

Integration of MATLAB Simulink® Models with the Vertical Motion Simulator

Emily K. Lewis¹

*Science Applications International Corporation
NASA Ames Research Center, Moffett Field, CA, 94035*

and

Nghia D. Vuong²

NASA Ames Research Center, Moffett Field, CA, 94035

This paper describes the integration of MATLAB Simulink® models into the Vertical Motion Simulator (VMS) at NASA Ames Research Center. The VMS is a high-fidelity, large motion flight simulator that is capable of simulating a variety of aerospace vehicles. Integrating MATLAB Simulink models into the VMS needed to retain the development flexibility of the MATLAB environment and allow rapid deployment of model changes. The process developed at the VMS was used successfully in a number of recent simulation experiments. This accomplishment demonstrated that the model integrity was preserved, while working within the hard real-time run environment of the VMS architecture, and maintaining the unique flexibility of the VMS to meet diverse research requirements.

Nomenclature

<i>CAMAC</i>	=	Computer Automated Measurement and Control
<i>DAVE-ML</i>	=	Dynamic Aerospace Vehicle Exchange Markup Language
<i>IG</i>	=	Image Generator
<i>I/O</i>	=	Input/Output
<i>OTW</i>	=	Out-The-Window
<i>PEP</i>	=	Project Engineer Panel
<i>SCRAMNet</i>	=	Shared Common Random Access Memory Network
<i>SEP</i>	=	Simulation Engineer Panel
<i>VMS</i>	=	Vertical Motion Simulator

I. Introduction

THE Vertical Motion Simulator (VMS) at the NASA Ames Research Center, shown in Fig. 1, is a six-degree-of-freedom flight simulator designed to provide realistic motion cues for high fidelity piloted simulation. Research on pilot motion cueing has shown that lateral and vertical motion cueing has a significant effect on pilot-vehicle performance and control activity.¹ Furthermore, realistic motion cueing allows simulations to better predict pilot workload and in-flight performance.² The high-fidelity, large motion capability at the VMS enables it to provide a realistic cueing environment where the resultant pilot cueing modalities and control techniques do not differ significantly from flight.³ This level of realism enables the VMS to deliver high quality research data that translates to flight.

The VMS facility includes an adaptable simulation environment that can support a wide range of aeronautical research. A large number of actual and conceptual aircraft have been studied at the VMS over its 32-year history. These include various helicopters, Vertical/Short Take-Off and Landing and Conventional aircraft, tilt-rotors, airships, spacecraft, and the Space Shuttle. The VMS has played an important role in the development of several major programs which are currently flying, such as the V-22 tilt-rotor and Joint Strike Fighter, by providing critical

¹ Senior Simulation Engineer, NASA ARC SimLabs, Mail Stop 243-6, and AIAA Member.

² Simulation System Engineer, NASA ARC SimLabs, Mail Stop 243-5, and AIAA Member.

design data on aircraft handling qualities and flight control development. It was also used to train Space Shuttle Orbiter pilots on the approach, landing, and rollout phase of flight.

The VMS host environment provides a robust and flexible set of development, debugging and execution tools which enables researchers and engineers to conduct research quickly and effectively. The adaptable simulation environment has been critical to the facility's ability to support a variety of aerospace vehicles (as mentioned previously) and research topics, including handling qualities, guidance and displays, flight control design and evaluation, concept demonstration and feasibility studies, accident investigations, human pilot cueing research and simulation fidelity requirements.

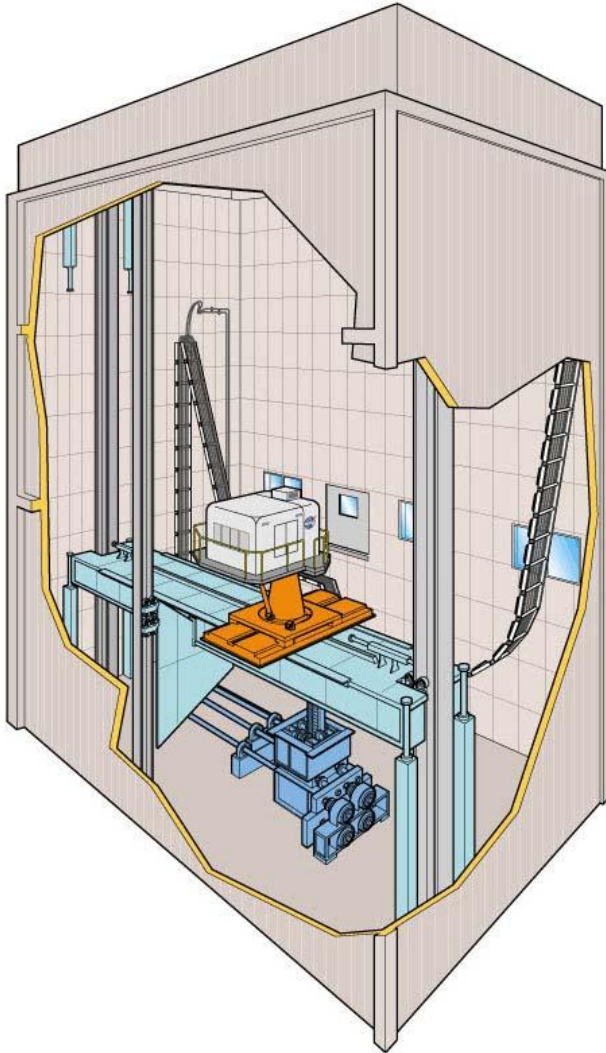


Figure 1. Cut Away View of Vertical Motion Simulator

execution control was transferred to the MATLAB workspace. The NASA Langley Research Center simulation engineers took a different approach for the SAREC-ASV (Simulink-Based Simulation Architecture for Evaluating Controls for Aerospace Vehicles) and the B-737 Linear Autoland Model efforts.^{5,6} Their method was to program and run the entire simulation in MATLAB in order to provide a desktop simulation capability of sufficient fidelity to effectively derive and evaluate aircraft control laws and control system components. While valuable for development work, this scheme is not appropriate for a piloted simulation. Several real-time and non-real-time applications in academia and industry use Simulink's target integration mechanism for off-the-shelf simulation platforms.⁷⁻⁹ None of these approaches, however, were suitable for the VMS due to its stringent or "hard real-time" requirements and the necessity to simulate diverse vehicles. "Hard real-time" means that the execution of all software computations must be completed within a specific frame time. A more suitable approach for the VMS is the MOSAIC (Model-Oriented Software Automated Interface Converter) developed at The Netherlands' National

Over the operational history of the VMS, simulation vehicle modeling standards have evolved towards a graphical development environment where complex systems can be created and tested before actual hardware implementation. Many in the aerospace simulation industry have adopted MathWorks products such as MATLAB® and Simulink® to develop models.

