Software for physical modelling survey design and acquisition

Henry C. Bland and Paul R. MacDonald

INTRODUCTION

The elastic modelling system simulates real-world seismic surveys in the laboratory. Models are typically constructed using materials of differing acoustic velocities (Plexiglas, aluminium). Two transducers are moved about the surface of the model by motorized arms running under computer control. One transducer operates as a transmitter and the other transducer operates as a receiver. The received data trace is recorded and stored on the computer in a manner that is identical to conventional seismic recording.

The physical modelling system has evolved over a period of ten years. During this time, its hardware has received incremental upgrades, but little work has gone into software. During the summer of 1999, the authors developed new software with the goal of improving the reliability of the system, adding a number of long-needed features, and making the program expandable and maintainable. The culmination of these efforts is a program for physical model survey design and acquisition called "PUMA"[1].

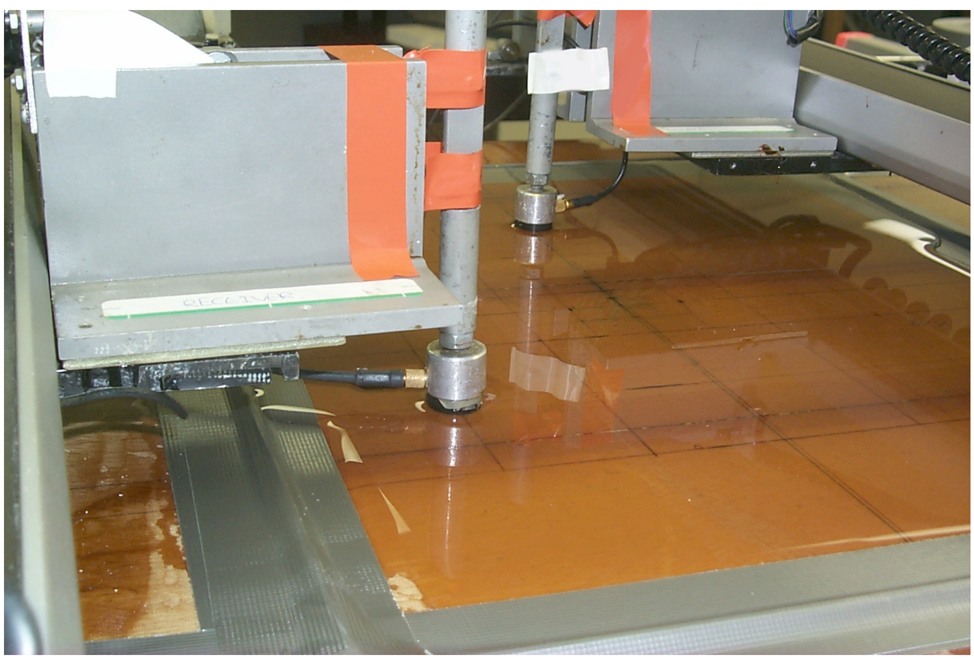


Figure 1. Transducers transmit and receive ultrasonic signals through a physical model during a simulated 3-D seismic survey.

[1] The name "PUMA" is purportedly an acronym for "Physical Ultrasonic Modelling Acquisition".

Figure 2.The physical modelling system consists of the modelling jig (left), the computer and electronics systems (right). The configuration above shows a small water tank inserted into the jig aperture to simulate marine surveys.

SYSTEM OVERVIEW

The physical modelling system is comprised of several components shown schematically in Figure 3. A piezoelectric source transducer is excited by a digitally generated waveform. The ultrasonic signal emitted from the source transducer travels through the model, and reaches the receiving transducer. The receiving transducer generates a very small electrical signal, which is digitised, processed (stacked and filtered) and stored to disk. For each shot location and receiver location, the controlling system must manipulate the modelling system to move the transducers into new positions. A series of switches sense the position of the transducer actuating mechanism, allowing its location to be determined at various stages of a survey acquisition. A joystick allows an operator to interactively move the transducers to calibrated locations prior starting a physical modelling survey.

