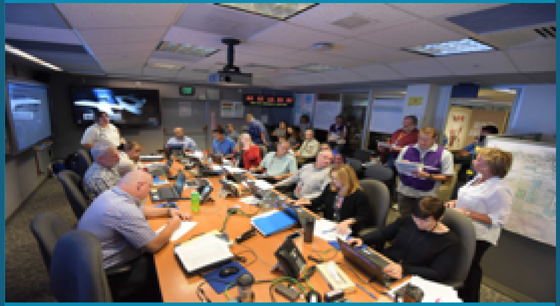
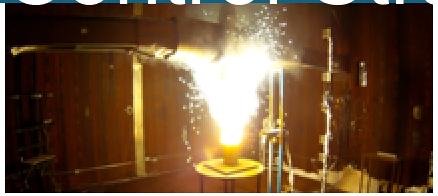
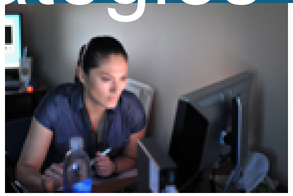
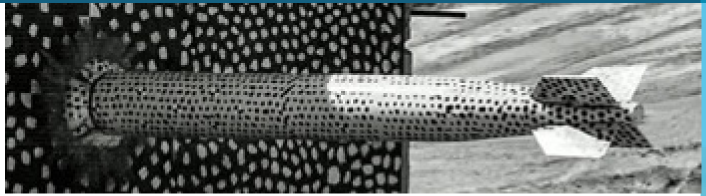
SAND2022-1034C

Rattlesnake: A Custom Combined-Environments Vibration Controller with Flexible Control Strategies









IMAC XL MIMO Vibration Short Course

Dan Rohe

Sandia National Laboratories is a multimission laboratory managed and operated by National Technology & Engineering Solutions of Sandia, LLC, a wholly owned subsidiary of Honeywell International Inc., for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-NA0003525.

Sandia National Laboratories is a multimission laboratory managed and operated by National Technology & Engineering Solutions of Sandia, LLC, a wholly owned subsidiary of Honeywell International Inc., for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-NA0003525.

What is Rattlesnake?

The *Rattlesnake* Vibration Controller: a flexible testbed for Vibration Research

Targets Multiple Hardware Devices
- Initially targeted NI Hardware for mature programming interface and flexible hardware.
- Extended to HBK LAN-XI hardware
- Able to extend to new devices with minimal modifications to the controller

Can perform Synthetic Control
- Integrates modal EoMs from FE solution or state space matrices in real-time
- Test out control using FE models, save your shakers and hardware
- Evaluate setup parameters without wasting time in the laboratory

Extensible to multiple or combined environments
- Environments are abstracted with clean interface for easy extension
- Currently can run MIMO Random, MIMO Transient, and Time History Generator
- Plans to expand to MIMO Sine, Nonlinear Force Appropriation, and Modal Testing
- Set up test profiles to start, modify, or stop environments automatically

Written in Python
- Easy to read, write, extend
- Open source

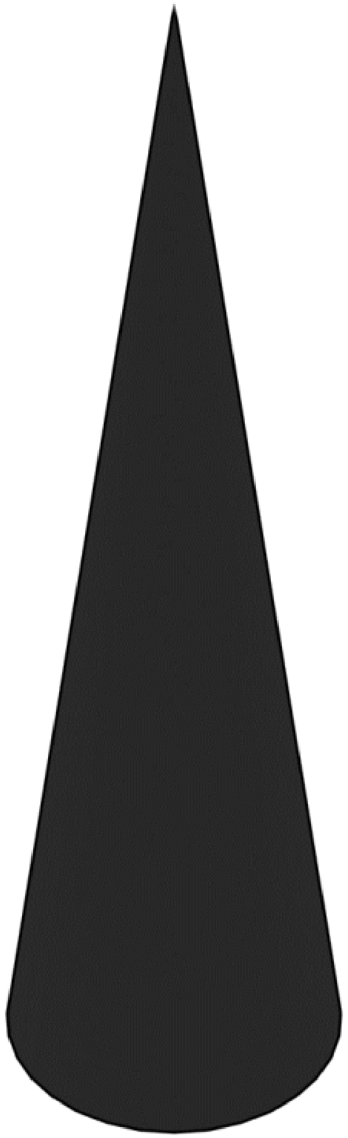
Multiprocessing Capable
- Performs computations in parallel to speed up responsiveness

Program your own control laws
- Control laws can be loaded into the controller without any modification to the controller itself

Update System ID during control
- Useful for nonlinear systems

Apply transformations to responses or outputs
- 6DoF transformations can be applied to control table rigid body motion
- Modal transformations can be applied to control to modal quantities

Designed for the lazy engineer
- Minimize data entry, load test directly from spreadsheet or data file
- All data and metadata from test stored to netCDF file on disk, can load settings from this file to re-create test



Download at: https://github.com/sandialabs/rattlesnake-vibration-controller

Vibration Research and Development is Hampered by Commercial Nature of Software

What if I have a new idea for a control law?

- Contact vendors to try to implement your idea
- Write a controller from scratch

What if commercial software doesn't work?

- No ability to see implementation, might be difficult to tell what is wrong
- Need to wait for vendor support/bugfixes before a test can proceed

As we at Sandia were pushing our vibration controllers harder and harder, we recognized the need for something we had more control over:

- Add new control laws, modal and kinematic transformations, large channel and exciter counts
- Need to see how and why things break down and update implementation
- Avoid program delays due to vendor's slow response time.

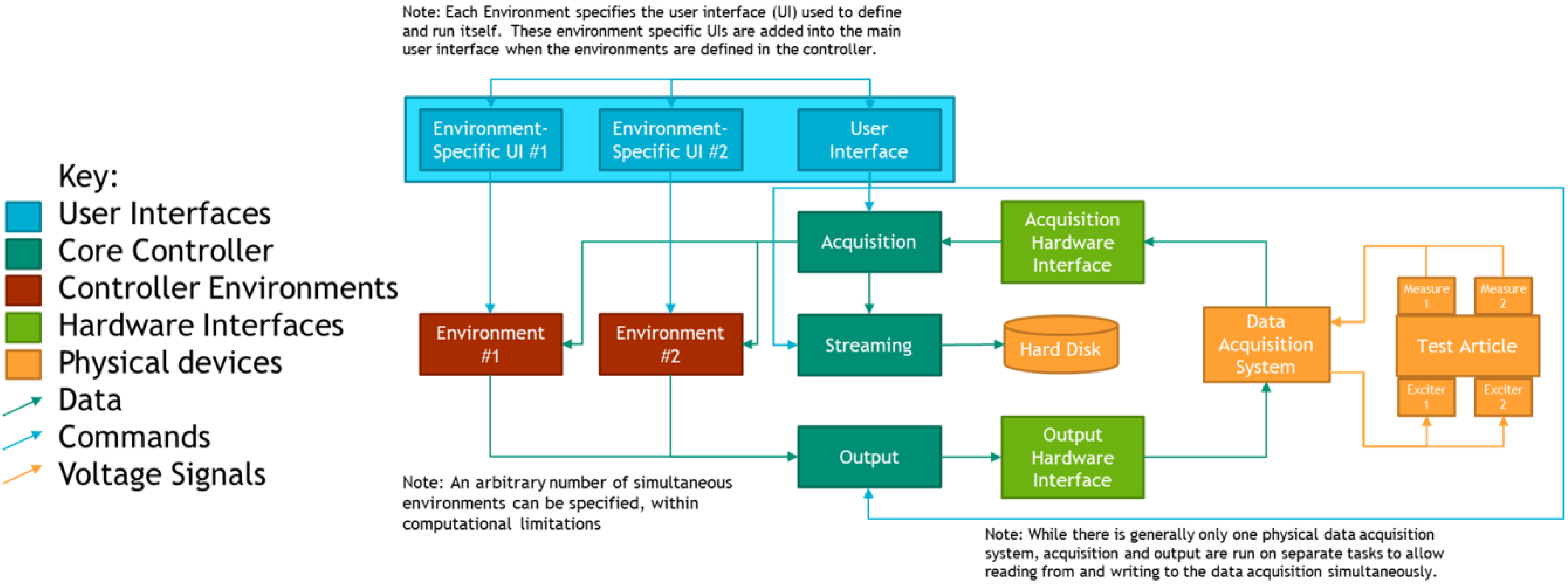
Note: Commercial software is invaluable for production or high-value tests where consistency and hardware safety is paramount.

How does Rattlesnake Work?

Rattlesnake Overview

Rattlesnake is a controller framework that enables flexible addition or removal of components.

- Environments generate data for the output
- Output interface translates Rattlesnake commands into hardware instructions to excite the structure
- Acquisition interface recovers response data from the hardware
- Acquisition sends acquired data to environments
- Environments read data to generate next output signal



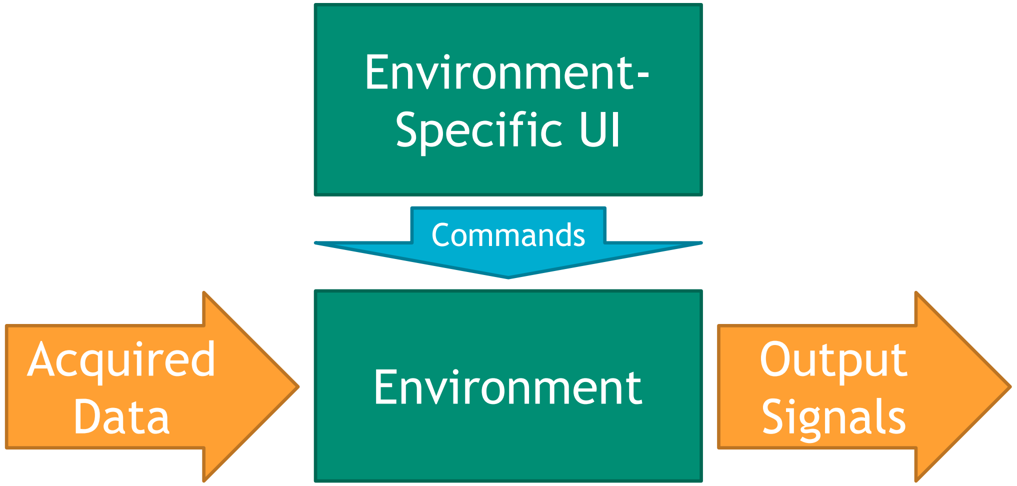
What is an environment?

