

Seismic System Modeling

Brad Thoen
MTS Systems Corporation

What Models Are Provided?

- Forward (valve to sensor) model
- Inverse (sensor to valve) model
- Control system analysis tools
- Realtime “model-in-the-loop”

What Effects are Modeled?

- Servovalve and actuator dynamics, including:
 - Pressure switching
 - Nonlinear flow gain effects
 - Servovalve spool overlap/underlap
 - Variable volume effects with piston stroke
 - Kinematic and compressibility flows
 - Cross-piston leakage
 - Friction
- Actuator geometry cross-coupling
- Rigid body dynamics in 6 DOFs
- One specimen resonant mode
- Accumulator bank pressure change with flow

What Effects are *NOT* Modeled?

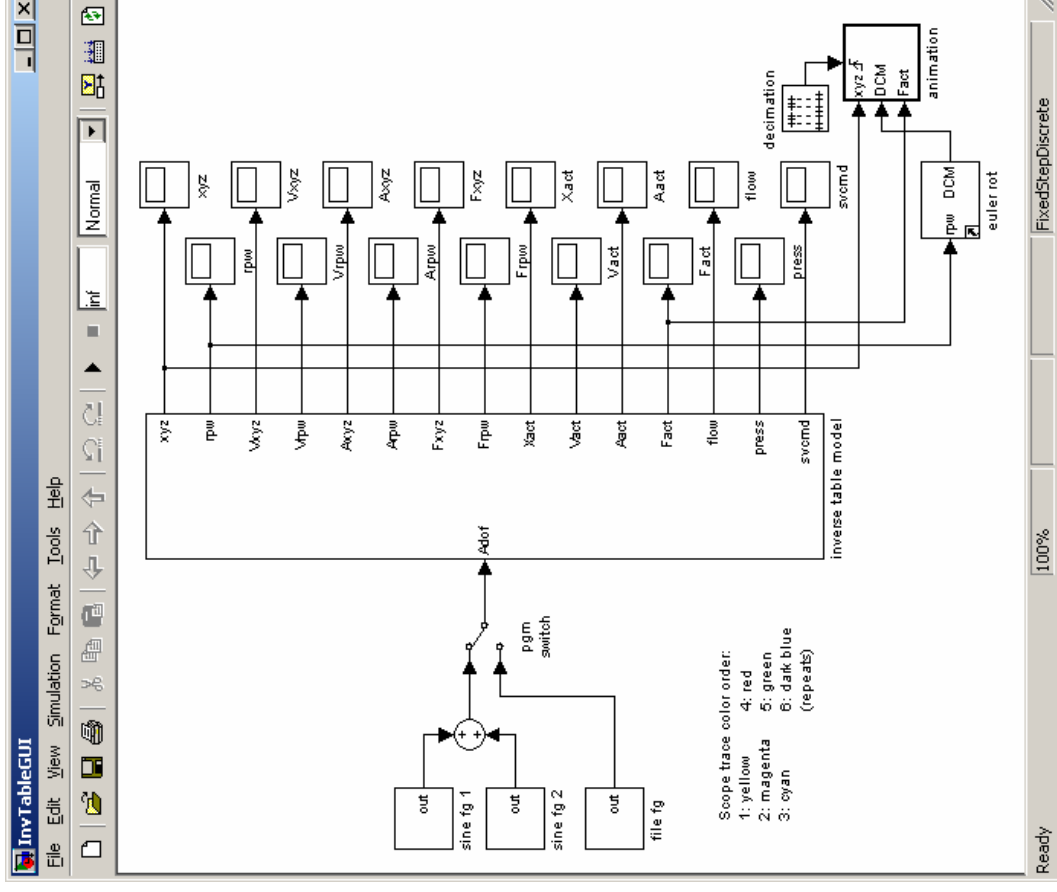
- Higher table, specimen, and foundation resonant modes
- Specimen nonlinearity (highly application specific)
- Hydraulic distribution losses
- Physical limiting, except inside servovalve & actuator

It is expected that the customer will add additional dynamic effects to the model as needed.

Inverse Model

Given a desired acceleration profile, compute servovalve opening (and displacement, velocity, force, and flow).

Useful for predicting whether test will exceed physical capacity.



[illegible]

Forward Model

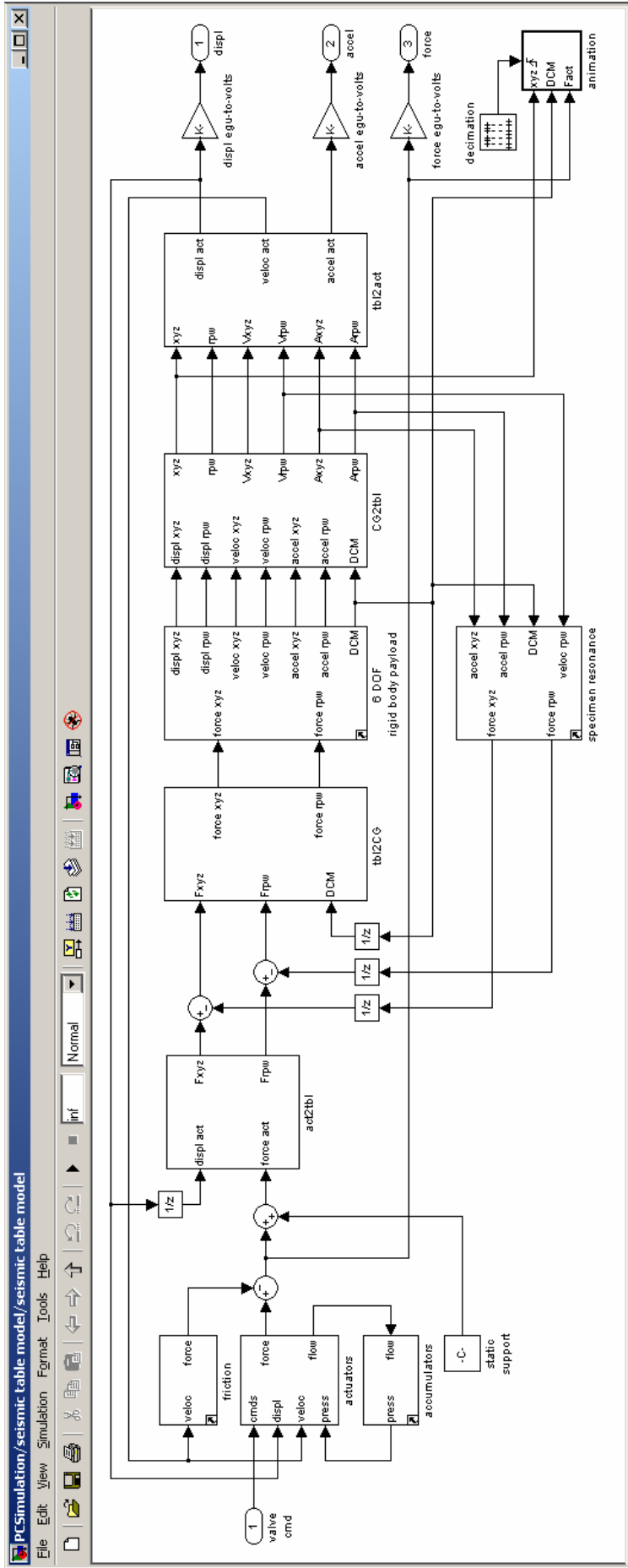
Given a desired servovalve opening, compute displacement, acceleration, and force.

Useful for:

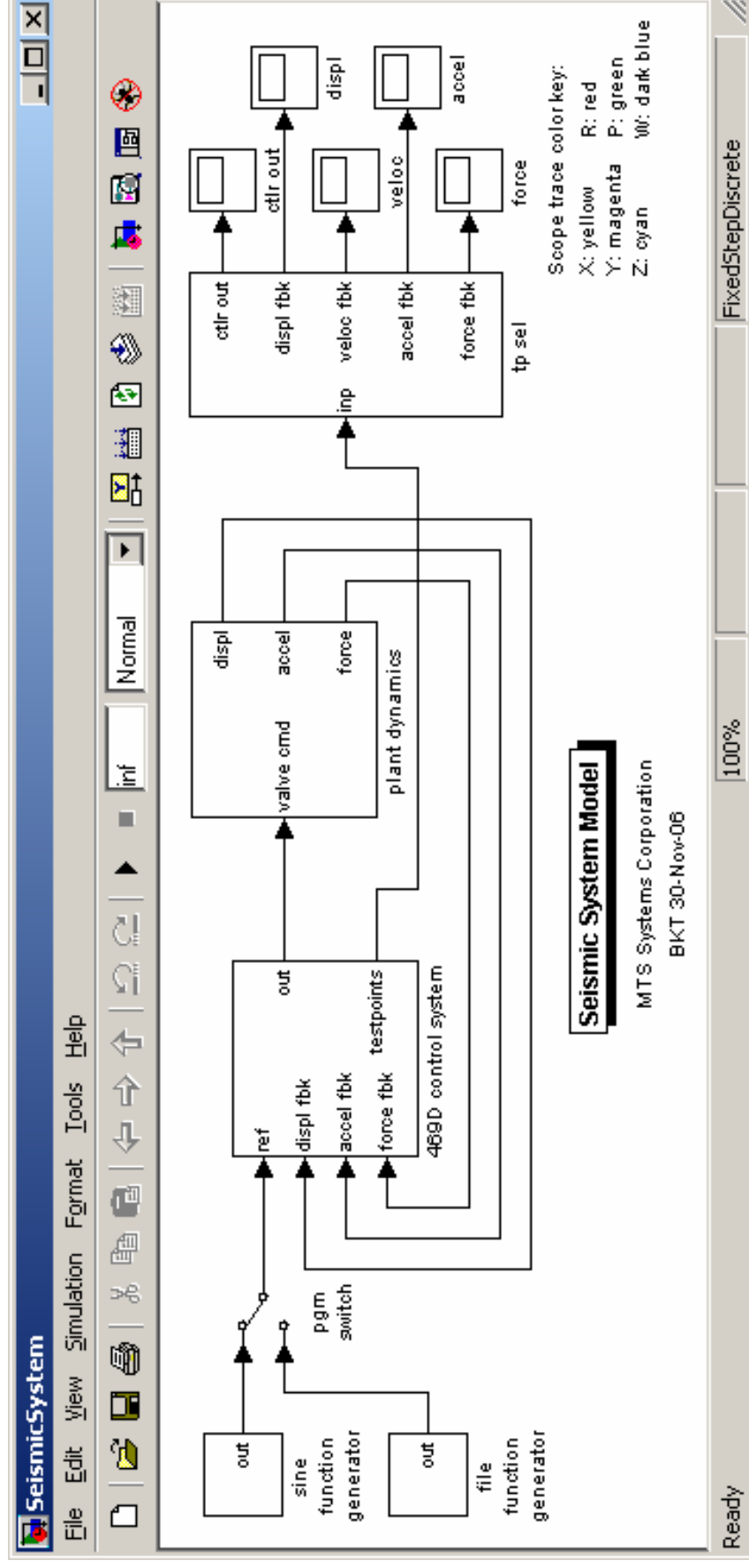
- Operator training
 - Predicting closed-loop control fidelity
 - Test rehearsal
 - Offline tuning
- Note: Forward model must be driven by a controller.

