View metadata, citation and similar papers at core.ac.uk
brought to you by CORE
provided by NASA Technical Reports Server

A Cryogenic Fluid System Simulation in Support of Integrated Systems Health Management

John P. Barber[1], Kyle B. Johnston[2], and Matthew Daigle[3]

[1]SGT, Inc., Kennedy Space Center, FL. 32899, USA
john.p.barber@nasa.gov

[2]ENSCO, Inc., ASE, Kennedy Space Center, FL. 32899, USA
kyle.b.johnston@nasa.gov

[3]NASA Ames Research Center, Moffett Field, CA. 94035, USA
matthew.j.daigle@nasa.gov

ABSTRACT

Simulations serve as important tools throughout the design and operation of engineering systems. In the context of systems health management, simulations serve many uses. For one, the underlying physical models can be used by model-based health management tools to develop diagnostic and prognostic models. These simulations should incorporate both nominal and faulty behavior with the ability to inject various faults into the system. Such simulations can therefore be used for operator training, for both nominal and faulty situations, as well as for developing and prototyping health management algorithms. In this paper, we describe a methodology for building such simulations. We discuss the design decisions and tools used to build a simulation of a cryogenic fluid test bed, and how it serves as a core technology for systems health management development and maturation.

1. INTRODUCTION

In modern systems engineering practices, modeling and simulation serve as foundational elements throughout the design process. Systems health management (SHM) technologies, which focus on monitoring system behavior, detecting faults and other anomalies, isolating and identifying faults, and predicting component failures and other significant events, all rely on some type of system model. System simulations capable of modeling both nominal and faulty behavior can help in developing these models and in testing and validating SHM algorithms, and have an additional application for operator training with failure scenarios.

Barber et al. This is an open-access article distributed under the terms of the Creative Commons Attribution 3.0 United States License, which permits unrestricted use, distribution, and reproduction in any medium, provided the original author and source are credited.

Simulations can effectively serve as virtual testbeds. For development and validation of SHM algorithms, such simulation testbeds are extremely useful since validation requires injecting faults, which is often difficult, costly, or unsafe to perform on real systems. In (Poll et al., 2007) an electrical power distribution system testbed and its corresponding simulation testbed are described. In (Balaban et al., 2013) a simulation testbed for a planetary rover is described. In (Goodrich et al., 2009) a simulation testbed for a cryogenic fluid system is discussed. Each of these simulation testbeds have the ability to inject faults and are used for development and prototyping of health management algorithms. Other examples of simulation-based SHM include (Agusmian, 2013) and (Biswas, 2007).

We describe in this paper the development of another simulation package for a cryogenic fluid system, extending in many respects the preliminary work presented in (Goodrich et al., 2009). The simulation is being developed for a cryogenic testbed (CTB) that, through a network of pipes, valves, pumps, and filters, transfers liquid nitrogen from a storage tank to an external tank representing that of a space vehicle. The purpose of the CTB is to mature SHM technologies for ground systems operations. Developing a simulation for this system presents many challenges, due to the large number of components, the large number of possible system modes, and complex two-phase physics.

This paper focuses on the development of the CTB simulation software, named CryoSim. We discuss the tools used to build the simulation model, and how the challenges of building such a simulation are addressed. The system architecture used for CryoSim is both model and domain agnostic. It can be easily adapted for use with other system models and simulation domains, thus serving as a general architecture for designing virtual testbeds for SHM purposes.

In Section 2, we present a brief overview of the SHM system and other factors that motivated the development of CryoSim. In Section 3, we discuss the internal architecture of CryoSim and present the design methodology used to develop the model. Section 4 details the external interfaces we have developed for CryoSim to facilitate its use as a virtual testbed. We present a number of results in Section 5 to illustrate the key features of CryoSim and its use in an SHM context. Section 6 concludes the paper.

2. BACKGROUND

CryoSim is one element of an integrated SHM system being developed for the CTB. The SHM system architecture consists of a set of health management tools connected via a message bus based on a publish/subscribe protocol. During operation, the target system periodically publishes a set of messages containing system measurement data (pressures, temperatures, flow rates, etc.). The health management tools receive this data, perform analysis, and then publish messages indicating the health status of the monitored system.

CryoSim was developed as a drop-in replacement data source for the CTB. CryoSim uses a simulation to produce data which accurately represents a range of system behaviors, including fault scenarios. This data is then published to the message bus using the same protocols as the CTB. In this manner, CryoSim acts as a virtual testbed, enabling the development and testing of a suite of health management tools without large numbers of costly test runs on the CTB.

The architecture and implementation of CryoSim described in this paper were primarily motivated by its intended use as a testbed for SHM systems. This application drove the development of features such as component-level fault simulation and the message bus interface, which we discuss in later sections. The possibility of using a simulator package like CryoSim as a data source for other systems dictated the modular, model-agnostic design approach that we followed. Additional considerations during development included its potential application as an operator training environment, and the ability to support varying levels of simulation fidelity with the same model. The remainder of this paper describes the methods we used to ensure that CryoSim could meet this set of objectives.

3. CRYOSIM ARCHITECTURE

This section describes the architecture of CryoSim, starting with an overview of its modular architecture. We then discuss key elements of the approach we used to develop the system model. Finally, we describe the operation of the initialization and control modules, and how they interact with the model and the external interfaces present in CryoSim.

3.1. Overview

The CryoSim software was developed to meet the following objectives:

- Provide a medium-fidelity, multi-domain system model incorporating cryogenic fluid flow elements, a pneumatic actuation system, and an I/O and control system
- Model both nominal system behavior and the effects of any of a discrete set of failure modes injected at any location in the system
- Publish model input and output signal values to a message bus interface using the same protocols as the CTB system
- Allow user specification of input signals, model parameters and fault injection commands
- Record all simulation data, including inputs, parameter values, outputs and status/warning messages to a file for offline analysis
- Provide an interactive graphical interface allowing full control over the simulation environment, including system inputs and fault injection

In order to support the various use cases and configurations of CryoSim, the package was developed using a modular architecture. The core consists of two required modules: the CTB system model which is implemented as a hierarchical Simulink® model, and an initialization and control module consisting of a set of MATLAB® functions. If desired, the message bus and GUI modules can also be enabled for a simulation run or set of runs, but are not required to access any of the core functionality of CryoSim. Figure 1 shows a block diagram of the CryoSim architecture.