Traditionally, aircraft math models simulated at the VMS were provided by the researcher in the form of block diagrams, algorithm descriptions, or sets of equations that would be programmed and tested by VMS engineers. With the advances in computing, researchers now have access to improved desktop vehicle model development and simulation tools. Routinely, they are now able to provide a mature model, often in MathWorks' Simulink, which can be implemented on the VMS. In these cases integrating the Simulink models directly into the VMS environment, instead of reprogramming the delivered math models, can eliminate programming errors and reduce the simulation implementation and validation time. The challenge faced by VMS engineers was to find a means of doing this efficiently and accurately.

As Simulink became more popular for model development, the demand for Simulink models to be integrated with simulation facilities with an existing and mature architecture increased. For example, at the Naval Air Systems Command at Patuxent River, the legacy hi-fidelity aircraft simulation environment, CASTLE (Controls Analysis and Simulation Test Loop Environment), was integrated with Simulink in order to perform flight control system development work.⁴ In that case the real-time aircraft simulation set-up and

Aerospace Laboratory (NLR), which automates the conversion from MATLAB's Real-Time Workshop® (RTW) code to a predefined Application Programming Interface.¹⁰ MOSAIC was used successfully in several aerospace projects in Europe such as the European Space Agency's Automated Transfer Vehicle project.¹¹ Though this approach was suitable for the VMS, it would have required substantial modification to work within the VMS's real-time operating environment. The VMS system has been refined over decades of operations. It includes sophisticated debugging and development tools, and other specialized functions, which give it the flexibility needed to meet current and future research requirements.

A method to integrate Simulink into the VMS was developed and was successfully implemented for a number of recent experiments. This paper will describe this method and its application. First, the paper will provide an overview of the VMS, including the hardware, software, and communication systems. Subsequent sections will describe different projects that have utilized Simulink and will provide a detailed discussion of the process to integrate Simulink models as well as challenges overcome and lessons learned.

II. Description of the Vertical Motion Simulator

The VMS is a one-of-a-kind research simulation system, comprised of many interchangeable components that can be tailored to simulate any aerospace vehicle and meet the specific needs of most NASA programs. Housed in a ten-story tower, with a 60-foot vertical displacement and 40-foot lateral displacement, the VMS, shown in Fig. 1, offers the largest motion range^{1,3} of any simulator in the world. In addition, a set of five interchangeable and reconfigurable cabs, a variety of out-the-window (OTW) visual scenes, an accurate control force-feel system, and flexible flight instrument and engineering displays allows almost any aerospace vehicle to be simulated on the VMS.

A. Motion System

The most striking feature of the VMS, and the characteristic for which it was named, is its large vertical motion capability, which enables the VMS to offer high fidelity motion.¹² The cockpit sits on a 70 foot long platform which spans the VMS tower and travels vertically ± 30 feet. This platform is supported by two columns that extend into 75-foot deep shafts. The columns are pressurized with nitrogen to neutralize the weight to provide smooth vertical motion. On top of the platform is a moving carriage which translates along the platform and provides a horizontal displacement of ± 20 feet. A sled sits on the carriage and provides ± 4 feet of the third translational degree of freedom.

The short translational and three rotational degrees-of-freedom are provided by a high performance four-axis motion generator with four large telescoping actuators. Fig. 2 illustrates the six-degrees-of-freedom mechanisms. Table 1 describes some of the motion system performance limits. These large limits allow the cab to sustain movements for a relatively long period of time, a characteristic that enables the VMS to reproduce realistic cues during critical phases of flight.

B. Pilot Cockpit Hardware

The flexible and dynamic research laboratory is achieved by way of a wide range of adaptable hardware components which can be put together in various ways to accurately represent most any vehicle, real or notional. Five different interchangeable cabs are available which are used to represent the cockpit of the simulated vehicle.

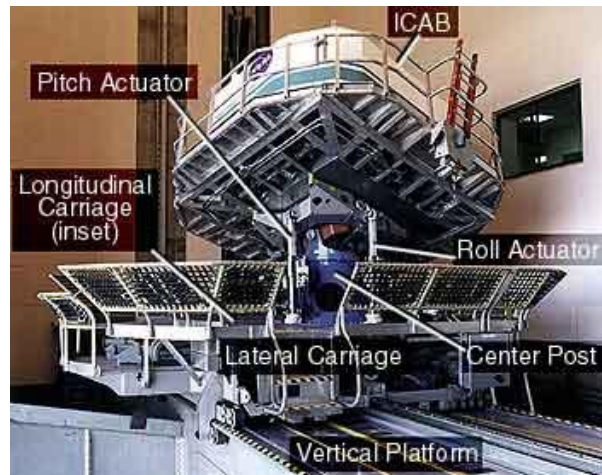


Figure 2. Vertical Motion Simulator Cab in the Tower

Table 1. Vertical Motion Simulator System Performance Limits

VMS NOMINAL OPERATIONAL MOTION LIMITS			
Axis	Displacement (ft)	Velocity (ft/sec)	Acceleration (ft/sec ²)
Vertical	± 30	16	24
Lateral	± 20	8	16
Longitudinal	± 4	4	10
	(deg)	(deg/sec)	(deg/sec ²)
Roll	± 18	40	115
Pitch	± 18	40	115
Yaw	± 24	46	115

The cabs are outfitted with appropriate pilot control inceptors, out-the-window (OTW) visual scene displays, and flight instruments. Fig. 3 shows an example cab that was used for a tilt-rotor experiment.

A large collection of control inceptors (e.g. center stick, wheel and column, pedals) are available. The McFadden Force Loader system is used to deliver versatile, accurate, and realistic control cues to the pilot. With this system, control force feel characteristics - such as position trim, breakout, friction and non-linear gradient - can be easily tailored to meet the needs of a variety of research programs. In addition, a variety of passive controls are available for the cab, such as a throttle quadrant, trim hat, flap and gear handles, thumb wheels, buttons and lights.



Figure 3. Example Two-Seat Cab with Tilt-Rotor Controls and Displays

C. Computer Hardware

The VMS computer system architecture is comprised of a host computer, on which the aircraft math model executes, along with a number of other separate processors. The host computer, a 1GHz Hewlett Packard Alpha running Open Virtual Memory System (OpenVMS), provides the executive and real-time Input/Output (I/O) control functions, the interface for executing and debugging models, and control of the other processors (visual, motion, etc). In the near future, the host and operating system will be upgraded to a Linux based system. Other processors include computers that generate images to emulate the outside world, graphics workstations that create the cockpit flight instruments and lab engineering displays, a sound generator, and other processors or hardware that may be included in the real-time simulation as needed for a simulation study.

Three Rockwell Collins multi-channel Image Generators (IG) are available to produce the cab OTW view. Each scene uses a database that contains the relevant characteristics of the geographical location and a high-resolution inset of the specific area of interest. In addition, three-dimensional moving and/or stationary objects can be included to enhance visual cuing. Moreover, most cabs incorporate a collimated visual scene such that the view is focused at infinity which greatly enhances the sense of depth and realism. The IGs can also be used to drive a third person perspective, complete with moveable control surfaces, for research observation.