- Predicting closed-loop control fidelity
- Test rehearsal

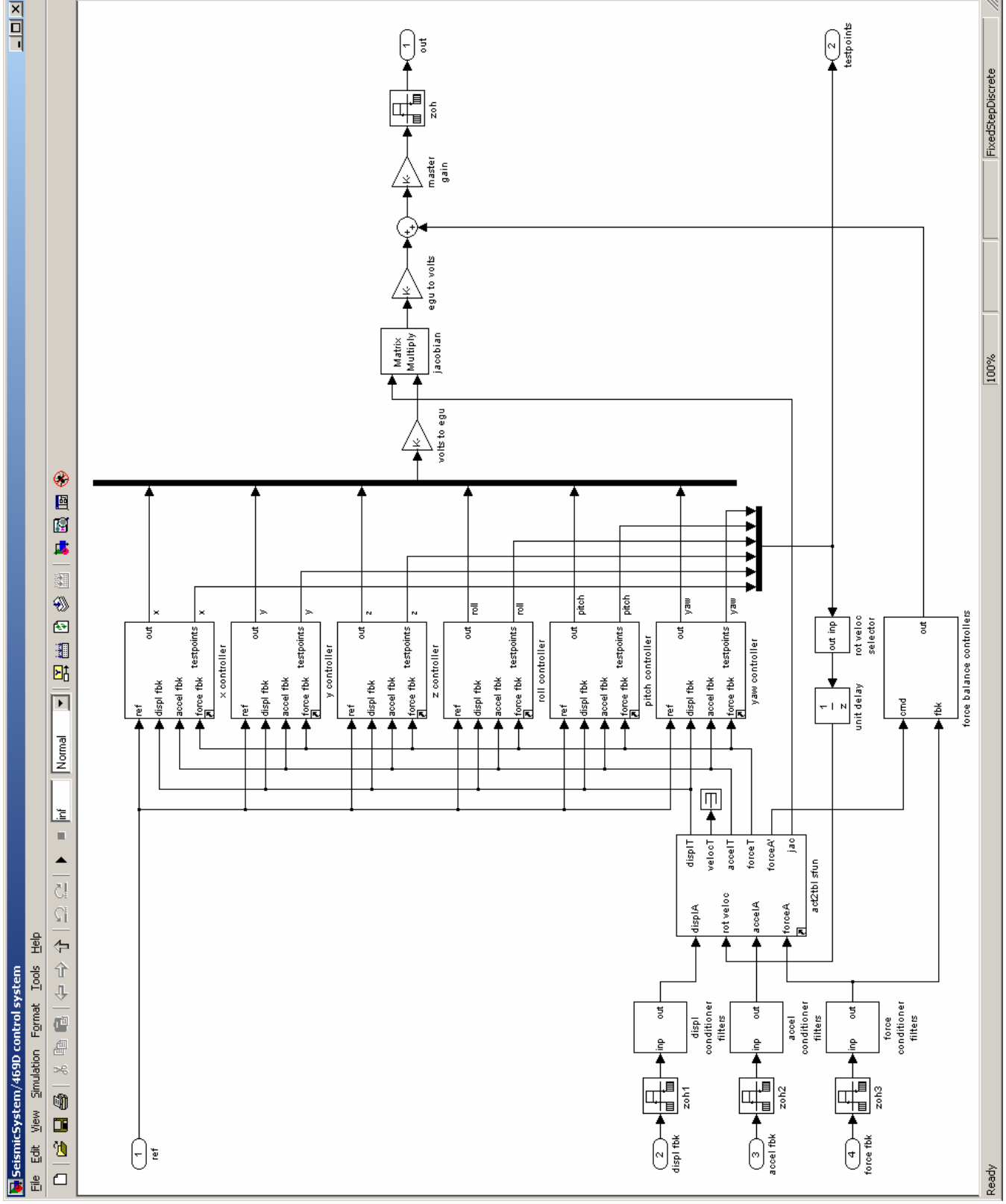
Note: Forward model must be driven by a controller.



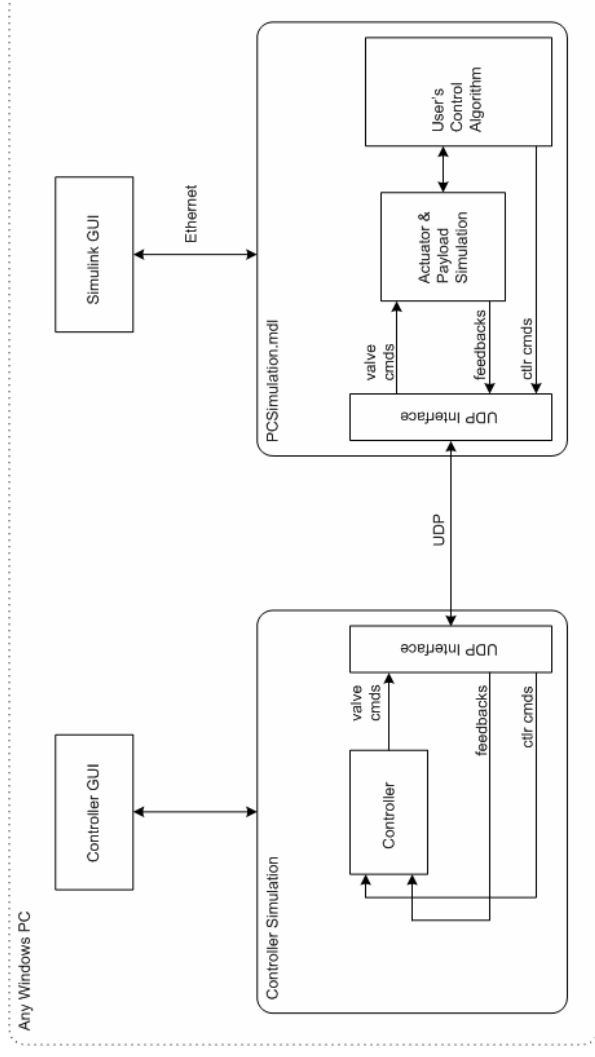
Forward Model Driven by Simulink Controller



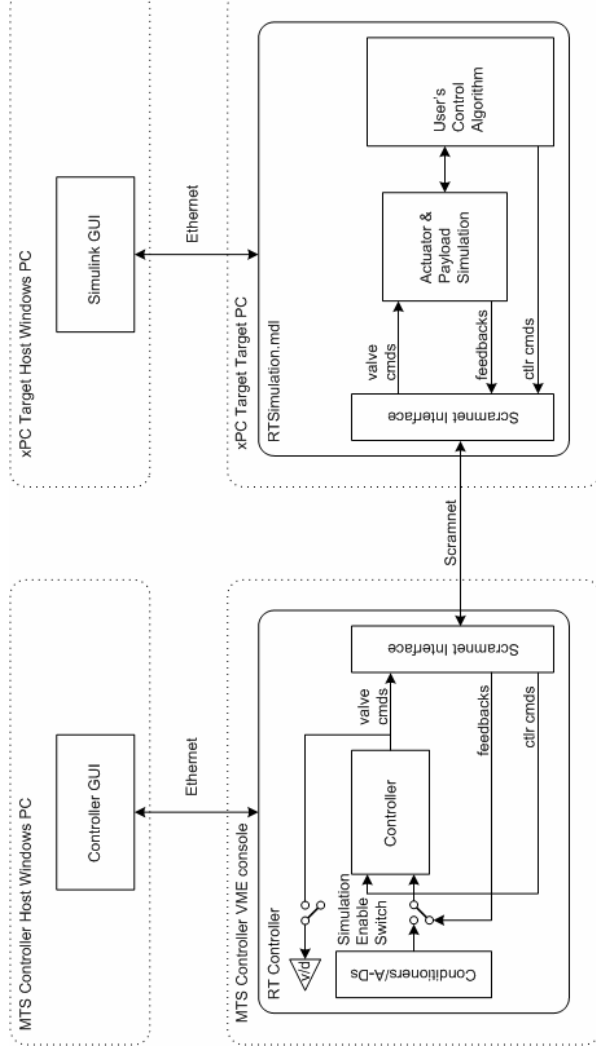
Forward Model Driven by Simulink Controller (cont.)