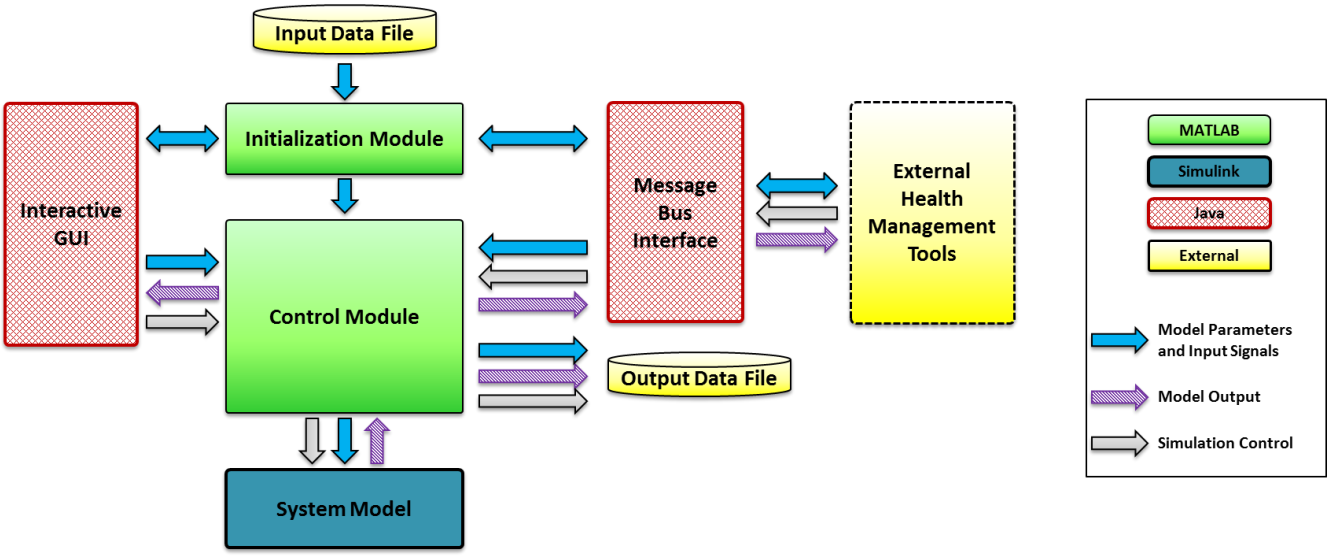


Figure 1. CryoSim block diagram

3.2. System Model

The CryoSim system model is a variable-fidelity, multi-domain physics-based model of the CTB. The model was developed using Simulink, without dependencies on additional toolboxes. We developed a set of component libraries to represent the various physical domains included in the system. Interaction between elements in different domains is incorporated in the component designs. The current implementation has libraries for cryogenic fluid flow, pneumatics, and electrical systems (including transducers and system I/O). Library components are instantiated in the model and connected to match the topology of the physical system. Related sets of interconnected components are organized into subsystems, which are connected together to form the complete model.

3.2.1. Component-based Design

The CryoSim system model is composed of a set of component models which are connected to match the CTB system topology. The component models are parameterized representations of a component's behavior in both nominal and faulty operating regimes. This methodology allows a single component model residing in a library to be used in the system model to represent a number of similar physical components, each having unique physical characteristics and behavior. For example, the library component used to represent a pneumatically actuated control valve, shown in Figure 2, has parameters describing the orifice diameters and flow coefficients of both the fluid flow path and the pneumatic actuator. Instances of this component are used in the model to represent valves with different geometries, simply by changing the parameters used for each component.

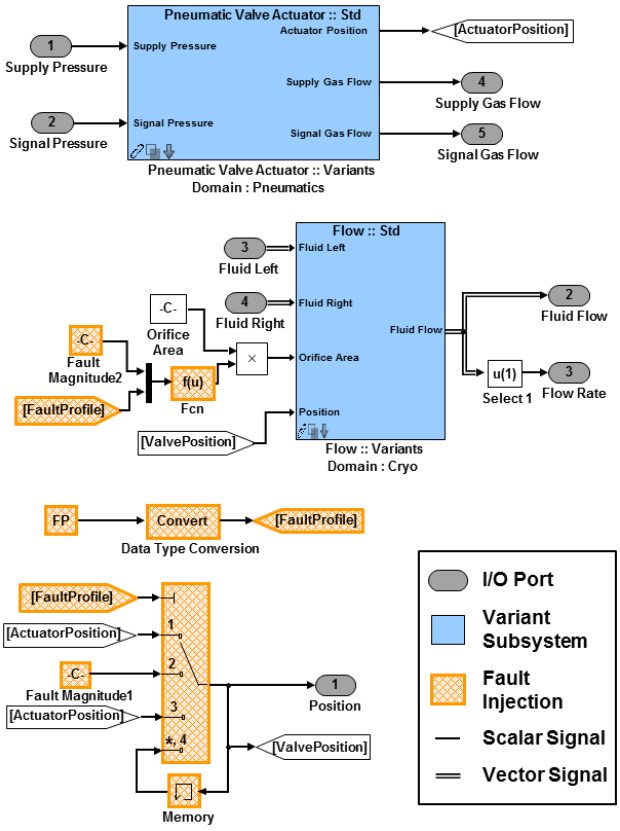


Figure 2. Control valve (CV) component model

3.2.2. Variable Model Fidelity

A core feature of CryoSim is support for different levels of simulation fidelity without requiring a user to make changes to the system model. We accomplish this by implementing multiple component or subcomponent-level models, each

providing a different level of fidelity and corresponding computational burden. The models can range from a non-computing element such as a constant output or signal pass-through, to a low-order model based on empirical behavior, to a high-fidelity model based on the underlying physics of the component being modeled. Our approach uses the variant subsystem functionality in Simulink to implement this behavior. When using variant subsystems, each component or subcomponent may have one or more variant instances, each representing a different model of the component's behavior. Before a simulation is run, one of the set of possible variants for each component is selected and made active for the simulation, while the remaining variant instances are disabled. This implementation allows the end user to select the desired simulation fidelity at run time using a single parameter, without the possibility of errors introduced by editing the model.