In general, an environment is a portion of the controller that:

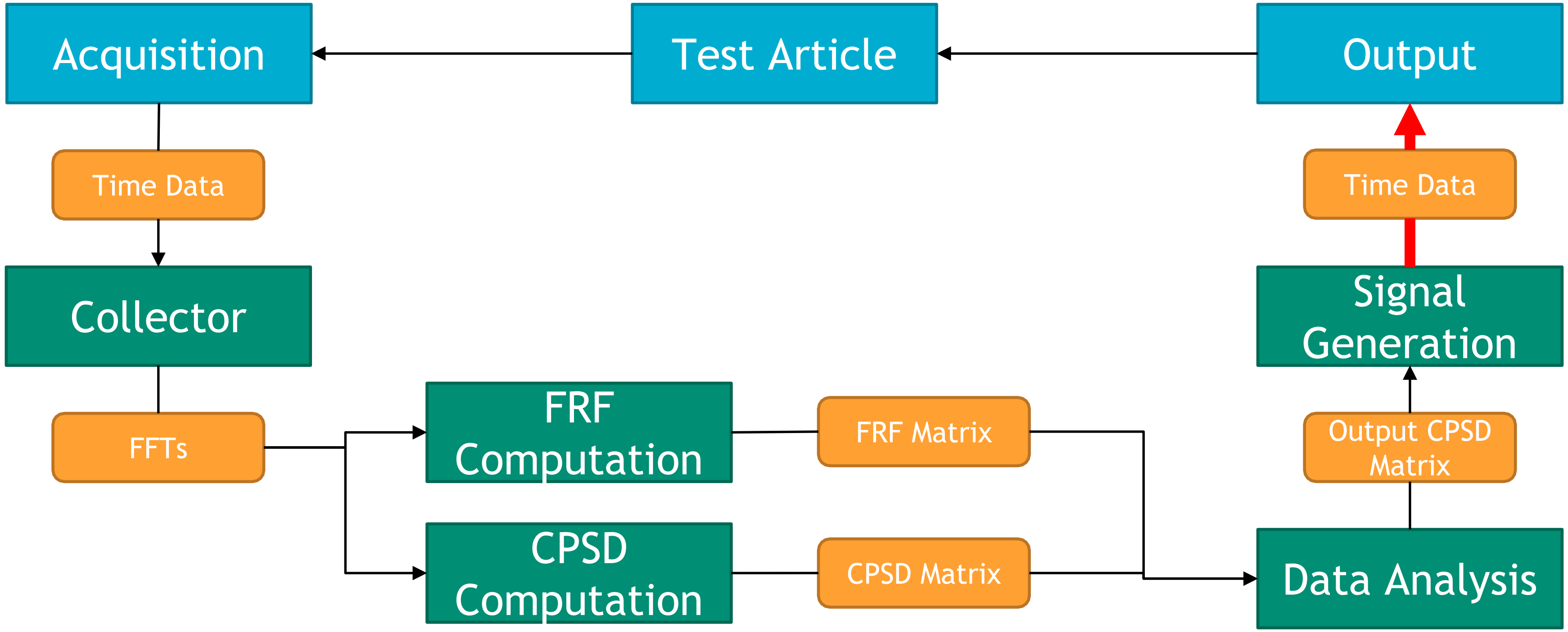
- accepts the previously measured data
- optionally analyzes the previously measured data for closed-loop response
- generates the next output signal required by the controller

The environment will consist of:

- An environment-specific user interface (UI)
- The environment implementation (run on a separate process from the UI)
- A set of metadata that completely specifies the environment (samples per frame, overlap, windows, etc.)



Example Environment: MIMO Random Vibration



Output is the only process that *needs* data to be received in a timely manner. Signal Generation must be able to create the next output signal by the time Output needs it.

All other processes can fall behind. E.g. if FRF computation falls behind, we simply use the most recent estimate of the FRF that we have.

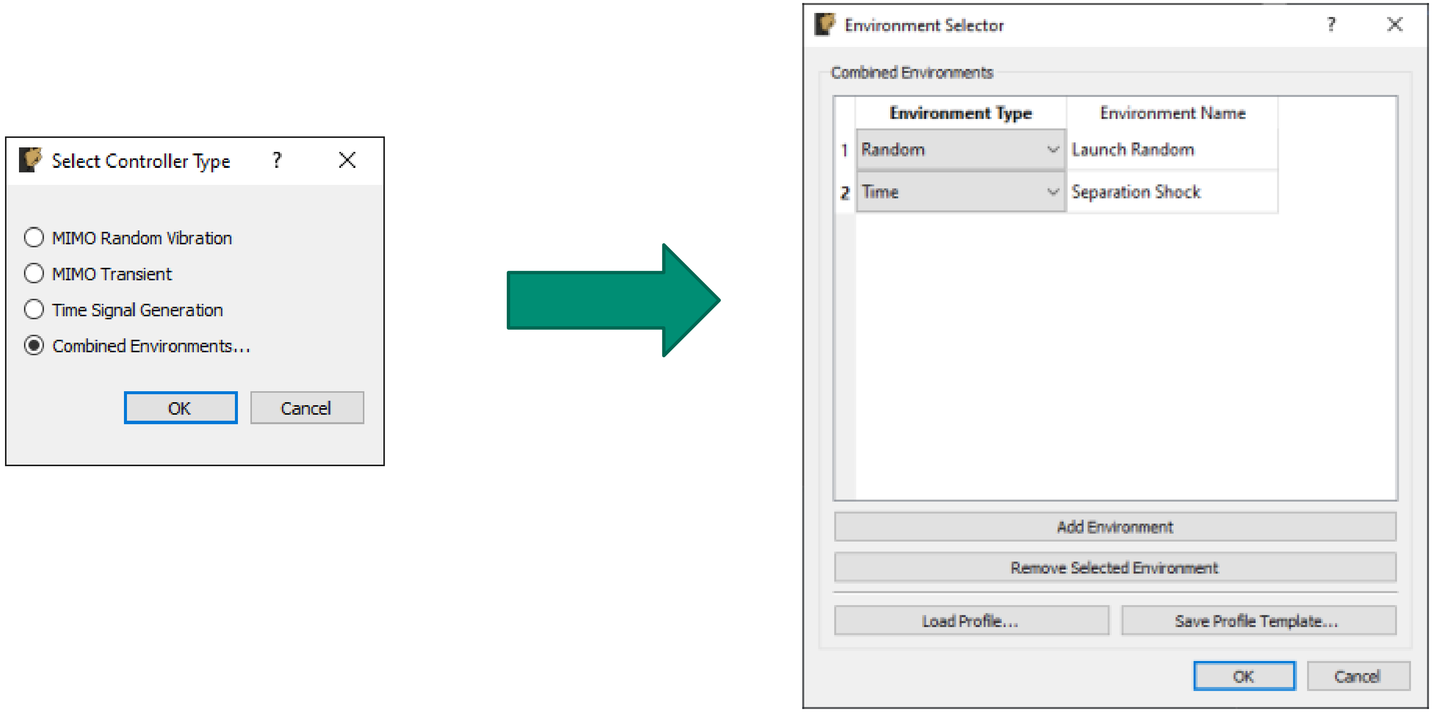
Rattlesnake Software Overview

Selecting the types of environments used in a test

Rattlesnake’s initial window prompts to select the type of test that will be run.

Users can select a single environment, or Combined Environments to run multiple environments simultaneously.

If combined environments is selected, another window will appear to allow users to select which environments are required.



Rattlesnake Main User Interface

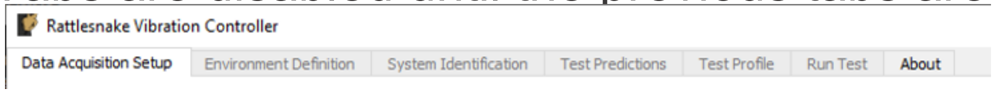
Once the environments are defined, the main user interface appears.

Rattlesnake utilizes a tabbed interface on the main window to house main components of the test

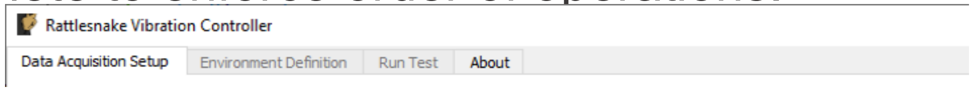
- **Data Acquisition Setup** – Sets up channel table and global data acquisition parameters
- **Environment Definition** – Sets up each environment in the test
- **System Identification** – Outputs flat random signals to identify relationships between output signals and control responses for each environment; only required by certain environments
- **Test Predictions** – Display predictions considering the system’s transfer functions and the defined environments to determine if a given test is feasible; only performed by certain environments
- **Test Profile** – Set up a test timeline using various commands to start, stop, or modify the environments; only used in combined environments mode.
- **Run Test** – Start or stop each environment, or run the test profile.

Depending on which environments are used in a given test, some of these tabs may not be visible.

Tabs are disabled until the previous tabs are complete to enforce order of operations.



Combined Environments Test with Environments requiring System ID



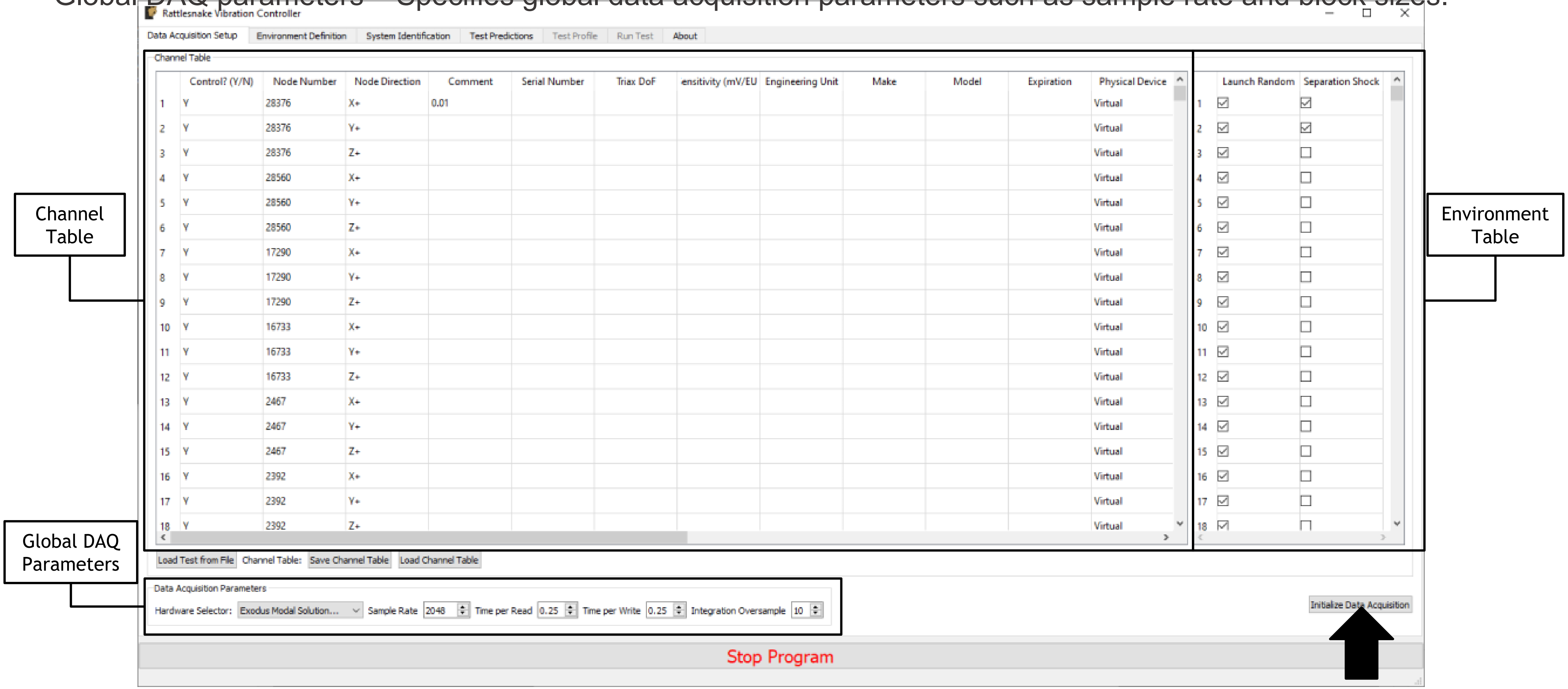
Single Environment Test not requiring System ID