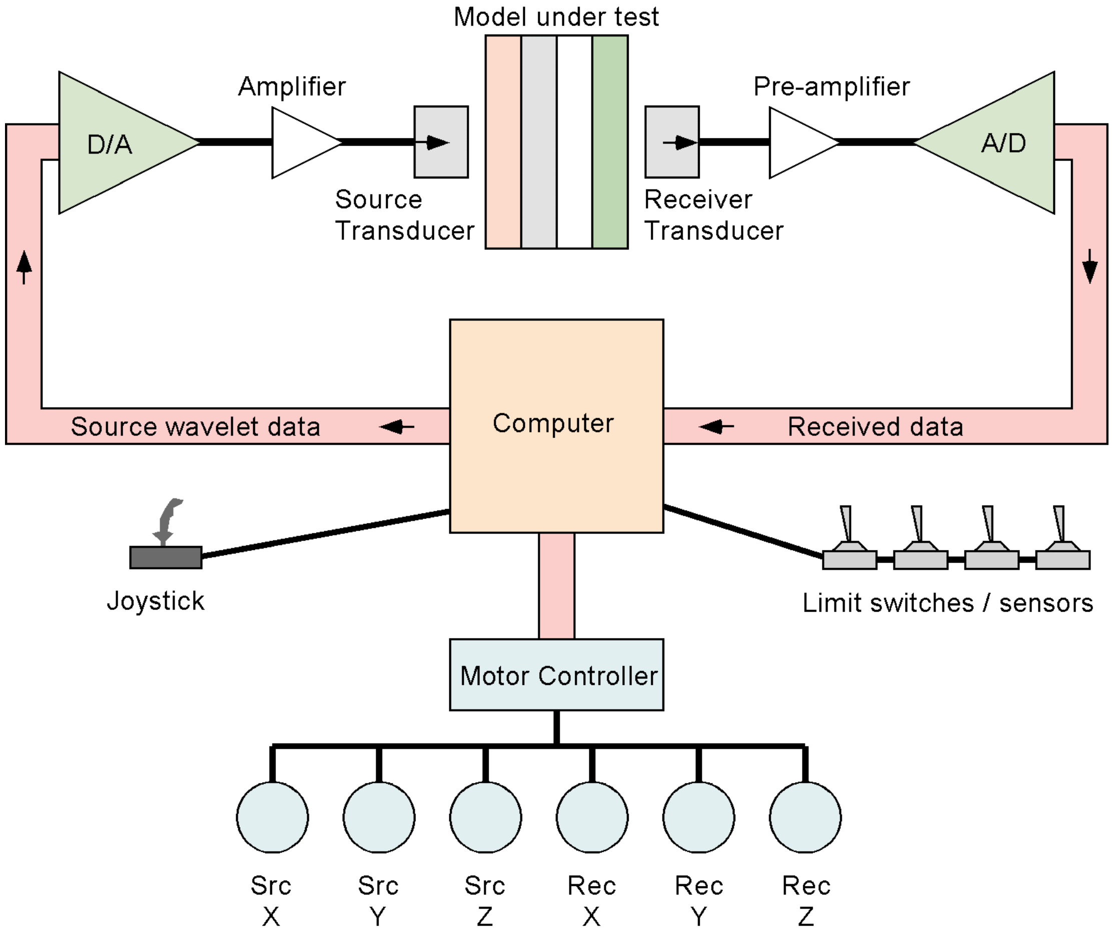


Figure 3. Schematic of the physical modelling.

Survey design

One simplifying feature of physical modelling surveys over real-world surveys is that surveys can be designed without concern for land access, or surface features. As a result, it is relatively easy to generate 2-D and 3-D surveys. Physical modelling survey design and survey shooting are often performed in iterations – changing the survey parameters slightly and observing the recorded data moments later. For this reason, it made sense to integrate the survey design program directly into the acquisition program.

