquely defined by the tread and/or flange wear values." These are the values that traditional wheel inspection techniques measure. They conclude that, "wheel profiles should be taken, using a wheel Profilometer, on a regular basis and used to monitor the condition of the wheels in general, and of the wheel flange angle in particular."

Even in the absence of an automated wheel profiling system, it should be noted that a number of hand held wheel profile measurement devices are commercially available such as the IEM Wheel Profilometer (Please see Appendix C for Product Brochure). It may make sense to use such tools in periodic inspections. Wheel profile measurements are used extensively in Europe as part of the normal scheduled wheel inspection process. And although the use of hand-held profilometry is somewhat more complicated than current standard operating procedures for wheel inspection, it should not be impossible to incorporate a slightly advanced technique for the management of high speed trains.

As will be seen below, changes in wheel profile have a significant impact on the response of a truck to a lateral disturbance. The research performed demonstrated that one of the outcomes of a fully implemented system will be a maintenance report that will identify specific trucks as having wheels that require detailed inspection using an advanced Profilometer.

b Ongoing Maintenance Monitoring

As a truck's components age and wear, its ability to dissipate rail induced energy degrades. The "normal" dissipation rate following the passage over a track disturbance such as a switch point may decrease. While the net result may not indicate that hunting is taking place, it may well indicate that the truck has a growing propensity to hunt. For example, if a standard dissipation rate over the worst track disturbance is N cycles and one or more trucks on the train require some multiple of N to return to a stable condition, the hunting monitoring system should report that data for review by the maintenance crew.

For example, work performed by the KONI Corporation, a manufacturer of shock absorbers and dampers, has demonstrated the impact of damper degradation on the ability of a truck to return to stability following a lateral impact. Simulation work by KONI, illustrated in Figure 1, shows,

> "the number of cycles before the oscillations decay ... as a function of damping factor, and it should be appreciated that this relationship is independent of the initial oscillation amplitude.

The damping factor D is the ratio of c/c_c where c is the damping coefficient and c_c the critical damping coefficient."[3]

c Performance Based Truck Inspection

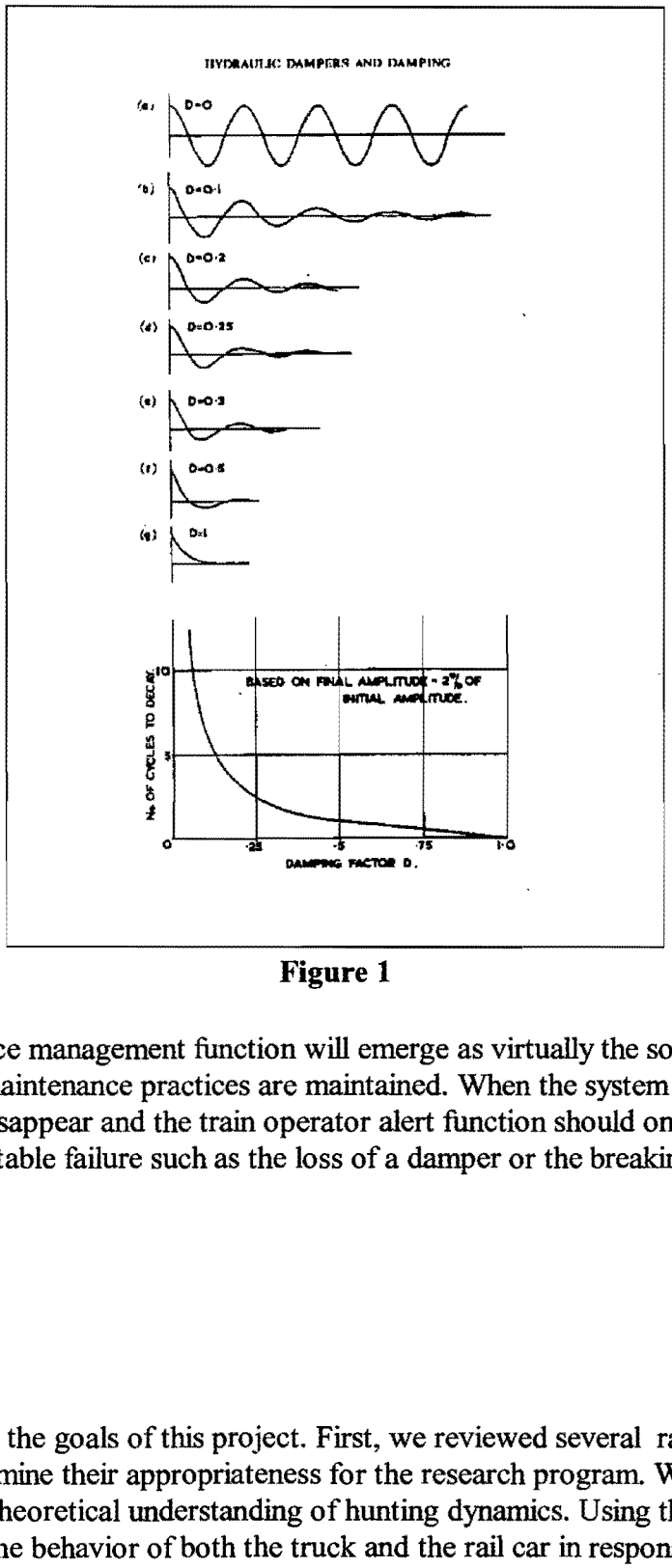


Figure 1

The research revealed a very specific signature for the response of a well performing truck to a lateral disturbance. As a given truck deviates from that response, a maintenance management report could be generated triggering a comprehensive truck inspection focussing on such elements as a visual inspection of such items as the signs of leakage in any of the damper systems, gauging the pressure of the air bag, checking the alignment of the truck, running the truck through a damper test station, or in severe cases, running the truck through a truck simulation station. In the truck simulation station, the truck would be removed from the care and placed on a station where its performance at speed could be simulated and inspected. We believe that when this system is fully developed and implemented, this maintenance management function will emerge as virtually the sole purpose of the system so long as normal maintenance practices are maintained. When the system is fully functioning, hunting should virtually disappear and the train operator alert function should only come into play in response to an unpredictable failure such as the loss of a damper or the breaking of a spring.

III Research Program

A Methodology

IEM chose the following work plan to fulfill the goals of this project. First, we reviewed several rail vehicle dynamics simulation models to determine their appropriateness for the research program. We then used the selected model to develop a theoretical understanding of hunting dynamics. Using the simulation model, we were able to explore the behavior of both the truck and the rail car in response to a variety of stress conditions such rail induced lateral shocks of various magnitudes and at various speeds, component failures such as broken springs and worn dampers, and normal wear conditions

such as changes in wheel profile. Based on the simulation work, we developed a more robust, dynamic definition of hunting incorporating digital signal processing.

Following the simulation work, we entered into a verification phase consisting of two components. The first component was to acquire actual truck dynamic data from field testing on AMTRAK equipment. The second component was to test the various hypotheses that we had developed against truck dynamics data gathered by other researchers including the Trailer Train Corporation and the Transportation Test Center.

B Simulation

1 Need for Simulation

IEM believed that train simulation would play a valuable role in this project for a number of reasons. The primary need for reliance on simulation is that simulation was the only feasible way to gain experience with dangerous conditions posed by hunting within the scope of work of the project. Using simulation, it was possible to analyze hunting behavior as a function of such factors as train speed, wheel profile, and lateral disturbances of various sizes. This computer simulation:

i Guided us both in the development of the specifications for and design of the hunting detection system;
ii Helped us identify mechanical wear conditions which could give a truck a propensity to hunt;
iii Helped us identify remediation strategies.

C Model Evaluation

IEM evaluated three train dynamic simulation models based on criteria of ease of use, availability, customer support, price, prior use in hunting analysis, validation experience, and overall utility for our study. The three models evaluated were Nucar, AgemCAD, and DYNSIM.

1 Selection of DYNSIM

We selected DYNSIM for its high ranking in all evaluation criteria. Not only was it designed specifically to model truck dynamics, but it has an excellent track record for its effectiveness in recommending proven design changes with many trucks in the field including a wide variety of locomotive, freight, and passenger car trucks. Most important, DYNSIM has been verified by its use by the Budd Corporation in designing modifications to the AMFLEET truck. According to George and Harold List, "DYNSIM's power can be seen in its ability to correctly predict subtle changes in angles of attack, critical frequencies, harmonic responses, hunting behavior, etc. which result from changes in wheel/rail geometry, creep coefficients, spring rates, shock absorber characteristics, wear limits at pedestal guides, and machining or assembly tolerances."[4]

Finally, DYNSIM was developed and is supported by Dr. George List of Rensselaer Polytechnic Institute who could provide a great deal of cost-effective training and support.

D Simulation Findings

IEM ran over 300 simulations using DYNSIM. These runs concentrated on tracking patterns of lateral acceleration on the truck over time as a function of changes in wheel profile, speed, amount of lateral displacement, creep coefficients (a measure of the coefficient of friction between the wheel and the rail), wear and tear of various truck components such as dampers, and truck design - two or three piece trucks. For each run, we generated a database of lateral acceleration data, graphed the output and then ran the numbers through our spectral analysis program.

There were two primary goals for the simulation program. The first was to determine if digital signal processing could be used to provide a more useful quantitative definition of hunting. In order to use DSP as an analytic tool, two requirements must be met:

- We must determine the characteristic frequency of the hunting phenomenon.

- This frequency must consistent as a function of various conditions of speed, truck design, the size of the lateral disturbance.

The second goal was to identify hunting patterns to determine what kind of sensor (g levels, frequencies) and what kind of mathematical analytic functions would be required to distinguish hunting from other, non-critical, vibration patterns.

1 The Hunting Frequency

Figures two through five summarize twenty-four simulation runs and offer compelling evidence that a characteristic spectral frequency for hunting exists. What we can see is that for both two piece trucks and for three piece trucks with both 0.2 and 0.4 inch lateral disturbance and at speeds ranging from 40 to 65 MPH, the characteristic frequency of hunting is both consistent and reliable at between two and three Hz. There are several significant implications for these charts.

- The first is that *hunting can be clearly defined as a pattern of oscillating lateral accelerations with a two to three Hz frequency.*

- The second is that the peak frequency of hunting is similar at all the tested speeds. This implies that by tracking the frequency of the truck oscillation, we can identify the *onset* of hunting even when the amplitude of the event is well below that of traditional definitions.

- The third is that two piece trucks have a similar response to the two piece trucks used in the AMFLEET cars. This means that the greater wealth of hunting data acquired over time about the behavior of freight car trucks has relevance and learning value for the AMFLEET trucks and that the solution developed for the AMFLEET cars has an application in the much larger freight market.

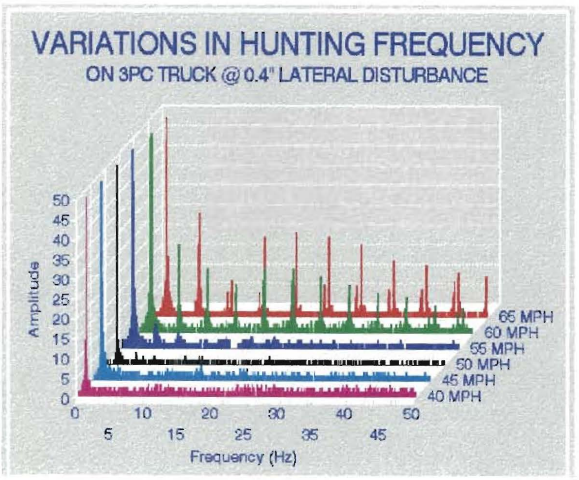


Figure 2

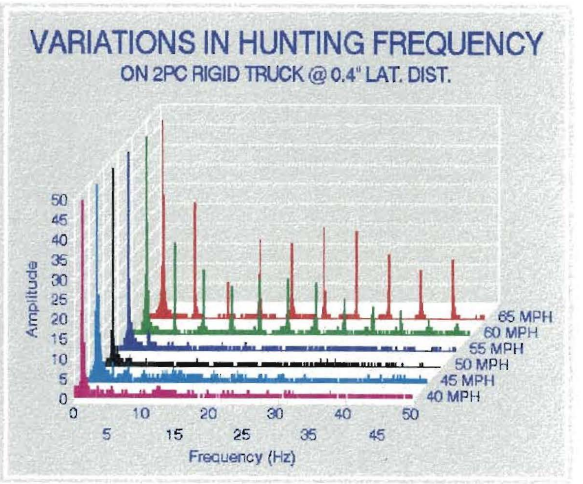


Figure 3

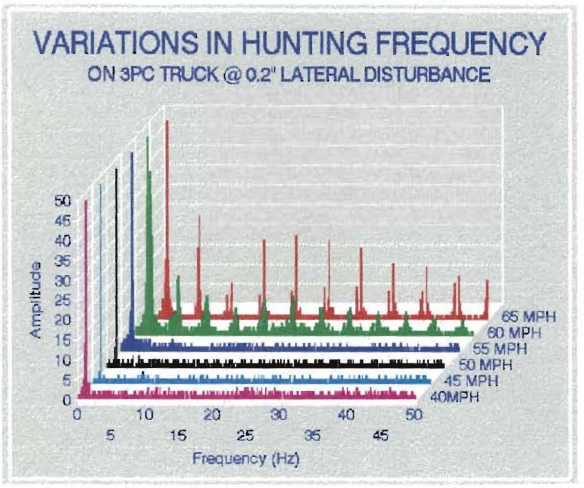


Figure 4

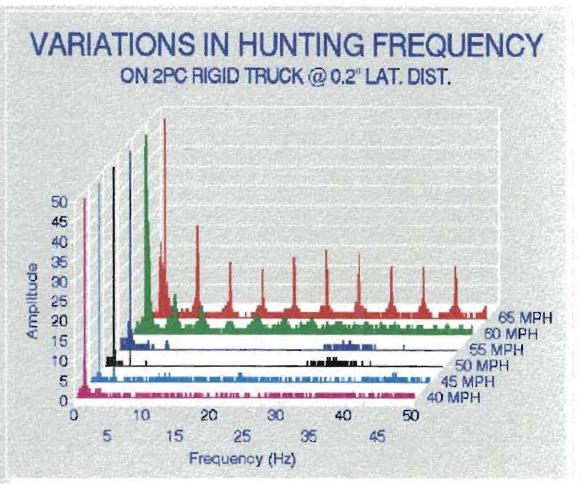


Figure 5

2 Slope

Even though the simulation work showed that spectral frequency analysis can be used as a reliable indicator of the onset of hunting, it is not a sufficient reason to alert the engineer to slow the train. Additional simulation test runs revealed additional factors than must be incorporated in any kind of real time feedback system for the train engineer.

The next six charts (Fig. 6-11) demonstrate that identifying the spectral signature of the lateral acceleration may not be a sufficient test to determine if there is a hunting problem. In Figure 6 showing the performance of a two piece truck at 40 MPH subjected to a 0.2 inch lateral disturbance, the truck hunts (matching the characteristic signature illustrated in Figure 7). but the amplitude of the hunting is low and is diminishing over time. Figure 8 shows the same truck and disturbance with speed equal to 50 MPH. Here the amplitude of the hunting is higher and the slope is increasing. However, Figure 10 showing the same truck at 60 MPH, displays a catastrophically growing

amplitude to the time domain data. *Clearly the slope of the changes in amplitude constitutes a second factor which needs to be considered in determining our definition of hunting.*

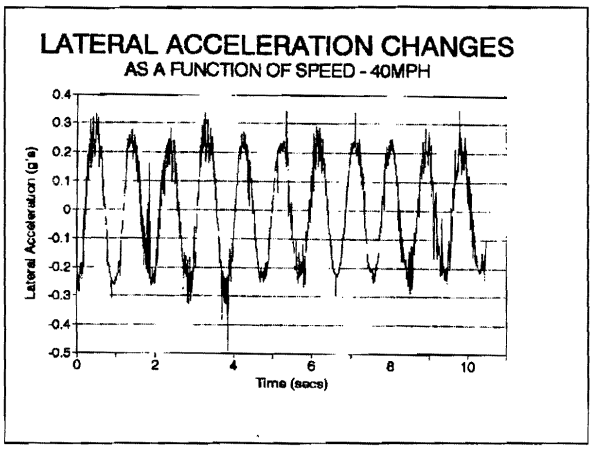


Figure 6

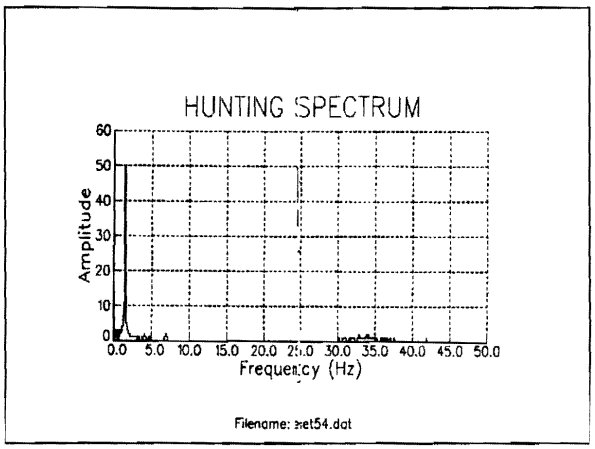


Figure 7

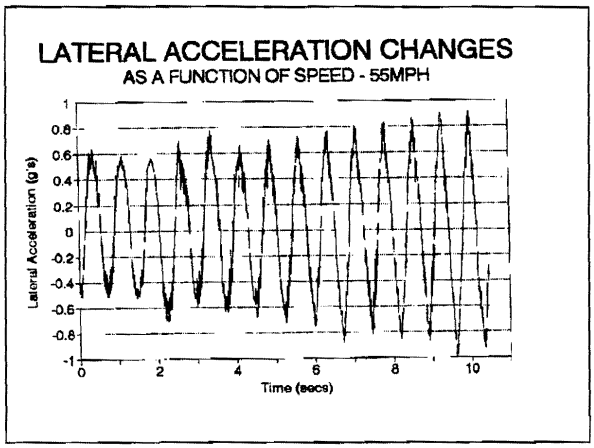


Figure 8

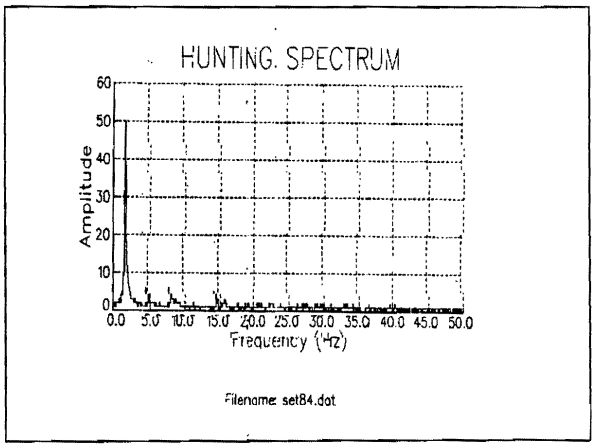


Figure 9

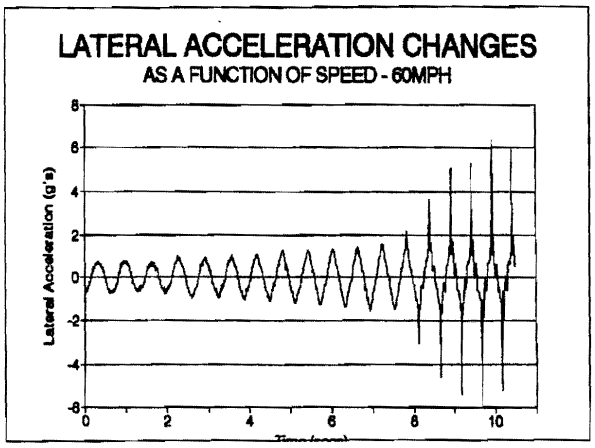


Figure 10

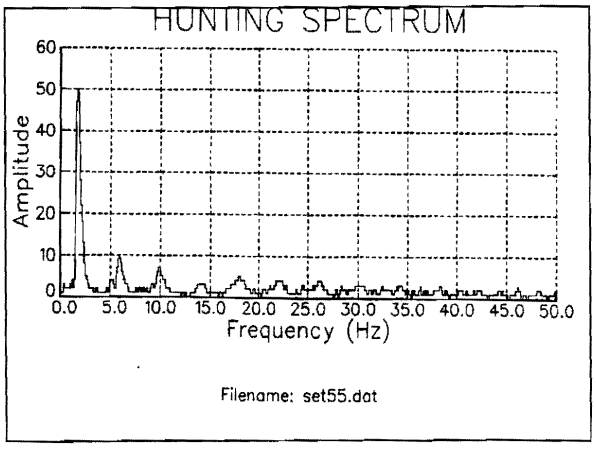


Figure 11

3 Duration

Another signal characteristic to monitor is the duration of the hunting behavior. In Figure 10, there are close to 8 seconds prior to the hunting phenomenon accelerating to a dangerous level. Figure 12 illustrates a different feature. Even though the slope is level indicating a non-catastrophic hunting condition, the duration of the incident is continuing even after a full ten seconds. This is in contrast to some of performance of the trucks in our AMFLEET tests where the truck returned to a stable condition in milliseconds. *So another condition to include in our hunting definition will be duration.*

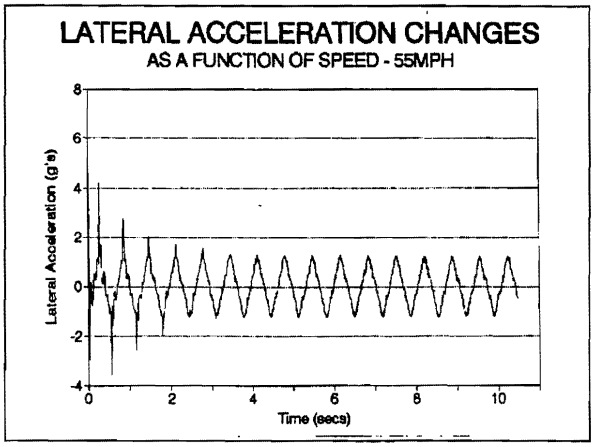


Figure 12

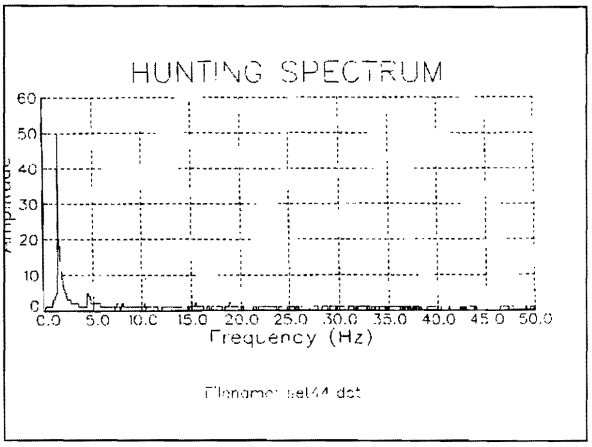


Figure 13

4 Peak Amplitude

It also stands to reason that the peak amplitude will be a primary consideration in determining if a hunting event should be recorded or reported to the engineer. *So peak RMS values of the time domain data are another key factor.*

In sum, the simulation work demonstrated that the definition of hunting must include four signature characteristics:

- peak amplitude of the lateral acceleration,
- the slope of the root mean square values of the periodic peaks,
- the frequency of the oscillations, and
- the duration of the event.

These factors could be used to develop the early warning system called for by the original solicitation. At the same time, the system could provide the data called for in the fleet management portion of the solicitation by charting the relative behaviors of the different trucks in response to a known track disturbance. For example, if, in a given train, one truck produced significantly higher amplitudes to the series of track induced disturbances than all the other trucks, it could be flagged for special inspection. We believe that as the data base grows, we will be able to derive specific signatures for such wear items as worn wheels, worn bolsters, worn dampers, and worn pedestals.

IV Verification

It had been our original intent to test these hypotheses against data acquired by AMTRAK and Bombardier regarding the performance of their trucks to verify our simulation results with some real data. It turned out that this data was more difficult to acquire than we had originally anticipated. Therefor, we were forced to embark on a verification program requiring the acquisition of our own data from AMTRAK trains. We developed a prototype data acquisition system including a sensor system and a rugged industrial computer based on performance specifications determined by our simulation programs.

A Software

One of the key elements to our data acquisition system was the development of a custom software program to capture, record, analyze, and report on the vibration data. The system was designed to interface with the vibration analysis hardware, perform Fast Fourier Transform analysis in real time, apply the curve fitting algorithms developed during the simulation phase of the project, conduct data modeling, provide intelligent data extraction algorithms, and provide a convenient, graphical data reporting function. (Please see Appendix B for source code.)

B Hardware

During the initial testing of the program, we used an off-the-shelf portable FFT recorder, the Diagnostic Instruments Model PS22. (Please see Appendix A for operating specifications.) There were pluses and minuses to the use of this product. The three negatives were that it did not provide us with the data analysis capabilities such as real time curve fitting of the data, it was limited in its data post processing capabilities, and it was too expensive to use on a broad scale. The positive was that it provided us with a calibrated base line against which we could measure the performance of our custom designed FFT recorder.

Basic Sensitivity: (measured)	101.9 mv/g @ 160Hz
Frequency Response:	1 Hz to 5 Khz ±5%
Max. Accel. W/o clipping:	50g
Max. Shock w/o Dammage:	5000g
Bias Level:	9.4 volts
Power Supply:	2.2 ma @ 15-30vDC
Resonant Frequency:	20KHz (typical)
Noise Floor	0.010 mv/rms (typical)
Transverse Sensitivity:	5% (measured)
Time Constant:	NA
Temperature (Maximum):	250°F
Mounting Torque:	35 inch/lbs.

Figure **14**

Our box is based on a standard IEM Industrial Computer modified to include a color touch screen flat panel display and 1.2 gigabyte hard drive. In addition to the software described above, we also developed a custom filter for the two Model 1026 Vibra-Metrics, Inc. accelerometers. The technical specifications for the accelerometers are included in Figure 14. The specifications for the A/D Converter for our system are included in Figure 15. Figure 16 shows the IEM Rugged Industrial Data Logger. Figure 17 shows the schematic of the low frequency filter we used to reduce the noise level of the signals that we captured.

Channels:	8 single-ended.
Resolution:	12 bits.
Input Range:	Bipolar: ± 5V.
Overvoltage:	Continuous ± 30V Max.
Conversion type:	Successive Approximation.
Converter:	AD574 or equivalent
Conversion Speed:	25 microsecond max.
Accuracy:	0.015 % of reading ± 1 bit.
Linearity:	±1 bit.
Trigger Mode:	Software Trigger
Data Transfer:	Program Control

Figure 16

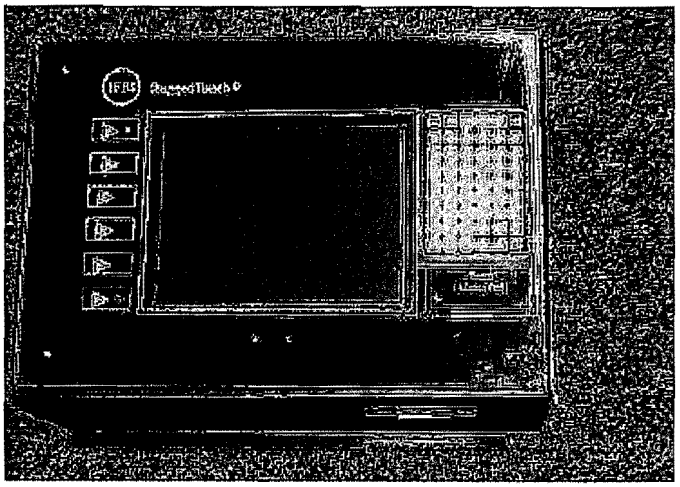


Figure 15

Prior to the field testing, we calibrated our unit in-house by tracking the responses of the accelerometers when placed on a vibration table of known operating specifications. We also compared our results to the results generated by the UPTIME unit with favorable results.

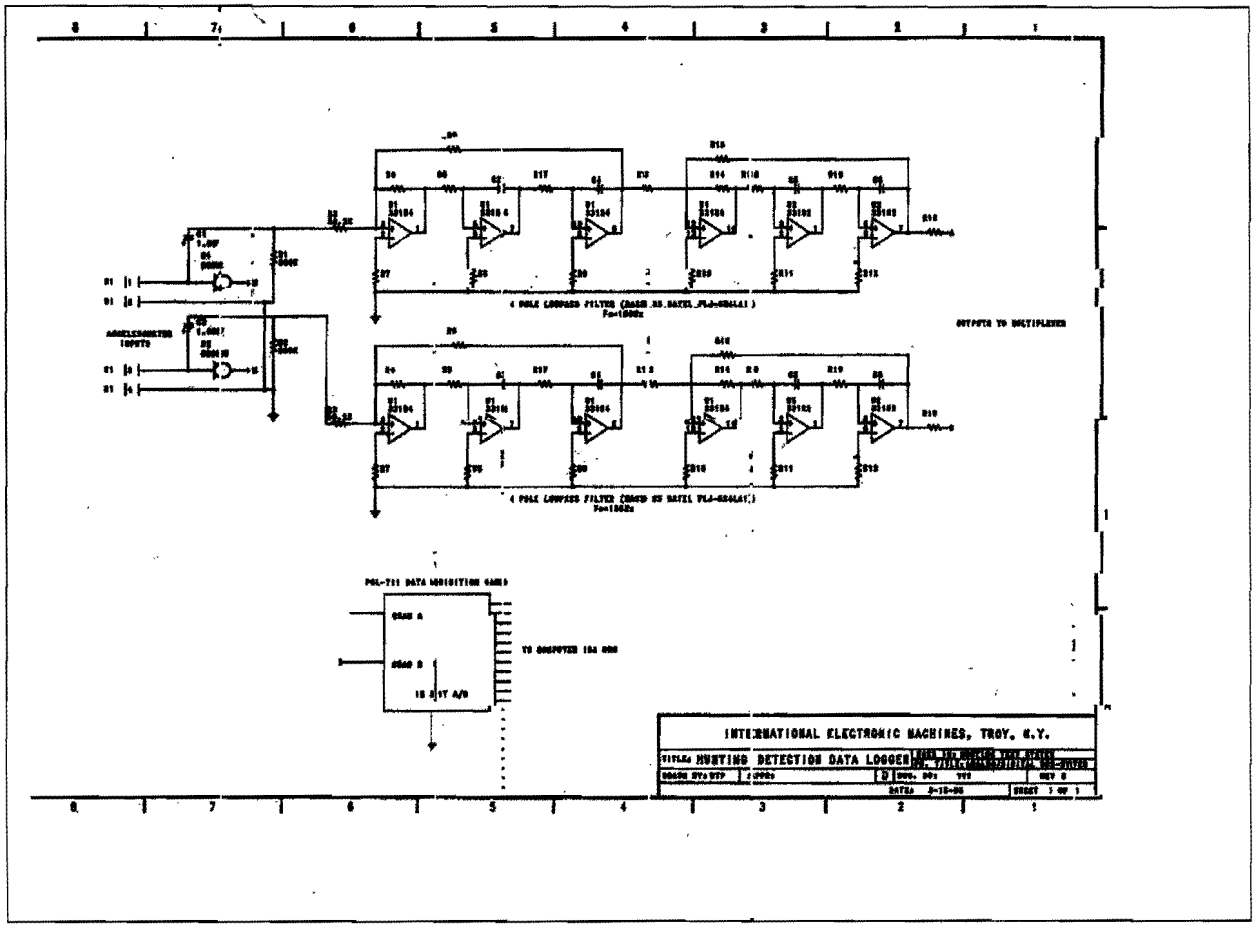


Figure 17 - Schematic of Low Frequency Filter

C Data Acquisition System

The data acquisition system specifications are included in the following Box.

Hunting (onset/actual) Detector Preliminary Specifications

1. **Frequency range:** 1 Hz to 40 Hz

2. **Lines** needed to avoid picketing: 80
   Frequency increments/frequency resolution: 40/80= 0.5 Hz
   Time Window for finest resolution sine wave: 1/0.5 = 2s

3. **Unwanted Accelerations:** 100G-150G
   Hunting Related Accelerations: <10G

4. **Hunting Time Duration:**
   Burst mode 15-45 seconds
   Continuous mode indefinite

5. **Truck Displacement:** 2" PTP
   (source D.J.Reynolds)

6. **Hunting Detection Choices (Development System):**
   a. Mount two accelerometers on the bolster as far apart as possible monitoring longitudinal acceleration (along the track) to detect yaw. Band pass the output from left and the right accelerometers and compare the amplitude and phase of the signals looking for comparable high levels of acceleration and desired phase difference. The output has to remain true for at least N successive cycles to be defined as hunting.

   b. Mount accelerometer on the left side frame above the wheel on one axle and the right side frame above the wheel on the other axle. Compute four spectra averages to achieve good S/N. Compare the left and the right spectra in terms of frequency amplitude and phase.

7. **Analog conversion speed:** 2.56 x highest frequency content
   80 x 2.56 ==> 205 times/sec

8. **Analog conversion accuracy:** 12 bit conversion

9. **Accelerometer Specifications:**
   G Range: up to 50G
   Output Voltage: 100mV/G
   Resolution: 0.1G
   Temperature Linearity: up to 225 degrees Fahrenheit
   Axis: lateral axis
   longitudinal axis (for yaw calculation)
   vertical axis (for comfort calculation)
   (cross correlation of the three axes to insure reliable hunting detection)

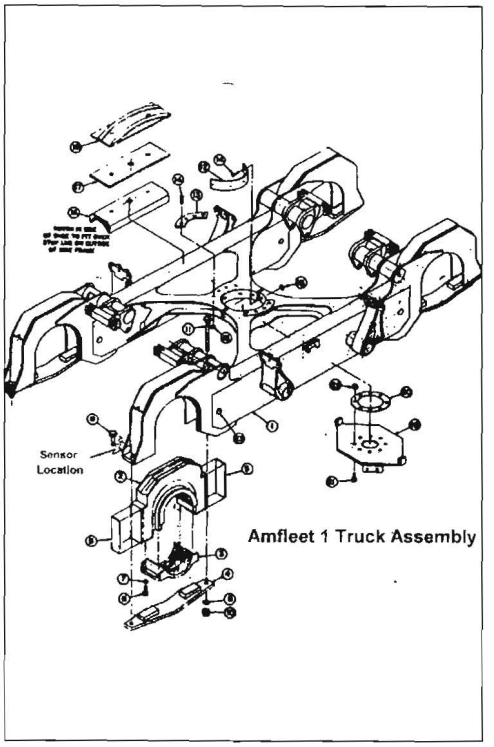


Figure 19

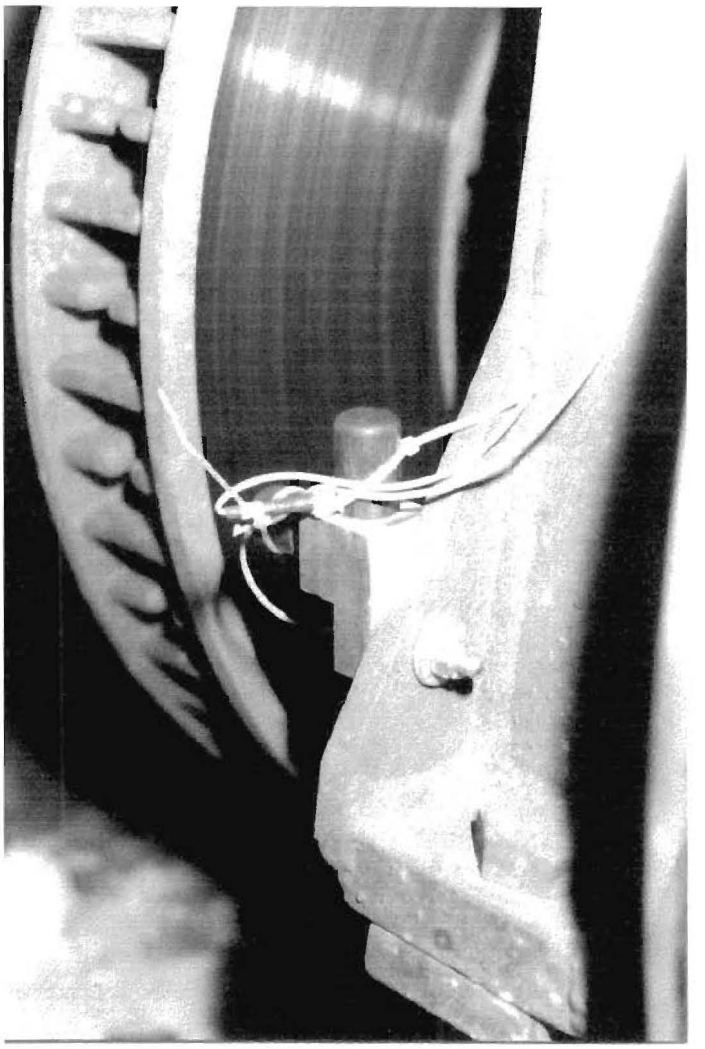


Figure 20

To capture the required data, we installed the two accelerometers on the truck frame of the lead truck of AMTRAK's track geometry can and ran the wires up into the cab where we captured the data using IEM's Rugged Touch Industrial Data Logger. Figures 19 and 20 show the placement of the two accelerometers.

D Field Testing

We ran one round trip between Rensselaer and New York City and three round trips between Union Station in Washington and Penn Station, New York.

The data generated by these trips revealed that very little hunting was taking place. In response to lateral disturbances from track, wind, and other trains, trucks were oscillating but were returning to a stable condition as designed.

Prior to acquiring the data, IEM measured the four wheels of the truck where the accelerometers were placed with the IEM wheel Profilometer. The profiles, displayed in Figure 21, show that all four wheels are close to a new profile.

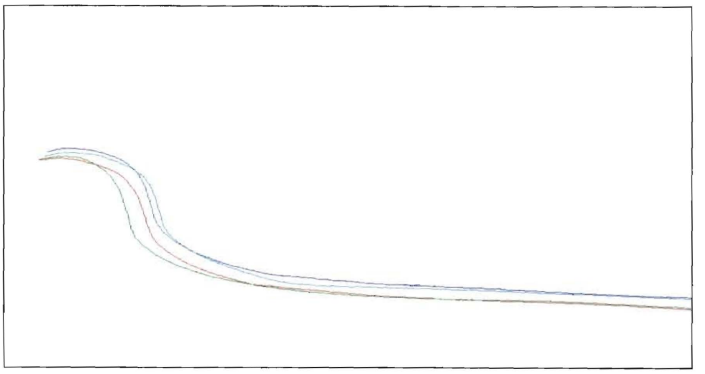


Figure 21

As can be seen from the following eight charts, the AMTRAK trucks are performing very well, taking minor hits (lateral shocks of 0.2 to 0.4 g's) and returning quickly to stability. There are two points to notice about these charts. First, notice the way the lateral acceleration rates diminish over time (217 points represent approximately 1.45 seconds). Second, notice the consistency of the frequency in the spectral analysis charts. Here the common frequency is around 10 Hz instead of the one to three Hz found in our simulation studies. We attribute this difference to the different truck design and to the fact that the duration and number of cycles for each of the transients were not quite sufficient to conduct reliable FFT analysis.

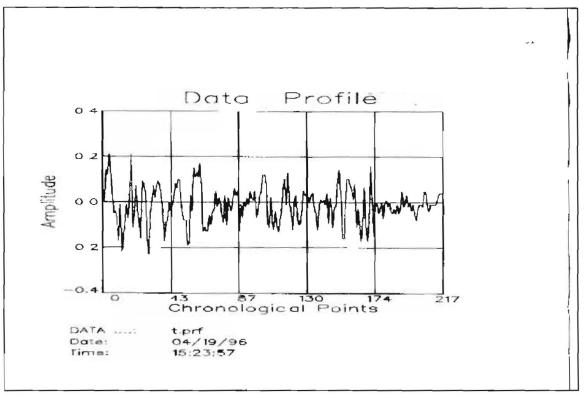


Figure 22

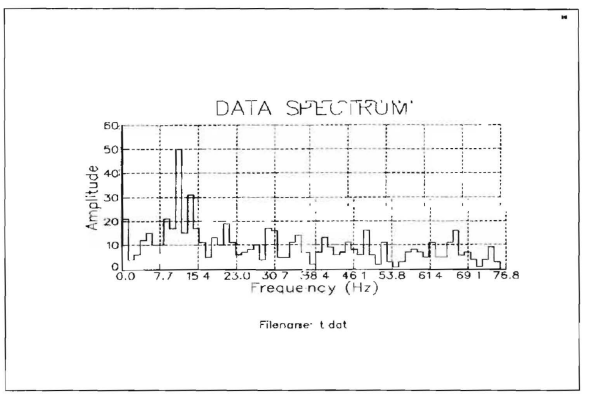


Figure 23

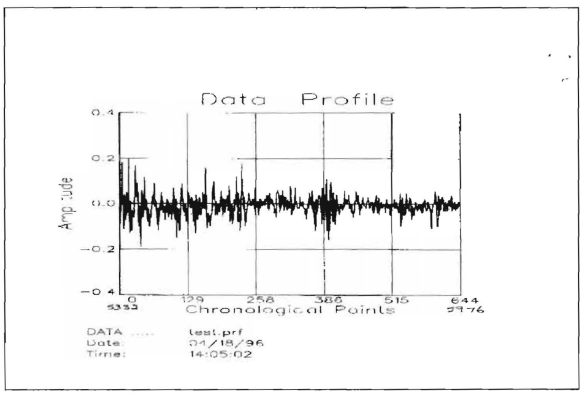


Figure 24

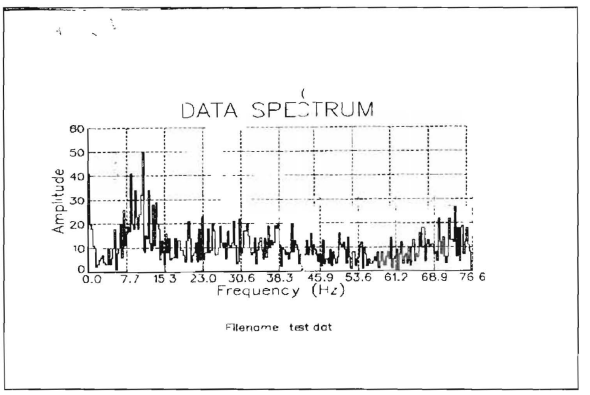


Figure 25

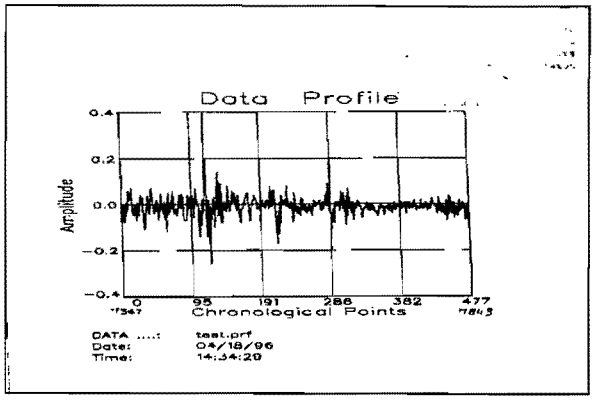


Figure 26

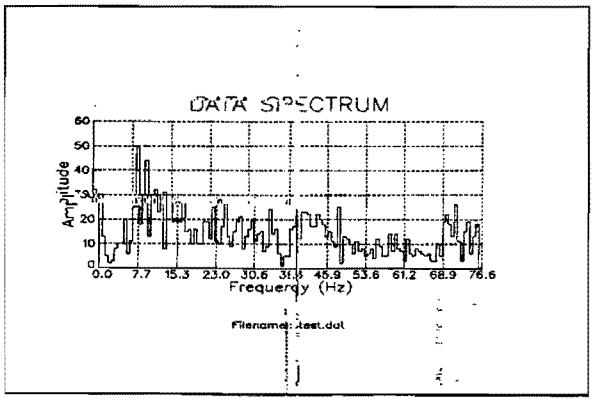


Figure 27

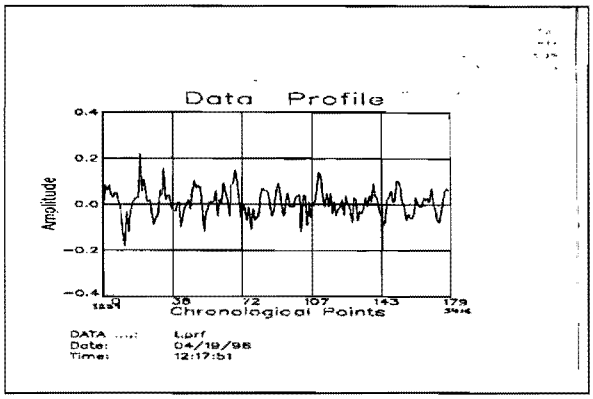


Figure 28

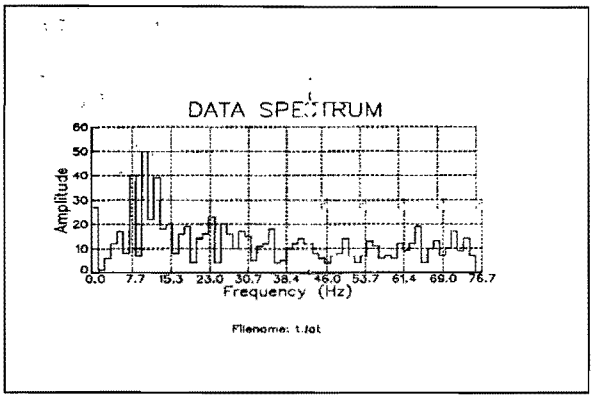


Figure 29

Since the data acquisition rate of the system was 150 points per second, each run from Washington to New York generated approximately 1.89 million data points. To make sense of this mass of data, we scanned the data to attempt to identify patterns or groups of data which seemed to have similar characteristics. One such group seemed to be the truck response to a transient. Figures 30 - 39 illustrate this pattern. We then attempted to develop a mathematical algorithm which could summarize or describe the pattern. Algorithms tested included straight line fits of the transient peaks, polynomial fits to the transient peaks and polynomial fits to the root mean squares of the peaks of the transients. We determined that a fourth order polynomial equation could be used to describe the normal response of a well functioning truck to a routine transient.