Data Acquisition Setup

Channel Table – List of channels in a test, including which channels are used for control

Environment Table – Specifies which channels are associated with each environment; only for combined environments

Global DAQ parameters – Specifies global data acquisition parameters such as sample rate and block sizes.

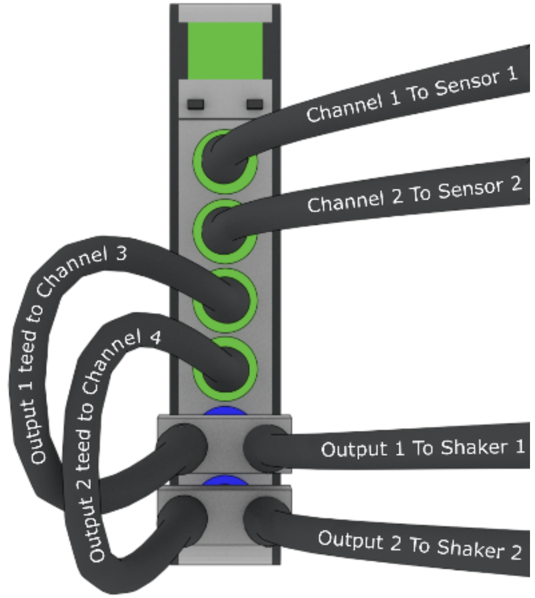


A note on instrumentation setup…

Rattlesnake requires the output signal to be measured by the acquisition to allow for synchronizing input and output of arbitrary data acquisition systems.

Generally this will mean output signals will be teed between the exciters and acquisition channels

There will be one row in the channel table for each measurement sensor as well as each shaker.



Acquisition Channels (rows 1–6); Output Channels (rows 7–10)

		Test Article Definition			Instrument Definition							Channel Definition								Output Feedback	
Channel Index	Control?	Node Number	Node Direction	Comment	Serial Number	Triax DoF	Sensitivity (mV/EU)	Engineering Unit	Make	Model	Calibration Exp Date	Physical Device	Physical Channel	Type	Minimum Value (V)	Maximum Value (V)	Coupling	Excitation Source	Excitation Value	Physical Device	Physical Channel
1	Y	103	Y+	103Y+	HH70		10.059	g	Endevco	7250AM-1		PXI1Slot2	ai0	Acceleration	-3.5	3.5		Internal	0.004		
2	Y	105	Y+	105Y+	HE70		9.927	g	Endevco	7250AM-1		PXI1Slot2	ai1	Acceleration	-3.5	3.5		Internal	0.004		
3	Y	112	Y+	112Y+	GT02		10.203	g	Endevco	7250AM-1		PXI1Slot2	ai2	Acceleration	-3.5	3.5		Internal	0.004		
4	Y	114	Y+	114Y+	DG65		9.802	g	Endevco	7250AM-1		PXI1Slot2	ai3	Acceleration	-3.5	3.5		Internal	0.004		
5	Y	119	Y+	119Y+	GY95		10.041	g	Endevco	7250AM-1		PXI1Slot2	ai4	Acceleration	-3.5	3.5		Internal	0.004		
6	Y	122	Y+	122Y+	FL05		9.908	g	Endevco	7250AM-1		PXI1Slot2	ai5	Acceleration	-3.5	3.5		Internal	0.004		
7	N	902	Y+	Shaker 1			1000	V				PXI1Slot2	ai8	Voltage	-3.5	3.5		None		PXI1Slot5	ao0
8	N	906	Y+	Shaker 2			1000	V				PXI1Slot2	ai9	Voltage	-3.5	3.5		None		PXI1Slot5	ao1
9	N	920	Y+	Shaker 3			1000	V				PXI1Slot2	ai10	Voltage	-3.5	3.5		None		PXI1Slot6	ao0
10	N	924	Y+	Shaker 4			1000	V				PXI1Slot2	ai11	Voltage	-3.5	3.5		None		PXI1Slot6	ao1

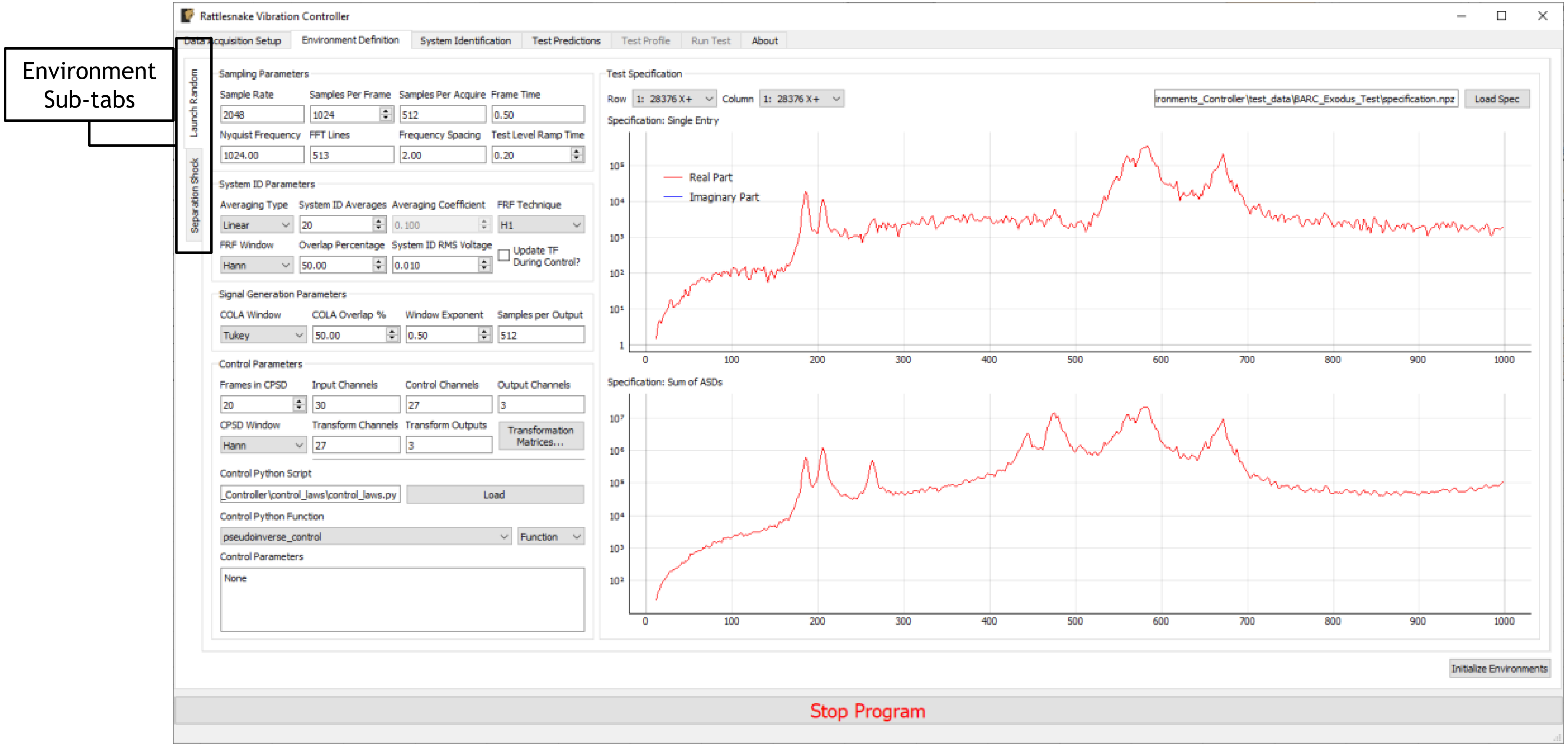
Input Device and Channel

Output Device and Channel

Defining Environment Parameters

Environment Definition has one sub-tab for each environment in the test.