The modelling systems is designed to perform both 2-D and 3-D surveys. Past experience tells us that fixed offset surveys are of great utility, as a model can be quickly surveyed and (without processing) structural features identified. With this in mind, PUMA was written to generate 2-D and 3-D surveys with fixed offset geometries as well as the more conventional variable-offset geometries. Figures 4 and 5 show examples of the user interface for survey design. The output from the survey design module is a text file in a format reminiscent of SEG-P1 format (true SEG-P1 export is an available feature). Even though the survey design module is built-into the program, there is no tie between the survey design component and the acquisition component of PUMA. As a result, surveys can be imported from external survey design applications should the need arise.

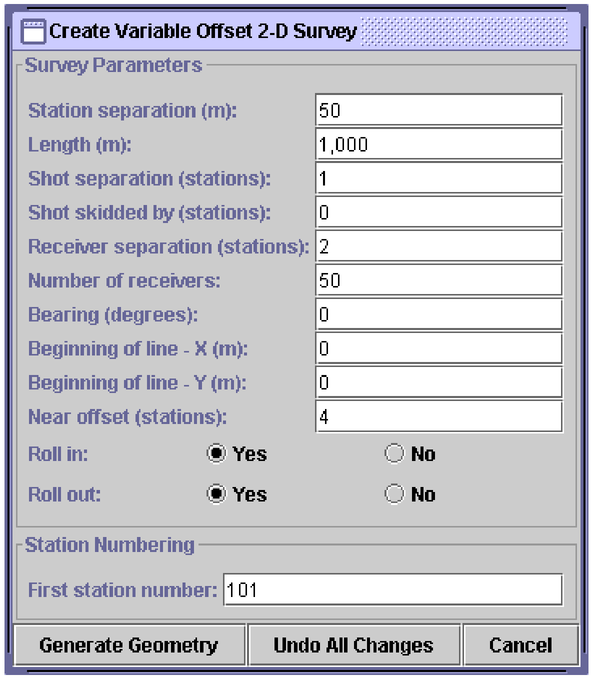


Figure 4. Survey design dialog for variable offset 2-D surveys.

