Integrated G&C Implementation within IDOS - A Simulink Based Reusable Launch Vehicle Simulation

Joseph E. Fisher, Tim Bevacqua,
Douglas A. Lawrence and J. Jim Zhu
School of EECS - Ohio University
Athens, Ohio 45701

Michael Mahoney
Universal Space Lines, LLC.
New Port Beach, California 92660

ABSTRACT

The implementation of multiple Integrated Guidance and Control (IG&C) algorithms per flight phase within a vehicle simulation poses a daunting task to coordinate algorithm interactions with the other G&C components and with vehicle subsystems. Currently being developed by Universal Space Lines LLC (USL) under contract from NASA, the Integrated Development and Operations System (IDOS) contains a high fidelity Simulink vehicle simulation, which provides a means to test cutting edge G&C technologies. Combining the modularity of this vehicle simulation and Simulink's built-in primitive blocks provide a quick way to implement algorithms. To add discrete-event functionality to the unfinished IDOS simulation, Vehicle Event Manager (VEM) and Integrated Vehicle Health Monitoring (IVHM) subsystems were created to provide discrete-event and pseudo-health monitoring processing capabilities.

Matlab's Stateflow is used to create the IVHM and Event Manager subsystems and to implement a supervisory logic controller referred to as the Autocommander as part of the IG&C to coordinate the control system adaptation and reconfiguration and to select the control and guidance algorithms for a given flight phase. Manual creation of the Stateflow charts for all of these subsystems is a tedious and time-consuming process. The Stateflow Autobuilder was developed as a Matlab based software tool for the automatic generation of a Stateflow chart from information contained in a database. This paper describes the IG&C, VEM and IVHM implementations in IDOS. In addition, this paper describes the Stateflow Autobuilder.

INTRODUCTION

In a robust, fault tolerant flight control system, a controller design that incorporates vehicle health status, sensor fidelity, and control effector limits is crucial in order to maintain stability under adverse conditions. This, in part, involves monitoring and reacting to possibly hundreds of signals generated by the vehicle health monitoring system in order to detect and react to failures. A system that undergoes adaptation or modification based upon discrete events is referred to as a hybrid system.

Simulation of a system with continuous dynamics that are influenced by discrete events [1] such as failures is a necessary tool for the development of a hybrid control system. One simulation software package on the market is Matlab with Simulink and Stateflow developed by The Mathworks.

Stateflow provides the capability within a Simulink model to implement supervisory logic that interprets system monitoring inputs so that a hybrid, adaptive control system can be developed and simulated. The Stateflow Autobuilder [2] software tool has been developed to generate working Stateflow charts from a Microsoft Access database or from an Excel spreadsheet. In the Microsoft Access format, a GUI provides a user-friendly interface for beginner level users to enter and maintain the database, and the Excel format allows advanced users to quickly build the database by using a spreadsheet format.

The Stateflow Autobuilder supports a multi-university effort to develop integrated guidance and control technology for NASA's second generation reusable launch vehicle that features a top-level supervisory controller referred to as the Autocommander [3]. The Stateflow Autobuilder can easily be adapted for more general use in discrete-event or hybrid flight control system design and simulation, such as implementing Vehicle Event Manager (VEM) and Integrated Vehicle Health Monitoring (IVHM) subsystems that are presented in this paper.

Universal Space Lines (USL) has taken advantage of The Mathworks' Simulink simulation package to create a generic simulation model that is loaded with specific vehicle parameters to form a high fidelity vehicle simulation. As different vehicle models arise, only the specific parameters for a given vehicle are loaded into Integrated Development and Operations Systems (IDOS) to create a new vehicle simulation. Once G&C algorithms have been implemented in IDOS, the modular implementation design inherent to Simulink models presented in this paper provides easy swapping out of modified or updated G&C components.

Also presented in this paper are the results of adding Time Varying Bandwidth (TVB), which is one component of the fault tolerant flight control system. Time Varying Bandwidth is based upon singular perturbation theory [4]. The fast inner loop and the slow outer loop bandwidths should abide by the time scale separation principle in order to ensure stability. Upon receiving the TVB flag from the Autocommander, the controller appropriately scales back the bandwidths of both loops, which slows the controller and maintains the required time scale separation [5,6].

The implementation of the IG&C described in this paper occurred within less than 2 months.

IDOS DESCRIPTION

Integrated Development & Operations System (IDOS) is a software development environment for launch vehicle flight mechanics, which is currently under development by USL. IDOS consists of three components: IDOS Management System (IMS), Flight Control Software (FCSW –Figure 1) and Vehicle (and Environment) Simulation, and Real Time Test Bed (RTTB).

Major components of IDOS are the Flight Control Software and Vehicle (and Environment) Simulation (FVS), implemented in a Simulink model. The FCSW portion of FVS is where the integrated guidance and control algorithms are implemented. The block-diagram structure of Simulink allows for easy implementation and development of these algorithms. Also, the capability of incorporating C and Fortran S-functions allows for fast implementation of pre-existing code.

The IDOS Management System and Real Time Test Bed are currently being developed by USL. These components are part of the complete IDOS software tool that will allow faster than real time and real time simulations.

ALGORITHM IMPLEMENTATION

The block diagram structure of Simulink enabled a very modular implementation of the guidance and control algorithms as shown in Figures 3 thru 5. The modularity was achieved by using subsystem blocks to form a hierarchal framework for the algorithms to reside in. This modularity provides an easy way to update individual algorithms, as well as compare different algorithms.

The top level of this hierarchy contains the Navigation, Guidance, Control, and Autocommander subsystems, shown in Figure 2. These four subsystems are the key components of the integrated guidance, navigation and control (IGNC) algorithms.

The next level of the hierarchy separates the five flight phases in the guidance and control subsystems, as shown in Figure 3 for the control subsystem. The guidance and control systems for each flight phase are contained within enabled subsystems. A signal from the Autocommander then enables the appropriate subsystem for simulation.

This structure allows the user to make changes to different algorithms without affecting the rest of the simulation. As long as the input and output structure is maintained, old blocks can be cut out and new ones inserted. This provides an easy way to update the model as improvements to individual algorithms are made.

The next level of the hierarchy further separates the algorithms into individual components. An example is shown in Figure 4, where the ascent controls block is separated into a controller block and a control allocation block.

The separation of the control algorithms makes the model extremely modular, allowing very specific changes to be made without altering the rest of the simulation. It also enables multiple algorithms to be included within one model as shown in Figure 5.

If different algorithms exist for a single component, switching between algorithms is done by changing the value of an algorithm selection variable. This variable enables the subsystem that is desired to be used. Figure 5 shows the two types of controllers, Trajectory Linearization Control (TLC) and Sliding Mode Control (SMC), implemented in the same model. An initialization program is run prior to simulation where the user is prompted to choose the different algorithms. This allows different algorithms to be compared easily, with no modification to the simulation code.

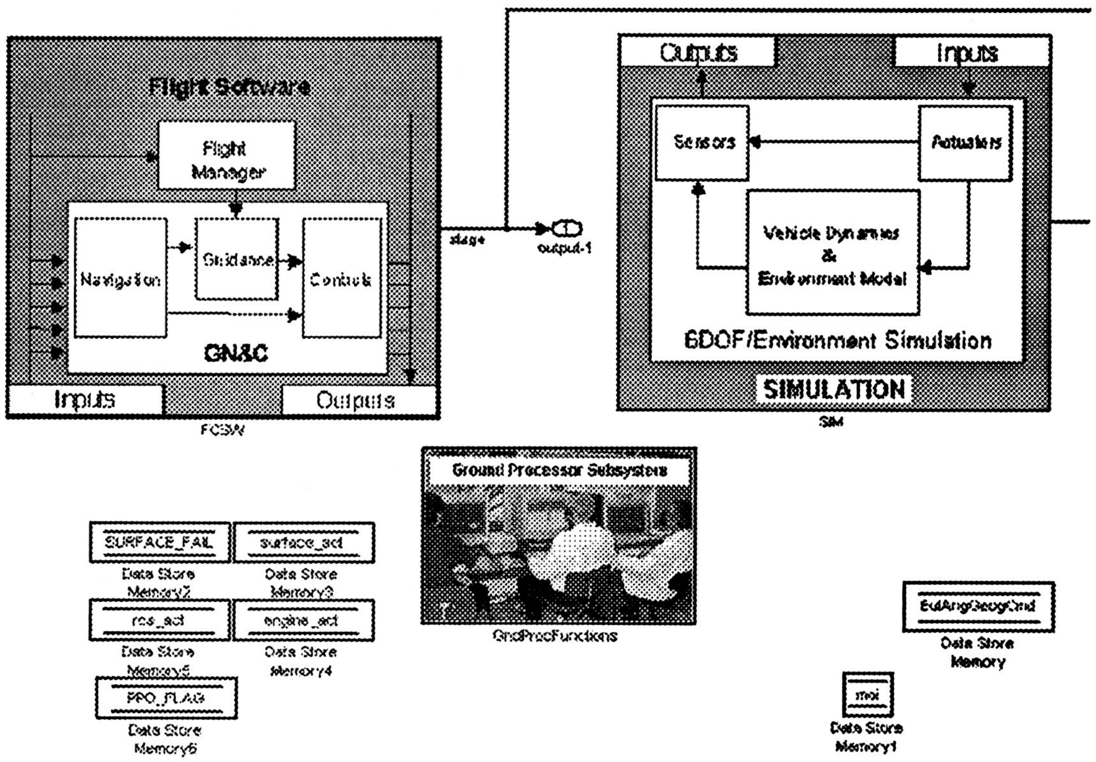


Figure 1: Top Level System of the FVS.

