Large-scale Windows 95-based data-acquisition system using LabVIEW

L. Mandrake[a] and W. Gekelman
Department of Physics and Astronomy, LAPD Plasma Laboratory, UCLA, Los Angeles, California 90095

(FAST-TRACK PAPER: Received 13 May 1997; accepted 17 June 1997)

Hardware and software for a high-speed, pulsed-data-acquisition system are described. Data can be obtained via any device attachable to a VXI crate, GPIB controller, or directly via serial or parallel ports. Stepper motors controlled via serial port automate probe movement. LabVIEW and custom C++ modules are used to handle setup, data gathering and processing, and user interface. Experimental parameters can be controlled at each point via any GPIB-ready device. *© 1997 American Institute of Physics.* [S0894-1866(97)02105-6]

INTRODUCTION

The need for large-volume data sets in plasma physics (and many other areas) has mandated computer-assisted data acquisition for many years.[1] Often, this need finds solution via a digital oscilloscope, and indeed modern oscilloscopes offer powerful functions, storage capability, digitization options, and GPIB transport of data. However, to characterize innately three-dimensional structures such as exist in plasma requires thousands of spatial sampling locations. These data sets are presently of order 1 Gbytes in size and are expected to be two orders of magnitude greater within several years. They must be transferred to disk for storage and analysis; however, GPIB offers low throughput that causes unacceptable lag in data runs that can already take days to complete.

One solution is to custom-create an entire data-acquisition system from scratch, with manual code written to control each device and handle all user-interface issues. In the past, this was the only option available.[2] A DAQ (data-acquisition) system was created by interfacing a VAX 3800 to a CAMAC crate with Fortran code and VMS. The user interface utilized the VMS screen-management utility, which was cumbersome to program, but at the time was nearly all that existed. This system was in use for over five years, but upgrading it became difficult as new devices became available. Finally, CAMAC has given way to VXI technology, offering bus transfer rates that are 10 times faster.

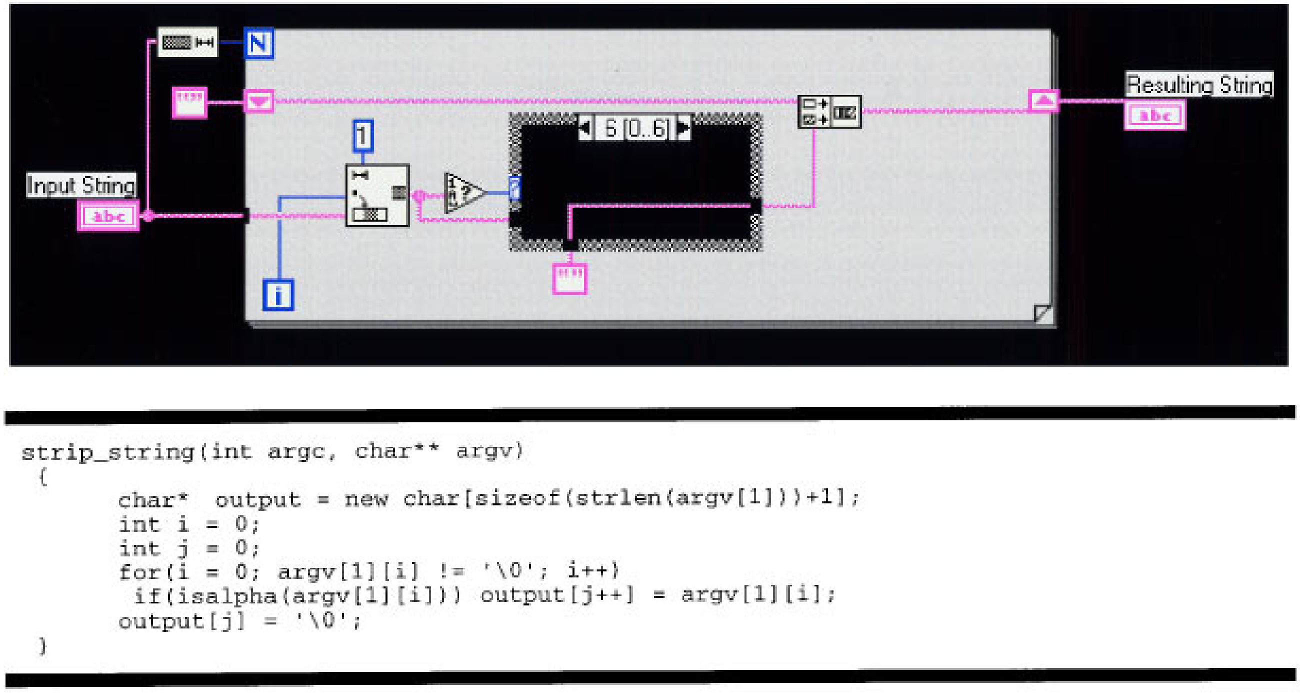
Modern computing technology has provided inexpensive PC-based workstations that are capable of the speed required by DAQ application. This combined with the advent of Windows 95/NT has brought about a stable, multitasking environment capable of handling the many networking issues involved in a large-scale DAQ system. Via the commercial package LabVIEW (National Instruments, http://www.natinst.com), which offers a powerful and easily customizable user interface and VXI crate technology, a system was achieved that fully replaces our previous DAQ system and performs an order of magnitude faster. Nevertheless, commercial packages still do not offer powerful, instant data-acquisition systems. In this article we show the hardware and software considerations involved in the design and development of such a system.

I. ORIGINAL EXPERIMENTAL NEEDS

While the DAQ system described herein is equally applicable to a wide range of scientific measurement uses, it is helpful to understand the actual experimental needs surrounding its development. This lab chiefly measures very high frequency oscillations of **E** and **B** fields within a large (10-m-long, 50-cm-diam) plasma[3] that can exceed the current sampling technology of around 5 GHz. More slowly changing parameters such as density and temperature are also measured that can oscillate at kilohertz or lower frequencies. Hence, the DAQ system should be able to handle multiple timescales (and hence the very different sort of equipment necessary to measure them). However, the chief characteristic of our experimental setup to affect the design of the DAQ system is that our experiment only lasts a few milliseconds with structure on the order of nanoseconds; it is therefore impossible for any modern sampling technology to record all of the events in a single experiment. The experiment is reproduced once per second, allowing construction of the time regions of interest through the sum of acquired data. In this way, we may piece together the entire time evolution of the plasma shot-by-shot, as well as using averaging techniques to improve our received signal.

In this way, the DAQ system is based on the idea of a single experimental trigger firing on the order of once per second, wherein the data must be offloaded, processed, stored, and the devices armed for the next acquisition. Failure to meet this criterion will cause shots of the plasma to be ‘‘missed,’’ multiplying data run time. As our data runs tend to consist of up to one million shots, a single multiplication by a factor of 2 can yield a completion time of two weeks instead of one.