Dual core processor Hewlett Packard workstation graphic engines are used to drive the cab flight instruments and engineering displays. These can include Head-Up Displays and Head-Down Displays (HDDs) such as Primary Flight Displays (PFDs), Navigation, and Engine displays. Standard aircraft cockpit instruments are available and can be used as-is or modified for a particular experiment. Lab engineering displays include real-time variable monitoring, electronic strip charts, and control activity or pilot performance displays. These are often created specifically for a particular study.

An Advanced Simulation Technology Inc (ASTi) sound generation system is used to create the aircraft flight noise. Sound is heard through speakers and headsets in the cab and lab. A typical sound model consists of air noise, engine whine, rotor, and gear sounds. It can also comprise call outs, warning tones, or special aural cuing sounds.

Each of these components are shown in the system diagram in Fig. 5, along with the communication connections between the components, which are discussed in the next subsection. In this figure and for the purpose of this paper, a “driver” is an algorithm which is programmed to control elements of a device or graphics executable.

D. Communication

1. Communication Devices

Communication between the host and attached computational processors is handled over a real-time network using Ethernet protocols.¹⁴ An in-house “raw Ethernet” protocol is used for cab flight instruments, lab engineering displays, sound, and auxiliary processing.

A Computer Automated Measurement and Control (CAMAC) real-time data acquisition system is used to provide an interface between the cockpit passive controls and the host computer. The CAMAC is an industry standard high-performance data acquisition interface by Kinetic Systems. It supports the various input/output requirements of a simulation.

Examples of CAMAC driven I/O include the throttle quadrant, trim hat, thumb wheel, flap and gear handles, buttons, and indicator lights in the cab; and the strip charts, Simulation Engineer Panel (SEP), and Project Engineer Panel (PEP) buttons in the lab. The SEP and PEP are two panels of push-button switches and indicator lights that can be customized to set-up, interact with, and monitor the simulations.

A Shared Common Random Access Memory Network (SCRAMNet) is used for communication between the host computer and the Motion Control Unit, which provides motion system commands and accelerometer sensor feedback; the pilot control loaders; and other functions as needed. The SCRAMNet is a reflective memory ring, a network of computers (the host plus other devices), that share a common memory. These computers are connected by fiber optic

cables and use special hardware, developed by Systran, to map this shared memory into the user memory space of each computer system. All computers on the SCRAMNet share the same memory.

Several devices are available for data collection. A high capacity removable disk is used to record time history data. A list is created with all variables to be recorded. Values for these variables can be recorded every frame to the data disk. This time history data can be converted to a variety of formats or put onto an external drive immediately

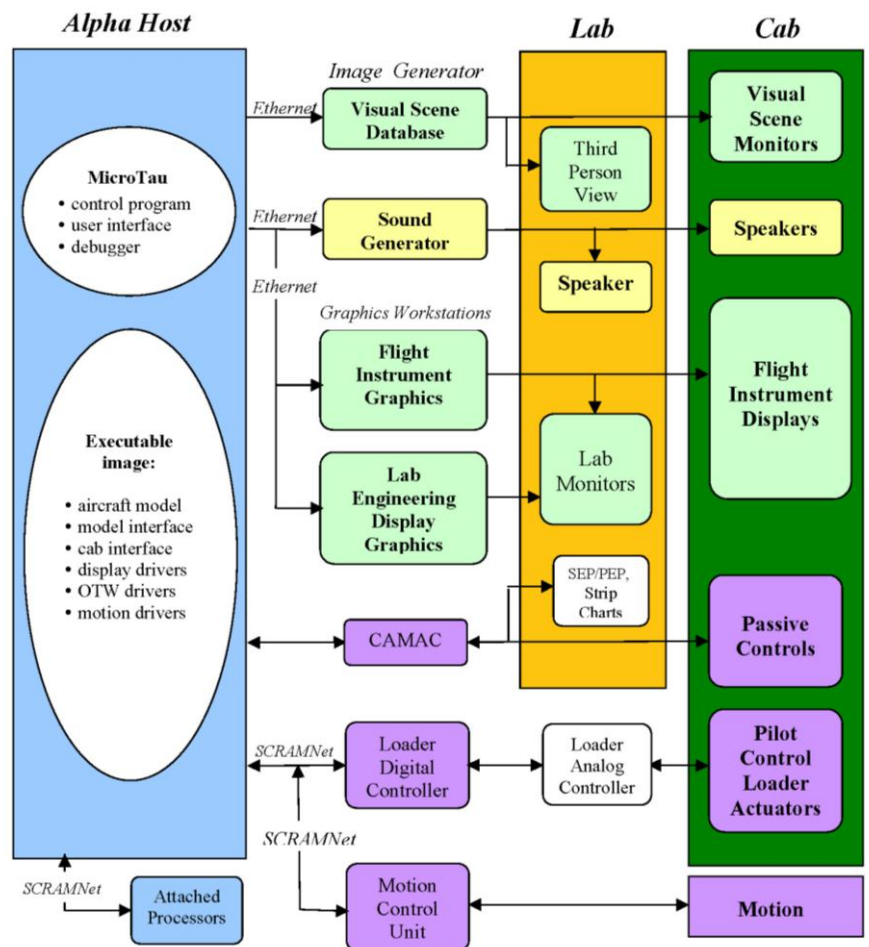


Figure 5. Vertical Motion Simulator System Diagram

after an evaluation run. Other data collection options include two DVD recorders which can be used to make audio and visual recordings of any display, and a high-resolution digital video recorder can be used to make quality digital recordings.

2. List Driven Input and Output

The VMS communication system uses special runtime configurable I/O lists to send and receive data between the host and all the devices mentioned above. This arrangement enables fast and easy I/O changes to any simulation component while running by eliminating the need to suspend, re-compile, or relink any code. This flexibility is made possible because of the symbol table, a list of global variables, which associates a variable (symbol) with its address, names, and value. All necessary variables are static and their addresses are known all the time. Thus any variable in the symbol table can be added to an I/O list at any time during a simulation experiment in a matter of moments. The current VMS operating system is unable to handle C code structure members or variables, so any symbol on an I/O list must be a FORTRAN variable. Moreover, any number of variables may be defined in a script file and brought into an executing simulation at one time. This offers the engineers the ability to make rapid configuration changes and accelerates development work. The list driven I/O concept is depicted in Fig. 6. Each arrow shown in this diagram represents a set of I/O lists which arrange the transfer of data between the host and each simulation component.

