NASA/TM—2016-219089 AIAA–2016–1425

Benchmarking Model Variants in Development of a Hardware-in-the Loop Simulation System

Eliot D. Aretskin-Hariton
Glenn Research Center, Cleveland, Ohio

Alicia M. Zinnecker
N&R Engineering, Cleveland, Ohio

Jonathan L. Kratz and Dennis E. Culley
Glenn Research Center, Cleveland, Ohio

George L. Thomas
N&R Engineering, Cleveland, Ohio

April 2016

NASA/TM—2016-219089 AIAA–2016–1425

Benchmarking Model Variants in Development of a Hardware-in-the Loop Simulation System

Eliot D. Aretskin-Hariton
Glenn Research Center, Cleveland, Ohio

Alicia M. Zinnecker
N&R Engineering, Cleveland, Ohio

Jonathan L. Kratz and Dennis E. Culley
Glenn Research Center, Cleveland, Ohio

George L. Thomas
N&R Engineering, Cleveland, Ohio

Prepared for the
Scitech 2016
sponsored by the American Instititue of Aeronautics and Astronautics
San Diego, California, January 4–8, 2016

National Aeronautics and
Space Administration

Glenn Research Center
Cleveland, Ohio 44135

April 2016

Acknowledgments

This work was sponsored by the Transformative Aeronautics Concepts Program (TACP) at the NASA Glenn Research Center.

Benchmarking Model Variants in Development of a Hardware-in-the Loop Simulation System

Eliot D. Aretskin-Hariton
National Aeronautics and Space Administration
Glenn Research Center
Cleveland, Ohio 44135

Alicia M. Zinnecker
N&R Engineering
Cleveland, Ohio 44130

Jonathan L. Kratz and Dennis E. Culley
National Aeronautics and Space Administration
Glenn Research Center
Cleveland, Ohio 44135

George L. Thomas
N&R Engineering
Cleveland, Ohio 44130

Distributed engine control architecture presents a significant increase in complexity over traditional implementations when viewed from the perspective of system simulation and hardware design and test. Even if the overall function of the control scheme remains the same, the hardware implementation can have a significant effect on the overall system performance due to differences in the creation and flow of data between control elements. A Hardware-in-the-Loop (HIL) simulation system is under development at NASA Glenn Research Center that enables the exploration of these hardware dependent issues. The system is based on, but not limited to, the Commercial Modular Aero-Propulsion System Simulation 40k (C-MAPSS40k). This paper describes the step-by-step conversion from the self-contained baseline model to the hardware in the loop model, and the validation of each step. As the control model hardware fidelity was improved during HIL system development, benchmarking simulations were performed to verify that engine system performance characteristics remained the same. The results demonstrate the goal of the effort; the new HIL configurations have similar functionality and performance compared to the baseline C-MAPSS40k system.

Nomenclature

ADC	Analog-to-Digital Converter
C-MAPSS40k	Commercial Modular Aero-Propulsion System Simulation 40k
CSP	Control System Platform
DAC	Digital-to-Analog Converter
ECU	Engine Control Unit
EPM	Engine Plant Model

EPR	Engine Pressure Ratio
FMV	Fuel Metering Valve
HIL	Hardware-in-the-Loop
HPC	High-Pressure Compressor
LPC	Low-Pressure Compressor
MAE	Mean Absolute Error
NCAP	Network-Capable Application Processor
ODE	Ordinary Differential Equation
PLA	Power Lever Angle
SM	Surge Margin
STIM	Smart Transducer Interface Module
SXD	Smart Transducer
TEDS	Transducer Electronic Datasheet
UDP	User Datagram Protocol
UI	User Interface
VBV	Variable Bleed Valve
VSV	Variable Stator Vane

I. Introduction

Distributed architectures for control systems are widely-used in the automotive industry, but adoption by the aerospace industry has been slow due to the perception that the risks associated with such implementations outweigh the potential benefits.[1, 2] Traditional engine control systems have a centralized architecture, wherein the control algorithm and data conversion and processing functionality (such as averaging sensor measurements) reside in the Engine Control Unit (ECU). A distributed architecture is characterized by off-loading of the conversions between the analog and digital domains to processors local to the actuator and sensor hardware. This requires a digital control network and electronics capable of operating in the harsh engine environment. This need for high-temperature electronics represents a primary barrier in the progress toward a distributed engine control system.