[a] Corresponding author; 1000 Veterans Avenue, 15-70 Rehabilitation Building 24 Los Angeles, CA 90095; e-mail: mandrake@lords.com



```
strip_string(int argc, char** argv)
 {
       char*  output = new char[sizeof(strlen(argv[1]))+1];
       int i = 0;
       int j = 0;
       for(i = 0; argv[1][i] != '\0'; i++)
        if(isalpha(argv[1][i])) output[j++] = argv[1][i];
       output[j] = '\0';
 }
```

Figure 1. Top: A LabVIEW program that strips nonalphanumeric characters from a string. Here, the large grey box represents a For loop. Within, the checkered white box represents a Case statement, showing what will happen on only one of six cases. All the other cases also have wired code within the Case box, but cannot be shown simultaneously due to the ''switching'' visual nature of case statements in LabVIEW. Bottom: A corresponding function in C++, achieving the same results as the above graphical code (with the exception of further memory handling required later in this code).

II. OVERVIEW OF LABVIEW AND THE *G* LANGUAGE

LabVIEW utilizes a language known as G, a general-purpose programming system much like C or Fortran. However, unlike its predecessors, G is entirely graphical—no text code need ever be entered into the LabVIEW system. Icons representing modular functions of a program are wired together, functioning when all of their inputs are ''ready.'' This fundamental difference from traditional ''line counter'' methods of programming creates an entirely new paradigm with strong strengths and weaknesses.

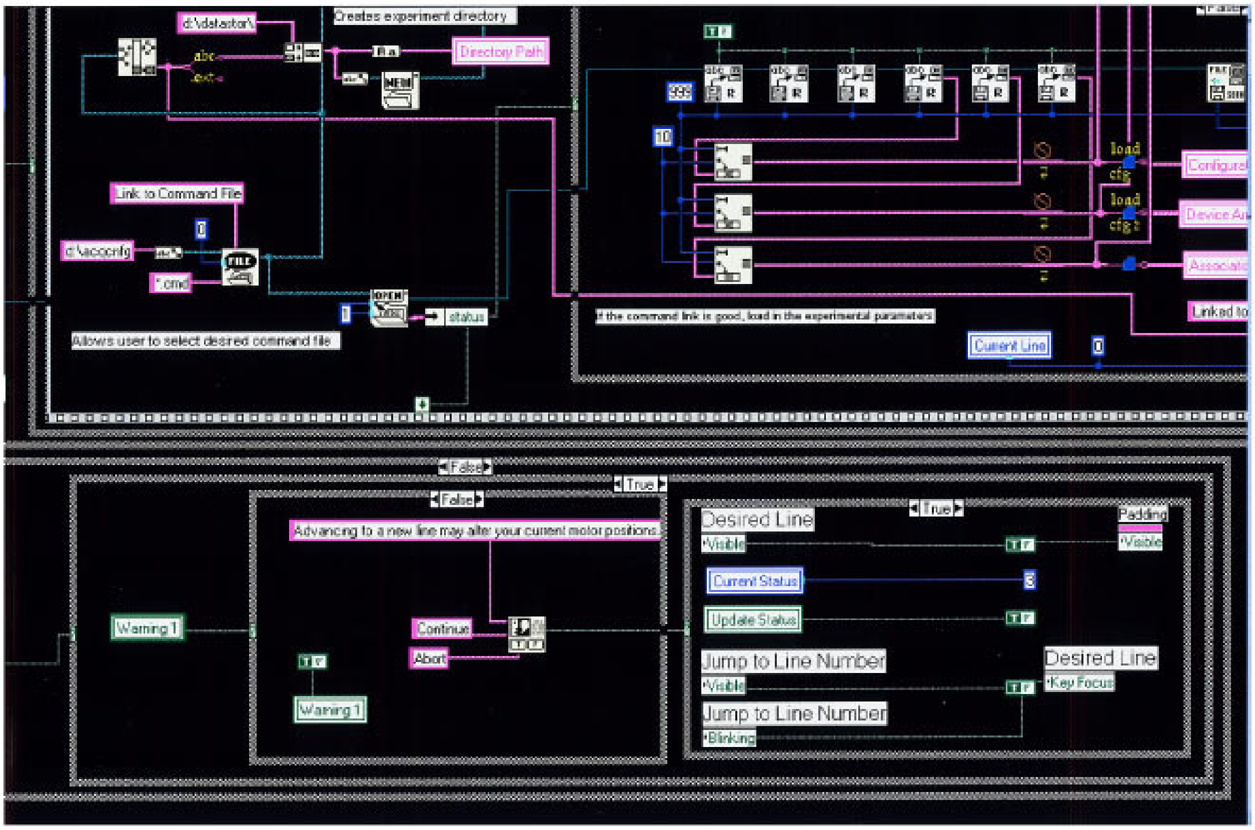
For development of simple, instrument-controlling applications, LabVIEW comes equipped with a library of high-level premade modules that provide existing functionality. However, for a system as large as ours, we were forced to utilize the lower-level basic drivers for each device. This was detrimental to some of the advantages of LabVIEW, for it is clearly optimized for rather simple applications.

A. LabVIEW advantages

User interfaces are often responsible for a significant fraction of coding effort in any project. Due to its graphical nature, creating a user interface in LabVIEW is exceptionally easy. Objects for buttons, toggles, graph outputs, number input/output, etc. are already available with a single menu selection. In this way, the tremendous overhead of a user interface is eliminated via an easily customizable ''front panel'' that all LabVIEW modules possess. Originally designed to simulate actual, physical devices commonly found in a lab, LabVIEW's available interface objects offer an acceptable selection for most needs. In this application, the look-and-feel desired was not of an actual oscilloscope or multimeter, however, resulting in some effort spent overcoming the ''traditional'' nature of the interface objects.

Due to the fact that most variables in a traditional program are in actuality used as temporary storage for processing and manipulation (and hence, do not contain data in which the user is most likely interested), reading a text-based program is often clouded with the need to know what variables actually matter. Due to the graphical nature of LabVIEW, no temporary variables exist. Wires connecting one module to another represent the path of data flow; actual memory allocation is handled transparently by the system. In this way, programmers present and future can more easily follow the flow of meaningful information without referring to variable description comments. See Fig. 1 for an example LabVIEW function (or ''diagram'') and its C++ equivalent.

Trivial errors such as syntax, accidental misspellings, and variable/function declaration slow the production effort of even experienced programmers. In LabVIEW, syntax has been replaced with intuitive, graphical representations that offer most of the flexibility and power of traditional languages but with a built-in, context-sensitive editor that instantly alerts the programmer to an invalid connection.



Creates experiment directory

Allows user to select desired command file

If the command link is good, load in the experimental parameters

Advancing to a new line may alter your current motor positions

Figure 2. A screen shot of a more advanced LabVIEW program. The series of Case and While boxes can become quite involved (each checkered line represents a conditional). This is the main controlling program that runs the DAQ system interface. Normally, LabVIEW programs are not this dense with conditionals due to proper modular programming. However, main user interfaces can require this complexity.

B. LabVIEW disadvantages

Due to the fact that the LabVIEW program exists in a two-dimensional plane, one can become boxed into a region that has already been surrounded by other diagrams. In sufficiently complex programs, expanding the program to allow space for further modifications can be a tedious task. This can be mostly alleviated by proper modular construction and forethought in diagram design, but remains a serious concern for future versions of LabVIEW and those interested in its use.

Objects on the programming diagram often represent physical objects on the user interface (for example, a knob or switch). There is an inability to cut, paste, and edit code as smoothly as text programmers are commonly familiar with, causing frustration for those parts of a program that are very similar. Ease of modification has been increased at the expense of difficulty in duplication.