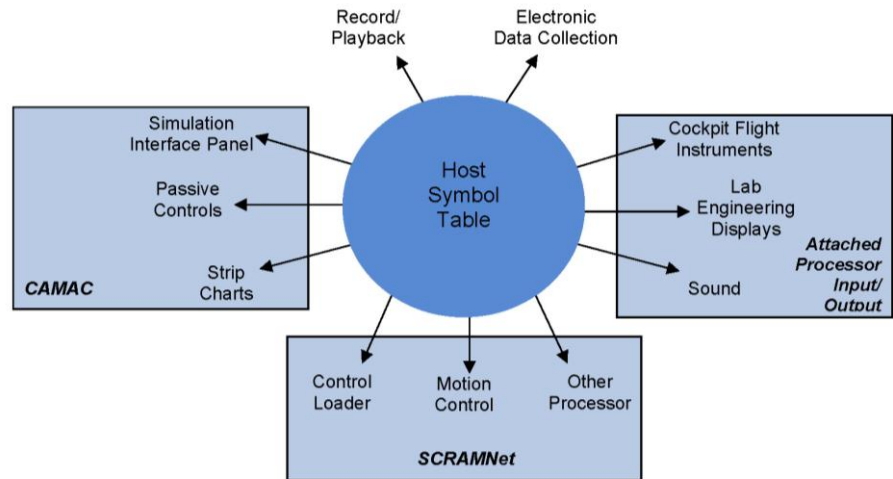


Figure 6. Symbol Table Interface Diagram

E. Software

Like the hardware, the VMS software has been designed to support a flexible and rapid development capability while maintaining an accurate and high-fidelity environment for simulation experiments. This is achieved by way of a set of software components, the backbone of which is the in-house developed MicroTau real-time environment. MicroTau provides effective tools for developing and testing aircraft simulation models and executing these models in a hard real-time piloted experiment. It controls the other simulation processors, performs the aircraft model calculations and supplies the user interface to the real-time job.¹³ Available interactions include an on-line real-time debugger, an operational window in which engineers can monitor variables and simulation execution, and the facility to deposit values into any number of variables either interactively or by loading script files. This system makes changing configurations fast, easy, and repeatable during experiment operations.

Another important component is a software library which contains various modules that perform model-independent functions such as calculating the aircraft equations of motion, driving the cockpit visual scene, controlling the motion system and recording electronic data. In addition, this library also includes routines, such as a gear model, aircraft trimmer, dynamic check generator and a variety of filters, which are available to use and modify as needed for any particular experiment. Typically, a simulation will comprise the model-independent library routines as well as other modules specific to the vehicle, such as the interface between the cab and model, the control system logic, the aircraft math model and the software to drive the cockpit and lab displays. Vehicle model software can be developed at the VMS or by the visiting researchers.

Models, external processors, and software packages of common industry standard types can be integrated into the VMS system. Although the VMS library and legacy code is written in FORTRAN, all major languages and software environments are supported including, but not limited to, C/C++, FORTRAN, and Ada. Any modules in the area marked vehicle specific in the VMS Software Structure diagram shown in Fig. 7 can be integrated into the VMS.

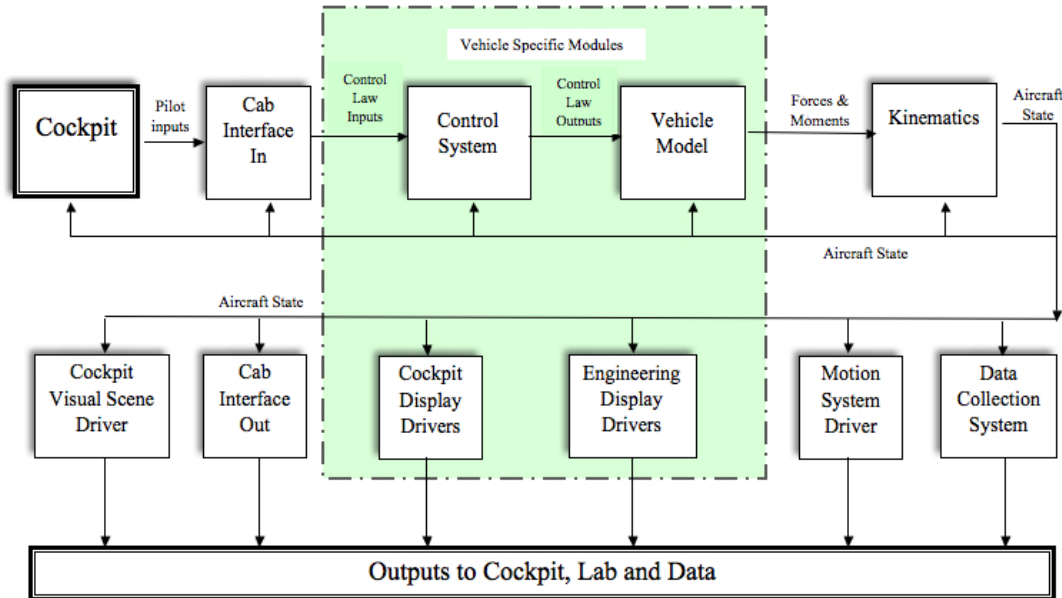


Figure 7. Vertical Motion Simulator Software Structure

III. Research Requirements that have Driven Simulink Integration

Integration of MATLAB Simulink models is largely driven by research model requirements. Simulink models may be developed at the VMS, accepted directly from the visiting research team, or as a software package from third party sources. The diversity of the model origins is indicative of the differences in their characteristics. Models may include external code, m-file code, lookup tables, and specialized Simulink blocks. For most blocks, Simulink is able to directly convert core MATLAB functions into RTW generated code; however, external code resources must be handled through MATLAB's S-function mechanism. In addition to the variation in code sources, Simulink models also have a variety of run modes. The VMS simulator architecture is a hard real-time, continuous, hold, and run system; therefore, the run and hold mode logic must be introduced during the conversion process. The integration procedure must be flexible in order to cater to a variety of model requirements while maintaining clear accounting of the conversion process in order to ensure maintenance of model integrity.

The Simulink models that have been integrated at the VMS range in size and complexity. They can be organized into three general categories: flight instrument drivers, advanced control laws, and complete aircraft models. One example of the former is a guidance model, programmed in Simulink, which calculated pursuit guidance and flight director commands, that was used to drive new symbols on a PFD during an aircraft handling qualities experiment.¹⁵ Another example from a recent experiment utilized a complete moving map HDD with fuel contour lines that was driven from an integrated Simulink model. Several helicopter experiments over the past years have studied modern or advanced control laws which were developed in Simulink. One project that studied the handling qualities of different classes of rotorcraft utilized five different Simulink models, each of which contained a full helicopter simulation. A notable recent Simulink integration effort at the VMS comprised a series of experiments that studied the handling qualities of a notional large tilt-rotor. For these tilt-rotor simulations, the entire aircraft model including fully-moveable nacelles, a rotorcraft turbulence model and a dynamic check capability, was provided in Simulink.¹⁶ The applications of Simulink are diverse, and this reality presents a challenge for existing integration standards.

IV. Process for Integrating Simulink Models with the VMS

The process for integrating Simulink models into the VMS host environment is designed to achieve several goals. The integration must maintain the integrity of the functions provided by the model. The dynamic behavior of the model must pass a series of tests to ensure this reliability. The process must be robust enough to afford researchers the ability to perform development in Simulink while retaining the capacity to monitor, modify and record all desired variables during the experiment. In addition to preserving model integrity and access to needed data, the process must also meet facility expectations for rapid deployment of new changes to the experiment.