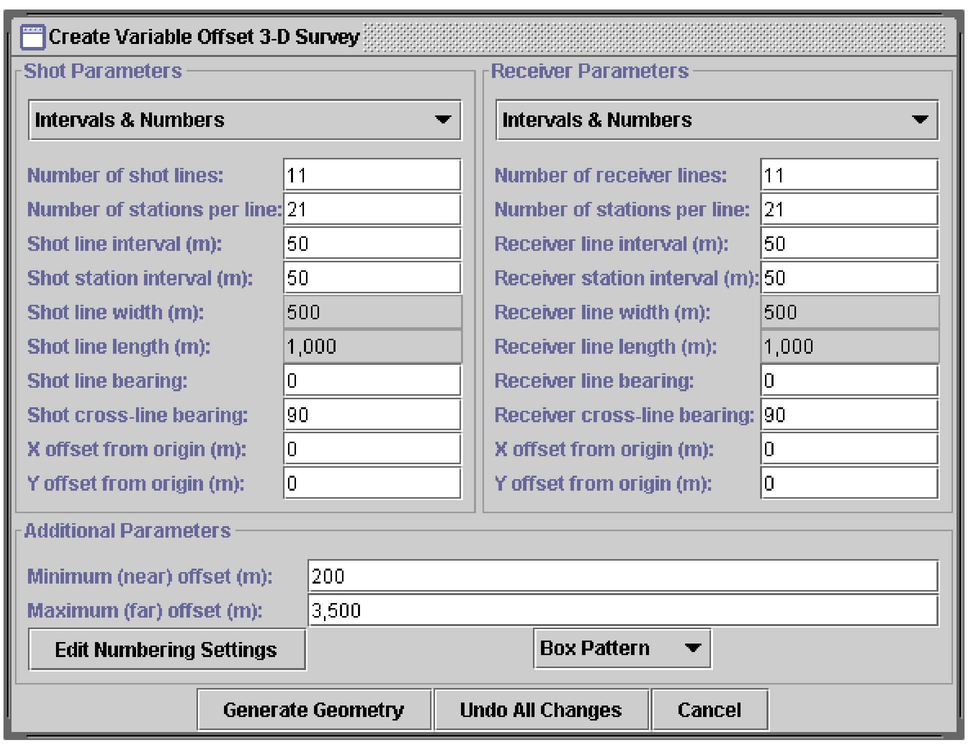


Figure 5. Survey design dialog for variable offset 3-D surveys.

Survey acquisition

The primary job for the software is to interact with all the different subsystems that perform the survey. Table 1 details the different tasks that the software must perform during acquisition.

Table 1. Tasks performed by the acquisition system during survey acquisition.

Emit source wavelet
Acquire received trace
Filter and stack received traces
Display received data on console
Save stacked traces to disk
Determine next X, Y, and Z position for source and receiver
Instruct motor controller to move motors
Monitor motor controller for progress and status
Obtain motion feedback from switches and sensors
Provide a status overview for operator

Some tasks, such as data display, stacking, and data saving, are limited in speed by the computer (CPU and disk speed). Other tasks, such as moving motors and acquiring data, are limited in speed by external hardware. Most importantly, many of the tasks have prerequisites: correctly sequencing the tasks is always of utmost importance. PUMA optimises the overall rate of acquisition by performing CPU intensive tasks while waiting for external events to take place (e.g. stacking traces while waiting for the transducers to be placed into position).

While PUMA is performing a survey, received data must be stacked, displayed and stored. Since random noise is a large component of the received data, stacking greatly improves signal quality. Unlike real-world acquisition, there is no cost associated with repeating source points several times. The time-overhead for high-fold stacking is negligible, as several hundred sources can be acquired within a second. We have traditionally stacked 32 times at each source point, but may increase this substantially now that the software can acquire and stack more quickly. After acquiring and stacking traces, the traces are displayed on the screen for the operator to view. Real-time display is important, as the operator must be able to see problems such as poor transducer coupling, bad connections, and inadequate signal levels as they arise. At the same time as data is displayed, it is stored to disk in SEG-Y format.

Diagnostics

One of the main flaws of the old physical modelling program was that it did not include adequate diagnostic facilities. One very important goal of this project was to provide a diagnostic module for each subsystem: the source wavelet generator, the data acquisition system, the motor controller, and joystick. The source waveform generator diagnostic module allows direct access to the waveform generator controls. It lets one generate waveforms from a pre-set list (sine waves, square waves, triangle waves) or from manual entry of sample values. The data acquisition diagnostic module displays the raw data received from the acquisition system in real-time using an oscilloscope-like display, and the motor controller diagnostic allows one to operate any of the six positioning motors directly. Finally, a joystick / motor control system diagnostic module allows one to interactively drive the transducers using the joystick for control. These diagnostics are crucial to the system, as the system is complex, and tricky to trouble-shoot without the aid of specifically designed software.