Forward Model Driven by Actual Controller

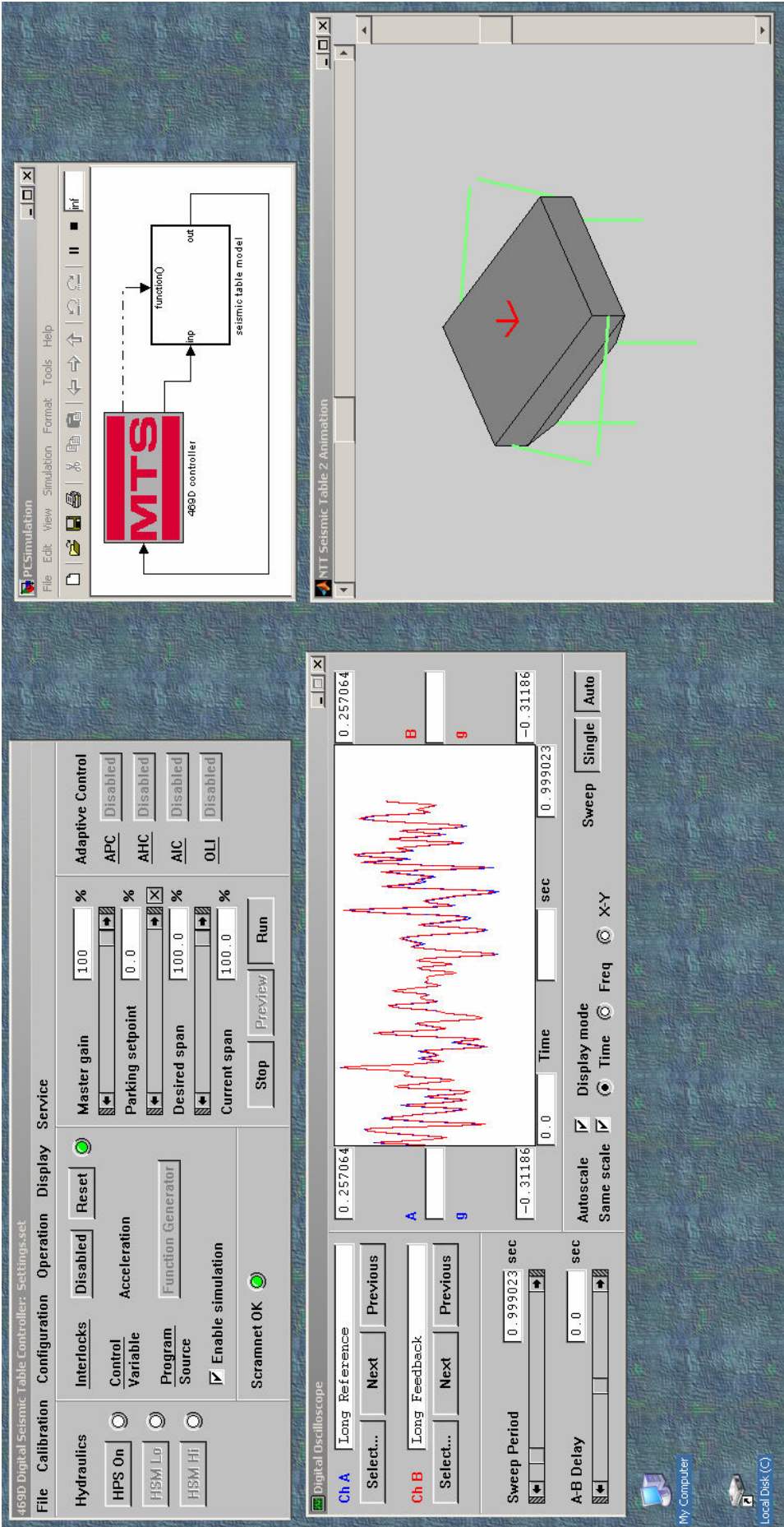


in non-realtime
without controller
hardware

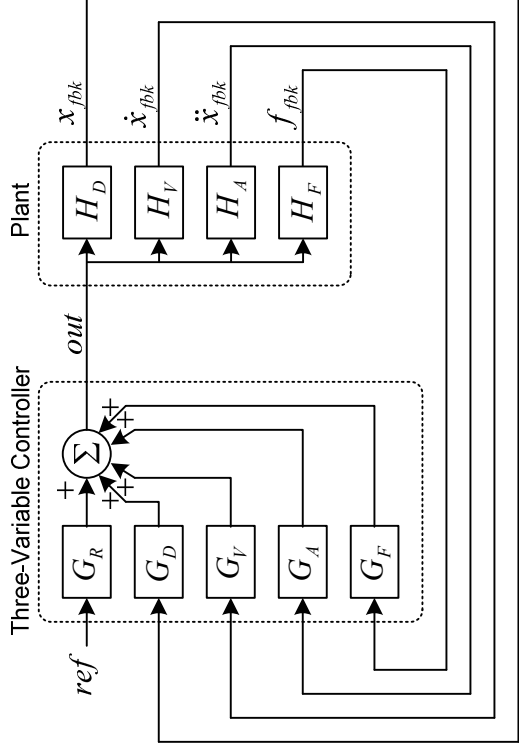


in realtime
with controller
hardware

Forward Model Driven by Actual Controller in Non-Realtime: Example



Control System Analysis Tools: Controller Transfer Functions



(from TVC Transfer Functions.pdf)

$$G_{COM}(z) = k_M G_{DLY}(z) G_{INT}(z) \prod_{i=1}^5 G_{NOT_i}(z)$$

$$G_R(z) = G_{COM}(z) \cdot \left[(k_{PF} + k_P) G_{RD}(z) \cdot \frac{10}{D_{MAX}} + k_{VP} G_{RV}(z) \cdot \frac{10}{V_{MAX}} + k_{JF} G_{RJ}(z) \cdot \frac{10}{A_{MAX}} \right]$$

$$G_D(z) = -k_P G_{COM}(z) \cdot \frac{10}{D_{MAX}}$$

$$G_V(z) = -k_V G_{COM}(z) \cdot \frac{10}{V_{MAX}}$$

$$G_A(z) = -k_A G_{COM}(z) \cdot \frac{10}{A_{MAX}}$$

$$G_F(z) = -k_{DP} G_{COM}(z) G_{DP}(z) \cdot \frac{10}{F_{MAX}}$$

$$G_{OL} = G_D H_D + G_V H_V + G_A H_A + G_F H_F$$

$$\frac{A}{R} = \frac{G_R H_A}{1 - G_{OL}}$$

Used by control engineers to determine

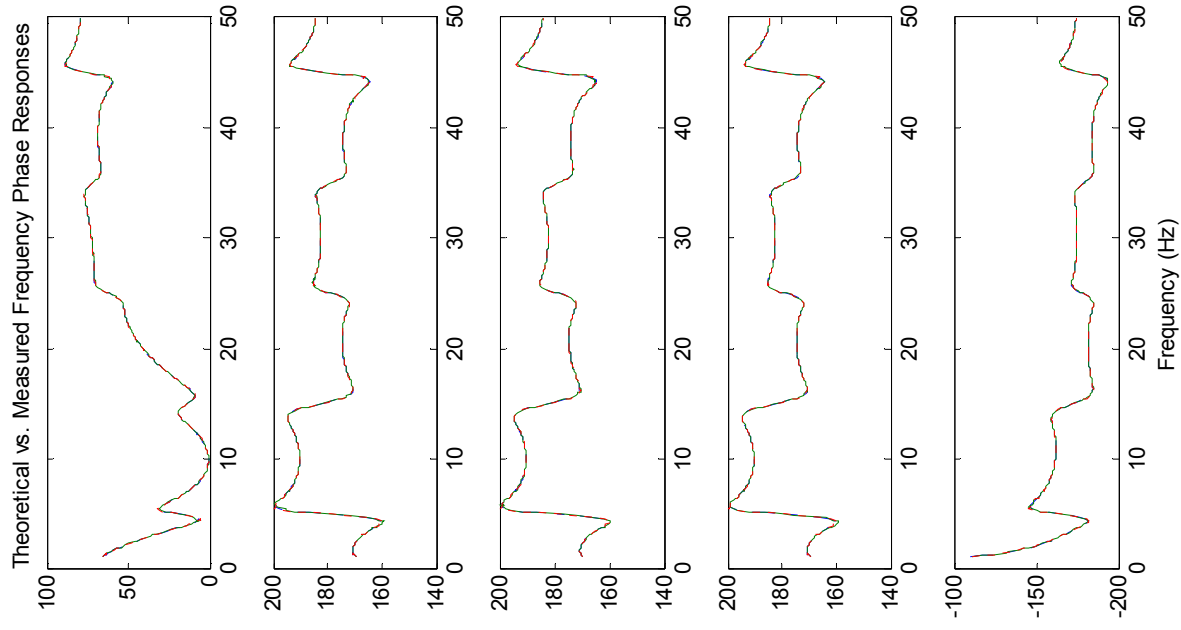
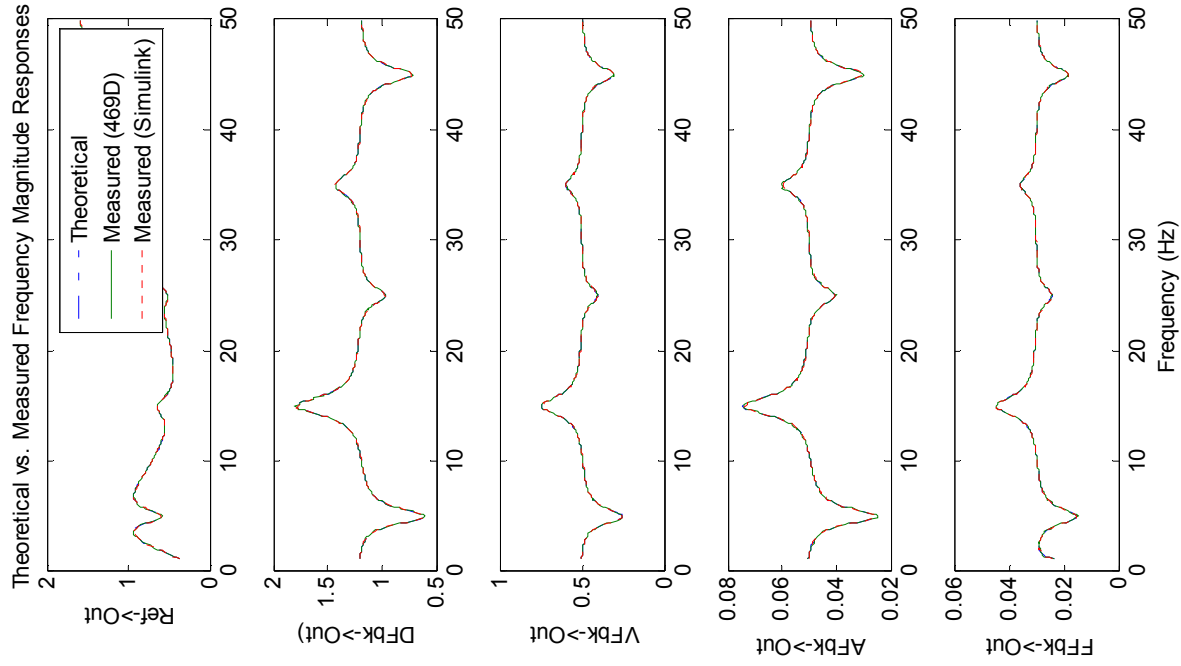
- Stability margins.
- Closed-loop response.
- Disturbance rejection.

Control System Analysis Tools: Controller Transfer Functions (cont.)