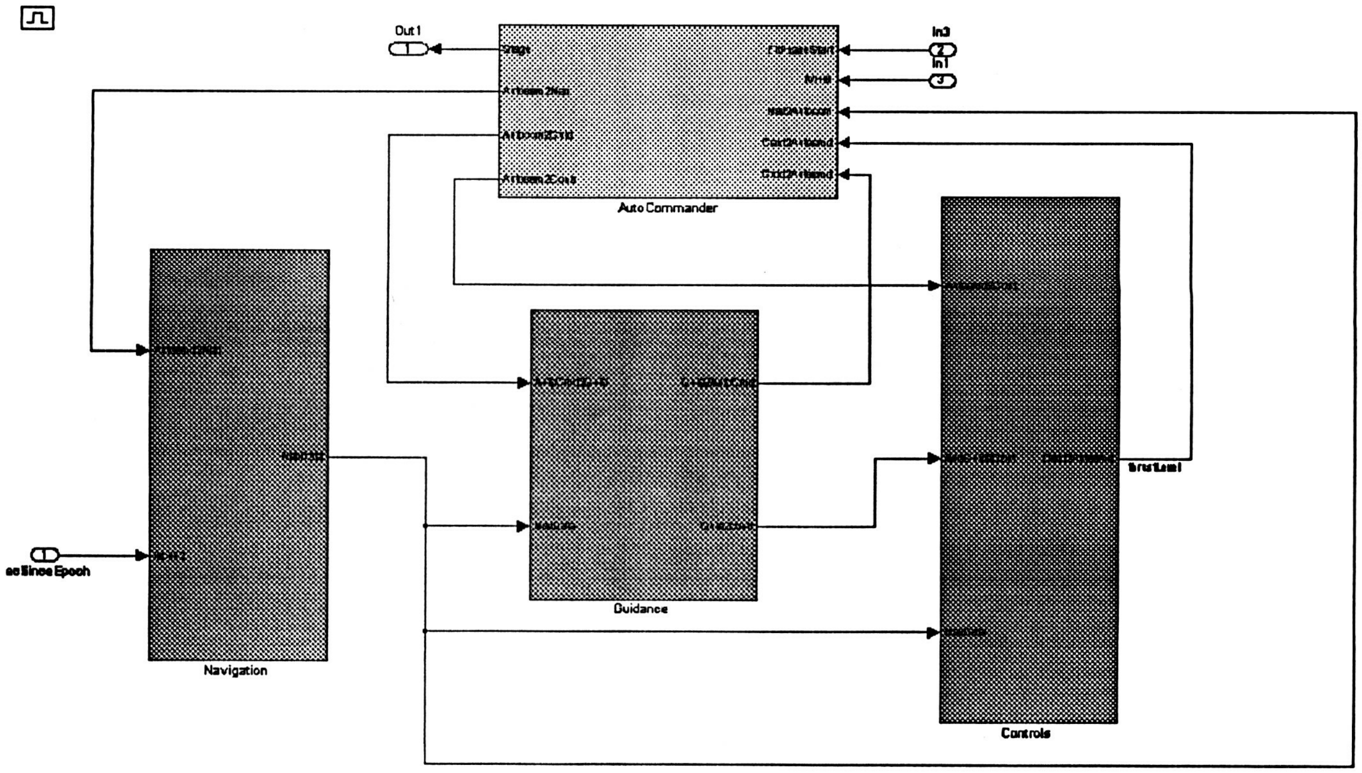


Figure 2: The IG&C Subsystems.

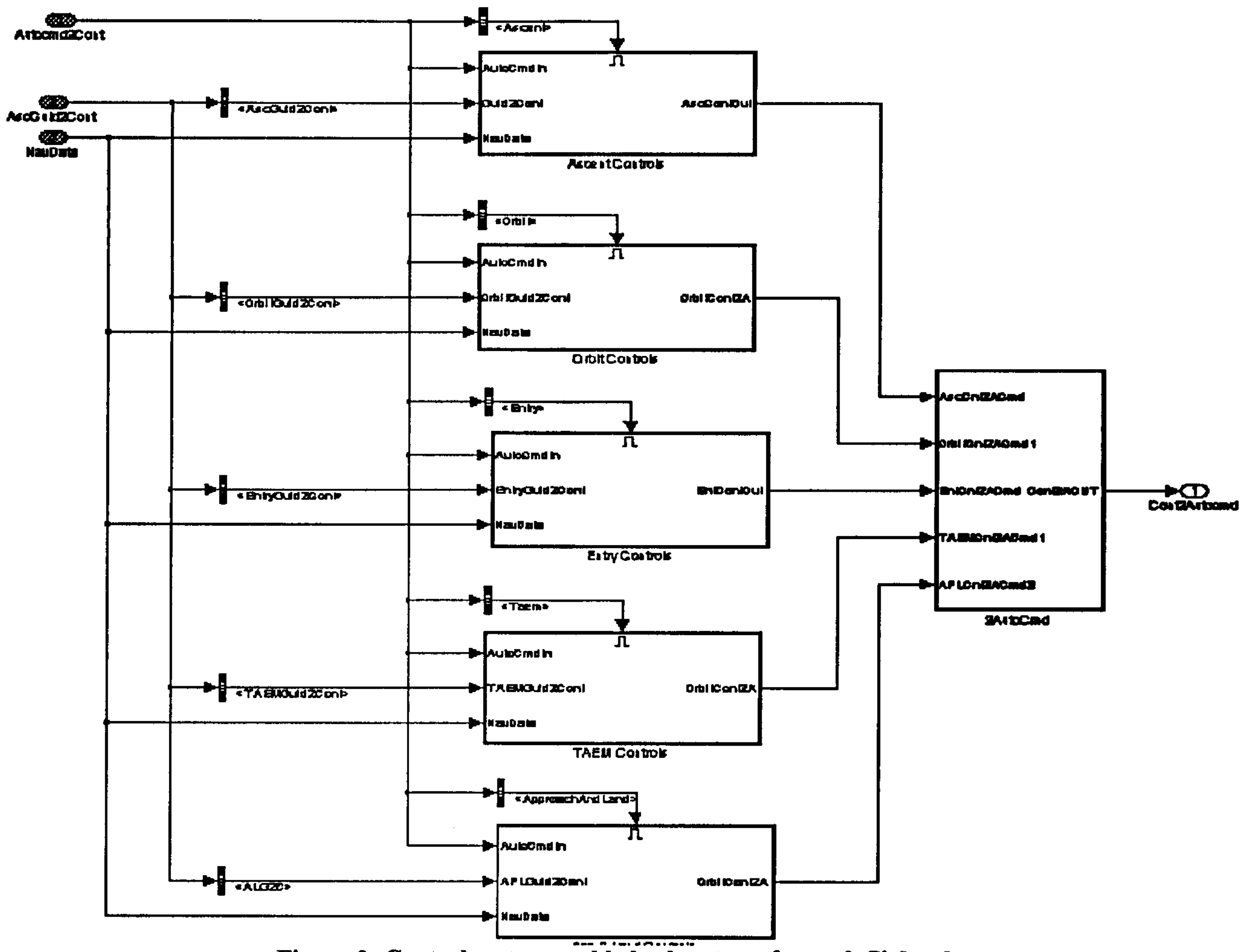


Figure 3: Control system enabled subsystems for each flight phase.

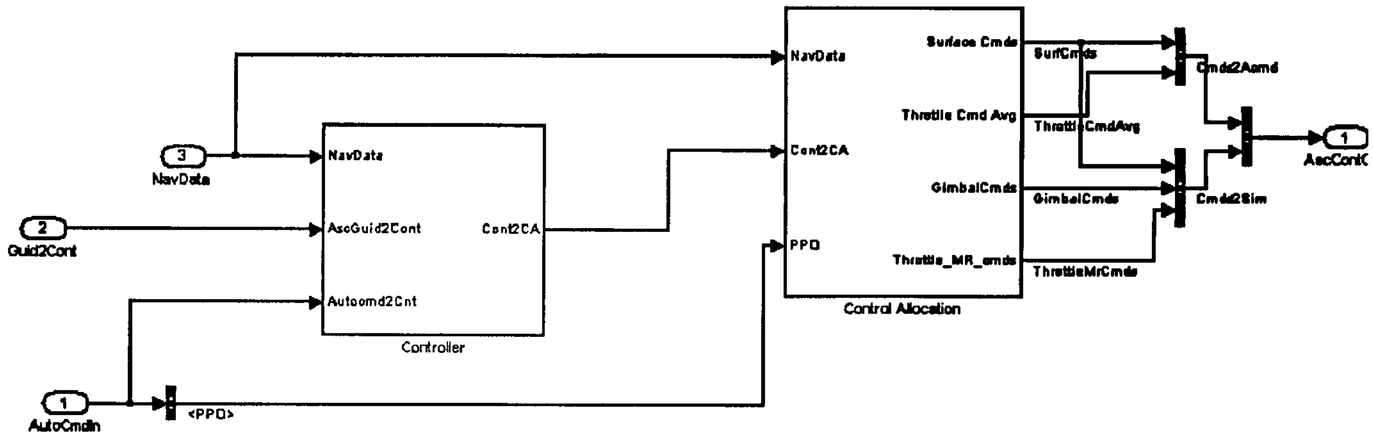


Figure 4: The controller and control allocation subsystems.