We will need to define all environments before proceeding

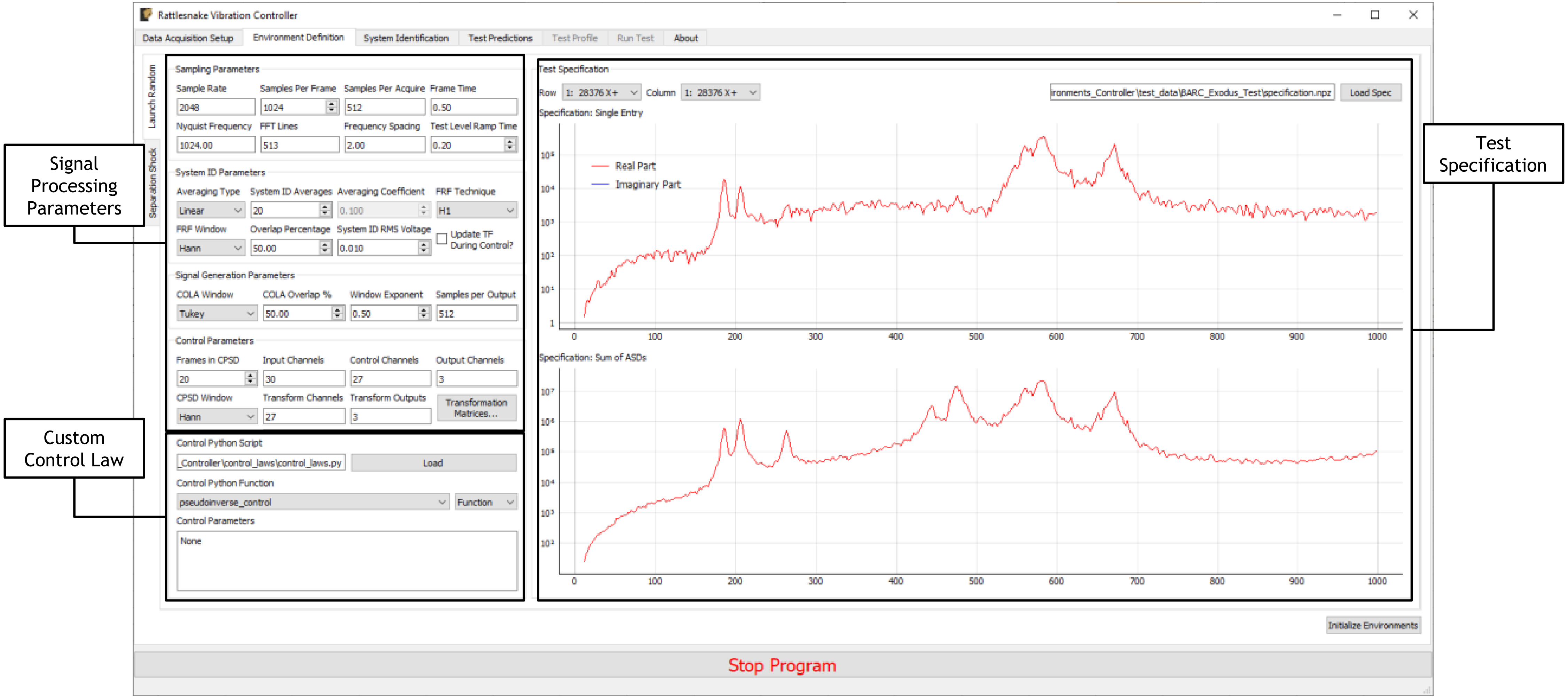


Defining control parameters – MIMO Random Vibration

MIMO Random environment allows definition of all signal processing parameters

A custom control law can be imported via a short Python script to implement e.g. closed-loop, buzz test, shape-constrained, or other strategies

Test specification CPSD matrix can be loaded from Matlab (.mat) or Numpy (.npz) files



Custom Control Laws for MIMO Random Vibration

To define a control law, simply define a Python function that accepts specific arguments and returns an output CPSD matrix.

$$f\left(\mathbf{H}_{xv}, \mathbf{S}_{xx}, \mathbf{S}_{xx,k}, \mathbf{S}_{vv,k}, \dots\right) \rightarrow \mathbf{S}_{vv,k+1}$$

Control laws can range from a few lines of code to large functions

```
def pseudoinverse_control(transfer_function,  # Transfer Functions
                          specification, # Specifications
                          buzz_cpsd, # Buzz test in case cross terms are to be computed
                          extra_parameters = '',
                          last_response_cpsd = None, # Last Response for Error Correction
                          last_output_cpsd = None, # Last output for Drive-based control
                          ):
    # Invert the transfer function using the pseudoinverse
    tf_pinv = np.linalg.pinv(transfer_function)
    # Return the least squares solution for the new output CPSD
    return tf_pinv@specification@tf_pinv.conjugate().transpose(0,2,1)
```

Simple pseudoinverse control

```
def match_trace_pseudoinverse(transfer_function,  # Transfer Functions
                              specification, # Specifications
                              buzz_cpsd, # Buzz test in case cross terms are to be computed
                              extra_parameters = '',
                              last_response_cpsd = None, # Last Response for Error Correcti
                              last_output_cpsd = None, # Last output for Drive-based contr
                              ):
    # If it's the first time through, do the actual control
    if last_output_cpsd is None:
        # Invert the transfer function using the pseudoinverse
        tf_pinv = np.linalg.pinv(transfer_function)
        # Return the least squares solution for the new output CPSD
        output = tf_pinv@specification@tf_pinv.conjugate().transpose(0,2,1)
    else:
        # Scale the last output cpsd by the trace ratio between spec and last response
        trace_ratio = trace(specification)/trace(last_response_cpsd)
        trace_ratio[np.isnan(trace_ratio)] = 0
        output =  last_output_cpsd*trace_ratio[:,np.newaxis,np.newaxis]
    return output
```

Closed-loop control to specification Trace

```
def shape_constrained_pseudoinverse(transfer_function,  # Transfer Functions
                                    specification, # Specifications
                                    buzz_cpsd, # Buzz test in case cross terms are to be computed
                                    extra_parameters = '',
                                    last_response_cpsd = None, # Last Response for Error Correction
                                    last_output_cpsd = None, # Last output for Drive-based control
                                    ):
    shape_constraint_threshold = float(extra_parameters)
    # Perform SVD on transfer function
    [U,S,Vh] = np.linalg.svd(transfer_function,full_matrices=False)
    V = Vh.conjugate().transpose(0,2,1)
    singular_value_ratios = S/S[:,0,np.newaxis]
    # Determine number of constraint vectors to use
    num_shape_vectors = np.sum(singular_value_ratios >= shape_constraint_threshold,axis=1)
    # We have to go into a For Loop here because V changes size on each iteration
    output = np.empty((transfer_function.shape[0],transfer_function.shape[2],transfer_function.shape[2]),dtype=complex)
    for i_f,(V_f,spec_f,H_f,num_shape_vectors_f) in enumerate(zip(V,specification,transfer_function,num_shape_vectors)):
        # Form constraint matrix
        constraint_matrix = V_f[:,:num_shape_vectors_f]
        # Constraint FRF matrix
        HC = H_f@constraint_matrix
        HC_pinv = np.linalg.pinv(HC)
        # Estimate inputs (constrained)
        SxxC = HC_pinv@spec_f@HC_pinv.conjugate().T
        # Convert to full inputs
        output[i_f] = constraint_matrix@SxxC@constraint_matrix.conjugate().T
    return output
```

Shape Constrained Pseudoinverse Control

Alternative Control Law Definitions

Functions are useful, but some control laws require additional capabilities:

- Functions do not maintain state between calls, all internal data is lost when function returns
- For control laws with “set up” operations that should only be called once, need additional logic

Rattlesnake also accepts other control law definitions:

- Generator function – Maintains state between calls, but still requires additional logic to
- Class – Arbitrary data can be stored to and accessed from the object, functions can be defined that are called at various times in the controller to set up the control law.

```python
import numpy as np

class buzz_control_class:
    def __init__(self,specification : np.ndarray,extra_control_parameters : str,
                 transfer_function : np.ndarray = None,  # Transfer Functions
                 buzz_cpsd : np.ndarray = None, # Buzz test in case cross terms are to be computed
                 last_response_cpsd : np.ndarray = None, # Last Response for Error Correction
                 last_output_cpsd : np.ndarray = None, # Last output for Drive-based control
                 ):
        # Store the specification to the class
        self.specification = specification

    def system_id_update(self,transfer_function : np.ndarray, buzz_cpsd : np.ndarray):
        # Update the specification with the buzz_cpsd
        self.specification = self.match_coherence_phase(self.specification,buzz_cpsd)

    def control(self,transfer_function : np.ndarray,
                last_response_cpsd : np.ndarray = None,
                last_output_cpsd : np.ndarray = None) -> np.ndarray:
        # Perform the control
        tf_pinv = np.linalg.pinv(transfer_function)
        return tf_pinv@self.specification@tf_pinv.conjugate().transpose(0,2,1)

    def cpsd_coherence(self,cpsd):
        num = np.abs(cpsd)**2
        den = (cpsd[:,np.newaxis,np.arange(cpsd.shape[1]),np.arange(cpsd.shape[2])]*
               cpsd[:,np.arange(cpsd.shape[1]),np.arange(cpsd.shape[2]),np.newaxis])
        den[den==0.0] = 1 # Set to 1
        return np.real(num/
                       den)

    def cpsd_phase(self,cpsd):
        return np.angle(cpsd)

    def cpsd_from_coh_phs(self,asd,coh,phs):
        return np.exp(phs*1j)*np.sqrt(coh*asd[:,:,np.newaxis]*asd[:,np.newaxis,:])

    def cpsd_autospectra(self,cpsd):
        return np.einsum('ijj->ij',cpsd)

    def match_coherence_phase(self,cpsd_original,cpsd_to_match):
        coh = self.cpsd_coherence(cpsd_to_match)
        phs = self.cpsd_phase(cpsd_to_match)
        asd = self.cpsd_autospectra(cpsd_original)
        return self.cpsd_from_coh_phs(asd,coh,phs)
```