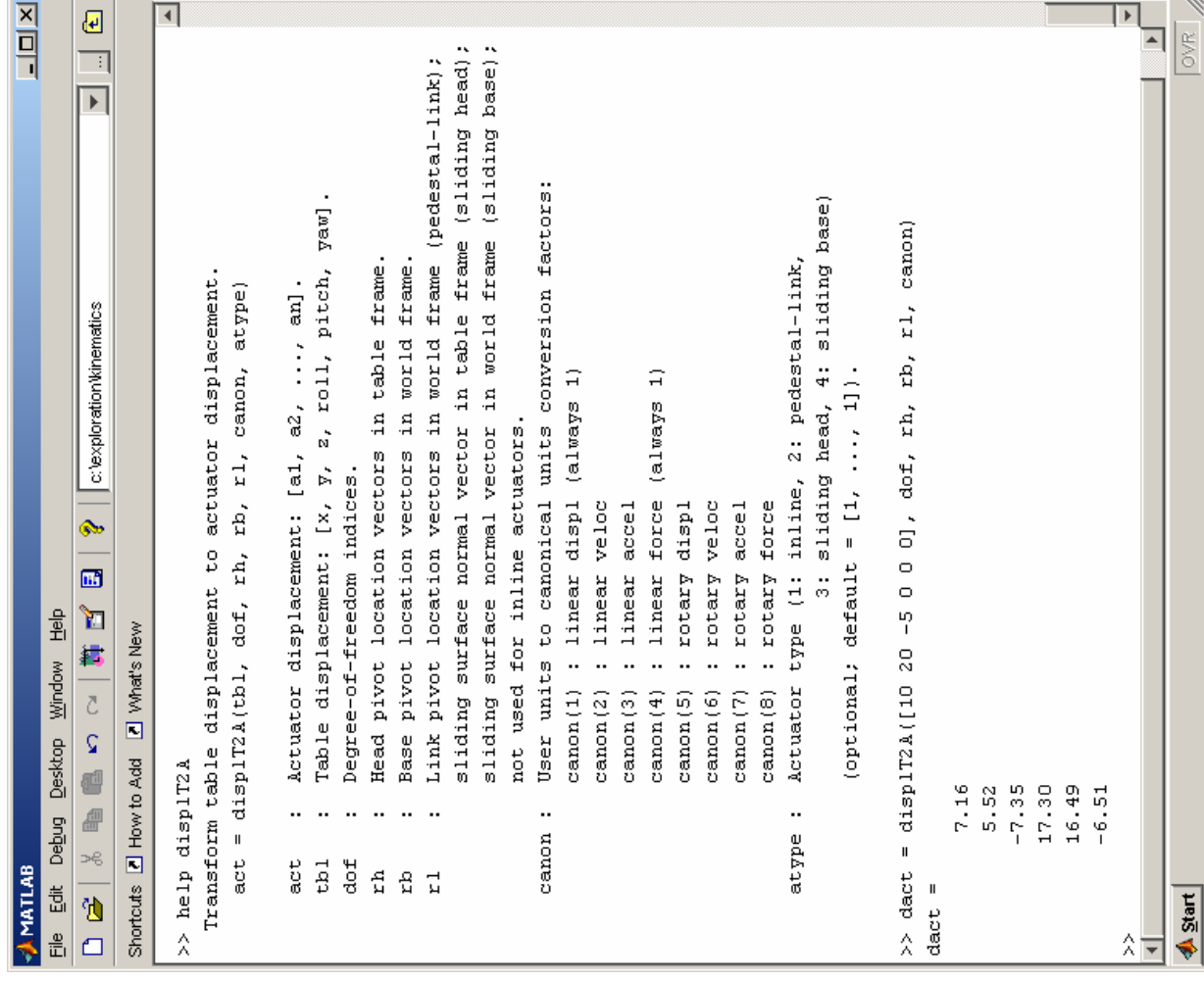
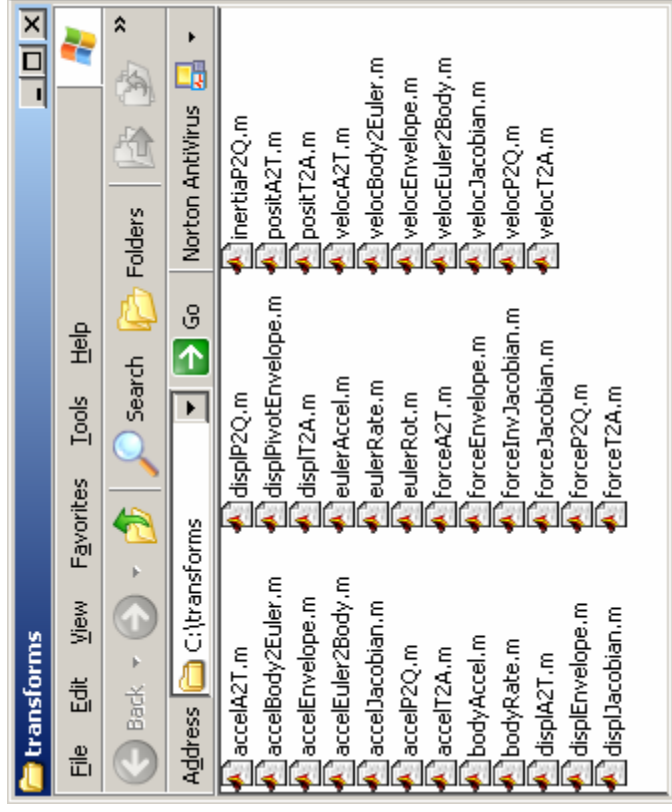
Matlab function “tvcreponse.m”:

```
>> help tvcreponse
Calculate the frequency response of a three-variable controller.
[rfrf, dfrf, vfrf, afrf, ffrf] = ...
tvcreponse(fhp, flp, damp, mgain, gains, notch, canon, max, mode, dly, f, fsamp)
rfrf : Frequency response from reference input to ctrlr output at eval freqs.
dfrf : Frequency response from displ fbk input to ctrlr output at eval freqs.
vfrf : Frequency response from veloc fbk input to ctrlr output at eval freqs.
afrf : Frequency response from accel fbk input to ctrlr output at eval freqs.
ffrf : Frequency response from force fbk input to ctrlr output at eval freqs.
fhp  : Reference generator highpass cut-in frequency (Hz).
flp  : Reference generator lowpass cutoff frequency (Hz).
damp : Reference generator filter damping ratio.
mgain : Master gain.
gains : Three-variable ctrlr gains (kpf, kvf, kaf, kjf, kp, kv, ka, kdp, fdp, ki).
notch : Notch filter parameters (freq1, bw1, depth1; ... ; freq5, bw5, depth5).
mode  : Operation mode (1: displ, 2: veloc, 3: accel).
canon : Canonical units conversion factors ([displ veloc accel jerk]).
max    : Maximum physical range ([displ veloc accel force]).
dly    : Controller delay (samples).
f      : Evaluation frequencies (Hz).
fsamp  : Sample frequency (Hz).
```

Control System Analysis Tools: Controller Transfer Functions (cont.)



Control System Analysis Tools: Kinematic Transform Functions



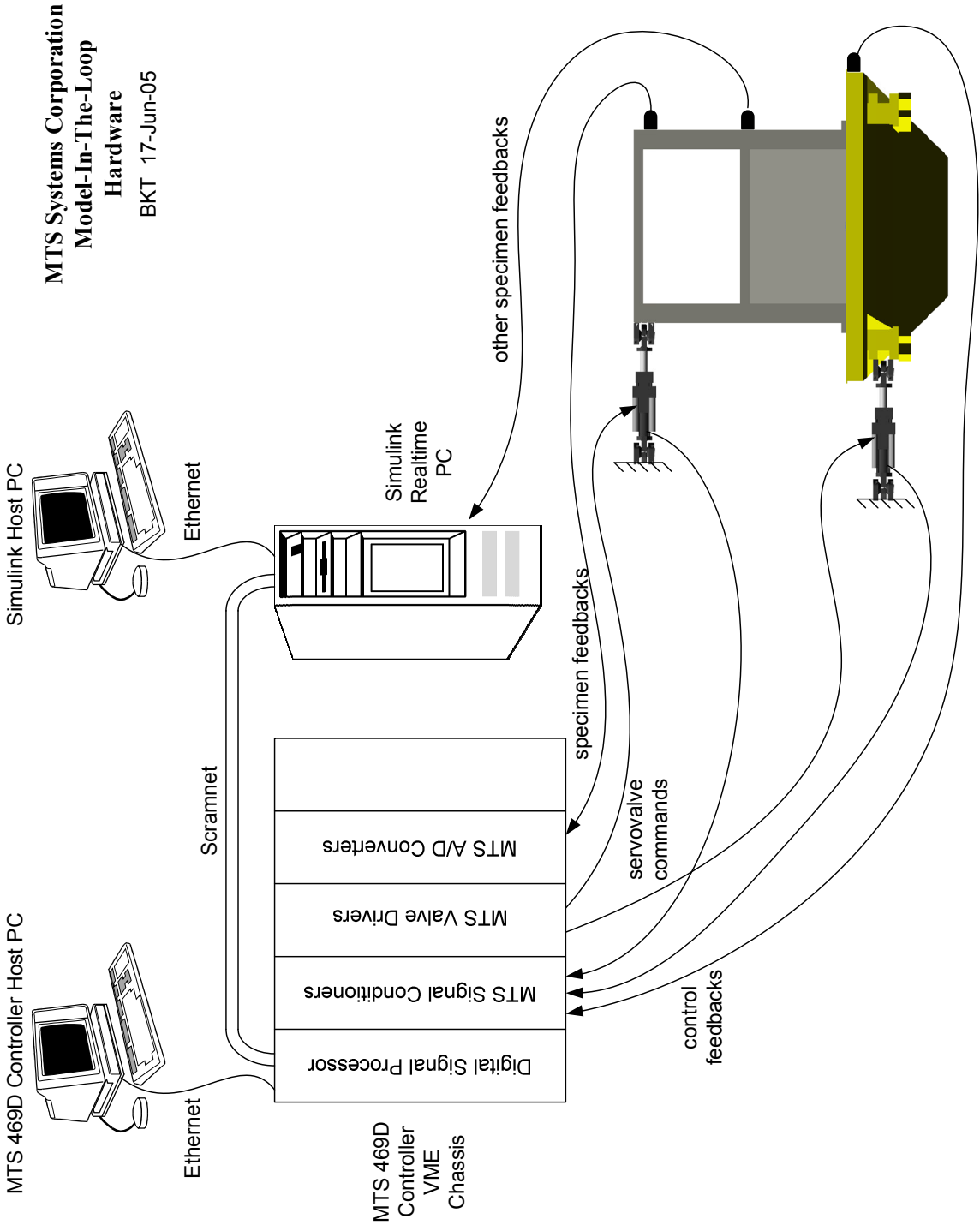
Realtime Model-In-The-Loop

Integrate your own model, control algorithm, or data acquisition with MTS's controller and run it in realtime.

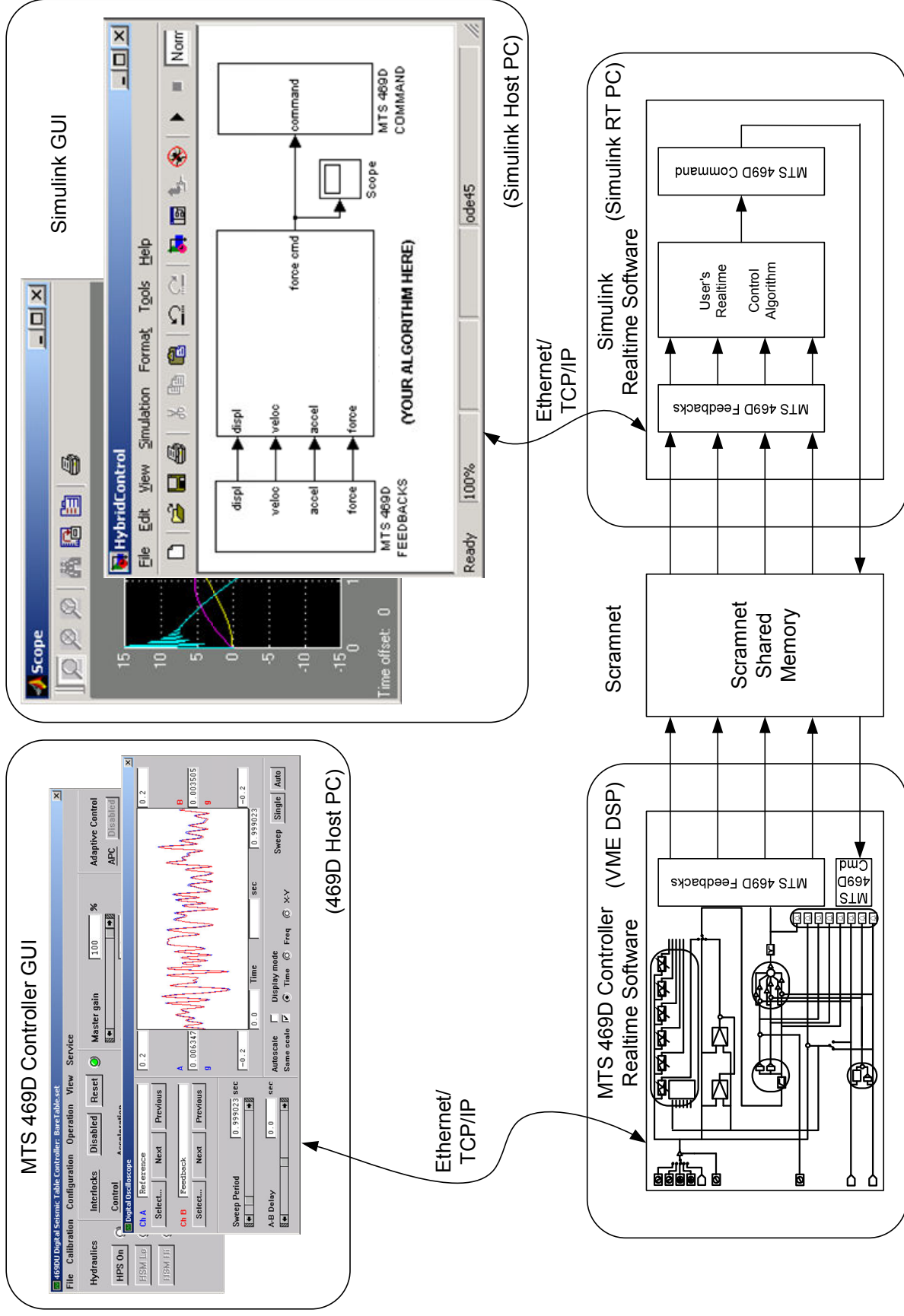
Applications:

- “Rehearse” a test by performing an end-to-end check of hardware and software.
- Design your own controller.
 - Example: “Hybrid” structures testing, consisting of a real structure on the table and a “virtual” structure modeled in the computer.
- Automate a test by integrating command waveform generation with data acquisition .