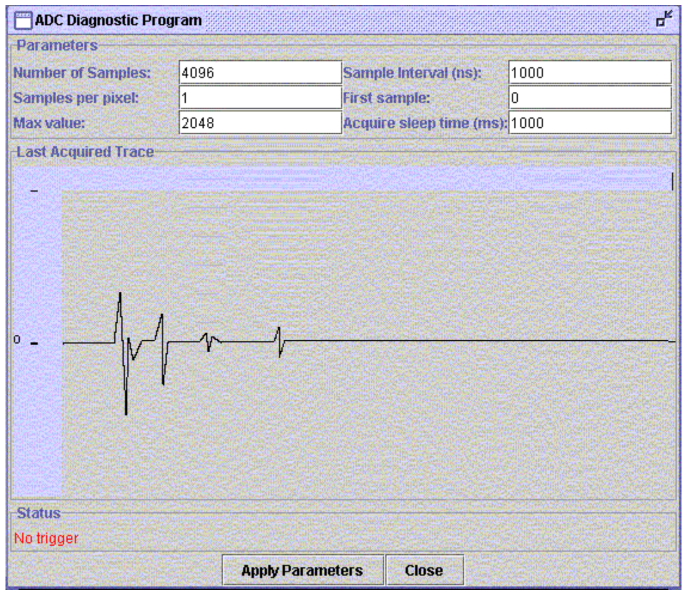


Figure 6. Diagnostic module for the data acquisition subsystem.

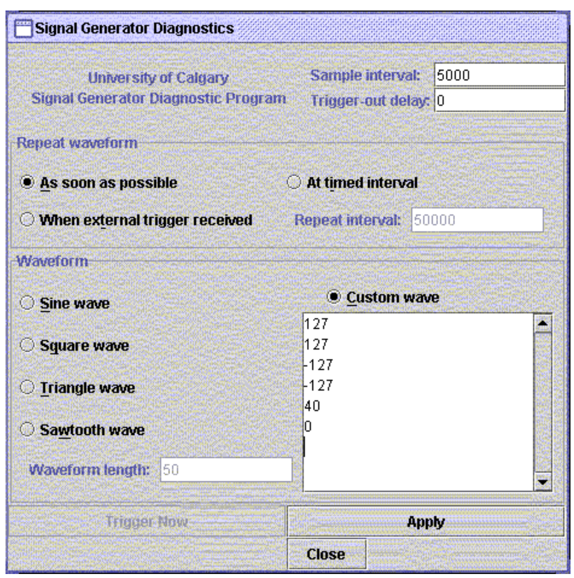


Figure 7. Diagnostic module for the source waveform generator.

IMPLEMENTATION

The old version of the physical modelling control software was written in Microsoft QuickBasic running under the MS-DOS operating system. This software was plagued with problems resulting from memory limitations imposed by MS-DOS, the lack of a graphical user interface, slow speed, and antiquated programming models. In implementing PUMA, it was decided to use up-to-date programming techniques and languages. Java was chosen because of its maintainability, rich library of functions for graphics, it's user interface library, and its inherent support for multithreaded coding. A three-layer structure was used in implementing the system. The application is written primarily in Java, while access to hardware is obtained through a middle layer written in C++. The middle layer provides a Java interface to the Windows device drivers. It was necessary to write some device driver code to allow access to custom hardware from within Microsoft Windows. This model effectively isolates the application-layer code from the harder-to-maintain low-level code.

Table 2. Implementation of a multi-layered programming model.

Layer	**Language**	**Amount of code**
PUMA Application	Java	98%
Java native interface to Windows hardware	C++	1%
Windows device driver	C++	1%

Multithreading

As mentioned earlier, PUMA simultaneously handles a large number of tasks while acquiring a survey. Since many of these tasks contain delays based on external hardware, CPU usage, and disk usage, optimum scheduling of the tasks is crucial in order acquire data quickly. Execution speed is very important, since 3-D surveys can contain millions of traces, and each trace must be acquired one-geophone-at-a-time. Rather than explicitly code the task scheduling, we rely on Java's multithreading capabilities to schedule the tasks for us. Each task is written in it's own thread, and the competition for CPU time automatically sequences the threads in the optimal fashion. Threads that wait for external events (such as a thread waiting for a motor to reach its destination) are automatically suspended, while other CPU-intensive threads get full-attention. Order-of-operation issues between threads are handled using Java's built-in object locking mechanisms. Any thread that waits on another thread is automatically removed from the execution queue. As a result, only those threads with work to do will be executed. An additional thread takes care of user-interface operations, allowing use of the user-interface (including survey design) while a survey is being carried-out. Although there is some overhead in non-explicit coding of task scheduling, the code is greatly simplified using a multithreaded approach.