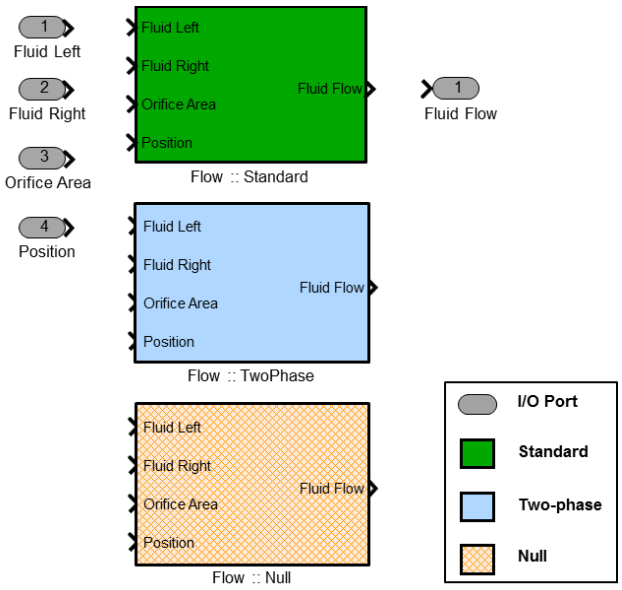


Figure 3. Internal block diagram of a variant subsystem

Figure 3 shows the internal configuration of the variant subsystem used in our model to compute fluid flow, which is representative of the cryogenic fluid domain in our model. The variant subsystem contains three variant instances: a standard fluid flow model, a two-phase fluid flow model, and a null model. The standard fluid model is based on the Bernoulli equation for laminar, incompressible, inviscid flow (Granger, 1995). This option provides a computationally-efficient model which yields usable accuracy for our SHM system when used to simulate the post-chilldown phase of the CTB system's operation, where the flowing fluid exists primarily in the liquid phase. However, it is not accurate for operating regimes such as system chilldown, where the cryogenic fluid is in a mix of liquid and vapor phases.

The two-phase fluid flow model is based on a stratified flow approximation that assumes the gas and liquid are split into two layers with gas on top and liquid on the bottom. The model considers heat transfer with the walls and between the layers, including evaporation/condensation and boiling. The two-phase model provides a much higher level of fidelity, especially in the chilldown phase where there are large temperature variations in the system resulting in significant quantities of liquid being converted to vapor. The tradeoff associated with the two-phase model is increased computation time for higher model fidelity compared to the standard flow model.

The null component variants for the cryogenic fluid library consist of signal terminations on the input ports, and output ports set to constant values. The null variants are effectively empty blocks that require no computation during a simulation. The use of null variants allows unneeded portions of the system model to be disabled for a given simulation run, resulting in significant increases in simulation speed. In general, care must be taken when implementing the null variants in order to provide appropriate boundary conditions for the non-null portion of the system. For example, the null variant for a fluid tank connected to a pipe network should provide output signals representing a static state (pressure, flow, temperature, etc.), rather than null or grounded signals.

It should be noted that the effective use of variant subsystems requires that each variant instance for a given component have the same connectivity. We implement this by using vector-based signals to connect component instances. For example, two scalar signals are needed for the variables representing cryogenic flow in the standard flow model, while four signals are needed for the two-phase model. Similarly, the variables needed to describe a fluid element require either two or seven scalar signals. We combine the groups of scalar signals into vector signals, and use mux/demux blocks at the input and output ports of the variant instances to route the signals internally, as shown in Figure 4. Signals that are not needed by a particular set of variants are grounded or terminated inside of the variant instances. To avoid problems associated with Simulink's ability to propagate signal data types and dimensionality, it is good practice to explicitly specify signal properties at the input and output ports of each component variant. When implemented in this way, changing from one set of variants to another does not require any changes to the model's topology. This allows an end-user to safely change the model's variants and simulation fidelity without the risk of altering connections within the model, and without requiring the model designer to maintain separate system models for each set of variants.

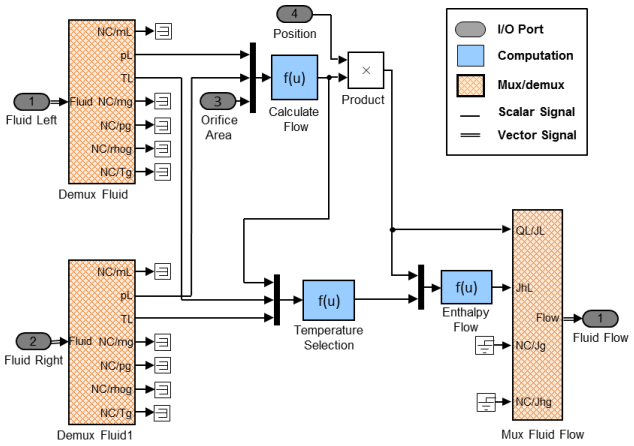


Figure 4. Block diagram for the Standard Fluid Flow variant

The use of variant subsystems easily extends to mixed-domain components. The component model of a pneumatically-actuated control valve shown in Figure 2 includes elements from both the cryogenic fluid flow and pneumatics domains. Because the variant subsystems are implemented as domain-specific sets of variant instances, the control valve model contains two variant subsystems, which can be changed independently.

Our implementation allows the user to specify desired variants via a set of control parameters. For each simulation domain, a global variant parameter determines the particular variant implementation for that domain. Thus, a user desiring a high-fidelity simulation of the cryogenic domain that does not require high fidelity in the pneumatics domain can simply specify the appropriate variant control parameters before running a simulation.

A second group of variant controls in CryoSim allows particular segments of the system to be toggled between the usual domain-specific variant components and null components. If a particular segment is not required for a simulation, the components in that segment can be set to use the null variant instances instead of the normal component models, reducing the computational load required to model the entire system. Implementing this functionality in CryoSim approximately doubles the simulation speed in cases where all unneeded segments are disabled.

3.2.3. Fault Modeling

In order for the simulation data to accurately represent system faults, the desired failure mechanisms must be incorporated into the component designs. During a simulation, when a given fault is injected onto a particular component, the effects of the fault will automatically propagate through to the entire system in accordance with the underlying model behavior. The fault mode, magnitude and injection time are implemented as parameters for each component, enabling fine-grained control over fault behavior in the model. Faults can be injected or cleared either before a simulation is run, or during the run.

In keeping with our system model's design, the implementation of a specific fault mode will depend on the desired level of fidelity. For many of our component models, a multiport switch is introduced in the path of a signal of interest, allowing different transformations to be applied to the signal depending on the selected fault mode. An advantage of this approach is that new fault modes can be added to a component without requiring any rework of the model or other library components. Similarly, a higher fidelity representation of a particular fault can be incorporated into a component if there is a specific need.

The control valve component model, shown in Figure 2, illustrates this approach to fault modeling. The blocks used to implement the fault modes are shown with a hatched background. This component model has four available fault modes:

1. Nominal behavior: when this mode is active, the actuator position calculated by the "Pneumatic Valve Actuator :: Variants" block is passed without modification to the "Flow :: Variants" block.
2. Stuck pneumatic actuator: in this fault mode, the position of the pneumatic valve actuator is set to a fixed value, regardless of the value of the controlling "Signal Pressure" input. When this mode is active, the calculated actuator position is ignored. A user-determined fault magnitude parameter is used instead, originating from the "Fault Magnitude1" block shown in the diagram.
3. Blockage: this fault mode models an obstruction in the fluid flow path of the valve. The pneumatic actuator position is not affected by this fault mode, so it is passed through to the "Flow :: Variants" block. To model the effects of the blockage, the nominal valve orifice area is scaled down by multiplication with the user-determined fault magnitude parameter.
4. Frozen: this fault mode represents a condition where the pneumatic actuator does not respond to its controlling input signal, similar to the "Stuck" fault described above. However, in the "Frozen" mode, the actuator position is held to its value immediately prior to the fault mode becoming active. This can be seen in Figure 2 as the "Memory" block at the bottom of the diagram.