LabVIEW utilizes ‘‘switchable’’ boxes to represent decision statements such as Case. These boxes can only show one possible diagrammatic response at a time (either True or False for a Boolean variable, etc.). It is then impossible to ever see all of a LabVIEW program at once without toggling all such boxes, therefore making a traditional printout impossible. See Fig. 2 for an example of a complex LabVIEW program in which only a fraction of the actual code is visible.

Due to the fact that LabVIEW diagrammatic programs are not text-based, it is often very challenging to document code efficiently. LabVIEW does present the ability to tag an object with text or insert text as an object into the diagram, but the barrier to code documentation has been raised sufficiently that code commenting is not possible on a line-by-line basis. However, modular functions have an excellent documentation feature that is intelligently linked to the on line help, almost making up for this inadequacy.

III. HARDWARE SETUP

In its native implementation, the DAQ system utilized a Pentium 150 with 24 Mbytes of RAM and a 4.2-Gbyte hard drive for program and data storage. As shown in Fig. 3, the computer is then connected via TCP/IP Ethernet to other computers more suited to large-scale data processing and visualization. An Internet connection is mandatory to maintain current driver revisions. Commercial software (NFS Maestro by Hummingbird Communications, http://www.hummingbird.com) allows transparent mounting of Unix drives across the network from the PC, permitting data to be stored on the local hard drive or remotely as desired.

Aside from acquiring the data, the DAQ system must also control the experiment and move probes in and out of the device through vacuum seals. The serial port allows communication with stepper motor controllers connected to the stepper motors responsible for such movement. Stepper

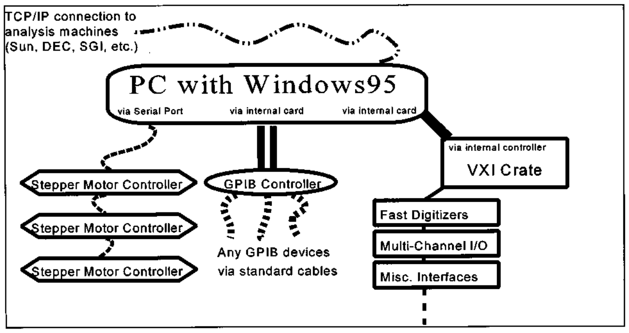


Figure 3. Hardware setup diagram.

motor controllers may be connected together up to eight per serial port and controlled via a simple ASCII-based language.

For flexible device control, GPIB [488.1—1987 IEEE (R1994) Standard Digital Interface for Programmable Instrumentation (ANSI)] functionality is provided via an internal controller (National Instruments, http://www.natinst.com). While GPIB is capable of acquiring data, it is limited by its inherent speed, rendering it too slow for large volume data sets. Despite the listed speed potentials of 1 mbytes/s throughput tends to be much lower than the above-quoted numbers due to device response times and protocol overhead. Still, GPIB is valuable for controlling wave function generators, timers/delays, and other low-bandwidth devices as it remains at present commonly available as an option on most scientific instruments.

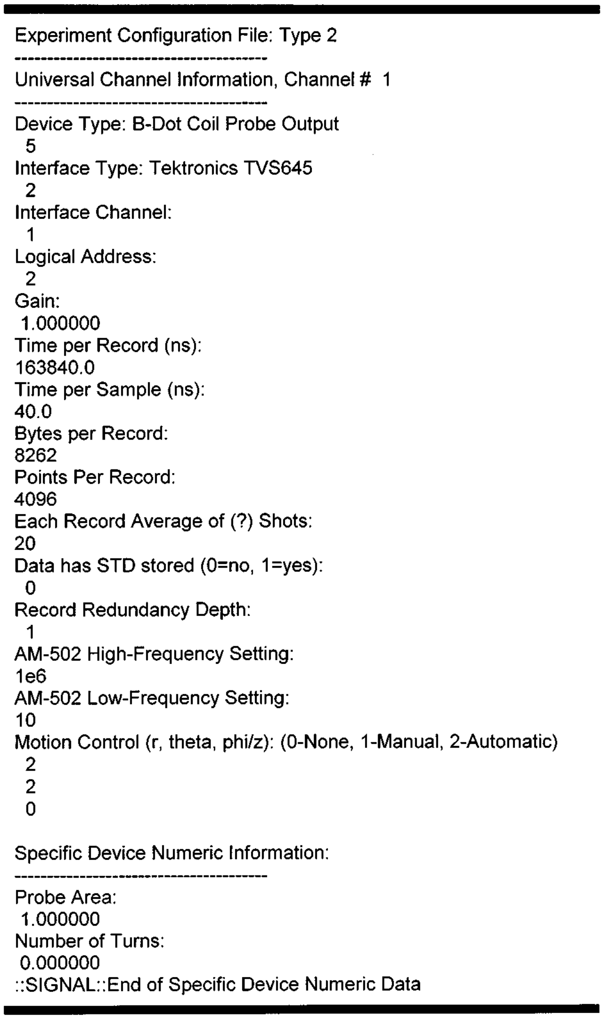
VXI [1155—1992 IEEE Standard VMEbus Extensions for Instrumentation: VXIbus (ANSI)] is the modern equivalent of CAMAC. A crate-based technology, VXI devices are large card-based objects that can be inserted into a VXI crate. The crate then talks to the PC via dedicated cards in both the PC and the VXI crate. The transfer rate between the computer and controller is around 33 mbytes/s, though again the actual transfer rate tends to be dramatically slower due to device response times. Most *A* to *D* or *D* to *A* devices are available for VXI with many optimizations for channels, speed, memory, price, etc. For this lab's application, high-speed digitizers (Tektronix, http://www.textronix.com) (8-bit, 5-GHz max) were implemented. Our connector from the create to the PC is via a device by National Instruments despite the fact our crate is from Tektronix. A number of other manufacturers also produce VXI modules and crates.

IV. CUSTOMIZABILITY AND REPRODUCIBILITY

For our design, the two main issues were customization and reproducibility. The first need stems from the fact that we have a single, large experimental device upon which many scientists perform unrelated experiments using different input and output devices. Users must be able to rapidly select between different setups swiftly and easily. The second need of reproducibility of the DAQ system setup is essential for the averaging techniques we employ.

The solution used was a collection of menus to record all customizable parameters of the system, from those aspects irrelevant to data acquisition but mandatory for physical understanding (operator's name and experimental parameters) to critical DAQ parameters (data record size and sampling rate). These are written to disk by the setup program for permanent storage in a format that is human-readable. Later, these files can be reloaded and modified by the user for future experiments or simply examined to remind the user of what parameters were used. In this way, all aspects of experimental setup are kept entirely separate from the act of acquisition itself.