On each chart is a line indicating the polynomial equation fitted to the root mean squares of the time series data.

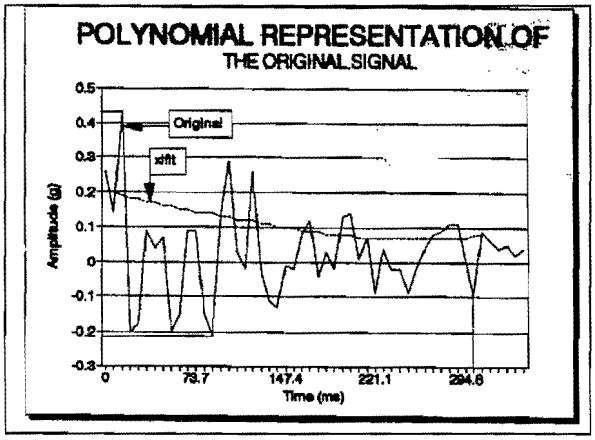


Figure 30

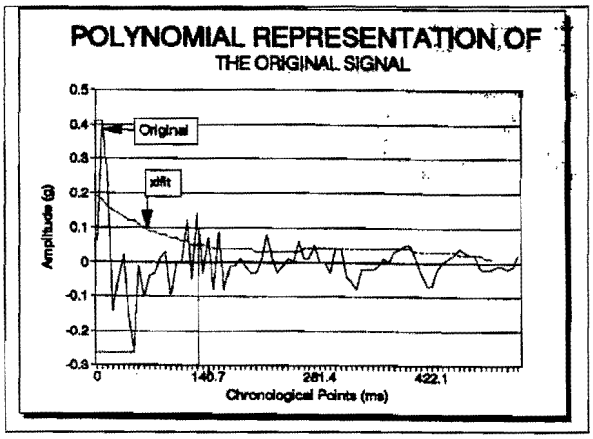


Figure 31

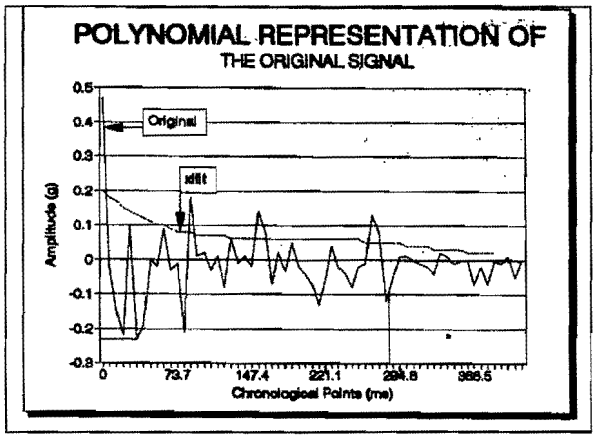


Figure 32

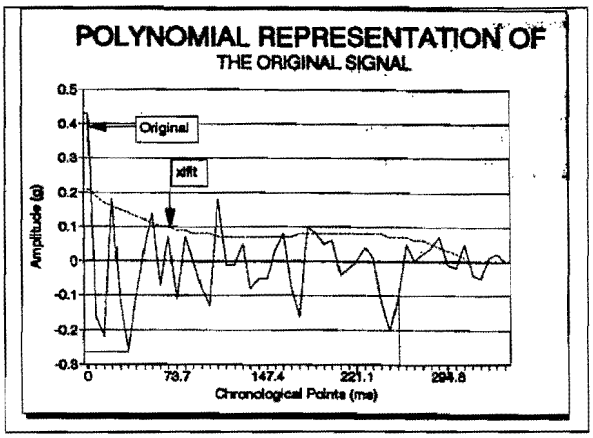


Figure 33

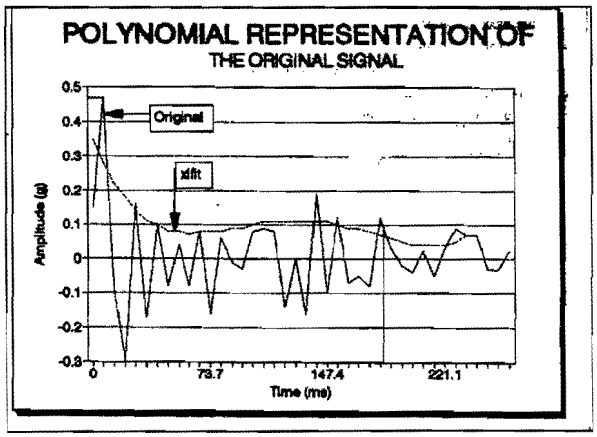


Figure 34

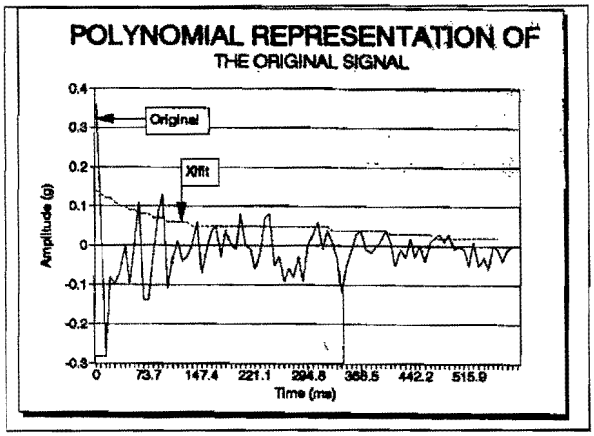


Figure 35

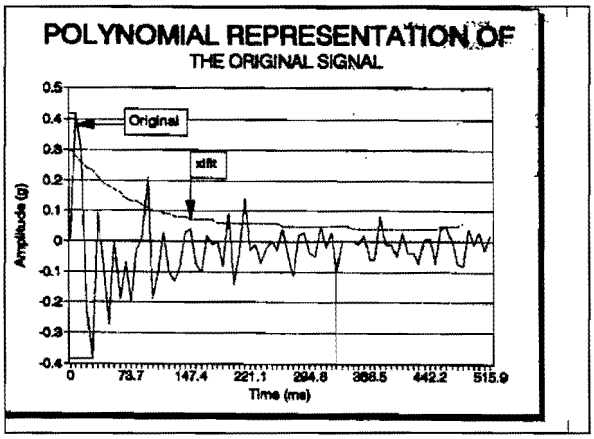


Figure 36

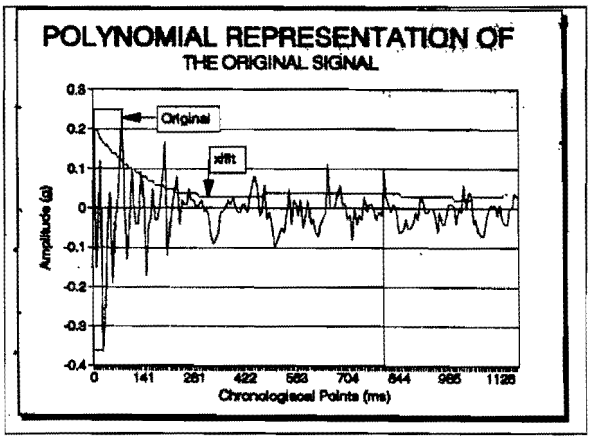


Figure 37

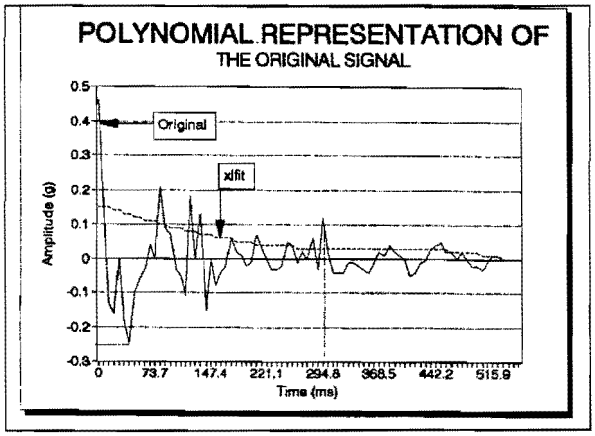


Figure 38

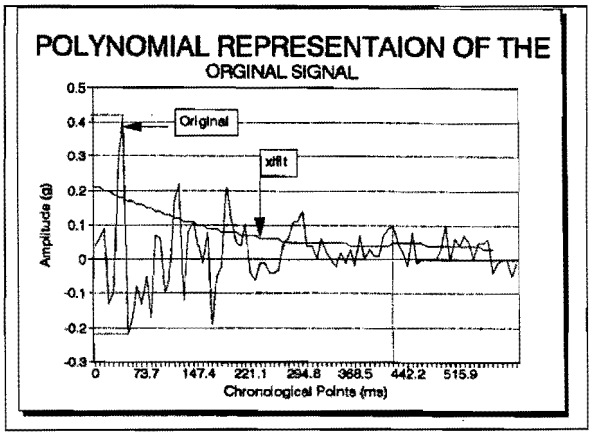


Figure 39

Each of these polynomial representations of the original signal is in the form of $A_1 + A_2x + A_3 x^2 + A_4 x^3 + A_5 x^4$.

To test the validity of the equation, we attempted to recreate the original data using the formula. The following chart shows that for 11 different transients, the values of A_2, A_3, A_4, and A_5 are similar. In fact we do not even show A_4 and A_5 since they essentially lie on top of one another. In the chart, the value of A_1 represents the maximum value of the lateral acceleration which is mathematically represented by the Y intercept and the squares indicate the time required for the transient to work itself out and the truck to return to a stable position.

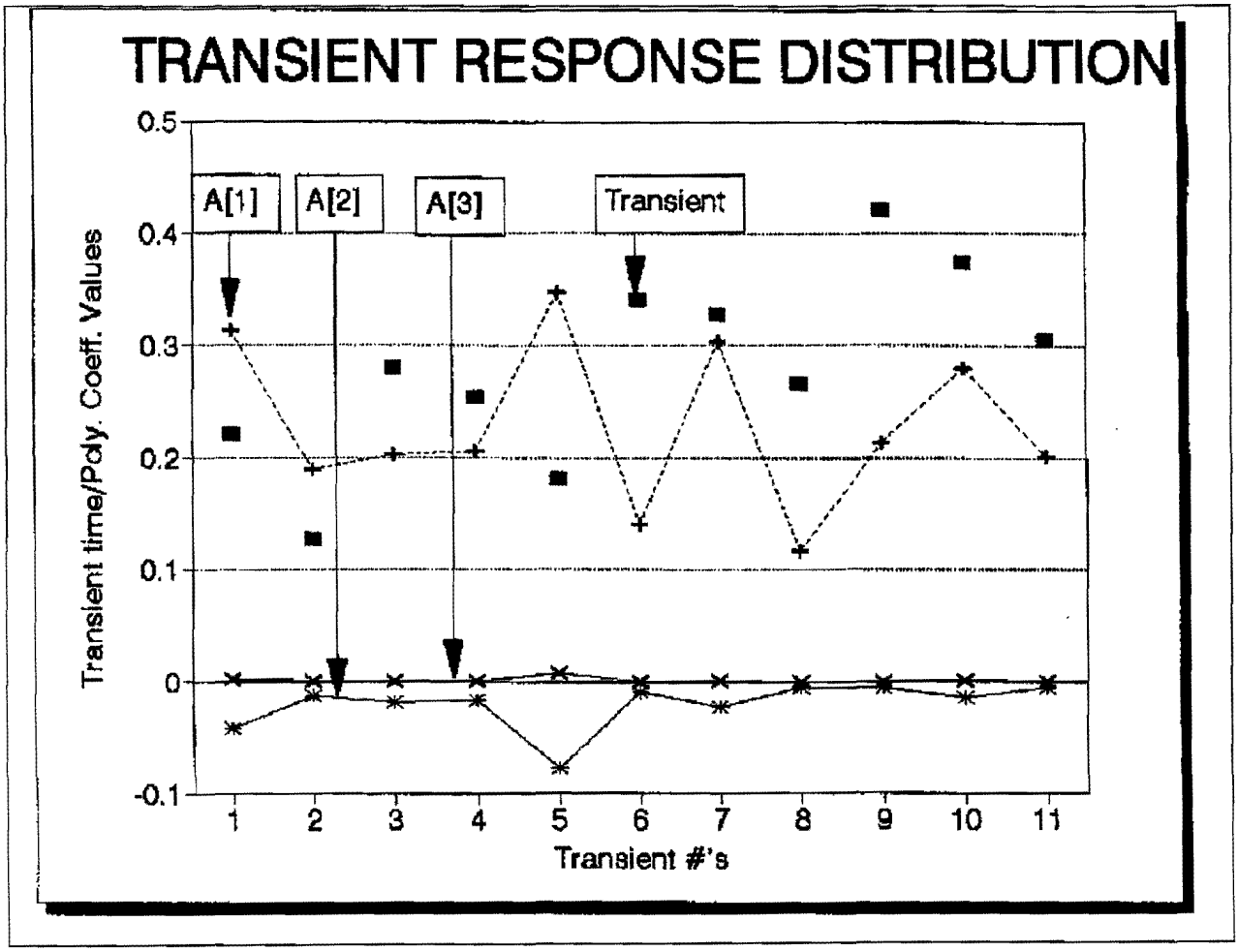


Figure 40

In addition to tracking the lateral acceleration, we also tracked the vertical acceleration during these runs. Charts 41 - 46 show that there is a strong correlation between the two:

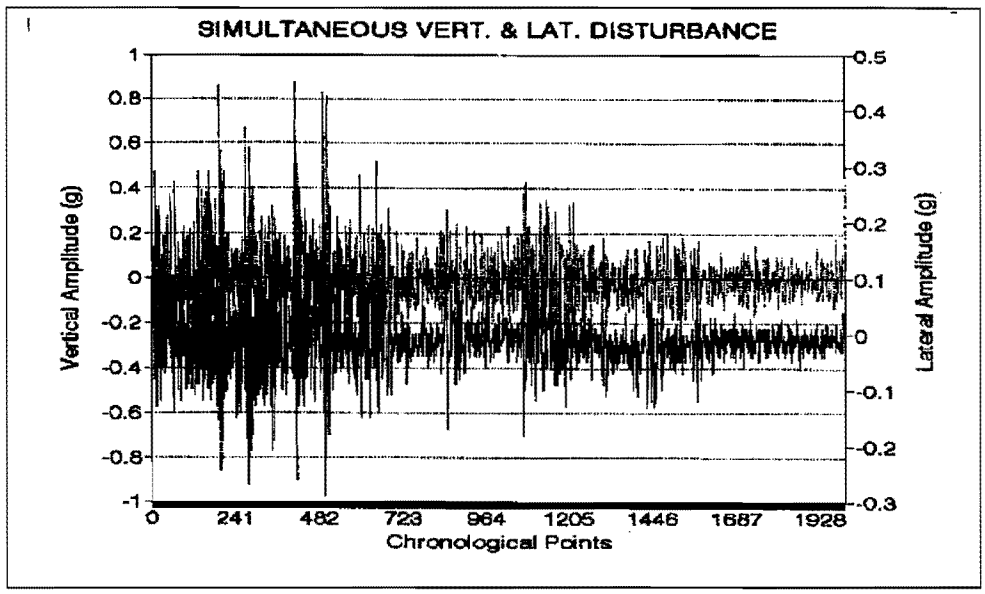


Figure 41

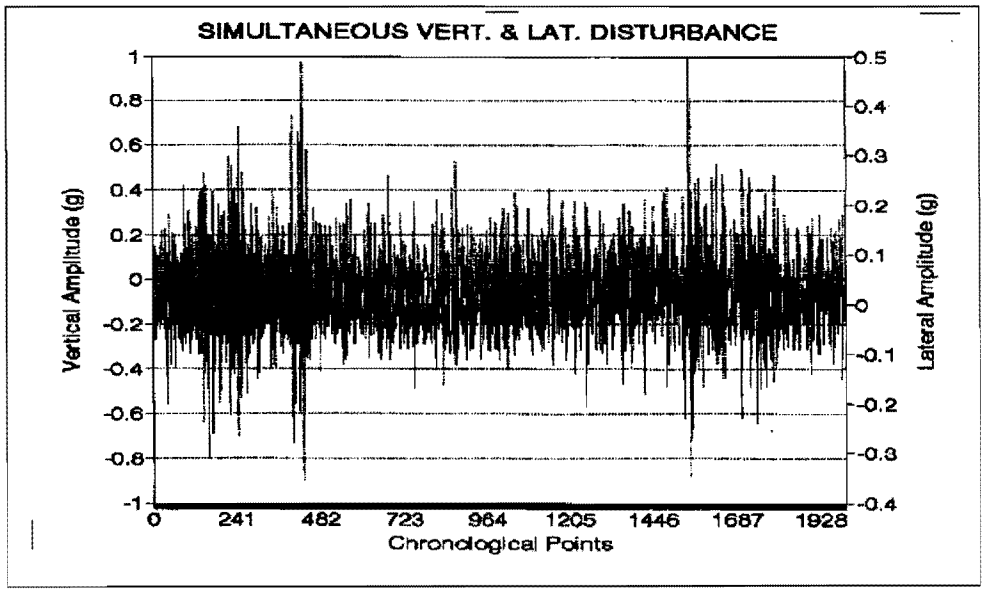


Figure 42

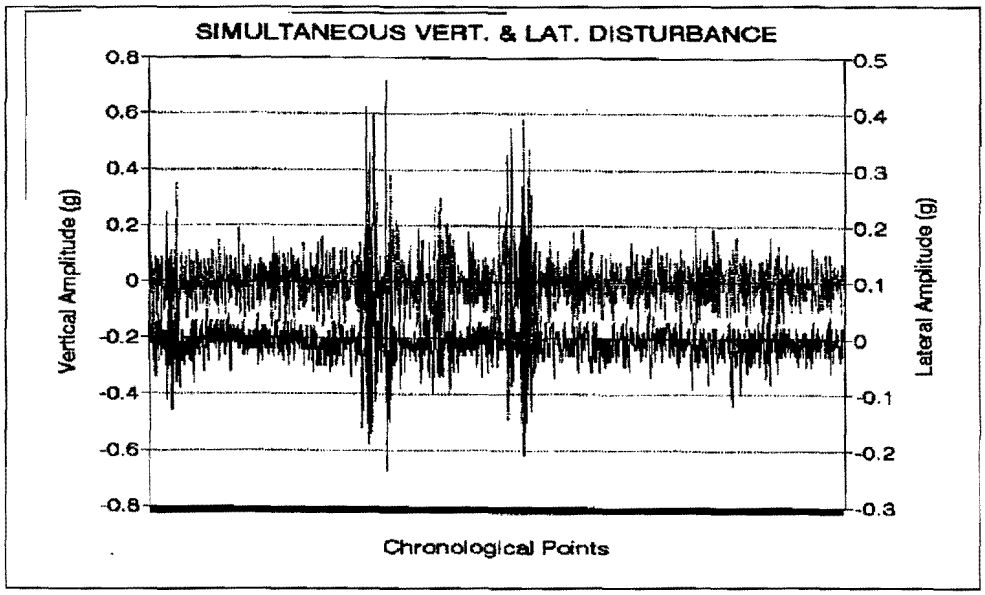


Figure 43

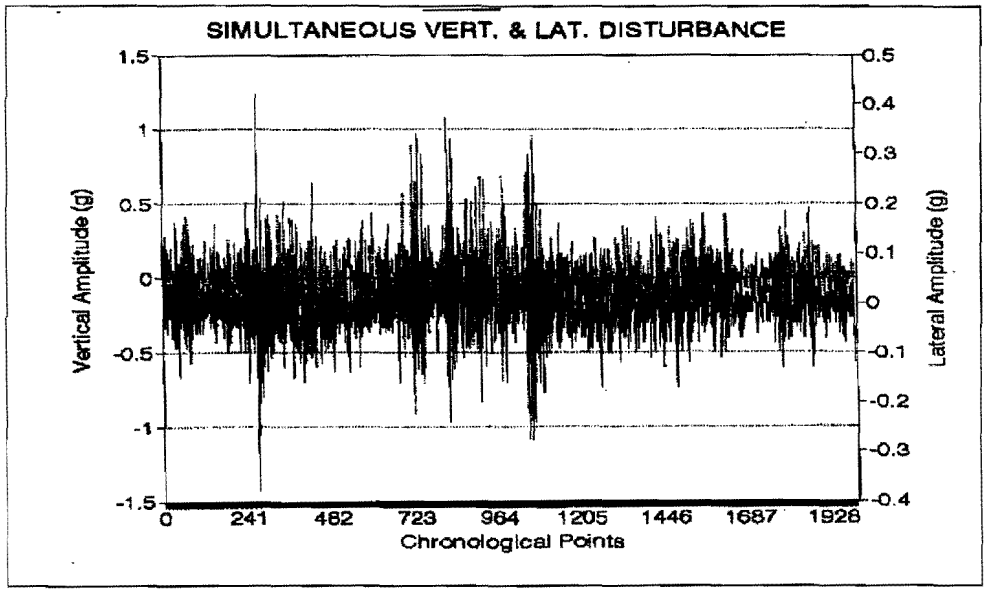


Figure 44

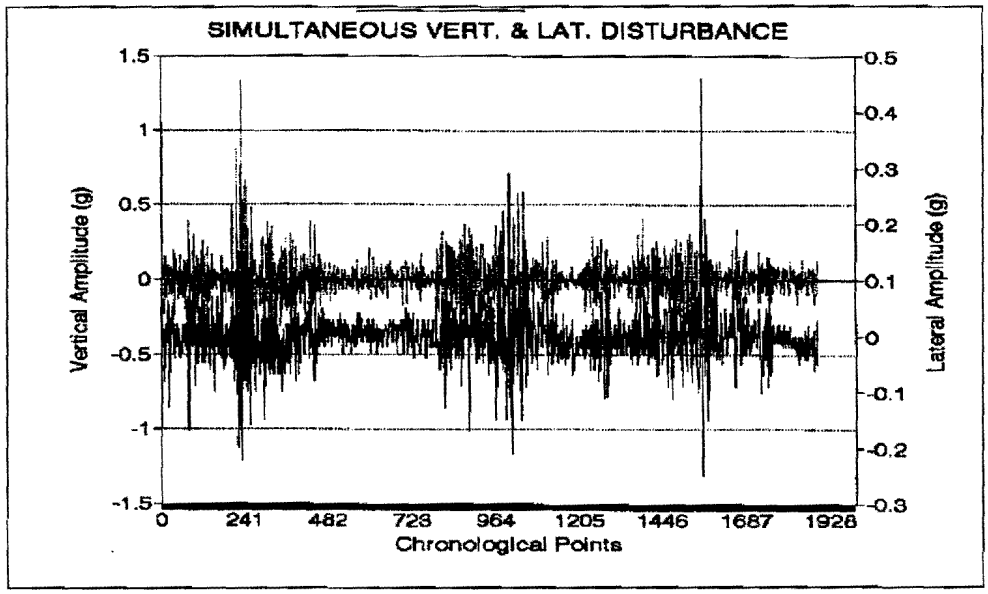


Figure 45

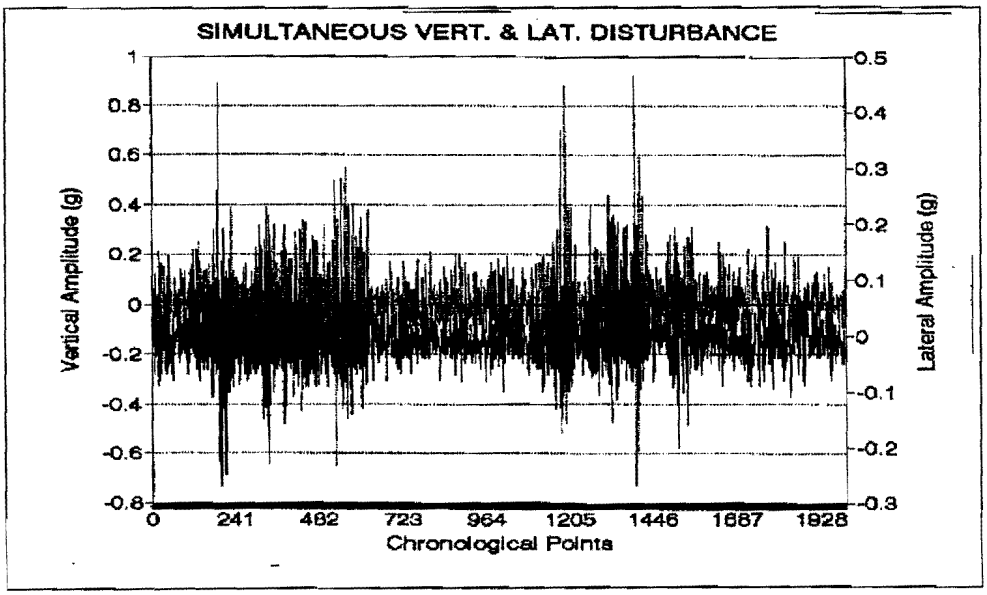


Figure 46

When we ran the frequency analysis on vertical acceleration data we found some surprisingly strong spectral signatures as can be seen on Figures 47 - 51. :

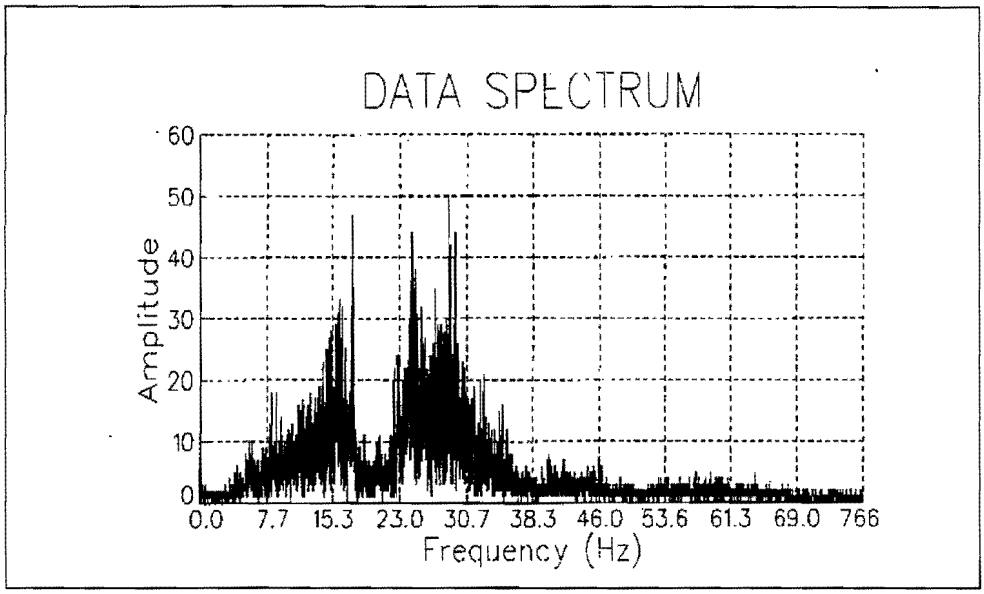


Figure 47

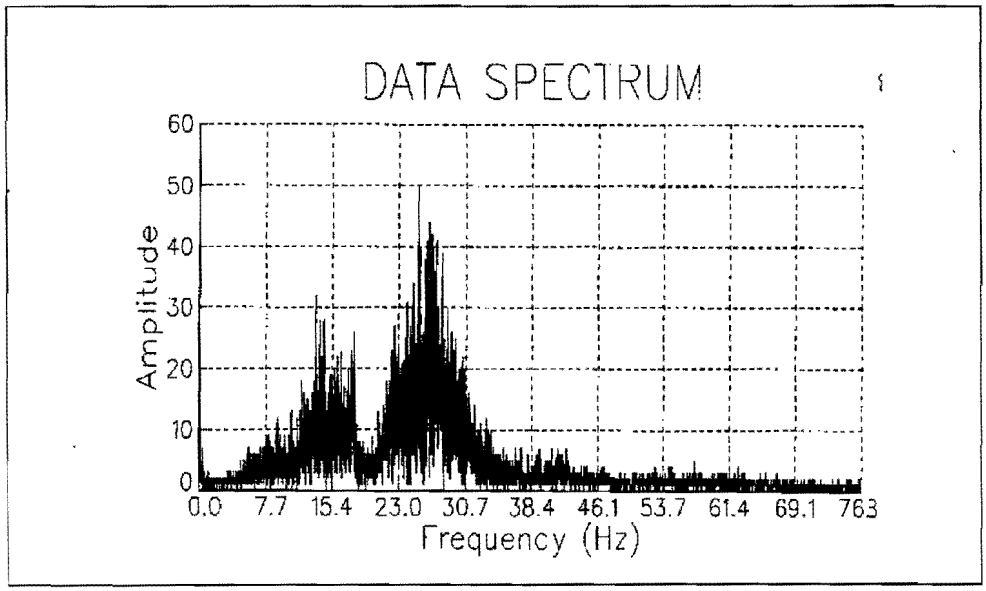


Figure 48

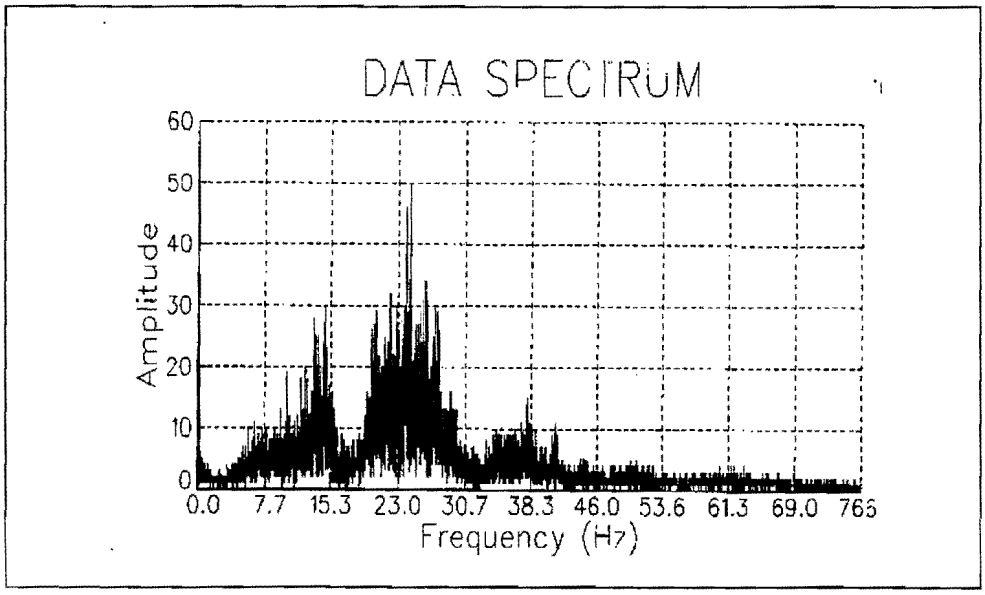


Figure 49

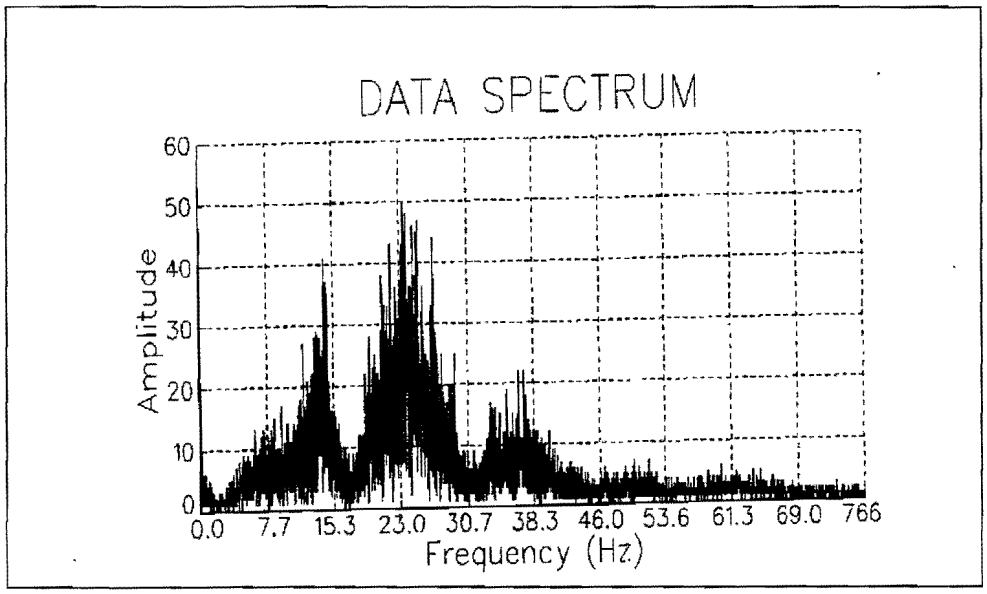


Figure 50

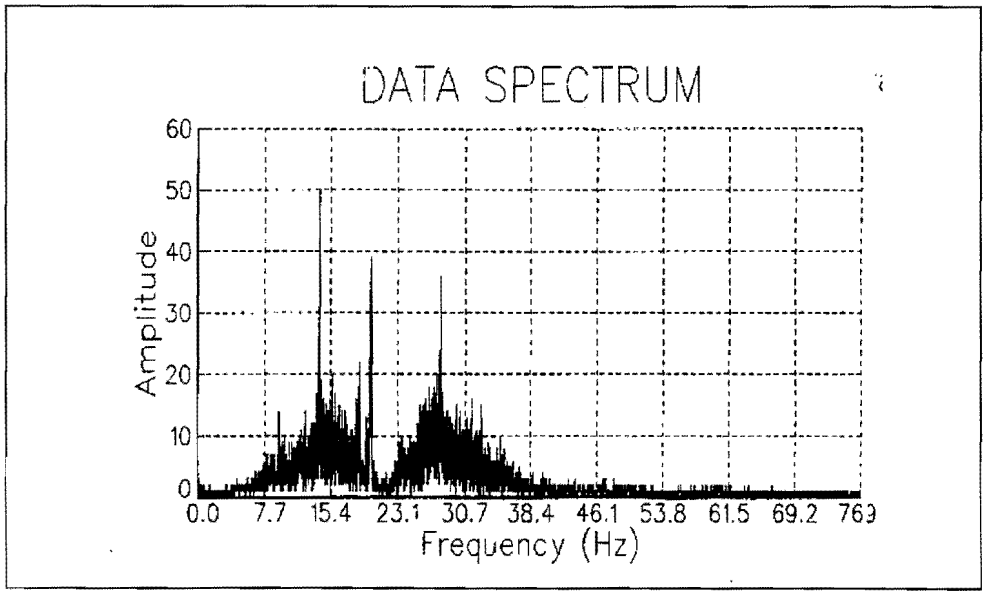


Figure 51

We now believe that the vertical rocking motion described by these signatures represent "ballast memory" stemming from the days of bolted rail as described in an ENSCO study.[5] The interesting aspect to this analysis is that it shows that in addition to monitoring hunting and truck performance, the proposed system can also be used to monitor track performance. Just as we believe that it will be possible to develop specific signatures for such truck conditions as worn wheels or aging dampers, we also believe that it will be possible to develop specific signatures for such track conditions as worn switch points and ballast deterioration.

E The Null Case

Having demonstrated the presence of reliable, consistent signatures for truck behaviors where hunting is present, we also chose to look at the frequency spectrums of truck acceleration data where there was no discernable hunting. Figures 52 and 53 show that *the hunting "signature" is not present when there is no hunting.*

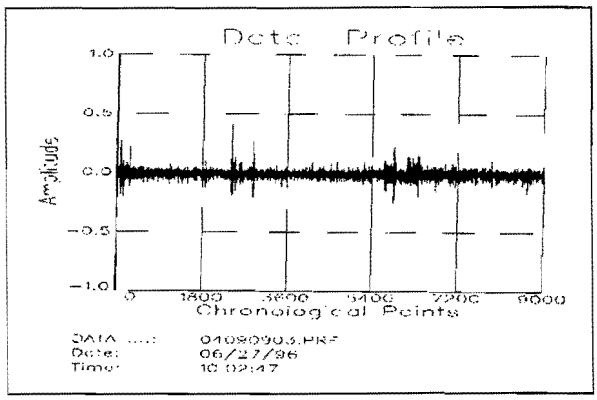


Figure 52

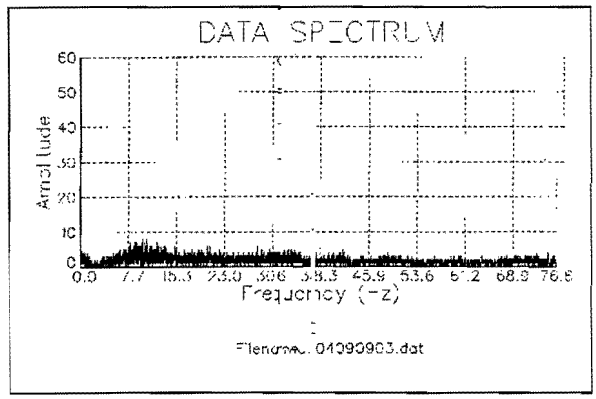


Figure 53

F Hunting Confirmed - the TTX Data

We were still concerned that even though we had verified the application of and utility of digital signal processing in the monitoring of truck behavior, we had not had the opportunity to verify our findings about hunting based on real data. Fortunately, we managed to obtain some actual hunting data from the Trailer Train Corporation and as demonstrated in 54 - 57. This data did confirm our findings that hunting does have a clear and consistent spectral signature.