As noted earlier, the approach we have used for fault modeling allows the addition or modification of fault modes for a given component with no impact to the normal behavior of the component or overall system. The complexity of any particular fault mode's implementation is determined by the model designer. A simple low-order approximation can be used for faults that do not require high-fidelity modeling, such as the "Stuck" and "Frozen" fault modes for the CV

component. For these faults, we do not model the mechanics of a failure within any particular type of pneumatic actuator. Instead, we approximate the behavior of the actuator in a manner that minimizes the computational resources needed for the fault models. For fault modes where increased fidelity is desired, the fault can be incorporated into the physics of the affected subsystem, such as the "Blockage" fault for the CV component and the statistical wear model used for the filter component described in Section 5.4.

3.2.4. Extensibility

It is anticipated that CryoSim will be required to provide simulation data representing different configurations of the CTB system. For example, a valve replacement or rerouting of pipes in some segment of the system would constitute a modified configuration that would require corresponding changes to the model. The component-based design is well-suited to this requirement, as changes to a particular subsystem can be made by adding or removing that subsystem's components and connecting them to match the new system topology.

Additionally, the component libraries allow the rapid creation of models of other systems that utilize the same component types. The modular nature of the CryoSim system architecture, which separates the model from the initialization, control and external interfaces, allows virtually all of the supporting code to be reused for a new system model without modification.

3.3. Initialization Module

The initialization module is responsible for setting up the simulation environment and external interfaces before a simulation is run. To support the multiple interfaces and use cases for CryoSim, the initialization module must provide a number of entry points while enforcing consistent behavior throughout the simulation process. That is, a simulation controlled through the GUI must accept the same inputs and provide the same outputs as a standalone simulation run or one controlled via the message bus. Additionally, the initialization module must validate user-supplied input signals and parameters, and ensure that the set of parameters and signals presented to the simulation module is complete and well-defined. CryoSim uses the base MATLAB workspace to store and process initialization and simulation data, allowing both the initialization and control modules to interact with the system model (via internal MATLAB/Simulink system calls), and the external interfaces (via MATLAB/Java interaction, described later).

3.3.1. Input Data

To ensure that all signals and parameters are defined prior to a simulation run, the initialization module loads a default configuration file, which contains all of the required values. If desired, a user-defined input file can be loaded after the default data file. The initialization module first validates the user-specified data for type, range and dimensionality. The validated data is then merged into the default configuration, with user-specified values always taking precedence over the default values. To avoid unexpected simulation results, the user is notified if the data in their input file is either incomplete or contains invalid entries.

3.3.2. Batch Mode

One important use-case of the CryoSim package is running sets of simulations to generate data for parametric and Monte Carlo analysis. To spare a user from the effort needed to generate unique input files for each set of desired parameter values, the initialization module includes an interface to allow a user-supplied calling function to execute the full initialization procedure once and then run a set of simulations. For each simulation run within the set, the calling function passes an arbitrary set of parameter and signal values which are used to override the default values for that run. This enables a simple user-supplied script to run a batch of simulations with a unique combination of parameter and signal values for each run.

3.4. Control Module

The control module is implemented as an "Interpreted MATLAB function" block inside of the Simulink system model. This function is executed at a predetermined rate as part of the simulation process, and has access to the model's input and output signal values during the simulation. Additionally, because it executes as a MATLAB function, the control function has access to both the base MATLAB workspace and the external GUI and message bus Java interfaces.

During each iteration of the control function, new values for the model's control signals and fault status are read from tables in the base MATLAB workspace. The module then queries the GUI and message bus interfaces (if present) for any user-generated control signals or fault commands, which are merged into the default tables. Values received from these interfaces always take precedence over the default values generated by the initialization module using the input file. The updated control signals and faults are then sent to the model, and the model's current input and output values are published to the GUI and message bus interfaces.

Another important function of the control module is the ability to control the real-world execution speed of the simulation by comparing the simulation clock to a system clock, and introducing appropriate delays if necessary. This functionality allows the CryoSim module to act as a substitute data source for the actual CTB system, which publishes measurements and system input values at one second intervals. Correct behavior in this mode requires that the host computer can execute the simulation at a rate of at least one

simulation time step per real-world time step. For simulation runs used for offline data generation and model development, the execution speed control can be disabled, allowing the simulation to run as fast as possible in a given computing environment.

Upon termination of the model's execution, a cleanup function is executed. This function creates an output file which can be used for offline analysis and to meet data retention requirements. The output file contains all of the data needed to reproduce the simulation run (model parameters, input signals and injected faults), as well as the simulation outputs and message log generated by the CryoSim module. With the exception of the simulation outputs and message log, the data structure of the output file is identical to the input file. Thus, a particular simulation can be re-run simply by stripping these tables from an output file, then using it as the input file for a new simulation.

4. EXTERNAL INTERFACES

4.1. Overview

We have developed two external interfaces to integrate CryoSim into our SHM system architecture. First, we created an interactive GUI that enables full control of the simulation, from starting and stopping a simulation, to injecting faults and changing model parameters and signals. The second interface consists of an adapter used to connect to the message bus interface used by the SHM system. This adapter allows CryoSim to publish simulation data onto the bus in a manner identical to the physical system it models, functioning as a virtual test bed for the SHM system.

Both interfaces were implemented in Java®, making use of MATLAB's ability to directly access Java objects and methods. The use of Java provides two key advantages. First, by developing Java classes to handle the computations needed for the GUI and message bus, we reduce the computational burden on MATLAB, enabling faster simulation speeds. This improvement in performance is primarily due to the greater control over threading available in Java. The second advantage is the availability of commercial GUI toolkits for industrial controls and systems. We used the GLG Toolkit (www.genlogic.com) for CryoSim, which minimized the effort needed to produce the GUI.

4.2. Graphical User Interface

The CryoSim GUI was designed to meet the following objectives:

- The GUI can control simulation execution, including start/stop/pause commands and control of execution speed.
- The GUI displays the current values of the simulation's output signals, and also includes the ability to produce time-series plots of past values of these signals.
- The GUI allows the user to specify input values (control signals), inject faults and modify other simulation parameters, both before a simulation run and during its execution.
- The GUI has two operational modes: an interactive mode which runs a simulation and generates new output data, and a playback mode which replays data recorded from a prior simulation run, and thus does not require the use of the MATLAB/Simulink software.