Realtime Model-In-The-Loop: Hardware View



Realtime Model-In-The-Loop: Software View



**Realtime Model-In-The-Loop:
SCRAMNET Distributed Shared Memory**

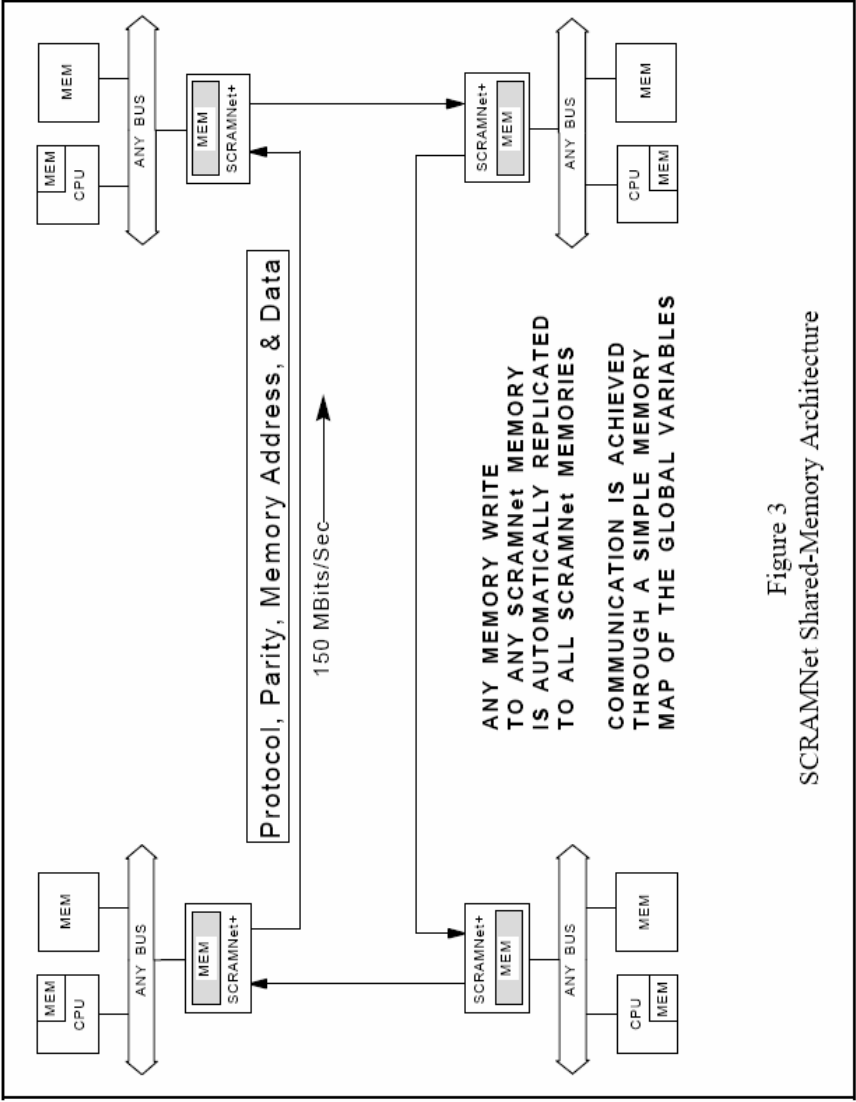


Figure 3
SCRAMNet Shared-Memory Architecture

Modeling Environments: Simulink

Simulink® 6

Simulation and model-based design

Simulink® is a platform for multidomain simulation and model-based design of dynamic systems. It provides an interactive graphical environment and a customizable set of block libraries that let you accurately design, simulate, implement, and test control, signal processing, communications, and other time-varying systems.

Add-on products extend the Simulink environment with tools for specific modeling and design tasks and for code generation, algorithm implementation, test, and verification.

Simulink is integrated with MATLAB® providing immediate access to an extensive range of tools for algorithm development, data visualization, data analysis and access, and numerical computation.

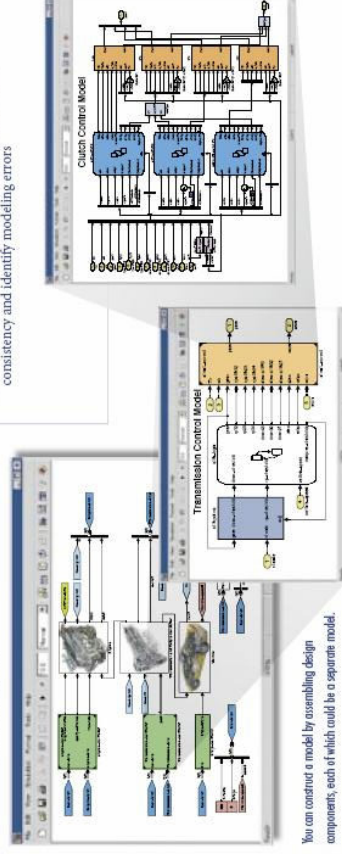
Creating and Working with Models

With Simulink, you can quickly create, model, and maintain a detailed block diagram of your system using a comprehensive set of predefined blocks. Simulink provides tools for hierarchical modeling, data management, and subsystem customization, making it easy to create concise, accurate representations, regardless of your system's complexity.



KEY FEATURES

- Extensive and expandable libraries of predefined blocks
- Interactive graphical editor for assembling and managing intuitive block diagrams
- Ability to manage complex designs by segmenting models into hierarchies of design components
- Model Explorer to navigate, create, configure, and search all signals, parameters, and properties of your model
- Ability to interface with other simulation programs and incorporate hand-written code, including MATLAB algorithms
- Option to run fixed- or variable-step simulations of time-varying systems interactively or through batch simulation
- Functions for interactively defining inputs and viewing outputs to evaluate model behavior
- Graphical debugger to examine simulation results and diagnose unexpected behavior in your design
- Full access to MATLAB for analyzing and visualizing data, developing graphical user interfaces, and creating model data and parameters
- Model analysis and diagnostics tools to ensure model consistency and identify modeling errors



You can construct a model by assembling design components, each of which could be a separate model.

Modeling Environments: LabView

The LabVIEW Environment

NI LabVIEW is the graphical development environment designed specifically for engineers and scientists who need to create flexible and scalable test, measurement, and control applications rapidly and at minimal cost. LabVIEW is a fully functional graphical programming language with the flexibility of a traditional, text-based language. With interactive Express technology such as automatic code generation, measurement assistants, application templates, and configurable Express VIs, LabVIEW makes development fast and easy for all users, regardless of programming experience. Using the intuitive graphical LabVIEW interface, you can focus on solving the measurement or control problem at hand rather than the programming necessary to implement it.

Virtual Instruments

The basic building block of a LabVIEW application is a virtual instrument (VI), which consists of a front panel, where you design a user interface, and a block diagram, where you create graphical code.

Modularity and Hierarchy

LabVIEW VIs are modular in design, so you can run any VI by itself or as a subVI. As a result, your LabVIEW programs can easily scale from simple data logging applications to highly sophisticated control systems depending on your needs.

Dataflow Programming

NI LabVIEW uses a patented dataflow programming model that frees you from the sequential architecture of text-based programming languages. The graphical code is highly intuitive for engineers and scientists familiar with block diagrams and flowcharts. Because the execution order in LabVIEW is determined by the flow of data between nodes and not by sequential lines of text, you can easily create block diagrams that execute multiple operations in parallel. Additionally, the parallel nature of LabVIEW makes multitasking and multithreading simple to implement. You can easily assign thread priorities from a menu selection with no programming.

Compiled Execution

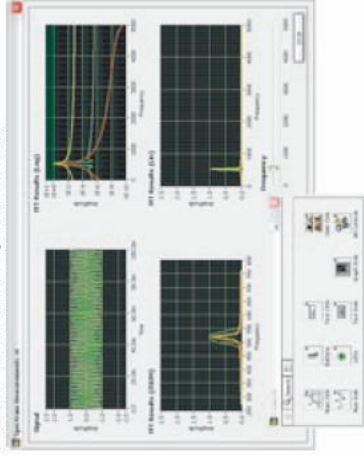
In many applications, execution speed is critical. LabVIEW is a graphical programming environment with a compiler that generates optimized code with execution speeds comparable to compiled C programs. You can use the LabVIEW VI profiler to analyze and optimize time-critical operations. LabVIEW increases your productivity without sacrificing execution speed.

Open Environment

LabVIEW is an open environment that provides easy connectivity to third-party software. Connect to your existing applications through LabVIEW with .NET assemblies, ActiveX DLLs, and a wide array of networking protocols. You also can create stand-alone executables and shared libraries in LabVIEW to call from other software environments.

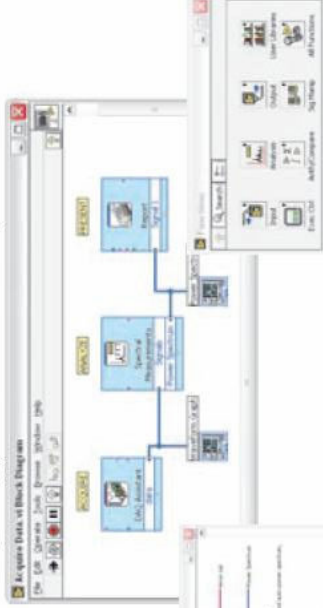
The LabVIEW Environment

Front Panel – The user interface of your virtual instrument.



Controls Palette – Drag and drop engineering-specific controls and indicators onto the front panel to quickly create a fully-customized user interface.

Block Diagram – The graphical code that defines the behavior of your virtual instrument.