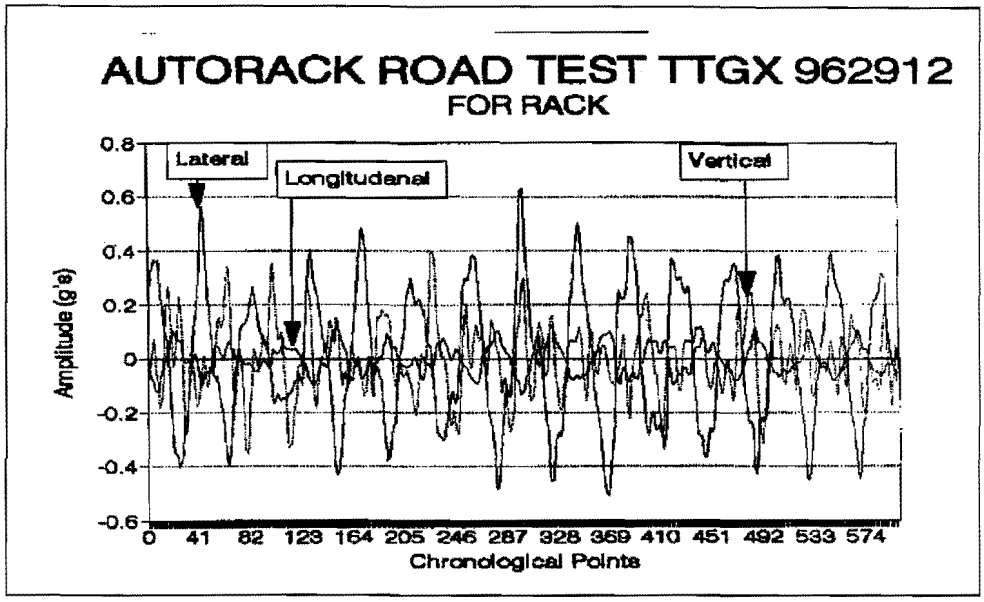


Figure 54

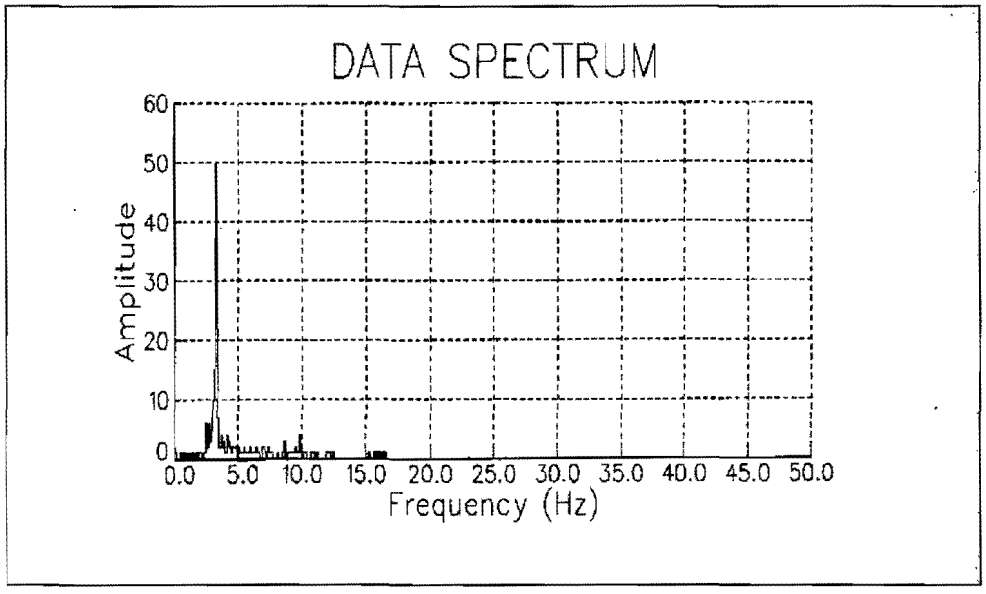


Figure 55

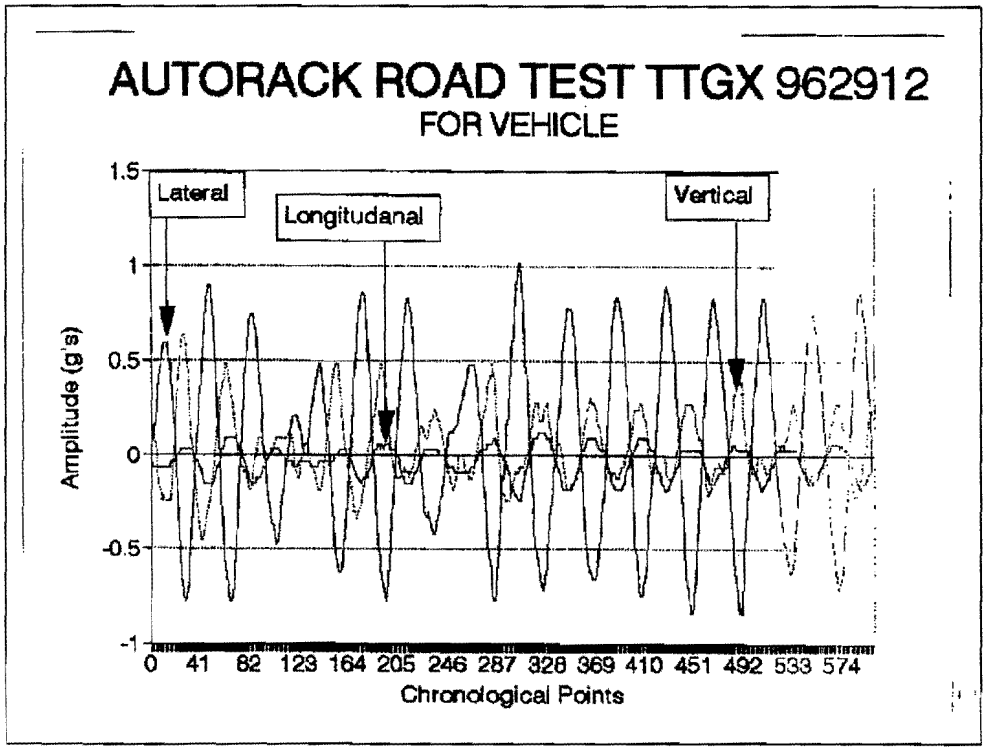


Figure 56

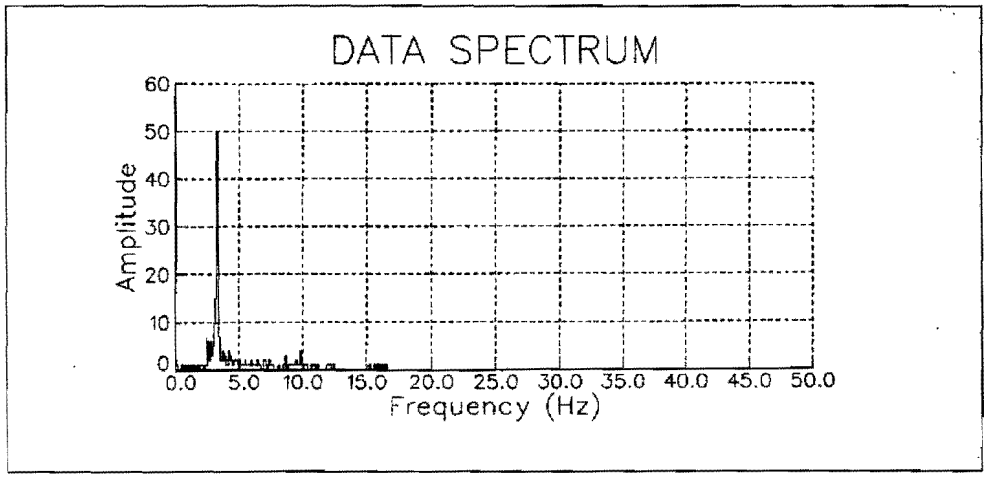


Figure 57

V Research Conclusions

A Signature Identification

This research has demonstrated that digital signal processing can be used to identify characteristics of truck behavior including both appropriate responses to lateral disturbances and hunting as a function of speed, wear and tear, and other factors. Signature analysis functions include Fast Fourier Transform analysis and several calculations including slope, amplitude, and duration of the root mean squares of the peak amplitudes of the lateral accelerations.

It has thus demonstrated the feasibility of developing an inexpensive truck-based vibration monitoring system. In addition to providing valuable real time data regarding hunting, this system could also monitor other conditions which have identifiable vibration signatures. Such conditions include bearing failures, slid flat spots, and track defects. As the experience with this system grows, IEM believes that this system will have the potential for providing a primary input to the scheduling of track maintenance work, wheel work, and truck overhaul work.

For example, by monitoring lateral accelerations due to track inputs and incorporating geographic location data, the system will have the capability of identifying those track sections causing the most serious disturbances. Conditions that may be detectable will include switch point wear and the deterioration of sub-surface ballast conditions. Additionally, by comparing truck performance both over time and relative to one another, the system will have the ability to identify those trucks needing periodic maintenance.

The simulation and verification work described above provides ample evidence that a truck-based acceleration monitoring system is capable of performing both functions described in the Phase I solicitation. It can reliably detect the onset of hunting as well as performing a screening function to provide an early warning system identifying trucks with a propensity to hunt.

B The Proposed Hunting Detection System

The proposed Hunting Detection System would consist of a set of vertical and lateral accelerometers on each truck. Each set of the two sets of accelerometers would be hooked up to a digital signal processor which would provide a preliminary screening of the data for disturbance signatures consisting of combinations of amplitude, duration, slope and frequency. As disturbances are detected, they are reported to a central processing unit for the train where they are recorded and matched against a geographic data base. Then, as necessary, appropriate signals are sent to the engineer for remedial action.

The following flow charts describe the operation of the system in three different modes: pure hunting detection, lateral accelerations as a result of random track disturbance or a track disturbance operating without the benefit of a geographic data log of known disturbances, and lateral accelerations resulting from known track disturbances.

The primary engine of the Hunting Detection System is described in the following notes and accompanying flow chart. We then add two variations to the basic model. The first describes an approach which would identify specific types of track disturbances based on pure analysis and real time calculation. The second approach will match transient impacts against a known data base of track conditions which is geographically coded.

C The Basic System

The lateral acceleration data is acquired by using a low frequency, 100mv/G accelerometer capable of detecting down to 0.5 Hz. The output from the accelerometer is AC coupled so that any DC bias is not amplified. The accelerometer data is converted to digital information using a 12-bit A/D converter. The sampling frequency of the data is currently set at 100 Hz to adequately capture any frequencies up to about 40 Hz. However, the input signal is processed through an anti-aliasing filter set at 40 Hz. The input data sample is stored in the memory for further processing.

The input data is processed to compute a discrete Fourier transform (DFT or FFT). The spectra are divided into 80 lines to obtain a 0.5 Hz resolution/line. Therefore, the initial data acquisition time to obtain the finest resolution sine wave will be two seconds (1/resolution). The FFT is performed using a fast digital signal processor (DSP) capable of producing the FFT in less than 0.2 milliseconds[6]. Using the FFT information, one can also compute the spectral density information.

In order to improve the performance associated with FFT, at least four FFT results will be averaged to produce an average FFT. However, to cut down on the total time required to obtain four FFTs, a moving data window scheme will be used. The moving data window will allow us to keep most of the last data set and update only a small part before computing the FFT.

Hunting detection is based on looking for a certain response level in the average FFT spectra. The frequency band currently being investigated covers from 1Hz to 5Hz. So, if we detect abnormally high response in this region, the system will register that the hunting mode has been detected.

Every time hunting is detected, the system records the event in a running log. (As we develop experience, we believe that through the use of artificial intelligence algorithms and/or expert system technologies, we will be able to refine the functions which identify and classify hunting events). The system also reports the event to the central controller so that appropriate actions are taken by the driver. The FFT spectra may also show other frequency responses such as bearing defects or flat spot defects. The bearing defects typically range from 14.6 Hz @ 20 MPH to 29.1 Hz @ 40 MPH[7]. Other types of bearings may show higher frequency responses. Fortunately, a flat spot p roduces a clang every time the wheel rotates through assuming that there is typically one flat spot per wheel. At speeds of 95 MPH with 27 inch wheel, the flat spot's frequency is around 19.7 Hz[8]. Also, the flat spots will produce continuous clangs making it differentiable from hunting spectra.

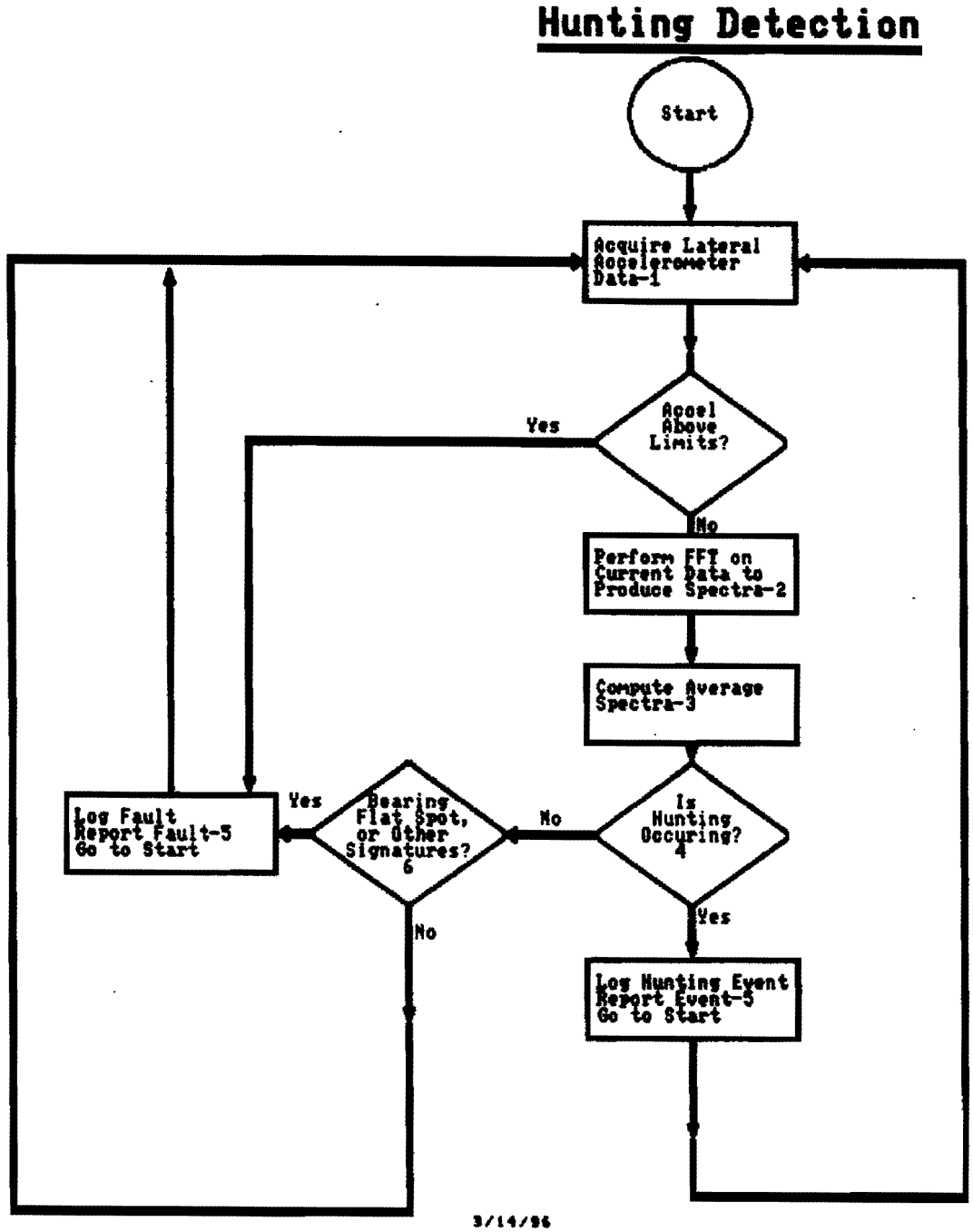

Hunting Detection
Start
Acquire Lateral Accelerometer Data-1
Accel Above Limits?
Yes
No
Perform FFT on Current Data to Produce Spectra-2
Compute Average Spectra-3
Is Hunting Occuring? 4
No
Yes
Bearing Flat Spot, or Other Signatures? 6
Yes
No
Log Fault Report Fault-5 Go to Start
Log Hunting Event Report Event-5 Go to Start
3/14/96

Figure 58

D Model Showing Location Independent Track Disturbance

The average spectra will be used as a signature corresponding to any track disturbances in the last n seconds during which the data was taken.

In order to produce a more robust diagnostic system, we will limit the different possible spectra to only the conditions which we are interested in. These conditions include speed in excess of the minimum speed at which hunting or desired response occurs, response from a working accelerometer, vehicle not on a curve, lateral acceleration above a minimum level, and adequate signal to noise ratio. The actual limits assigned to these conditions will be determined after the field data is available.

Considering all the factors that can change a lateral disturbance effect on a truck, the system cannot rely on the absolute lateral acceleration levels. Therefore, every time domain signature will be brought to the same level, normalized, using the absolute lateral response level produced by the first two transients contained in the time domain data. Also, a transient definition table containing salient features of the transient will be computed. Some of these features include: total time for the transient to reach certain level, slope of rate of transient decay/increase, peak transient levels, and number of cycles contained in the transient signature.

This step compares the current signature against existing signature templates in the system. The matching algorithm will be based on spectral comparison (frequencies and their magnitudes), time domain definition table comparison, and dynamic template matching (shrink/expand signature until it matches). Signature matching is a well-known science in the human voice processing area[9].

If a match is found in the existing template collection, the current signature is understood.

If a match is not found, then the current raw signature is compared against the signatures of other trucks acquired at the same time by the same disturbance. This process of relative comparison provides a convenient measure of current truck's response as compared to the other trucks in the consist. For example, if a truck produces much higher levels of G forces while other trucks do not, the algorithm suggests that we ought to look at that truck.

If a match is found in the template table, the current response is classified as a certain defect.

A relative comparison at the template level is performed to cross check if all trucks behaved somewhat similarly to the same disturbance. For example, all trucks should indicate a lateral disturbance of 0.4" in response to the same disturbance. But if a truck response classifies as 1.0" lateral disturbance, then a report will be generated indicating the possibility that some of the suspension elements are worn out.

This step performs a simple comparison based on responses from all the trucks in the consist. The algorithm is based on population distribution and detecting outliers.

Every time hunting is detected, the system records the event in a running log. The system also reports the event to the central controller so that appropriate actions are taken by the driver.

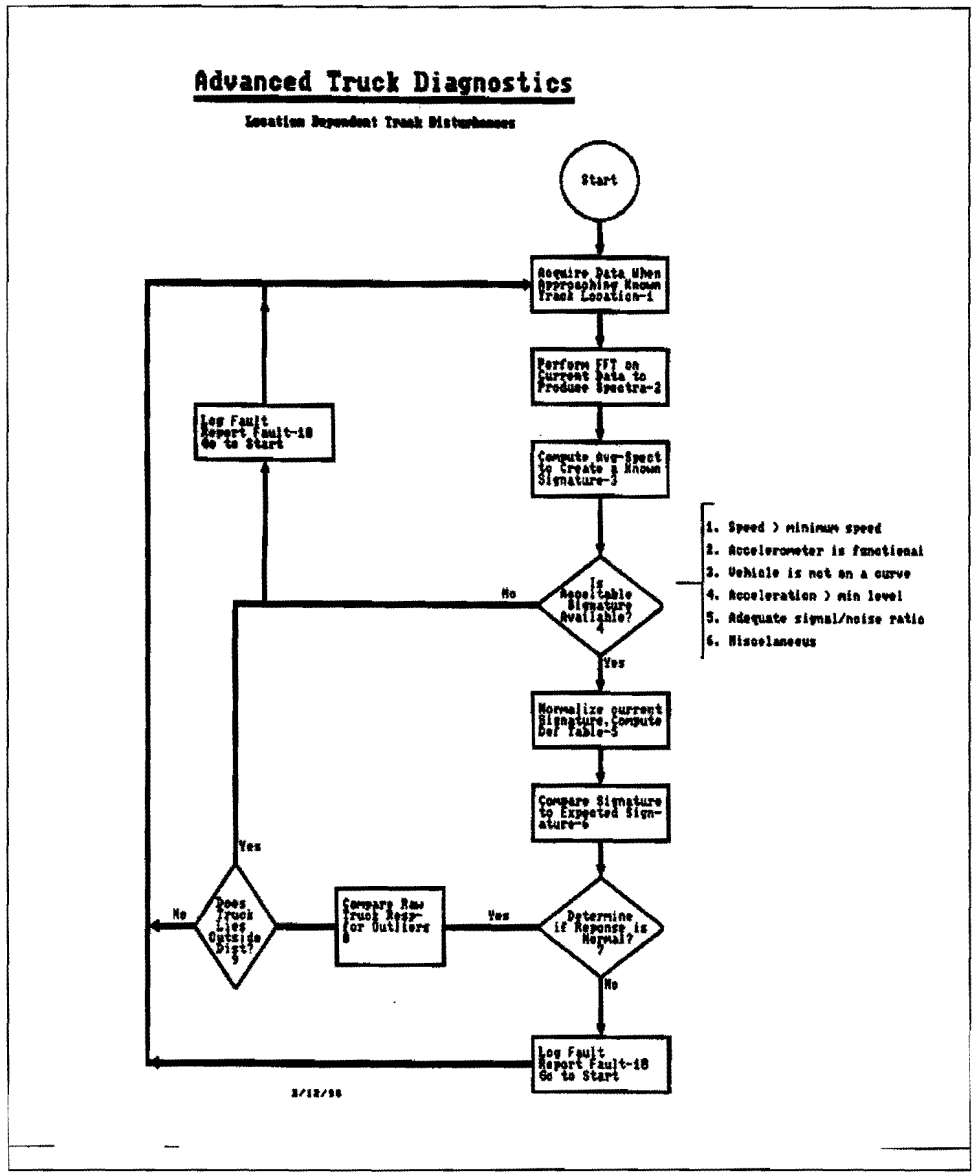


Figure 59

E Model Showing Location Dependent Track Disturbance

The actual data is not acquired until the train approaches a known location on the track. This known location may be a previously geocoded switch point capable of producing a known response.

This step compares the current signature against existing signature templates in the system. The matching algorithm will be based on spectral comparison (frequencies and their magnitudes), time domain definition table comparison, and dynamic template matching (shrink/expand signature until it matches).

If a current signature is well within the expected response at that location, the response is considered normal.

If current response is normal, then the current signature is compared against the signatures of other trucks acquired at the same time by the same disturbance. This process of relative comparison provides a convenient measure of current truck's response as compared to the other trucks in the consist. For example, if a truck produces much higher levels of G forces while other trucks do not, the algorithm suggests that we ought to look at that truck.

This step performs a simple comparison based on responses from all the trucks in the consist. The algorithm is based on population distribution and detecting outliers.

Every time hunting is detected, the system records the event in a running log. The system also reports the event to the central controller so that appropriate actions are taken by the driver.

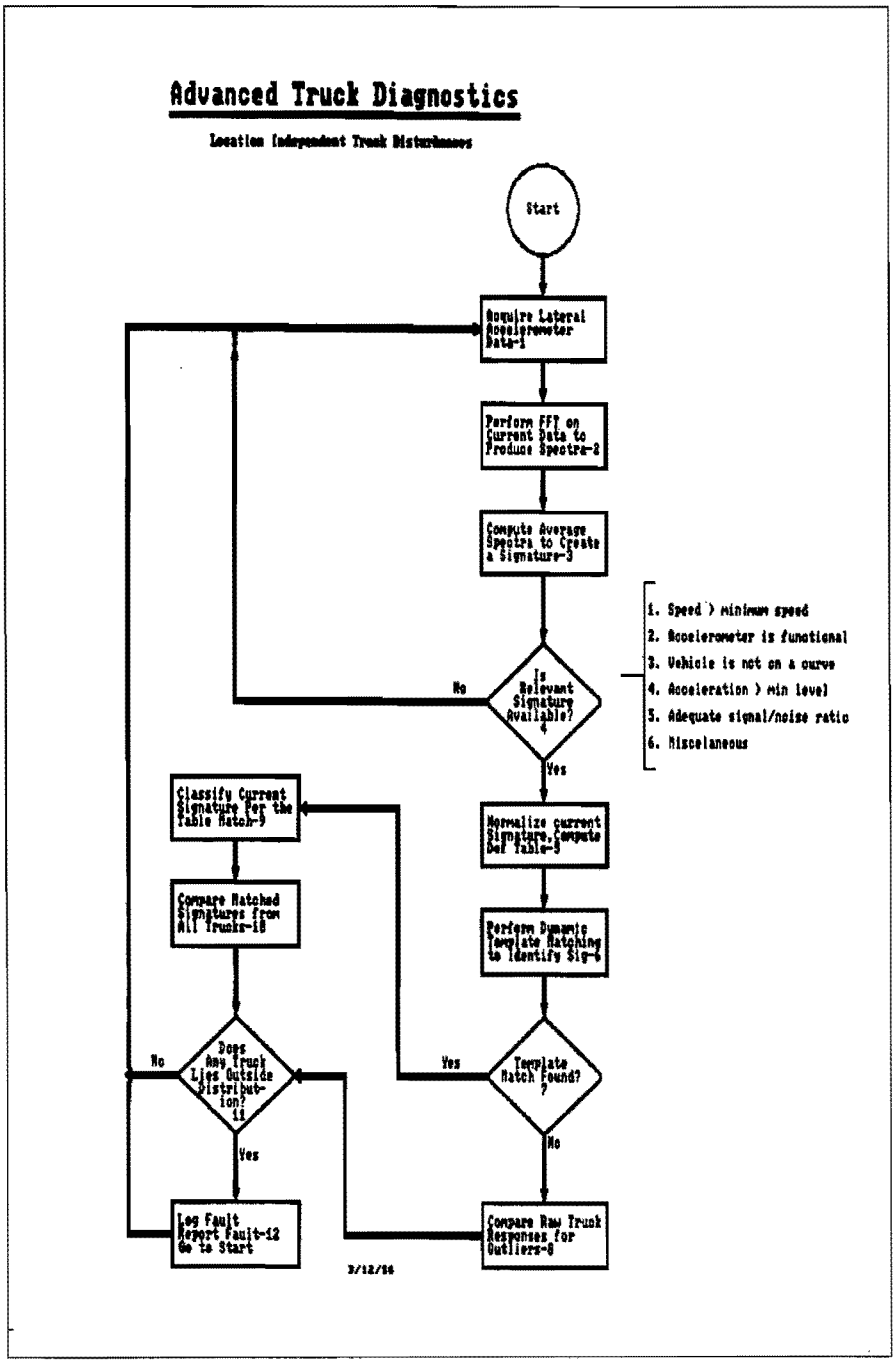


Figure 60

However, we have not yet addressed effective mitigation strategies once the onset of hunting is detected. During our simulation work and our market research, we identified three effective approaches to controlling the onset of hunting in real time. They are:

F Mitigation Strategies

i Deceleration
ii Changing the Coefficient of Friction
iii Actuating an Active Suspension System.

1 Deceleration

Deceleration is the most common approach used to stop hunting. However, it may not be the most effective approach. Hans True, *op cit.,* reports that the speed to which you must decelerate to stop hunting once it has begun is below the speed at which it starts. Consequently, a mitigation strategy based on deceleration may not be optimal. (We did not have the opportunity to model or test the impacts of micro changes in speed on hunting onset conditions.)

2 Top of Rail Lubrication

Another approach which holds promise is the application of top of the rail lubrication. IEM is currently working on a major research project for the New York State Energy Research and Development Authority on the development of an applicator for a top of the rail lubrication product. It has long been a common experience in the freight industry that the incidence of hunting is reduced during precipitation. The application of a top of rail lubricant builds on that experience.

We have found that application of a limited amount of lubricant to control the coefficient of friction between the wheel and the rail between 0.1 and 0.2 has the effect of increasing the critical speed at which hunting occurs for any given truck design and reduces the amplitude of the lateral accelerations once hunting begins. Figure 61 shows the effect of a 0.4 inch lateral disturbance at 50 mph with no lubrication; figure 62 shows the effect with lubrication reducing the coefficient of friction by 50%. The charts show that a reduction of 50 to 75% in the coefficient of friction reduces the peak amplitude of the lateral oscillations by approximately two thirds and reduces the residual amplitudes by approximately one third.

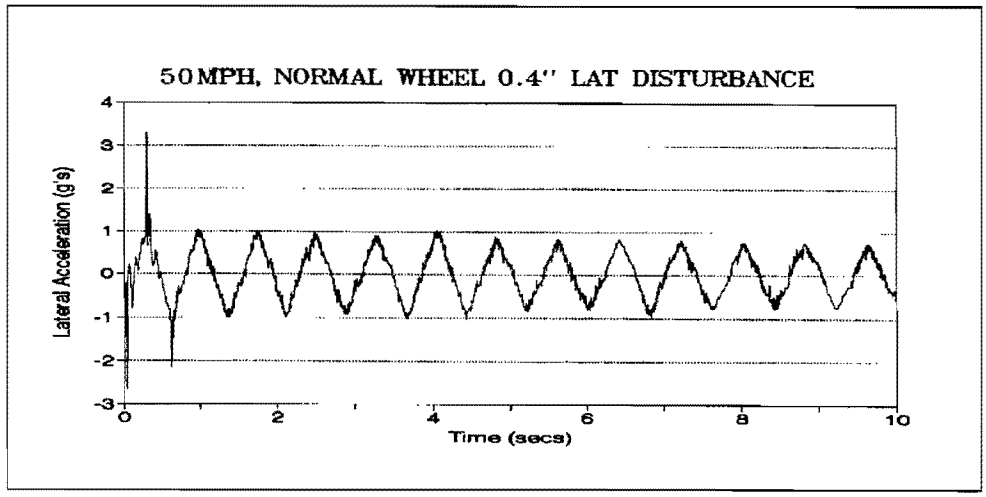


Figure 61

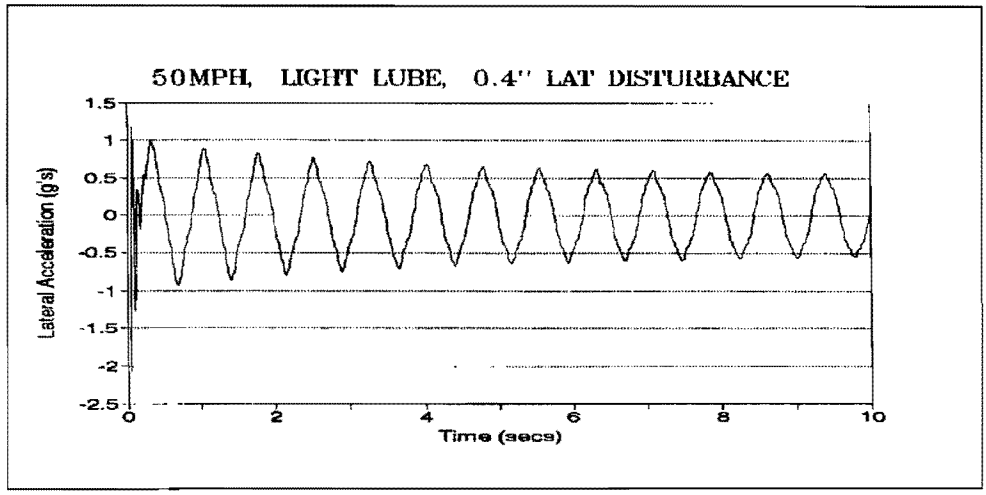


Figure 62

3 Active Suspension

Another promising technology that can be brought to bear on the problem of hunting mitigation is the use of electrically actuated hydraulic suspension. The proposed hunting detection system could be used as an input to the brains of such a smart suspension which could be adjusted to be stiffer during travel on tangent track, loosened on curves, and stiffened again in response to the onset of hunting.

G Responsiveness of the Work Program

In our original proposal, we posed a series of questions that the research program must address. Here are the original questions and our conclusions based on the research results.

1 Can a vibration analysis be used to identify the onset of hunting? *Yes. Clearly the spectral analysis shows that hunting has a very clear and reliable frequency which can be detected in real time.*

2 Is vibration analysis the best approach or are there more practical approaches? *Vibration analysis is more reliable than other approaches.*

   i For example, do we need to measure changes in the suspension load levels? *Measuring changes in suspension load levels adds no significant value to the monitoring of lateral acceleration.*

3 What measurements should be taken? *Monitoring lateral acceleration is sufficient to monitor hunting; however, the addition of a vertical acceleration monitor would add a valuable ride quality monitoring capability.*

4 What is the minimum number of inputs necessary to accurately and reliably detect the onset of hunting? *One input per truck would be sufficient.*

5 Is lateral acceleration measurement sufficient? *Yes.*

6 Is a yaw measurement sufficient? *Not necessary if a lateral acceleration measurement is taken.*

7 Can including additional measurements such as vertical or horizontal acceleration provide useful, cost-effective, reliable data regarding initial onset levels? *See above, vertical acceleration is useful in monitoring ride quality.*

8 What is the optimal location for the sensors:

   i Truck side frame? *Yes.*
   ii Axle end cap? *No.*
   iii Adjacent to center pin? *No.*
   iv Bolster? *No.*
   v On drive shaft or transmission for powered trucks? *No.*

9 What specific sensors are available or will have to be developed to capture the required data? It will be necessary to develop new sensors specific to the application.

10 What data processing capability will be required to analyze the data in real time? *IEM has identified a fast digital signal processor (DSP) capable of producing the FFT in less than 0.2 milli-seconds (ADSP-2181-256 point complex FFT, ANALOG DEVICES 9/94).*

11 Is it sufficient to monitor the magnitude of the time domain vibration data or is it necessary to conduct a spectral analysis of the time domain to interpret the frequency content of the vibration data? *Both measurements are necessary.*

12 What is the set of input readings which should produce an alert? *Alerts should only be triggered when a combination of amplitude, duration, and slope of the time domain data coupled with the hunting frequency in the spectral analysis data indicate that hunting is present, is severe, and is growing to an unsafe or uncomfortable condition.*

13 What is the optimal location for the sensors?

i car body? *No.*
ii truck? *Yes.*
iii both? *No.*

14 What are the design questions that need to be addressed in sensor design and location regarding:

i durability
ii complexity
iii power supply
iv communications
v calibration
vi maintainability
vii etc? *It is apparent from an review of the available sensors in the marketplace that a custom sensors suitable for the environmental strains of this application will have to be developed.*

VI The Need for Future Research

A Additional Data Analysis

The work that was performed during this Phase I research program relied heavilly on simulation. The simulation work was then verified through referencing actual data acquired during some test runs of AMTRAK's track geometry car and an analysis of some freight car data acquired by TTX. The results of this research merely demonstrate the robustness of the approach. However, additional data needs to be gathered from an assortment of AMFLEET trucks in service to provide additional depth to these conclusions.

One weakness of the data that was gathered from the field tests was that the track geometry car is very well maintained with excellent wheel profiles. While this data was useful in developing the signature of the response of a well-tuned truck to a lateral disturbance and to demonstrating the Null Hypothesis, it did not provide a valid confirmation of the hunting signatures. To obtain this level of confirmation, we would have to acquire much more data including data from trucks which are displaying a tendency to hunt. One approach to acquiring this data would be to equip multiple trucks in a given train (preferably one with freshly trued wheels and one with worn wheels) with the prototype system so that comparisons could be made in terms of each truck's ability to dissipate lateral impacts at various speeds.

We could also incorporate speed data and geographic data into the real time recording device to add those inputs to the analysis. Acquiring good data has been a major challenge throughout this project. Additional data has recently become available from the AAR's Transportation Test Center. We have not had an opportunity to analyze this data. Doing so would be valuable.

B Sensor Design

There is no commercially available accelerometer capable of providing the required data while surviving, maintenance-free, in the railway environment. IEM has developed initial plans for the design of a smart accelerometer designed for railway service. Central to the design of this sensor would be an automated self-calibrating feature.

The calibration of accelerometers can degrade due to three factors: offset drift, gain drift, and mechanical properties drift. The primary source of stress which causes this drift is changes in temperature. To compensate for these drift characteristics, IEM has developed a sensor design which incorporates a built-in ground switch. When the switch is set, the acceleration level should be 0.0 g's. If it is not so, a built-in calibration program can reset the sensitivity of the sensor back to 0.0. This test would check for offset drift.

Gain drift could be self calibrated by applying a known voltage to the system electronics to stimulate a known output. To the extent that output varies, a self correcting algorithm could be applied.

Mechanical properties' drift could be checked and corrected by incorporating a built-in acceleration stimulus (hammer) into the sensor. Activating this trigger should produce a known response. Correction algorithms could operate in the manner described above.

Additionally, a quick field test could be to check for 0.0 readings when the train is stopped on flat ground with no ambient vibrations. Finally, IEM realizes that whatever sensor is developed for the railways, it must be durable, inexpensive, and easy to install and change quickly in the field without resorting to specialized tools or equipment. IEM has established a working relationship with the Vibrametrics Corporation to develop these sensors should additional research funding become available. Prototype models of this sensor should be developed and tested in Phase II.

C Power Spectral Densities

During the Phase I program, we began to do some work to incorporate an analysis of the density of the frequency spectra. The density of the spectra incorporates an analysis of the area beneaththe spectrum curve. This is as opposed to simply charting the peak spectral values. The utility of the power spectral density analysis is that for maintenance management purposes it may provide a more reliable indicator of a maintenance problem than an analysis of the spectral peaks. More work needs to be done in this area.

---

1. Development of Advanced Suspension Systems for High Speed Railcars - The Metroliner, A Case Study. Francis E. Dean

2. Advanced Notice of Proposed Rulemaking (ANPRM) on rail passenger equipment safety standards. Paragraph F item 7.

3. Batchelor, G.H.; Stride, R. C. T.; "Hydraulic Dampers and Damping, Journal of the Institution of Locomotive Engineers"; Paper presented to the Midlands Centre of the Institution of Locomotive Engineers on 22nd January 1969 at Midland Hotel, Derby

4. DYNSIM, A software Package for Computer-Aided Railroad Equipment Design Projects. George F. List, Harold A. List

5. Classifying Track by Power Spectral Density, John C. Corbin and William M. Kaufman

6. ADSP-2181-256 point complex FFT, ANALOG DEVICES 9/94

7. On-line Acoustic Detection of Bearing Defects, Joe Bambara, U.S. Patent # 4,790,190, Servo Corporation.

8. Techniques for Detecting Flat Wheels..., Frank Svet, U.S. Patent # 4,129,276, GRS Corp.

9. Sadaoki, Furui; "Speaker-Independent Isolated Word Recognition using Dynamic Features of Speech Spectrum"; IEEE Transaction on Acoustics, Speech, and Signal Processing; Vol. ASSP-34, No. 1, February 1986, pp 52-58.

References

Allen, R. A.; Elkins, J. A., Dincher, J. T., and Leary, J. F.; "Effect of Truck Performance on Train Resistance", Association of American Railroads, pp 103-127

Anon; "Truck Hunting Accelerates Wear", Railway Locomotives and Cars; v 147, n 4, May 1973; pp 7-10

Batchelor, G.H.; Stride, R. C. T.; "Hydraulic Dampers and Damping, Journal of the Institution of Locomotive Engineers"; Paper presented to the Midlands Centre of the Institution of Locomotive Engineers on 22nd January 1969 at Midland Hotel, Derby

Cooperrider, N. K.; "The Hunting Behavior of Conventional Railway Trucks", Transactions of the ASME, May 1972; pp 752-762

Corbin, J. C., Kaufman, W. M.; "Classifying Track by Power Spectral Density", ENSCO, Inc., Springfield, Virginia

Elkins, J. A., Allen, R. A.; "Verification of a Transit Vehicle's Curving Behavior and Projected Wheel/Rail Wear Performance", Journal of Dynamic Systems, Measurement, and Control; September 1982, Vol. 104, pp 247-255

Hawthorne, V. T.; "Method of Eliminating Truck Hunting in Railway Trucks", US Patent 4,393,957; July 19, 1983

Kumar, S., Kim, J, S., Prasanna Rao, D. L., Qian, Lixin,; "Effect of Kinematic Oscillation on Tractive Characteristics of Steel Wheel on Rail", Journal of Engineering for Industry; May 1983, Vol. 105; pp 61-63

List; "PC DYNSIM, a Software Package for the Evaluation of the Dynamic Behavior of Railway Vehicles", Railway Engineering Associates, Inc.

List, G.; "Dynamic Simulation of Railroad Vehicles on a Microcomputer", Department of Civil Engineering, Rensselaer Polytechnic Institute, Troy, NY 12180, U.S.A.

List, G., List, D. A.; "DYNSIM, A Full-Featured Simulation Package for Analyzing the Vehicle Dynamics of Railroad Locomotives and Cars"

List, G., List, H. A,; "DYNSIM, a Software Package for Computer-Aided Railroad Equipment Design Projects', The American Society of Mechanical Engineers, Report 83-WA/RT-3

List, H. A.; "An Evaluation of Recent Developments in Rail Car Truck Design", Transactions of the ASME, 71-RR-1

List, H. A., Caldwell, W. N., Marcotte, P.; "Proposed Solutions to the Freight Car Truck Problems of Flange Wear and Truck Hunting"; ASME Journal 75-WA/RT-8

Marcotte, P, Caldwell, W. N., List, H. A.; "Performance Analysis and Testing of a Conventional Three-Piece Freight Car Truck Retrofitted to Provide Axle Steering", Journal of Dynamic Systems, Measurement, and Control; March 1982, Vol, 104; pp 93-99

Petersen, K., Christel, L., Pourahmadi, F., Fathi, Y., Bryzek, J., NovaSensor, L.,; "Acceleration Sensors for Railway Diagnostics", Sensors, February 1993; pp 11-13

Railway Engineering Associates Inc. "A Review of the ASF/TTX Testing of an 89' Flat Car on 39' Stagger Jointed Track at Pueblo in May, 1990"

Reynolds, D. J.; "Hunting in Freight Cars (A Brief Description of Cause and Cures, with Some Results of Road Tests of Various Modifications)", ASME Report 74-RT-2

Sadaoki, Furui; "Speaker-Independent Isolated Word Recognition using Dynamic Features of Speech Spectrum"; IEEE Transaction on Acoustics, Speech, and Signal Processing; Vol. ASSP-34, No. 1, February 1986, pp 52-58

True, H; "Does a Critical Speed for Railroad Vehicles exist?", ASME/IEEE Joint Railroad Conference, 1994

True, H; "Asymmetric Hunting and Chaotic Motion of Railroad Vehicles"; ASME/IEEE Joint Railroad Conference, 1992

Yokose, K.; "Calculation on Hunting of High Speed Railway Truck - Problems of Truck Design for SANYO SHIN KANSEN", Quarterly Reports, Vol. 11 No. 2, 1970; pp 108-113

Yokose, K., Igarashi, M., Takayanagi, J.; "Fundamental Study on Truck Hunting Considering Non-Linearity of Creep Force", Bulletin of JSME; Vol. 29, No. 28, February 1986, pp 541-547

Appendix A Diagnostic Instruments PL22 FFT Analyser/DSO Specifications

INPUT

No of channels	2
Voltage ranges	+/- 5mV to +/- 5V p-p (autoranged or selectable in 1,2,5 sequence)
Max Input voltage	> +/- 18V
Transducer interface	4mA current source in-built on each channel for standard accelerometer (15V source)
Scaling	Scaling to engineering units available
Frequency ranges	DC to 25Hz through DC to 20kHz in 9 ranges (selectable in 1,2,5 sequence) with anti-aliasing filter 0-40kHz without anti-aliasing filter
Input sensitivity range	130dB
Input dynamic range	70dB (12-bit A/D converter)
Input filter	25Hz to 20kHz low pass per channel (selectable in 1,2,5 sequence of off) > 130 dB/octave roll-off
Input impedance	1MOhm
Input coupling	DC, AC, accelerometer or ground
Sampling rate	102.4kHz per channel maximum (coupled to frequency range)
Sampling rate/bandwidth ratio	2.56:1

PROCESSOR

Memory size	256 kbytes of non volatile SRAM (1M byte optional)
Data record length	256 to 4096 points selectable in binary steps
No of set-up and storage memories	Up to 600, user definable
Process	Selectable by menu
Post Processing	On last record acquired or any stored time domain record
User interface	8-way function keypad with menu interface to screen Cursor keypad Numeric input keypad
Real-time clock	Indication of seconds, minutes, hours and date on stored readings

DISPLAY

Resolution	256x128 pixels
Type	High contrast LCD alphanumeric/graphic
Display format	One/two channel graphic display
	Continuous scan of displayed data by cursor
Vertical magnification	By cursor control on processes
Horizontal scroll	Horizontal scroll by key control
Text information	Channel X-Y data displayed to full resolution at cursor position
Harmonic cursors	20 harmonic cursors available

TRIGGER

Control Modes	All menu selectable
	Normal
	Free-run
	Single shot
Trigger point	Normal
	Pre- or post-trigger, set by no of points after/before trigger point
	Channel B delayed
Level	Variable with resolution of 1 part in 100, + or - slope
Trigger source	Internal or external
Sample clock source	Internal or external

SELECTABLE PROCESSING

Time Domain Processing

Channel Data	Channel A data
	Channel B data
	Channel A and B data
Channel A - Channel B	Differential input
Differentiation	Channel A or Channel A and B
Integration	Channel A or Channel A and B

Spectrum Analysis

Autospectrum	100 to 1600 lines resolution
	Log or linear amplitude display
	Amplitude or amplitude and phase display
Cross-spectrum	as above
1/3 Octave Analysis	15 or 31 band selectable display
1/1 Octave Analysis	5 or 10 band display

Correlation

Autocorrelation	Windows-see below
Cross-correlation	Windows-see below

Other Processes

Coherence	100-800 lines, windows-see below
Transfer Function	100-800 lines, windows-see below
RMS Calculation	Variable over frequency band selected
Spect + T	Time domain displayed with spectrum equivalent
Orbit	CHA versus CHB in time domain

Process Windows

Process	Rectangular, Flat-top, Hanning, Hamming or Force/Expontential
Averaging	Number selectable between 2 and 4096 Linear, Exponential and Peak Hold available

INTERFACES

RS232	Dump display to graphics printer (Epson type) or HPGL plotter Transfer file to host computer (Variable baud rate to 38400)

GENERAL

Battery type	NiCd rechargeable (internal)
Time between charges	5-6 hours continuous running. Auto shutdown facility
Memory back-up	By Lithium cell up to 10 years
Weight	4kgs approx
Size	256(W) x 250(H) x 82(D) mm
Operating temperature	0 to 50 degrees Celsius

ANSI BAND NO	1/3 OCTAVE CENTRE FREQ (Hz)	OCTAVE CENTRE FREQ (Hz)	A WEIGHTING (dB)
7	5		-*
8	6.3		-*
9	8	8	-*
10	10		-70.4
11	12.5		-63.4
12	16	16	-56.7
13	20		-50.5
14	25		-44.7
15	31.5	31.5	-39.4
16	40		-34.6
17	50		-30.3
18	63	63	-26.2
19	80		-22.5
20	100		-19.1
21	125	125	-16.1
22	160		-13.4
23	200		-10.9
24	250	250	-8.6
25	315		-6.6
26	400		-4.8
27	500	500	-3.2
28	630		-1.9
29	800		-0.8
30	1000	1000	0
31	1250		+0.6
32	1600		+1.0
33	2000	2000	+1.2
34	2500		+1.3
35	3150		+1.2
36	4000	4000	+1.0
37	5000		+0.5
38	6300		-0.1
39	8000	8000	-1.1
40	10000		-2.5
41	12500		-4.3
42	16000	16000	-6.6
43	20000		-9.3

*The 'A' weighting attenuation for these frequency bands is greater than the dynamic range of the PL22.