4.2.1. Java-MATLAB Interface

As shown in Figure 1, the interface between CryoSim and the GUI includes a change from the MATLAB environment used in CryoSim to a Java-based GUI. To enable full interactivity, user inputs to the GUI must be passed to the CryoSim initialization and control modules, and model outputs must be sent from the control module to the GUI, all without stalling or otherwise interrupting the simulation. As noted above, the use of Java for the GUI's internal computations provides significantly more control over the threading and scheduling of these computations, compared to the single-threaded Simulink environment of the simulation. This is important because the CryoSim control module is part of the Simulink model, and any blocking or delay due to interaction with the GUI has the potential to significantly reduce simulation speed.

Communication between MATLAB and Java can be implemented in MATLAB via the built-in javaMethod() functionality, and in Java using the third-party matlabcontrol API (http://code.google.com/p/matlabcontrol). MATLAB access to Java objects and methods is well-documented and supported by Mathworks, Inc., and serves as the basis of most of our interface. On the other hand, Java access to the MATLAB environment via matlabcontrol makes use of an undocumented interface, although some information is available through third parties (Altman, 2013 and Altman, 2011). Additionally, the matlabcontrol API provides only a limited feature set compared to the use of javaMethod(), providing further weight to our decision to use javaMethod() calls whenever possible.

As shown in Figure 5, the data flow between the GUI and CryoSim can be classified into initialization procedures, which take place before a simulation run, and interactive control during the simulation run. The data exchanged between the GUI and CryoSim during initialization includes model parameters and input signals as well as a table containing faults to be injected during the simulation. During the initialization stage, CryoSim loads a user-specified input file which provides the parameters and signals used for a simulation run. CryoSim then pushes this data to the GUI, where it is used to initialize the information presented to the user. The user can then modify signals, parameter values, and the fault injection table. When the user has created the

desired simulation scenario, the simulation can be started via the simulation controls available in the GUI.

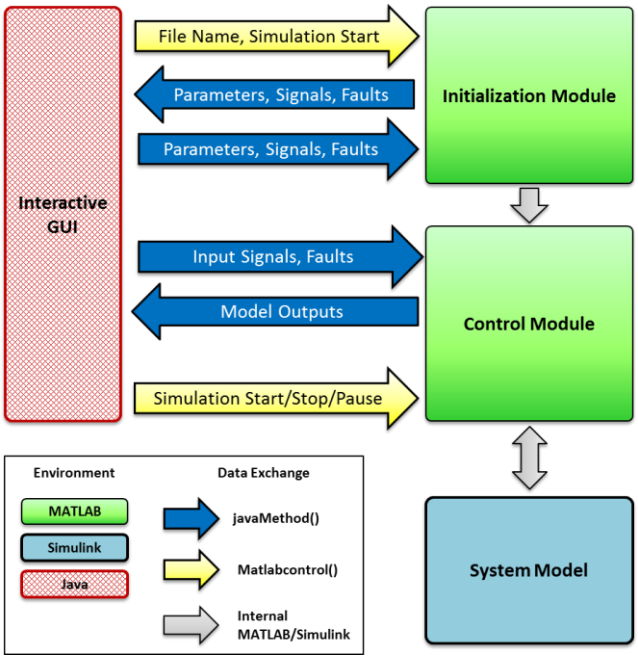


Figure 5. CryoSim-GUI interface

During a simulation, the interaction between CryoSim and the GUI is handled through the use of javaMethod() calls in CryoSim's control module. The control module pushes a vector containing the current output signal values to the GUI, where they are used to update the GUI's display of the model's state. The control module then queries the GUI for any updated input signals or faults that the user might have supplied, and provides them to the model. In this implementation, all user inputs to the GUI are applied to the model at the next time step after the GUI has made them available. This mode of interaction, when coupled with the flexible threading available in Java, allows the simulation to run independently and without risk of blocking from the GUI. The only direct control the GUI has over CryoSim is the set of simulation commands (start, stop and close simulation), which use the matlabcontrol API and can thus execute as soon as activated by the user.

4.2.2. CryoSim-GUI Interaction

The GUI provides full interactive control of CryoSim using the controls shown in Figure 6. This diagram shows the main GUI window, which consists of a number of panels and controls, numbered here for reference in the text. The system shown has been simplified from the full CryoSim model in order to reduce its visual complexity for illustration purposes.

As discussed earlier, CryoSim makes use of an input file to provide parameters, input signals and faults for a particular simulation run. The GUI menu bar (#1) allows the user to select an input file, which is shown in the configuration panel (#2). This panel also allows the user to specify the simulation length (#3), inject faults (#4) and modify input signals (#5). The Simulation Status panel (#6) displays status messages, simulation progress and the simulation clock. The Simulation Controls panel allows the user to start, stop or pause the simulation (#7) and control the simulation speed (#8).

The System Schematic Panel (#9) contains a graphical representation of the system model's components and topology. Sensor components are included in the model in locations corresponding to CTB sensor locations. Their outputs are shown using text boxes in the GUI (#10) and are updated during each iteration of the control module. Additional 'virtual sensor' outputs in the model provide simulation data for locations in which there is no corresponding sensor in the actual system. The GUI makes use of these additional signals to provide more data to the user for exploratory data analysis, allowing for more detailed understanding of the system's behavior.

As mentioned earlier, the GUI graphics were developed using the GLG Toolkit, which provides an interface whereby the visual appearance of an element can dynamically change based on a state variable's value. We use this functionality to display valve positions (indicated by the color of the valve's body, #11), pipe flow rates and system temperatures.

In addition to serving as a visual representation of the model's state, the GUI allows interactive control of the model's inputs and fault injection status. The user can click on the text label for any component and bring up a component detail window. The contents of this window vary depending on the nature of the component, but can include a time-series plot of the component's input and output signals, controls to allow the user to override the default input signals with a user-determined value, and the ability to inject a fault into the component, either immediately or at some future time during the simulation. Additionally, clicking on a valve body in the main GUI window will toggle the valve's position between fully open and fully closed. All user-supplied input signals are implemented as overrides to the signal values contained in the input file. Thus, if a user sets a valve position using either the toggle functionality or the component detail window, the valve will remain under user control for the remainder of the simulation run, rather than respond to any pre-scheduled changes in the input file.