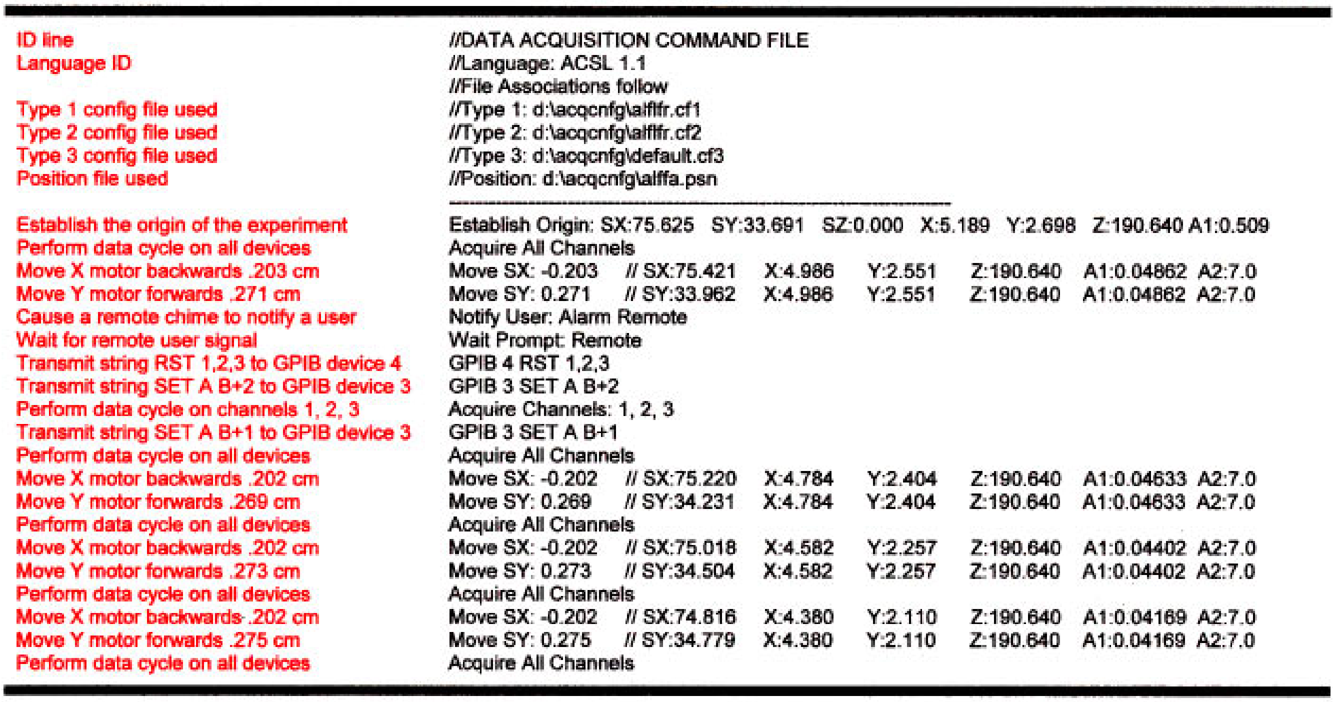
Furthermore, the setup files were split into three separate types. The first contains user information that is critical to the understanding of physical phenomenon in the data but is irrelevant to the data acquisition itself. The second type of configuration file contains a detailed account of those input devices connected to the system including VXI digitizers, GPIB oscilloscopes, or any other source of input as well as all customizable parameters of said devices (see Fig. 4 for a sample configuration file). It also contains information on which probes are connected to stepping mo-



```
Experiment Configuration File: Type 2
---------------------------------------
Universal Channel Information, Channel # 1
---------------------------------------
Device Type: B-Dot Coil Probe Output
 5
Interface Type: Tektronics TVS645
 2
Interface Channel:
 1
Logical Address:
 2
Gain:
 1.000000
Time per Record (ns):
163840.0
Time per Sample (ns):
40.0
Bytes per Record:
8262
Points Per Record:
4096
Each Record Average of (?) Shots:
20
Data has STD stored (0=no, 1=yes):
 0
Record Redundancy Depth:
 1
AM-502 High-Frequency Setting:
1e6
AM-502 Low-Frequency Setting:
10
Motion Control (r, theta, phi/z): (0-None, 1-Manual, 2-Automatic)
 2
 2
 0

Specific Device Numeric Information:
---------------------------------------
Probe Area:
 1.000000
Number of Turns:
 0.000000
::SIGNAL::End of Specific Device Numeric Data
```

Figure 4. A sample configuration file. All such files in this system are ASCII-based and editable. The configuration files double as reference files for users.



```
ID line                                       //DATA ACQUISITION COMMAND FILE
Language ID                                   //Language: ACSL 1.1
                                              //File Associations follow
Type 1 config file used                       //Type 1: d:\acqcnfg\alflfr.cf1
Type 2 config file used                       //Type 2: d:\acqcnfg\alflfr.cf2
Type 3 config file used                       //Type 3: d:\acqcnfg\default.cf3
Position file used                            //Position: d:\acqcnfg\alffa.psn
                                              ------------------------------------------------------------
Establish the origin of the experiment        Establish Origin: SX:75.625  SY:33.691  SZ:0.000  X:5.189  Y:2.698  Z:190.640 A1:0.509
Perform data cycle on all devices             Acquire All Channels
Move X motor backwards .203 cm                Move SX: -0.203   // SX:75.421   X:4.986   Y:2.551   Z:190.640   A1:0.04862  A2:7.0
Move Y motor forwards .271 cm                 Move SY: 0.271    // SY:33.962   X:4.986   Y:2.551   Z:190.640   A1:0.04862  A2:7.0
Cause a remote chime to notify a user         Notify User: Alarm Remote
Wait for remote user signal                   Wait Prompt: Remote
Transmit string RST 1,2,3 to GPIB device 4    GPIB 4 RST 1,2,3
Transmit string SET A B+2 to GPIB device 3    GPIB 3 SET A B+2
Perform data cycle on channels 1, 2, 3        Acquire Channels: 1, 2, 3
Transmit string SET A B+1 to GPIB device 3    GPIB 3 SET A B+1
Perform data cycle on all devices             Acquire All Channels
Move X motor backwards .202 cm                Move SX: -0.202   // SX:75.220   X:4.784   Y:2.404   Z:190.640   A1:0.04633  A2:7.0
Move Y motor forwards .269 cm                 Move SY: 0.269    // SY:34.231   X:4.784   Y:2.404   Z:190.640   A1:0.04633  A2:7.0
Perform data cycle on all devices             Acquire All Channels
Move X motor backwards .202 cm                Move SX: -0.202   // SX:75.018   X:4.582   Y:2.257   Z:190.640   A1:0.04402  A2:7.0
Move Y motor forwards .273 cm                 Move SY: 0.273    // SY:34.504   X:4.582   Y:2.257   Z:190.640   A1:0.04402  A2:7.0
Perform data cycle on all devices             Acquire All Channels
Move X motor backwards .202 cm                Move SX: -0.202   // SX:74.816   X:4.380   Y:2.110   Z:190.640   A1:0.04169  A2:7.0
Move Y motor forwards .275 cm                 Move SY: 0.275    // SY:34.779   X:4.380   Y:2.110   Z:190.640   A1:0.04169  A2:7.0
Perform data cycle on all devices             Acquire All Channels
```

Figure 5. A sample command file. Each line represents a single, distinct action to be performed by the DAQ system. The red text to the left describes each line and is not part of the command file itself.

tors and which are fixed or moved manually. The third type of configuration file contains motion information [for example, along which axes (X,Y,Z) the probes will move] and links to a separately provided file that contains position information. This fourth file (called a Position file) contains a complete list of all spatial positions to be visited in the desired order and angles representing the probe orientation at that point in space. Together, all four of these files are combined by the setup program to generate a single, large text file called a Command file using the ACSL (pronounced ''axel,'' Abbreviated Command Scripting Language) language, which is merely a set of commands understood by the DAQ system.