A process was developed to integrate Simulink models into VMS simulations efficiently while allowing researchers the flexibility of continuing development work in the MATLAB environment. This procedure utilizes Real Time Workshop, a MathWorks utility that will generate C code from Simulink block diagrams. A number of challenges were solved and improvements made to the process as engineers became more familiar with the MathWorks products. Some of the lessons learned will also be discussed. The general steps of this procedure are as follows and are depicted in Fig. 8, Simulation Itegration Process.

- 1) Prepare the Simulink Model for integration into the VMS simulation environment.
- 2) Generate C code using RTW.
- 3) Compile the RTW code and build the model library.
- 4) Create RTW variable linkage by mapping the C structure variables into FORTRAN common blocks.
- 5) Create an interface wrapper to call the RTW code and manage the I/O
- 6) Link the RTW code library with other simulation modules to build an executable image
- 7) Test the model to verify the integrity of the integrated code.

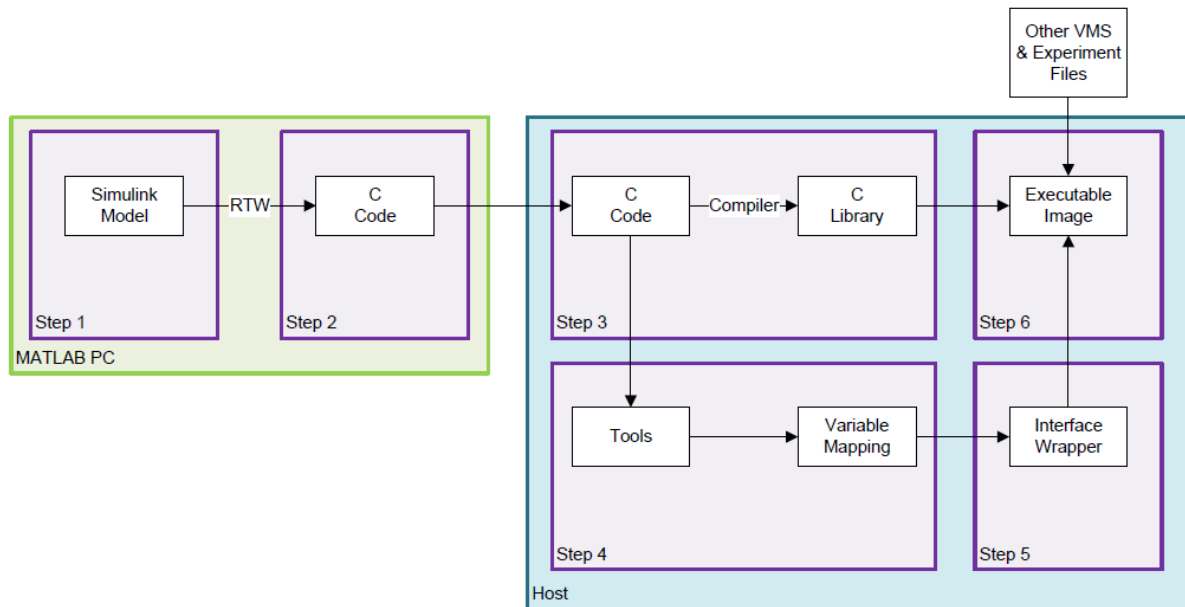


Figure 8. Simulink Integration Process

A. Simulink Model Preparation

The first step of the Simulink integration process is to prepare the model for code generation. On a desktop PC, the Simulink model is loaded into MATLAB. It is initialized, and the model data is loaded into the MATLAB Workspace, usually using script files to automate the process. The Workspace is the set of variables that exist in memory during a MATLAB session. A primary consideration during the model preparation step is to ensure that all needed inputs, outputs and parameters are accessible by the host symbol table in the generated code. RTW partitions model variables into a number of global data structures for this use.¹⁷ The global data structures the VMS integration utilizes are listed below.

- The model_U structure, which contains external inputs via the top-level Simulink Inport blocks and is used to send required inputs into the model from the host computer.
- The model_Y structure, which contains external outputs via the top-level Outport blocks and is used by the host to obtain the model outputs.
- The model_P structure, which contains block parameters that are used for a variety of purposes such as gains, scale factors and flags.

For a variable to be included in the global input or output data structures, it must be in the top-level block as an input port or output port. To ensure that all required inputs and outputs are included in the generated code, a number of signals must often be pulled to the top-level block and the ports correctly named. Access to inputs, outputs and named signals in code generated from Simulink models seems to be a common problem^{6,7,10} and as such will be discussed in some detail.

Input structure members take the name of the input port signal. Output port names, however, must be explicitly defined, using the “show name” option in the output port format pull-down, to provide the output structure member with the desired name. Otherwise, the output structure members will take generic Simulink symbolic names. By default, block parameters (such as gains and scale factors) are not tunable and are stored in the model_P structure with Simulink symbolic names. Sometimes this is not a problem, and access to the parameters is not needed. However, depending on the model being integrated, the ability to see and modify some block parameters is required. This capability is made available through the ‘Simulation Configuration Parameters’ dialogue box.¹⁸

Under the ‘Optimization Signals and Parameters’ option in the ‘Simulation Configuration Parameters’ pane, the ‘Inline Parameters’ option must be enabled. By default, this option is off which results in parameters being packed into the model_P structure with generic names and non-adjustable values in the generated code. When ‘Inline Parameters’ is enabled, parameters appear as constants in the generated code. To make the desired parameters tunable, they must individually be removed from inlining and given a storage class. This is done in the ‘Model Parameter Configuration’ pane.

Within the ‘Model Parameter Configuration’ dialogue box, all variables that exist in the MATLAB workspace are loaded. The list is used to select the names of the workspace variables that need to be adjustable. To make a variable adjustable, it must be selected and added to the table labeled “Global (tunable) parameters.” When a parameter is selected to be global (tunable) the user can control whether or not it is included in the model_P structure by declaring the storage class. The default Simulink storage class for the global parameters is ‘SimulinkGlobal,’ which will leave the tunable parameter in the model_P structure. If the ‘ExportedGlobal’ storage class is selected the parameter is exported as a global variable independent of the Simulink data structures. The ‘ExportedGlobal’ storage class is typically chosen for any variable which needs to be modified, recorded or used in any way.

In addition to declaring the necessary inputs, outputs, and parameters, a few other steps are required to configure the model for code generation. These are all done in the ‘Simulation Configuration Parameters’ box. First, the Solver is set to Fixed-Step type. As mentioned previously, the VMS runs in hard real-time, thus code execution is constrained such that all computations must be completed within a specific frame time. The fixed-step size solver is necessary to work within the hard real-time structure in order to guarantee the completion of code execution within that frame time. During one experiment, it was found that, even though the RTW generated code was created using a fixed step size, some Simulink blocks nevertheless had an incorrect frame time. This happened when the block sample time was hard coded instead of being set to inherit. For a hard real-time simulation, all blocks with a non-inherited frame time must be identified and corrected.