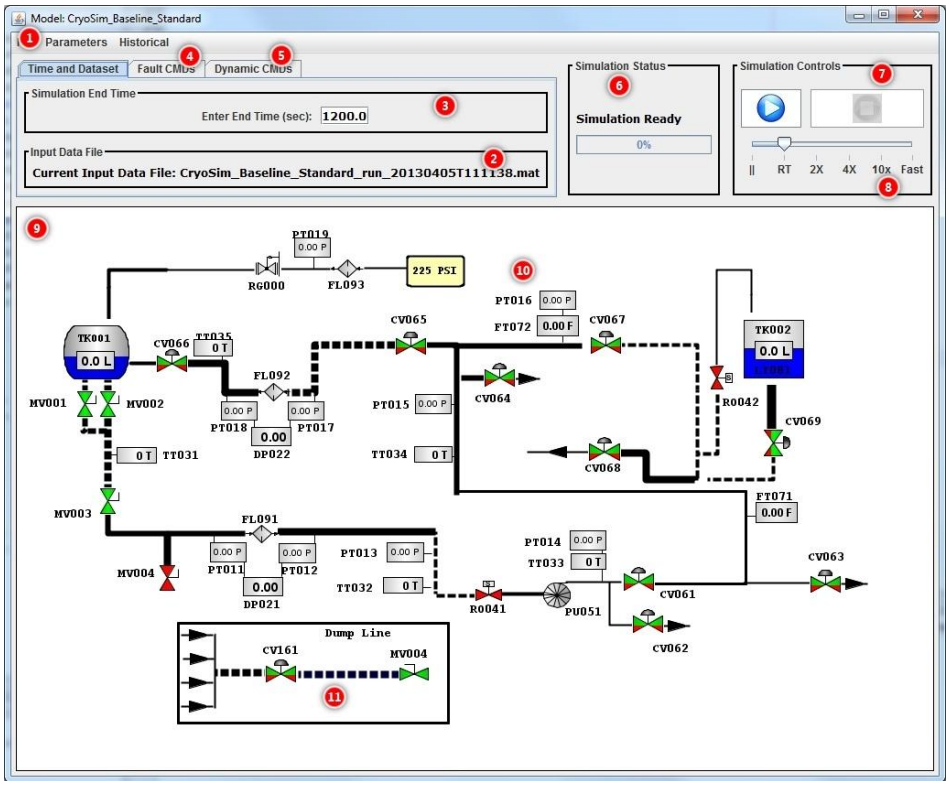


Figure 6. CryoSim GUI

4.3. Message Bus Interface

The primary objective of the CryoSim package is to act as a data source substitute for the CTB system. This architecture makes use of a publish/subscribe message bus based on the Internet Communications Engine (Ice), a suite of communications middleware developed by ZeroC (www.zeroc.com). Using Ice, a message bus based on the publish/subscribe paradigm was developed as the core of the integrated health system. In this configuration, a system being monitored will publish sensor readings indicating its current state to the message bus. Health management modules then subscribe to this published data, perform analysis, and publish messages based on this analysis. The same framework was used in (Poll et al., 2007) and (Balaban et al., 2013).

During a simulation run, CryoSim publishes the values of the model's output signals, which represent all of the available sensors and test points of the CTB system. By controlling the simulation's execution speed to match real-world time, the data published by CryoSim matches the real-world CTB data in timing and format. This enables developers of health management modules to use CryoSim as a virtual testbed without modification. Additionally, the publish/subscribe message bus architecture allows remote control over the CryoSim system through the same interface.

Within CryoSim, the message bus interface is implemented as a hybrid MATLAB/Java construct. The Ice software generates a Java class interface containing the user-defined message formats. The message classes are then combined with Java code that implements the necessary publish and subscribe functionality. The CryoSim initialization module then instantiates the Java classes, establishing communications with the message bus via calls to their methods.

5. RESULTS

In this section we present the results of several simulation runs to demonstrate key aspects of CryoSim. We first provide some information on simulation accuracy and speed, followed by an example of component-level fault injection, followed by demonstration of the use of variant subsystems to trade simulation fidelity for speed. Finally, we show an example of a prognostic health management algorithm interacting with the simulation to produce an analysis.

5.1. Simulation Accuracy and Speed

CryoSim uses the Simulink environment for determining the time-varying solution to the set of ordinary differential equations (ODE) that constitute the system model. The choice of solver algorithm and error tolerances affects both simulation speed and accuracy. For our model, the variable step size ode45 solver works well with the "Standard" fluid

variants, while the increased stiffness of the two-phase model requires the use of the ode23s solver, also a variable-step algorithm. For the standard model, a typical simulation runs at approximately 3x speed on a 3.3 GHz workstation, although the time required for a given simulation can vary significantly depending on the dynamics of the model inputs. We have obtained acceptable results using the default relative error tolerance of 1e-3, although some integration noise can be seen in the fluid flow rates. This noise can be reduced by lowering the error tolerance at the cost of decreased simulation speed. The two-phase model is still under development, but is expected to run more slowly than the standard model due to the increased complexity of the underlying physics.

We have validated the model against data recorded from the CTB in the post-chilldown state. In this scenario, the simulated pressures matched the measured data with a maximum error of 0.23%, while the fluid flow rate matched the measured rate to better than 0.1%, considerably less than the measurement noise. Actuation times of the pneumatically-controlled valves had lower fidelity, with a mean 10-90% rise-time error of 0.58 seconds in absolute terms, and 17.7% error relative to the observed actuation times. However, the accuracy of the pneumatics domain components is adequate for most of the intended uses of CryoSim, which focus on system behavior in the cryogenic fluid domain.

5.2. Fault Injection

To demonstrate the fault injection capability of CryoSim, we ran a basic simulation with a single fault injected during the run. This example used the standard flow variant rather than the two-phase flow. The model was initialized in a post-chilldown state, where virtually all of the cryogenic fluid is in the liquid phase. Figure 7(a) shows a schematic representation of a part of the system, including the control valve CV201. This valve starts in the fully open position, and at t=30 seconds, a "Stuck" fault is injected with a magnitude of 50%.

Figure 7(b) shows the outputs of four pressure sensors in the model. In the schematic PT134 is shown immediately upstream of the faulty valve, and PT147 is located at the end of the section shown in the schematic. PT112 is not shown, but is located further upstream from the valve, while PT193 is further downstream. When the fault is injected at t=30, the upstream pressures increase slightly, while the downstream pressures experience a more significant decrease due to the increased drop at the valve.