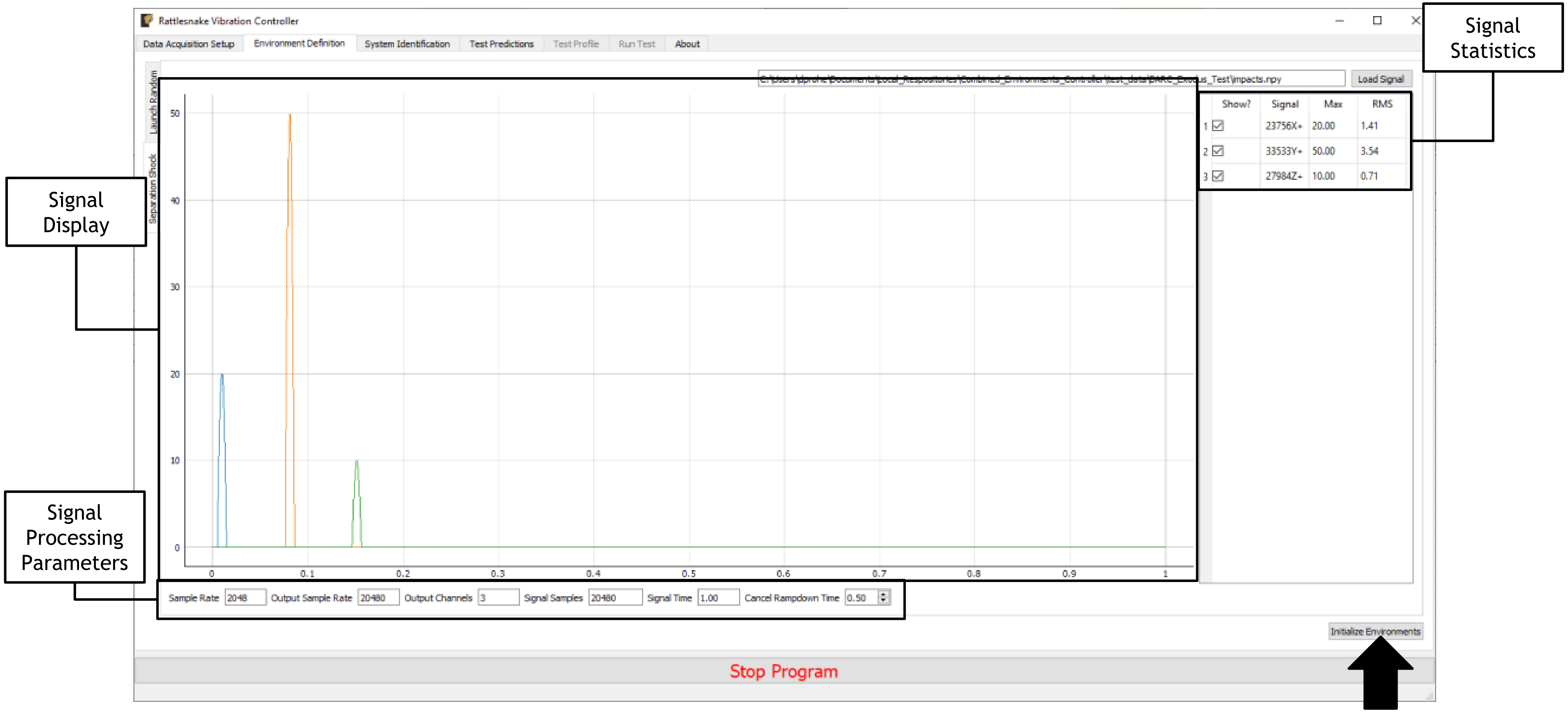
Defining control parameters – Time History Generation

We must now define our second environment, which is a simple time history generator

Load in signal from a Matlab (.mat) or Numpy (.npy or .npz) function

Signal will be output as is to the exciters

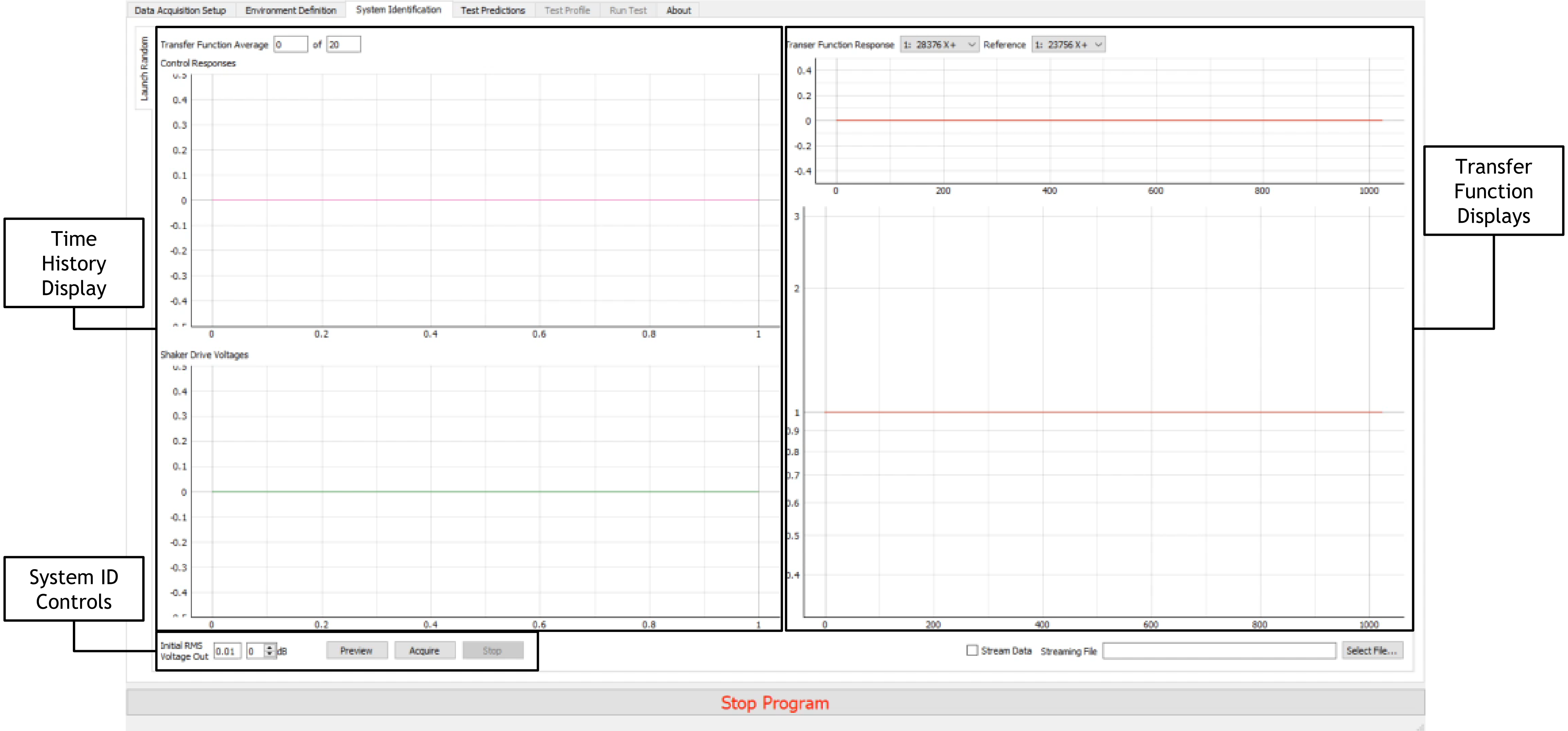
When finished we initialize the environments which sends all environment parameters to the various portions of the controller



System Identification

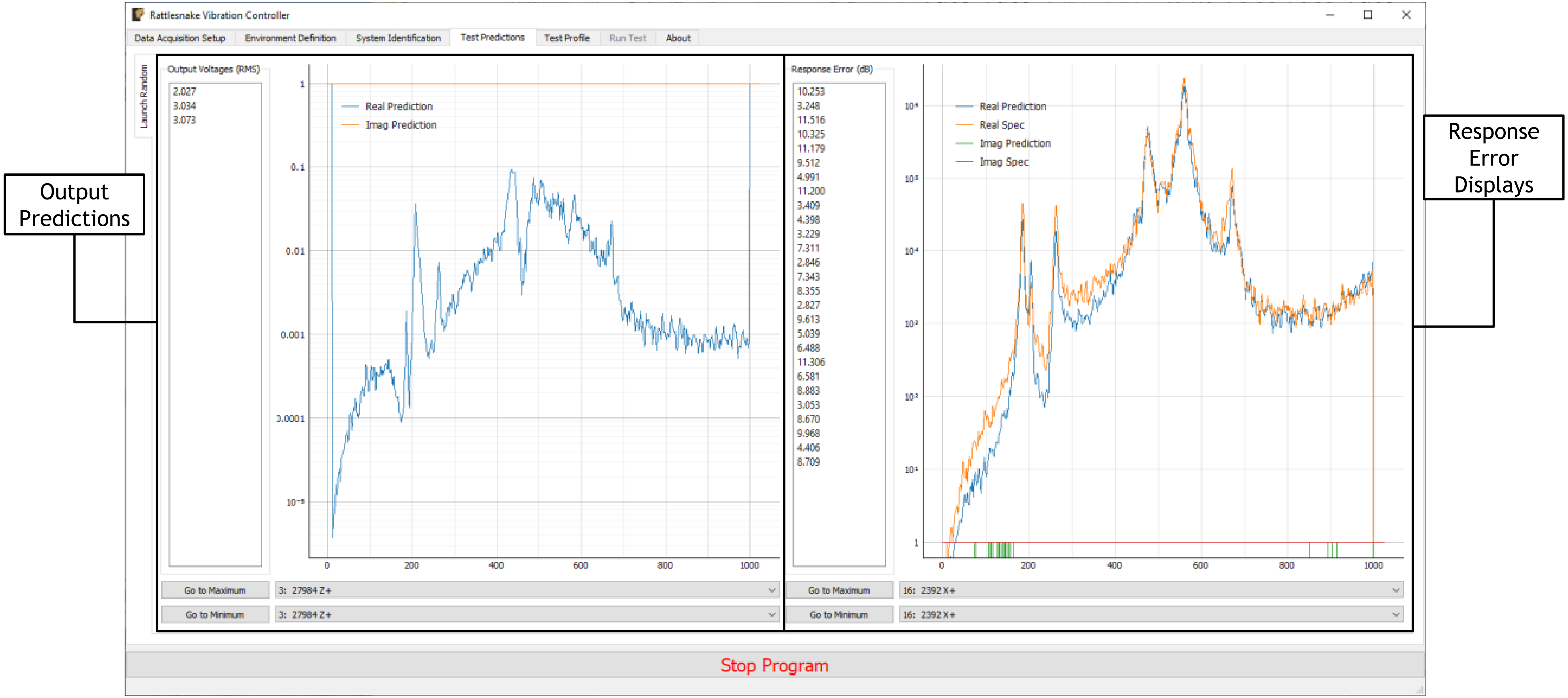
With environments defined, Rattlesnake will perform a System Identification