Next, the Generic Real-time (GRT) Target and the C language are selected. RTW has several different target configuration formats available, each of which are suited to different applications. GRT is the RTW target configuration that uses the real-time code format and supports external mode communication and static memory allocation in order to create custom rapid prototyping.¹⁷ It can execute in hard real-time when it is explicitly connected to a real-time clock. Last, since the RTW code is compiled on another computer, the Generate Code Only option is selected.

Often there are additional changes that must be made to the Simulink diagrams for the generated code to work within the VMS system. Variable initialization is sometimes a challenge, as it is for other facilities.¹¹ One reason this happens is that, in order to properly trim an aircraft and initialize subsystems, filters often need non-zero initial values. Simulink does not provide a means to set an initial value with their library transfer functions. To resolve this, the transfer functions must be reprogrammed from scratch using integrators and with an initial value input built in.

As mentioned previously, aircraft simulations must be trimmed and all models must be correctly initialized. However, Simulink models may not have compatible trimming methods or none may exist at all. Models also may

contain conflicting or incompatible handling control, execution schemes, and reset mechanisms. With one simulation the trimming problem was solved in the following manner. The Simulink model was modified to include a trim flag that would turn off some sections during the trim and would capture output values as they converged. To trim the Simulink model on the host computer, numerous variables required initial values, specific to each trim condition. Trim files that included these parameters were generated using an aircraft trimmer programmed in MATLAB, and then converted to a script file format that could be loaded on the host during real-time operation. Though less flexible than a real-time trimmer, this method nevertheless provided a stable trim for all initial conditions flown during the experiment. A real-time trimmer already available in the existing VMS software was modified to take account of all the necessary trim variables and was included in the build. Although this trimmer worked for many conditions, it took several minutes and did not always converge.

B. Real-Time-Shop Code Generation

The second step of the Simulink integration process is to use RTW to generate C code. This procedure is initiated in the Simulink model window by invoking the code generator. The generation progression can be monitored in the MATLAB Command Window. All generated files are listed as they are written. At the end of this process, a final statement is shown stating that the procedure has been successfully completed. If any errors occur during the code generation, they must be resolved before the generated code can be used.

RTW generates a set of source, header, and other files. Some are model specific and some are common RTW utility files. The model source comprises an initialize function, a terminate function, a step or update function and scheduling code.¹⁹ The generated files are transferred to the host computer. Additional source and header files may be needed which are not supplied by the RTW generation. For instance, a necessary top-level file that calls the model functions is not generated. The additional modules are identified and copied to the host as well. All are then integrated into the VMS simulation environment during the subsequent steps.

C. Generated Code Compilation

The third step of the Simulink integration process is to compile the RTW code and build the model library. This library consists of the RTW generated model specific files, RTW supplied generic routines, and other required MATLAB modules. One important routine is a top-level main-type file that can be used to call the model functions. A modified version of the MathWorks provided GRT target main program, grt_main.c, is used to provide the entry points into the RTW generated code. The same routine can be used for all Simulink models.

Makefiles are created to compile these routines on the hosts and link them into a model library. A makefile is a command file in which a group of routines to be compiled is listed. It also organizes the compiler switches, invokes the compiler and creates the libraries. RTW generates a text file which lists the values of some compiler specified defines, such as the exact model name and the number of continuous states, which must be set correctly. These definitions are handled through a series of compile switches set in the makefile.

RTW often creates long variable names, an occurrence which causes difficulties for the VMS as well as other simulation facilities.¹¹ On the Alphas, names longer than 31 characters are truncated. This often results in the variable names no longer being unique, a condition that causes a link-time error. A compile-time switch is added to the makefile that allows the names to remain unique while still being truncated. Finally, a compile-time switch in the makefiles is also used as part of the global variable mapping scheme, which will be discussed next.

D. Global Variable Mapping

The fourth step of the Simulink integration process is to map the C structure variables into FORTRAN common blocks. As discussed above, RTW generated C code data is partitioned into global structures that have all the root level inputs and outputs, as well as exported global variables (such as gains, scales, and flags). The generated code allocates storage for each data structure.¹⁹ Also discussed above, in the VMS simulation system, in order to have access to a variable for experiment set-up and monitoring, or for use on an I/O transfer list (e.g., simulator cab and display communication, and data collection), it must have a memory location in the symbol table and be a FORTRAN variable. Thus, a scheme was developed to map each C structure member to a FORTRAN common block element, and each C variable with an exported global storage class designation to a single element FORTRAN common block. In the RTW model header, the top level structures and exported parameters are automatically declared global using the external keyword which specifies that they will be defined elsewhere, thus allowing an external linkage. Fig. 9 shows a snippet of a RTW generated header in which the top level structures and exported global parameters are declared. In the makefile, discussed in subsection C above, a compile switch is used which makes the global definitions and creates a linkage between the FORTRAN common block and global C external

structures. Thus, the RTW C global external structure members and FORTRAN common block variables of the same name are equivalent and exist in the same memory location.

In the RTW Generated header file `My_Model.h`:

```
/* External inputs (root inport signals with auto storage) */
extern ExternalInputs_My_Model_U;

/* External outputs (root outports fed by signals with auto storage) */
extern ExternalOutputs_My_Model_Y;

/ * Exported Global Parameters
extern real_T Adjustable_Gain1;          /* Variable: Adjustable_Gain1
extern real_T Adjustable_Scale1;        /* Variable: Adjustable_Scale1
```

Figure 9. Example Generated Model Header-File Declarations

The simple compile time switch is now used to create the external variable linkage in lieu of a previous method which required a new header to be included in every generated source file, a task which was cumbersome and also had to be repeated for each iteration of code generation. This earlier method utilized the C code `#PRAGMA` statement to map any “external model” to a common block. Since the compile time switch is easy to implement and only set once, it is deemed the preferred technique.

The second half of the process to generate the C/FORTRAN linkage is the creation of a FORTRAN include file in which the common block variable definitions are made. The structure and exported global parameter declarations in the generated model header file are utilized for this, as well as a tool developed for this purpose, which is discussed below. Each RTW variable data type is also mapped to an equivalent FORTRAN data type, and the type declarations are put into the FORTRAN include file as well. This include file is used in the interface wrapper routine, discussed in subsection E below, as well as in all other modules that need access to the RTW C variables. A snippet of the declarations in the header file and the corresponding common block definitions in the include file are shown in Figs. 9 above and 10 below.

In this manner, two RTW structures, the external inputs and external outputs, are mapped into two similarly named FORTRAN common blocks. In addition, any number of exported global parameters – variables to which we need access but which were not included in the other structures – are mapped to individual FORTRAN common blocks. Sometimes, in addition to the model inputs, outputs, and exported global parameters, the constant parameters and block data structures are also needed and are then mapped to common blocks in the same way.