Functions Palette – Define functionality by dropping VIs, constants, and programming structures onto the Block Diagram and wiring them together.

Context Help – Easily access detailed help for every VI and structure.

Getting Started

Don't start from scratch. Select from 19 VI templates, six sophisticated VI architectures, more than 500 included example programs, and thousands of example programs on the Web to start your application.



Example Finder – Search more than 500 VI examples included with LabVIEW and thousands more on the Web.

Creating Your Application

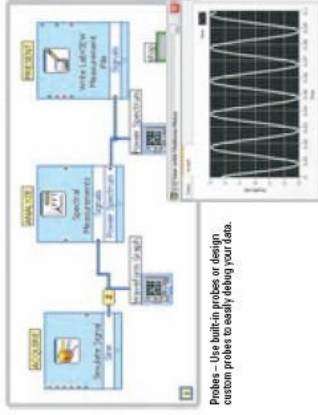
Use interactive LabVIEW development tools to start developing immediately and finish projects faster.



Express VIs – Point and click to configure measurement tasks with minimal programming.

Debugging Your Application

Verify that your program is robust and accurate with built-in debugging tools.



Probes – Use built-in probes or design custom probes to easily debug your data.

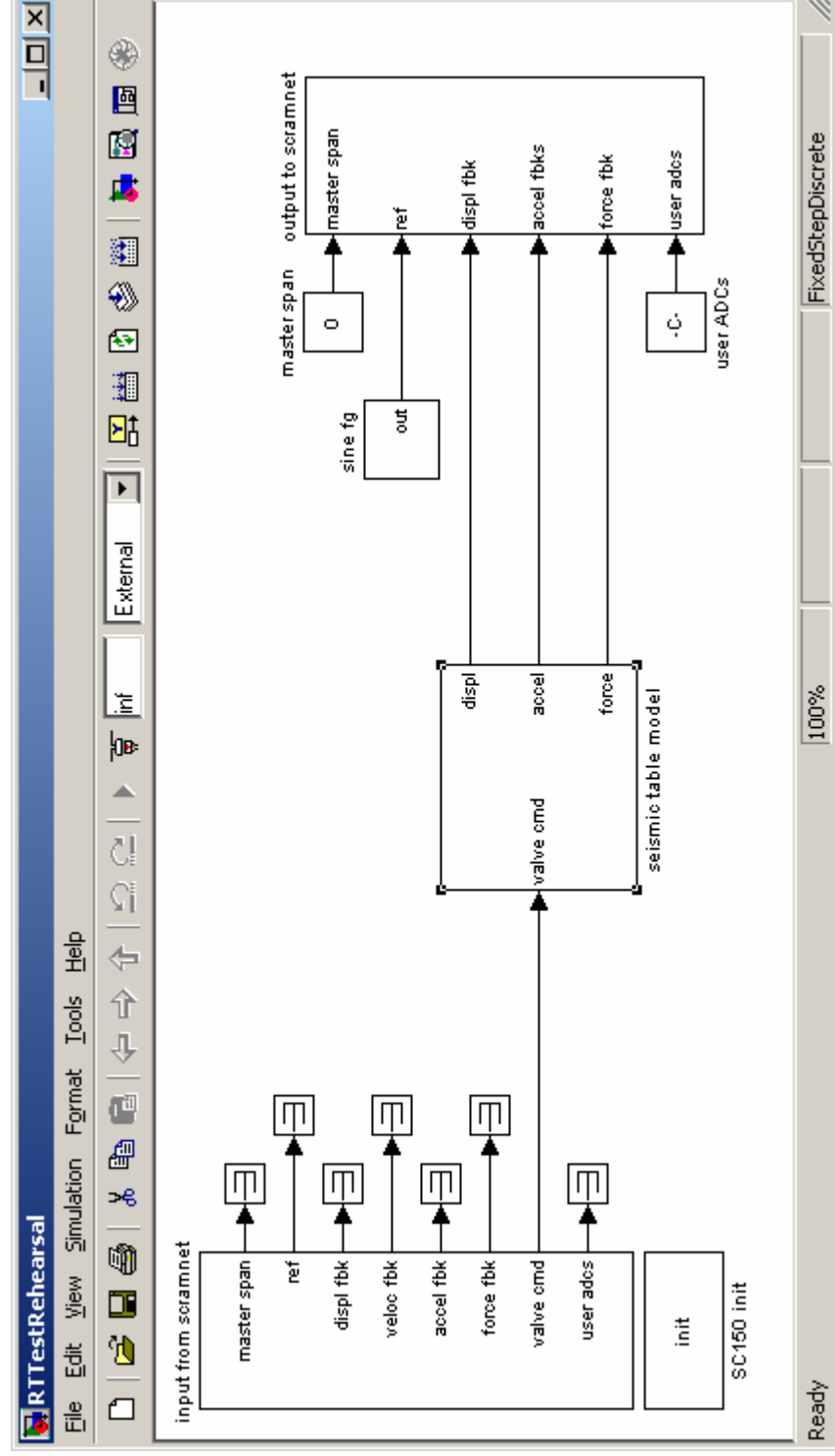


Highlight Execution and Step Into/Out of Over – Use highlight execution to visualize the flow of your program. Step into or out of a function or application using standard debugging features.

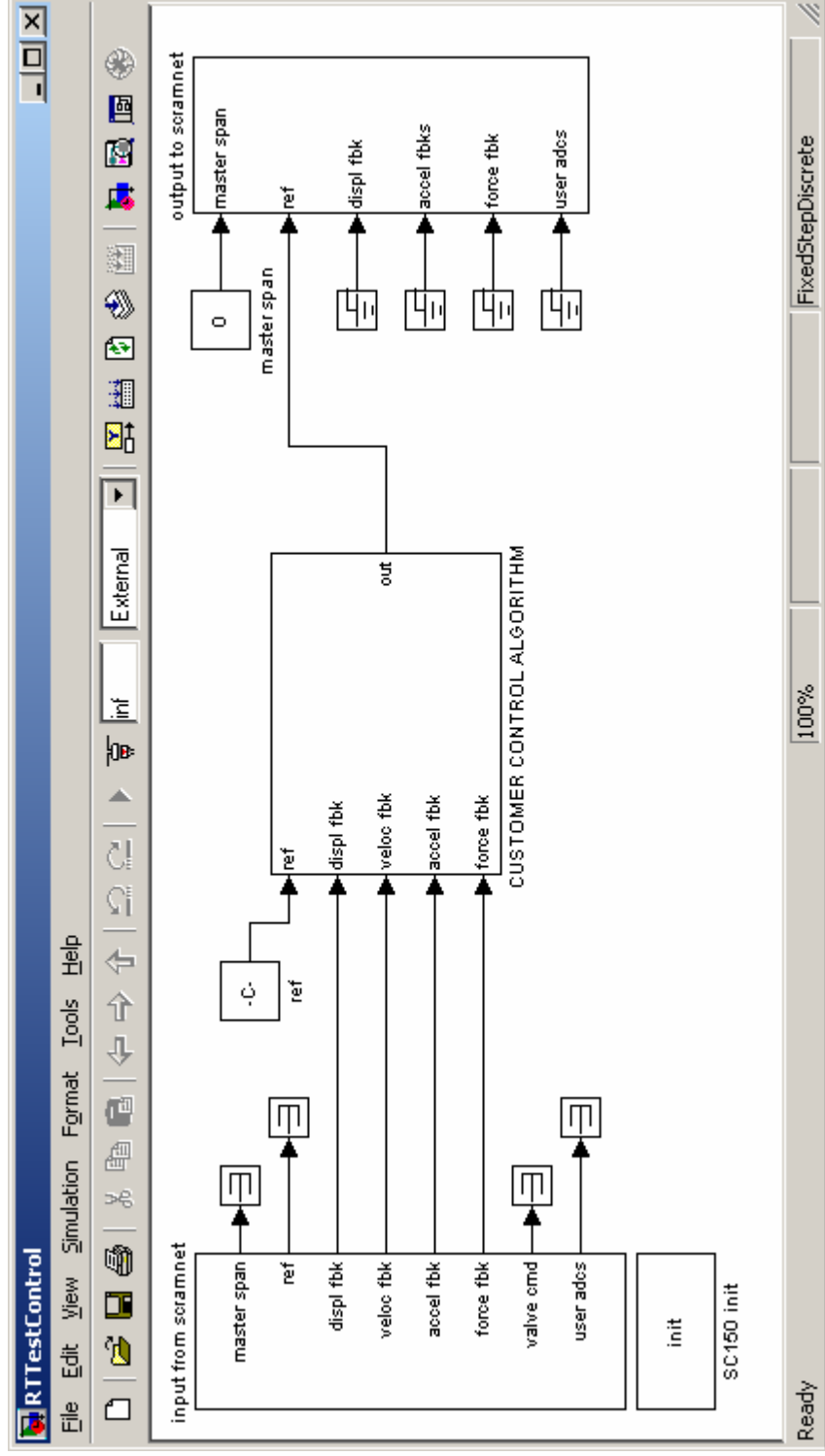
Assistants and Code Generation – Use the interactive DAO Assistant and Instrument I/O Assistant to interface with I/O and automatically generate code that you can run or modify.

Realtime Test Rehearsal Model

Useful for performing an end-to-end check of hardware and software system components before turning on hydraulics.