The use of the Command file is an architectural choice with many advantages. As in the example Command file (Fig. 5), the header of the file contains information on all other configuration files used in the generation (and future execution) of this particular Command file. Following this header is an exact listing of every action the DAQ system shall perform, in ACSL (as defined above, see Fig. 6) that can be viewed and edited with a standard text editor (though it is normally generated by the configuration program). In this way, the Command file is the central facet of a data-acquisition run. Its name is used as a key for where to store the acquired data (to prevent users from accidentally overwriting each other's data). It contains links to all information required to perform a reproducible data run. Finally, it contains a systematic listing of all actions to be performed. The DAQ system itself is freed from any configuration issues entirely, becoming a script executor.

Current Command Set of ACSL Language

Acquire All Channels	Arms and retrieves data from all devices
Acquire Channels: **n1**, **n2**, **n3**, ...	Arms and retrieves data from specified channels
Notify User: Alarm Console	Causes the computer to make an alarm
Notify User: Alarm Remote	Causes the computer to signal a remote chime by the machine
Wait: **n1**	Waits **n1** milliseconds
Wait Prompt: Console	Requests the user to hit a key before continuing
Wait Prompt: Remote	Requests the user to push a remote switch before continuing
GPIB **n1** **COMMAND_STRING**	Sends a GPIB command string to GPIB device **n1**
Move **X**: **n1** // x: **x** y: **y** z: **z** a1: **a1** a2: **a2**	Moves the **X**(Y,Z) motor **n1** cm, resulting in a new location
Establish Origin: sx: **sx** sy: **sy** sz: **sz** x: **x** y: **y** z: **z** a1: **a1**	Establishes the current position of the stepper motors to the specified position if their position is undefined, or else moves to this position.

Figure 6. Current supported commands in the ACSL language. This language can easily be expanded to incorporate new functionality for future users.

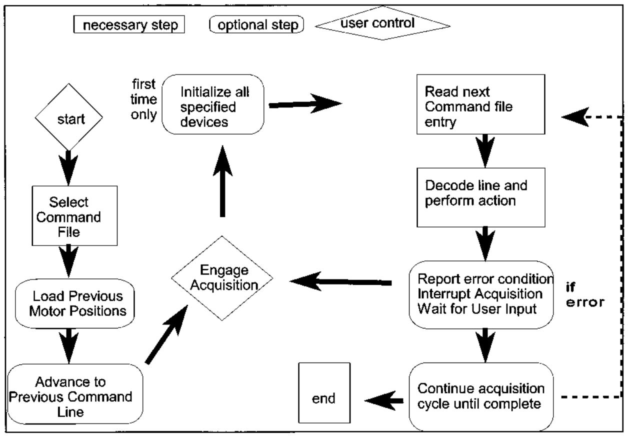

necessary step
optional step
user control
start
first time only
Initialize all specified devices
Read next Command file entry
Select Command File
Decode line and perform action
Engage Acquisition
Report error condition Interrupt Acquisition Wait for User Input
if error
Load Previous Motor Positions
Advance to Previous Command Line
end
Continue acquisition cycle until complete

Figure 7. Flow of DAQ system after Command file has been generated.

Further advantage of the Command architecture is the ease in which new commands can be added to ACSL to enhance the ability of the DAQ system while maintaining old Command file integrity; a program developer with some proficiency in LabVIEW could easily add such commands. The third configuration step (after specifying user information and I/O setup) allows the user to type ''Commands at Each Point.'' If nothing is specified, this defaults to ''Acquire all Channels'' (meaning arm all devices specified and store their data), the user is free to enter any number of text lines into this area utilizing ACSL. For instance, the user might want to issue a command to set a delayed pulse generator to a certain time after the main experimental trigger, take some shots, then set the generator to another time delay, and take more shots. This would be performed at each spatial location, so long as the hardware involved is able to receive GPIB input.

Command files also grant the user complete control in the event of errors, in that they can immediately see what has been executed and what command caused the error. Therefore, by recording the line in the Command file where the Acquisition stopped, they can resume the acquisition with continuity of their data files. (See Fig. 7 for the cycle of operation of the DAQ system once configuration files have been generated.)

V. SPEED

For example, consider data measured on 10 planes with each plane possessing on the order of 500 spatial positions. Given that moving between spatial locations takes on the order of a second and a 1-Hz experiment-repetition rate, an experiment can take a day of continuous data acquisition. In practice, this quantity is increased by the need for assistance by the experimentalist during the acquisition.

We generally utilize averaging techniques to improve our signal-to-noise ratio. This effectively multiplies our DAQ time by whatever record redundancy factor we desire to use for averaging (often between 10 and 20). Hence, it is in our interest to gather such nearly identical position/parameter shots as swiftly as possible. Unfortunately, even with efficient programming, LabVIEW was only able to gather a record once every 1.5 s while controlling all potential features simultaneously, failing to match our 1-Hz rep rate, and therefore costing us twice as much time as should be required. Fortunately, many high-quality digitizers (such as our VXI devices) allow on-board averaging once voltage ranges have been specified, averaging continuously at thousands of records per second if so desired. Implementation of this feature allowed us to asymptote to the desired rep rate.

VI. INTERFACING C^{++} TO LabVIEW/AUTOGAIN

Though the G language comprising LabVIEW (National Instruments, http://www.natinst.com) is powerful, there are certain tasks for which it is not well-suited. One of them is the manual writing of data to disk in a precise file format. LabVIEW comes equipped with the ability to make CINs (code interface nodes) wherein the user may link compiled C++ code. For rapid reading and writing of binary data files, these CINs provide the required flexibility. LabVIEW will create the prototype function headers and linking references, easing the insertion of the required C++ code. Besides our reading and writing functions, only one other module required C++: that of the Autogain feature.

The Autogain feature is necessary due to the inherent accuracy of digitizers. The user will generally not know the height of incoming signals. If the range is set too small by the DAQ system, the signal will be clipped. If it is too large, the bit-depth of the digitizer may not properly resolve the waveform of interest. As the probes move through the experimental volume, the signal amplitude can change by orders of magnitude. It is therefore necessary to have an automatic procedure to calculate the maximum signal and center the voltage range upon it.