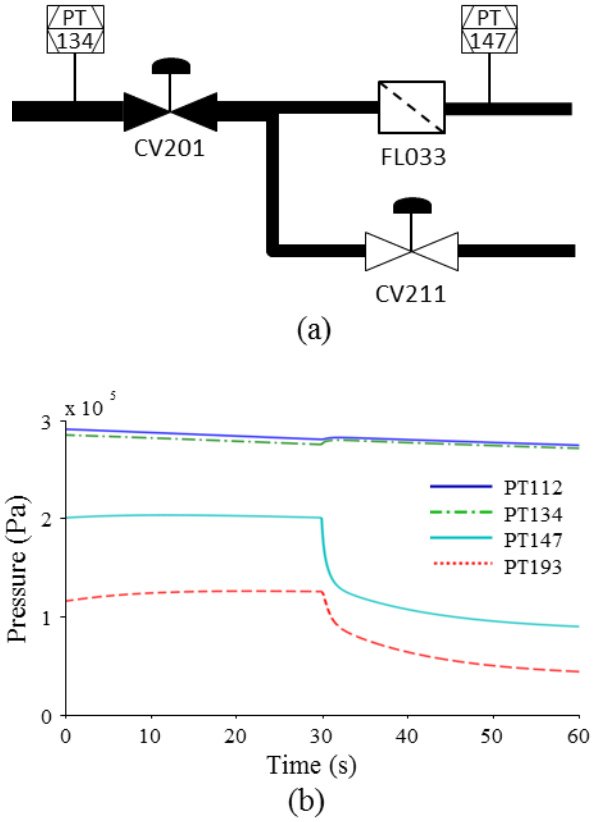


Figure 7. Fault injection example

5.3. Variable Simulation Fidelity

To show the use of variant subsystems to selectively control model fidelity in a specified domain, we simulated the actuation speed of a pneumatically-actuated control valve component, using the "Standard" pneumatics variant which computes gas pressures and flows and uses these values to determine the actuator's position, and compared this run with the "Basic" pneumatics variant, which replaces the pneumatics-domain components with simple behavioral models. For the pneumatically-actuated control valve, the "Basic" variant replaces two nonlinear pressure-computing elements with simple first-order lowpass filters. The behavior of these two variants compared to experimental data is shown in Figure 8. Note that the "Standard" variant more accurately represents the observed data than the "Basic" variant.

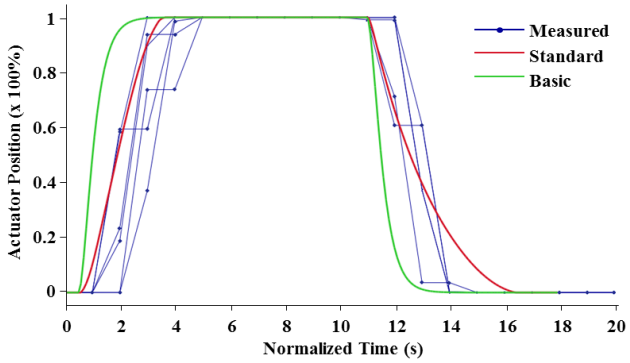


Figure 8. Behavior of pneumatic valve actuator models

The use of domain-specific variants to select appropriate levels of fidelity can significantly reduce simulation time. In this example, the use of the reduced-fidelity "Basic" variant for the pneumatics domain resulted in small differences in valve actuation speed that did not significantly affect the signals of interest in the cryogenic fluid domain. The difference in simulation speed for a run with 1000 seconds of simulation time was dramatic, with the "Standard" variant requiring 676 seconds (1.48x) and the "Basic" variant requiring 354 seconds (2.82x).

5.4. Prognosis Example

To demonstrate how the simulation interacts with a health management algorithm over the message bus, we select a prognosis example demonstrating prognostics of a cryogenic filter. Filters are often periodically replaced on a time-based maintenance schedule. Moving to a condition-based maintenance paradigm can prevent a healthy filter from being replaced and a damaged filter from being used.

The purpose of the filter is to prevent particles contaminating the fluid from moving through to other parts of the system. As fluid passes through a filter, particle matter will collect at the filter and decrease its effective area, thus increasing the pressure drop across the filter for the same flow rate. This behavior is captured in the following equations.

$$Q(t) = \rho\, c\, A(t) \sqrt{\frac{2|\Delta p(t)|}{\rho}} sign\big(\Delta p(t)\big) \qquad (1)$$

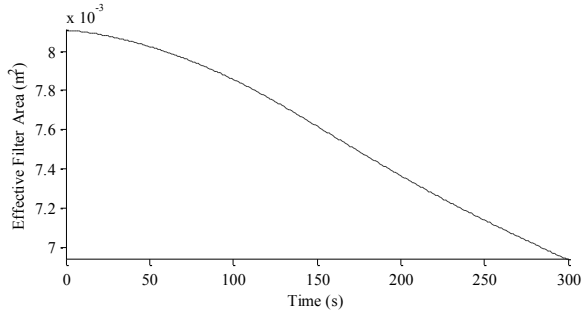
$$\dot{A}(t) = -wQ \qquad (2)$$

Here, Q is the volumetric flow, ρ is the fluid density, c is the flow coefficient, A is the effective filter area, Δp is the pressure drop across the filter, and w is a wear parameter representing the percentage contamination per unit length of fluid (which is, in general, stochastic). The effective filter area decreases as a function of the contamination and the flow rate through the filter.

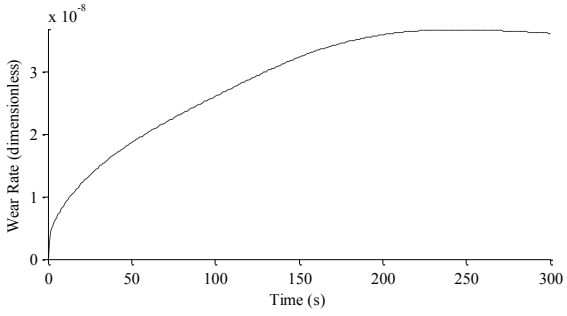
In this model, the pressure difference is an input and the flow is an output. A model-based prognosis algorithm is used in which the health state of the filter (A and w) is estimated, and this estimate is then used as the initial state in predicting end of life (EOL) and remaining useful life (RUL) of the component (Daigle & Goebel, 2013). For the filter, EOL is defined as the time point at which the effective filter area drops below some specified limit, in this case, 50% of its nominal area.

The prognostics module receives over the message bus the measured values of the differential pressure and the flow, and these serve as inputs to the estimation algorithm (an unscented Kalman filter, see (Julier & Uhlmann, 2004) and (Daigle et al., 2012) for details). The module makes periodic predictions for filter EOL and RUL, and publishes these back to the message bus.