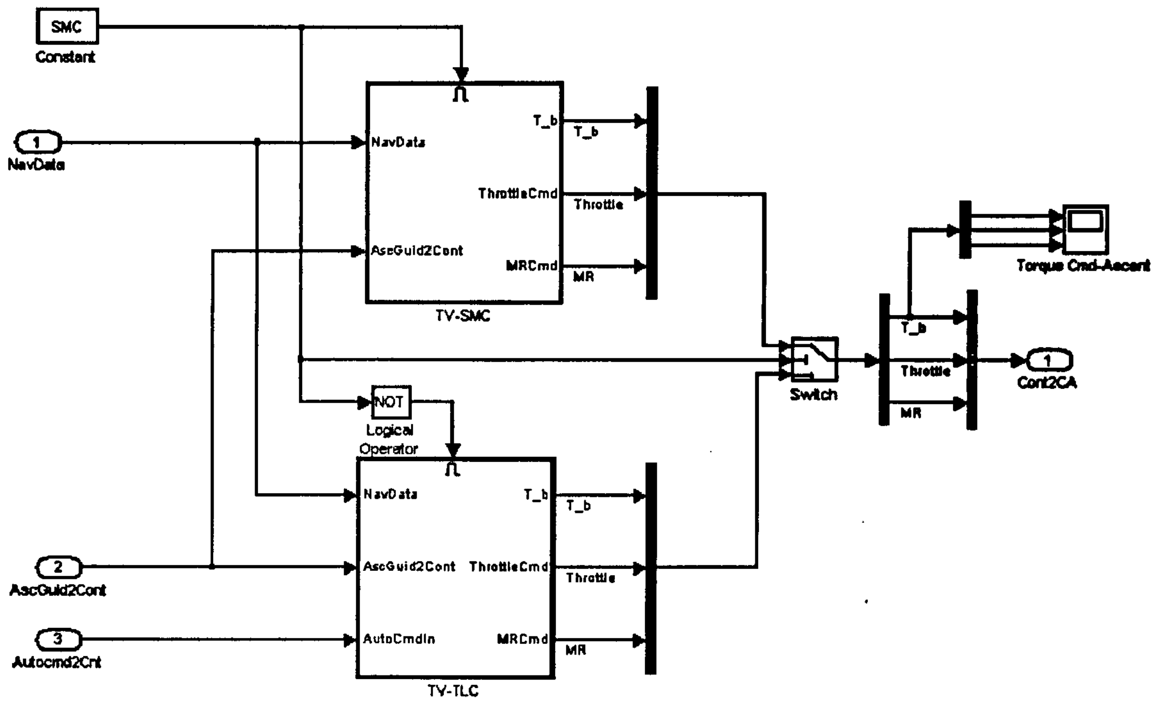


Figure 5: Choice of controller types: Sliding Mode Controller and Trajectory Linearization Controller.

STATEFLOW SUBSYSTEMS

In the past, creating a Stateflow chart for use in a Simulink simulation of a large-scale guidance and control system was a tedious and time-consuming process. Manual manipulation of Stateflow's graphical user interface (GUI) was required to individually create elements of a Stateflow chart [7]. Moreover, updating a chart to reflect revisions posed a daunting task for a large system. An alternative solution to overcome these problems is to be able to automatically produce a Stateflow chart from a database. Managing the parameters and elements of a simulation via a database provides the ability to efficiently modify or update the simulation model by simply changing information within the database.

The Autocommander, Vehicle Event Manager and IVHM subsystems use Stateflow charts. These charts are automatically generated from a spreadsheet/table database and inserted into the IDOS environment via the Stateflow Autobuilder software as describe in the following subsection.

Stateflow

Stateflow supports the execution of concurrent automata by setting the decomposition parameter of the chart to parallel. The states that are immediate children of this chart are referred to as AND (parallel) states, which are distinguished by dashed borders as shown in Figures 9 & 10. This is a particularly important feature of Stateflow that allows a chart to be separated into different automata. In Figures 9 & 10, there are two and three, respectively, separate automaton that operate concurrently. The traditional states of an automaton are called OR (exclusive) state, which have solid line borders.

Another key feature of Stateflow is that when a transition occurs or while in a state (entry, during, exit and on event) actions can be implemented. Actions may consist of setting flags, calling Stateflow and Matlab functions, defining variables, and broadcast events, etc.

Two different types of operators can trigger a transition from one state to another, an event broadcast or a valid logical expression. Labels of transitions may contain either one or both types of operators. The scope of an event dictates which Stateflow objects receives its broadcast, which triggers a transition. It can be an "Input from Simulink", "Local" to a group of states, or "Output to Simulink". The syntax of the logical expression is identical to an "if" statement in standard Matlab syntax with an exception of being encased within a square brackets. The data variables that are used within a transition's logical expression or within a states action statement must be defined in the data dictionary of the chart. Data variables have a scope of "Local", "Input from Simulink", "Output to Simulink", "Temporary," or "Constant". Either "Input from Simulink" or "Output to Simulink" scope for both events and data variables require connection of the Stateflow chart to the Simulink model.

Stateflow Autobuilder

The Autobuilder uses the Application Programming Interface (API) of Stateflow 4.2 (and later) to automate the construction of a complete working chart. API only provides a way to insert Stateflow objects into a chart, but not the capability to automatically define a Stateflow object's parameters, such as position, size, or the label of a state. Beyond the typical states, events and transitions that characterize automata, the Autobuilder creates data variables, junctions, functions (graphical or text) and the remaining Stateflow objects to allow users to utilize the full features of Stateflow in the development of a hybrid system.

In addition to generating a Stateflow chart, wireless connection blocks (Goto and From blocks) are generated to provide a plug and play capability for the chart. Figure 11 shows a generated Stateflow chart and the wirelessly connected input and output ports that are generated by the Autobuilder. Within the Simulink model, there must be preexisting Goto/From blocks with tags that respectively match the tags of the generated Goto/From blocks to complete the "connection" that is normally made with a line. A tag is a label that defines the connection between a Goto block and a From block and vice-versa. The Goto/From blocks that are generated are tagged with the output/input variable or event name concatenated with the string "PORT". For example, an output to Simulink variable that is named "input1" would be connected to a Goto block with a tag of "input1PORT". Additionally, anywhere in the Simulink model that this variable value or signal is used would require a connection to a From block with a matching tag of "input1PORT".

With an automaton in mind, the first task in generating a chart is to fill out the fields in the database. The first spreadsheet/table to be filled out is for the state information. Within this spreadsheet, the state's name, parents name, entry action label, during action label, exit action label, on event action label and the type (AND/OR) of state are the required information for generating states, which is shown Figure 6. For transitions, the required information fields that are shown in Figure 7 are the transition's label, source (name), and destination (name). For data variables the required fields are name, parent name, data scope, data type and initial value as shown in Figure 8. In a similar fashion, additional spreadsheets/tables are required to be filled out for events, functions, boxes, notes, and junctions.

After the Stateflow Autobuilder program is executed, the resulting Stateflow chart is shown in Figure 11. The placement of the Stateflow chart block into a Simulink model is based upon the position of the chart block it is going to replace. To insert a new chart into a simulation, first a new chart must be copied and placed by the user from the Simulink block library. In addition, sufficient space must be available around the existing Simulink model prior to running the Autobuilder to allow proper placement and sizing of the Stateflow chart without overlay of other blocks. The graphical layouts of the states, sizes of the states as well as the paths of the transitions that are generated are organized in such a manner that the produced Stateflow chart is generally compact and easily readable.

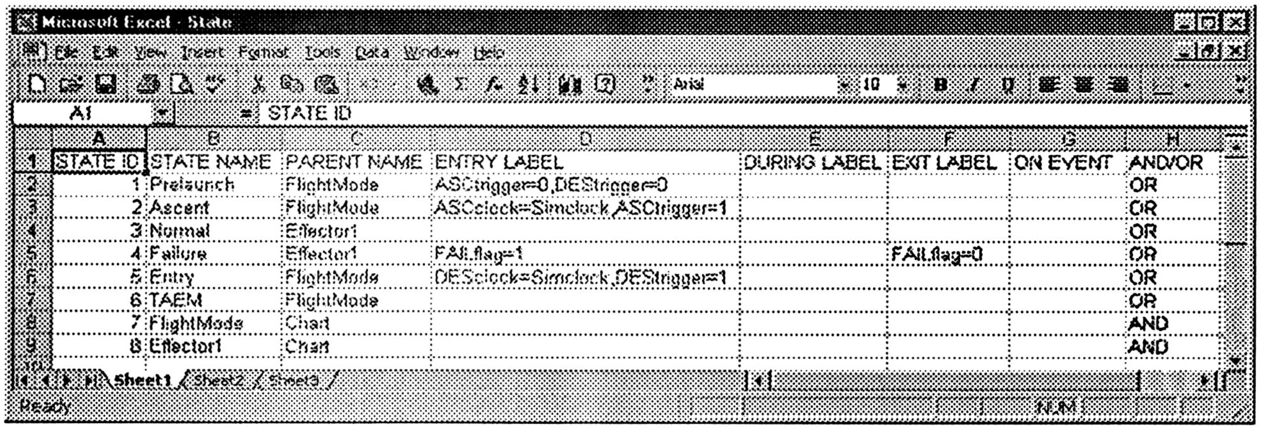
Vehicle Event Manager

The addition of the Vehicle Event Manager (VEM) makes IDOS a powerful tool to simulate off-nominal flight scenarios that are associated with a discrete-event driven simulation. The ease of generating vehicle failures provides a method to test and validate G&C algorithms very quickly under various failure scenarios. Unlike other vehicle simulation, any possible failure scenario can be implemented and up-and-running in just a few minutes.