File: huntmain.c

Wheel Hunting Data Acquisition Software

```
////////////////////////////////////////////////////////////
// File: huntmain.c                                       //
//                                                        //
// Wheel Hunting Data Acquisition Software                //
// Main program                                           //
//                                                        //
// Copyright 1995: IEM Corporation                        //
//                                                        //
// Proprietary Licensed Information.                      //
// Not to be duplicated, used or disclosed                //
// except under terms of license.                         //
//                                                        //
// Revision History:                                      //
////////////////////////////////////////////////////////////

#include "hunthead.h"   // Compiler includes
#include "huntstru.h"   // Constant and structure declarations
#include "huntllb.h"    // Function Prototypes
#include "huntmain.h"   // Global declarations, initialization

#include <ctools.h>

#undef   BLACK                          // avoid redef warnings
#undef   RED
#undef   WHITE
#include <graphic.h>                    // graphic declaration file

extern  char *neta(char *a1, char *a2);    // Function prototypes
void    display_help(void);
long    getspeed(void);
extern  long reada2d(int channel);
void    timer_init(void);
int     xlfit(int NPT);

float   profile_length = 4.51f;
float   y_travel = 1.5f;                                // default is 1.5 high
 flange
float   xcord_fnt;                              //local global
float   huge x[MAX_I_POINTS], y[MAX_I_POINTS];
float   huge x_spectrum[MAX_I_POINTSHALF], y_spectrum[MAX_I_POINTSHALF];

#if DELTA_DESIRED
float   huge d[MAX_I_POINTS], z[MAX_I_POINTS];
#else
float   huge d[1], z[1];
#endif

int             drive = 0;                                              // di
sk drive is C by default
float   accel_min_level = 0.2f, accel_sustain_level=0.2f;
int             startspeed = 30;
int             useaccel_2 = FALSE;
int             spectrum_flag = TRUE;
int             keep_data = FALSE;
int             show_bins = FALSE;
int             desired_freq = 15;
int             amplitude_required = 20;                // bin height
int             frequency_cutoff=10;                            // highest freque
ncy of interest
int             continue_count=-1;
int             autopilot = FALSE;
float   scale_factor = 0.00244;                 // default scale factor
long    oldspeed = 0;

#if PC_SIM  == 1
long    xpos_sim = 0, ypos_sim = 0;
```



```
#endif

int     wait_time = 0;                                  //a2d wait time default
int plotdata = FALSE;
int     low_range = FALSE;                              // limit spectrum to 50Hz
 in display spectrum
int     second_accel_desired = FALSE;   // take one accel in record. dummy in 2nd
int show_output = FALSE;

void    main(int argc, char *argv[])
{
    struct  profile *current_struct, *default_struct;
    FILE    *error, *stream;
    _short_ curno, line_cnt;
    boolean done, dir_display;      /* mainline execution finished */
    char    buf[80], buff[80], ch;
    int         i,key, ret , j, k, l;
    boolean     color_flag;
        char *scr_file[6];              //Pointer to a string of characters
        float   rmsvalue, maxy;


    /* screen definition filenames  */

    static char *scr_file_pc[] = {
        "c:\\hunting\\main.scr","c:\\hunting\\default.scr","c:\\hunting\\record.s
cr"
    };


        // set the flag defaults

    color_flag = FALSE;
        keyboard_flag = TRUE;
        speed_trigger = FALSE;          // default is not trigger on speed


    /* Allocate memory */

        for(i=0; i<3; i++){
            if((scr_file[i] = (char *)malloc(30))==NULL){
                printf("\nNot Enough Memory - Check System");
            getalpha();
                goto home;
        }
        }
    if((current_struct = (struct profile *)malloc(sizeof(struct profile)))==NULL)
{
        printf("\nNot Enough Memory - Check System");
        getalpha();
        goto home;

    }

    if((default_struct = (struct profile *)malloc(sizeof(struct profile)))==NULL)
{
        printf("\nNot Enough Memory - Check System");
        getalpha();
        goto home;

    }

    /* Scan command line for all flags/files on the command line */

    while ( --argc > 0 )
    {
```

```
        ++argv;
        if ( ( ch = **argv ) == '-' || ch == '/' )
        {   /* we are processing a flag */
            while ( ch = *(++(*argv)) )
            {
                switch( ch )
                {
                                case '?':
                case 'h':
                case 'H':
                        display_help();
                                goto home;

                                case 's':                                       // start
speed
                                case 'S':
                                        startspeed = (*(++(*argv)) - '0')*10;
                                        break;

                                case 'w':               // wait time after a2d to
 settle
                                case 'W':
                                        wait_time = (*(++(*argv)) - '0')*10;
                                        break;

                                case 'm':                                       // minimu
m accel level
                                case 'M':

                                        accel_min_level = (*(++(*argv)) - '0')* 0
.02f;
                                        break;          // same as 0.2 g

                                case 'c':
                                case 'C':                                       // contin
ue accel level
                                        accel_sustain_level = (*(++(*argv)) - '0'
)* 0.02f;
                                        break;          // same as 0.2 g

                                case 'd': /* debug flag allows screen display of
debug information */
                case 'D':
                    debug_flag = TRUE;
                    break;

                                case 't': // trigger on speed when chosen
                                case 'T':
                                        speed_trigger = TRUE;
                                        break;

                                case 'g': // plot the data on screen while acq
                                case 'G':
                                        plotdata = TRUE;
                                        break;

                                case 'k': // keep the data after acq
                                case 'K':
                                        keep_data = TRUE;
                                        break;

                                case 'u': // use second accelerometer
                                case 'U':
                                        useaccel_2 = TRUE;
                                        break;
```



```
                                case 'p':
                                case 'P':               // show delta y
                                        delta_y_flag = TRUE;
                                        break;

                                case 'b':               // show bins
                                case 'B':
                                        show_bins = TRUE;
                                        break;

                                case 'a':                                       // Amplit
ude
                                case 'A':
                                        amplitude_required = (*(++(*argv)) - '0')
*10;
                                        break;

                                case 'f':                                       // Freque
ncy cutoff in save
                                case 'F':
                                        frequency_cutoff = (*(++(*argv)) - '0')*4
;
                                        break;

                                case 'z':
                                case 'Z':               // keep going for n readi
ngs z*10 times
                                        continue_count = (*(++(*argv)) - '0')*20;
                                        break;

                                case 'y':
                                case 'Y':               // keep going upon startu
p
                                        autopilot = TRUE;
                                        break;

                                case 'r':
                                case 'R':               // low range of spectrum
 is desired
                                        low_range = TRUE;
                                        break;

                                case 'v':
                                case 'V':               // take the 2nd accel dat
a in record
                                        second_accel_desired = TRUE;
                                        break;

                                case 'j':
                                case 'J':               // show printf in record-
F1
                                        show_output = TRUE;
                                        break;

                                case 'x':
                                case 'X':
                                        ch = (*(++(*argv)) - '0');
                                        if(ch == 1)scale_factor = 0.00244f;
                                        if(ch == 2)scale_factor = 0.00488f;
                                        if(ch == 3)scale_factor = 0.0244f;
                                }

                        }
                }
        }

//      timer_init();           // initialize the timer 4/2/96 - using onboard ti
m
```

```
er

        xcord_fnt = 2.7f;

    /* Find out what type of video card is installed. Make choices for
          colors and destination screen address based on video cards      */

        if(_setvideomode(_TEXTC80) == 0) {   /* determine video attributes */
            disp = 1;
            date_frgnd = frgnd = (short)HILITE;
            date_bkgnd = bkgnd = BLACK;
            error_clr = HILITE;
            map_seg = 0xB000;       /* mono screen start address */
        }

        else {
            disp = 0;
            frgnd = BLACK;
            bkgnd = WHITE;
            date_frgnd = (short)_BRIGHTWHITE;
            date_bkgnd = 1;
            error_clr = RED;
            map_seg = 0xB800;       /* typical CGA/EGA/VGA screen start address *
/
        }

    /* Display sign on logo */


    for(i=0; i<3; i++)strcpy(scr_file[i], scr_file_pc[i]);

        TOP_COL  = 10;          // was 20 once
        recd_col = 45;
        L_COL = 21;
        R_COL = 79;
        MSG_R_COL = 79;
        MSG_COL = 21;


    /* Reset screen variables for all other screens */

    _clearscreen( _GCLEARSCREEN );
    _settextcolor( frgnd );
    _setbkcolor( bkgnd );
    outp(0x220+10,0);                           /* A/D CHANNEL ALWAYS 0  220h+10
*/


    done = FALSE;
    curno = 0;
    map_bytes = 4000;       /* typical screens have only 4000 bytes */

    /* Read default parameters from a file */

    if( read_def_values("c:\\hunting\\DEFAULT.SYS", default_struct, &data_smooth_
flag) == FALSE)
        error_out( FILE_READ, "Cannot read DEFAULT.SYS-Check File");


    /* Allocate screen arrays and read screen data from files */

    for(i = 0; i < 3; i++) {

        if((scr[i] = MMALLOC(unsigned char, 4000))== NULL){
            printf("\nNot Enough Memory - Check System");
            getalpha();
```



```
                goto home;
        }

        if((error = fopen(scr_file[i], "r")) == NULL) {
            error_out( FILE_READ, scr_file[i]);
            fclose(error);
                goto home;

        }
        else {
            fread(scr[i], 1, map_bytes, error);
            fclose(error);
        }
}


_setvideomode(_TEXTC80);
_settextcolor( frgnd );
_setbkcolor( bkgnd );


dir_display = FALSE;        /* do not erase screen if dir is displayed */

DISPLAY_SCR(curno);                 /* put up active screen */

/* Execute the main loop */

    if(autopilot == TRUE){
        // copy defaults to current structure

        clear_profile_info(current_struct);
        copy_struct(current_struct, default_struct);

        // Initialize a few other fields

        strcpy(current_struct->date_field, datebuf);
        strcpy(current_struct->time_field, timebuf);

        record_profile(current_struct);
    }

while(!done){

    if(!dir_display)
        DISPLAY_SCR(curno);             /* put up active screen */

    /* reset screen parameters */

    _displaycursor( _GCURSOROFF);        /* shut off cursor */
    _settextwindow(1, 1, 25, R_COL+1);
    _settextcolor( date_frgnd );
    _setbkcolor( date_bkgnd );
    _settextposition(2, 72);
    _strdate(datebuf);
    _strtime(timebuf);
            if(small_display_flag == FALSE)
            _outtext(datebuf);                  /* display date & time */
    _settextcolor( frgnd );
    _setbkcolor( bkgnd );
    _settextwindow(T_ROW, L_COL, B_ROW, R_COL);
    _displaycursor( _GCURSOROFF);
    _wrapon(_GWRAPON);

    key = getfunc();

    /* Main line function dispatch */
```

```
        switch (key){

#if 0
                case KY_F1:                     // Compute spectra of all matchin
g files
                case '1':
                                buf[0]='\0';
                                prompt("Type Match Template: ", ALPHANUMERIC, 8,
buf);
                                if(buf[0] != '\0'){
                                        strcat(buf, ".dat");
                                        _dos_findfirst(buf, _A_NORMAL, &c_file);
                                        if((stream = fopen(c_file.name,"r"))==NUL
L){
                                        error_out( FILE_READ, "Cannot read DATA F
ile-Check File");
                                                goto home;
                                        }
                                    for(ii=0; ii< 10002; ii++) {
                                                _settextposition(10,10);printf("R
#: %5d", ii);
                                                if(feof(stream))break;
                                                fscanf(stream,"%s %s %s %s %s\n",
buffer,dummy,buffer2,dummy,dummy);
                                                if(useaccel_2 == FALSE)
                                                        yaxis_temp = (float)atof(
buffer);
                                                else
                                                        yaxis_temp = (float)atof(
buffer2);
                                                xaxis_temp = (float)ii;
                                        x[ii] = xaxis_temp;
                                        y[ii] = yaxis_temp;
                                }
                                    fclose(stream);
                                        xrealft(ii, sample_freq);

#endif

#if 0
        case KY_F1:         // Speed Data
                case '1':
                        _clearscreen( _GCLEARSCREEN );
                        _settextcolor( frgnd );
                _setbkcolor( bkgnd );
                        while(1){
                                _clearscreen( _GCLEARSCREEN );
                                _settextposition( 5,5);
                                printf("%ld\n",reada2d(0));
                                _settextposition( 6,5);
                                printf("%ld\n", reada2d(1));
                                if(kbhit()==1) break;
                        }
                        break;

                        while(1){
                            _clearscreen( _GCLEARSCREEN );
                        _settextposition(10, 30);
                                printf("Speed: %ld", getspeed());
                                if(kbhit()==1) break;
                        }
                        break;
#endif

        case KY_F2:         // Record Data
```



```
                case '2':

                dir_display = FALSE;
                data_available = 0;

                // copy defaults to current structure

                clear_profile_info(current_struct);
                copy_struct(current_struct, default_struct);

                // Initialize a few other fields

                strcpy(current_struct->date_field, datebuf);
                strcpy(current_struct->time_field, timebuf);

                record_profile(current_struct);

                break;

                    case KY_F3:                 // Zero out encoders
                    case '3':

                                        buf[0]='\0';
                                        prompt("Press Y When Ready: ", ALPHANUMERIC, 1, b
uf);
                                        if(buf[0] == 'Y' || buf[0] == 'y'){
                                            XENC_init();
                                            YENC_init();
                                    // Make this the origin for the encoders
                                    XENC_reset();               // zero out x encoder. ca
n use reset later.
                                YENC_reset();           // loaded with zeros. can use res
et later.
                                        }
                                break;

                    case KY_F4:
                    case '4':
                clear_profile_info(current_struct);
                copy_struct(current_struct, default_struct);
                ret = prompt_for_file(86, current_struct);
                break;

#if 0
                    case 'L':
                    case 'l':
                clear_profile_info(current_struct);
                copy_struct(current_struct, default_struct);
                ret = prompt_for_file(87, current_struct);
                break;
#endif

        case KY_F5:     // Print a profile
                case '5':

                        if( small_display_flag == FALSE){
                    dir_display = FALSE;
                clear_profile_info(current_struct);
                    copy_struct(current_struct, default_struct);
                ret = prompt_for_file(DISPLAY_NPRINT, current_struct);
                        }
                break;

        case KY_F6:     // Display a graph
                case '6':
```

```
            dir_display = FALSE;
            clear_profile_info(current_struct);
            copy_struct(current_struct, default_struct);
            ret = prompt_for_file(DISPLAY_ONLY, current_struct);
            break;

        case KY_F7:    // Change Defaults
                case '7':

            dir_display = FALSE;

            change_defaults(default_struct);
            write_def_values("DEFAULT.SYS", default_struct, data_smooth_flag);
            clear_profile_info(current_struct);
            copy_struct(current_struct, default_struct);

            break;

        case KY_F8:    // List profiles
                case '8':

            clear_text();   /* Clear the text area */
            _settextposition(2, 5);
            _outtext("Please Wait For Directory...");

            /* write directory to file 'tmpdir.$$$' */

            ret = system_call("del TMPDIR.$$$");
            if (ret != 0) break;
            ret = system_call("dir *.prf > TMPDIR.$$$");
            if (ret != 0) break;
            // test ret code

            clear_text();   /* Clear the text area */
            _settextposition(1, 1);

            /* display recently created file */

            if ((stream = fopen("tmpdir.$$$", "r")) != NULL) {
                line_cnt = 0;

                /* dump the first three lines */
                fgets(buff, (R_COL - L_COL + 1), stream);
                fgets(buff, (R_COL - L_COL + 1), stream);
                fgets(buff, (R_COL - L_COL + 1), stream);
                fgets(buff, (R_COL - L_COL + 1), stream);
                while (fgets(buff, (R_COL - L_COL + 1), stream) != NULL) {
                    _outtext(buff);

                    if (++line_cnt == (B_ROW - T_ROW)) {
                        if (keyboard_flag == TRUE) {
                            _outtext("-MORE-Space Bar to Contine,ESC to Quit");
                            while( !kbhit() )
                                ;
                            if(getch() == ESC_key){
                                goto dir_end;
                            }
                            clear_text();   /* Clear the text area */
                            _settextposition(1, 1);
                        }
                        else {
                            _outtext("-- MORE --  Touch Text Area, F10 Key to Qui
t");

                            if(getfunc() == KY_F10)
                                goto dir_end;
                        }
```



```
                        line_cnt = 0;
                        clear_text();   /* Clear the text area */
                        _settextposition(1, 1);

                    }
                }
                if (ferror(stream) != 0) {
                    error_out(FILE_READ, "for List Wheel Profiles");
                    break;
                }
                _displaycursor( _GCURSOROFF);
                _outtext(" -- END OF DIRECTORY -- ");
                fclose(stream);
            }
            else{  // Could not open file
                fclose(stream);
                error_out( FILE_READ, "for List Wheel Profiles");
            }
dir_end:

            if(keyboard_flag == TRUE)
                dir_display = TRUE;
            else
                getfunc();

            if (stream != NULL)
                fclose(stream);
            break;

        case KY_F9:    // List  - add file selection menu
                case '9':

                        buf[0] = '\0';
            if(1                        ){
                dir_display = FALSE;
                prompt("FILENAME?(add extension)",
                FILENAME, FILENAME_LENGTH+4,buf);

                // should beef up this check

                if (buf[1] == ':' ){
                    if ( disk_ok(buf[0]) == FALSE){
                        error_out(0,"Floppy drive not ready");
                        break;
                    }
                }

                /* now check the filename */

                if((access(buf, 0)) != 0 || buf[0] == '\0'){
#if 0
                    _settextposition(MSG_ROW, MSG_COL);
                    error_out(ONLY_MESS,"File Does Not Exist");
#endif
                                        clear_text();   /* Clear the text area */
                                    if(chose_file(buf,"*.*") != 0)break;
                }
                {

                    strcpy(buff, "type ");
                    strcat(buff, buf);
                    strcat(buff, " > TMPDIR.$$$");

                    DISPLAY_SCR( curno );
                    clear_text();   /* Clear the text area */
                    _settextposition(2, 5);
```

```
                    _outtext("Please Wait For Listing...");

                    /* write listing  to file 'tmpdir.$$$' */

                    ret = system_call("del TMPDIR.$$$");
                    if (ret != 0) break;
                    ret = system_call(buff);
                    if (ret != 0) break;
                    // test ret code

                    clear_text();   /* Clear the text area */
                    _settextposition(1, 1);
                    _outtext("FILENAME: ");
                    _outtext(buf);
                    _settextposition(2, 1);

                    /* display recently created file */

                    if ((stream = fopen("tmpdir.$$$", "r")) != NULL) {
                        line_cnt = 0;

                        /* display all lines */

                        while (fgets(buff, (R_COL - L_COL + 1), stream) != NULL)
{

                            _outtext(buff);

                            if (++line_cnt == (B_ROW - T_ROW-1)) {
                                if (keyboard_flag == TRUE) {
                                    _outtext("-MORE-Space Bar to Contine,ESC to Q
uit");

                                    while( !kbhit() )
                                        ;
                                    if(getch() == ESC_key){
                                        goto list_end;
                                    }
                                    clear_text();   /* Clear the text area */
                                    _settextposition(1, 1);
                                    _outtext("FILENAME: ");
                                    _outtext(buf);
                                    _settextposition(2, 1);
                                }
                                else {
                                    _outtext("-- MORE --  Touch Text Area, F10 Ke
y to Quit");

                                    if(getfunc() == KY_F10)
                                        goto list_end;
                                }
                                line_cnt = 0;
                                clear_text();   /* Clear the text area */
                                _settextposition(1, 1);
                                _outtext("FILENAME: ");
                                _outtext(buf);
                                _settextposition(2, 1);

                            }
                        }
                        if (ferror(stream) != 0) {
                            error_out(FILE_READ, "for File Listing");
                            break;
                        }
                        _displaycursor(_GCURSOROFF);
                        _outtext(" -- END OF LISTING -- ");
                        fclose(stream);
                    }
                    else{  // Could not open file
```



```
                            fclose(stream);
                            error_out( FILE_READ, "for File Listing");
                        }
list_end:

                        if(keyboard_flag == TRUE)getfunc();

                        if (stream != NULL)fclose(stream);

                    }
                    _clearscreen(_GCLEARSCREEN );
                }
                break;

            case 'U':   // Use other accel
                case 'u':

                        buf[0] = '\0';
                DISPLAY_SCR(curno);                 /* put up active screen */
                        if(prompt("Which Accel for Disp/Spect/Plot (1/2)? ", ALPH
ANUMERIC, 2, buf)==0){
                                if (buf[0] == '1')useaccel_2 = FALSE;
                                if (buf[0] == '2')useaccel_2 = TRUE;
                        }
                        break;

                case 'd':
                case 'D':
                        while(!kbhit()){
                                _settextposition(10,10);
                                printf("C1: %ld           ", reada2d(0));
                                _settextposition(14,10);
                                printf("C2: %ld           ", reada2d(1));
                        }
                        break;


                case 'r':
                case 'R':                           // rms value
                    _clearscreen(_GCLEARSCREEN );
                        i = current_struct->no_of_points;
                        printf("\nRMS VALUES - Number of points %4d\n", i);

                        for(j=0; j<i; j++)
                                printf("[%4.2f,%4.2f]",x[j],y[j]);

                        for(j=0; j<i-4; j++){
                                        rmsvalue = sqrt(( (y[j]*y[j]) + (y[j+1]*y
[j+1]) +
                                        (y[j+2]*y[j+2]) + (y[j+3]*y[j+3]) )/4.0f)
;
                                        y[j] = rmsvalue;
                        }
                        stream  = fopen("rmsval.dat","w");
                        for(j=0; j<i-4; j++){
                                printf("[%4.2f,%4.2f]",x[j],y[j]);
                                fprintf(stream, "%4.2f , %4.2f\n", x[j], y[j]);
                        }
                        fclose(stream);
                        break;

                case 'p':
                case 'P':                           // peak value
                    _clearscreen(_GCLEARSCREEN );
                        i = current_struct->no_of_points;
                        printf("\nPEAK VALUES - Number of points %4d\n", i);
```

```
                        for(j=0; j<i; j++){                                     // zero o
ut peak array
                                y_spectrum[j]=0.0f;
                        }

                        for(j=0; j<i; j++){                                     // elimin
ate negative data
                                if(y[j] <= 0.0f) y[j] = 0.0f;
                        }
                        l = k = 0;
                        while(k<i){
                                maxy = 0.0f;
                                for(j=l; j<i; j++){
                                        if(y[j] >= maxy) maxy = y[j];   // find p
eak
                                        else break;
                                }
                                y_spectrum[j-1] = maxy;
                                for(k=j; k<i; k++){
        // find zero crossing
                                        if(y[k] == 0.0f)break;
                                }
                                l = k;
                        }


                        k = 0;
                        for(j=0; j<i; j++){
                                if(y_spectrum[j] > 0.0f){
                                        y[k] = y_spectrum[j];
                                        x[k] = j;
                                        printf("{%4.2f,%4.2f}\n",x[j],y[j]);
                                        k++;
                                }
                        }

                        stream  = fopen("peakval.dat","w");
                        for(j=0; j<i; j++){
                                if(y_spectrum[j]!=0.0f)
                                        fprintf(stream, "%4.2f , %6.2f\n",(float)
j, y_spectrum[j]);
                        }
                        fclose(stream);
                        printf("Number of peaks: %3d", k);
                        i = k;
                        getfunc();
                        break;

                case 'f':       // poly fit on the entire i points from current p
k vals

                    _clearscreen( _GCLEARSCREEN );
                        xlfit(i);                       // or try xfit(i)
                        getfunc();
                        break;

                case 'F':       // poly fit on the RMS values for the current dat
a

                    _clearscreen( _GCLEARSCREEN );
                        xlfit(i-4);             // or try xfit(i-4) for straight
line fit
                        getfunc();
                        break;
```



```
        case KY_F10:   // Exit to DOS
                case '0':

                        buf[0] = '\0';
                dir_display = FALSE;
                DISPLAY_SCR(curno);                     /* put up active screen */
                        if(prompt("EXIT - ARE YOU SURE (Y/y)? ", ALPHANUMERIC, 2,
 buf)==0){
                                if (buf[0] == 'Y' || buf[0] == 'y')done = TRUE;
                                else done = FALSE;
                        }
                        break;


        default :
                break;
        }   /* end switch */
    }       /* while */


    _displaycursor(_GCURSOROFF);
    _clearscreen( _GCLEARSCREEN );
    _outtext("LEAVING PROGRAM .......\n");

    /*      restore back to original        */

    _setvideomode(_DEFAULTMODE);

home:
    for(i=0; i<3; i++)free(scr[i]);
        for(i=0; i<3; i++)free(scr_file[i]);
    free(default_struct);
    free(current_struct);
}

void    display_help()
{
             _clearscreen( _GCLEARSCREEN );
         printf("IEM HUNTING SOFTWARE HELP\n\n\n");
         printf(">hunt [flags] \n\n");
         printf("-a/A : ampl/bin height required (1-9)*10 in RECORD-F5 save\n");
         printf("-b/B : show spectrum of the current DFT \n");
         printf("-c/C : accel to continue acq (1=.2g,9=1.8g)\n");
         printf("-d/D : turn on debugging of the system\n");
         printf("-f/F : frequency cutoff for RECORD-F5 save (1-9)*4\n");
         printf("-g/G : graph while acquiring data\n");
         printf("-k/K : keep data after DFT decision in RECORD-F5\n");
         printf("-m/M : minimum accel to start data acq (1=0.2g,9=1.8g)\n");
         printf("-p/P : turn on delta y for phase comparison\n");
         printf("-s/S : start speed to use (1=10, 9=90) - NA\n");
         printf("-t/T : turn on speed trigger to start speed sensor\n");
         printf("-u/U : use the second accelerometer in display/plot \n");
         printf("-w/W : wait factor for a2d to settle down (1-9)\n");
         printf("-x/X : scale factors for a2d. 1=0.00244,2=0.00488\n");
         printf("-y/Y : work with F2-F1 to autopilot record\n");
         printf("-z/Z : continue recording until (1-9)*20\n");
}

#if 0
// Grab the speed from a speed sensor
// Disable speed sensor if user choses to do so

#define SPEEDTO 10000000
#define SPEED   550
long    getspeed(void)
```

```
{
long    i,j;

if(speed_trigger == FALSE)return(999);          // no speed trigger is needed
else {

// get the actual speed from the sensor

for(i=0; i<SPEEDTO; i++){
        if( (inp(SPEED)&1) <1)break;            // wait for a low
}
if(i == SPEEDTO)return(10000001);       // timed out on low

for(i=0; i<SPEEDTO; i++){
        if( (inp(SPEED)&1) >0)break;            // wait for low to high
}
if(i == SPEEDTO)return(10000002);       // timed out on high

for(j=0; j<SPEEDTO; j++){
        if( (inp(SPEED)&1) <1)break;            // wait for a high to low
}
if(j == SPEEDTO)return(10000003);       // timed out on low

for(i=0; i<SPEEDTO; i++){
        if( (inp(SPEED)&1) >0)break;            // wait for low to high
}
if(i == SPEEDTO)return(10000004);       // timed out on high
else return(i+j);
}

}
#endif
long    getspeed(void)
{
        return(99999);
}
```

File: huntchng.c

Wheel Hunting Data Acquisition Software

```
////////////////////////////////////////////////////////////////
// File: huntchng.c
        //
//                                                            //
// Wheel Hunting Data Acquisition Software                    //
// Main program                                               //
//                                                            //
// Copyright 1995: IEM Corporation                            //
//                                                            //
// Proprietary Licensed Information.                          //
// Not to be duplicated, used or disclosed                    //
// except under terms of license.                             //
//                                                            //
// Revision History:                                          //
////////////////////////////////////////////////////////////////

#include "hunthead.h"   // Compiler includes
#include "huntext.h"    // External variable declarations
#include "huntstru.h"   // Constant and structure declarations
#include "huntllb.h"    // Function Prototypes
#include <ctools.h>

extern   float  profile_length;
extern   int    keyboard_flag;
extern   int    delta_y_flag;
extern   int    desired_freq;

void change_defaults(struct profile *default_struct)
{
    register boolean    done = FALSE;
    char buffer[100];
    int no_of_points, freq;
        float wheel_diameter;
    int key, i;

    while( !done ){

        set_screen(1);

        /* Now display current settings of default variables */
        _settextposition(5,recd_col);
        _outtext("                    ");
        _settextposition(5,recd_col);
        _outtext(default_struct->operator_id);

        _settextposition(7,recd_col);
        _outtext("                    ");
        _settextposition(7,recd_col);
        _outtext(default_struct->location_id);

        _settextposition(9,recd_col);
        _outtext("                    ");
        _settextposition(9,recd_col);
        _outtext(default_struct->loco_numb);

        _settextposition(11,recd_col);
        _outtext("                    ");
        _settextposition(11,recd_col);
        _outtext(default_struct->loco_type);

        _settextposition(13,recd_col);
        _outtext(default_struct->wheel_dia);

        _settextposition(15,recd_col);
        _outtext("                    ");
        _settextposition(15,recd_col);
```



```
        _outtext(default_struct->std_overlay);

        sprintf(buffer,"%4d",default_struct->no_of_points);
        _settextposition(17,recd_col);
        _outtext(buffer);

        sprintf(buffer,"%4d",(int)default_struct->frequency);
        _settextposition(19,recd_col);
        _outtext(buffer);


        key = getfunc();

        switch (key){

        case KY_F2:
                case '2':
            prompt("OPERATOR ID? ", ALPHANUMERIC, OP_ID_LEN,
            default_struct->operator_id);
            break;

        case KY_F3:
                case '3':
            prompt("LOCATION ID? ", ALPHANUMERIC, LOC_ID_LEN,
            default_struct->location_id);
            break;

        case KY_F4:
                case '4':
            prompt("LOCO NUMBER? ", ALPHANUMERIC,LOCO_NUM_LEN,
            default_struct->loco_numb);
            break;

        case KY_F5:
                case '5':
            prompt("LOCOMOTIVE TYPE? ", ALPHANUMERIC, LOCO_TYPE_LEN,
            default_struct->loco_type);
            break;

        case KY_F6:
                case '6':
            prompt("WHEEL DIAMTER?  ", FLOAT_NUMERIC, 6,buffer);
            wheel_diameter = (float)atof(buffer);
            if (wheel_diameter < 20.0f || wheel_diameter > 45.0f) {
                error_out(ONLY_MESS,
                "Wheel Diameter must be 20 to 45");
                wheel_diameter = 9999.0f;
                 _settextposition( MSG_ROW, MSG_COL);
                if(keyboard_flag == TRUE)
                    _outtext("                                                  ");
            }
            if(wheel_diameter != 9999.0f)
                    sprintf(default_struct->wheel_dia,"%6.4f",wheel_diameter);

            break;

        case KY_F7:
                case '7':
                        buffer[0]='\0';
                    prompt("STD OVERLAY FILE? ", ALPHANUMERIC, OVERLAY_FILE_LEN,
            buffer);
                        if(buffer[0]!='\0'){
                                strcpy(default_struct->std_overlay, buffer);
                        } else {
                    clear_text();
                        if(chose_file(buffer,"*.pr?") == 0){
```

```
                        for(i=0;i<8 && buffer[i]!='.';i++);
                        buffer[i]='\0';
                        strcpy(default_struct->std_overlay, buffer);
                    }
                }
            break;


        case KY_F8:
                case '8':

            prompt("NUMBER OF POINTS? ", ALPHANUMERIC, 5,buffer);
            no_of_points = (int)atol(buffer);
            if (no_of_points < 10 || no_of_points > 10000) {
                error_out(ONLY_MESS,
                "Number of points must be 10 to 10000");
                no_of_points = 32000;
                _settextposition( MSG_ROW, MSG_COL);
                if(keyboard_flag == TRUE)
                    _outtext("                                          ");
            }
            if(no_of_points != 32000)
                default_struct->no_of_points = no_of_points;
            break;

        case KY_F9:
                case '9':

            prompt("DESIRED DATA WAIT (1/f)? ", ALPHANUMERIC, 4,buffer);
            freq = (int)atol(buffer);
            if (freq< 0 || freq > 500) {
                error_out(ONLY_MESS,
                "Wait must be 1 to 500");
                freq = 9999;
                _settextposition( MSG_ROW, MSG_COL);
                if(keyboard_flag == TRUE)
                    _outtext("                                          ");
            }
            if(freq != 9999){
                default_struct->frequency = (float)freq;
                                desired_freq = freq;
                        }
            break;

        case KY_F10:   // Return to main menu
                case 'O':
                done = TRUE;
            break;

        default:
            break;
        }  // end of switch stmt
    } // end of while not done
}
```

File: huntenc.c

Wheel Hunting Data Acquisition Software

```
////////////////////////////////////////////////////////////
// File: huntenc.c                                        //
//                                                        //
// Wheel Hunting Data Acquisition Software                //
                          //
//                                                        //
// Copyright 1995: IEM Corporation                        //
//                                                        //
// Proprietary Licensed Information.                      //
// Not to be duplicated, used or disclosed                //
// except under terms of license.                         //
//                                                        //
// Revision History:                                      //
////////////////////////////////////////////////////////////

#define TEST 0

#include "xyenc.h"

#include <conio.h>
#include <stdio.h>
#include <math.h>
#include <conio.h>
#include <ctools.h>
#include <stdlib.h>
#include <float.h>
#include <string.h>

#include "huntstru.h"
#include "huntllb.h"

extern  float   profile_length;
extern  float   y_travel;

void XENC_init(void)
{
          outp(XENC_controlx1, 32);       // Master Control Register Reset
       outp(XENC_controlx1,121);   // Input Register Init
       outp(XENC_controlx1,128);   // Output Register Init
       outp(XENC_controlx1,  1);   // Address Pointer Resert for Preset & output


    // Load 0 in 3 8 bit chunks
       outp(XENC_datax1,  0);      // LSB
       outp(XENC_datax1,  0);      // MiddleSB
       outp(XENC_datax1,  0);      // MSB

          outp(XENC_controlx1,  8);       // Preset Reg Load
       outp(XENC_controlx1,193);   // Set Quadrature Mode

}

void YENC_init(void)
{
          outp(YENC_controly1, 32);       // Master Control Register Reset
       outp(YENC_controly1,121);       // Input Register Init
       outp(YENC_controly1,128);       // Output Register Init
       outp(YENC_controly1,  1);       // Address Pointer Resert for Preset & ou
tput


    // Load 0 in 3 8 bit chunks

       outp(YENC_datay1,  0);          // LSB - see xenc_load calculations
       outp(YENC_datay1,  0);          // MiddleSB - see below
       outp(YENC_datay1,  0);          // MSB
```



```
          outp(YENC_controly1,  8);       // Preset Reg Load
       outp(YENC_controly1,193);       // Set Quadrature Mode

}

void XENC_reset(void)
{
    outp(XENC_controlx1, 1);        // Address pointer reset for preset & output

    // Load 2000 Decimal in 3 8 bit chunks
    outp(XENC_datax1,  0);          // LSB
    outp(XENC_datax1,  0);          // MiddleSB
    outp(XENC_datax1,  0);          // MSB

    outp(XENC_controlx1, 8);        // Preset reg load

}

void YENC_reset(void)
{
    outp(YENC_controly1, 1);        // Address pointer reset for preset & output

    // Load 0 in 3 8 bit chunks

    outp(YENC_datay1,  0);          // LSB - see xenc_load calculations
    outp(YENC_datay1,  0);          // MiddleSB - see below
    outp(YENC_datay1,  0);          // MSB

    outp(YENC_controly1, 8);        // Preset reg load
}


#if PC_SIM == 1

extern long xpos_sim;
extern long ypos_sim;

long XENC_read(void)
{
    if( xpos_sim < 10000)
        return(++xpos_sim);
    else return(xpos_sim);
}

long YENC_read(void)
{
    if( ypos_sim < 4000)
        return(++ypos_sim);
    else return(ypos_sim);
}

#else
// Returns value in Hex
long XENC_read(void)
{
    unsigned long b1,b2;

    outp(XENC_controlx1,  1);
    outp(XENC_controlx1,  3);

    b1 = inp(XENC_datax1);
    b2 = inp(XENC_datax1);
//  b3 = inp(XENC_datax1);

//  return((long) ((b3<<16) + (b2<<8) + b1));
```

```
    return((long) ((b2<<8) + b1));

}

// Returns value in Hex
long YENC_read(void)
{
    unsigned long b1,b2;

    outp(YENC_controly1,  1);
    outp(YENC_controly1,  3);

    b1 = inp(YENC_datay1);
    b2 = inp(YENC_datay1);
//  b3 = inp(YENC_datay1);

//  return((long) ((b3<<16) + (b2<<8) + b1));
    return((long) ((b2<<8) + b1));
}
#endif

#if TEST
#include <ansi.h>
#include <ctools.h>

void main(void);

void main(void)
{
    int key;

    ERASE_ALL;


    for (;;) {
        // Display Options
            CUR_POS(3,5);
        printf("TEST PROGRAM FOR X-Y Encoders for the WHEEL PROFILOMETER");
        CUR_POS(5,1);
        printf(
        "F1-X init  F2-Y init  F3-X Reset  F4-Y Reset ");
        CUR_POS(6,1);
        printf("F10-Read            ESC-Exit");
        // Get Key
            key = getkey();

        switch(key) {

        case KY_F1:
            XENC_init();
            CUR_POS(9,1);
            printf("Init  X");
            sleep(1);
            CUR_POS(9,1);
            printf("       ");
            break;

        case KY_F2:
            YENC_init();
            CUR_POS(9,1);
            printf("Init  Y");
            sleep(1);
            CUR_POS(9,1);
            printf("       ");
            break;
```



```
        case KY_F3:
            XENC_reset();
            CUR_POS(9,1);
            printf("Reset X");
            sleep(1);
            CUR_POS(9,1);
            printf("       ");
            break;

        case KY_F4:
            YENC_reset();
            CUR_POS(9,1);
            printf("Reset Y");
            sleep(1);
            CUR_POS(9,1);
            printf("       ");
            break;


        case KY_F10:
            {
                float xdist,ydist;
                xdist = XENC_read()*.1;
                ydist = YENC_read()*.1;
                CUR_POS(10,10);
                printf("                                                  ");
                CUR_POS(10,10);
                printf("X = %10.4f    Y = %10.4f",xdist, ydist);
            }

            break;


        case KY_ESC:
            goto exit;
        }

    }
exit:
    ERASE_ALL;
}
#endif
```