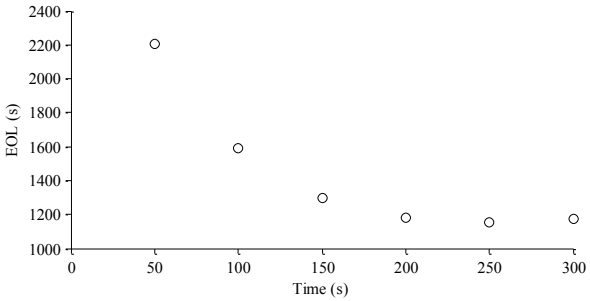
As a demonstration we consider a wear parameter value of 3×10^{-8}. Figure 9 shows the estimated filter area, wear parameter, predicted EOL, and predicted RUL. By 200 s the estimates and predictions begin to converge.



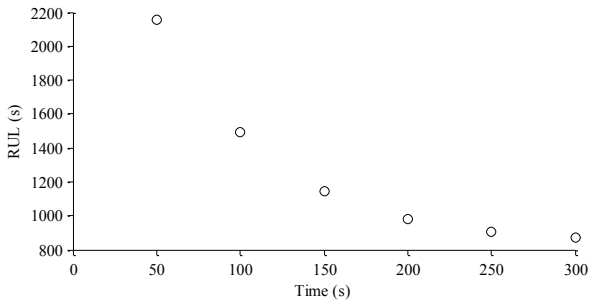
(a)



(b)



(c)



(d)

Figure 9. Filter prognostics results.

6. CONCLUSION

In this paper, we have discussed the design and application of CryoSim, a simulation-based virtual testbed for SHM applications. The core of CryoSim is a component-based multi-domain system model created in Simulink. Our design includes component-level fault modeling and per-simulation adjustment of model fidelity through the use of variant subsystems. To facilitate the use of CryoSim, we created a Java-based GUI which allows full interactive control of the simulation. Additionally, we have integrated an interface to an external publish/subscribe message bus, enabling CryoSim to function as a drop-in replacement for the CTB system.

We are currently in the process of converting the Simulink model to a standalone version written in C/C++, making use of the Simulink Coder™ software package. This will enable users of CryoSim to run simulations without the need for a license for MATLAB and Simulink. The two-phase cryogenic fluid flow model is under development. When complete, it will be incorporated into CryoSim, taking advantage of the capability to update library components and domain-specific variant subcomponents without impact to previously-available functionality. Because of the model-agnostic design of the CryoSim framework, we anticipate that this architecture will be used for future SHM applications with other multi-domain system models.

ACKNOWLEDGEMENT

This work was funded in part by the NASA Automated Cryogenic Loading Operations (ACLO) project under the Office of the Chief Technologist (OCT), the Advanced Ground Systems Maintenance (AGSM) Project under the Ground Systems Development and Operations program and the Integrated Ground Operations Demonstration Unit (IGODU) Project under the Advanced Exploration Systems (AES) program.

REFERENCES

Agusmian P.O., Sas, P. and Van Brussel, H. (2013). Modeling and simulation of the engagement dynamics of a wet friction clutch system subjected to degradation: An application to condition monitoring and prognostics. *Mechatronics*, vol. 23, no. 6, pp. 700-712.

Altman, Y. (2013). *Undocumented MATLAB*, http://undocumentedmatlab.com/

Altman, Y., (2011). Undocumented Secrets of MATLAB-Java Programming, Chapman and Hall/CRC

Balaban, E., Narasimhan, S., Daigle, M., Roychoudhury, I., Sweet, A., Bond, C., & Gorospe, G. (2013, May). Development of a Mobile Robot Test Platform and Methods for Validation of Prognostics-Enabled Decision Making Algorithms, *International Journal of Prognostics and Health Management*, *4*(1).

Biswas, G., Mahadevan, S. (2007, March) A Hierarchical Model-based approach to Systems Health Management. *Proceedings of the 2007 IEEE Aerospace Conference*.

Daigle, M., & Goebel, K. Model-based Prognostics with Concurrent Damage Progression Processes. (2013, May). *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, *43*(4), 535-546.

Daigle, M., Saha, B., & Goebel, K. (2012, March). A Comparison of Filter-based Approaches for Model-based Prognostics. In *Proc. of the 2012 IEEE Aerospace Conference*.

Goodrich, C., Narasimhan, S., Daigle, M., Hatfield, W., Johnson, R., & Brown, B. (2009, June). Applying Model-based Diagnosis to a Rapid Propellant Loading System. In *Proc. of the 20th International Workshop on Principles of Diagnosis*, 147-154.

Granger, R. A. (1995). Fluid Mechanics. New York, NY: Dover.

Julier, S. J., & Uhlmann, J. K. (2004, March). Unscented filtering and nonlinear estimation. In *Proc. of the IEEE*, *92*(3), 401–422.

Poll, S., Patterson-Hine, A., Camisa, J., Garcia, D., Hall, D., Lee, C., Mengshoel, O., Neukom, C., Nishikawa, D., Ossenfort, J., Sweet, A., Yentus, S., Roychoudhury, I., Daigle, M., Biswas, G., & Koutsoukos, X. (2007, May). Advanced Diagnostics and Prognostics Testbed. In *Proc. of the 18th International Workshop on Principles of Diagnosis*, 178-185.

BIOGRAPHIES

John P. Barber received the B.S. and Ph.D. degrees in Electrical Engineering in 2001 and 2006 from Brigham Young University, Provo, UT, USA. His research and professional activities have focused on process development and control, as well as modeling and simulation in a number of domains. He is currently an employee of Stinger Ghaffarian Technologies, supporting NASA via the ESC contract at Kennedy Space Center. He is a member of the IEEE.

Kyle B. Johnston received the B.S. degrees in Physics and Astrophysics in 2004 and the M.S. degree in Space Science in 2006 from the Florida Institute of Technology. He has been a primary and co-author on a number of astrophysics journals while working in the academic space. Since 2007 he has worked in the private sector, specializing in the development of scientific simulations, analytical analysis of large datasets, and intelligent passive sensor development. He is currently an employee of ENSCO, Inc. Aerospace Engineering division. Professional society memberships include the American Statistical Association and the American Astronomical Society.

Matthew Daigle received the B.S. degree in Computer Science and Computer and Systems Engineering from Rensselaer Polytechnic Institute, Troy, NY, in 2004, and the M.S. and Ph.D. degrees in Computer Science from Vanderbilt University, Nashville, TN, in 2006 and 2008, respectively. From September 2004 to May 2008, he was a Graduate Research Assistant with the Institute for Software Integrated Systems and Department of Electrical Engineering and Computer Science, Vanderbilt University, Nashville, TN. From June 2008 to December 2011, he was an Associate Scientist with the University of California, Santa Cruz, at NASA Ames Research Center. Since January 2012, he has been with NASA Ames Research Center as a Research Computer Scientist. His current research interests include physics-based modeling, model-based diagnosis and prognosis, simulation, and hybrid systems. He is a member of the IEEE.