In Excel spreadsheets, four parameters are to be entered per failure event, and any combination of failure events can form a failure scenario. The first parameter is the start time of a failure event. The second parameter is the end time of the event. The third parameter is the type of failure. Finally, the fourth parameter is a 0 to 1 scalar multiplier or a set point at which the component has failed, which is specific to the system and dependant on the type of failure. Currently, only one failure type can occur per component per simulation run. Eventually, the VEM capabilities will be extended by vectorizing the event parameters, which will allow multiple types of failures to occur at different times per failed component.

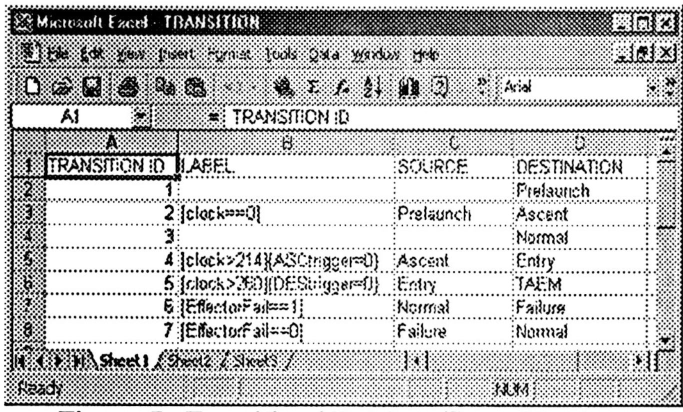
A Matlab script file reads the Excel spreadsheet data and converts it into a text file. Within the Vehicle Event Manager subsystem, an S-function reads the text data and inputs it into the VEM's Stateflow chart.

Currently within the VEM's Stateflow chart, there are 42 generic YES/NO (ON/OFF) events as shown in the upper AND superstate within Figure 9. These events only use the start and stop times to set a flag over the corresponding time interval. Also, there are 22 effector events that are comprised of 8 surface deflector failure events, 10 RCS thruster failure events, and 4 engine failure events as shown in the lower AND superstate in Figure 9. These events use start and stop times to represent effector failures over the corresponding time interval. The associated event type that is used to specify the failure mode are: 1(nominal), 2 (reduced control authority), 3 (set at a position), and 4 (stuck at current position). The failure value specified in the fourth parameter dictates either a 'set at' position or a signal multiplier value between 0 and 1 (0% to 100%) to characterize remaining control authority. This parameter is not used for the stuck effector failure event.



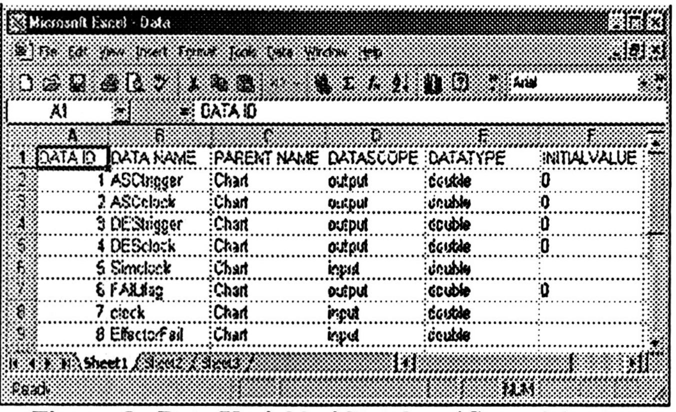
STATE ID	STATE NAME	PARENT NAME	ENTRY LABEL	DURING LABEL	EXIT LABEL	ON EVENT	AND/OR
1	Prelaunch	FlightMode	ASCtrigger=0,DEStrigger=0				OR
2	Ascent	FlightMode	ASCclock=Simclock,ASCtrigger=1				OR
3	Normal	Effector1					OR
4	Failure	Effector1	FAILflag=1		FAILflag=0		OR
5	Entry	FlightMode	DESclock=Simclock,DEStrigger=1				OR
6	TAEM	FlightMode					OR
7	FlightMode	Chart					AND
8	Effector1	Chart					AND

Figure 6: States' Information Spreadsheet.



TRANSITION ID	LABEL	SOURCE	DESTINATION
1			Prelaunch
2	[clock==0]	Prelaunch	Ascent
3			Normal
4	[clock>214]{ASCtrigger=0}	Ascent	Entry
5	[clock>360]{DEStrigger=0}	Entry	TAEM
6	[EffectorFail==1]	Normal	Failure
7	[EffectorFail==0]	Failure	Normal

Figure 7: Transitions' Database/Spreadsheet.



DATA ID	DATA NAME	PARENT NAME	DATASCOPE	DATATYPE	INITIALVALUE
1	ASCtrigger	Chart	output	double	0
2	ASCclock	Chart	output	double	0
3	DEStrigger	Chart	output	double	0
4	DESclock	Chart	output	double	0
5	Simclock	Chart	input	double	
6	FAILflag	Chart	output	double	0
7	clock	Chart	input	double	
8	EffectorFail	Chart	input	double	

Figure 8: Data Variables' Database/ Spreadsheet.

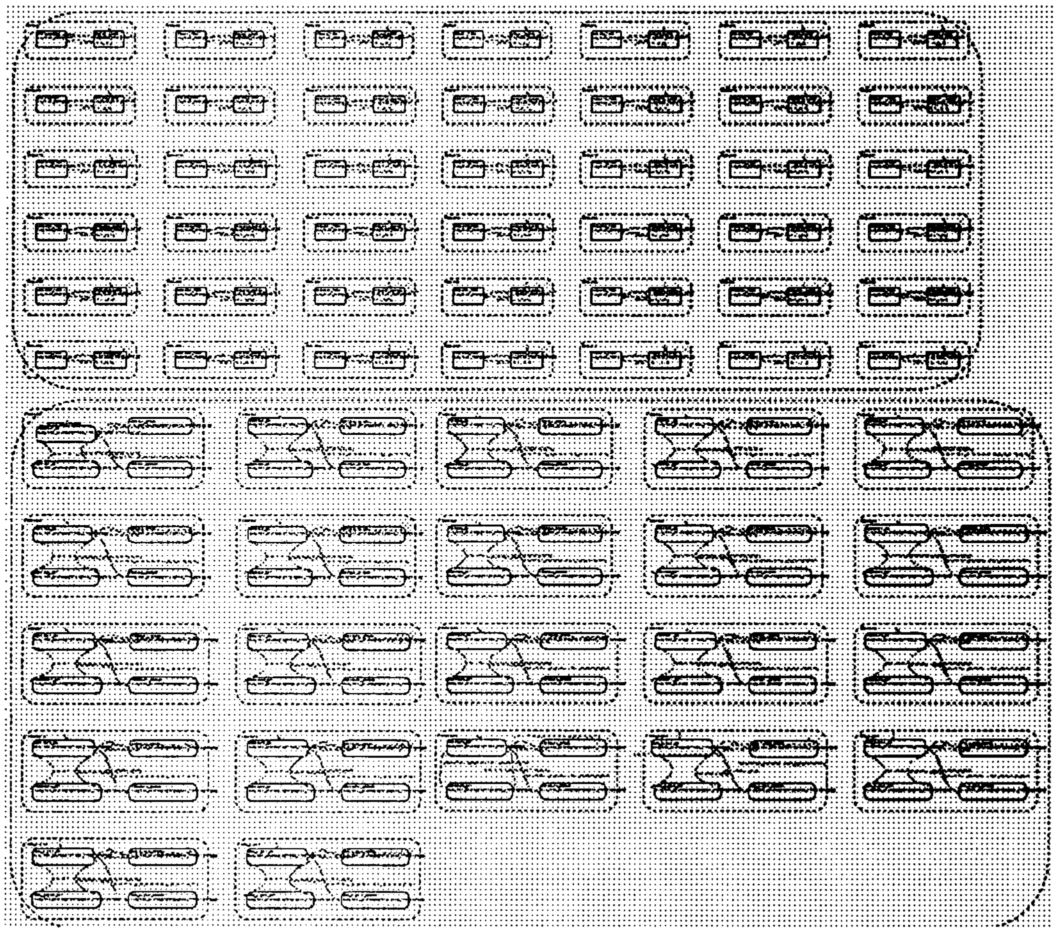


Figure 9: Vehicle Event Manager Stateflow Chart.