- Only specific environments require system identification



Test Predictions

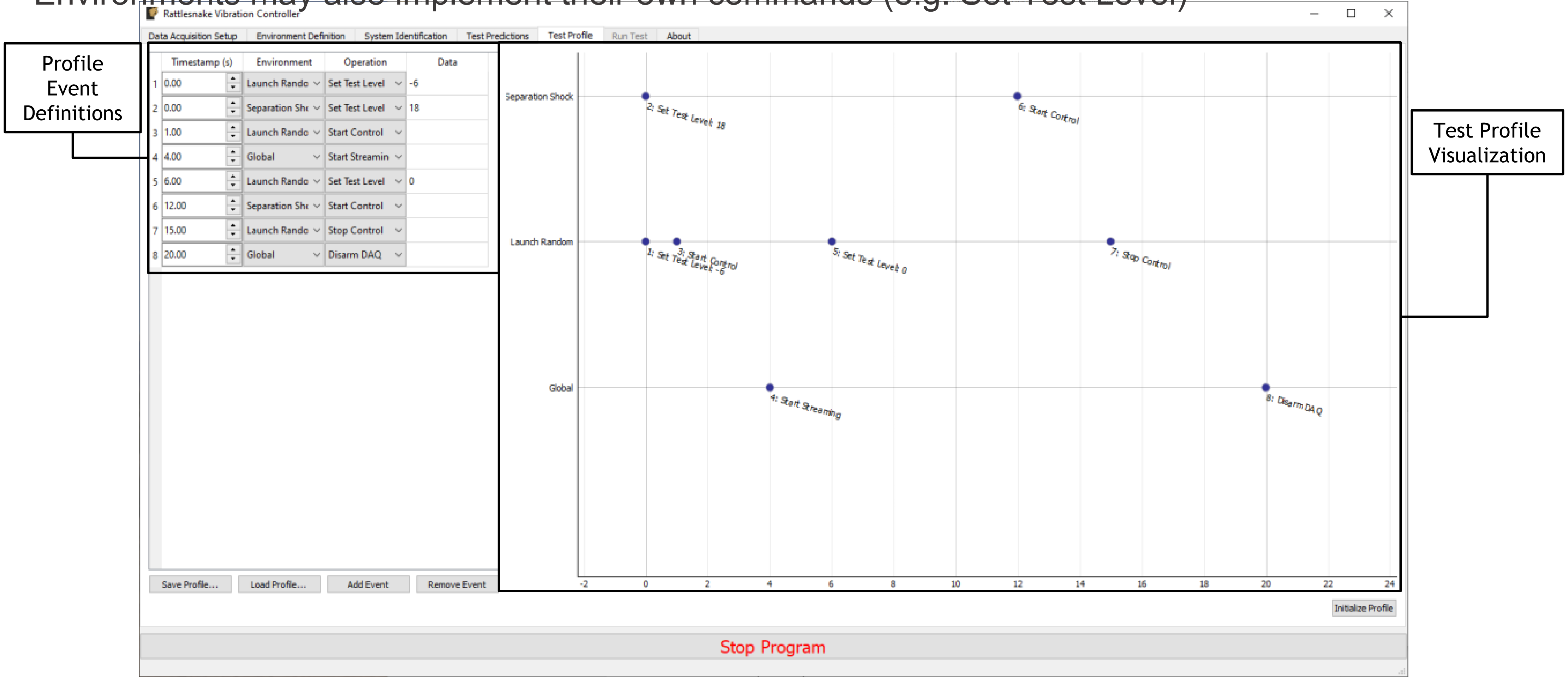
With the system identification, environment parameters, and control law defined, Rattlesnake can make predictions for the given environment.



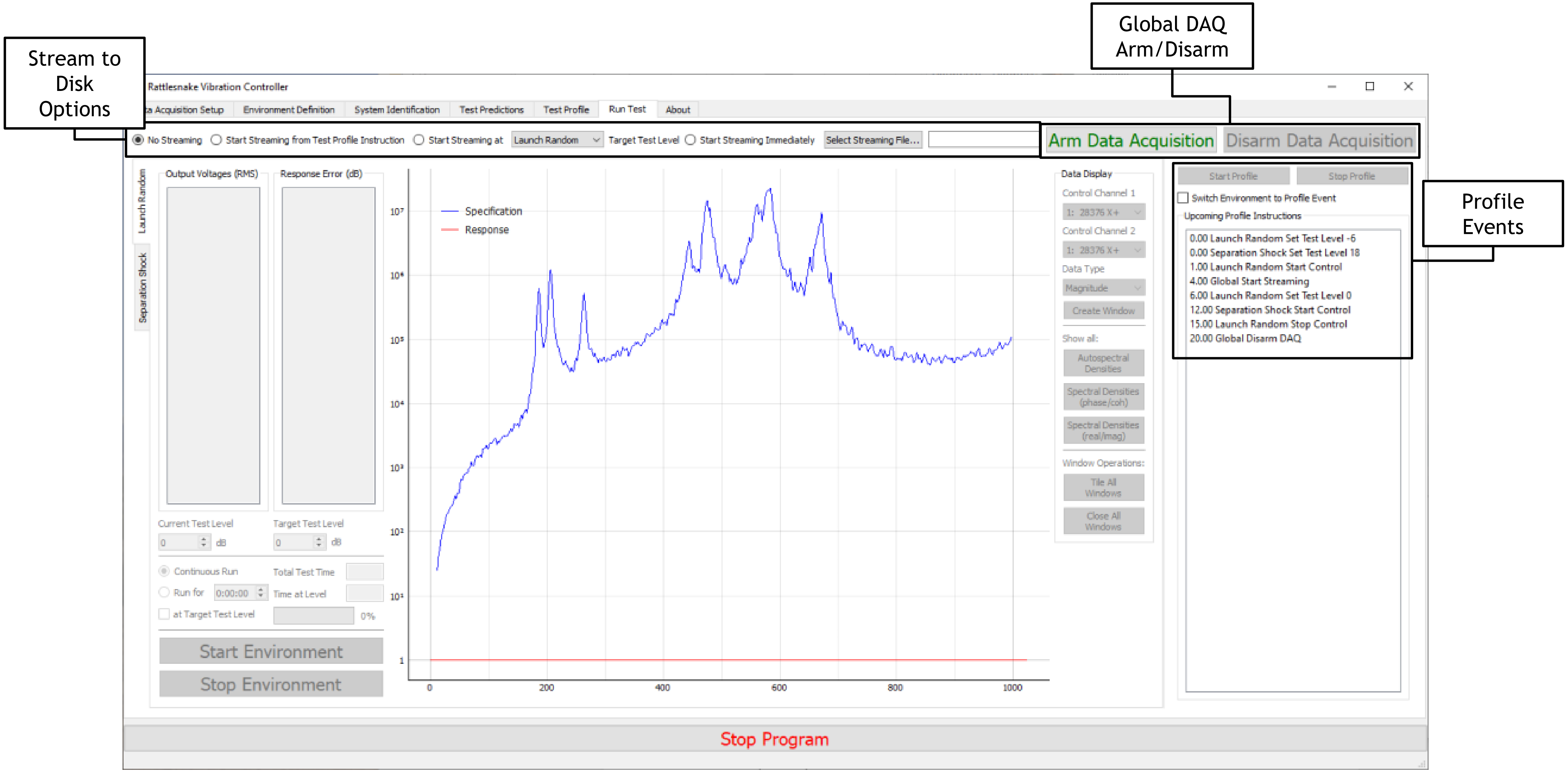
Test Profile

Combined environments tests can be complex, so Rattlesnake allows users to set up test timelines.

- All environments can be started or stopped automatically
- Environments may also implement their own commands (e.g. Set Test Level)