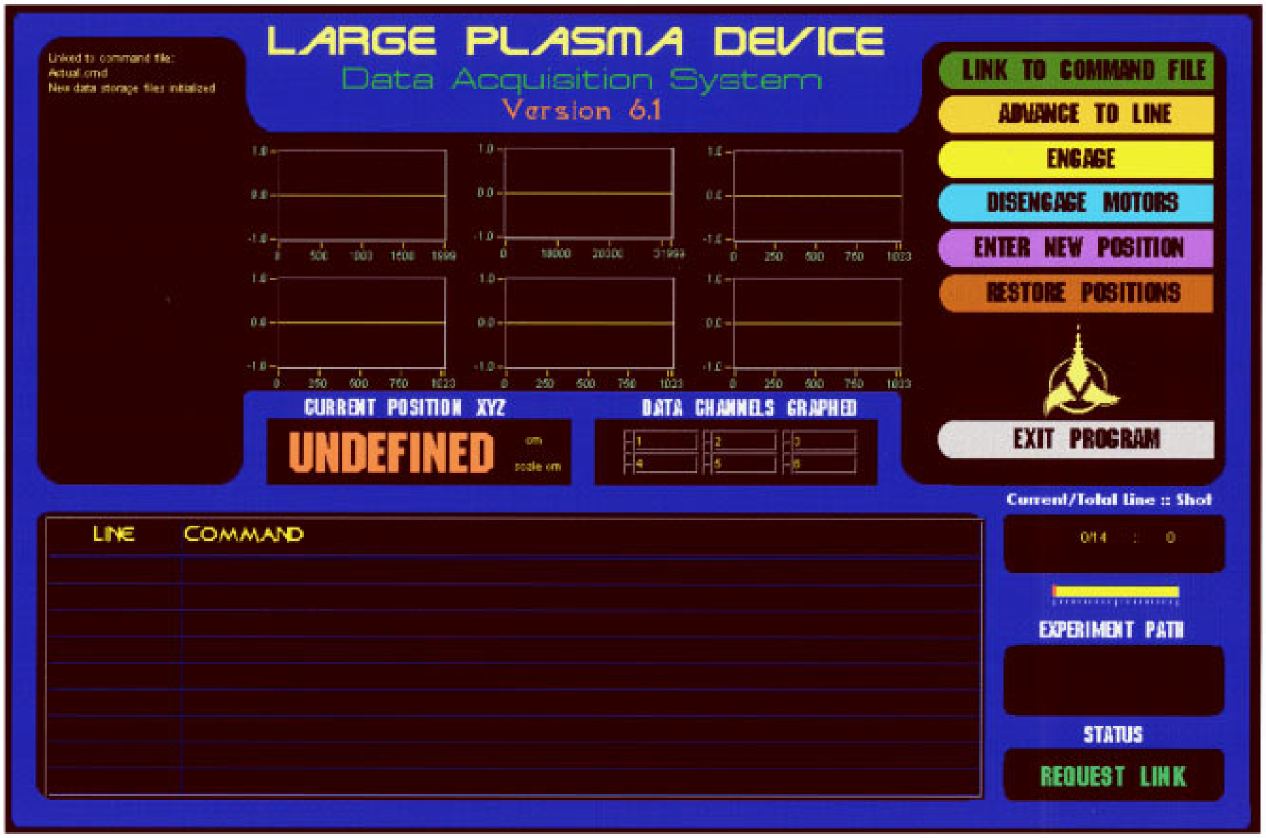
When hardware averaging is used, a sometimes-clipped signal distorts a waveform beyond use. Since our implementation demands use of the rapid hardware averaging in the digitizers, we must be absolutely sure that a signal has been properly bracketed before releasing control to the hardware procedures.

A. Shot-by-shot analysis method

Previous versions of DAQ systems used in this lab employed a method of handling the clipping/average problem that demanded program access to each shot during the averaging process (and is therefore only useful for those systems not intending to use hardware averaging). Once the averaging began, if any record became clipped, the record was thrown out and the range was increased to ''the next highest level.'' This caused the resolution of data to steadily decrease throughout the averaging process if signals fluctuated enough to become clipped, and the user was guaranteed that no clipped data would be included in an average. However, due to the fact that this process requires the computer to talk back to the acquiring device and perform software processing, this scheme never achieved the target 1-Hz rep rate.

B. Careful packing method

The method created for the current DAQ system is designed for noisy data that will be hardware averaged into a usable signal. It is designed to have as little tolerance for



Linked to command file:
Actual.cmd
New data storage files initialized

UNDEFINED

0/14 : 0

REQUEST LINK

Figure 8. DAQ system interface before linking to a Command file. Notice the stepper motor positions currently register UNDEFINED, meaning that the first Establish Origin command will inform the computer where it should assume the motors are. In contrast, the user could enter the current position of the motors, and the DAQ system would then move the motors to the appropriate origin requested by the Command file.

clipping as possible while still optimizing for speed. The first step is to begin at a user-specified maximum voltage range. Next, the function attempts to magnify any visible signal to occupy 75% of the usable range of the device. If at any time the signal's oscillations cause it to occupy more than 80% of the record, the function increases the range to constrain the signal to 75% once again. It then takes several extra shots to ''watch'' the signal and ensure it is not leaving the assigned region of bandwidth. The number of these extra shots is user-determined based on the expected noisiness of their data.

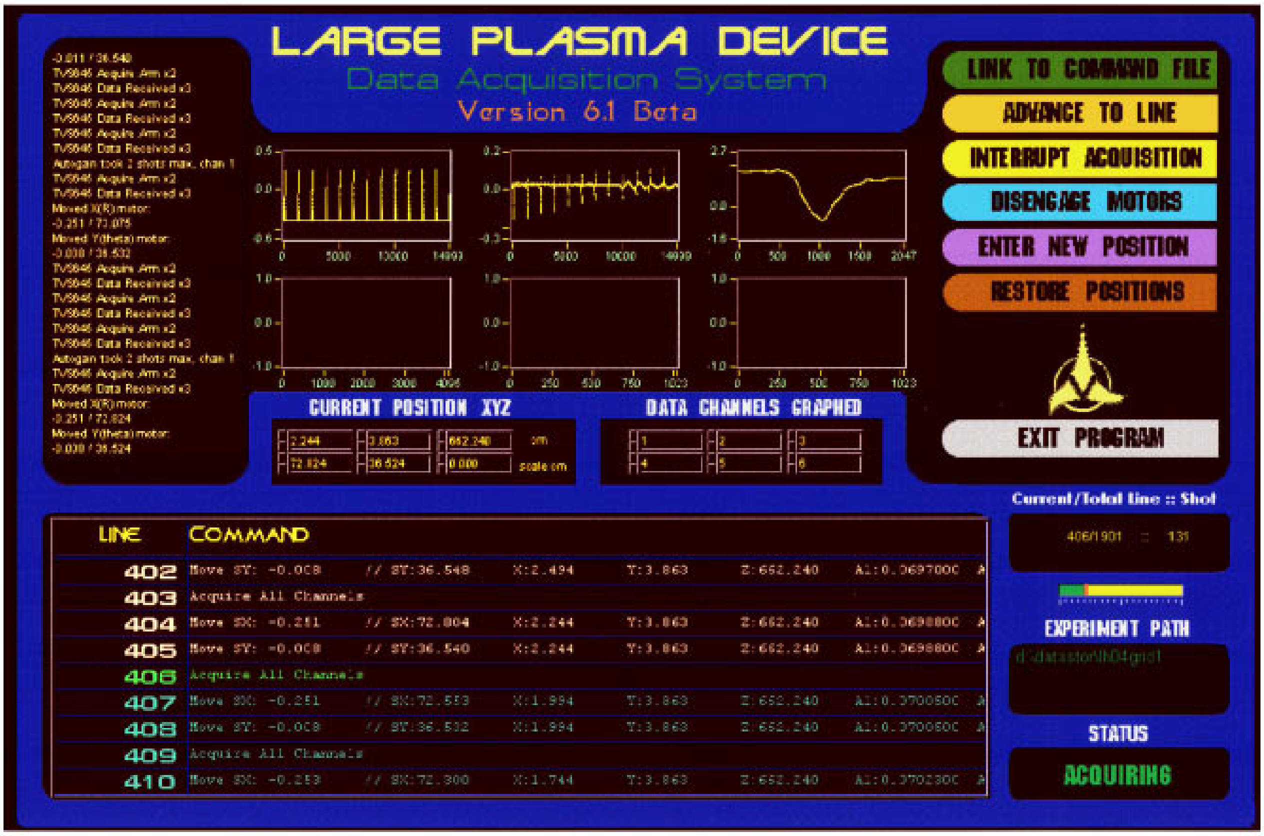
For relatively calm data, the function ''wastes'' two shots, one at the maximum range possible, and the second to confirm the data has been properly bracketed. This is the minimum overhead possible using this implementation. For noisy data, however, the user should specify a number of extra shots of one, two, or even four. If the user is intending to average over 25 shots, the addition of four shots to ensure no clipping is a reasonable price. Due to the extensive use of comparison statements in this function, it was easier to implement in C++. It is interesting to note that introducing the Autogain feature requires a slowing of the DAQ process (since the feature is required to ''examine'' the data beforehand, requiring additional shots that are not stored as data). However, once the Autogain is finished, on-board hardware averaging can be used to match any rep rate desired. Hence, the Autogain feature allows our performance to asymptote towards perfect time efficiency in the limit of large averaging samples, whereas it actually increases data run time in the limit of small averaging samples.