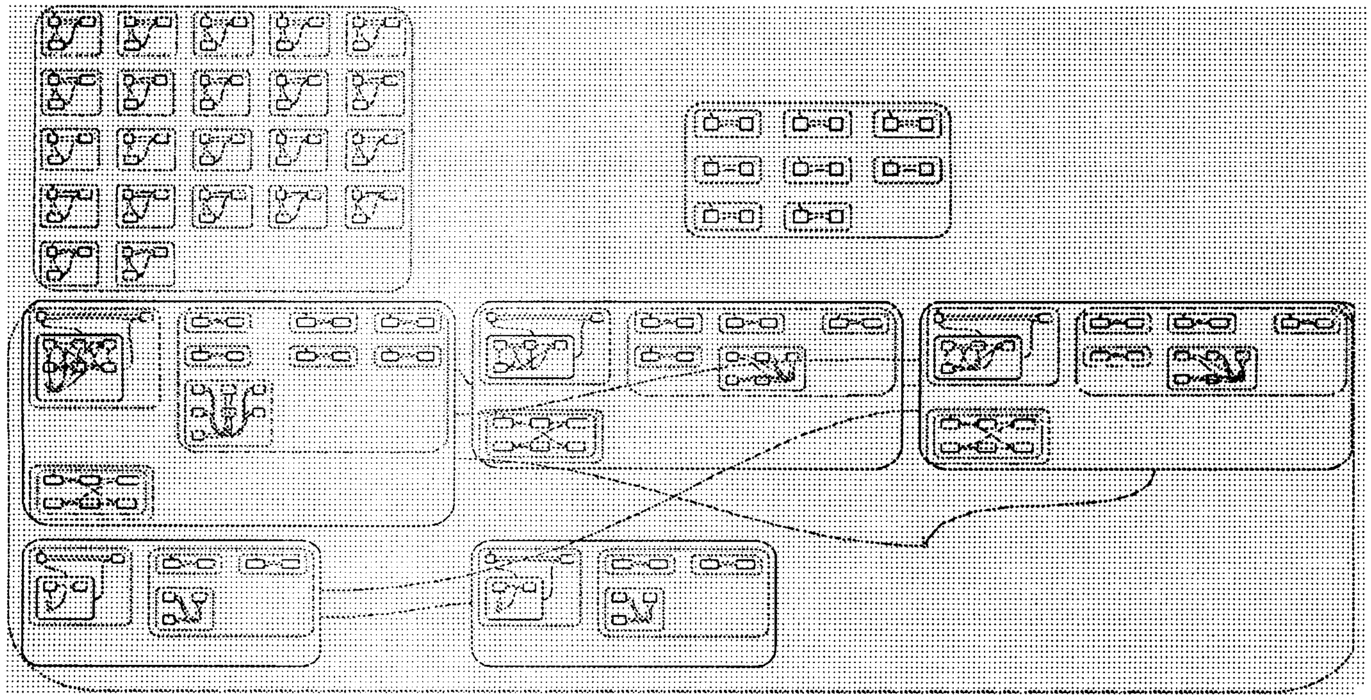


Figure 10: Autocommander Stateflow Diagram.

Integrated Vehicle Health Monitor

The IVHM subsystem block currently takes the flags/signals from the Vehicle Event Manager subsystem block and individually adds a variable delay to each signal. This delay time is to represent time required by a real vehicle's IVHM to process vehicle data and to diagnose any vehicle problems. Two different methods are used to achieve the variable delay. The first method uses a Stateflow chart to add delay for the 42 ON/OFF failure events. To accomplish the delay in the binary signal, the simulation time at which the signal from the Vehicle Event Manager becomes 1 is stored. The amount of delay is then added to this time and when the clock simulation time reaches this new summed value the Stateflow chart sets the corresponding output to 1. When the input signal from the Vehicle Event Manager goes back to 0, a similar approach is taken to add the delay. The second method to create the signal delay is to use a variable time transport delay block, which was used for the remaining 22 effector failure signals. The two inputs to this block are the delay time and the signal. Each signal requires a separate transport delay to have independent delays.

Autocommander

The Autocommander subsystem block is divided into four parts: input organizer, output organizer, Stateflow chart, and an algorithm section.

Even with a preliminary implementation of our integrated G&C algorithms in IDOS, it is clear that data organization is required for efficiently handling I/O communication. A hierarchal data structure was created for the input and output organizer with the use of bus creator and bus selector Simulink Blocks. As an example of how the data is organized, for a signal named "signal" that is fed into a bus, "bus1", and in turn the bussed signal is fed into another bus, "bus2", the data structure for the original signal would be "bus2.bus1.signal".

A subsystem block in the Autocommander system contains the Autocommander Stateflow chart, the bus selection block to specify the inputs to the chart and a bus creation block to multiplex the chart's output. Currently, the chart is divided into three sections (shown in Figure 10): Control Allocation (CA - top left), Time Varying Bandwidth (TVB - top right), and Flight Manager (FM - bottom). The CA section processes IVHM data and sends a Reduced Control Authority (RCA) flag to the G&C control allocation. The value of the RCA flag has a value between 0 and 1 that is a scale of the percentage of remaining control authority with "0" being not operational. The TVB section sends a flag to the attitude control system. A TVB flag of 1 means that the commanded effector deflection is greater than the effector deflection limit; otherwise the TVB flag is 0. This flag causes the controller to adapt to effector saturation. The Flight Manager section sequences the flight phases and contains the baseline Nominal, Reconfigure and Abort structure of the Autocommander. In addition, ascent and entry flight phases each contain an algorithm sequencer that dictates the order and time of algorithm execution within the Autocommander's Algorithm section shown in Figure 11. Currently, placeholder algorithms are used to send a completion flag to the Stateflow chart after 1 second of run time for each algorithm so that the algorithms are serially sequenced.

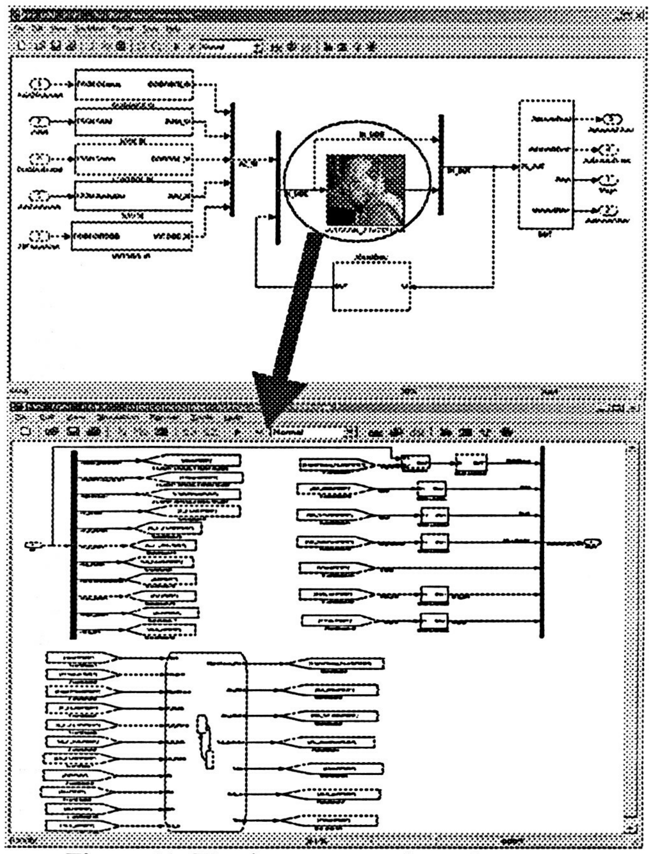


Figure 11: The Autocommander subsystems.

SIMULATION RESULTS

The following section presents a comparison of two ascent simulation runs with the same multiple surface effector failures. The simulation run without Time Varying Bandwidth (TVB) results are shown in Figures 12 thru 23, and the second run with TVB results are shown in Figures 24 thru 35. In both of these simulation runs, all eight surface effectors fail by 50% near the time of lift off. In all the command verses actual plots, the blue solid line is the commanded response and the green dashed line is the actual (sensed) response.

In theory, the TVB addition to the controller should provide greater stability during the mach regions and severe disturbances of flight than without TVB. Figure 36 and Figure 37 show the scaling back of the controller's inner and outer loop bandwidths for the given example. Clearly, without the TVB the vehicle crashes as indicated by the altitude in Figure 12 and by the system instability that are shown in Figures 13 thru 15. Comparing the same plots for the simulation run with TVB, Figure 24 shows the vehicle making it to orbit and Figure 25 thru 27 shows the system stability, but with a poor tracking performance in the 100 to 200 second range. The TVB sacrifices tracking performance for stability as shown in this example.

Figures 16 thru 23 and Figures 28 thru 35 show the commanded verses actual surface deflection angles for their corresponding simulation run. Both of these sets of figures show the 50% failure of all surface effectors from the beginning of the simulation, which verifies the actions of the Vehicle Event Manager.

NO TVB

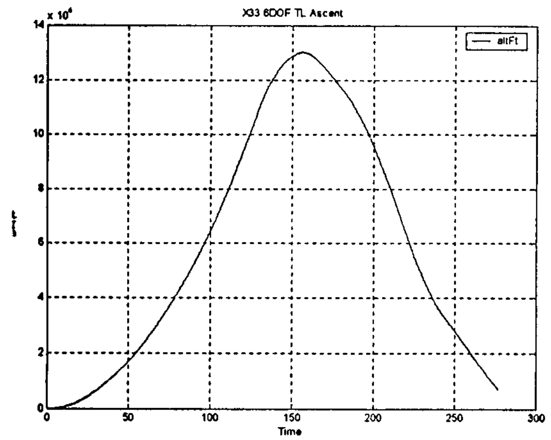


Figure 12: Altitude (ft.)

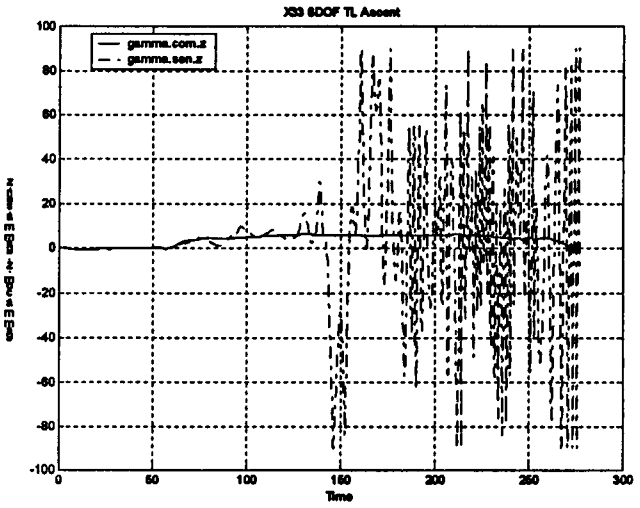


Figure 13 : Commanded vs. actual yaw (deg.)