File: huntmisc.c

Wheel Hunting Data Acquisition Software

```
////////////////////////////////////////////////////////////
// File: huntmisc.c                                       //
//                                                        //
// Wheel Hunting Data Acquisition Software                //
//                                                        //
// Copyright 1995: IEM Corporation                        //
//                                                        //
// Proprietary Licensed Information.                      //
// Not to be duplicated, used or disclosed                //
// except under terms of license.                         //
//                                                        //
// Revision History:                                      //
////////////////////////////////////////////////////////////

#include "hunthead.h"   // Compiler includes
#include "huntext.h"    // External variable declarations
#include "huntstru.h"   // Constant and structure declarations
#include "huntllb.h"    // Function Prototypes
#include <ctools.h>
#undef   BLACK                  /*  avoid redef warnings      */
#undef   RED
#undef   WHITE
#include <graphic.h>            /* Include all needed files */

extern  float huge x[MAX_PLOT_POINTS],y[MAX_PLOT_POINTS];
extern int keyboard_flag;
extern  char timebuf[];
extern int small_display_flag;
int             getfunc(void);
int     getalpha(void);
void    display_line(char *text_line, float x, float y);
void    prepare_grid(void);
extern  int perform_rdfft(int npts, float sampling_freq, char *temp);
extern  void    *getfield(FILE *stream, char *buffer);
extern  void    skiptoeol(FILE *stream);
extern  void    compute_line(int points);
extern  int             useaccel_2;

char    bell_sound[4]={'\7','\7','\7','\0'};

/*************************************************************/
/*                                                           */
/*  error_out() displays messages in the message box,        */
/*  using error_type to determine the appropriate message    */
/*  and error_msg as additional information.  The message    */
/*  is displayed for DELAY seconds, or until a key is pressed.  */
/*                                                           */
/*  There is no return value.                                */
/*                                                           */
/*************************************************************/


void error_out(const int error_type, char *error_msg)
{
    register short      old_color;

    if(error_type==TIME_OUT || error_type==GEN_ERR || error_type==BAD_PACK
        || error_type==RECORD_BAD )
        _setvideomode(_DEFAULTMODE);

    _settextwindow(MSG_ROW, MSG_COL, MSG_ROW+1, MSG_R_COL);
    _displaycursor(_GCURSOROFF);
    _clearscreen(_GWINDOW);

    _settextwindow(1, 1, MSG_ROW+1, MSG_R_COL+1);
```



```
    _settextposition( MSG_ROW, MSG_COL);

    /* save original foreground color */

    old_color = _settextcolor( error_clr );


    /* bell, buzzer, or other noise routine should be placed here */

        sound(2000,20);

    switch (error_type) {
    case FILE_READ :
        _outtext("ERROR : Cannot read file - ");
        _settextposition(MSG_ROW + 1, MSG_COL);
        _outtext( error_msg );
        break;

    case FILE_WRITE :
        _outtext("ERROR :Cannot write to file - ");
        _settextposition(MSG_ROW+1, MSG_COL );
        _outtext( error_msg );
        break;

    case TIME_OUT :
        _outtext("Communication Error-Check Hardware");
        _settextposition( MSG_ROW + 1, MSG_COL);
        _outtext(error_msg);
        break;

    case BAD_PACK :
        _outtext(
        "Bad Data Recieved in Communication - ");
        _settextposition( MSG_ROW + 1, MSG_COL);
        _outtext(error_msg);
        break;

    case BAD_FILE :
        _outtext("Attempt To Read Non-Data File");
        break;

    case GEN_ERR :
        _outtext("General System Error-Try Again");
        _settextposition(MSG_ROW+1, MSG_COL);
        _outtext( error_msg );
        break;

    case NO_RECS :
        _outtext("No Data to Display or Print");
        _settextposition(MSG_ROW+1, MSG_COL);
        _outtext( error_msg );
        break;

    case ONLY_MESS :
        _outtext("GENERAL ERROR ");
        _settextposition(MSG_ROW+1, MSG_COL);
        _outtext(error_msg);
        break;

    default :
        _settextposition(MSG_ROW+1, MSG_COL);
        _outtext( error_msg );
        break;
    }

    delay(8);
```

```
    /* restore original foreground color */

    _settextcolor( old_color );
}

/*************************************************************/
/*                                                           */
/*  question() prints the string 'string' in the message box */
/*  and waits for a response from the user.  The variable    */
/*  !getfile' is used when a filename is the desired response.*/
/*  question() will then allow only valid filenames in response */
/*                                                           */
/*  the return value is a pointer to the string containing the */
/*  user's response.                                         */
/*                                                           */
/*************************************************************/

int prompt(char *string, int data_type, int length, char * reply) {

    char *c;


    c = question(string, data_type, length);
    if(keyboard_flag == TRUE){
        settextposition(MSG_ROW, MSG_COL);
        if(small_display_flag == FALSE)
        _outtext("                                                  ");
                else
        _outtext("                                        ");

    }
    // if user enters a cr, then do nothing

    if (strlen(c) != 0){
        strcpy(reply, c);
                return(0);
        } else return(1);


//    else
//              *reply = '\0';

}



char *question(char *string, int data_type, int length)
{
    static char         answer[MAXCHAR];
    register _short_    ch, j;
    _short_             flen, ext, string_len;
    char                *c;
    int key;
    int cursorx,cursory;

    // Keyboard and touch screen have different places for writing message

    if (keyboard_flag == TRUE) {
        cursorx = MSG_ROW;
        cursory = MSG_COL;
    }

    /* set up screen */

    _settextwindow(cursorx, cursory, cursorx+1, R_COL);
```



```
    _displaycursor(_GCURSORON);
    _clearscreen(_GWINDOW);
    _settextwindow(1, 1, 25, R_COL+1);
    _settextposition(cursorx, cursory);
    _outtext(string);
    _displaycursor(_GCURSOROFF);

    j = ext = flen =0;

    if((c = MMALLOC(char, 2)) == NULL){
        printf("\nNot Enough Memory - Check System");
        getalpha();
        abort();
    }
    *(c + 1) = '\0';
    string_len = (_short_)strlen(string);


    /* get input one character at a time */

    for (;;){

        key = getalpha();          /* fetch a character */
        ch = (char) key;

        /* return pointer to answer */

        if (ch == KY_ENTER) {                          /* enter was hit */
            answer[j] = '\0';
            _displaycursor(_GCURSOROFF);
            free(c);
            return((char *)answer);
        }

        /* ESC key.  Return a NULL value */

        else if (ch == KY_ESC) {                       /* escape was hit */
            free(c);
            answer[0] = '\0';
            return((char *)answer);
        }

        /* backspace over previous character */

        else if (j > 0 && ch == KY_BACK) {
            j--;
            _settextposition(cursorx, cursory + j + string_len);
            _outtext(" ");
            _settextposition(cursorx, cursory + j + string_len);
            if (j == ext)
                ext = 0;

            else if ( j == flen)
                flen =0;
        }
        else if (data_type == FILENAME ){
            if (isalnum(ch) || ch == ':' || ch == '\\'|| ch == '.'){
                if ((ch == '.' || ch == '\\') && j >0 && answer[j-1] == ch){
                    beep();
                    continue;
                }


                /* do not allow more than three char after '.' */
                else if ( (ext > 0) && ( (j-ext) > 3) &&  (ch != '\\' ) ){
                    beep();
```

```
                continue;
            }
            else if ( (flen > 0) && ( (j-flen) > 8) &&  (ch != '\\' )
                &&  (ch != '.') ){
                beep();
                continue;
            }
            else{
                if (ch == '\\'){
                    flen =j;
                    ext = 0;
                }
                else if ( ch == '.'){
                    flen =0;
                    ext = j;
                }
                _settextposition(cursorx, cursory + j + string_len);
                *c = (char) ch;
                _outtext( c );
                answer[j++] = *c;
                if ( j > MAXCHAR)
                    break;
            }

        }
        else{
            beep();
            continue;
        }

    }


    /* if other data types are needed, be sure only valid characters are ente
red */

    else if( data_type == NUMERIC && ch >= '0' && ch <= '9'){
        _settextposition(cursorx, cursory + j + string_len);
        *c = (char) ch;
        if( (int)j < length){
            answer[j++] = *c;
            _outtext( c );
        }
    }
    else if( data_type == FLOAT_NUMERIC && ((ch >= '0' && ch <= '9') || ch ==
'.')){
        _settextposition(cursorx, cursory + j + string_len);
        *c = (char) ch;
        if( (int)j < length){
            answer[j++] = *c;
            _outtext( c );
        }
    }
    else if( data_type == ALPHANUMERIC && ((ch >= '0' && ch <= '9') || (ch >=
'A' && ch<= 'Z')
        || (ch >='a' && ch<= 'z') )) {
        _settextposition(cursorx, cursory + j + string_len);
        *c = (char) ch;
        if( (int)j < length){
            _outtext( c );
            answer[j++] = *c;
        }
    }
}   /* end for */
}
```



```
void delay(int seconds)
{
    register time_t     end_time;

    /* wait DELAY seconds, or until a key is pressed */

    end_time = time(NULL) + seconds;
    while ((end_time > time(NULL)) ) {
        if (keyboard_flag == TRUE) {
            if (kbhit() != 0) break;  // break out of the delay
        }

    }
}
/*********************************************************/
/*                                                       */
/*   Module Name:       DISK_OK - Gets floppy drive status */
/*                                                       */
/*      Author: ZFM               Last Date: 09/15/91    */
/*                                                       */
/*   Function:                                           */
/*       Uses DOS service function to get status         */
/*           returns TRUE, if drive is ready             */
/*                   FALSE if drive not ready            */
/*                                                       */
/*                                                       */
/*                                                       */
/*********************************************************/
int     disk_ok(char drive)
{
    boolean retcode;
    unsigned status = 0;
    struct diskinfo_t disk_info;
    if ( toupper(drive) == 'A')
        disk_info.drive = 0;
    else if ( toupper(drive) == 'B')
        disk_info.drive =1;
    else
        return(TRUE);


    disk_info.head =0;
    disk_info.track = 1;
    disk_info.sector =1;
    disk_info.nsectors = 1;

    status =  _bios_disk (_DISK_VERIFY, &disk_info);
    status = status >>8;
    if ( status == 0x80){
        retcode = FALSE;
    }
    else{
        retcode = TRUE;
    }
    return(retcode);

}

/*********************************************************/
/*                                                       */
/*   Module Name:       PRINTER_OK - Gets printer status  */
/*                                                       */
/*      Author: ZFM               Last Date: 09/15/91    */
/*                                                       */
/*   Function:                                           */
/*       Uses DOS service function to get printer status */
```

```
/*          returns 1   , if printer is available        */
/*                  0   , if printer not available       */
/*                                                       */
/*      Calls:  int86                                    */
/*                                                       */
/*                                                       */
/*******************************************************/

int printer_ok(void)
{
    unsigned status, data;
    int ret_code;
    union REGS inregs, outregs;
    int  i,j;

    // just in case where a printer is on power save
    // wait a little while longer

    /* Make sure printer has updated the status bits latch      */

for(j=0; j<10; j++){

    inregs.h.ah = 0x0;      /* character output selection */
    inregs.h.al = 0x0d;     /* output a <cr>              */
    inregs.x.dx = LPT1;     /* output to LPT1             */

    int86( 0x17, &inregs, &outregs );   /* output to printer */

    for( i=0; i<20000; i++)ret_code = 0;    /* wait a little */


    status = _bios_printer( _PRINTER_STATUS, LPT1, data);
    if( (status & 0x29) || !(status & 0x80) || !(status & 0x10) ){
        ret_code = 0;
    }
    else {
        ret_code = 1;
              break;
        }
 }


    return (ret_code);
}
/*******************************************************/
/*                                                     */
/*   Module Name:       GET_BAUD  - Gets baud rate from */
/*                      Blaise baud_rate table         */
/*                                                     */
/*      Author: ZFM                Last Date: 09/15/91 */
/*   Function:                                         */
/*       Use:specify -300 on command line for 300 baud */
/*           returns 0 to 7 corresponding to baud rate */
/*                   7   , if not one of the standard baud */
/*                                                     */
/*                                                     */
/*                                                     */
/*******************************************************/
int get_baud( char *crate)
{
    static int baud_rate[8] ={
        110,150,300,600,1200,2400,4800,9600    };
    int brate;
    int i;

    brate = atoi(crate);            /* convert string to integer */
```



```
    for ( i=0; i<7; i++){
        if (brate == baud_rate[i]){
            return(i);
        }
    }

    return (7);                                     /* default is 9600 baud */
}


/*******************************************************/
/*                                                     */
/*   Module Name:       VALID_FILE - Checks for valid  */
/*                      filename and creates it.       */
/*                      File is opened for overwrite!!! */
/*                                                     */
/*                                                     */
/*      Author: ZFM                Last Date: 09/15/91 */
/*                                                     */
/*   Returns:                                          */
/*      File stream, if filename is valid              */
/*      NULL       , if filename is invalid            */
/*                                                     */
/*******************************************************/
FILE *valid_file(char *fname)
{
    char buf[41];
    int index,next;
    FILE * stream;

    stream = fopen(fname,"w");
    if (stream == NULL){
        strcpy (buf,fname);
        if ( (index = findsl (buf,0) ) == MAXCHAR)
            return(stream);

        if( index  > 0){
            if (buf[index-1] != ':' && buf[index-1] != '.')
                index =-1;
        }
        next = ++index;

        while ( (index = findsl(buf,next) ) < MAXCHAR){
            buf[index] = '\0';
            if (  mkdir(buf) == ENOENT){
                return(stream);
            }
            buf[index] = '\\';
            next = ++index;

        }

        stream = fopen(fname,"w");
    }
    return(stream);

}
/*******************************************************/
/*                                                     */
/*   Module Name:       FINDSL - Finds '\' in a string */
/*                                                     */
/*      Author: ZFM                Last Date: 09/15/91 */
/*                                                     */
/*   Returns:                                          */
/*      index of element if '\' is found               */
```

```
/*      Other wise, returns MAXCHAR                             */
/*                                                              */
/****************************************************************/
int findsl (char *buf, int start)
{
    int index = MAXCHAR;
    int i;
    for (i=start; i<MAXCHAR+1; i++){
        if (buf[i] == '\\'){
            index = i;
            break;
        }
    }
    return(index);
}

void    beep(void)
{
sound(1000,40);
}

void    float_to_string(float float_number,char buffer[])
{

    char    tmp_buf[15];
    int     i,j;

    ltoa((long)(float_number*10000),buffer,10);
    j = strlen(buffer);
    if(j<4){
        strcpy(tmp_buf,"");
        for(i=j; i<4; i++)strcat(tmp_buf,"0");
        strcat(tmp_buf,buffer);
        strcpy(buffer, tmp_buf);
        j = strlen(buffer);
    }
    for(i=j; i>j-5; i--)
        buffer[i+1]=buffer[i];
    buffer[i+1] = '.';

    //strcpy(buffer,"5.345");

}

void set_screen(int curno)
{

    //  _setvideomode(_DEFAULTMODE);
    DISPLAY_SCR(curno);
    _displaycursor( _GCURSOROFF);       /* shut off cursor */
    _settextwindow(1, 1, 25, 80);
    _settextcolor( date_frgnd );
    _setbkcolor( date_bkgnd );
    _settextposition( 2, 72);
    _strdate(datebuf);
    if(small_display_flag == FALSE)
            _outtext( datebuf);                 /* display date & time */
    _strtime(timebuf);
    _settextcolor( frgnd );
    _setbkcolor( bkgnd );
}

/* This windows 18 by 58 */

void clear_text(void)
{
```



```
    _settextwindow( T_ROW, L_COL, B_ROW, R_COL);
    _clearscreen(_GWINDOW);
}


int system_call(char * command_line)
{
    int ret_val = 0;

    if (system(command_line) != 0) {
        switch (errno) {
        case E2BIG :
            error_out(GEN_ERR, "Enviroment size exceeds 32K");
            ret_val = -1;
            break;
        case ENOENT :
            error_out(GEN_ERR, "Cannot locate COMMAND.COM");
            ret_val = -1;
            break;
        case ENOEXEC :
            error_out(GEN_ERR, "Cannot execute COMMAND.COM");
            ret_val = -1;
            break;
        case ENOMEM :
            error_out(GEN_ERR, "Insufficient or corrupted memory");
            ret_val = -1;
            break;
        default :
            ret_val = 0;
            break;
        }
    }
    return(ret_val);
}



// return 0 if ok
// return < 0 if error

int prompt_for_file(int option, struct profile   *current_struct)
{
    char buf[100], overlay_name[30];
    char filename[50];
    struct profile *overlay_struct;
        float   yaxis_temp;
        FILE    *stream;
        char    dummy[20];
    int         npts, good_points_y = 0, good_points_z = 0, ii;
    char        buffer[20], buffer2[20];
        float   ystart=-5.0f, yend=5.0f;

    if((overlay_struct = (struct profile *)malloc(sizeof(struct profile)))==NULL)
 {
        printf("\nNot Enough Memory - Check System");
        getalpha();
        abort();
    }

    strcpy(overlay_name, current_struct->std_overlay);

//      printf("\nOverlay name on entry: %s", overlay_name);
//      getfunc();

        buf[0] = '\0';  // start with nothing in it. prompt will not touch if cr
pressed
```

```
    prompt("PROFILE FILE NAME? ", FILENAME, FILENAME_LENGTH,buf);

    if( buf[0] != '\0' ){

        if ((buf[0] == 'a' || buf[0] == 'b') && (buf[1]==':')){
            if (disk_ok(buf[0]) == FALSE){
                data_available = -1;
                error_out(ONLY_MESS, "Disk Drive not Ready");
                free(overlay_struct);
                return -1;
            }
        }

        clear_profile_info(current_struct);
        strcat(buf,".prf");
        _settextposition(TOP_ROW,TOP_COL);
        _outtext("Please wait for data retrieval...");
        _settextposition(TOP_ROW+1,TOP_COL);
        _outtext("                                    ");

        if(read_profile_struct(buf, current_struct)<0){
            data_available = -1;
            clear_profile_info(current_struct);
            error_out(FILE_READ, "Data File Not Found-Check Name");
            free(overlay_struct);
            return -1;
        }
        else {
            data_available = 1;
        }
    } else {
      clear_text();
         if(chose_file(buf,"*.pr?") == 0){

            clear_text();
        clear_profile_info(current_struct);
//      strcat(buf,".prf");
        _settextposition(TOP_ROW,TOP_COL);
        _outtext("Please wait for data retrieval...");
        _settextposition(TOP_ROW+1,TOP_COL);
        _outtext("                                    ");

        if(read_profile_struct(buf, current_struct)<0){
            data_available = -1;
            clear_profile_info(current_struct);
            error_out(FILE_READ, "Data File Not Found-Check Name");
            free(overlay_struct);
            return -1;
        }
        else  data_available = 1;

          } else data_available = 0;
        }

    if (data_available == 0) {
        error_out(NO_RECS, "No Graph To Display/Print");
        free(overlay_struct);
        return -1;
    }

    if(strcmpi(overlay_name,"NOFILE") != 0){
        strcpy(filename, overlay_name);
        strcat(filename, ".prf");

        clear_profile_info(overlay_struct);
```



```
        if(read_profile_struct(filename, overlay_struct)<0){
            error_out(FILE_READ, "Overlay File Not Found-Check Name");
            free(overlay_struct);
            return -1;
        }
        if(overlay_struct->no_of_points != current_struct->no_of_points){
            error_out(FILE_READ, "Unequal Data in Overlay File");
            free(overlay_struct);
            return(-1);
        }
    }


    //
    // Plot the mess on the screen/printer
    //

    strcpy(overlay_struct->std_overlay, overlay_name);


    // Piggy back data file name in current_struct

    strcpy(current_struct->std_overlay, buf);

    // data is passed in current_struct
        // overlay is passed in overlay_struct
        // overlay filename in overlay_struct also
        //

        if(option == DISPLAY_ONLY || option == DISPLAY_NPRINT){
            plot_data(option, current_struct, overlay_struct);
        }
    else if(option == 86){
                strcpy(buffer, current_struct->data_file);
                strcat(buffer, ".dat");
                if((stream = fopen(buffer,"r"))==NULL){
            error_out( FILE_READ, "Cannot read DATA File-Check File");
                    goto home;
                }
            npts = current_struct->no_of_points;
            for(ii=0; ii< npts; ii++) {
                        if(feof(stream))break;

                        getfield(stream, buffer);
                        getfield(stream, buffer2);
                        skiptoeol(stream);

//                      fscanf(stream,"%s %s %s %s %s\n",buffer,dummy,buffer2,dum
my,dummy);
                        if(useaccel_2 == FALSE)
                                yaxis_temp = (float)atof(buffer);
                        else
                                yaxis_temp = (float)atof(buffer2);
                x[ii] = yaxis_temp;
        }
            fclose(stream);
                strcpy(buffer, current_struct->data_file);
                strcat(buffer, ".dat");
                perform_rdfft(ii, current_struct->frequency, buffer);   // real d
iscrete fft
        }
    else if(option == 87){
                strcpy(buffer, current_struct->data_file);
                strcat(buffer, ".dat");
                if((stream = fopen(buffer,"r"))==NULL){
            error_out( FILE_READ, "Cannot read DATA File-Check File");
```

```
                    goto home;
                }
            npts = current_struct->no_of_points;
            for(ii=0; ii< npts; ii++) {
                    if(feof(stream))break;

                    getfield(stream, buffer);
                    getfield(stream, buffer2);
                    skiptoeol(stream);

//                  fscanf(stream,"%s %s %s %s %s\n",buffer,dummy,buffer2,dum
my,dummy);
                    if(useaccel_2 == FALSE)
                            yaxis_temp = (float)atof(buffer);
                    else
                            yaxis_temp = (float)atof(buffer2);
              y[ii] = yaxis_temp;
                  x[ii] = (float)ii;
        }
            fclose(stream);
                strcpy(buffer, current_struct->data_file);
                strcat(buffer, ".dat");
                compute_line(ii);
        }
home:
    free(overlay_struct);
    return(0);
}

// Interface subroutine for function keys on left of screen
int getfunc(void)
{
int key;

if (keyboard_flag == TRUE)key = getkey();
return(key);
}

// Interface subroutine for function keys on left of screen
int getalpha(void)
{
int key;

if (keyboard_flag == TRUE)key = getkey();
return(key);
}

/* Display the grid on the screen for display only */

void prepare_grid(void)
{
    graf("%3.1f",0.0f,1.0f,5.0f,"%3.1f",0.0f,0.50f,2.0f);
    sympick(12);
}

/*      Display text and associated value for screen */

void    display_line(char *text_line, float x, float y)
{
    prtfnt(x , y, text_line, 0.20f, 0);
}
```

File: huntio.c

Wheel Hunting Data Acquisition Software

```
////////////////////////////////////////////////////////////
// File: huntio.c                                         //
//                                                        //
// Wheel Hunting Data Acquisition Software                //
                        //
// Clears a profile structure, reads and writes           //
// a profile structure to disk                            //
//                                                        //
// Copyright 1995: IEM Corporation                        //
//                                                        //
// Proprietary Licensed Information.                      //
// Not to be duplicated, used or disclosed                //
// except under terms of license.                         //
//                                                        //
// Revision History:                                      //
////////////////////////////////////////////////////////////

#include "hunthead.h"   // Compiler includes
#include "huntext.h"    // External variable declarations
#include "huntstru.h"   // Constant and structure declarations
#include "huntllb.h"    // Function Prototypes

#define   DEBUG_CALCS   0

extern  char    *ftoaa(float innum, int precision);

extern  float   right_limit;

void copy_struct(struct profile *current_struct,struct profile *default_struct);

/*  Clear out all fields of a profile structure */

void    clear_profile_info(struct profile   *current_struct)
{
    strcpy(current_struct->operator_id,"NA");
    strcpy(current_struct->location_id,"NA");
    strcpy(current_struct->loco_type,"0");
    strcpy(current_struct->loco_numb,"NA");
    strcpy(current_struct->std_overlay,"NOFILE");
    strcpy(current_struct->wheel_dia,"NA");
    strcpy(current_struct->date_field,"NA");
    strcpy(current_struct->time_field,"NA");
    current_struct->no_of_points = 500;
    current_struct->frequency = 50.0f;

}

/*  Write out all fields of a profile structure */

int  write_profile_struct(char filename[], struct profile *current_struct)
{
    char    buffer[80];                 /* local scratch pad */
    int     i, dec, sign;
    FILE    *stream;
        char    *buf;

    if((stream = fopen(filename,"w"))!=NULL){
        fprintf(stream, "%-10s ...Operator Id\n",current_struct->operator_id);
        fprintf(stream, "%-10s ...Location Id\n",current_struct->location_id);
        fprintf(stream, "%-10s ...Loco Type\n",current_struct->loco_type);
        fprintf(stream, "%-10s ...Loco Number\n",current_struct->loco_numb);
        fprintf(stream, "%-10s ...Wheel Diameter\n",current_struct->wheel_dia);
        fprintf(stream, "%-10s ...Data File\n",current_struct->data_file);
        fprintf(stream, "%-10s ...Standard Overlay\n",current_struct->std_overlay
);
        fprintf(stream, "%-8s %-8s ...Date and Time\n",current_struct->date_field
,
```



```
current_struct->time_field);
        itoa(current_struct->no_of_points, buffer, 10);
        fprintf(stream, "%-10s ...Total Readings\n",buffer);
                buf = fcvt(current_struct->frequency,3, &dec, &sign);
                for(i=strlen(buf); i>=dec;i--)buf[i+1]=buf[i];
        buf[dec]='.';
        fprintf(stream, "%-10s ...Data Frequency\n",buf);

        i = 1;                  /* things went ok */
    }
    else {
        i = -1;                 /* return -1 for error */
    }

    fclose(stream);
    return(i);                  /* 1 for successful write */
}

/*  Read in all fields of a profile structure */

int  read_profile_struct(char filename[], struct profile *current_struct)
{
    char    buffer[80];                 /* local scratch pad */
    int     i;
    FILE    *stream;

        if((stream = fopen(filename,"r")) != NULL ){
        fscanf(stream, "%s",buffer);
        skip(stream);
        if(buffer[0] != '.')strcpy(current_struct->operator_id, buffer);
        else strcpy(current_struct->operator_id, "NA");

        fscanf(stream, "%s",buffer);
        skip(stream);
        if(buffer[0] != '.')strcpy(current_struct->location_id, buffer);
        else strcpy(current_struct->location_id, "NA");

        fscanf(stream, "%s",buffer);
        skip(stream);
        if(buffer[0] != '.')strcpy(current_struct->loco_type, buffer);
        else strcpy(current_struct->loco_type, "0");

        fscanf(stream, "%s",buffer);
        skip(stream);
        if(buffer[0] != '.')strcpy(current_struct->loco_numb, buffer);
        else strcpy(current_struct->loco_numb, "NA");

        fscanf(stream, "%s",buffer);
        skip(stream);
        if(buffer[0] != '.')strcpy(current_struct->wheel_dia, buffer);
        else strcpy(current_struct->wheel_dia, "NA");

        fscanf(stream, "%s",buffer);
        skip(stream);
        if(buffer[0] != '.')strcpy(current_struct->data_file, buffer);
        else strcpy(current_struct->data_file, "NA");

        fscanf(stream, "%s",buffer);
        skip(stream);
        if(buffer[0] != '.')strcpy(current_struct->std_overlay, buffer);
        else strcpy(current_struct->std_overlay, "NA");

        fscanf(stream, "%s %s",current_struct->date_field,current_struct->time_fi
eld);
        skip(stream);
```

```
        fscanf(stream, "%s",buffer);
        skip(stream);
        current_struct->no_of_points = atoi(buffer);

        fscanf(stream, "%s",buffer);
        skip(stream);
        current_struct->frequency = (float)atof(buffer);

                i = 1;

    }
    else {
        i = -1;                 /* return -1 for error */
    }

    fclose(stream);
    return(i);                  /* 1 for successful write */
}

/*  Skip trailing comments on a line */

void    skip(FILE *stream)
{
    for(;;){
        if(feof(stream) == 0 && fgetc(stream) != '\n')continue;
        else return;
    }
}

/*  Copy structure from default structure */

void    copy_struct(struct profile *current_struct,struct profile *default_struct
)
{
    strcpy(current_struct->operator_id, default_struct->operator_id);
    strcpy(current_struct->location_id, default_struct->location_id);
    strcpy(current_struct->loco_numb, default_struct->loco_numb);
    strcpy(current_struct->loco_type, default_struct->loco_type);
    strcpy(current_struct->wheel_dia, default_struct->wheel_dia);
    strcpy(current_struct->std_overlay, default_struct->std_overlay);
    current_struct->no_of_points =  default_struct->no_of_points;
    current_struct->frequency =  default_struct->frequency;
}

/*  Read in some of the fields of a profile structure */

int  read_profile_struct_partial(char filename[], struct profile *current_struct)
{
    char    buffer[80];             /* local scratch pad */
    int     i;
    FILE    *stream;

    if((stream = fopen(filename,"r"))!=NULL){

        fscanf(stream, "%s ",buffer);
        skip(stream);
        if(buffer[0] != '.')strcpy(current_struct->operator_id, buffer);
        else strcpy(current_struct->operator_id, "");

        fscanf(stream, "%s ",buffer);
        skip(stream);
        if(buffer[0] != '.')strcpy(current_struct->location_id, buffer);
        else strcpy(current_struct->location_id, "");

        fscanf(stream, "%s ",buffer);
        skip(stream);
```



```
        if(buffer[0] != '.')strcpy(current_struct->loco_type, buffer);
        else strcpy(current_struct->loco_type, "0");

        fscanf(stream, "%s ",buffer);
        skip(stream);
        if(buffer[0] != '.')strcpy(current_struct->loco_numb, buffer);
        else strcpy(current_struct->loco_numb, "");

                for(i=0; i<11; i++){                                    // ??????
??????????
                fscanf(stream, "%s ",buffer);
            skip(stream);
                }

        fscanf(stream, "%s ",buffer);
        skip(stream);
        if(buffer[0] != '.'){
                current_struct->no_of_points = atoi(buffer);
        }
        else current_struct->no_of_points = 500;

        fscanf(stream, "%s ",buffer);
        skip(stream);
        if(buffer[0] != '.'){
                current_struct->frequency = (float)atof(buffer);
        }
        else current_struct->frequency = 50.0f;


        // can possibly import more data for usage

        i = 1;
    }
    else {
        i = -1;                 /* return -1 for error */
    }
    fclose(stream);
    return(i);                  /* 1 for successful write */
}
```

File: huntrecd.c

Wheel Hunting Data Acquisition Software

```
////////////////////////////////////////////////////////////
// File: huntrecd.c                                       //
//                                                        //
// Wheel Hunting Data Acquisition Software                //
                        //
// Record Profile                                         //
//                                                        //
// Copyright 1995: IEM Corporation                        //
//                                                        //
// Proprietary Licensed Information.                      //
// Not to be duplicated, used or disclosed                //
// except under terms of license.                         //
//                                                        //
// Revision History:                                      //
////////////////////////////////////////////////////////////
//                                                        //
// Took out the speed traps entirely for now.             //
// Look at huntrecd.bak for old speed trap work.          //
//                                                ZFM 2/22/96
      //
//                                                        //
//********************************************************

#include "hunthead.h"   // Compiler includes
#include "huntext.h"    // External variable declarations
#include "huntstru.h"   // Constant and structure declarations
#include "huntllb.h"    // Function Prototypes

#include "xyenc.h"
#include <ctools.h>
#include <types.h>
#include <timeb.h>

#define  NO_GRAPHICS    0       // do not put graphics if 1

#undef   BLACK
#undef   RED
#undef   WHITE

#include        <graphic.h>                     /* Include all needed files *
/

#define         MAX_READLOOP_TIMEOUT    4000    /* timeout if just sitting there
*/
#define         CONTACT_CONTROL                 890             // sensor control
 port 37Ah
#define         CONTACT_READ                    889             // read port 379h
#define         CONTACT_ON                              72                      /
/ contacts are on when 72

int             diskfree(int dr);
int     getaccel(float *accel_output_a, float *accel_output_b);
void    wait_freq_time(int);
long    reada2d(int channel);
extern  void init_plot(void);
extern  void draw(int pnum,float yf);
extern  int perform_rdfft(int, float, char *);          // real discrete fft
extern  long getspeed(float);

extern  float profile_length;
extern  float y_travel;
extern  int keyboard_flag;
extern  int omithard_flag;
extern  char timebuf[];
extern  long XENC_read(void);
extern  long YENC_read(void);
```



```
extern  void prepare_grid(void);
extern  int  small_display_flag;
extern  float delta_y;                          // we will only allow delta_y for
 reading
extern  drive;
extern  float   accel_min_level;
extern  float   accel_sustain_level;
extern  float   scale_factor;
extern  int             startspeed;
extern  int             speed_trigger;
extern  int             wait_time;
extern  int             plotdata;
extern  int             keep_data;
extern  int             desired_freq;
extern  int             autopilot;
extern  int             second_accel_desired;

//      Local to this file

struct  find_t c_file;
int     key, column;
char    buff[30], filebuf[20], filetmpbuf[20], extbuf[10], ch;
char    target_filename[20], overlay_filename[20], temp_name[20];
int     number_points, timeout;
int     i, overlay_partial_available;
int     done = FALSE, ov_contacts = FALSE;
int     indx, old_indx = 0, plot_indx;
struct  profile *ovl_struct;
float   xfloat, yfloat, xtravel_inc, ydelta, xcoord;
float   xfloat_old, yfloat_old, yfloat_last, xfloat_last=0.0f;
int     dl, dh;                 // used by a2d

extern  float   huge x[MAX_I_POINTS], y[MAX_I_POINTS];
extern  float   huge x_spectrum[MAX_I_POINTSHALF], y_spectrum[MAX_I_POINTSHALF];
extern  int     useaccel_2;
extern  int     amplitude_required;
extern  int             frequency_cutoff;
extern  int             continue_count;
extern  int             show_output;

//

void    record_profile(
struct  profile   *current_struct
)
{
    FILE    *stream;
        int             j=0, pindex, oldpindex = 0;
        char    *buffer;                        // scratch pad
        float   accel_output_a, accel_output_b;
        int             count, actual_bins;
        float   sampling_freq, timediff, current_dia;
        struct  timeb st;
        struct  timeb et;
        long    speed, curspeed, timestamp;
        char    *timeline;

        done = FALSE;
    overlay_partial_available = FALSE;
        column = 1;

    if(( buffer = (char *)malloc(20))==NULL ){
        printf("\nNot Enough Memory - Check System");
        getalpha();
        abort();
    }
```

```
        strcpy(target_filename, "DEFAULT.PRF");


    while( !done ){
            indx = timeout = old_indx = 0;
                xfloat_last = 0.0f;

        set_screen(2);
        _strtime(timebuf);
        strcpy(current_struct->time_field, timebuf);

        /* Now display current settings of default variables    */

        _settextposition(5,recd_col);
        _outtext("                ");
        _settextposition(5,recd_col);
        _outtext(current_struct->loco_numb);

        _settextposition(7,recd_col);
        _outtext("                ");
        _settextposition(7,recd_col);
        _outtext(current_struct->wheel_dia);

                _strdate(temp_name);
                _strtime(buffer);
                target_filename[0]=temp_name[0];
                target_filename[1]=temp_name[1];
                target_filename[2]=temp_name[3];
                target_filename[3]=temp_name[4];
                target_filename[4]=buffer[0];
                target_filename[5]=buffer[1];
                target_filename[6]=buffer[3];
                target_filename[7]=buffer[4];
                target_filename[8]='\0';
                strcpy(temp_name, target_filename);
                strcat(target_filename,".prf");
                strcpy(current_struct->data_file, temp_name);
                strcat(temp_name,".dat");

        _settextposition(9 ,recd_col);
        _outtext("                ");
        _settextposition(9 ,recd_col);
        _outtext(target_filename);

        strcpy(overlay_filename,current_struct->std_overlay);
        _settextposition(11,recd_col);
        _outtext("                ");
        _settextposition(11,recd_col);
        _outtext(overlay_filename);

                current_dia = (float)atof(current_struct->wheel_dia);

                if(autopilot == TRUE)goto jump;

        key = getfunc();

        switch (key) {

        case KY_F1:     // Record
                case '1':

                jump:
                        do {

                        if( diskfree(drive) >0 ){
```



```
                                error_out(FILE_WRITE, "Not Enough Disk Space");
                                break;
                        }

                number_points = current_struct->no_of_points;   // # pts to scan in X
 AXIS

                // make filename for the output or target
                        // have the filename ready always

                        _strdate(temp_name);
                        _strtime(buffer);
                        target_filename[0]=temp_name[0];
                        target_filename[1]=temp_name[1];
                        target_filename[2]=temp_name[3];
                        target_filename[3]=temp_name[4];
                        target_filename[4]=buffer[0];
                        target_filename[5]=buffer[1];
                        target_filename[6]=buffer[3];
                        target_filename[7]=buffer[4];
                        target_filename[8]='\0';
                        strcpy(temp_name, target_filename);
                        strcat(target_filename,".prf");
                        strcpy(current_struct->data_file, temp_name);
                        strcat(temp_name,".dat");

                    if((stream = fopen(temp_name,"w"))==NULL){
                                printf("Cannot open output DATA file %s", temp_na
me);
                        }

                        // Now grab the rest of the data set
                        // wait until start speed is reached

                        _clearscreen(_GCLEARSCREEN);

                        if(plotdata == TRUE){
                                if(_setvideomode(6)==0){
                                        printf("Error:videomode\n");
                                        getfunc();
                                }
                                init_plot();
                        }

                        count = 0;
                        oldpindex = 0;

                        // Continue until speed > start speed

                        while( 1 ){

                                getaccel(&accel_output_a, &accel_output_b);

                                // If acceleration < minimum level

                            _settextposition(18,10);
                        _outtext("Waiting for acceleration to excede         ");

                                if(accel_output_a >= accel_min_level || accel_out
put_b >= accel_min_level || kbhit()){

                                        // Take at least MAXREADINGS readings of
data or until accel is high

                                        ftime(&st);
                                        while( count <= number_points || accel_ou
t
```

```
put_a > accel_sustain_level ||
                                        accel_output_b > accel_sustain_level )
{
                                            if(show_output== TRUE){
                                            _settextposition(19,10);

                                         printf("%5d", count);
                                            }
                                            getaccel(&accel_output_a, &accel_
output_b);
                                            count++;
                                            if(plotdata == TRUE){
                                                    pindex = count/160;
                                                    if(pindex > oldpindex){
                                                            oldpindex=pindex;
                                                            init_plot();
                                                    }
                                                    if(useaccel_2 == FALSE)
                                                            draw(count-oldpin
dex*160, accel_output_a);
                                                    else
                                                            draw(count-oldpin
dex*160, accel_output_b);
                                            }
                                            ftime(&et);
                                            timeline = ctime(&(et.time));
                                            fprintf(stream, "%4.2f,%4.2f,%.2s
%.2s%.2s.%hu\n", accel_output_a, accel_output_b, &timeline[11], &timeline[14], &t
imeline[17],et.millitm);
//
//                                          y[count]=accel_output_b;
                                            x[count]=accel_output_a;
                                            wait_freq_time(desired_freq);
                                            if(count >= number_points)goto sk
ip;             // stop after a while
                                            if(kbhit())goto stopdata;
                                        }
                                }
                        }
                        skip:
                        ftime(&et);
                        timediff = ( (float)(et.time*1000 + et.millitm) -(float)(
st.time*1000 + st.millitm) ) /1000.0f;
                        if(timediff > 0.0f){
                                sampling_freq = (float)count/(timediff);
                                printf(" Rate (f): %6.3f", sampling_freq);
                } else {
                                printf(" Slow down sample rate/user aborted");
                                getfunc();
                                break;
                        }
                current_struct->no_of_points = count;
                        current_struct->frequency = sampling_freq;

//                      actual_bins = perform_rdfft(count, sampling_freq);

                        printf(" Set#: %3d",continue_count);
                        if(continue_count<0)
                                getfunc();
                        continue_count--;

                // write profile header file to the disk
                write_profile_struct(target_filename,current_struct);
                        fclose(stream);

                        }while(continue_count > 0);
```



```
                        stopdata:
                        if(autopilot == TRUE)autopilot = FALSE;
                _setvideomode(_TEXTC80);
                        break;

        case KY_F2:     // Wheel Diameter
                case '2':

                prompt("WHEEL DIAMETER? ", ALPHANUMERIC, WHL_DIA_LEN,
                current_struct->wheel_dia);
                break;

        case KY_F3:     // Target File name
                case '3':

                prompt("OUTPUT FILENAME? ", ALPHANUMERIC, FILENAME_LENGTH,
                target_filename);
                        for(i=0; i<strlen(target_filename) && target_filename[i]!
='.';i++);
                        if(target_filename[i] != '.')strcat(target_filename,".PRF
");
                break;

        case KY_F4:     // Overlay file name
                case '4':

                if(prompt("OVERLAY NAME? ", ALPHANUMERIC, FILENAME_LENGTH,
                overlay_filename) == 0)
                        strcpy(current_struct->std_overlay,overlay_filename);
                        else{
                                clear_text();
                                if(chose_file(overlay_filename,"*.pr?") == 0)
                                        strcpy(current_struct->std_overla
y,overlay_filename);
                        }
                        break;

        case KY_F5:     // Perform FFT
                case '5':

                        if( diskfree(drive) >0 ){
                                error_out(FILE_WRITE, "Not Enough Disk Space");
                                break;
                        }

                number_points = current_struct->no_of_points;   // # pts to scan in X
 AXIS

                // make filename for the output or target - always there

                        // Now grab the rest of the data set
                        // wait until start speed is reached

                        _clearscreen(_GCLEARSCREEN);

                        if(plotdata == TRUE){
                                if(_setvideomode(6)==0){
                                        printf("Error:videomode\n");
                                        getfunc();
                                }
                                init_plot();
                        }
                        count = 0;
                        oldpindex = 0;

                        // Continue until speed > start speed
```

```
put_a > accel_sustain_level ||
                                        accel_output_b > accel_sustain_level )
{
                                            if(show_output== TRUE){
                                            _settextposition(19,10);

                                         printf("%5d", count);
                                            }
                                            getaccel(&accel_output_a, &accel_
output_b);
                                            count++;
                                            if(plotdata == TRUE){
                                                    pindex = count/160;
                                                    if(pindex > oldpindex){
                                                            oldpindex=pindex;
                                                            init_plot();
                                                    }
                                                    if(useaccel_2 == FALSE)
                                                            draw(count-oldpin
dex*160, accel_output_a);
                                                    else
                                                            draw(count-oldpin
dex*160, accel_output_b);
                                            }
                                            ftime(&et);
                                            timeline = ctime(&(et.time));
                                            fprintf(stream, "%4.2f,%4.2f,%.2s
%.2s%.2s.%hu\n", accel_output_a, accel_output_b, &timeline[11], &timeline[14], &t
imeline[17],et.millitm);
//
//                                          y[count]=accel_output_b;
                                            x[count]=accel_output_a;
                                            wait_freq_time(desired_freq);
                                            if(count >= number_points)goto sk
ip;             // stop after a while
                                            if(kbhit())goto stopdata;
                                        }
                                    }
                            }
                            skip:
                            ftime(&et);
                            timediff = ( (float)(et.time*1000 + et.millitm) -(float)(
st.time*1000 + st.millitm) ) /1000.0f;
                            if(timediff > 0.0f){
                                    sampling_freq = (float)count/(timediff);
                                    printf(" Rate (f): %6.3f", sampling_freq);
                    } else {
                                    printf(" Slow down sample rate/user aborted");
                                    getfunc();
                                    break;
                            }
                    current_struct->no_of_points = count;
                            current_struct->frequency = sampling_freq;

//                          actual_bins = perform_rdfft(count, sampling_freq);

                            printf(" Set#: %3d",continue_count);
                            if(continue_count<0)
                                    getfunc();
                            continue_count--;

                    // write profile header file to the disk
                    write_profile_struct(target_filename,current_struct);
                            fclose(stream);

                            }while(continue_count > 0);
```



```
                            stopdata:
                            if(autopilot == TRUE)autopilot = FALSE;
                    _setvideomode(_TEXTC80);
                            break;

                case KY_F2:     // Wheel Diameter
                        case '2':

                    prompt("WHEEL DIAMETER? ", ALPHANUMERIC, WHL_DIA_LEN,
                    current_struct->wheel_dia);
                    break;

                case KY_F3:     // Target File name
                        case '3':

                    prompt("OUTPUT FILENAME? ", ALPHANUMERIC, FILENAME_LENGTH,
                    target_filename);
                            for(i=0; i<strlen(target_filename) && target_filename[i]!
='.';i++);
                            if(target_filename[i] != '.')strcat(target_filename,".PRF
");
                    break;

                case KY_F4:     // Overlay file name
                        case '4':

                    if(prompt("OVERLAY NAME? ", ALPHANUMERIC, FILENAME_LENGTH,
                    overlay_filename) == 0)
                            strcpy(current_struct->std_overlay,overlay_filename);
                            else{
                                    clear_text();
                                    if(chose_file(overlay_filename,"*.pr?") == 0)
                                            strcpy(current_struct->std_overla
y,overlay_filename);
                            }
                            break;

                case KY_F5:     // Perform FFT
                        case '5':

                            if( diskfree(drive) >0 ){
                                    error_out(FILE_WRITE, "Not Enough Disk Space");
                                    break;
                            }

                    number_points = current_struct->no_of_points;   // # pts to scan in X
 AXIS

                    // make filename for the output or target - always there

                            // Now grab the rest of the data set
                            // wait until start speed is reached

                            _clearscreen(_GCLEARSCREEN);

                            if(plotdata == TRUE){
                                    if(_setvideomode(6)==0){
                                            printf("Error:videomode\n");
                                            getfunc();
                                    }
                                    init_plot();
                            }
                            count = 0;
                            oldpindex = 0;

                            // Continue until speed > start speed
```

```
                        while( 1 ){

                                getaccel(&accel_output_a, &accel_output_b);

                                // If acceleration < minimum level

                             _settextposition(18,10);
                          _outtext("Waiting for acceleration to excede        ");

                                if(accel_output_a > accel_min_level || accel_outp
ut_b > accel_min_level){

                                        // Take at least MAXREADINGS readings of
data or until accel is high

                                        ftime(&st);
                                        while( count <= number_points || accel_ou
tput_a > accel_sustain_level ||
                                           accel_output_b > accel_sustain_level )
{
                                         _settextposition(19,10);
                                    printf("P#: %5d", count);
                                                getaccel(&accel_output_a, &accel_
output_b);

                                                count++;
                                                speed = getspeed(current_dia);
                                                if(plotdata == TRUE){
                                                        pindex = count/160;
                                                        if(pindex > oldpindex){
                                                                oldpindex=pindex;
                                                                init_plot();
                                                        }
                                                        if(useaccel_2 == FALSE)
                                                                draw(count-oldpin
dex*160, accel_output_a);

                                                        else
                                                                draw(count-oldpin
dex*160, accel_output_b);

                                                }
                                                x[count] = accel_output_a;
                                                y[count] = accel_output_b;
                                                wait_freq_time(desired_freq);
                                                if(count >= number_points || kbhi
t())goto skip2;         // stop after a while
                                                }
                                        }
                                }
                                skip2:
                                ftime(&et);
                                timediff = ( (float)(et.time*1000 + et.millitm) -(float)(
st.time*1000 + st.millitm) ) /1000.0f;
                                if(timediff > 0.0f){
                                        sampling_freq = (float)count/(timediff);
                                        printf(" Rate (f): %6.3f", sampling_freq);
                        } else {
                                        printf(" Slow down sample rate/user aborted");
                                        getfunc();
                                        goto skip3;
                                }

                  current_struct->no_of_points = count;
                                current_struct->frequency = sampling_freq;

                                // perform real discrete fft
```



```
                        _settextposition(20,10);
                                actual_bins = perform_rdfft(count, sampling_freq, "CURREN
T");

                                // may want to perform some qualification test on DFT
                                // x_spectrum and y_spectrum contain information

                                if(keep_data == TRUE){

                                        for(i=0; i<actual_bins; i++){
                                                if((int)x_spectrum[i] > frequency_cutoff)
goto skip3;
                                                if((int)y_spectrum[i] > amplitude_require
d)break;
                                        }

                                // write profile header file to the disk

                                _settextposition(20,10);
                                        printf("Now Saving the data");
                                write_profile_struct(target_filename,current_struct);

                                if((stream = fopen(temp_name,"w"))==NULL){
                                        printf("Cannot open output DATA file %s",
 temp_name);
                                        getfunc();
                                } else {
                                        for(i=0; i<count; i++)
                                                fprintf(stream, "%4.2f,%4.2f,0\n"
, x[i], y[i]);
                                        fclose(stream);
                                }
                                }

                                skip3:
                        _setvideomode(_TEXTC80);
                                break;

                        case KY_F9:
                        case '9':
                                while(!kbhit()){
                                        _clearscreen( GCLEARSCREEN);
                                _settextposition(20,10);
                                        printf("Speed is %3d",getspeed(current_dia));
                                }
                                break;

                case KY_F10:    // Return to main screen
                        case '0':

                        done = TRUE;
                        break;

                default:
                        break;
                }
        }
        free(buffer);
}

int             diskfree(int dr)
{
struct  diskfree_t drive;
float   totbytes;

        _dos_getdiskfree(dr, &drive);           // test for space on drive
```

```
        totbytes = (float)(drive.avail_clusters) * (float)(drive.sectors_per_clus
ter) * (float)(drive.bytes_per_sector);
        if( totbytes < 50000.0f)return(1);
        else return(0);
}

// Grab the accelerometer inputs

int getaccel(float *accel_output_a, float *accel_output_b)
{

// read the two a2d channels and scale for G levels

*accel_output_a = reada2d(0)*scale_factor;      // (.0012/bit) / (0.1volt/g)

if(second_accel_desired == TRUE)
        *accel_output_b = reada2d(1)*scale_factor;      // (.0012/bit) / (0.1volt
/g)
else
        *accel_output_b = 0.0f;                 // dummy in a zero value

return(0);
}

void wait_freq_time(int input_freq)
{
int     j, k;

for(j=0; j<input_freq; j++)k;

}

// take multiple readings and average out the answer
// if do not need multiple reads, then optimize for speed

long reada2d(int channel)
{

      outp(0x220+10,channel);                           /* A/D CHANNEL ALWAYS 0
220h+10 */

          for(dl=0; dl<wait_time; dl++)dh++;    // kill some time

      outp(0x220+12,0);         /* SOFTWARE TRIG - not outportB */
      while (inp(0x225>15));
      return((inp(0x225)*256+inp(0x224)-2048));

}

#if 0
//      This routine uses the MUX board infront of the a2d board.
//  The MUX board is removed as it is holding the data values between channels.
//                                                      ZFM 4/6/96

// take multiple readings and average out the answer
// if do not need multiple reads, then optimize for speed

long reada2d(int channel)
{

//      outp(0x220+10,0);                               /* A/D CHANNEL ALWAYS 0
220h+10 */
              outp(0x220+13,channel);                   /* SELCT AMPLIFIER CHANNE
L 0 or 1*/

          for(dl=0; dl<wait_time; dl++)dh++;    // kill some time
```



```
      outp(0x220+12,0);         /* SOFTWARE TRIG - not outportB */
      do {
          dh=inp(0x225);                /* READ HIGH BYTE DATA*/
      } while (dh>15);
      dl=inp(0x224);                    /* READ LOW BYTE DATA */

//            outp(0x220+13,2);                 /* SELCT AMPLIFIER CHANNEL for gr
ound */
      return((dh*256+dl-2048));
}
#endif
```