In the corresponding FORTRAN include file:

For root inport signals with auto storage in `ExternalInputs_My_Model` structure:

```
REAL (KIND=REAL_T) :: input_var1
REAL (KIND=REAL_T) :: input_var2

COMMON / My_Model_U / input_var1
COMMON / My_Model_U / input_var2
```

For root outports fed by signals with auto storage in `ExternalOutputs_My_Model` structure:

```
REAL (KIND=REAL_T) :: output_var1
REAL (KIND=REAL_T) :: output_var2

COMMON / My_Model_Y / output_var1
COMMON / My_Model_Y / output_var2
```

For Exported Global Parameters:

```
REAL (KIND=REAL_T) :: Adjustable_Gain1
COMMON / Adjustable_Gain1 / Adjustable_Gain1

REAL (KIND=REAL_T) :: Adjustable_Scale1
COMMON / Adjustable_Scale1 / Adjustable_Scale1
```

Figure 10. Example Common Block Definitions

The tool used in the variable mapping process (mentioned above) is a series of language parsers that have been developed to expedite and automate the introduction of RTW C code to the host environment. The parser is designed to convert from any source code to any target code by taking advantage of the repetitive structure of computer code. Computer code is largely defined by unique identifiers that designate the start and end point of different regions. Parsers can take advantage of these topographical features in computer code to map functional regions of one code into another. The tool is a set of functions used to manipulate language such as: find-and-replace, find-string, find-section, find-and-cut, and find-and-merge. These functions act as the basic building blocks that can be assembled using scripts and setting files in order to convert from one language to another automatically. Specialized rules and scripts for the language parsers are present to create the VMS compatible interface include files from the RTW generated modules. Once all variables are converted to compatible language sharing blocks, the source code can be combined using an interface.

E. RTW Interface Wrapper

The fifth step of the Simulink integration process is to create the interface wrapper. This interface has three parts: an executive routine to make the function calls that initialize and execute the RTW generated model appropriately, a module to prepare the inputs, and a module to handle the outputs. As discussed above in subsection B, the RTW generated model source comprises an initialize function, a terminate function, and a step or update function,¹⁹ but not a calling routine. Thus a “main” module, which calls the RTW generated functions, is added to the model library. It is called from the interface wrapper, once to initialize the model, and in every frame while it is running. The interface wrapper input module modifies existing VMS variables, or creates new signals as needed, to provide all necessary model inputs. Likewise, the interface wrapper output module accepts and modifies, as needed, the model outputs in order to provide any required hooks into the real-time system software (e.g. to drive the motion, visuals, or displays). Although the framework and much of the logic is the same for each Simulink simulation, the interface wrapper is specific to the particular model being integrated.

As was previously mentioned, variable initialization is sometime a challenge. Another reason this happens is that the RTW model initialization function zeros out all the inputs when it is called. To get around this, a second set of all the inputs are created in the wrapper code into which the desired initial values are stored. These values are then loaded back into the actual input variables after the initialization call has been completed.

F. Executable Image

The sixth step of the Simulink integration process is to create the simulation executable load. The RTW model library and data header are linked with the interface wrapper and other VMS module object files and libraries to create the executable image. This is done on the host by creating a top-level build file that lists the names and locations of all required modules and then invoking a MicroTau application load building procedure. The build procedure links the listed libraries, object files and data tables with the system libraries to create an executable image.¹³

G. Test and Verification

The seventh and final step of the Simulink integration process is to verify that the Simulink model was properly integrated and that no errors were introduced in the process. The simulation is tested using static and dynamic checks. For the static checks, the aircraft is trimmed in various conditions and the model outputs are compared to those in Simulink for the same configuration. For dynamic checks, time history data appropriate to the model integrated is recorded and compared to the corresponding data collected in Simulink and/or to data delivered by the researchers.

As experienced by other facilities, upgrades to MATLAB and Simulink versions require changes to the integration process.⁵ New releases have new features which result in changes to both the Simulink diagrams as well as the interface procedure. Moreover, it was discovered that all who are involved in a project – model developers and simulation engineers - must have the same version of MATLAB, Simulink and Real-Time Workshop. Code from newer versions is incompatible with that from older versions. Reconciling regular changes and updates to a multitude of diverging versions is a difficult challenge across industry. On occasion, the VMS facility has had to obtain older versions of MATLAB and Simulink in order to effectively integrate delivered models.

V. Embedding VMS Code into Simulink

Several mechanisms were developed to integrate Simulink models, and they have been modified to adhere to research and host requirements. RTW can generate C code for all standard Simulink blocks, and is described in

detail above. This mechanism accounts for the bulk of Simulink code compiled on the host. Sometimes, however, Simulink blocks may themselves reference external code blocks that may not be native to the MATLAB environment. In these cases, it is necessary to first convert the blocks into an S-function that can be referenced by Simulink during the link stage. MATLAB S-functions are Simulink blocks written in a computer language. The S-function types are described using the base language, e.g. C, C++, and Fortran, as well as the level of feature support desired. This paper focuses on Level 2 S-function in C. The Level 2 S-function defines the number of in and out ports as well as the initialization order of each data port, and it is a superset of all S-function features. Customization of this interface is largely responsible for conforming all model symbols with VMS host variables by making them available to the host after RTW converts the model to C code using the GRT Target. The VMS host environment uses a custom S-function template and conversion procedure to tailor the interface components as appropriate. This procedure is set once during simulation development and can be reused during trials to adapt changes. The repeatability of the process is key to operational efficiency and rapid transition between model changes.

The parser scripts from the RTW integration tool inherit the experience gained from a long history of collaborative simulation at the VMS facility. The parser tool package is related to code procedures used to parse DAVE-ML (Dynamic Aerospace Vehicle Exchange Markup Language) source, based on XML standards, to the VMS host environment.²⁰ DAVE-ML is a model exchange standard that aims to facilitate model exchange between different facilities. The VMS regularly encounters situations that require language-to-language conversions; therefore, software tools must be flexible to any language, repeatable, automatic, and contain self-check mechanisms. At present, the tools do not contain self-check and validation methods, which is one of several aspects still being developed for compliance with DAVE-ML standards. As such, check cases and procedures are in place to validate model behavior after conversion. The parser functions are currently sufficient and acceptable for Simulink conversion.

The third tool is a radical reversal of the conversion process. By providing a native Simulink port of existing VMS host capabilities, research teams can develop models with VMS components such as the physics equations of motion. The block is a custom function wrapper that can be selected and placed into existing Simulink diagrams such as any other native block. Input and output ports automatically populate, allowing predictable interface connections to develop within the model. The model gains the ability to simulate exact functionality provided by the VMS because the function block encapsulate the compiled version of identical code object in the VMS. This capability is currently provided upon request as an effort to further extend the usefulness of NASA capabilities to the research community. Modules developed for use at the VMS are time-tested tools that have been proven valuable and effective for use in many air and space vehicle simulations. In addition to the benefit to the research model, the native Simulink port is able to build directly from Simulink RTW as a generic target. The ports translate directly into VMS host ports with the proper symbolic names and allow the generated C code to integrate directly into the existing VMS host environment; therefore, a significant amount of time is saved by skipping steps such as ensuring port alignment and variable interfacing.