VII. USER INTERFACE

An interface must be both intuitive and pleasing to use, allowing the user to focus on the physics of the situation rather than program configuration and mechanics, especially on long data runs. With LabVIEW's graphical front panels, a single-screen control panel from which all aspects of the DAQ could be managed was created (Fig. 8).

As the data are gathered, it is important to allow the user to view what is being acquired. Since screen real-estate is always at a premium, six graphs of the data were chosen to demonstrate data integrity and correlation. However, the system can function up to 255 channels in the general case. The user can select which channels are displayed on the screen at any time before or during a data run. (See Fig. 9 for a shot of the DAQ system in operation.)

Operational errors are an inevitable eventuality in any DAQ application. The system therefore has an error-detection mechanism that monitors all hardware instruments for local error conditions, as well as observing the time required to take each data point for any anomalous delays. If a single point acquisition has timed out, the pro-



```
-0.011 / 36.548
TVS646 Acquire Arm x2
TVS646 Data Received x3
TVS646 Acquire Arm x2
TVS646 Data Received x3
TVS646 Acquire Arm x2
TVS646 Data Received x3
Autogain took 2 shots max, chan 1
TVS646 Acquire Arm x2
TVS646 Data Received x3
Moved X(R) motor:
-0.251 / 73.075
Moved Y(theta) motor:
-0.038 / 36.532
TVS646 Acquire Arm x2
TVS646 Data Received x3
TVS646 Acquire Arm x2
TVS646 Data Received x3
TVS646 Acquire Arm x2
TVS646 Data Received x3
Autogain took 2 shots max, chan 1
TVS646 Acquire Arm x2
TVS646 Data Received x3
Moved X(R) motor:
-0.251 / 72.824
Moved Y(theta) motor:
-0.038 / 36.524
```

```
2.244      3.863      662.240    cm
72.824     36.524     0.000      scale cm
```

```
1    2    3
4    5    6
```

```
LINE   COMMAND
402    Move SY: -0.008    // SY:36.548   X:2.494   Y:3.863   Z:662.240   A1:0.069700C   A
403    Acquire All Channels
404    Move SX: -0.251    // SX:72.804   X:2.244   Y:3.863   Z:662.240   A1:0.069880C   A
405    Move SY: -0.008    // SY:36.540   X:2.244   Y:3.863   Z:662.240   A1:0.069880C   A
406    Acquire All Channels
407    Move SX: -0.251    // SX:72.553   X:1.994   Y:3.863   Z:652.240   A1:0.070050C   A
408    Move SY: -0.008    // SY:36.532   X:1.994   Y:3.863   Z:652.240   A1:0.070050C   A
409    Acquire All Channels
410    Move SX: -0.253    // SX:72.300   X:1.744   Y:3.863   Z:652.240   A1:0.070230C   A
```

```
406/1901  ::  131
d:\datastor\lh04grid1
ACQUIRING
```

Figure 9. DAQ system interface in operation. The data just acquired are shown in the six yellow graphs in the center of the screen, while their content is controlled in the six numeric entries to their lower right. The Command file currently being executed is shown in the lower half of the screen, showing what commands are coming, being executed, and already complete. A continual post-log (another words list of actions after they have been performed) scrolls in the upper left of the screen. Current line number of the Command file and corresponding shot number is shown to the mid right, while the destination directory of the data and current status of the DAQ system is displayed in the lower right. Finally, the internal and physical stepper motor positions are listed to the left of the data channel controls.

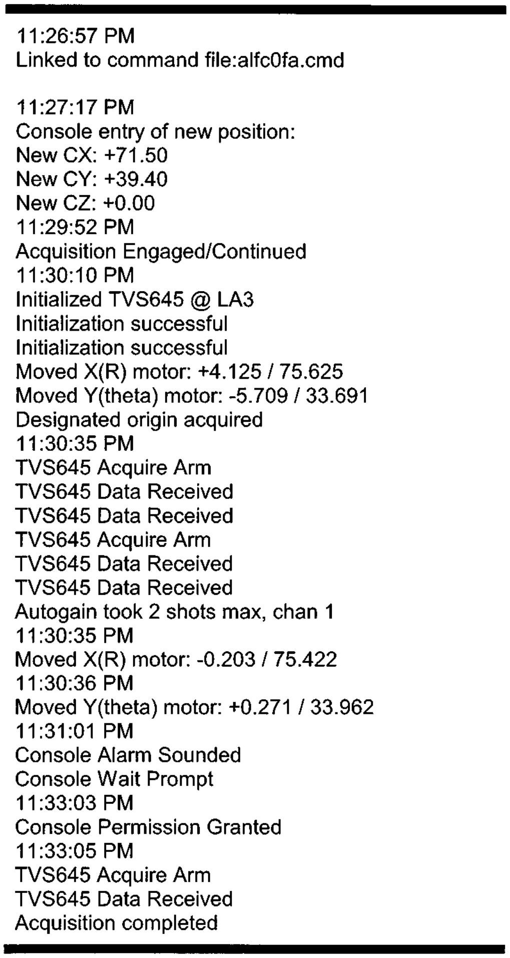
gram warns the user and interrupts acquisition. The program can be reinitiated by use of a Retry button for contentless or quick-fix errors. To ensure that no power loss or other catastrophic failure corrupts the data, files are opened, burst written to disk, and closed immediately so that there is very little chance of corruption. (The program spends $< 0.1\%$ of its time writing to disk.) If the data themselves are corrupted, the Command file may be restarted at any point, overwriting the corruption while preserving intact data. The system also keeps a continuous log file of those actions which have been completed and the rough time of their completion, to expedite failure tracking. (See Fig. 10 for a sample log file.)

The location of stored data is based upon the name of the Command file. Each Command file represents a single data run, and therefore a unique place on the disk for storage of the acquired data. Further, each channel is aware (by user setup) of the kind of physical data being obtained (B probe, E probe, Langmuir probe, etc.). The file extension for the data is chosen based on this physical use, to assist the user in separating out related data. Together, these conventions make it difficult for one user to overwrite another's data.