RESULTS

The entire project, from design to working prototype took less than five person-months to complete. This initial version of PUMA is now operational, and some refinements are now underway. Specifically, PUMA is being modified to work with a wider-bandwidth data acquisition subsystem. PUMA's interaction with position sensing switches is being improved so that the system can confirm that the actuating arms are correctly located during any portion in an acquisition.

One of the greatest concerns when implementing the system was that Java wouldn't have the efficiency to operate the system with enough speed. Java has a reputation of being a slow language/environment because it is not compiled into machine code. Instead, java is compiled into a portable "byte code" which is then interpreted by a second program (the Java Virtual Machine). We were pleasantly surprised by the speed at which the program runs – the speed of execution is not limited by the speed of Java code, but by the rate at which the motors operate. Even more surprising is that the seismic traces can be drawn at an acceptably speedy rate, and scrolling through seismic is as fast as any other C-coded program. Indeed the slowest section of code is not the data acquisition, stacking or display, but the disk output section. In order to create truly "standard" SEG-Y data files, data are written to disk in non-native byte order (as required by the standard) using a very inefficient (but portable) technique. This technique uses one "disk write" operation for every four bytes of data. If execution speed ever becomes an issue, we may consider re-writing this section of code for efficiency (possibly at the cost of portability).

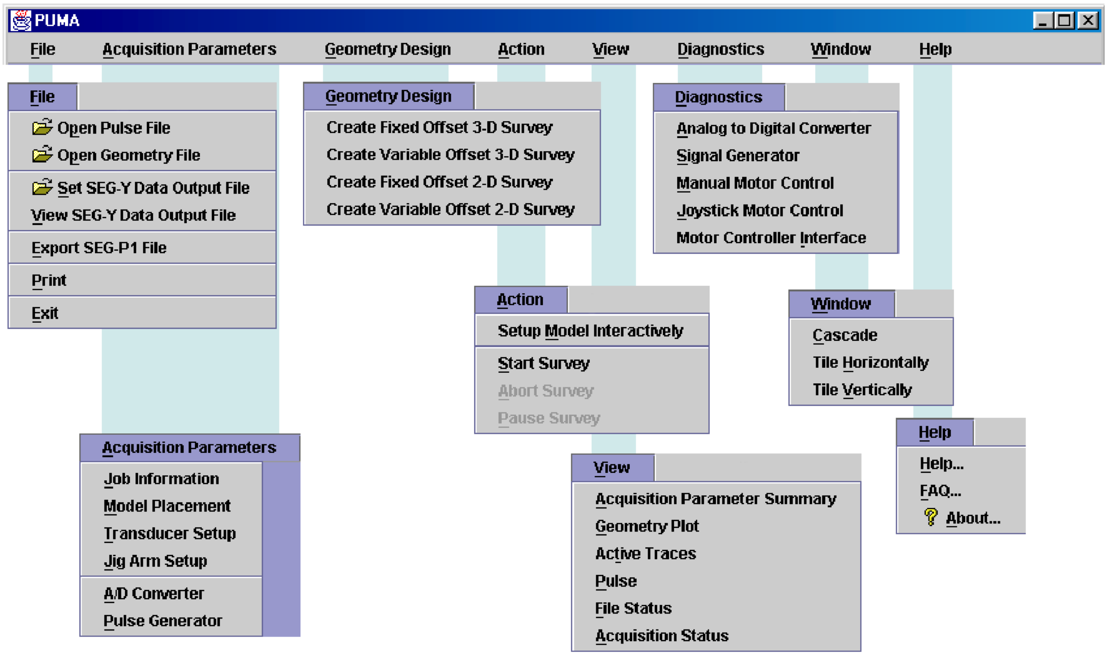
CONCLUSION

The physical modelling software was been successfully replaced. The new software supports a variety of new geometries, the ability to handle arbitrary geometries, better acquisition quality control, and significantly easier operation. This project was the first of its kind (within CREWES) to be written in Java. We are happy with the results, both in terms of execution speed, and development time. It is our hope that our choice of language will make this code expandable and maintainable for the foreseeable future.