The Simulink port wrapper template can be extended to encapsulate model code. The template and procedure can wrap aerodynamic lookup tables, and was used to convert an F-16 reference model^{21,22} during the development of a new simulation architecture at the VMS facility for the Operational Based Vision Assessment (OBVA)²³ simulator, a joint venture between NASA Ames and the U.S. Air Force. The wrapper allowed for rapid, automated, and repeatable introduction and interface of aircraft flight characteristics in hours, a procedure that may take weeks traditionally. The OBVA simulator is a good candidate for the wrapper because it operates with MATLAB Simulink as the real-time data analysis interface. The system is MATLAB centric and all aircraft control, physics, and aerodynamic models reside in Simulink. Native Simulink port modules add to model utility and reduce procedural complexity during integration with the VMS host environment.

VI. Conclusions

The VMS, with its realistic motion cueing and flexible simulation operating environment, is an ideal platform for simulating real and notional vehicles. Feedback from the research community indicated a demand for seamlessly integrating models developed in graphical environments, such as MATLAB/Simulink, while maintaining the ability to perform rapid data processing and ensuring data integrity. In response to this demand, the VMS facility developed a process to quickly incorporate Simulink models into the VMS simulation environment. Real Time Workshop is used to generate C code from the Simulink model on a desktop PC. The generated files are then transferred to the host computer, and compiled. An interface wrapper is created which includes a mechanism to link the Simulink variables to the host system. The RTW generated code is included in an executable image with other VMS software.

Once the integrity of the integration is thoroughly verified, the Simulink model is fully integrated into the VMS simulation environment.

This procedure to integrate Simulink models into the VMS has been used successfully for several piloted simulation experiments. The VMS's flexible real-time software infrastructure ensures that current and future changes to Simulink can be accommodated in a fast and efficient manner. Familiarity with the MathWorks tools and an ability to quickly integrate Simulink models into the VMS has greatly enhanced the ability to rapidly develop, integrate, and simulate vehicle models as required by research programs. This capacity to integrate Simulink models directly into the VMS real-time simulation structure has improved efficiency and reduced potential errors, while allowing researchers to retain the development flexibility of the MATLAB environment.

Acknowledgments

The authors wish to thank Bosco Dias, Gordon Hardy, and Ben Lawrence for their contributions and expertise.

References

- ¹Danek, G. L., “Vertical Motion Simulator Familiarization Guide”, NASA TM-103923, 1993.
- ²Aponso, B. L., Tran, Duc T., and Schroeder, J. A., “Rotorcraft Research at the NASA Vertical Motion Simulator,” NASA Ames Research Center, AHS Annual Forum, Montreal, Canada, April-May, 2008.
- ³Aponso, B. L., Beard, S. D., and Schroeder, J. A., “The NASA Ames Vertical Motion Simulator – A Facility Engineered for Realism”, NASA Ames Research Center, Royal Aeronautical Society Spring 2009 Flight Simulation Conference, London, UK.
- ⁴Magyar, T. J., Page, A. B., “Integration of the CASTLE Simulation Executive with Simulink,” Naval Air Systems Command, Patuxent River. MD, AIAA May 2001.
- ⁵Christhilf, D.M, and Bacon, B. J., “Simulink-Based Simulation Architecture for Evaluating controls for Aerospace Vehicles,” NASA Langley, AIAA.
- ⁶Hogg, E.F., “B-737 Linear Autoland Simulink Model”, NASA/CR-2004-213021.
- ⁷Allen, M. J., Beyer, E.W., Hales, S.A., and Ilvedson, C. R., “Leveraging MathWorks Tools to Develop a Simulation Framework for Diverse Customers,” AIAA Modeling and Simulation Technologies Conference, August 2011, Portland, Oregon.
- ⁸Landers, S., “Real-time Pilot-in-the-Loop and hardware-in-the-Loop Simulation at Gulfstream”, ADI Users Society, San Diego, California, December, 2007.
- ⁹Quaranta, G., Mantegazza, P., “Using MATLAB-Simulink RTW to Build Real Time Control Applications in User Space with RTAI-LXRT”, Dipartimento di Ingegneria Aerospaziale, Politecnico di Milano, Milano, Italy
- ¹⁰Moelands, J. M., et al, “Automatic Model Transfer from MATLAB/Simulink to Simulation Model Portability 2”, NLR-TP-2006-674, National Aerospace Laboratory NLR,SESP, Noordwijk, November 2006.
- ¹¹Bodeman, C. D. and DeRose, F., “The Successful Development Process with MATLAB Simulink in the Framework of ESA’s ATV Project”, IAC-04-U.3.b.03, Vega IT GmbH, Darmstadt, Germany.
- ¹²Schroeder, J. A., “Helicopter Flight Simulation Motion Platform Requirements,” NASA Ames Research Center, NASA/TP-1999-208766, Moffett Field, California, AIAA, July 1999, p 69.
- ¹³Schroeder, J. A. and Grant, P. R., “Pilot Behavioral Observations in Motion Flight Simulation,” Federal Aviation Administration, Moffett Field, CA, 94035, and University of Toronto, Toronto, Ontario, Canada, M3H 5T6, AIAA Modeling and Simulation Technologies Conference, 2-5 August 2010, Toronto, Ontario, Canada.
- ¹⁴MicroTau Users Guide, Contract No. NAS 2-98084, NASA Ames Research Center, Moffett Field, California, 9403.5
- ¹⁵Hardy, G. “Programmable Portable Guidance Display Users Manual”, NASA/TM-2012-215983, 2012, proposed.
- ¹⁶Lawrence, B., Malpica, C.A. and Theodore, C. R., “The Development of a Large Civil TiltRotor Simulation for Hover and Low-Speed Handling Qualities Investigations,” NASA Ames Research Center, Moffett Field, California, European Rotorcraft Forum, Paris, France, September, 2010.
- ¹⁷The MathWorks Training Services, “Real-Time Workshop Fundamentals” The MathWorks, Inc., 2009.
- ¹⁸“Real-Time Workshop Users Guide”, The MathWorks, Inc., Natick, MA 01760-2098, 2011
- ¹⁹McBroom, M., Erkinen, T., and Behr, M., “Integrating Simulink with other Simulation Environments,” AIAA-2010-7776.
- ²⁰Murri, D. G., and Jackson, E. B., “Flight Simulation Model Exchange,” NASA TM-2011-217085/Volume I, NESC-RP-09-00598, Apr. 2011.

²¹Nguyen, L. T., Ogburn, M. E., et. Al., "Simulator Study of Stall/Post-Stall Characteristics of a Fighter Airplane With Relaxed Longitudinal Static Stability," NASA Technical Paper 1538, Dec. 1979.

²²Russell, R. S., "Non-linear F-16 Simulation using Simulink and Matlab," Technical Report, University of Minnesota, June 2003.

²³Sweet, B. T., Giovannetti, D. P., "Design of an Eye Limiting Resolution Visual System Using Commercial-Off-the-Shelf Equipment," AIAA-2008-6847.