Running the Test – Pre-arm



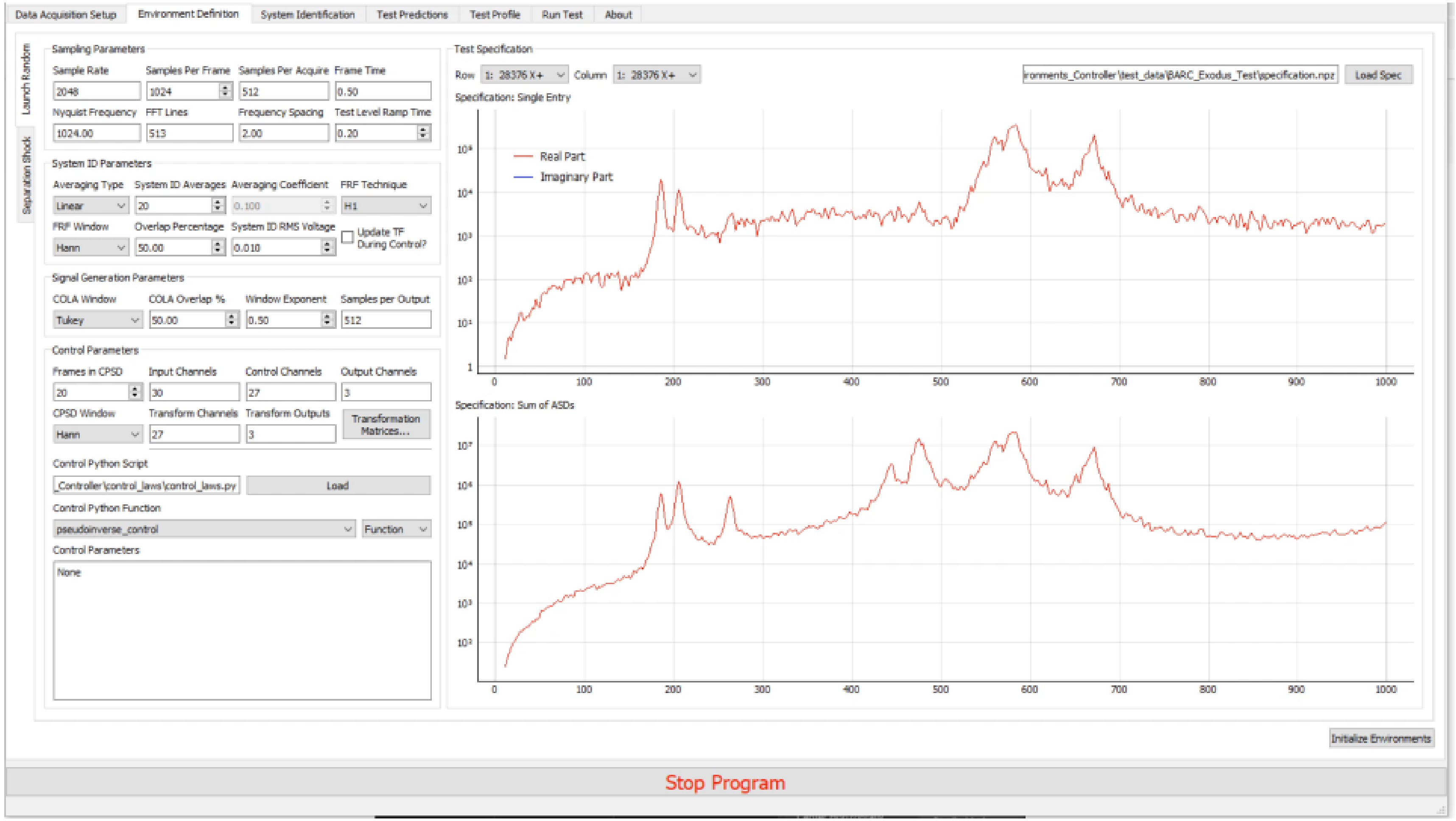
Running the Test – Running the Profile

Key Rattlesnake Strengths

Developing New Control Laws

Rattlesnake makes it easy to implement new control laws into its framework

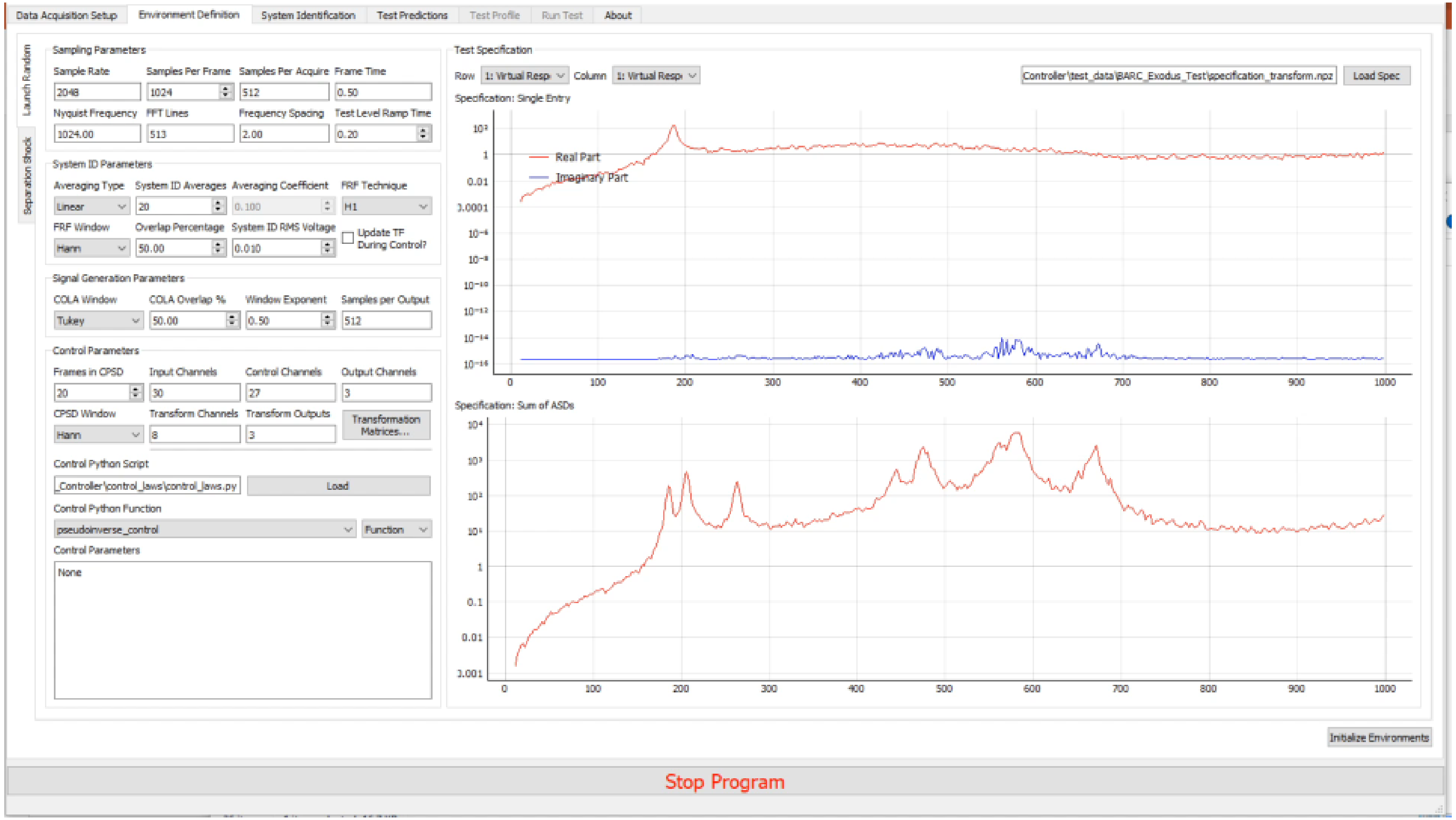
- Generally less than 20 lines of code to produce even complex, closed-loop control laws.
- Can investigate effects of control laws virtually prior to fielding on real systems



Response and Output Transformations

Rattlesnake implements capabilities for response and output transformations.

- Can implement 6-DoF and Modal Transformations
- Example Use Case: For test articles with significant unit-to-unit variability, may make sense to control to “modes” than to control to “responses”. Target modes can be moved through the bandwidth to accommodate variations in the test article.



Applications

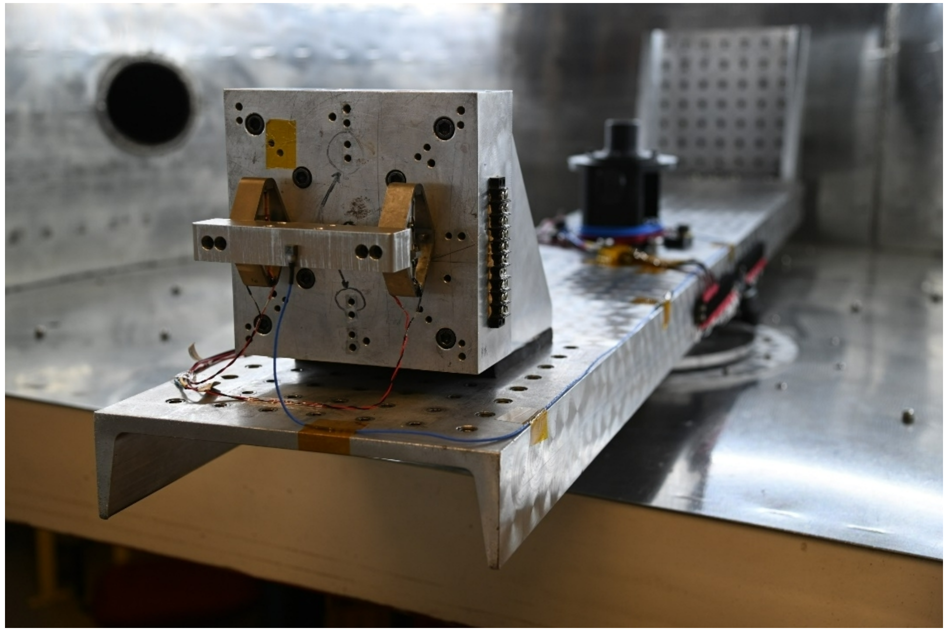
Component Vibrafuge – Controlling combined environments with Rattlesnake

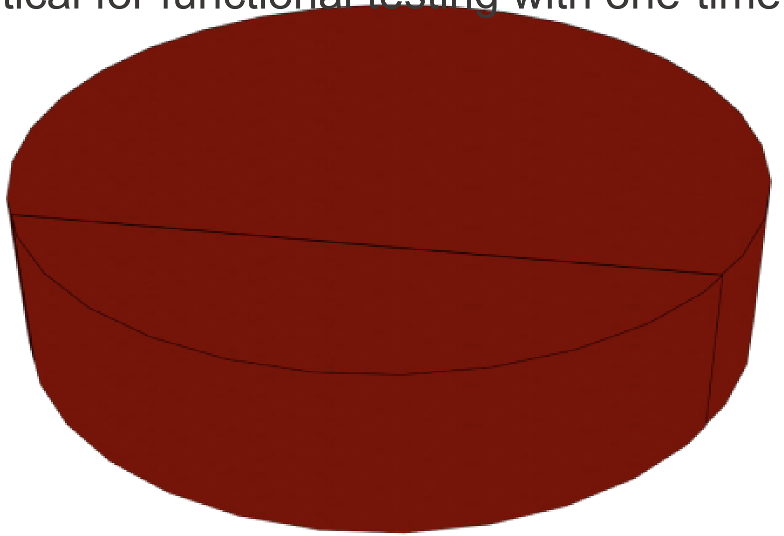
Proof-of-concept combined environments testing was performed using the Component Vibrafuge as a test-bed.

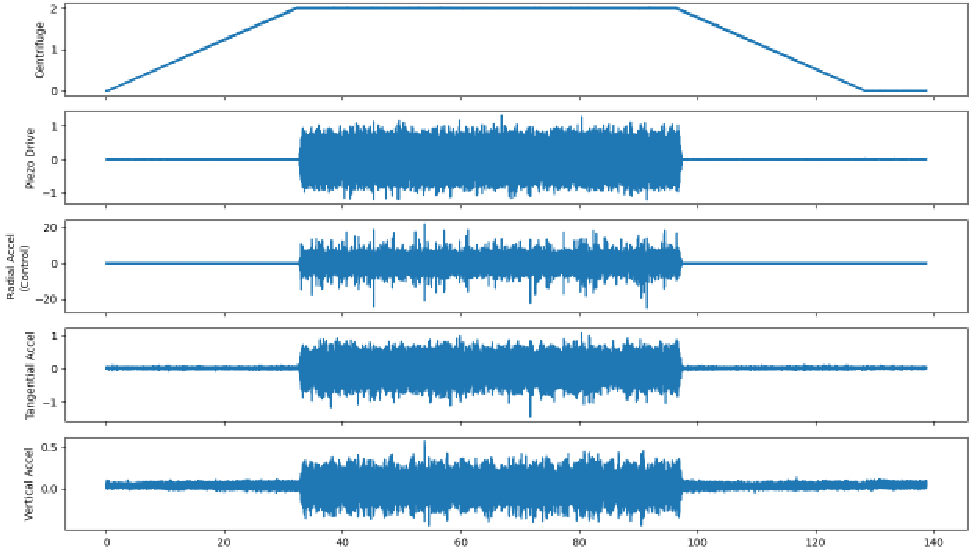
- **Centrifuge environment** controlled using open-loop time signal generation capabilities
- **Vibration environment** controlled with Random Vibration capabilities
- Used test profile capabilities to synchronize environments, run vibration when centrifuge hits test level.

One-click running of test profiles greatly simplifies combined environments tests.