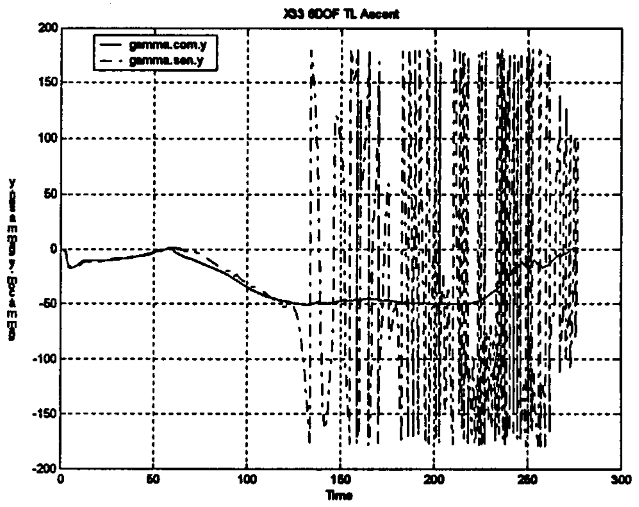


Figure 14: Commanded vs. actual pitch (deg.)

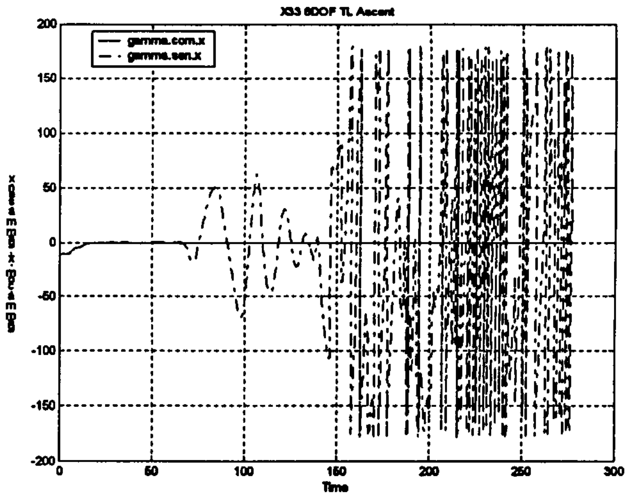


Figure 15: Commanded vs. actual roll (deg.)

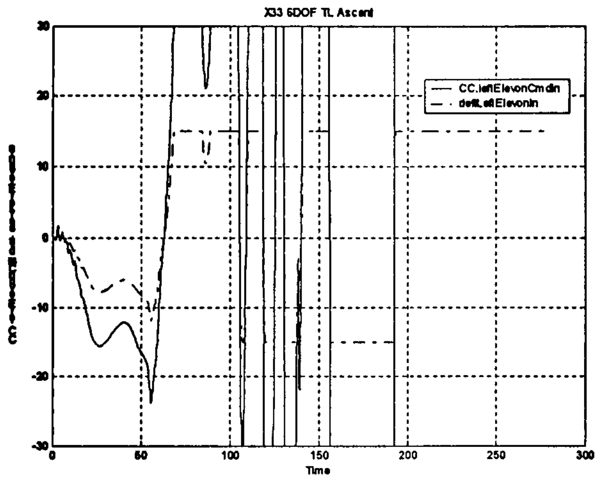


Figure 16: Commanded vs. actual left inboard elevon deflections (deg.)

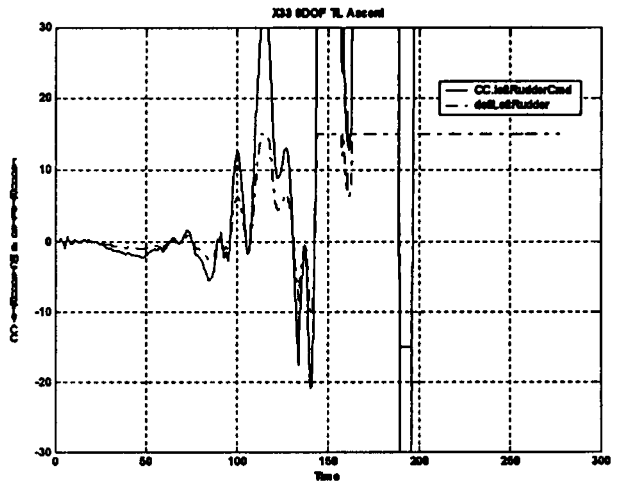


Figure 19: Commanded vs. actual left rudder deflections (deg.)

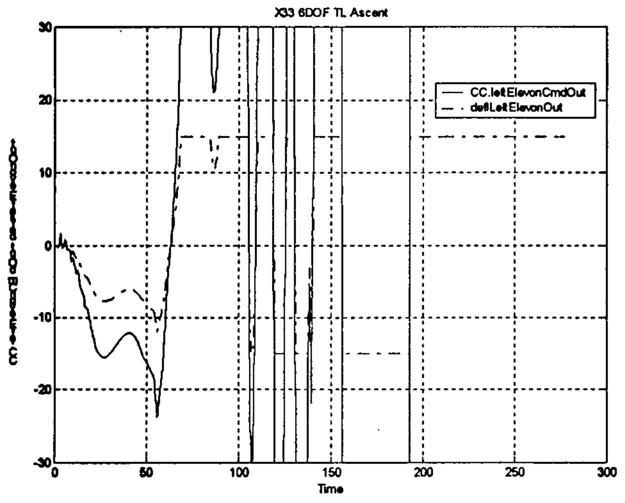


Figure 17: Commanded vs. actual left outboard elevon deflections (deg.)

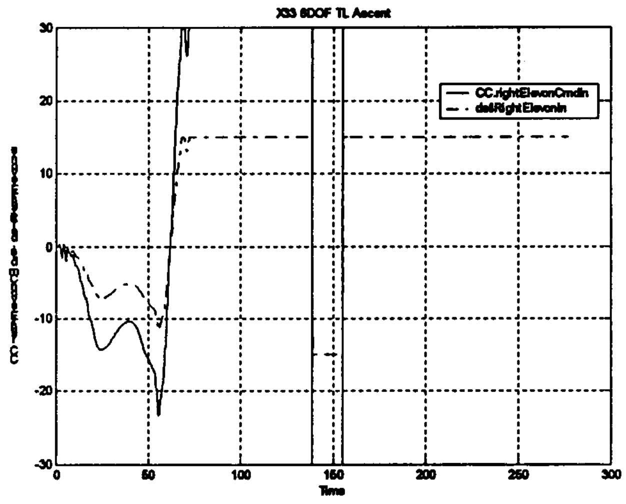


Figure 20: Commanded vs. actual right inboard elevon deflections (deg.)

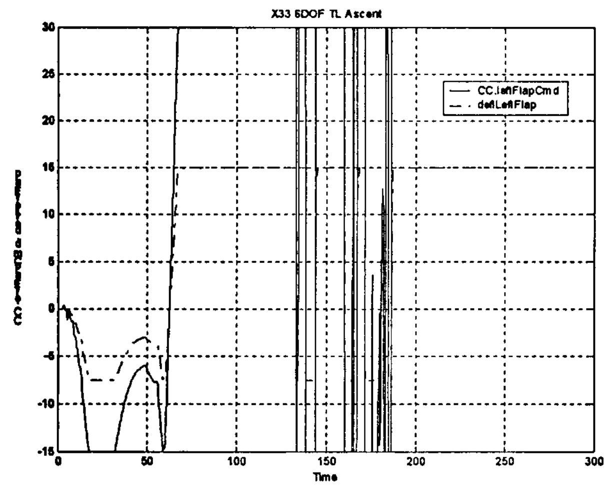


Figure 18: Commanded vs. actual left flap deflections (deg.)

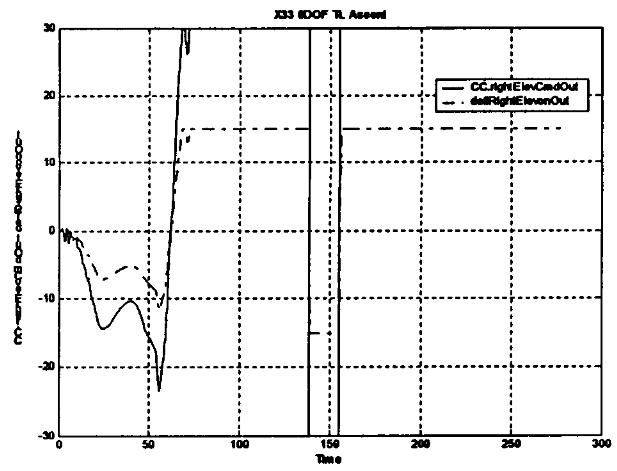


Figure 21: Commanded vs. actual right outboard elevon deflections (deg.)