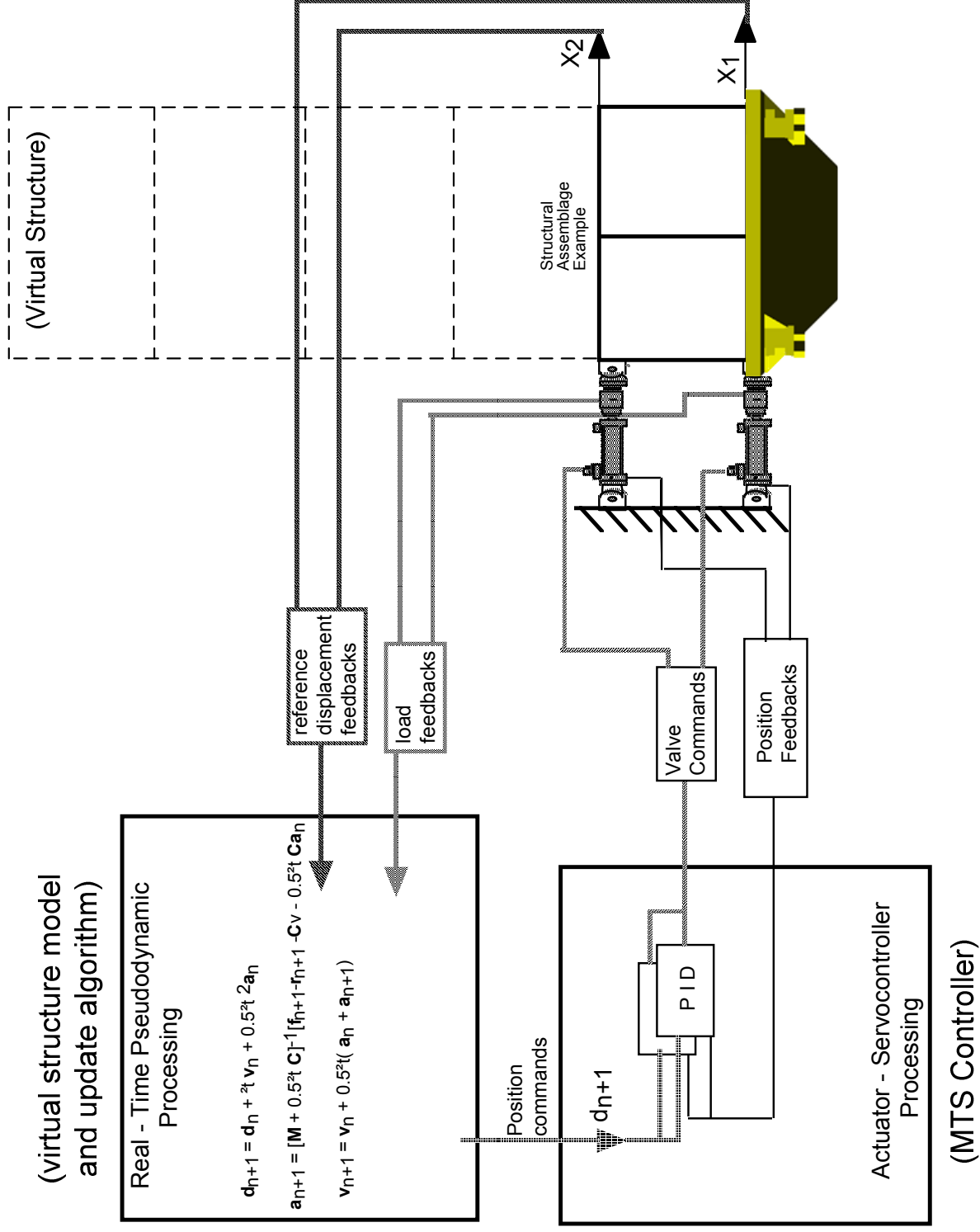


Realtime Test Control Model

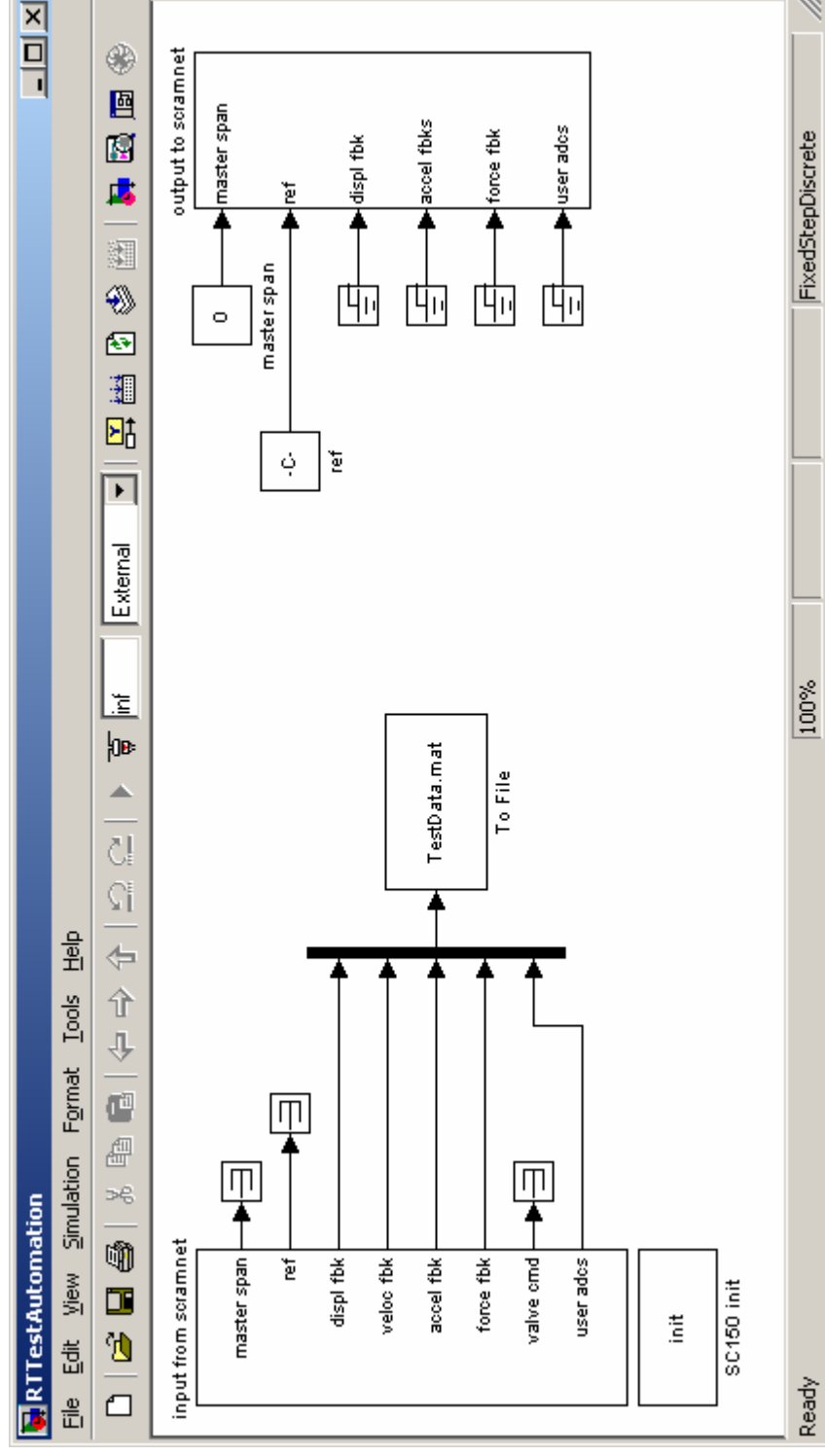


Realtime Test Control Model Example:

Fast Hybrid Structural Test System



Realtime Test Automation Model



Note: University of Nevada at Reno used this technique to automate a series of 700+ tests on base isolators, saving over 82 hours of testing time. See paper *Seismic Shake Table Test Automation Using Shared Memory*.

Realtime Test Automation (cont.)

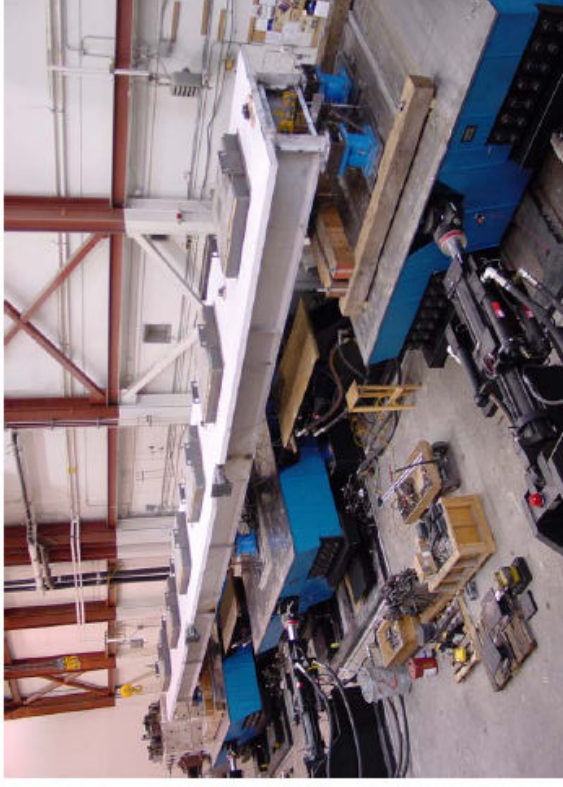


Seismic Shake Table Test Automation Using Shared Memory

Patrick N. Laplace¹

The Large Scale Structures Laboratory (LSSL) at UNR has owned and operated two 50-ton MTS shake tables since 1994 and added a 3rd 50-ton MTS shake table in 2002. The primary focus of the LSSL is multiple-span bridges and bridge component testing, which required 3 shake tables that can be operated independently or simultaneously for single or multiple-table tests. This is a unique requirement due to the challenges of operation and the lack of previous multiple-table installations. Operation and control become critical once multiple tables are utilized simultaneously with various specimens spanning the multiple tables.

One bridge research project required over 700 earthquake motions to be applied to a 2-span base-isolated bridge using the laboratory's 3 shake tables, shown below.



The requirements were:

- Apply 700 + earthquake motion combinations
- 20 different earthquake records
- For each test, various combinations of earthquake records, amplitudes, axes, and phase
- Rest time of 120 seconds between motions to allow bridge base isolation system to rest.

¹ Research Assistant Professor / Senior Laboratory Manager, Rogers and Weiner Large Scale Structures Laboratory, University of Nevada, Reno, Reno, NV 89557, Email: laplace@unr.edu

Offline Tuning

Idea: Tune the controller offline” using a specimen model rather than the specimen.

Perfect specimen model not necessary to achieve substantial improvement.

Use actual controller to drive forward model, either in non-realtime or realtime.



13th World Conference on Earthquake Engineering
Vancouver, B.C., Canada
August 1-6, 2004
Paper No. 960

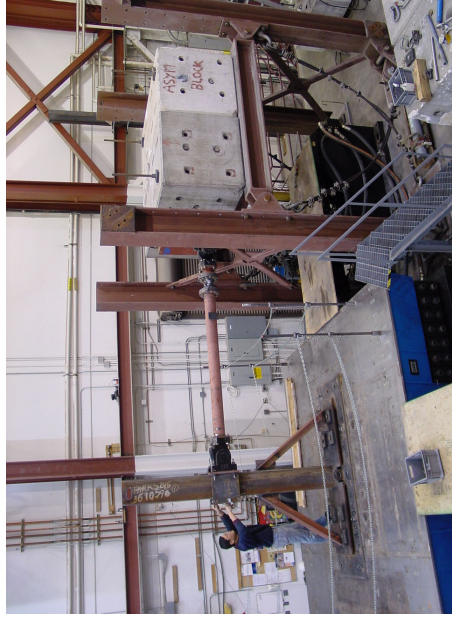
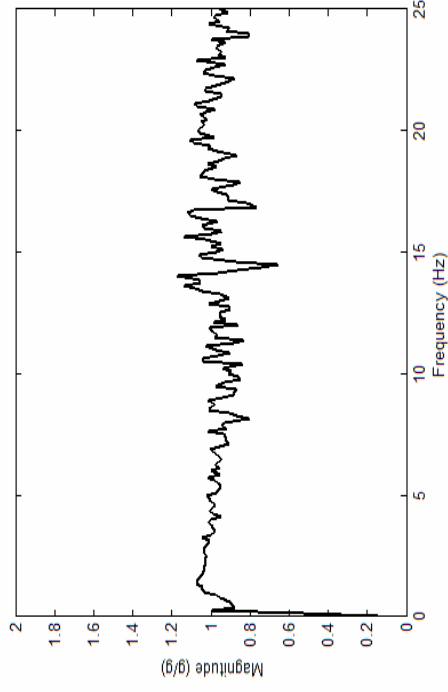
OFFLINE TUNING OF SHAKING TABLES