- Critical for functional testing with one-time-use items







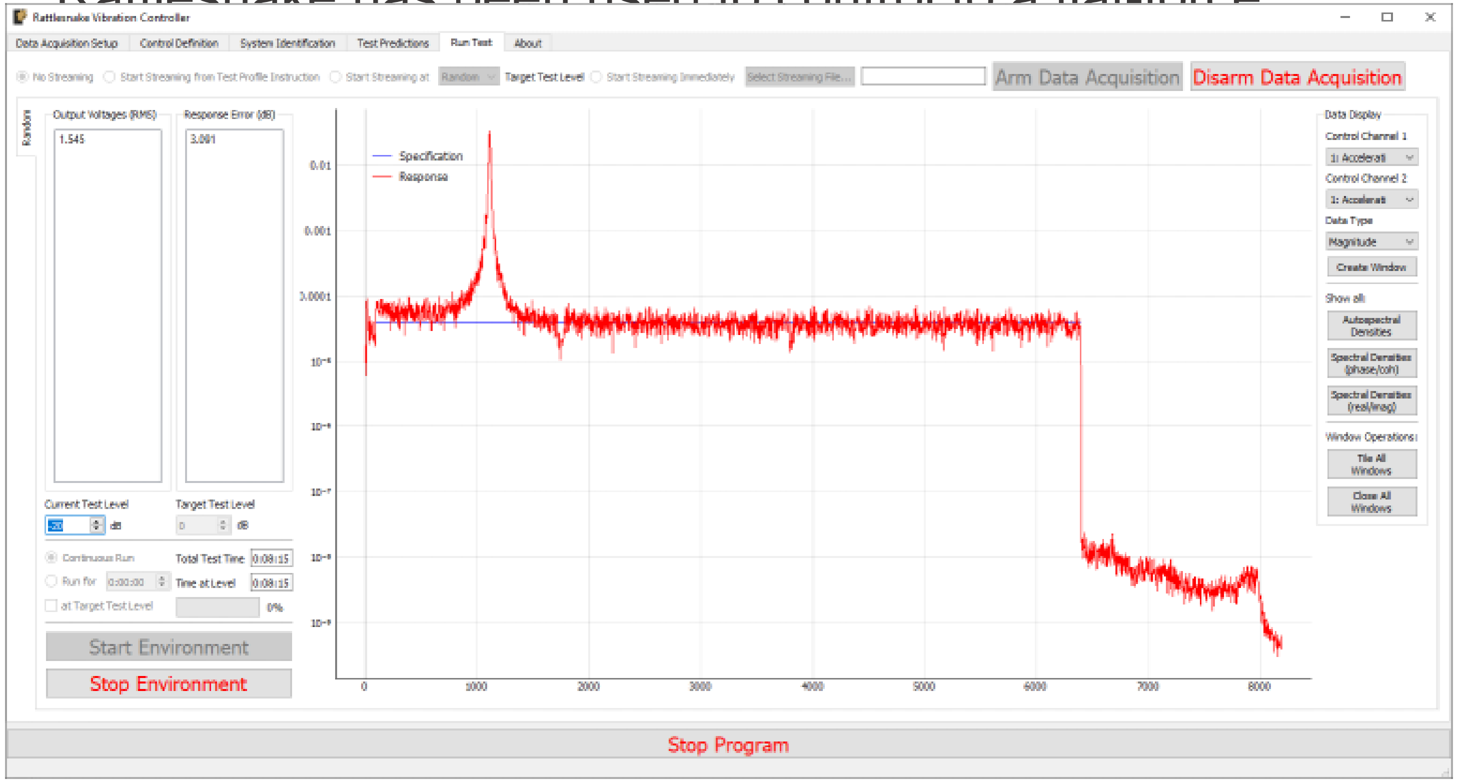
PI: David Siler

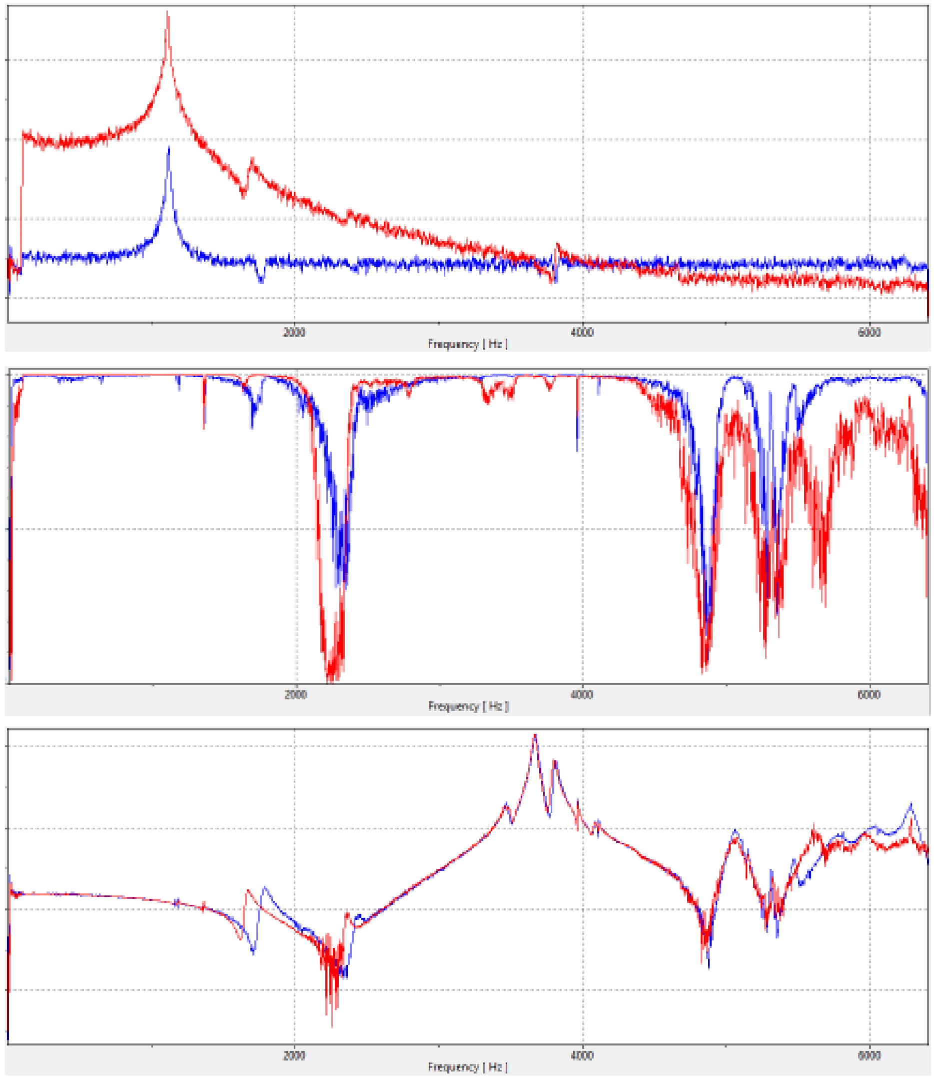
Modal Testing – Tuning excitation for tests

A **shaped input spectrum** might be necessary for modal testing:

- DIC – Limited displacements at high frequencies, need to increase force to compensate
- High-frequency modal – Shaker force falls off after shaker resonance.

Rattlesnake has been used to control to a flat-force





6DoF + Modal Shakers – Flexible capabilities enable non-standard test configurations

6 Degree-of-Freedom (6DoF) shaker can excite to much higher levels than small modal shakers, but base excitation lacks control authority.

Supplement 6DoF with modal shakers to “fill in the gaps” of the 6DoF test.

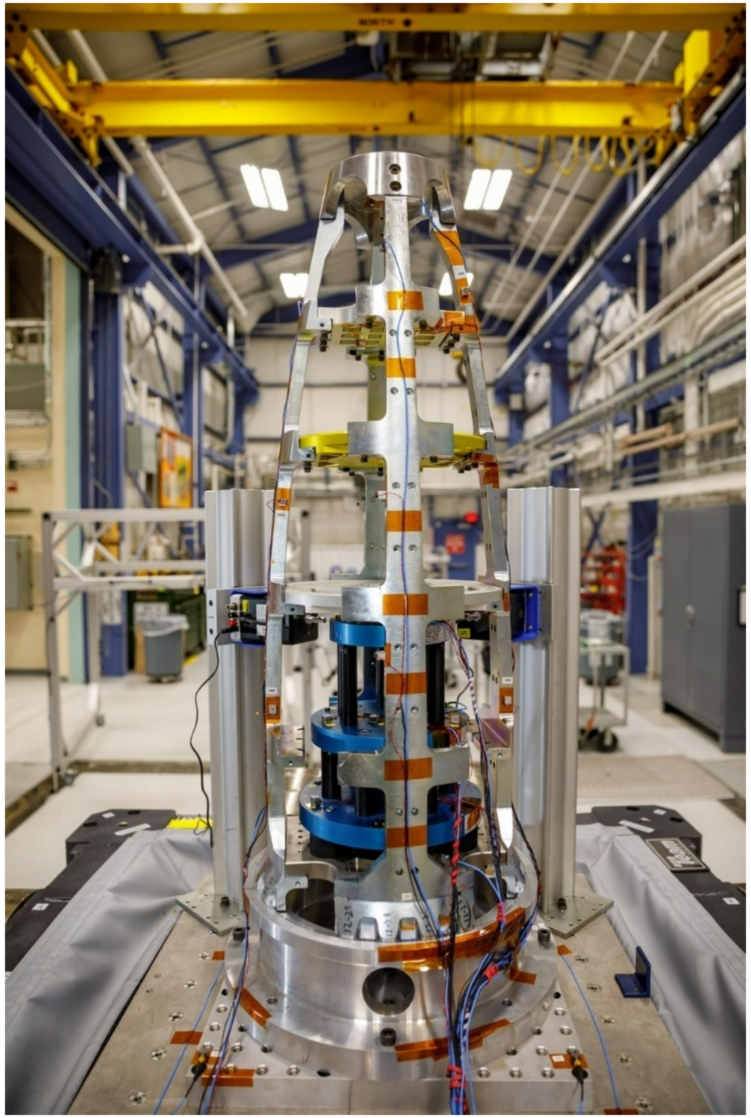
Issues with initial commercial controller resulted in trying out Rattlesnake.

Control problem consisted of 14 drives controlling 21 control accelerometers.

Wrote custom control law for this test combining shape constraint and buzz test approaches.

Able to demonstrate improved responses due to additional shakers:

- Initial 6-DoF test did not excite the torsion of the Wedding Cake sufficiently
- Additional modal shakers targeting that motion improved response.



PI: Kevin Cross and Brandon Zwink

Conclusions

Rattlesnake provides a flexible framework to perform MIMO and Combined Environments research and development.

Integration of custom control laws enable rapid iteration on new control strategies
- Researchers don't need to write their own complete control systems
- Researchers don't need vendors to implement ideas into their proprietary control systems

Flexible environment interface allows extension of Rattlesnake to new environments
- New environments need only implement the interface defined by Rattlesnake
- Multiple environments can be run simultaneously, with a test timeline available for repeatable testing.

Flexible hardware interface allows extension of Rattlesnake to new hardware devices
- Currently supports two data acquisition programming interfaces (NI DAQmx and LAN-XI OpenAPI)
- Can also perform testing on synthetic "hardware" by integration of modal equations of motion or state space matrices

Rattlesnake is already being used for research applications at Sandia National Laboratories
- Has been used to combine vibration and centrifuge tests
- Has been used on tests with up to 275 channels (~50 control) and 12 shakers

You can use Rattlesnake too!
- Download at: https://github.com/sandialabs/rattlesnake-vibration-controller

Questions?