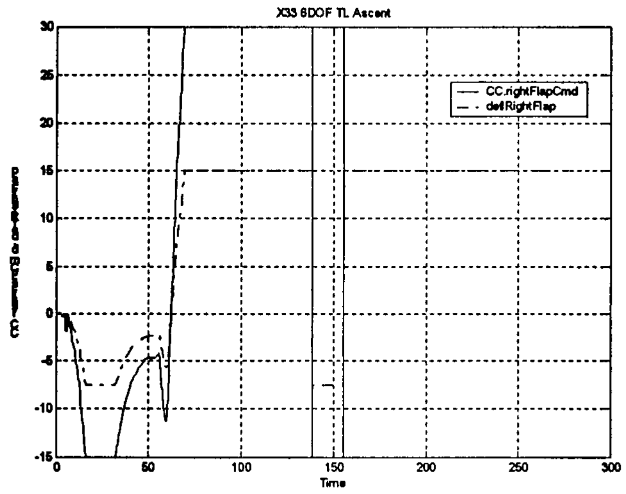


Figure 22: Commanded vs. actual right flap deflections (deg.)

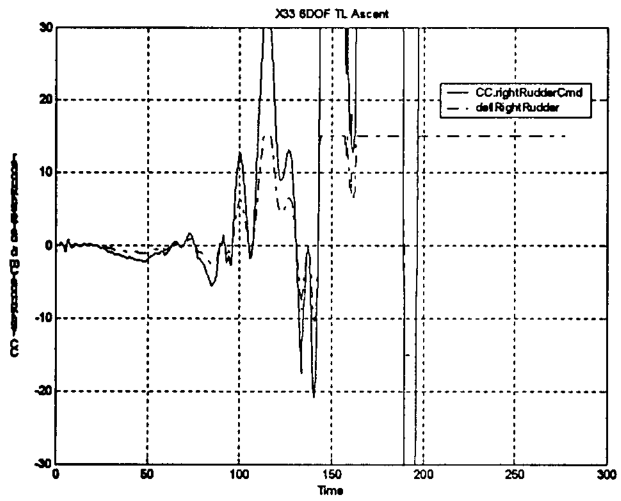


Figure 23: Commanded vs. actual right rudder deflections (deg.)

TVB

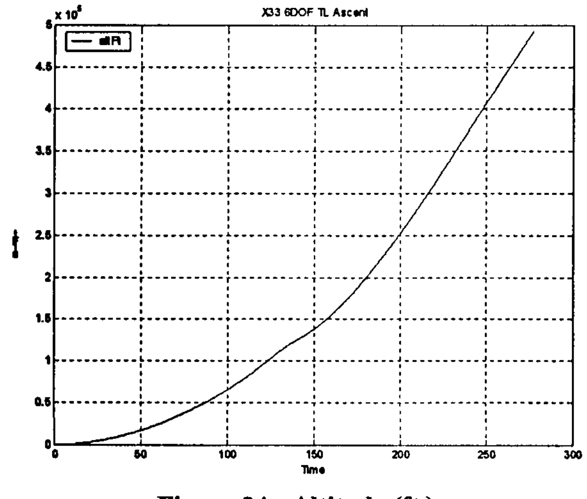


Figure 24: Altitude (ft.)

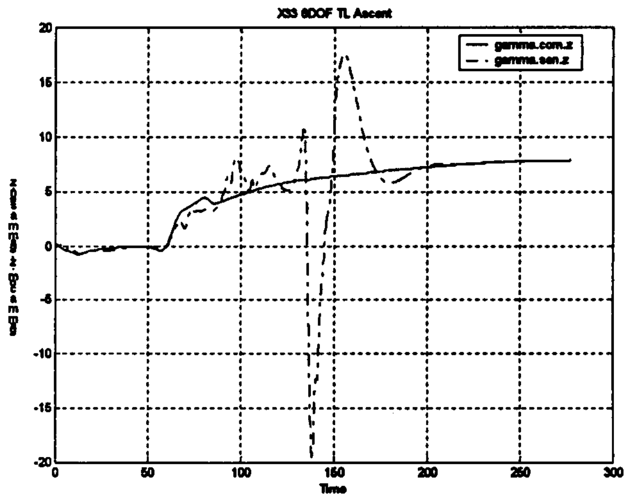


Figure 25: Commanded vs. actual yaw

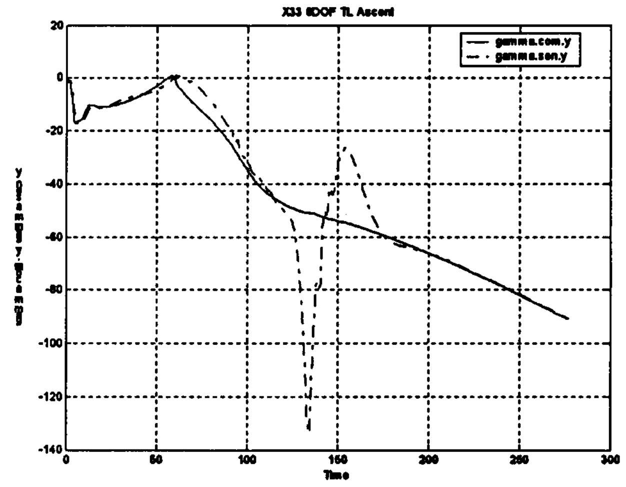


Figure 26: Commanded vs. actual pitch (deg.)

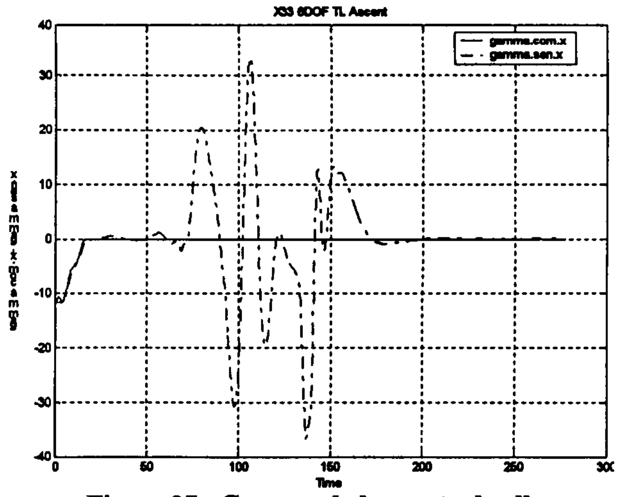


Figure 27: Commanded vs. actual roll

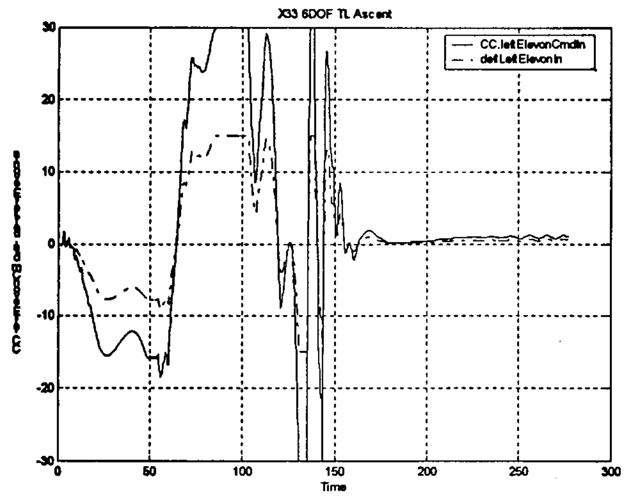


Figure 28: Commanded vs. actual left inboard elevon deflections (deg.)

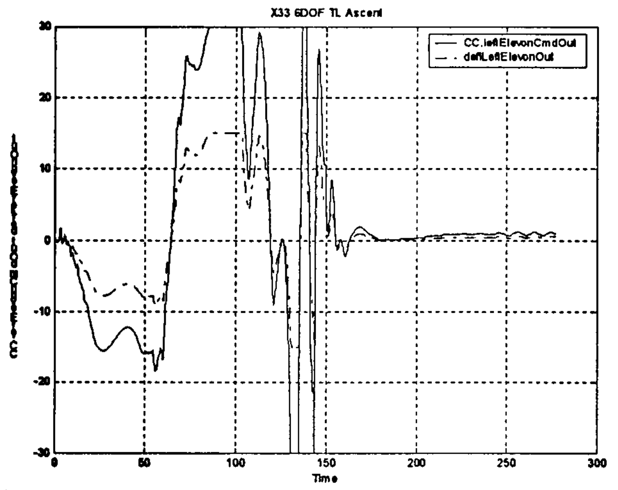


Figure 29: Commanded vs. actual left outboard elevon deflections (deg.)

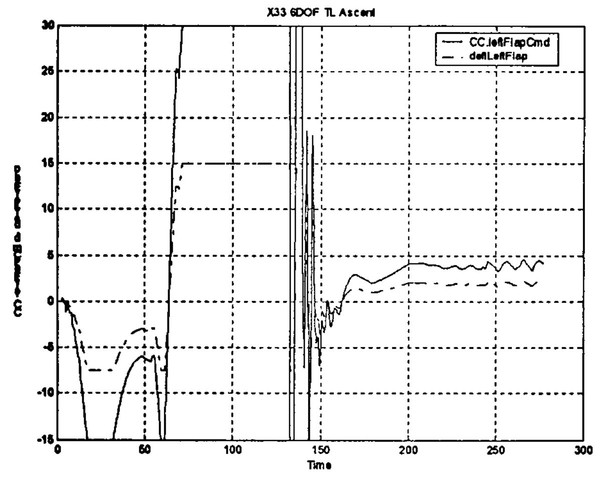


Figure 30: Commanded vs. actual left flap deflections (deg.)