File: huntfile.c

Wheel Hunting Data Acquisition Software

```
////////////////////////////////////////////////////////////
// File: huntfile.c                                       //
//                                                        //
// Wheel Hunting Data Acquisition Software                //
                        //
// File Selection Program                                 //
//                                                        //
// Copyright 1995: IEM Corporation                        //
//                                                        //
// Proprietary Licensed Information.                      //
// Not to be duplicated, used or disclosed                //
// except under terms of license.                         //
//                                                        //
// Revision History:                                      //
////////////////////////////////////////////////////////////

#include "hunthead.h"
#include "huntllb.h"
#include <graph.h>          /* graphic declaration file */
#include <ctools.h>

#define st_c0 0         // item pointer 0 to 44
#define st_c1 15
#define st_c2 30

//      These numbers have to be wrt to the window selected

#define ptr_st_c_LARGE 1        // st ptr in col 15 and add width
#define st_c_LARGE              5       // avoid side menus
#define tcw_LARGE  18           // 13 + 5 for ptr

#define ptr_st_c_SMAL 1         // st ptr in col 15 and add width
#define st_c_SMAL          2            // avoid side menus
#define tcw_SMAL    13          // 12 + 1 for ptr

#define st_ROW                              1           // used for text display
#define MAX_ENTRIES            45               // three 15 item columns
extern int         small_display_flag;

void highlite(char *buffer,int row,int k);
void unhighlite(char *buffer,int row,int k);
int chose_file(char *buffer, char *filespec);

int        ptr_st_c;
int        st_c;
int    tcw;

#if 0
void main(void);
void main()
{
char    filename[20];

chose_file(filename);
printf("Selection %s\n", filename);

}
#endif

int chose_file(char *filename, char *filespec)
{
struct find_t c_file;
char   *buffer[500], prompt_ch[5], prompt_blk[5];
int        i = 0, j, key, k, pages, pagenum, jj, row, last_entry;

if(small_display_flag == 1){
```



```
        ptr_st_c = ptr_st_c_SMAL;
        st_c = st_c_SMAL;
        tcw = tcw_SMAL;
        strcpy(prompt_ch, "*");
        strcpy(prompt_blk, " ");
} else {
        ptr_st_c = ptr_st_c_LARGE;
        st_c = st_c_LARGE;
        tcw = tcw_LARGE;
        strcpy(prompt_blk,"   ");
        strcpy(prompt_ch, "==>");
}

if((buffer[i] = (char *)malloc(13))==0){
            printf("\nNot Enough Memory - Check System");
            getch();
            abort();
}

_dos_findfirst(filespec,_A_NORMAL, &c_file);
strcpy(buffer[i], c_file.name);


while(_dos_findnext(&c_file) == 0){

        // First allocate memory

        if(i == 499){
        printf("\nToo Many Files-Delete Files");
        getch();
        goto quit_for_now;
        }
        if((buffer[++i] = (char *)malloc(13))==0){
        printf("\nNot Enough Memory - Check System");
        getch();
        goto quit_for_now;
        }
        strcpy(buffer[i], c_file.name);
}

pages = i/MAX_ENTRIES+1;                                    // number of MAX_ENTRIES
entry pages. 1 to n
pagenum = 1;                                // st with page number one

for(j=(pagenum-1)*MAX_ENTRIES; j<(MAX_ENTRIES*pagenum) && j<=i; j++){
                k=j/st_c1;                                                      /
/ st_c1 entries per column
                if(j== 0 || j==st_c1 || j==st_c2)row = 0;
                else row++;
        _settextposition(row+st_ROW, st_c+(k*tcw));    // give tcw chars to each
 col
        _outtext(buffer[j]);
}


while(1){
        k = jj = row = 0;
        _settextposition(row+st_ROW, ptr_st_c+(k*tcw));
        _outtext(prompt_ch);
        highlite(buffer[((pagenum-1)*MAX_ENTRIES) + jj],row,k);

    for (;;) {
       key = getfunc();
          k = jj/st_c1;                                     // jj is the entry counte
r from 0 to 44
                                                                  // determine whic
h
```

```
 column as jj=o to 44
                                                        // k = o for col-
1, and 2 for col-3
                                                        // row = o to 14
always = st_c1 rows
       switch(key) {
             default:
                    break;
                        case 27:
                                    goto quit_for_now;
                                    break;
             case 13:
                    goto endselection;
                    break;
                        case KY_UP:
                                    if(jj>0){
                                          _settextposition(row+st_ROW, ptr_
st_c+(k*tcw));
                                          _outtext(prompt_blk);
                                          unhighlite(buffer[((pagenum-1)*MA
X_ENTRIES) + jj],row,k);
                                          jj--; row--;
                                           if(jj== 0 )row = 0;
                                           if(jj==14 || jj==29)row = 14;
                                           k = jj/st_c1;
                                          _settextposition(row+st_ROW, ptr_
st_c+(k*tcw));
                                          _outtext(prompt_ch);
                                          highlite(buffer[((pagenum-1)*MAX_
ENTRIES) + jj],row,k);
                                    }
                              break;
                        case  KY_DN:
                                    if( pagenum < pages )
                                          last_entry = 44;
                                    else
                                          last_entry = i-(MAX_ENTRIES*(pag
enum-1));
                                    if(jj < last_entry ){
                                          _settextposition(row+st_ROW, ptr_
st_c+(k*tcw));
                                          _outtext(prompt_blk);
                                          unhighlite(buffer[((pagenum-1)*MA
X_ENTRIES) + jj],row,k);
                                          jj++;row++;
                                    if(jj== 0 || jj==st_c1 || jj==st_c2)row = 0;
                                          k = jj/st_c1;
                                          _settextposition(row+st_ROW, ptr_
st_c+(k*tcw));
                                          _outtext(prompt_ch);
                                          highlite(buffer[((pagenum-1)*MAX_
ENTRIES) + jj],row,k);
                                    }
                              break;
                        case  KY_RIGHT:
                                    if( pagenum < pages )
                                          last_entry = 44;
                                    else
                                          last_entry = i-(MAX_ENTRIES*(pag
enum-1));
                                    if(jj <= last_entry- 15 ){
                                          _settextposition(row+st_ROW, ptr_
st_c+(k*tcw));
```



```
                                          _outtext(prompt_blk);
                                          unhighlite(buffer[((pagenum-1)*MA
X_ENTRIES) + jj],row,k);
                                          jj = jj +15;
                                    if(jj== 0 || jj==st_c1 || jj==st_c2)row = 0;
                                          k = jj/st_c1;
                                          _settextposition(row+st_ROW, ptr_
st_c+(k*tcw));
                                          _outtext(prompt_ch);
                                          highlite(buffer[((pagenum-1)*MAX_
ENTRIES) + jj],row,k);
                                    }
                              break;
                        case  KY_LEFT:
                                    if( (jj-15) >= 0){
                                          _settextposition(row+st_ROW, ptr_
st_c+(k*tcw));
                                          _outtext(prompt_blk);
                                          unhighlite(buffer[((pagenum-1)*MA
X_ENTRIES) + jj],row,k);
                                          jj = jj -15;
                                    if(jj== 0 || jj==st_c1 || jj==st_c2)row = 0;
                                          k = jj/st_c1;
                                          _settextposition(row+st_ROW, ptr_
st_c+(k*tcw));
                                          _outtext(prompt_ch);
                                          highlite(buffer[((pagenum-1)*MAX_
ENTRIES) + jj],row,k);
                                    }
                              break;
                        case KY_PGDN:
                        case 'D':
                                    if(pagenum < pages)pagenum++;
                                    k = jj = row = 0;
                              clear_text();
                                    for(j=(pagenum-1)*MAX_ENTRIES; j<(pagenum*MAX_ENT
RIES) && j<=i; j++){
                                    _settextposition(row+st_ROW, st_c+(k*tcw));      /
/ give tcw chars to each col
                                      _outtext(buffer[j]);
                                                jj++;
                                    if(jj== 0 || jj==st_c1 || jj==st_c2)row = 0;
                                                else row++;
                                                k = jj/st_c1;
                                    }
                                    row = jj = k = 0;
                                    _settextposition(row+st_ROW, ptr_st_c+(k*tcw));
                                    _outtext(prompt_ch);
                                    highlite(buffer[((pagenum-1)*MAX_ENTRIES) + jj],r
ow,k);
                                    break;

                        case  KY_PGUP:
                        case  'U':
                                    if(pagenum > 1)pagenum--;
                                    k = jj = row = 0;
                              clear_text();
                                    for(j=(pagenum-1)*MAX_ENTRIES; j<(pagenum*MAX_ENT
RIES) && j<=i; j++){
                                    _settextposition(row+st_ROW, st_c+(k*tcw));      /
/ give tcw chars to each col
                                      _outtext(buffer[j]);
                                                jj++;
                                    if(jj== 0 || jj==st_c1 || jj==st_c2)row = 0;
```

```
                        else row++;
                                k = jj/st_c1;
                        }
                        row = jj = k = 0;
                        _settextposition(row+st_ROW, ptr_st_c+(k*tcw));
                        _outtext(prompt_ch);
                        highlite(buffer[((pagenum-1)*MAX_ENTRIES) + jj],r
ow,k);
                        break;
      }
   }
}
endselection:
        strcpy(filename, buffer[((pagenum-1)*MAX_ENTRIES) + jj]);
        for(j=0; j<=i; j++)free(buffer[j]);
        return(0);


//      Make sure that all allocations are properly freed

quit_for_now:
        for(j=0; j<=i; j++)free(buffer[j]);
        return(1);
}


void highlite(char *buffer,int row,int k)
{
int old_cor;
        old_cor = _gettextcolor();
        _settextposition(row+st_ROW, st_c+(k*tcw));
        if(small_display_flag == 1)_settextcolor(17);
        else _settextcolor(31);
        _outtext(buffer);
        _settextcolor(old_cor);
}
void unhighlite(char *buffer,int row,int k)
{
int old_cor;
        old_cor = _gettextcolor();
        _settextcolor(old_cor);
        _settextposition(row+st_ROW, st_c+(k*tcw));
        _outtext(buffer);
}
```

File: huntdefs.c

Wheel Hunting Data Acquisition Software

```
////////////////////////////////////////////////////////////
// File: huntdefs.c
//                                                        //
// Wheel Hunting Data Acquisition Software                //
// Main program                                           //
//                                                        //
// Copyright 1995: IEM Corporation                        //
//                                                        //
// Proprietary Licensed Information.                      //
// Not to be duplicated, used or disclosed                //
// except under terms of license.                         //
//                                                        //
// Revision History:                                      //
////////////////////////////////////////////////////////////

#include "hunthead.h"   // Compiler includes
#include "huntext.h"    // External variable declarations
#include "huntstru.h"   // Constant and structure declarations
#include "huntllb.h"    // Function Prototypes


extern  float   profile_length;
extern  int             delta_y_flag;
extern  int             how_often_plot;
extern  float   BL_YOFFSET;
extern  float   TL_XOFFSET;
extern  float   delta_y;
extern  float   right_limit;
extern  float   xoffset;                // coordinate system x offset point
extern  float   yoffset;                // coordinate system y offset point
extern  int   desired_freq;

int  read_def_values(char *filename, struct profile *default_struct, int *data_sm
ooth_flag)
{
    FILE *stream;
    char buffer[MAX_LINE_LENGTH];

    clear_profile_info(default_struct);

    if((stream = fopen(filename,"r"))!=NULL){
        // add checking for ret code
        fgets(default_struct->operator_id,MAX_LINE_LENGTH,stream);
        default_struct->operator_id[strlen(default_struct->operator_id)-1]='\0';

        fgets(default_struct->location_id,MAX_LINE_LENGTH,stream);
        default_struct->location_id[strlen(default_struct->location_id)-1]='\0';

        fgets(default_struct->loco_numb,MAX_LINE_LENGTH,stream);
        default_struct->loco_numb[strlen(default_struct->loco_numb)-1]='\0';

        fgets(default_struct->loco_type,MAX_LINE_LENGTH,stream);
        default_struct->loco_type[strlen(default_struct->loco_type)-1]='\0';

        fgets(default_struct->wheel_dia,MAX_LINE_LENGTH,stream);
        default_struct->wheel_dia[strlen(default_struct->wheel_dia)-1]
            ='\0';

        fgets(default_struct->std_overlay,MAX_LINE_LENGTH,stream);
        default_struct->std_overlay[strlen(default_struct->std_overlay)-1]= '\0';

        fgets(buffer,MAX_LINE_LENGTH,stream);
        default_struct->no_of_points = (int)atol(buffer);

        fgets(buffer,MAX_LINE_LENGTH,stream);
        default_struct->frequency = (float)atoi(buffer);
```



```
                desired_freq = (int)default_struct->frequency;

        fgets(buffer,MAX_LINE_LENGTH,stream);
        if(buffer[0]=='Y' || buffer[0]=='y')
            delta_y_flag = TRUE;
        else
            delta_y_flag= FALSE;

        fgets(buffer,MAX_LINE_LENGTH,stream);
        how_often_plot = atoi(buffer);
                if(how_often_plot <= 0) how_often_plot = 20;

        fclose(stream);
        return(TRUE);
    }
    else {

        // File did not exist, make one
        strcpy(default_struct->operator_id,"Zahid");
        strcpy(default_struct->location_id, "IEM");
        strcpy(default_struct->loco_numb, "0001");
        strcpy(default_struct->loco_type, "AMFLEET");
        strcpy(default_struct->wheel_dia, "26.000");

        strcpy(default_struct->std_overlay,"NOFILE");
        *data_smooth_flag= FALSE;
        default_struct->no_of_points = 500;
        default_struct->frequency = 50.0f;
        delta_y_flag = FALSE;
                how_often_plot = 20;
                delta_y = 0.10f;
        write_def_values(filename, default_struct, *data_smooth_flag);
        return(FALSE);
    }
}

int  write_def_values(char * filename, struct profile *default_struct, int data_s
mooth_flag)
{
    FILE *stream;
    char buffer[MAX_LINE_LENGTH];

    if((stream = fopen(filename,"w"))!=NULL){

        // add checking for ret code, non-zero is failure

        fputs(default_struct->operator_id,stream);
        fputs("\n",stream);
        fputs(default_struct->location_id,stream);
        fputs("\n",stream);
        fputs(default_struct->loco_numb,stream);
        fputs("\n",stream);
        fputs(default_struct->loco_type,stream);
        fputs("\n",stream);
        fputs(default_struct->wheel_dia,stream);
        fputs("\n",stream);
        fputs(default_struct->std_overlay,stream);
        fputs("\n",stream);
        sprintf(buffer,"%d",default_struct->no_of_points);
        fputs(buffer,stream);
        fputs("\n",stream);
        sprintf(buffer,"%d",(int)default_struct->frequency);
        fputs(buffer,stream);
        fputs("\n",stream);
        if(delta_y_flag == TRUE)
        {
```

```
            fputs("Y",stream);
            fputs("\n",stream);
        }
        else
            {
            fputs("N",stream);
            fputs("\n",stream);
        }

        sprintf(buffer,"%3d",how_often_plot);                    // scan length
        fputs(buffer,stream);
        fputs("\n",stream);

        fclose(stream);
        return(TRUE);
    }
    else
        {
        return(FALSE);
    }
}
```

File: huntplot.c

Wheel Hunting Data Acquisition Software

```
////////////////////////////////////////////////////////////
// File: huntplot.c                                       //
//                                                        //
// Wheel Hunting Data Acquisition Software                //
                        //
// Plot Program                                           //
//                                                        //
// Copyright 1995: IEM Corporation                        //
//                                                        //
// Proprietary Licensed Information.                      //
// Not to be duplicated, used or disclosed                //
// except under terms of license.                         //
//                                                        //
// Revision History:                                      //
////////////////////////////////////////////////////////////

#include "hunthead.h"   // Compiler includes
#include "huntstru.h"   // Constant and structure declarations
#include "huntllb.h"    // Function Prototypes

#undef   BLACK                /*  avoid redef warnings      */
#undef   RED
#undef   WHITE
#include <graphic.h>          /* Include all needed files */

extern  long    bkgnd;
extern  short   frgnd;

extern  unsigned int    map_seg;        /* screen address for fast image restore
*/
extern  unsigned int    map_offset;     /* screen offset address  */

extern  char    datebuf[];
extern  char    timebuf[];
extern  int             small_display_flag;
extern  int     useaccel_2;

#define  Y_START        -1.0f
#define  Y_END          +1.0f
#define  Y_INC          +0.5f


#define  TEXT_START_X           0.5f
#define  TEXT_START_Y           0.0f

#define  TEXT_START_X_COL2      2.0f
#define  TEXT_START_X_COL3      3.7f
#define  TEXT_START_X_COL4      4.9f


#define  TEXT_START_Y_1         0.9f
#define  TEXT_START_Y_2         0.6f
#define  TEXT_START_Y_3         0.3f

extern int keyboard_flag;
extern int delta_y_flag;

extern  float huge x[MAX_I_POINTS], y[MAX_I_POINTS];

#if DELTA_DESIRED
extern  float huge d[MAX_I_POINTS], z[MAX_I_POINTS];
#else
extern  float huge d[1], z[1];
#endif

int             grid_flag = 0;          // local grid control
```



```
extern  float   xcord_fnt;              //local global
void    prnt_calcs(struct profile * current_struct, char *area, int option);
char    *ftoaa(float innum, int precision);
void    drawto(float xf,float yf,float xf2,float yf2);
void    draw(int pnum,float yf);
void    init_plot(void);
void    *getfield(FILE *stream, char *buffer);
void    skiptoeol(FILE *stream);

void plot_data(int option,
struct profile * current_struct,                // data structure
struct profile * overlay_struct)                // overlay structure
{
    int         i, done = 0, boxdrawn = 0;
    int         npts, good_points_y, good_points_z, nptsovl, original_npts;
    float       max_difference = 0.02f, min_difference = -0.02f, grads = .01f;
    char        tempbuf[30], *buffer, *buffer2;
        float   dist, xstart=0.0f, xend=5.0f, ystart=Y_START, yend=Y_END, xinc=1.
0f, yinc=Y_INC;
        float   *fx, *fy, xdif = 9999.0f, ydif = 9999.0f, theta = 0.0f;
        int             *find,ii, line1, line2, ch, j, bug;
        float   tempfloat;
        float   xaxis_temp, yaxis_temp;
        FILE    *stream, *streamout;
        char    dummy[20];

    if((fx = (float *)(malloc(64)))==NULL){
        printf("\nNot Enough Memory - Check System");
            getalpha();
        goto home;
        }
    if((fy = (float *)(malloc(64)))==NULL){
        printf("\nNot Enough Memory - Check System");
            getalpha();
        goto home;
        }
    if((find = (int *)(malloc(sizeof(int))))==NULL){
        printf("\nNot Enough Memory - Check System");
            getalpha();
        goto home;
        }
    if((buffer = (char *)(malloc(100)))==NULL){
        printf("\nNot Enough Memory - Check System");
            getalpha();
        goto home;
        }
    if((buffer2 = (char *)(malloc(100)))==NULL){
        printf("\nNot Enough Memory - Check System");
            getalpha();
        goto home;
        }

        if( small_display_flag == TRUE){
                line1 = line2 = WHITE;
        } else {
                line1 = RED;
                line2 = CYAN;
        }

    max_difference = -999.0f;
    min_difference = 999.0f;

    for (i=0; i< MAX_PLOT_POINTS; i++){
        x[i] = y[i] = 0.0f;     // Make sure all output is zero
    }
```

```
#if DELTA_DESIRED
    for (i=0; i< MAX_PLOT_POINTS; i++){
        z[i] = d[i] = 0.0f;     // Make sure all output is zero
    }
#endif

    good_points_y = good_points_z = 0;
    npts = original_npts = current_struct->no_of_points;


        // For now, use these defaults

        xend = (float)npts;
        xinc = (float)npts/5.0f;
        if(xinc < 1.0f)xinc = 10.0f;

        strcpy(buffer, current_struct->data_file);
        strcat(buffer, ".dat");
        if((stream = fopen(buffer,"r"))==NULL){
        error_out( FILE_READ, "Cannot read DATA File-Check File");
                goto home;
        }

    for(ii=0; ii< npts; ii++) {
                if(feof(stream))break;

                getfield(stream, buffer);
                getfield(stream, buffer2);
                skiptoeol(stream);

//              fscanf(stream,"%s %s %s %s %s\n",buffer,dummy,buffer2,dummy,dummy
);
//              printf("%s %s\n", buffer, buffer2);
                if(useaccel_2 == FALSE)
                        yaxis_temp = (float)atof(buffer);
                else
                        yaxis_temp = (float)atof(buffer2);

//              printf("%4.2f", yaxis_temp);
//              getfunc();

                xaxis_temp = (float)ii;
           x[good_points_y] = xaxis_temp;
           y[good_points_y] = yaxis_temp;
               good_points_y++;
    }
        _settextposition(10,10);
        printf("%5d", ii);

    fclose(stream);

        if(good_points_y == 0){
                error_out(NO_RECS,"Profile all zeros");
                goto home;
        }

    /* If a valid overlay filename was specified, copy all the needed
           points from the overlay structure to z array.
        */

    good_points_z = 0;
    if(strcmpi(overlay_struct->std_overlay,"NOFILE") != 0){
                strcpy(buffer, overlay_struct->data_file);
                strcat(buffer, ".dat");
//              printf("Overlay 2 %s",buffer);
//              getfunc();
```



```
                if((stream = fopen(buffer,"r"))==NULL){
            error_out( FILE_READ, "Cannot read Overlay DATA File-Check File");
                        goto home;
                }
            nptsovl = overlay_struct->no_of_points;
        for(i=0; i< nptsovl; i++) {
                        if(feof(stream))break;

                        getfield(stream, buffer);
                        getfield(stream, buffer2);
                        skiptoeol(stream);

//                      fscanf(stream,"%s %s %s %s %s\n",buffer,dummy,buffer2,dum
my,dummy);
                        xaxis_temp = (float)i;
                        if(useaccel_2 == FALSE)
                                yaxis_temp = (float)atof(buffer);
                        else
                                yaxis_temp = (float)atof(buffer2);
                        if((float)yaxis_temp>=ystart && (float)yaxis_temp<=yend )
{
                        z[i] = yaxis_temp;
                                d[i] = xaxis_temp;      // copy x for plot
                        }
            good_points_z++;    // omit all trailing zeros
        }
                if(good_points_z == 0){
                        error_out(NO_RECS,"Overlay all zeros");
                        goto home;
                }
    }

    /* Need to display the graph always. So prepare the screen */


redoplot:               // may need to move this above

    prepare_display_text(option, current_struct->std_overlay,
    overlay_struct->std_overlay);

        symht(.02);

        // always display net area when an overlay is specified

    if(strcmpi(overlay_struct->std_overlay,"NOFILE") != 0){
//      buffer = neta(current_struct->area,overlay_struct->area);
//              prnt_calcs(current_struct,buffer, 1);
    } else
//      prnt_calcs(current_struct,current_struct->area, 0);

    color(WHITE);
    sympick(12);                        /* Filled circle symbols */

    if(delta_y_flag == TRUE){
        if(strcmpi(overlay_struct->std_overlay,"NOFILE") != 0){
            for(i=0; i<good_points_z; i++) {
                d[i] = z[i] - y[i];             // overwrite d[] if delta needed
                if((float)d[i] > max_difference)max_difference = (float)d[i];
                if((float)d[i] < min_difference)min_difference = (float)d[i];
                //                                                  printf("%7.4f %7.
4f %7.4f\n", d[i], max_difference, min_difference);
            }
            if(min_difference == max_difference){
                min_difference = -0.02f;
                max_difference = +0.02f;
                for(i=0; i<good_points_z; i++)d[i] = 0.0f;      // set to some va
l
```

```
ue
            }
            grads = (float)((fabs(min_difference) + max_difference)/4.0f);
            graf("%3.0f",xstart, xinc, xend,"%7.4f",min_difference,grads ,max_dif
ference);
                        color(line1);
            curve(x,d,good_points_z,0);

        }
        else {
            endplot();
            stopplot();
            _setvideomode(_DEFAULTMODE);
            _settextcolor( frgnd );
            _setbkcolor( bkgnd );
            _settextposition(10, 20);
            _outtext("Please specify comparison profile!");
            getalpha();
            goto home;
        }
    }
    else {
                dashf(9);
        graf("%3.0f",xstart, xinc, xend,"%3.1f",ystart, yinc, yend);
                color(line1);
                for(bug=0; bug<good_points_y-1; bug++){
                  if(x[bug]>=xstart && x[bug]<xend
                  && x[bug+1]>=xstart && x[bug+1]<xend
                  && y[bug]>=ystart && y[bug]<yend
                  && y[bug+1]>=ystart && y[bug+1]<yend)
                                            uline(x[bug],y[bug],x[bug+1],y[bu
g+1],3);
                }
        dashf(8);

        /* now put up the overlay profile only if the valid overlay specified */

        if(strcmpi(overlay_struct->std_overlay,"NOFILE") != 0){
                        color(line2);
            curve(d,z,good_points_z,0);
        }
    }


        // Now perform some magic on the display
        //
        // Support zf (zoom flange area)
        // zt (zoom tread area), mr (move to right),ml,mu, and md (move) commands
.
        // Also, support smoothing of two types and save final image.
        // Also, possible macros for doing all this.
        // Also, a spread sheet for math equation interface.
        //

while(1){

tryagain:


        done = 0;
        *find = 0;
        boxdrawn = 0;

        curson(0, 0, &fx, &fy, find);

        // Now switch coordinates if defined in reverse
```



```
        if(*fx > *(fx+1)){
                tempfloat = *(fx+1);
                *(fx+1) = *fx;
                *fx = tempfloat;
        }
        if(*fy > *(fy+1)){
                tempfloat = *(fy+1);
                *(fy+1) = *fy;
                *fy = tempfloat;
        }

        symht(0.25F);
        color(WHITE);

        if(*find >= 2){
                uline(*fx,*fy,*fx, *(fy+1), 1);
                uline(*fx,*(fy+1),*(fx+1), *(fy+1), 0);
                uline(*(fx+1),*(fy+1),*(fx+1), *(fy), 0);
                uline(*(fx+1),*(fy),*(fx), *(fy), 0);
                boxdrawn = 1;
        }

        while(!done){
                ch=getch();
                color(BLACK);
                if(*find >= 2 && boxdrawn == 1){
                        uline(*fx,*fy,*fx, *(fy+1), 1);
                        uline(*fx,*(fy+1),*(fx+1), *(fy+1), 0);
                        uline(*(fx+1),*(fy+1),*(fx+1), *(fy), 0);
                        uline(*(fx+1),*(fy),*(fx), *(fy), 0);
                        boxdrawn = 0;
                }


                switch(ch){

                        case 'D':
                        case 'd':
                                                if(*find >= 2){
                                                dist=(float)sqrt(((*fx - *(fx+1)) * (*fx
- *(fx+1)))+

                                                ((*fy - *(fy+1)) * (*fy - *(fy+1))));

                                                buffer = ftoaa(dist, 3);
                                                if(small_display_flag == TRUE){
                                                        prtfnt(2.5f,6.7f,"D =",0.15f,0);
                                                        prtfnt(3.0f,6.7f,buffer,0.15f,0);
                                                        getch();
                                                        prtfnt(2.5f,6.7f,"              ",
0.15f,0);
                                                } else {
                                                        prtfnt(7.5f,6.7f,"D =",0.15f,0);
                                                        prtfnt(8.0f,6.7f,buffer,0.15f,0);
                                                        getch();
                                                        prtfnt(7.5f,6.7f,"              ",
0.15f,0);
                                                }
                                                }
                                                break;
                        case 'Z':
                        case 'z':
                                                if(*find >= 2 ){
                                                if(*fx > xstart && *fx < xend)
                                                        xstart = *fx;
                                                if(*(fx+1) >  xstart && *(fx+1) < xend)
```

```
                        xend   = *(fx+1);
                    if(*fy > ystart && *fy < yend)
                        ystart = *fy;
                    if(*(fy+1) > ystart && *(fy+1) < yend)
                        yend   = *(fy+1);

                        xinc   = (float)fabs((double)xsta
rt - (double)xend)/5.0f;
                        yinc   = (float)fabs((double)ysta
rt - (double)yend)/5.0f;

                        if( xinc < .05f) xinc = .05f;
                        if( yinc < .05f) yinc = .05f;

                        done = 1;
                    }
                    break;
            case 'S':
            case 's':
                    if(*find >= 2 ){
                    if(*fx > xstart && *fx < xend)
                        xstart = *fx;
                    if(*(fx+1) >  xstart && *(fx+1) < xend)
                        xend   = *(fx+1);
                    if(*fy > ystart && *fy < yend)
                        ystart = *fy;
                    if(*(fy+1) > ystart && *(fy+1) < yend)
                        yend   = *(fy+1);

                    xinc   = (float)fabs((double)xstart - (do
uble)xend)/5.0f;
                    yinc   = (float)fabs((double)ystart - (do
uble)yend)/5.0f;

                    if( xinc < .05f) xinc = .05f;
                    if( yinc < .05f) yinc = .05f;
                    done = 1;

                    }
                    break;
            case 'P':
            case 'p':                           // previous setti
ngs
                    xstart=0.0f;
                    xend=(float)original_npts;
                    xinc = (float)npts/5.0f;
                    if(xinc < 1.0f)xinc = 10.0f;

                    ystart=Y_START;
                    yend=Y_END;
                    yinc=Y_INC;
                    done = 1;
                    break;

            case 'C':
            case 'c':       // redraw
                    done = 1;
                    break;


            case 'F':
            case 'f':
                    curvefollow(0,0,x,y,good_points_y);
                    done = 1;
                    break;
            case 'Y':
```



```
                    case 'y':
                                printf("Points Captured: %4d\n", good_poi
nts_y);
                                done = 1;
                                getch();
                                break;
                    case 'G':
                    case 'g':
                                if(grid_flag == 0) grid_flag = 1;
                                else grid_flag = 0;
                                done = 1;
                                break;

                    case 27:
                                done = 1;
                                break;

                    case 'Q':       // save always to temporary buffer
                    case 'q':
                                goto hometime;

                    default:
                                done = 1;               // quit on anyoth
er char
                                break;
                }
        }

        if(ch == 'z' || ch == 'Z'|| ch == 'p'|| ch == 'P'|| ch == 'G'|| ch == 'C'
 || ch == 'c' || ch == 'g'
          || ch == 's' || ch == 'S'){

    endplot();
        stopplot();
        if(ch == 's' || ch == 'S'){

    _clearscreen( _GCLEARSCREEN );
        printf("Now ready to save new points");

         strcpy(buffer, current_struct->data_file);
         strcat(buffer, ".dat");
         if((stream = fopen(buffer,"r"))==NULL){
        error_out( FILE_READ, "Cannot read DATA File-Check File");
                goto donothing;
         }
         for(ii=0; ii< xstart; ii++) {
                if(feof(stream))break;
                getfield(stream, buffer);
                getfield(stream, buffer2);
                skiptoeol(stream);
     }
         buffer[0]='\0';
     _clearscreen( _GCLEARSCREEN );
        prompt("Enter output file (.DAT) without extension: ", ALPHANUMERIC, 8,
buffer);
         if(buffer[0] != '\0'){
                strcpy(buffer2, buffer);
                strcat(buffer, ".dat");
         }
         else goto donothing;
         streamout = fopen(buffer,"w");

         for(ii=xstart; ii< xend; ii++) {
                if(feof(stream))break;
                fgets(buffer,100,stream);
        j = strlen(buffer);
```

```
        buffer[j] = '\0';
                fputs(buffer,streamout);
        }
        fclose(streamout);

           // Now create a prf file with adjusted number of points

           strcpy(dummy, buffer2);                                          // rememb
er old file name
           strcat(buffer2, ".prf");                                         // create
 a new user prf file
           streamout = fopen(buffer2,"w");
           strcpy(buffer, current_struct->data_file);     // open original prf file
           strcat(buffer, ".prf");

           if((stream = fopen(buffer,"r"))==NULL){
          error_out( FILE_READ, "Cannot read PROFILE File-Check File");
                goto donothing;
           }
           for(ii=0; ii<5 ; ii++) {                                         // copy first 5 lines
                if(feof(stream))break;
                fgets(buffer,100,stream);
          j = strlen(buffer);
          buffer[j] = '\0';
                fputs(buffer,streamout);
        }
        fprintf(streamout, "%-10s      ...Data File\n",dummy);
           fgets(buffer,100,stream);                                        // skip over next line
           for(ii=0; ii<2 ; ii++) {                                         // copy 7,8 lines
                if(feof(stream))break;
                fgets(buffer,100,stream);
          j = strlen(buffer);
          buffer[j] = '\0';
                fputs(buffer,streamout);
        }

        itoa((xend-xstart+1), buffer, 10);         // get current sample number
        fprintf(streamout, "%-10s      ...Total Readings\n",buffer);
           fgets(buffer,100,stream);                                        // skip over next line
           fgets(buffer,100,stream);              // copy frequency line
        j = strlen(buffer);
        buffer[j] = '\0';
           fputs(buffer,streamout);

donothing:
        fclose(stream);
        fclose(streamout);

           }
                goto redoplot;
           }
           goto tryagain;
}

hometime:
       endplot();
       stopplot();

       if(option == DISPLAY_NPRINT){

           _setvideomode(_DEFAULTMODE);
                printf("Press Y/y to confirm printing ");
                ch = getch();
                if(ch != 'Y' && ch!='y')goto home;

           if ( printer_ok() == 1){
```



```
                prepare_print_text();

                // Also, put up the remaining items on the output

                _strdate(datebuf);
                _strtime(timebuf);

                prtfnt((float)TEXT_START_X,(float)TEXT_START_Y_1,   "DATA ....:",.15f
,0);
                prtfnt((float)TEXT_START_X_COL2,(float)TEXT_START_Y_1,
                current_struct->std_overlay,.15f,0);

       if(strcmpi(overlay_struct->std_overlay,"NOFILE") != 0){
                prtfnt((float)TEXT_START_X_COL3,(float)TEXT_START_Y_1,"OVL _ _ :",.15
f,0);
                strcpy(tempbuf,overlay_struct->std_overlay);
                strcat(tempbuf,".prf");
                prtfnt((float)TEXT_START_X_COL4,(float)TEXT_START_Y_1,tempbuf,.15f,0)
;
           }

#if 0
                prtfnt((float)TEXT_START_X,(float)TEXT_START_Y_2,   "Loco / Car: ",.1
5f,0);
                prtfnt((float)TEXT_START_X_COL2,(float)TEXT_START_Y_2,
                current_struct->loco_numb ,.15f,0);
#endif

                prtfnt((float)TEXT_START_X,(float)TEXT_START_Y_2,   "Date: ",.15f,0);
                prtfnt((float)TEXT_START_X_COL2,(float)TEXT_START_Y_2,datebuf ,.15f,0
);

//              prtfnt((float)TEXT_START_X,(float)TEXT_START_Y_3,"Wheel Serial: ",.15
f,0);
//              prtfnt((float)TEXT_START_X_COL2,(float)TEXT_START_Y_3,
//            current_struct->wheel_serial,.15f,0);

                prtfnt((float)TEXT_START_X,(float)TEXT_START_Y_3,"Time: ",.15f,0);
                prtfnt((float)TEXT_START_X_COL2,(float)TEXT_START_Y_3,timebuf,.15f,0)
;

                        // Now print the calculations

                    if(strcmpi(overlay_struct->std_overlay,"NOFILE") == 0){
//                        prtfnt(TEXT_START_X-.00f ,0.0f,"FT:",.15f,0);
//                            prtfnt(TEXT_START_X+.50f ,0.0f,current_struct->fl
ange_thickness,.15f,0);
//                      prtfnt(TEXT_START_X+1.0f,0.0f,"   FH:",.15f,0);
//                      prtfnt(TEXT_START_X+2.00f,0.0f,current_struct->flange_hei
ght,.15f,0);
                        }

                    if(strcmpi(overlay_struct->std_overlay,"NOFILE") != 0){
//                      prtfnt(TEXT_START_X+0.75f,0.0f,"DIF AREA (sq-in):",.15f,0
);
//                      buffer = neta(current_struct->area,overlay_struct->area);
//                            prtfnt(TEXT_START_X+3.25f,0.0f,buffer,.15f,0);
                        } else {
//                      prtfnt(TEXT_START_X+3.00f,0.0f,"AREA:",.15f,0);
//                            prtfnt(TEXT_START_X+3.75f,0.0f,current_struct->ar
ea,.15f,0);
                        }

           color(WHITE);
           sympick(12);                         /* Filled circle symbols */
```

```
                        dashf(1);                                       /
/ choose dash type 1
            if(delta_y_flag == FALSE){
                        graf("%4.0f",xstart, xinc, xend,"%3.1f",ystart, yinc, yen
d);
                curve(x,y,good_points_y,0);
                dashf(8);
                if(strcmpi(overlay_struct->std_overlay,"NOFILE") != 0){
                    curve(x,z,good_points_z,0);
                }
            }
            else {
                graf("%3.0f",xstart, xinc,xend,"%3.2f",min_difference,grads,max_d
ifference);
                curve(x,d,good_points_z,0);
            }

          hardcopy('L');
            endplot();
            stopplot();
        }
        else{
            error_out(0,"Printer not ready!");
        }
    }

    _setvideomode(_DEFAULTMODE);
    _settextcolor( frgnd );
    _setbkcolor( bkgnd );
    color(WHITE);

home:

        // Free all space

         free(fx);
         free(fy);
         free(find);
         free(buffer);
         free(buffer2);
         fcloseall();
}

/* Display the text items on the screen for display only */

void prepare_display_text(int option, char *buffer1, char *buffer2)
{
    char        databuf[80];
        int             backgrnd;


        if( small_display_flag == TRUE){
                backgrnd = BLACK;
        } else {
                backgrnd = BLUE;
        }

    bgnplot(1, 'g', "notek");
    startplot(backgrnd);    /* Initializes plot and sets the background color */

    metricunits(0);      /* Ensure scaling in inch units */
    rotate(0);           /* do not rotate the screen */
        if( small_display_flag == FALSE)
            page(9.f, 6.88f);   /* Sets the page size */
```



```
        else{
            page(4.5f, 6.88f);   /* Sets the page size */
                pgshift(0.0f, 0.0f);
        }

    tmargin(0.0f);          /* Set the margins */
    xname("Chronological Points");
    if(delta_y_flag == FALSE)
        yname("Amplitude");
    else if(option != 99)
        yname("Delta Y ");
    else
        yname("Amplitude");

    heading(" Hunting Profile");

        if( small_display_flag == FALSE)
        physor(1.0f, 1.80f);
        else
        physor(0.0f, 2.05f);

        if( small_display_flag == FALSE)
            area2d(7.f, 3.20f);           /* Sets the area of the plot */
    else{
            area2d(4.25f, 3.5f);           /* Sets the area of the plot */
        }


        color(WHITE);              /* Axes names and heading will be black */
    grid(9);                    // Grid of solid line
    if(grid_flag == 1){
            xfgrid(9, 2);              /* Draws grid through tick marks, 9 --> fi
ne dot */
        yfgrid(9, 2);
    }
    upright(1);


        // No delta-y requested. Overlay is specified - dual trace

    if(delta_y_flag == FALSE && (strcmpi(buffer2,"NOFILE") != 0))
        prtfnt(xcord_fnt,.75f,"LEGEND   |4|DATA:....   |3|OVERLAY:__ ",.15f,0);
        else if(delta_y_flag == TRUE && (strcmpi(buffer2,"NOFILE") != 0)){

        // Delta-y requested. Overlay is specified - single trace

        prtfnt(xcord_fnt,.75f,"LEGEND   |4|DELTA DATA:....   ",.15f,0);
        }
        else {

        // Normal single trace condition specified

        prtfnt(xcord_fnt,.75f,"LEGEND   |4|DATA:....   ",.15f,0);
        }

        color(WHITE);

    if(keyboard_flag == TRUE){
        if( option == DISPLAY_NPRINT || option == DISPLAY_ONLY)
            prtfnt(xcord_fnt,0.5f,"PRESS ESC FOR CAD, Q TO QUIT",.15f,0);
        else
            prtfnt(xcord_fnt,0.5f,"PRESS R/B TO START,OTHERS TO END",.15f,0);
    }

    strcpy(databuf, "DATA: ");
    strcat(databuf, buffer1);
```

```
    if(strcmpi(buffer2,"NOFILE") != 0){
        strcat(databuf, " OVERLAY: ");  // fix space---
        strcat(databuf, buffer2);
        strcat(databuf,".prf");
                if(small_display_flag == FALSE)
                prtfnt(2.0f,1.0f,databuf,.15f,0);
        else
                prtfnt(0.3f,1.4f,databuf,.15f,0);

    }
    else{
                if(small_display_flag == FALSE)
                prtfnt(3.2f,1.0f,databuf,.15f,0);
                else
                prtfnt(0.0f,1.4f,databuf,.15f,0);
        }

}

/* Prepare the output information for print only */

void prepare_print_text()
{
    /* Redraw the plot only for a vertical looking graph */

    bgnplot(0, 't', "plotdump.tkf");
    startplot(BLACK);    /* Initializes plot and sets the background color */
    font(1,"simplex.fnt",'\310');
    metricunits(0);      /* Ensure scaling in inch units */
    rotate(1);                  // rotate the page 90 degrees
    page(6.884f, 9.f);   // Sets the page size
    tmargin(0.0f);
    lmargin(0.0f);
    rmargin(0.0f);
    xname("Chronological Points");
    if( delta_y_flag == FALSE)
        yname("Amplitude");
    else
        yname("Delta Y ");

    heading(" Data  Profile");
    area2d(5.0f, 5.0f);             /* Sets the area of the plot */
    physor(1.0f, 2.0f);
    color(WHITE);                   /* Axes names and heading will be black */
    grid(1);
//  xfgrid(2, SMALL_TICKS);              /* Draws grid through tick marks, 9 -->
fine dot */
//  yfgrid(2, SMALL_TICKS);
    upright(1);
    sympick(12);                                    /* Filled circle symbols */

}

// option = 0:  print area
// option = 1:  print net area only. do not print FT and FH

void prnt_calcs(struct profile * current_struct, char *area, int option)
{
if(option == 0){

#if 0
    prtfnt(xcord_fnt-.20f ,0.2f,"FT:",.15f,0);
    prtfnt(xcord_fnt+.25  ,0.2f,current_struct->flange_thickness,.15f,0);
    prtfnt(xcord_fnt+.75f,0.2f,"   FH:",.15f,0);
    prtfnt(xcord_fnt+1.5f,0.2f,current_struct->flange_height,.15f,0);
```



```
#endif

//  prtfnt(xcord_fnt+2.75f,0.2f,"AREA:",.15f,0);
//  prtfnt(xcord_fnt+3.50f,0.2f,area,.15f,0);

} else {

//  prtfnt(xcord_fnt+0.50f,0.2f,"DIF AREA (sq-in):",.15f,0);
//  prtfnt(xcord_fnt+3.00f,0.2f,area,.15f,0);

}
}

char    *neta(char *a1, char *a2)
{
float net_area;
char  *buffer, buf[20];
int       sign, decimal, i;

                buf[0] = '\0';
                buffer = (char *)malloc(10);
        net_area = (float)atof(a1) - (float)atof(a2);
        if( net_area < 0.0f ) net_area = -net_area;
                buffer = fcvt((double)net_area,4,&decimal,&sign);

                if(decimal > 0){
                        buffer[strlen(buffer)+2] = '\0';
                        for(i=strlen(buffer); i>= decimal; i--)
                                buffer[i+1] = buffer[i];
                        buffer[decimal] = '.';
                } else {
                        for(i=0;i<=-decimal;i++){
                                strcat(buf,"0");
                        }
                        strcat(buf, buffer);
                        buf[0]='.';
                        strcpy(buffer, buf);
                }
                free(buffer);
                return(buffer);
}
char    *ftoaa(float innum, int precision)
{
char  *buffer, buf[20];
int       sign, decimal, i;

                buf[0] = '\0';
                buffer = (char *)malloc(10);
                buffer = fcvt((double)innum,precision,&decimal,&sign);

                if(decimal > 0){
                        buffer[strlen(buffer)+2] = '\0';
                        for(i=strlen(buffer); i>= decimal; i--)
                                buffer[i+1] = buffer[i];
                        buffer[decimal] = '.';
                } else {
                        for(i=0;i<=-decimal;i++){
                                strcat(buf,"0");
                        }
                        strcat(buf, buffer);
                        buf[0]='.';
                        strcpy(buffer, buf);
                }
                free(buffer);
                return(buffer);
}
```

```
void draw(int pnum,float yf)
{
int     xx, yy,x,y;

// offset for negative stuff - add 2.0 to yf

x = pnum;                                               // 160 pixels/ 5 inches
y = (int)((yf+2.0f) * 25);              //  24 pixels/ 2 inches

xx = x  + 16;
yy = -y + 116;                                  // 100 pixels to scale
if(pnum > 0 && pnum < 158){
        if(xx < 176 && yy >= 16)_lineto(xx, yy);
}
else _moveto(xx,yy);
}

void init_plot(void)
{
_clearscreen( GCLEARSCREEN);
_moveto(16,16); _lineto(16,116);
_moveto(16,66); _lineto(176,66);
_settextposition(1,4); _outtext("Acceleration Data");
}

void *getfield(FILE *stream, char *bufferrr)
{
char    ch;
int             c;

for(c=0; (c<80) && ((ch=getc(stream)) != '\n') && ch!= ',' && !feof(stream); c++)
{
        bufferrr[c] = ch;
}
bufferrr[c] = '\0';

}
void skiptoeol(FILE *stream)
{
char    ch;
int             c;

        for(c=0; (c<80) && ((ch=getc(stream)) != '\n') && !feof(stream); c++);

}
```

```
////////////////////////////////////////////////////////////////
// File: covsrt.c                                             //
//                                                            //
// Wheel Hunting Data Acquisition Software                    //
                        //
//                                                            //
// Copyright 1995: IEM Corporation                            //
//                                                            //
// Proprietary Licensed Information.                          //
// Not to be duplicated, used or disclosed                    //
// except under terms of license.                             //
//                                                            //
// Revision History:                                          //
////////////////////////////////////////////////////////////////

#define SWAP(a,b) {swap=(a);(a)=(b);(b)=swap;}

void covsrt(float **covar, int ma, int ia[], int mfit)
{
        int i,j,k;
        float swap;

        for (i=mfit+1;i<=ma;i++)
                for (j=1;j<=i;j++) covar[i][j]=covar[j][i]=0.0;
        k=mfit;
        for (j=ma;j>=1;j--) {
                if (ia[j]) {
                        for (i=1;i<=ma;i++) SWAP(covar[i][k],covar[i][j])
                        for (i=1;i<=ma;i++) SWAP(covar[k][i],covar[j][i])
                        k--;
                }
        }
}
#undef SWAP
/* (C) Copr. 1986-92 Numerical Recipes Software :!S$-)=. */
```