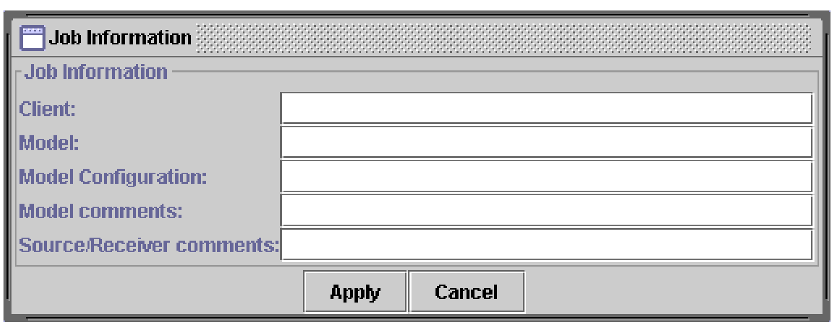
APPENDIX A.

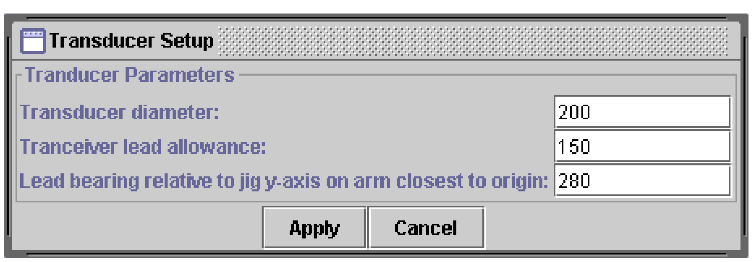
To best understand the features of PUMA, the menus and dialogs are presented in this appendix.

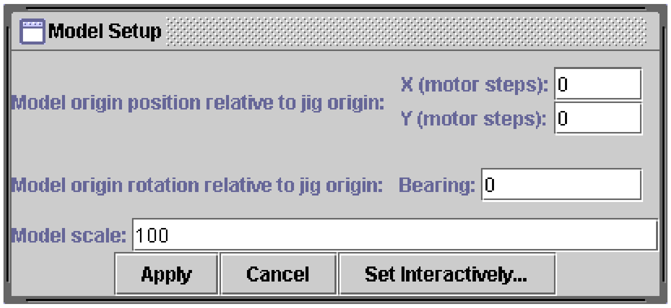
MENU STRUCTURE

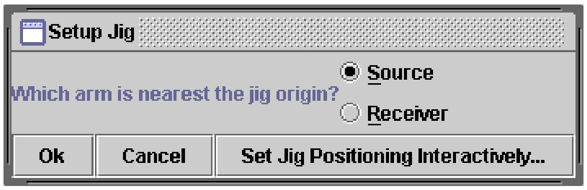


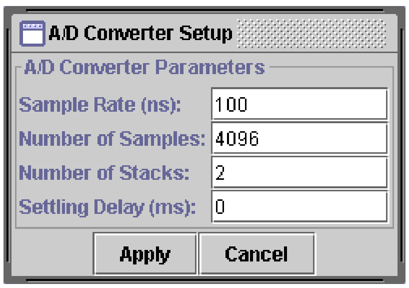
ACQUISITION PARAMETER DIALOGS






Model Setup
X (motor steps): 0
Model origin position relative to jig origin:
Y (motor steps): 0
Model origin rotation relative to jig origin: Bearing: 0
Model scale: 100
Apply
Cancel
Set Interactively...


Setup Jig
Source
Which arm is nearest the jig origin?
Receiver
Ok
Cancel
Set Jig Positioning Interactively...


A/D Converter Setup
A/D Converter Parameters
Sample Rate (ns): 100
Number of Samples: 4096
Number of Stacks: 2
Settling Delay (ms): 0
Apply
Cancel