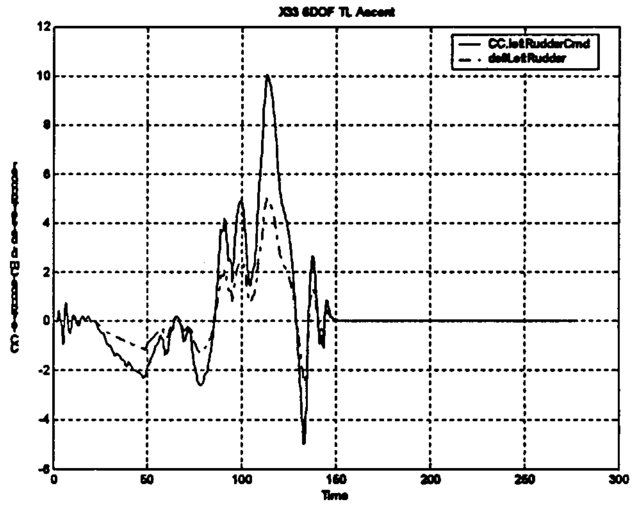


Figure 31: Commanded vs. actual left rudder deflections (deg.)

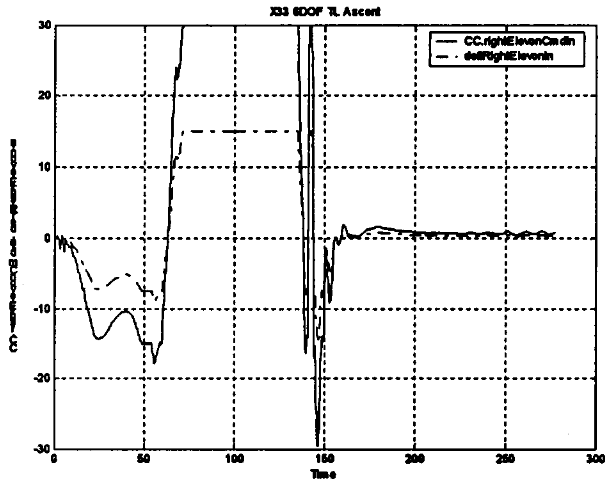


Figure 32: Commanded vs. actual right inboard elevon deflections (deg.)

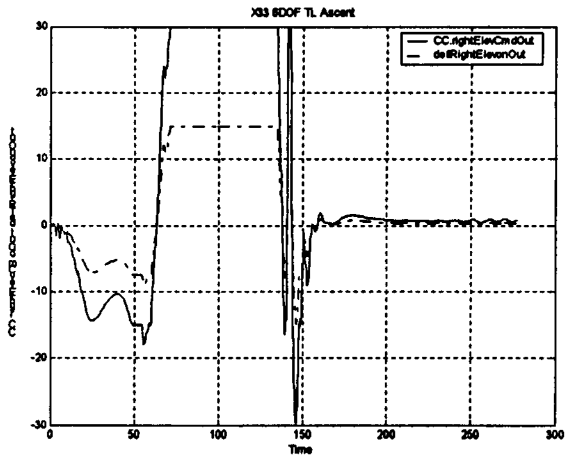


Figure 33 : Commanded vs. actual right inboard elevon deflections (deg.)

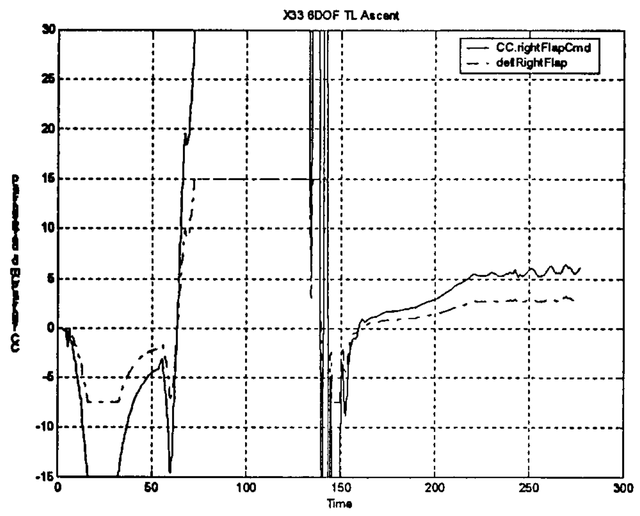


Figure 34: Commanded vs. actual right flap deflections (deg.)

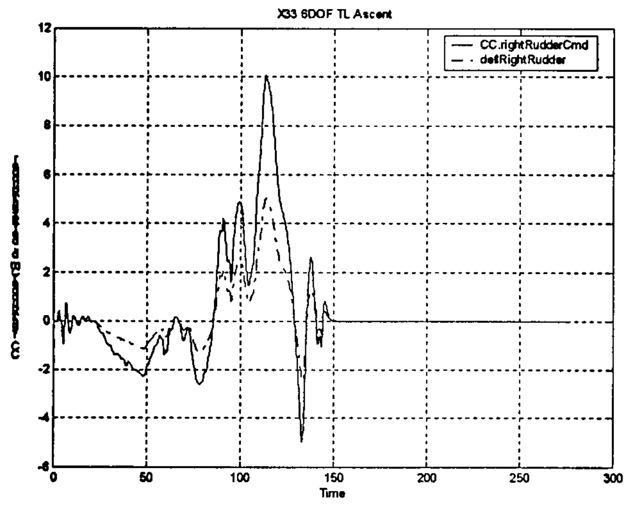


Figure 35: Commanded vs. actual right rudder deflections (deg.)

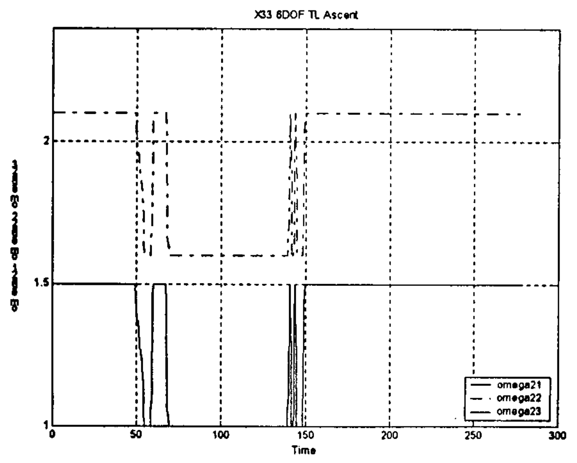


Figure 36: Inner loop bandwidth for the simulation run with TVB.

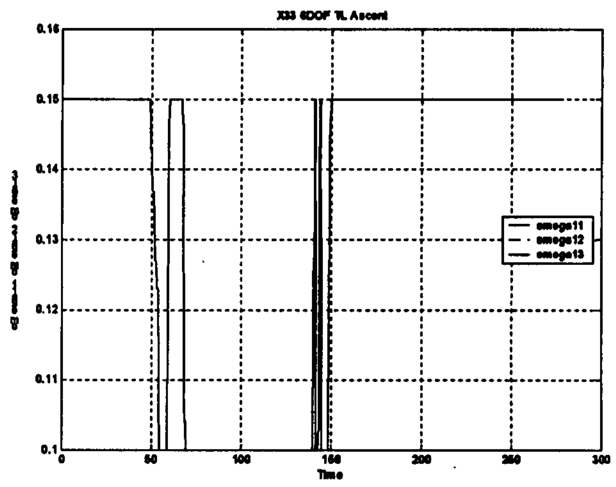


Figure 37: Outer loop bandwidth for the simulation run with TVB.

CONCLUDING REMARKS

The implementation of IG&C algorithms into a high fidelity Simulink vehicle simulation, such as IDOS's flight control software (FCSW), is a far less cumbersome process than implementing algorithms within traditional hard coded simulations. The modularity of the IG&C portion of the simulation model allows users to easily interchange algorithms with little additional effort. Signal traceability and analysis cabilities of Simulink provide a user a time saving and a confidence building way of verifying and testing new G&C algorithms.

ACKNOWLEDGEMENT

This work is supported by NASA Marshall Space Flight Center under the 2nd Generation Reusable Launch Vehicle program. A thank you goes out to Universal Space Lines for providing IDOS and support in our IG&C implementation.

REFERENCES

[1] C. G. Casssandras, S. Lafortune, *Introduction to Discrete Event Systems*, Kluwer Academic Publishers, Boston, MA, 1999.

[2] J. E. Fisher, D. A. Lawrence and J. J. Zhu, *Stateflow Autocoder*, Proceedings of the 34th Southeastern Symposium on System Theory, Huntsville, Alabama, March 18-19, 2002, IEEE 0-7803-7339-1, pp. 467-470.

[3] J. E. Fisher, D. A. Lawrence and J. J. Zhu, *Autocommander – A Supervisory Controller for Integrated Guidance and Control for the 2nd Generation Reusable Launch Vehicle*, AIAA Guidance, Navigation and Control Conference & Exhibit, AIAA-2000-4562 Monterey, California, August 5-8, 2002.

[4] Khalil, Hassan, *Nonliner Systems*, Prentice Hall, Upper Saddle River, NJ, 1996.

[5] J. Jim Zhu, Brad D. Banker, *X-33 Ascent Flight Controller Design by Trajectory Linearization – A singular perturbation Approach*, AIAA Guidance, Navigation and Control Conference & Exhibit, AIAA-2000-4159,Denver, Colorado, August 14-17, 2000.

[6] J. Jim Zhu, A. Scottedward Hodel, Kerry Funston and Charles E. Hall, *X-33 Entry Flight Controller Design by Trajectory Linearization – A singular perturbation Approach*, 24th Annual AAS Guidance and Control Conference, AAS-01-012, Breckenridge, Colorado, January 31 thru February 4, 2000.

[7] *Stateflow for State Diagram Modeling*: User's Guide Version 4, The Math Works, 2001.