File: xfit.c

Wheel Hunting Data Acquisition Software

```
////////////////////////////////////////////////////////////////
// File: xfit.c                                               //
//                                                            //
// Wheel Hunting Data Acquisition Software                    //
                          //
//                                                            //
// Copyright 1995: IEM Corporation                            //
//                                                            //
// Proprietary Licensed Information.                          //
// Not to be duplicated, used or disclosed                    //
// except under terms of license.                             //
//                                                            //
// Revision History:                                          //
////////////////////////////////////////////////////////////////
/* Driver for routine fit */

#include <stdio.h>
#define NRANSI
#include "nr.h"
#include "nrutil.h"

#define SPREAD 0.5
#define MAX_I_POINTS            9500 // max number of data points
extern  float   huge x[MAX_I_POINTS], y[MAX_I_POINTS];

int xfit(int NPT);

int xfit(int NPT)
{
        long idum=(-117);
        int i,mwt,j;
        float a,b,chi2,q,siga,sigb,*xx,*yy,*sig;
        FILE *stream;

        xx=vector(1,NPT);
        yy=vector(1,NPT);
        sig=vector(1,NPT);
        for (i=1;i<=NPT;i++) {
                xx[i] = x[i];
                yy[i] = y[i];
                sig[i]=SPREAD;
        }
        stream  = fopen("xfitout.dat","w");
        for (mwt=0;mwt<=1;mwt++) {
                fit(xx,yy,NPT,sig,mwt,&a,&b,&siga,&sigb,&chi2,&q);
                if (mwt == 0)
                        printf("\nIgnoring standard deviations\n");
                else
                        printf("\nIncluding standard deviations\n");
                printf("%12s %9.6f %18s %9.6f \n",
                        "a  =  ",a,"uncertainty:",siga);
                printf("%12s %9.6f %18s %9.6f \n",
                        "b  =  ",b,"uncertainty:",sigb);
                printf(stream,"%19s %14.6f \n","chi-squared: ",chi2);
                printf(stream,"%23s %10.6f \n","goodness-of-fit: ",q);
                fprintf(stream,"%12s %9.6f %18s %9.6f \n",
                        "a  =  ",a,"uncertainty:",siga);
                fprintf(stream,"%12s %9.6f %18s %9.6f \n",
                        "b  =  ",b,"uncertainty:",sigb);
                fprintf(stream,"%19s %14.6f \n","chi-squared: ",chi2);
                fprintf(stream,"%23s %10.6f \n","goodness-of-fit: ",q);
        }
        free_vector(sig,1,NPT);
        free_vector(yy,1,NPT);
        free_vector(xx,1,NPT);
        for(j=0; j<NPT; j++){
```



```
                fprintf(stream, "%4.2f , %4.2f\n", x[j], (a+b*x[j]) );
        }
        fclose(stream);
        return 0;
}
#undef NRANSI
/* (C) Copr. 1986-92 Numerical Recipes Software :!S$-)=. */
```

File: xrealft.c

Wheel Hunting Data Acquisition Software

```
////////////////////////////////////////////////////////////////
// File: xrealft.c                                            //
//                                                            //
// Wheel Hunting Data Acquisition Software                    //
                          //
//                                                            //
// Copyright 1995: IEM Corporation                            //
//                                                            //
// Proprietary Licensed Information.                          //
// Not to be duplicated, used or disclosed                    //
// except under terms of license.                             //
//                                                            //
// Revision History:                                          //
////////////////////////////////////////////////////////////////
/* Driver for routine realft */

#include "huntstru.h"   // Constant and structure declarations
#include <stdio.h>
#include <math.h>
#include <time.h>
#include <graph.h>
#define NRANSI
#include "nr.h"
#include "nrutil.h"

#define EPS 1.0e-3
#define WIDTH 50.0
#define PI 3.1415926

extern  int             show_bins;
extern  int             getfunc();
extern  float   huge x[MAX_I_POINTS], y[MAX_I_POINTS];
extern  float   huge x_spectrum[MAX_I_POINTSHALF], y_spectrum[MAX_I_POINTSHALF];
extern  void    show_spectrum(int,float,char *);
int     perform_rdfft(int npt, float sampling_freq, char *temp);        // real d
iscrete fft
void    compute_line(int points);
int             power[]={2,4,8,16,32,64,128,256,512,1024,2048,4096,8192,16384,327
68};

int perform_rdfft(int npt, float sampling_freq, char *file)             // real d
iscrete fft
{
        int i,n,nlim;
        float big,scal,*data,*size;
        char  starttime[20], endtime[20];
        long  npts;

        npts = (long)npt;
        for(i=1; i<15; i++){
                if(power[i]>npts)break;
        }
        npts = power[i-1];
        _settextposition(6,20);
    printf("DFT: %5d", npts);

        n = npts /2 ;

        data=vector(1L,npts);
        size=vector(1L,npts/2+1);

//      Now perform the analysis

                for (i=1;i<=npts;i++){
                        data[i]=x[i];
                }
```



```
                big = -1.0e10;

                _strtime(starttime);
                realft(data,npts,1);
                for (i=2;i<=n;i++) {
                        size[i]=(float)sqrt(SQR((float)data[2*i-1])+SQR((float)da
ta[2*i]));
                        if (size[i] > big) big=size[i];
                }
                size[1]=(float)fabs(data[1]);
                if (size[1] > big) big=size[1];
                size[n+1]=(float)fabs(data[2]);
                if (size[n+1] > big) big=size[n+1];
                scal=(float)WIDTH/big;
                _strtime(endtime);
                _settextposition(7,20);
                printf("%s-%s", starttime, endtime);
                for (i=1;i<=n;i++) {
                        nlim=(int) (0.5+scal*size[i]+EPS);
                        x_spectrum[i]=(float)i;
                        y_spectrum[i]=(float)nlim;
                }
        getfunc();

        if(show_bins == TRUE)
                show_spectrum(npts, sampling_freq, file);

        free_vector(size,1L,npts/2+1);
        free_vector(data,1L,npts);

        return (n);                                             // number of bins
 to expect
}
#undef NRANSI

void compute_line(int points)
{
        long    idum=(-117);
        int             i,j,k,maxi,start,z;
        float   a,b,chi2,q,siga,sigb,*xmax,*ymax,*sig;
        float   lowy, maxy;

        xmax=vector(1L,200);
        ymax=vector(1L,200);
        sig=vector(1L,200);
        k = 0;
        j=1;

        printf("Now computing cycles\n");

for(z=0; z<20; z++){
        lowy = 99.99f;
        for(i=k; i<points; i++){
                if(y[i] < lowy){                // find lowest point
                        lowy = y[i];
                        start = i;
                } else
                        break;
        }
        maxy=y[start];

        printf("Lowest # %3d %6.4f", start, y[start]);
        getfunc();

        for(k=start+1; k<points; k++){
                if(y[k] > maxy){
```

```
                        maxy=y[k];
                        maxi=k;
                } else
                        break;
        }
        xmax[j]=maxi;
        ymax[j]=y[maxi];
        j++;
}

        j--;                                    // throw away last point

        for(i=1; i<j; i++)printf("[#%3d(%6.3f %6.3f)]",i,xmax[i],ymax[i]);
        fit(xmax,ymax,j,sig,1,&a,&b,&siga,&sigb,&chi2,&q);
        printf("a= %9.6f  uncertainity= %9.6f\n",a,siga);
        printf("b= %9.6f  uncertainity= %9.6f\n",b,sigb);
        printf("Chi-sq= %9.6f\n",chi2);
        printf("Goodness= %9.6f\n",q);
        getfunc();
        free_vector(sig,1,200);
        free_vector(xmax,1,200);
        free_vector(ymax,1,200);
}
```

File: realft.c

Wheel Hunting Data Acquisition Software

```
////////////////////////////////////////////////////////////
// File: realft.c                                         //
//                                                        //
// Wheel Hunting Data Acquisition Software                //
                        //
//                                                        //
// Copyright 1995: IEM Corporation                        //
//                                                        //
// Proprietary Licensed Information.                      //
// Not to be duplicated, used or disclosed                //
// except under terms of license.                         //
//                                                        //
// Revision History:                                      //
////////////////////////////////////////////////////////////

#include <math.h>

void realft(float data[], unsigned long n, int isign)
{
        void four1(float data[], unsigned long nn, int isign);
        unsigned long i,i1,i2,i3,i4,np3;
        float c1=0.5,c2,h1r,h1i,h2r,h2i;
        double wr,wi,wpr,wpi,wtemp,theta;

        theta=3.141592653589793/(double) (n>>1);
        if (isign == 1) {
                c2 = -0.5;
                four1(data,n>>1,1);
        } else {
                c2=0.5;
                theta = -theta;
        }
        wtemp=sin(0.5*theta);
        wpr = -2.0*wtemp*wtemp;
        wpi=sin(theta);
        wr=1.0+wpr;
        wi=wpi;
        np3=n+3;
        for (i=2;i<=(n>>2);i++) {
                i4=1+(i3=np3-(i2=1+(i1=i+i-1)));
                h1r=c1*(data[i1]+data[i3]);
                h1i=c1*(data[i2]-data[i4]);
                h2r = -c2*(data[i2]+data[i4]);
                h2i=c2*(data[i1]-data[i3]);
                data[i1]=h1r+wr*h2r-wi*h2i;
                data[i2]=h1i+wr*h2i+wi*h2r;
                data[i3]=h1r-wr*h2r+wi*h2i;
                data[i4] = -h1i+wr*h2i+wi*h2r;
                wr=(wtemp=wr)*wpr-wi*wpi+wr;
                wi=wi*wpr+wtemp*wpi+wi;
        }
        if (isign == 1) {
                data[1] = (h1r=data[1])+data[2];
                data[2] = h1r-data[2];
        } else {
                data[1]=c1*((h1r=data[1])+data[2]);
                data[2]=c1*(h1r-data[2]);
                four1(data,n>>1,-1);
        }
}
/* (C) Copr. 1986-92 Numerical Recipes Software */
```

File: nrutil.c

Wheel Hunting Data Acquisition Software

```
////////////////////////////////////////////////////////////
// File: nrutil.c                                         //
//                                                        //
// Wheel Hunting Data Acquisition Software                //
                          //
//                                                        //
// Copyright 1995: IEM Corporation                        //
//                                                        //
// Proprietary Licensed Information.                      //
// Not to be duplicated, used or disclosed                //
// except under terms of license.                         //
//                                                        //
// Revision History:                                      //
////////////////////////////////////////////////////////////

#if defined(__STDC__) || defined(ANSI) || defined(NRANSI) /* ANSI */

#include <stdio.h>
#include <stddef.h>
#include <stdlib.h>
#define NR_END 1
#define FREE_ARG char*

void nrerror(char error_text[])
/* Numerical Recipes standard error handler */
{
        fprintf(stderr,"Numerical Recipes run-time error...\n");
        fprintf(stderr,"%s\n",error_text);
        fprintf(stderr,"...now exiting to system...\n");
        exit(1);
}

float *vector(long nl, long nh)
/* allocate a float vector with subscript range v[nl..nh] */
{
        float *v;

        v=(float *)malloc((size_t) ((nh-nl+1+NR_END)*sizeof(float)));
        if (!v) nrerror("allocation failure in vector()");
        return v-nl+NR_END;
}

int *ivector(long nl, long nh)
/* allocate an int vector with subscript range v[nl..nh] */
{
        int *v;

        v=(int *)malloc((size_t) ((nh-nl+1+NR_END)*sizeof(int)));
        if (!v) nrerror("allocation failure in ivector()");
        return v-nl+NR_END;
}

unsigned char *cvector(long nl, long nh)
/* allocate an unsigned char vector with subscript range v[nl..nh] */
{
        unsigned char *v;

        v=(unsigned char *)malloc((size_t) ((nh-nl+1+NR_END)*sizeof(unsigned char
)));
        if (!v) nrerror("allocation failure in cvector()");
        return v-nl+NR_END;
}

unsigned long *lvector(long nl, long nh)
/* allocate an unsigned long vector with subscript range v[nl..nh] */
{
```



```
        unsigned long *v;

        v=(unsigned long *)malloc((size_t) ((nh-nl+1+NR_END)*sizeof(long)));
        if (!v) nrerror("allocation failure in lvector()");
        return v-nl+NR_END;
}

double *dvector(long nl, long nh)
/* allocate a double vector with subscript range v[nl..nh] */
{
        double *v;

        v=(double *)malloc((size_t) ((nh-nl+1+NR_END)*sizeof(double)));
        if (!v) nrerror("allocation failure in dvector()");
        return v-nl+NR_END;
}

float **matrix(long nrl, long nrh, long ncl, long nch)
/* allocate a float matrix with subscript range m[nrl..nrh][ncl..nch] */
{
        long i, nrow=nrh-nrl+1,ncol=nch-ncl+1;
        float **m;

        /* allocate pointers to rows */
        m=(float **) malloc((size_t)((nrow+NR_END)*sizeof(float*)));
        if (!m) nrerror("allocation failure 1 in matrix()");
        m += NR_END;
        m -= nrl;

        /* allocate rows and set pointers to them */
        m[nrl]=(float *) malloc((size_t)((nrow*ncol+NR_END)*sizeof(float)));
        if (!m[nrl]) nrerror("allocation failure 2 in matrix()");
        m[nrl] += NR_END;
        m[nrl] -= ncl;

        for(i=nrl+1;i<=nrh;i++) m[i]=m[i-1]+ncol;

        /* return pointer to array of pointers to rows */
        return m;
}

double **dmatrix(long nrl, long nrh, long ncl, long nch)
/* allocate a double matrix with subscript range m[nrl..nrh][ncl..nch] */
{
        long i, nrow=nrh-nrl+1,ncol=nch-ncl+1;
        double **m;

        /* allocate pointers to rows */
        m=(double **) malloc((size_t)((nrow+NR_END)*sizeof(double*)));
        if (!m) nrerror("allocation failure 1 in matrix()");
        m += NR_END;
        m -= nrl;

        /* allocate rows and set pointers to them */
        m[nrl]=(double *) malloc((size_t)((nrow*ncol+NR_END)*sizeof(double)));
        if (!m[nrl]) nrerror("allocation failure 2 in matrix()");
        m[nrl] += NR_END;
        m[nrl] -= ncl;

        for(i=nrl+1;i<=nrh;i++) m[i]=m[i-1]+ncol;

        /* return pointer to array of pointers to rows */
        return m;
}

int **imatrix(long nrl, long nrh, long ncl, long nch)
```

```
/* allocate a int matrix with subscript range m[nrl..nrh][ncl..nch] */
{
        long i, nrow=nrh-nrl+1,ncol=nch-ncl+1;
        int **m;

        /* allocate pointers to rows */
        m=(int **) malloc((size_t)((nrow+NR_END)*sizeof(int*)));
        if (!m) nrerror("allocation failure 1 in matrix()");
        m += NR_END;
        m -= nrl;


        /* allocate rows and set pointers to them */
        m[nrl]=(int *) malloc((size_t)((nrow*ncol+NR_END)*sizeof(int)));
        if (!m[nrl]) nrerror("allocation failure 2 in matrix()");
        m[nrl] += NR_END;
        m[nrl] -= ncl;

        for(i=nrl+1;i<=nrh;i++) m[i]=m[i-1]+ncol;

        /* return pointer to array of pointers to rows */
        return m;
}

float **submatrix(float **a, long oldrl, long oldrh, long oldcl, long oldch,
        long newrl, long newcl)
/* point a submatrix [newrl..][newcl..] to a[oldrl..oldrh][oldcl..oldch] */
{
        long i,j,nrow=oldrh-oldrl+1,ncol=oldcl-newcl;
        float **m;

        /* allocate array of pointers to rows */
        m=(float **) malloc((size_t) ((nrow+NR_END)*sizeof(float*)));
        if (!m) nrerror("allocation failure in submatrix()");
        m += NR_END;
        m -= newrl;

        /* set pointers to rows */
        for(i=oldrl,j=newrl;i<=oldrh;i++,j++) m[j]=a[i]+ncol;

        /* return pointer to array of pointers to rows */
        return m;
}

float **convert_matrix(float *a, long nrl, long nrh, long ncl, long nch)
/* allocate a float matrix m[nrl..nrh][ncl..nch] that points to the matrix
declared in the standard C manner as a[nrow][ncol], where nrow=nrh-nrl+1
and ncol=nch-ncl+1. The routine should be called with the address
&a[0][0] as the first argument. */
{
        long i,j,nrow=nrh-nrl+1,ncol=nch-ncl+1;
        float **m;

        /* allocate pointers to rows */
        m=(float **) malloc((size_t) ((nrow+NR_END)*sizeof(float*)));
        if (!m) nrerror("allocation failure in convert_matrix()");
        m += NR_END;
        m -= nrl;

        /* set pointers to rows */
        m[nrl]=a-ncl;
        for(i=1,j=nrl+1;i<nrow;i++,j++) m[j]=m[j-1]+ncol;
        /* return pointer to array of pointers to rows */
        return m;
}
```



```
float ***f3tensor(long nrl, long nrh, long ncl, long nch, long ndl, long ndh)
/* allocate a float 3tensor with range t[nrl..nrh][ncl..nch][ndl..ndh] */
{
        long i,j,nrow=nrh-nrl+1,ncol=nch-ncl+1,ndep=ndh-ndl+1;
        float ***t;

        /* allocate pointers to pointers to rows */
        t=(float ***) malloc((size_t)((nrow+NR_END)*sizeof(float**)));
        if (!t) nrerror("allocation failure 1 in f3tensor()");
        t += NR_END;
        t -= nrl;

        /* allocate pointers to rows and set pointers to them */
        t[nrl]=(float **) malloc((size_t)((nrow*ncol+NR_END)*sizeof(float*)));
        if (!t[nrl]) nrerror("allocation failure 2 in f3tensor()");
        t[nrl] += NR_END;
        t[nrl] -= ncl;

        /* allocate rows and set pointers to them */
        t[nrl][ncl]=(float *) malloc((size_t)((nrow*ncol*ndep+NR_END)*sizeof(floa
t)));
        if (!t[nrl][ncl]) nrerror("allocation failure 3 in f3tensor()");
        t[nrl][ncl] += NR_END;
        t[nrl][ncl] -= ndl;

        for(j=ncl+1;j<=nch;j++) t[nrl][j]=t[nrl][j-1]+ndep;
        for(i=nrl+1;i<=nrh;i++) {
                t[i]=t[i-1]+ncol;
                t[i][ncl]=t[i-1][ncl]+ncol*ndep;
                for(j=ncl+1;j<=nch;j++) t[i][j]=t[i][j-1]+ndep;
        }

        /* return pointer to array of pointers to rows */
        return t;
}

void free_vector(float *v, long nl, long nh)
/* free a float vector allocated with vector() */
{
        free((FREE_ARG) (v+nl-NR_END));
}

void free_ivector(int *v, long nl, long nh)
/* free an int vector allocated with ivector() */
{
        free((FREE_ARG) (v+nl-NR_END));
}

void free_cvector(unsigned char *v, long nl, long nh)
/* free an unsigned char vector allocated with cvector() */
{
        free((FREE_ARG) (v+nl-NR_END));
}

void free_lvector(unsigned long *v, long nl, long nh)
/* free an unsigned long vector allocated with lvector() */
{
        free((FREE_ARG) (v+nl-NR_END));
}

void free_dvector(double *v, long nl, long nh)
/* free a double vector allocated with dvector() */
{
        free((FREE_ARG) (v+nl-NR_END));
}
```

```
void free_matrix(float **m, long nrl, long nrh, long ncl, long nch)
/* free a float matrix allocated by matrix() */
{
        free((FREE_ARG) (m[nrl]+ncl-NR_END));
        free((FREE_ARG) (m+nrl-NR_END));
}

void free_dmatrix(double **m, long nrl, long nrh, long ncl, long nch)
/* free a double matrix allocated by dmatrix() */
{
        free((FREE_ARG) (m[nrl]+ncl-NR_END));
        free((FREE_ARG) (m+nrl-NR_END));
}

void free_imatrix(int **m, long nrl, long nrh, long ncl, long nch)
/* free an int matrix allocated by imatrix() */
{
        free((FREE_ARG) (m[nrl]+ncl-NR_END));
        free((FREE_ARG) (m+nrl-NR_END));
}

void free_submatrix(float **b, long nrl, long nrh, long ncl, long nch)
/* free a submatrix allocated by submatrix() */
{
        free((FREE_ARG) (b+nrl-NR_END));
}

void free_convert_matrix(float **b, long nrl, long nrh, long ncl, long nch)
/* free a matrix allocated by convert_matrix() */
{
        free((FREE_ARG) (b+nrl-NR_END));
}

void free_f3tensor(float ***t, long nrl, long nrh, long ncl, long nch,
        long ndl, long ndh)
/* free a float f3tensor allocated by f3tensor() */
{
        free((FREE_ARG) (t[nrl][ncl]+ndl-NR_END));
        free((FREE_ARG) (t[nrl]+ncl-NR_END));
        free((FREE_ARG) (t+nrl-NR_END));
}

#else /* ANSI */
/* traditional - K&R */

#include <stdio.h>
#define NR_END 1
#define FREE_ARG char*

void nrerror(error_text)
char error_text[];
/* Numerical Recipes standard error handler */
{
        void exit();

        fprintf(stderr,"Numerical Recipes run-time error...\n");
        fprintf(stderr,"%s\n",error_text);
        fprintf(stderr,"...now exiting to system...\n");
        exit(1);
}

float *vector(nl,nh)
long nh,nl;
/* allocate a float vector with subscript range v[nl..nh] */
{
        float *v;
```



```
        v=(float *)malloc((unsigned int) ((nh-nl+1+NR_END)*sizeof(float)));
        if (!v) nrerror("allocation failure in vector()");
        return v-nl+NR_END;
}

int *ivector(nl,nh)
long nh,nl;
/* allocate an int vector with subscript range v[nl..nh] */
{
        int *v;

        v=(int *)malloc((unsigned int) ((nh-nl+1+NR_END)*sizeof(int)));
        if (!v) nrerror("allocation failure in ivector()");
        return v-nl+NR_END;
}

unsigned char *cvector(nl,nh)
long nh,nl;
/* allocate an unsigned char vector with subscript range v[nl..nh] */
{
        unsigned char *v;

        v=(unsigned char *)malloc((unsigned int) ((nh-nl+1+NR_END)*sizeof(unsigne
d char)));
        if (!v) nrerror("allocation failure in cvector()");
        return v-nl+NR_END;
}

unsigned long *lvector(nl,nh)
long nh,nl;
/* allocate an unsigned long vector with subscript range v[nl..nh] */
{
        unsigned long *v;

        v=(unsigned long *)malloc((unsigned int) ((nh-nl+1+NR_END)*sizeof(long)))
;
        if (!v) nrerror("allocation failure in lvector()");
        return v-nl+NR_END;
}

double *dvector(nl,nh)
long nh,nl;
/* allocate a double vector with subscript range v[nl..nh] */
{
        double *v;

        v=(double *)malloc((unsigned int) ((nh-nl+1+NR_END)*sizeof(double)));
        if (!v) nrerror("allocation failure in dvector()");
        return v-nl+NR_END;
}

float **matrix(nrl,nrh,ncl,nch)
long nch,ncl,nrh,nrl;
/* allocate a float matrix with subscript range m[nrl..nrh][ncl..nch] */
{
        long i, nrow=nrh-nrl+1,ncol=nch-ncl+1;
        float **m;

        /* allocate pointers to rows */
        m=(float **) malloc((unsigned int)((nrow+NR_END)*sizeof(float*)));
        if (!m) nrerror("allocation failure 1 in matrix()");
        m += NR_END;
        m -= nrl;

        /* allocate rows and set pointers to them */
```

```
        m[nrl]=(float *) malloc((unsigned int)((nrow*ncol+NR_END)*sizeof(float)))
;
        if (!m[nrl]) nrerror("allocation failure 2 in matrix()");
        m[nrl] += NR_END;
        m[nrl] -= ncl;

        for(i=nrl+1;i<=nrh;i++) m[i]=m[i-1]+ncol;

        /* return pointer to array of pointers to rows */
        return m;
}

double **dmatrix(nrl,nrh,ncl,nch)
long nch,ncl,nrh,nrl;
/* allocate a double matrix with subscript range m[nrl..nrh][ncl..nch] */
{
        long i, nrow=nrh-nrl+1,ncol=nch-ncl+1;
        double **m;

        /* allocate pointers to rows */
        m=(double **) malloc((unsigned int)((nrow+NR_END)*sizeof(double*)));
        if (!m) nrerror("allocation failure 1 in matrix()");
        m += NR_END;
        m -= nrl;

        /* allocate rows and set pointers to them */
        m[nrl]=(double *) malloc((unsigned int)((nrow*ncol+NR_END)*sizeof(double)
));
        if (!m[nrl]) nrerror("allocation failure 2 in matrix()");
        m[nrl] += NR_END;
        m[nrl] -= ncl;

        for(i=nrl+1;i<=nrh;i++) m[i]=m[i-1]+ncol;

        /* return pointer to array of pointers to rows */
        return m;
}

int **imatrix(nrl,nrh,ncl,nch)
long nch,ncl,nrh,nrl;
/* allocate a int matrix with subscript range m[nrl..nrh][ncl..nch] */
{
        long i, nrow=nrh-nrl+1,ncol=nch-ncl+1;
        int **m;

        /* allocate pointers to rows */
        m=(int **) malloc((unsigned int)((nrow+NR_END)*sizeof(int*)));
        if (!m) nrerror("allocation failure 1 in matrix()");
        m += NR_END;
        m -= nrl;


        /* allocate rows and set pointers to them */
        m[nrl]=(int *) malloc((unsigned int)((nrow*ncol+NR_END)*sizeof(int)));
        if (!m[nrl]) nrerror("allocation failure 2 in matrix()");
        m[nrl] += NR_END;
        m[nrl] -= ncl;

        for(i=nrl+1;i<=nrh;i++) m[i]=m[i-1]+ncol;

        /* return pointer to array of pointers to rows */
        return m;
}

float **submatrix(a,oldrl,oldrh,oldcl,oldch,newrl,newcl)
float **a;
```



```
long newcl,newrl,oldch,oldcl,oldrh,oldrl;
/* point a submatrix [newrl..][newcl..] to a[oldrl..oldrh][oldcl..oldch] */
{
        long i,j,nrow=oldrh-oldrl+1,ncol=oldcl-newcl;
        float **m;

        /* allocate array of pointers to rows */
        m=(float **) malloc((unsigned int) ((nrow+NR_END)*sizeof(float*)));
        if (!m) nrerror("allocation failure in submatrix()");
        m += NR_END;
        m -= newrl;

        /* set pointers to rows */
        for(i=oldrl,j=newrl;i<=oldrh;i++,j++) m[j]=a[i]+ncol;

        /* return pointer to array of pointers to rows */
        return m;
}

float **convert_matrix(a,nrl,nrh,ncl,nch)
float *a;
long nch,ncl,nrh,nrl;
/* allocate a float matrix m[nrl..nrh][ncl..nch] that points to the matrix
declared in the standard C manner as a[nrow][ncol], where nrow=nrh-nrl+1
and ncol=nch-ncl+1. The routine should be called with the address
&a[0][0] as the first argument. */
{
        long i,j,nrow=nrh-nrl+1,ncol=nch-ncl+1;
        float **m;

        /* allocate pointers to rows */
        m=(float **) malloc((unsigned int) ((nrow+NR_END)*sizeof(float*)));
        if (!m) nrerror("allocation failure in convert_matrix()");
        m += NR_END;
        m -= nrl;

        /* set pointers to rows */
        m[nrl]=a-ncl;
        for(i=1,j=nrl+1;i<nrow;i++,j++) m[j]=m[j-1]+ncol;
        /* return pointer to array of pointers to rows */
        return m;
}

float ***f3tensor(nrl,nrh,ncl,nch,ndl,ndh)
long nch,ncl,ndh,ndl,nrh,nrl;
/* allocate a float 3tensor with range t[nrl..nrh][ncl..nch][ndl..ndh] */
{
        long i,j,nrow=nrh-nrl+1,ncol=nch-ncl+1,ndep=ndh-ndl+1;
        float ***t;

        /* allocate pointers to pointers to rows */
        t=(float ***) malloc((unsigned int)((nrow+NR_END)*sizeof(float**)));
        if (!t) nrerror("allocation failure 1 in f3tensor()");
        t += NR_END;
        t -= nrl;

        /* allocate pointers to rows and set pointers to them */
        t[nrl]=(float **) malloc((unsigned int)((nrow*ncol+NR_END)*sizeof(float*)
));
        if (!t[nrl]) nrerror("allocation failure 2 in f3tensor()");
        t[nrl] += NR_END;
        t[nrl] -= ncl;

        /* allocate rows and set pointers to them */
        t[nrl][ncl]=(float *) malloc((unsigned int)((nrow*ncol*ndep+NR_END)*sizeo
f(float)));
```

```
        if (!t[nrl][ncl]) nrerror("allocation failure 3 in f3tensor()");
        t[nrl][ncl] += NR_END;
        t[nrl][ncl] -= ndl;

        for(j=ncl+1;j<=nch;j++) t[nrl][j]=t[nrl][j-1]+ndep;
        for(i=nrl+1;i<=nrh;i++) {
                t[i]=t[i-1]+ncol;
                t[i][ncl]=t[i-1][ncl]+ncol*ndep;
                for(j=ncl+1;j<=nch;j++) t[i][j]=t[i][j-1]+ndep;
        }

        /* return pointer to array of pointers to rows */
        return t;
}

void free_vector(v,nl,nh)
float *v;
long nh,nl;
/* free a float vector allocated with vector() */
{
        free((FREE_ARG) (v+nl-NR_END));
}

void free_ivector(v,nl,nh)
int *v;
long nh,nl;
/* free an int vector allocated with ivector() */
{
        free((FREE_ARG) (v+nl-NR_END));
}

void free_cvector(v,nl,nh)
long nh,nl;
unsigned char *v;
/* free an unsigned char vector allocated with cvector() */
{
        free((FREE_ARG) (v+nl-NR_END));
}

void free_lvector(v,nl,nh)
long nh,nl;
unsigned long *v;
/* free an unsigned long vector allocated with lvector() */
{
        free((FREE_ARG) (v+nl-NR_END));
}

void free_dvector(v,nl,nh)
double *v;
long nh,nl;
/* free a double vector allocated with dvector() */
{
        free((FREE_ARG) (v+nl-NR_END));
}

void free_matrix(m,nrl,nrh,ncl,nch)
float **m;
long nch,ncl,nrh,nrl;
/* free a float matrix allocated by matrix() */
{
        free((FREE_ARG) (m[nrl]+ncl-NR_END));
        free((FREE_ARG) (m+nrl-NR_END));
}

void free_dmatrix(m,nrl,nrh,ncl,nch)
double **m;
```



```
long nch,ncl,nrh,nrl;
/* free a double matrix allocated by dmatrix() */
{
        free((FREE_ARG) (m[nrl]+ncl-NR_END));
        free((FREE_ARG) (m+nrl-NR_END));
}

void free_imatrix(m,nrl,nrh,ncl,nch)
int **m;
long nch,ncl,nrh,nrl;
/* free an int matrix allocated by imatrix() */
{
        free((FREE_ARG) (m[nrl]+ncl-NR_END));
        free((FREE_ARG) (m+nrl-NR_END));
}

void free_submatrix(b,nrl,nrh,ncl,nch)
float **b;
long nch,ncl,nrh,nrl;
/* free a submatrix allocated by submatrix() */
{
        free((FREE_ARG) (b+nrl-NR_END));
}

void free_convert_matrix(b,nrl,nrh,ncl,nch)
float **b;
long nch,ncl,nrh,nrl;
/* free a matrix allocated by convert_matrix() */
{
        free((FREE_ARG) (b+nrl-NR_END));
}

void free_f3tensor(t,nrl,nrh,ncl,nch,ndl,ndh)
float ***t;
long nch,ncl,ndh,ndl,nrh,nrl;
/* free a float f3tensor allocated by f3tensor() */
{
        free((FREE_ARG) (t[nrl][ncl]+ndl-NR_END));
        free((FREE_ARG) (t[nrl]+ncl-NR_END));
        free((FREE_ARG) (t+nrl-NR_END));
}

#endif /* ANSI */
```