Bradford K. THOEN¹, Patrick N. LAPLACE²

SUMMARY

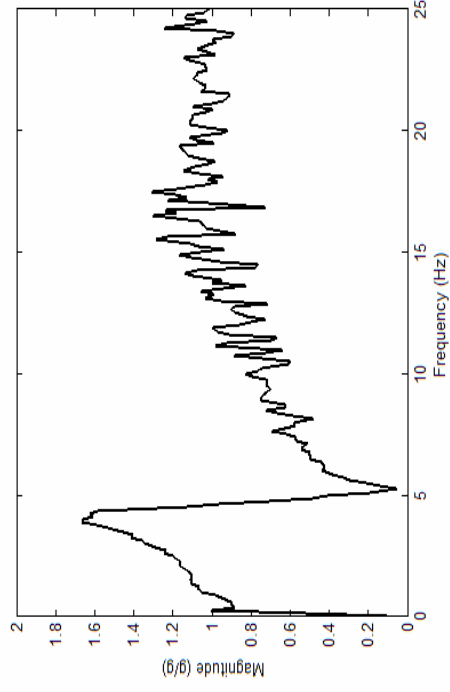
The control performance of shaking tables is greatly affected by the interaction between the table and the test specimen. This is because the dynamic characteristics of large specimens give significant force disturbances to the table. For this reason it is not appropriate to run the table with controller tunings determined using the bare table if good control is desired. The controller must be retuned with the specimen present. However, the shaking that is an unavoidable part of the tuning process can cause damage to the specimen. In this paper we present an alternative tuning method that involves tuning a dynamic model of the system. The actual digital control software is used to drive the dynamic model, thereby eliminating a large portion of modeling effort as well as the potential for error. Only the mechanical and hydraulic portions of the system need be modeled. The model is implemented in Mathworks Simulink®, and includes only those effects that characterize the dynamics to a degree sufficient for control tuning.

Bare Table Tuning

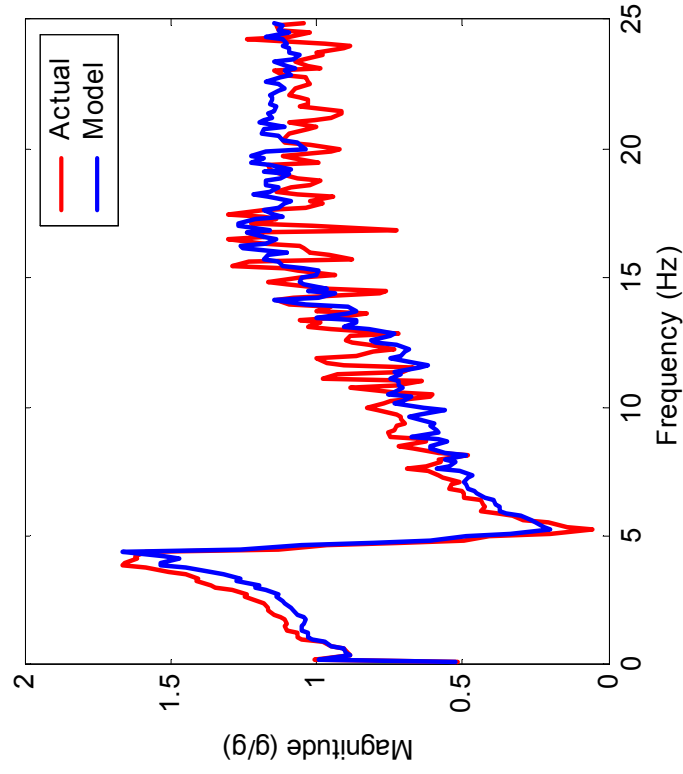


+ Heavy, Resonant Specimen

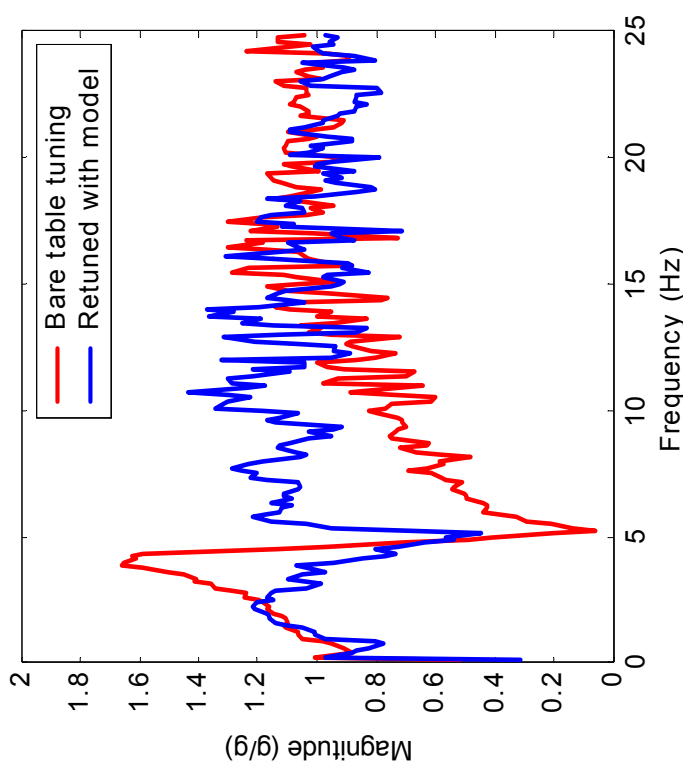
= Poor Response



Offline Tuning Results: Acceleration Frequency Responses

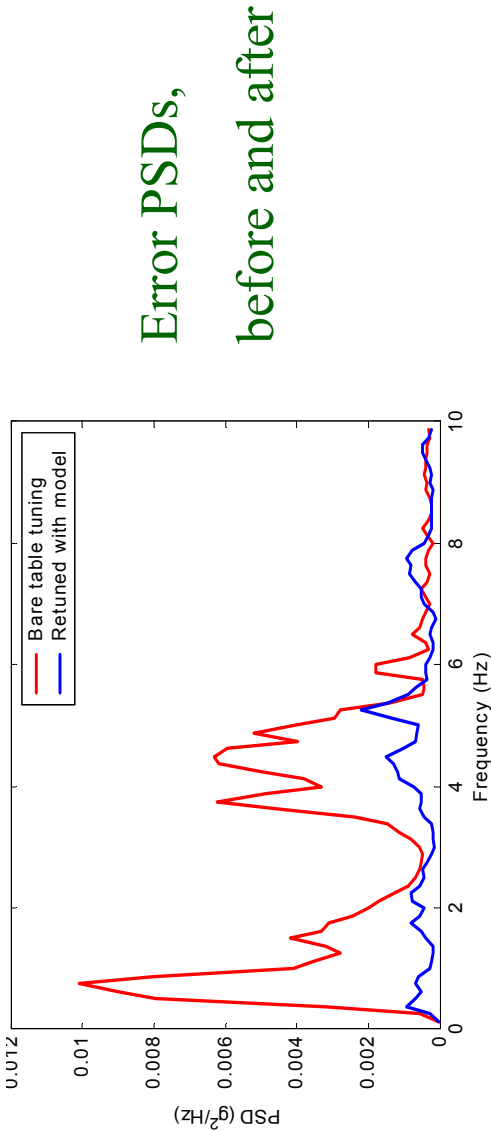
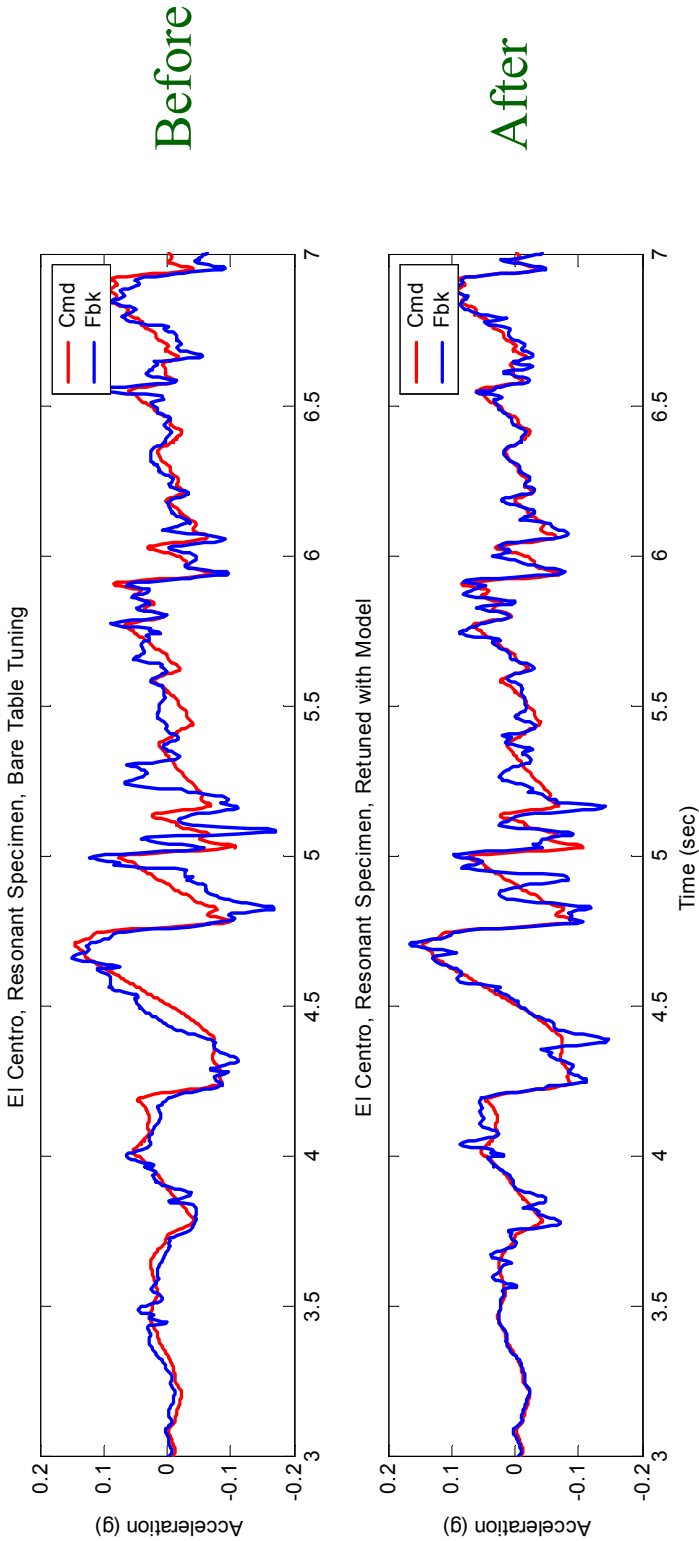


Actual and Model
Accel FRFs



Actual Accel FRFs,
before and after

Offline Tuning Results: Earthquake Time History Response



Error PSDs,
before and after

Caveats

- MTS provides models *free of charge* on an *as-is* basis.
- Customer is expected to add relevant dynamics and to validate model.
- Success depends on customer involvement and commitment.

Demonstrations

- Inverse model.
- Forward model driven by Simulink controller.
- Forward model driven by actual controller.