VIII. DATA FILE FORMAT

Due to the large volume of data this system is designed to acquire (each run can be several gigabytes), the format of the data was an important design consideration to not waste storage space. However, the format must also allow flexible, generalized storage of variable-record size data packets with sufficient information to allow extraction of physically meaningful data. The system utilizes a packet-based storage scheme, each packet associated with a header containing the required information (see Fig. 11). Information contained in the header included a shot number, which allowed the program to keep track of which data packets came from which experimental trigger (important for overwriting corrupted data at a later date), the length of the packet, the channel of origin of the data, the stepper motor encoding information, and the position and orientation of the probe.

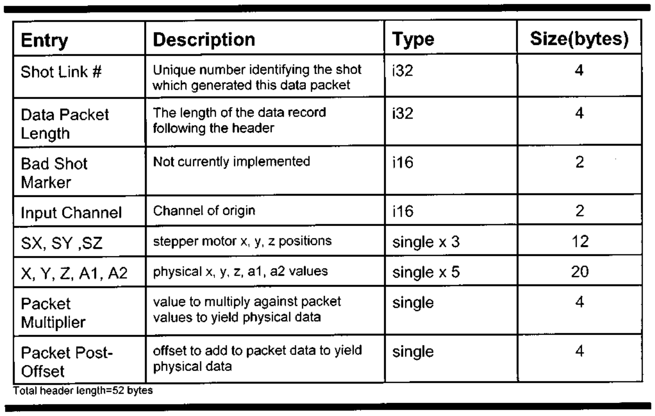
Two other entries in the packet header correspond to a resolution-optimization scheme. The data in each packet are stored as $i16$ values. Before storage, the DAQ system calculates the offset and gain required to center and scale thedata into the full $i16$ range. This gain and offset is stored



```
11:26:57 PM
Linked to command file:alfc0fa.cmd

11:27:17 PM
Console entry of new position:
New CX: +71.50
New CY: +39.40
New CZ: +0.00
11:29:52 PM
Acquisition Engaged/Continued
11:30:10 PM
Initialized TVS645 @ LA3
Initialization successful
Initialization successful
Moved X(R) motor: +4.125 / 75.625
Moved Y(theta) motor: -5.709 / 33.691
Designated origin acquired
11:30:35 PM
TVS645 Acquire Arm
TVS645 Data Received
TVS645 Data Received
TVS645 Acquire Arm
TVS645 Data Received
TVS645 Data Received
Autogain took 2 shots max, chan 1
11:30:35 PM
Moved X(R) motor: -0.203 / 75.422
11:30:36 PM
Moved Y(theta) motor: +0.271 / 33.962
11:31:01 PM
Console Alarm Sounded
Console Wait Prompt
11:33:03 PM
Console Permission Granted
11:33:05 PM
TVS645 Acquire Arm
TVS645 Data Received
Acquisition completed
```

Figure 10. Sample Log file, describing those actions which have been performed and any messages with which they are associated. They are chiefly used as a debugging tool.



Entry	Description	Type	Size(bytes)
Shot Link #	Unique number identifying the shot which generated this data packet	i32	4
Data Packet Length	The length of the data record following the header	i32	4
Bad Shot Marker	Not currently implemented	i16	2
Input Channel	Channel of origin	i16	2
SX, SY ,SZ	stepper motor x, y, z positions	single x 3	12
X, Y, Z, A1, A2	physical x, y, z, a1, a2 values	single x 5	20
Packet Multiplier	value to multiply against packet values to yield physical data	single	4
Packet Post-Offset	offset to add to packet data to yield physical data	single	4

Total header length=52 bytes

Figure 11. The format used to store a universal data packet header. Each packet is completely independent, possessing all information necessary to decompose the data record into usable physical data. Since our data records tend to be between 1024 and 15,000 characters long, this header implementation represents between 5% and 0.3% "wasted" space.

in the packet header so that the transform can later be undone to yield the original values. This method assures all data will appear to have 16 bits of resolution regardless of the device of acquisition used.

IX. WARNINGS AND TROUBLE SPOTS

Initially, our system took over a year to complete, due to extreme difficulty obtaining working drivers for the VXI crate devices. We initially began with a Sun station; however, the Tektronix drivers were unable to even communicate at a low level with our devices. After months of technical nonsupport, we decided to move to the PC (where the drivers are originally developed). The modern Windows 95 environment was greatly beneficial to development time, and functional drivers did seem available. Unfortunately, they were 16-bit drivers developed for the old Windows 3.1 system. They worked partially under Windows 95 for a few months, allowing us to lay down the basic design and functionality of the DAQ system; however, when we began to push the abilities of the Tektronix devices, once again the drivers began to fail. Finally, one year after buying our digitizers, 32-bit Windows 95 drivers for the Tektronix devices became available. Since that time, we have observed no other problems. We estimate that the development time with working drivers to have taken approximately four months.

When purchasing VXI devices for use with LabVIEW (or other similar system), make sure that the drivers exist before purchase, are bug-free, and finally are the modern 32-bit standard. The excellence of a manufacturer's hardware does not imply a one-to-one correspondence to that of their software.

The DAQ system designer should also be aware that there is a companion product, LabWindows CVI (National Instruments, http://www.natinst.com), which allows the graphical user interface design of LabVIEW but supported by entirely C++ coding in a way reminiscent of Microsoft Visual C++. This powerful fusion of the two techniques is a worthy prospect for investigation, and for systems with additional complication above the one presented here it is very possibly more applicable.

X. SUMMARY

The current PC computer market offers both hardware and software with sufficient power and sophistication to create (for the first time) data-acquisition systems appropriate for the scientific community. This change results in the fact that the most expensive items of a DAQ system are now the digitizers, not the computer. Herein was listed one suggested configuration, but uncountable others exist with some research and effort.

Previously, data-acquisition systems required extensive coding effort on a very low level. However, more modern products such as LabVIEW, LabWindows CVI, HP VEE, and other graphic-based developing environments offer an excellent alternative. Today, high-quality user interfaces can be inexpensively assembled, sparing more time and effort for the actual DAQ system decisions. Further, the advanced acquisition devices currently available are now designed first with PCs in mind and ported to other platforms at a later date. This transition to a user-friendly, low-cost medium represents a phase transition in DAQ design and complexity.

In the end, LabVIEW was chosen in an attempt to reduce production time and ensure ease of maintenance by future programmers. Despite the graphical complication that resulted in such a large-scale program, the strengths of LabVIEW were still sufficient to justify its use. Most of the negative aspects of LabVIEW outlined above can be overcome given time and practice; however, those interested in crafting a DAQ system from scratch should heed the warning that the LabVIEW programming environment is quite different from traditional text-based languages.

However, in the midst of these opportunities, some serious considerations must be kept in mind. Interfacing the desired hardware to the desired software is in general possible but has varying degrees of difficulty. Careful discussions with engineers at the respective companies of interest should be had before any purchases are made via overenthusiastic salespeople.

ACKNOWLEDGMENTS

This work was supported by the Office of Naval Research. The authors would also like to thank National Instruments for their extensive technical assistance and professionalism.

REFERENCES

1. W.N. Gekelman, Appl. Phys. **4**, 463 (1992).
2. W. Gekelman and L. Xu, Rev. Sci. Instrum. **57**, 1851 (1986).
3. W. Gekelman, H. Pfister, Z. Lucky, J. Bamber, D. Leneman, and J. Maggs, Rev. Sci. Instrum. **62**, 2875 (1991).