File: four1.c

Wheel Hunting Data Acquisition Software

```
////////////////////////////////////////////////////////////////
// File: four1.c                                              //
//                                                            //
// Wheel Hunting Data Acquisition Software                    //
                          //
//                                                            //
// Copyright 1995: IEM Corporation                            //
//                                                            //
// Proprietary Licensed Information.                          //
// Not to be duplicated, used or disclosed                    //
// except under terms of license.                             //
//                                                            //
// Revision History:                                          //
////////////////////////////////////////////////////////////////

#include <math.h>
#define SWAP(a,b) tempr=(a);(a)=(b);(b)=tempr

void four1(float data[], unsigned long nn, int isign)
{
        unsigned long n,mmax,m,j,istep,i;
        double wtemp,wr,wpr,wpi,wi,theta;
        float tempr,tempi;

        n=nn << 1;
        j=1;
        for (i=1;i<n;i+=2) {
                if (j > i) {
                        SWAP(data[j],data[i]);
                        SWAP(data[j+1],data[i+1]);
                }
                m=n >> 1;
                while (m >= 2 && j > m) {
                        j -= m;
                        m >>= 1;
                }
                j += m;
        }
        mmax=2;
        while (n > mmax) {
                istep=mmax << 1;
                theta=isign*(6.28318530717959/mmax);
                wtemp=sin(0.5*theta);
                wpr = -2.0*wtemp*wtemp;
                wpi=sin(theta);
                wr=1.0;
                wi=0.0;
                for (m=1;m<mmax;m+=2) {
                        for (i=m;i<=n;i+=istep) {
                                j=i+mmax;
                                tempr=wr*data[j]-wi*data[j+1];
                                tempi=wr*data[j+1]+wi*data[j];
                                data[j]=data[i]-tempr;
                                data[j+1]=data[i+1]-tempi;
                                data[i] += tempr;
                                data[i+1] += tempi;
                        }
                        wr=(wtemp=wr)*wpr-wi*wpi+wr;
                        wi=wi*wpr+wtemp*wpi+wi;
                }
                mmax=istep;
        }
}
#undef SWAP
/* (C) Copr. 1986-92 Numerical Recipes Software :!S$-)=. */
```

File: spectrum.c

Wheel Hunting Data Acquisition Software

```
////////////////////////////////////////////////////////////
// File: spectrum.c                                       //
//                                                        //
// Wheel Hunting Data Acquisition Software                //
                        //
//                                                        //
// Copyright 1995: IEM Corporation                        //
//                                                        //
// Proprietary Licensed Information.                      //
// Not to be duplicated, used or disclosed                //
// except under terms of license.                         //
//                                                        //
// Revision History:                                      //
////////////////////////////////////////////////////////////

#include "hunthead.h"   // Compiler includes
#include "huntstru.h"   // Constant and structure declarations
#undef   BLACK              /*  avoid redef warnings      */
#undef   RED
#undef   WHITE
#include <graphic.h>          /* Include all needed files */

extern  long    bkgnd;
extern  short   frgnd;
extern  int             low_range;

extern  float   huge x_spectrum[MAX_I_POINTSHALF], y_spectrum[MAX_I_POINTSHALF];
void    show_spectrum(int n, float sampf, char *file);

void show_spectrum(int n, float sampling_freq, char *filename)
{
float   max_freq, freq_incr, del_freq;
int             i, bincount;
char    tempfile[20], temp_name[20];
FILE    *stream;

        // Now display the spectrum

    bgnplot(1, 'g', "spectrum.tkf");
    startplot(BLACK);    /* Initializes plot and sets the background color */

    metricunits(0);      /* Ensure scaling in inch units */
    rotate(0);           /* do not rotate the screen */
    page(9.f, 6.88f);   /* Sets the page size */
    tmargin(0.0f);          /* Set the margins */
    xname("Frequency (Hz)");
    yname("Amplitude");
    heading("DATA SPECTRUM");
    area2d(7.f, 3.20f);             /* Sets the area of the plot */
        color(WHITE);               /* Axes names and heading will be black */
        grid(4);
    upright(1);
    max_freq = sampling_freq/2.0f;
        if(low_range == TRUE)max_freq = 50.0f;
        if(max_freq < 500.0f)
                freq_incr = max_freq/10.0f;
        else
                freq_incr = max_freq/5.0f;
        del_freq = sampling_freq / (float)n;
        graf("%4.1f",0,freq_incr,max_freq,"%-1.0f",0,10,60);
        bincount = n/2;
        for(i=0; i<bincount; i++)x_spectrum[i] = x_spectrum[i] * del_freq;
        histogram(x_spectrum, y_spectrum, bincount, 0, NULL);
//      bar(NUMOFBINS,barx,bary,MAXBARS,MAXBARS,0,NULL,NULL);
        strcpy(tempfile, "Filename: ");
        strcat(tempfile, filename);
```



```
    prtfnt(3.5f,0.5f,tempfile,.15f,0);
        getch();
    endplot();
    stopplot();


        // Now save the spectrum file in "s_filename.dat"
        // Notice that you cannot use a filename longer than 6 characters

        strcpy(temp_name,"s_");
        strcat(temp_name,filename);
        if((stream = fopen(temp_name,"w"))==NULL){
                        printf("Cannot open SPECTRUM output DATA file %s", temp_n
ame);
                        getfunc();
        } else {
                for(i=0; i<bincount; i++)
                                fprintf(stream, "%4.2f , %4.2f\n", x_spectrum[i],
 y_spectrum[i]);
                fclose(stream);
        }

    _setvideomode(_TEXTC80);
    _settextcolor( frgnd );
    _setbkcolor( bkgnd );
}
```

File: gaussj.c

Wheel Hunting Data Acquisition Software

```
////////////////////////////////////////////////////////////////
// File: gaussj.c                                             //
//                                                            //
// Wheel Hunting Data Acquisition Software                    //
                        //
//                                                            //
// Copyright 1995: IEM Corporation                            //
//                                                            //
// Proprietary Licensed Information.                          //
// Not to be duplicated, used or disclosed                    //
// except under terms of license.                             //
//                                                            //
// Revision History:                                          //
////////////////////////////////////////////////////////////////

#include <math.h>
#define NRANSI
#include "nrutil.h"
#define SWAP(a,b) {temp=(a);(a)=(b);(b)=temp;}

void gaussj(float **a, int n, float **b, int m)
{
        int *indxc,*indxr,*ipiv;
        int i,icol,irow,j,k,l,ll;
        float big,dum,pivinv,temp;

        indxc=ivector(1,n);
        indxr=ivector(1,n);
        ipiv=ivector(1,n);
        for (j=1;j<=n;j++) ipiv[j]=0;
        for (i=1;i<=n;i++) {
                big=0.0;
                for (j=1;j<=n;j++)
                        if (ipiv[j] != 1)
                                for (k=1;k<=n;k++) {
                                        if (ipiv[k] == 0) {
                                                if (fabs(a[j][k]) >= big) {
                                                        big=fabs(a[j][k]);
                                                        irow=j;
                                                        icol=k;
                                                }
                                        } else if (ipiv[k] > 1) nrerror("gaussj:
Singular Matrix-1");
                                }
                ++(ipiv[icol]);
                if (irow != icol) {
                        for (l=1;l<=n;l++) SWAP(a[irow][l],a[icol][l])
                        for (l=1;l<=m;l++) SWAP(b[irow][l],b[icol][l])
                }
                indxr[i]=irow;
                indxc[i]=icol;
                if (a[icol][icol] == 0.0) nrerror("gaussj: Singular Matrix-2");
                pivinv=1.0/a[icol][icol];
                a[icol][icol]=1.0;
                for (l=1;l<=n;l++) a[icol][l] *= pivinv;
                for (l=1;l<=m;l++) b[icol][l] *= pivinv;
                for (ll=1;ll<=n;ll++)
                        if (ll != icol) {
                                dum=a[ll][icol];
                                a[ll][icol]=0.0;
                                for (l=1;l<=n;l++) a[ll][l] -= a[icol][l]*dum;
                                for (l=1;l<=m;l++) b[ll][l] -= b[icol][l]*dum;
                        }
        }
        for (l=n;l>=1;l--) {
                if (indxr[l] != indxc[l])
```



```
                        for (k=1;k<=n;k++)
                                SWAP(a[k][indxr[l]],a[k][indxc[l]]);
        }
        free_ivector(ipiv,1,n);
        free_ivector(indxr,1,n);
        free_ivector(indxc,1,n);
}
#undef SWAP
#undef NRANSI
/* (C) Copr. 1986-92 Numerical Recipes Software :!S$-)=. */
```

File: gcf.c

Wheel Hunting Data Acquisition Software

```
////////////////////////////////////////////////////////////////
// File: gcf.c                                                //
//                                                            //
// Wheel Hunting Data Acquisition Software                    //
                          //
//                                                            //
// Copyright 1995: IEM Corporation                            //
//                                                            //
// Proprietary Licensed Information.                          //
// Not to be duplicated, used or disclosed                    //
// except under terms of license.                             //
//                                                            //
// Revision History:                                          //
////////////////////////////////////////////////////////////////

#include <math.h>
#define ITMAX 100
#define EPS 3.0e-7
#define FPMIN 1.0e-30

void gcf(float *gammcf, float a, float x, float *gln)
{
        float gammln(float xx);
        void nrerror(char error_text[]);
        int i;
        float an,b,c,d,del,h;

        *gln=gammln(a);
        b=x+1.0-a;
        c=1.0/FPMIN;
        d=1.0/b;
        h=d;
        for (i=1;i<=ITMAX;i++) {
                an = -i*(i-a);
                b += 2.0;
                d=an*d+b;
                if (fabs(d) < FPMIN) d=FPMIN;
                c=b+an/c;
                if (fabs(c) < FPMIN) c=FPMIN;
                d=1.0/d;
                del=d*c;
                h *= del;
                if (fabs(del-1.0) < EPS) break;
        }
        if (i > ITMAX) nrerror("a too large, ITMAX too small in gcf");
        *gammcf=exp(-x+a*log(x)-(*gln))*h;
}
#undef ITMAX
#undef EPS
#undef FPMIN
/* (C) Copr. 1986-92 Numerical Recipes Software :!S$-)=. */
```

File: fit.c

Wheel Hunting Data Acquisition Software

```
////////////////////////////////////////////////////////////
// File: fit.c                                            //
//                                                        //
// Wheel Hunting Data Acquisition Software                //
                          //
//                                                        //
// Copyright 1995: IEM Corporation                        //
//                                                        //
// Proprietary Licensed Information.                      //
// Not to be duplicated, used or disclosed                //
// except under terms of license.                         //
//                                                        //
// Revision History:                                      //
////////////////////////////////////////////////////////////

#include <math.h>
#define NRANSI
#include "nrutil.h"
extern  float gammq(float a, float x);

void fit(float x[], float y[], int ndata, float sig[], int mwt, float *a,
        float *b, float *siga, float *sigb, float *chi2, float *q)
{
        int i;
        float wt,t,sxoss,sx=0.0,sy=0.0,st2=0.0,ss,sigdat;

        *b=0.0;
        if (mwt) {
                ss=0.0;
                for (i=1;i<=ndata;i++) {
                        wt=1.0/SQR(sig[i]);
                        ss += wt;
                        sx += x[i]*wt;
                        sy += y[i]*wt;
                }
        } else {
                for (i=1;i<=ndata;i++) {
                        sx += x[i];
                        sy += y[i];
                }
                ss=ndata;
        }
        sxoss=sx/ss;
        if (mwt) {
                for (i=1;i<=ndata;i++) {
                        t=(x[i]-sxoss)/sig[i];
                        st2 += t*t;
                        *b += t*y[i]/sig[i];
                }
        } else {
                for (i=1;i<=ndata;i++) {
                        t=x[i]-sxoss;
                        st2 += t*t;
                        *b += t*y[i];
                }
        }
        *b /= st2;
        *a=(sy-sx*(*b))/ss;
        *siga=sqrt((1.0+sx*sx/(ss*st2))/ss);
        *sigb=sqrt(1.0/st2);
        *chi2=0.0;
        if (mwt == 0) {
                for (i=1;i<=ndata;i++)
                        *chi2 += SQR(y[i]-(*a)-(*b)*x[i]);
                *q=1.0;
                sigdat=sqrt((*chi2)/(ndata-2));
```



```
                *siga *= sigdat;
                *sigb *= sigdat;
        } else {
                for (i=1;i<=ndata;i++)
                        *chi2 += SQR((y[i]-(*a)-(*b)*x[i])/sig[i]);
                *q=gammq(0.5*(ndata-2),0.5*(*chi2));
        }
}
#undef NRANSI
/* (C) Copr. 1986-92 Numerical Recipes Software :!S$-)=. */
```

File: gser.c

Wheel Hunting Data Acquisition Software

```
////////////////////////////////////////////////////////////
// File: gser.c                                           //
//                                                        //
// Wheel Hunting Data Acquisition Software                //
                          //
//                                                        //
// Copyright 1995: IEM Corporation                        //
//                                                        //
// Proprietary Licensed Information.                      //
// Not to be duplicated, used or disclosed                //
// except under terms of license.                         //
//                                                        //
// Revision History:                                      //
////////////////////////////////////////////////////////////

#include <math.h>
#define ITMAX 100
#define EPS 3.0e-7

void gser(float *gamser, float a, float x, float *gln)
{
        float gammln(float xx);
        void nrerror(char error_text[]);
        int n;
        float sum,del,ap;

        *gln=gammln(a);
        if (x <= 0.0) {
                if (x < 0.0) nrerror("x less than 0 in routine gser");
                *gamser=0.0;
                return;
        } else {
                ap=a;
                del=sum=1.0/a;
                for (n=1;n<=ITMAX;n++) {
                        ++ap;
                        del *= x/ap;
                        sum += del;
                        if (fabs(del) < fabs(sum)*EPS) {
                                *gamser=sum*exp(-x+a*log(x)-(*gln));
                                return;
                        }
                }
                nrerror("a too large, ITMAX too small in routine gser");
                return;
        }
}
#undef ITMAX
#undef EPS
/* (C) Copr. 1986-92 Numerical Recipes Software :!S$-)=. */
```

File: gammq.c

Wheel Hunting Data Acquisition Software

```
////////////////////////////////////////////////////////////////
// File: gammq.c                                              //
//                                                            //
// Wheel Hunting Data Acquisition Software                    //
                          //
//                                                            //
// Copyright 1995: IEM Corporation                            //
//                                                            //
// Proprietary Licensed Information.                          //
// Not to be duplicated, used or disclosed                    //
// except under terms of license.                             //
//                                                            //
// Revision History:                                          //
////////////////////////////////////////////////////////////////

float gammq(float a, float x)
{
        void gcf(float *gammcf, float a, float x, float *gln);
        void gser(float *gamser, float a, float x, float *gln);
        void nrerror(char error_text[]);
        float gamser,gammcf,gln;

        if (x < 0.0 || a <= 0.0) nrerror("Invalid arguments in routine gammq");
        if (x < (a+1.0)) {
                gser(&gamser,a,x,&gln);
                return 1.0-gamser;
        } else {
                gcf(&gammcf,a,x,&gln);
                return gammcf;
        }
}
/* (C) Copr. 1986-92 Numerical Recipes Software :!S$-)=. */
```

File: xlfit.c

Wheel Hunting Data Acquisition Software

```
////////////////////////////////////////////////////////////////
// File: xlfit.c                                              //
//                                                            //
// Wheel Hunting Data Acquisition Software                    //
                          //
//                                                            //
// Copyright 1995: IEM Corporation                            //
//                                                            //
// Proprietary Licensed Information.                          //
// Not to be duplicated, used or disclosed                    //
// except under terms of license.                             //
//                                                            //
// Revision History:                                          //
////////////////////////////////////////////////////////////////

/* Driver for routine lfit */

#include <stdio.h>
#include <math.h>
#define NRANSI
#include "nr.h"
#include "nrutil.h"

#define SPREAD 0.1
#define NTERM 5
#define MAX_I_POINTS            9500 // max number of data points
extern  float   huge x[MAX_I_POINTS], y[MAX_I_POINTS];
int xlfit(int NPT);
void funcs(float x,float afunc[],int ma);

void funcs(float x,float afunc[],int ma)
{
        int i;

        afunc[1]=1.0;
        afunc[2]=x;
        afunc[3]=x*x;
        afunc[4]=x*x*x;
        afunc[5]=x*x*x*x;
//      afunc[6]=x*x*x*x*x;
//      afunc[7]=x*x*x*x*x*x;
//      afunc[8]=x*x*x*x*x*x*x;
//      afunc[9]=x*x*x*x*x*x*x*x;
//      for (i=3;i<=ma;i++) afunc[i]=sin(i*x);
}

int xlfit(int NPT)
{
        long idum=(-911);
        int i,j,*ia;
        float chisq,*a,*xx,*yy,*sig,**covar,value;
        FILE *stream;

        ia=ivector(1,NTERM);
        a=vector(1,NTERM);
        xx=vector(1,NPT);
        yy=vector(1,NPT);
        sig=vector(1,NPT);
        covar=matrix(1,NTERM,1,NTERM);

#if 0
        for (i=1;i<=NPT;i++) {
                xx[i]=0.1*i;
                funcs(xx[i],a,NTERM);
                yy[i]=0.0;
                for (j=1;j<=NTERM;j++) yy[i] += j*a[j];
```



```
                yy[i] += SPREAD*gasdev(&idum);
                sig[i]=SPREAD;
        }
#endif
        for (i=1;i<=NPT;i++) {
                xx[i] = x[i];
                funcs(xx[i],a,NTERM);
                yy[i] = y[i];
                sig[i]=SPREAD;
        }

        stream  = fopen("xlfitout.dat","w");
        for (i=1;i<=NTERM;i++) ia[i]=1;
        lfit(xx,yy,sig,NPT,a,ia,NTERM,covar,&chisq,funcs);
        printf("\n%11s %21s\n","parameter","uncertainty");
        for (i=1;i<=NTERM;i++){
                printf("  a[%1d] = %8.6f %12.6f\n",
                        i,a[i],sqrt(covar[i][i]));
                fprintf(stream, "  a[%1d] = %8.6f %12.6f\n",
                        i,a[i],sqrt(covar[i][i]));
        }
        printf("chi-squared = %12f\n",chisq);
        fprintf(stream,"chi-squared = %12f\n",chisq);
        printf("full covariance matrix\n");
        for (i=1;i<=NTERM;i++) {
                for (j=1;j<=NTERM;j++) printf("%12f",covar[i][j]);
                printf("\n");
        }
        printf("\npress RETURN to continue...\n");
        (void) getchar();
        /* Now check results of restricting fit parameters */
        for (i=2;i<=NTERM;i+=2) ia[i]=0;
        lfit(xx,yy,sig,NPT,a,ia,NTERM,covar,&chisq,funcs);
        printf("\n%11s %21s\n","parameter","uncertainty");
        for (i=1;i<=NTERM;i++){
                printf("  a[%1d] = %8.6f %12.6f\n",
                        i,a[i],sqrt(covar[i][i]));
                fprintf(stream,"  a[%1d] = %8.6f %12.6f\n",
                        i,a[i],sqrt(covar[i][i]));
        }

        printf("chi-squared = %12f\n",chisq);
        fprintf(stream,"chi-squared = %12f\n",chisq);
        printf("full covariance matrix\n");
        for (i=1;i<=NTERM;i++) {
                for (j=1;j<=NTERM;j++) printf("%12f",covar[i][j]);
                printf("\n");
        }
        printf("\n");
        for(j=0; j<NPT; j++){
                value = a[1]
                +(a[2]*x[j])
                +(a[3]*x[j]*x[j])
                +(a[4]*x[j]*x[j]*x[j])
                +(a[5]*x[j]*x[j]*x[j]*x[j]);
//              +(a[6]*x[j]*x[j]*x[j]*x[j]*x[j])
//              +(a[7]*x[j]*x[j]*x[j]*x[j]*x[j]*x[j]
//              +(a[8]*x[j]*x[j]*x[j]*x[j]*x[j]*x[j]*x[j])
//              +(a[9]*x[j]*x[j]*x[j]*x[j]*x[j]*x[j]*x[j]*x[j]);
                fprintf(stream, "%4.2f , %4.2f\n", x[j],value);
        }
        free_matrix(covar,1,NTERM,1,NTERM);
        free_vector(sig,1,NPT);
        free_vector(yy,1,NPT);
        free_vector(xx,1,NPT);
        free_vector(a,1,NTERM);
```

```
        free_ivector(ia,1,NTERM);
        fclose(stream);
        return 0;
}
#undef NRANSI
/* (C) Copr. 1986-92 Numerical Recipes Software :!S$-)=. */
```

File: lfit.c

Wheel Hunting Data Acquisition Software

```
////////////////////////////////////////////////////////////////
// File: lfit.c                                               //
//                                                            //
// Wheel Hunting Data Acquisition Software                    //
                          //
//                                                            //
// Copyright 1995: IEM Corporation                            //
//                                                            //
// Proprietary Licensed Information.                          //
// Not to be duplicated, used or disclosed                    //
// except under terms of license.                             //
//                                                            //
// Revision History:                                          //
////////////////////////////////////////////////////////////////

#define NRANSI
#include "nrutil.h"
void covsrt(float **covar, int ma, int ia[], int mfit);
void gaussj(float **a, int n, float **b, int m);

void lfit(float x[], float y[], float sig[], int ndat, float a[], int ia[],
        int ma, float **covar, float *chisq, void (*funcs)(float, float [], int))
{
        int i,j,k,l,m,mfit=0;
        float ym,wt,sum,sig2i,**beta,*afunc;

        beta=matrix(1,ma,1,1);
        afunc=vector(1,ma);
        for (j=1;j<=ma;j++)
                if (ia[j]) mfit++;
        if (mfit == 0) nrerror("lfit: no parameters to be fitted");
        for (j=1;j<=mfit;j++) {
                for (k=1;k<=mfit;k++) covar[j][k]=0.0;
                beta[j][1]=0.0;
        }
        for (i=1;i<=ndat;i++) {
                (*funcs)(x[i],afunc,ma);
                ym=y[i];
                if (mfit < ma) {
                        for (j=1;j<=ma;j++)
                                if (!ia[j]) ym -= a[j]*afunc[j];
                }
                sig2i=1.0/SQR(sig[i]);
                for (j=0,l=1;l<=ma;l++) {
                        if (ia[l]) {
                                wt=afunc[l]*sig2i;
                                for (j++,k=0,m=1;m<=l;m++)
                                        if (ia[m]) covar[j][++k] += wt*afunc[m];
                                beta[j][1] += ym*wt;
                        }
                }
        }
        for (j=2;j<=mfit;j++)
                for (k=1;k<j;k++)
                        covar[k][j]=covar[j][k];
        gaussj(covar,mfit,beta,1);
        for (j=0,l=1;l<=ma;l++)
                if (ia[l]) a[l]=beta[++j][1];
        *chisq=0.0;
        for (i=1;i<=ndat;i++) {
                (*funcs)(x[i],afunc,ma);
                for (sum=0.0,j=1;j<=ma;j++) sum += a[j]*afunc[j];
                *chisq += SQR((y[i]-sum)/sig[i]);
        }
        covsrt(covar,ma,ia,mfit);
        free_vector(afunc,1,ma);
```



```
        free_matrix(beta,1,ma,1,1);
}
#undef NRANSI
/* (C) Copr. 1986-92 Numerical Recipes Software :!S$-)=. */
```

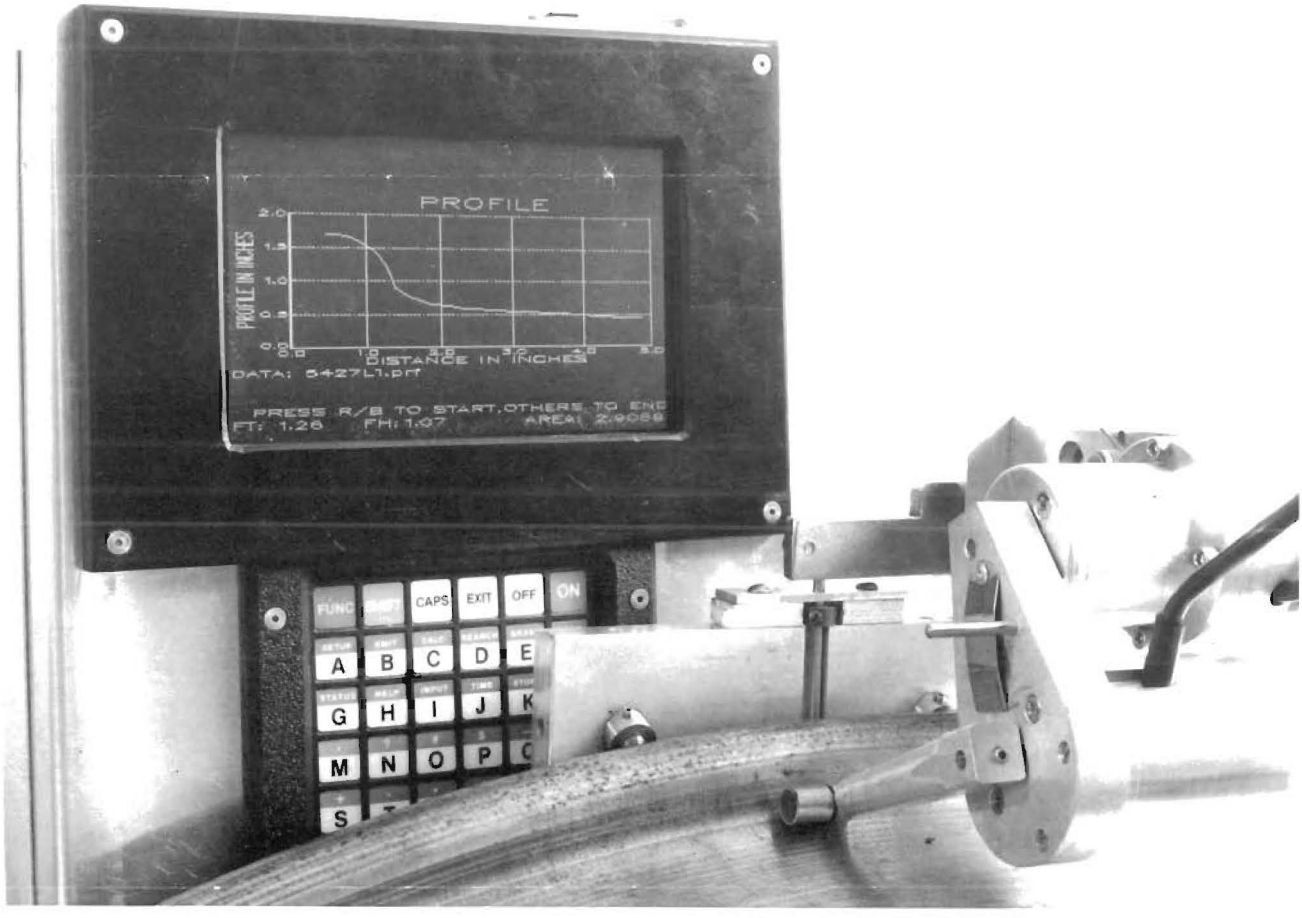
File: gammln.c

Wheel Hunting Data Acquisition Software

```
////////////////////////////////////////////////////////////////
// File: gammln.c                                             //
//                                                            //
// Wheel Hunting Data Acquisition Software                    //
                        //
//                                                            //
// Copyright 1995: IEM Corporation                            //
//                                                            //
// Proprietary Licensed Information.                          //
// Not to be duplicated, used or disclosed                    //
// except under terms of license.                             //
//                                                            //
// Revision History:                                          //
////////////////////////////////////////////////////////////////

#include <math.h>

float gammln(float xx)
{
        double x,y,tmp,ser;
        static double cof[6]={76.18009172947146,-86.50532032941677,
                24.01409824083091,-1.231739572450155,
                0.1208650973866179e-2,-0.5395239384953e-5};
        int j;

        y=x=xx;
        tmp=x+5.5;
        tmp -= (x+0.5)*log(tmp);
        ser=1.000000000190015;
        for (j=0;j<=5;j++) ser += cof[j]/++y;
        return -tmp+log(2.5066282746310005*ser/x);
}
/* (C) Copr. 1986-92 Numerical Recipes Software :!S$-)=. */
```

International Electronic Machines Corporation

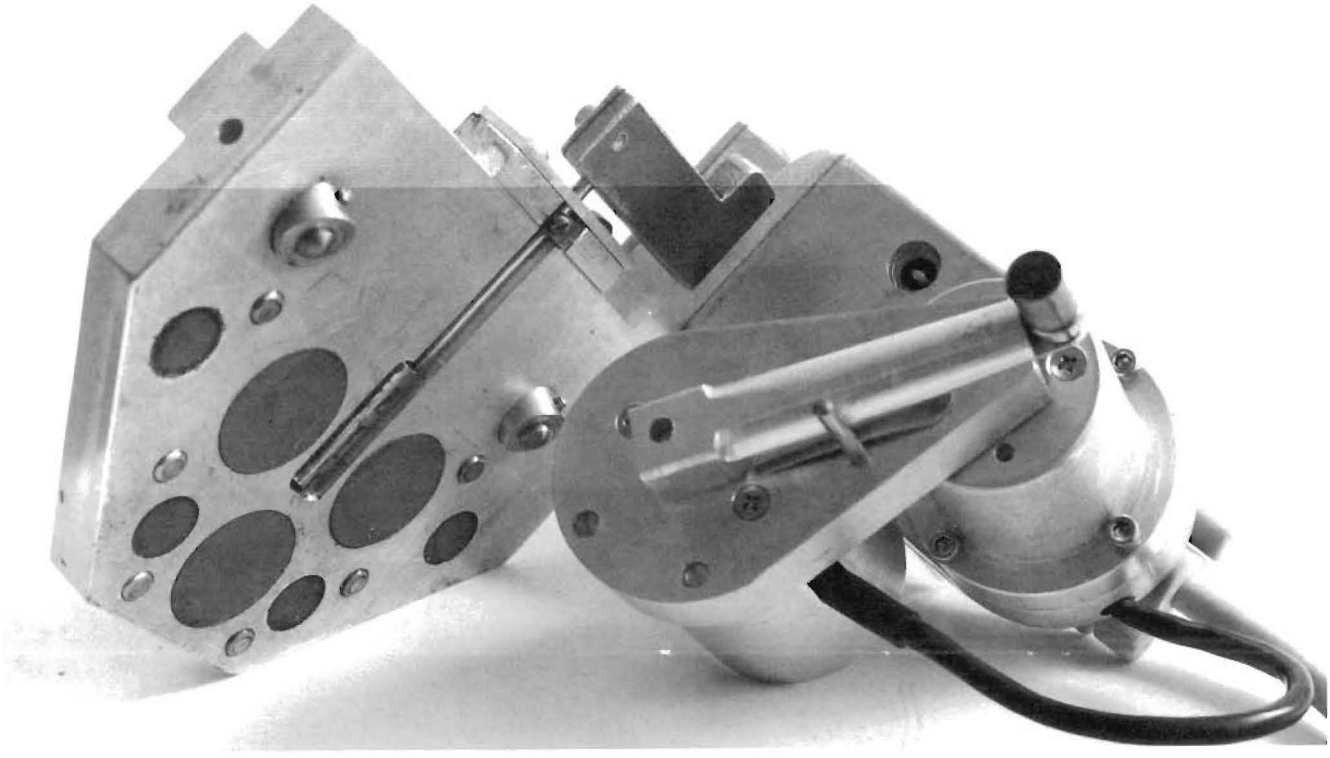
RR WHEEL PROFILOMETER



IEM

Provides Precise Cross Section Profiles of Standard Mounted RR Wheels

Fast, Accurate, Portable and PC Compatible



Gauge Head

Flexibility / Fit The Gauge Head is small enough to fit on virtually all mounted locomotive and transit wheels.

Reliable Reference Points The most important single variable in developing reliable, repeatable measurements is the mounting of the gauge head on the wheel. The IEM reference system uses a series of powerful magnets and a rr industry proven set of high resolution contact sensors to ensure the repeatable placement of the gauge on the wheel. When the contact sensors are not fully engaged, the system informs the operator with a clear message on the system display.

High Resolution The resolution of the system in the IEM 5/1000ths model at 0.0005 inches.

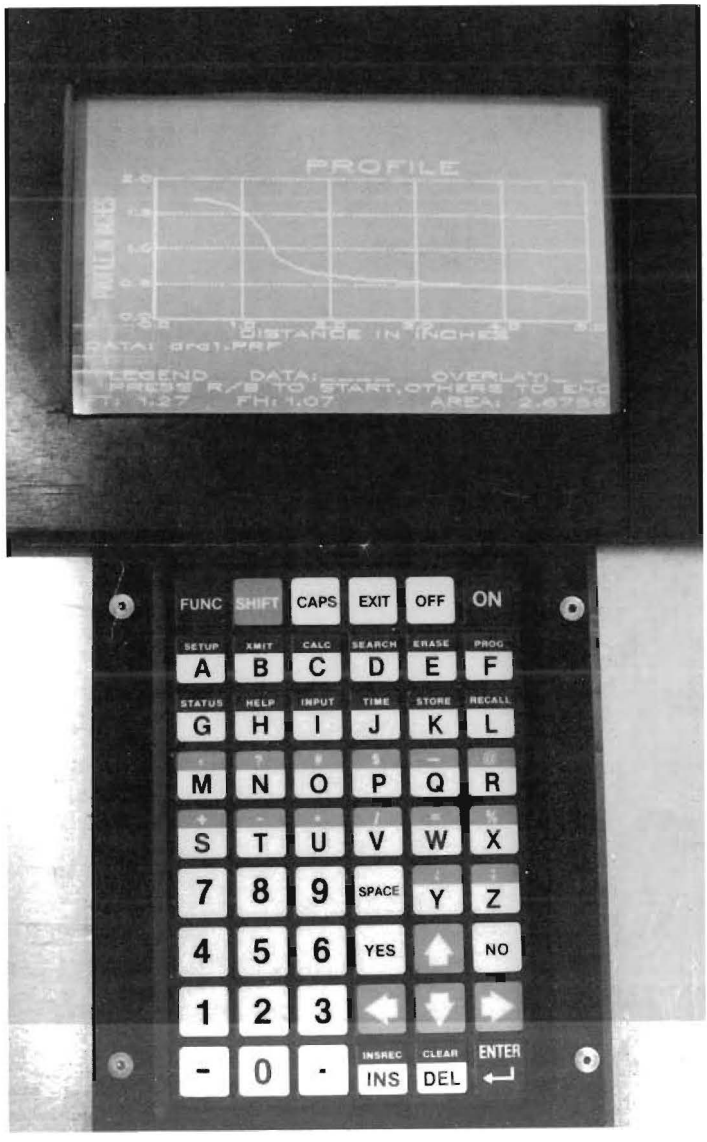
Top Of Flange Reference Points Accommodate Plastic Flow The IEM top of flange contact sensors are designed to contact the back face of the top of the flange to eliminate the possibility that the references can be distorted by plastic flow up the face of the flange.

Controller

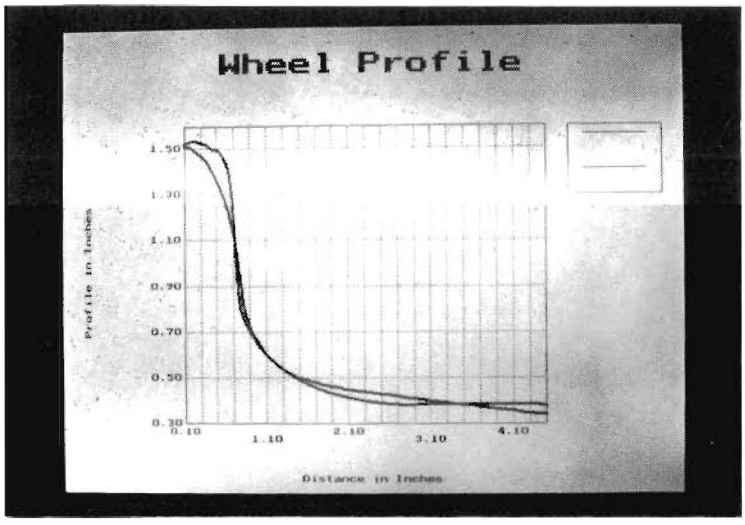
Rugged Industrial Pc The IEM Profilometer Controller is built specifically for the RR environment. It is part of a complete line of industrial computers produced by IEM. As such it includes such features as: rugged aluminum body, easily replaceable battery pack, full alphanumeric membrane keyboard, state of the art PCMCIA card which is resistent to shock, moisture and electro-magnetic interference, an industrial power supply, three expansion slots and can be upgraded to incorporate 4 meg of RAM.

Pc Compatibility The IEM software can operate directly on PC compatible computers.

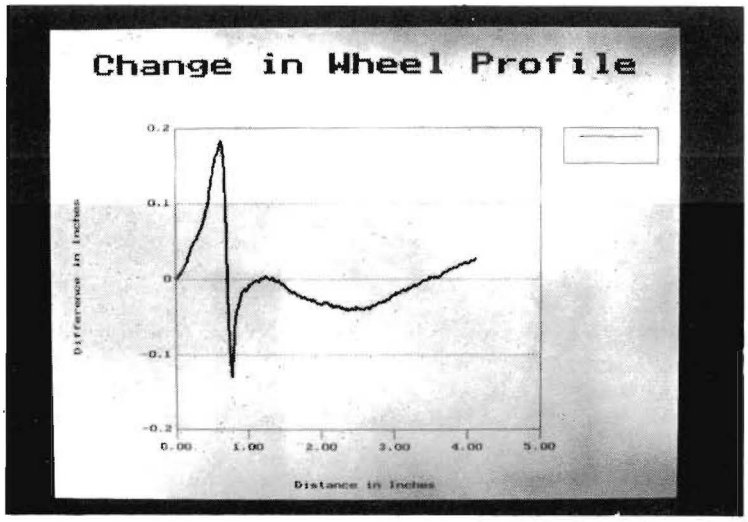
Display The IEM Profilometer comes equipped with a CGA full graphics screen display providing superior readability in bright sunlight.



Rugged System Controller has Full Alphanumeric Keyboard



Profiles Can Be Superimposed



Or Delta Y Can Be Plotted

Rugged Carrying Case The padded rugged over the shoulder carrying case is made out of a special oil resistent material.

Built-in Keyboard Port The Controller as a keyboard port so a standard PC keyboard can be used in office environments.

Replaceable Battery Pack The standard Battery Pack will provide three hours of uninterrupted operation. Back up units are standard equipment. The battery pack is easily installed.

Software

Multitude of Operator Support Features The software has been very carefully designed by experienced wheel measurement experts to be practical and easy to use in the field. The idea is to keep the number of keystrokes required to take measurements and store files to an absolute minimum once the operator gets under the car.

Menu Driven File Retrieval It is never necessary to remember the name of a file. Pressing the Enter Key twice automatically brings up a list of files relevant to the current operation. Using the arrow keys it is very easy to highlight the desired file and retrieve it.

Automatic File Name Generation The system uses the Car Number and the Wheel Position to automatically generate file names. If the wheel has been previously measured, the system automatically adds one to the file name suffix allowing up to ten measurements of the same wheel.

Automatic Wheel Positioning The system allows the operator to indicate the order in which he plans to measure the wheels and then automatically updates the wheel position as the operator moves from wheel to wheel.

Full Array of Data Analysis Features

Automatically Provides Standard RR Measurements The system automatically provides standard AAR compatible measurements of flange thickness and flange height. The rules governing these measurements can be adjusted for measurements of non AAR standard profiles.

Automatic Data Smoothing An automatic data smoothing function is available to eliminate data distortions caused by dirt and vibrations during measurement.

Calculates Lost Area In addition to the ability to automatically compare two profiles, the system automatically calculates the actual difference in the amount of service between the two.

Built-in Spreadsheet The Profilometer includes a leading built-in spreadsheet and data management package.

Built-in Cad Capabilities A built-in CAD package allows the operator full abilities to shift, move, and rotate profiles in the field for maximum ease of use.

Flexible Data Analysis Profiles can be superimposed on one another or the change in the Y axis can be plotted.

Unlimited Magnification Unlimited magnification is available through the built-in spreadsheet program.

ASCII Compatibility The IEM gauge produces ASCII output.

Compares Actuals To Drawings By using the optional DXF transfer utility, the Profilometer can compare actual profiles with original drawings.

Applications

QC of Wheel Profiling Equipment The Profilometer lets you see if the cut you're getting is the cut you want.

QC of Incoming Wheelset The Profilometer can compare original drawings with actual profiles to determine if incoming wheels are meeting specifications.

Analyze Ride Quality Reliably determine the impact of minor changes in wheel profile on ride quality.

Evaluate Maintenance Strategies Get hard data on actual wheel wear patterns. See the impact of alternate wheel profiles, alternative wheel truing schedules, and different flange lubrication and/or brake shoe technologies.

Wheel Maintenance Research Using the Profilometer it is now possible to detect precise patterns of wheel wear that would not show up in the three point profile determined by the finger gauge or the IEM Electronic Wheel Gauge. The Gauge will also play a valuable role in diagnosing patterns of wheel failure.

THE IEM RR WHEEL PROFILOMETER

Specifications

GAUGE HEAD

Horizontal Resolution Measures the worn profile at 200 points per inch with placement accuracy of 1 mil.

Vertical Resolution The 5/1000ths of an inch model has a resolution of 5/10,000ths of an inch and is repeatable to 5/1000ths of an inch.

Reference Reference options include back face and top of flange.

System accuracy 5/1000th of an inch.

Vertical Measurement Range 1.5 inches

Horizontal Measurement Range 5.3 inches

Measurement Time 3 - 5 seconds per wheel.

Size/Weight 6 by 5 by 7 inches / 3 Pounds

CONTROLLER

Microprocessor INTEL 286

Floppy drive PCMCIA Card

User Interface Full alphanumeric 54 key membrane keyboard

Operating temperature range from 0 C to +50 C.

Size/Weight 11.5 by 8 by 2.75 inches / 8 Pounds

Power Supply 110 VAC - 60 Hz. Internal rechargeable battery with minimum operating time of 3 hours.

SOFTWARE

Wheel Identification Schema

- Date
- Place
- Operator name
- Vehicle type
- Vehicle ID
- Wheel position (Axle #, L/R)
- Measurement Number
- Wheel SN
- Wheel DIA

Display of Data

- Superimposed display of two profiles
- Delta Y based on reference profile
- Delta Y MAX and X position
- Worn area
- Flange thickness
- Flange height

Built-in CAD Functions

- Zoom
- Move in all four directions
- Rotate
- Distance between two points
- Point by Point display
- Export to CAD compatible file formats

Printing support

- Over 100 printers including HP LaserJet II® and compatibles.

Plotting support

- Over 100 plotters including HP 7475A® and compatibles.

For More Information Contact:

INTERNATIONAL ELECTRONIC MACHINES CORPORATION

71 Fourth Street, Troy, New York 12180

(518) 273-2445 FAX (518) 273-2446

IEM