If the technological capability existed to enable off-loading the data processing functionality, the physical appearance of the entire control system could be changed. In particular, the overall size and weight of the system could be reduced because the multitude of wires connecting sensors and actuators to the controller could be replaced by a smaller, lighter network bus. The presence of this network creates inherent modularity in the system, wherein hardware nodes can be added, removed, or replaced without the need for costly control system redesign, as opposed to typical centralized control architectures, which require extensive modification when design changes are made. For example, in a centralized architecture, replacing a sensor may require modifying multiple wiring harnesses, as well as circuit board modifications and software reprogramming inside the ECU. In a distributed control system, any physical modifications would be isolated to the module level. This will encourage the development of a 'partial' certification process that does not require a control system, once certified, to undergo full re-certification after a hardware change. Instead, only the new module requires certification, since its interface with the system did not change. Modularity will also provide benefits in fault isolation and maintenance logistics throughout the engine life cycle.[1, 2]

One more-encompassing barrier in the development and implementation of distributed control for an aerospace application is related to the design process as a whole: the design, testing, and implementation of a controller are often the final steps in the development of an engine system, and often subject to strict time and budgetary constraints.[3] This limits the risks designers are willing to incur in developing new technologies and testing new ideas for the control system. The Hardware-in-the-Loop (HIL) system at the NASA Glenn Research Center is being used to develop different engine control architectures to decrease the risk inherent in developing distributed control systems. The system uses the Commercial Modular Aero-Propulsion System Simulation 40k (C-MAPSS40k) as a baseline engine simulation. Modifications of the system, with the eventual goal of adding network modeling capability, are explored through a series of benchmarking tests. This paper describes the step by step conversion from the self contained baseline model to the hardware in the loop model, and the validation of each step.

The paper is organized as follows: details about the HIL simulation system, including the physical implementation and the models under consideration, are provided in Section II. The results of the benchmarking simulations performed to date are presented in Section III, and discussion of these results is given in Section IV.

II. HIL simulation system details

An effort toward building a real-time Hardware-in-the-Loop (HIL) system is ongoing at NASA Glenn Research Center.[4,5] This system provides a framework for testing various controller architectures and was developed around C-MAPSS40k, a closed-loop zero-dimensional dynamic simulation of a twin-spool, high-bypass turbofan engine capable of producing up to $40,000$ lb$_f$ of thrust.[6,7] The controller in this simulation tracks an Engine Pressure Ratio (EPR) demand, mapped from the Power Lever Angle (PLA) input profile, and includes limiters to protect against surge in the two compressors, overspeed on the two spools, and combustor blowout.[8] Inputs to the model include the flight profile, defined by individual profiles for each environmental condition and PLA, along with profiles for the engine health parameters and controller parameters. Internal engine variables, such as pressures and temperatures, and controller-related variables, such as tracking references and actuator demands, in addition to flags indicating which limiters are active at each step in the simulation, are available as outputs from the simulation. These inputs and outputs are consistent across each of the system configurations considered in this benchmarking effort; however, the models themselves may be modified to produce additional outputs, as required.

In the general framework (Fig. 1), the Engine Plant Model (EPM) and Control System Platform (CSP) perform the calculations required of the closed-loop system, while the User Interface (UI) acts as a wrapper for the simulation by providing input profiles to, and collecting output data from, both the EPM and CSP. Smart sensors and actuators are then introduced to the controller, increasing the fidelity of the model. These Smart Transducer (SXD) models are considered 'smart' because they have been modeled around the IEEE 1451 specifications and because the signal conditioning, conversion, and processing functionality resides local to the hardware elements.[9–15] The SXD model library was initially implemented in the same MATLAB/Simulink® (The Mathworks, Inc.) environment as C-MAPSS40k.[16] The SXD library was then expanded to be used for creating SXD models on micro controller boards integrated with the HIL system, representing a distributed controller architecture.

In order to make the baseline C-MAPSS40k more representative of a networked HIL system, benchmarking tests were performed. These tests helped to verify that the restructuring of C-MAPSS40k did not impact overall system functionality. Five configurations of the CSP were considered: four implementations exclusively in the MATLAB/Simulink environment, and one case, demonstrating distributed hardware, where external microcontrollers were integrated with the system. Given a test flight profile, five simulations of each configuration were performed to validate system performance. The increasing complexity of the control models does result in an increase in the run time and this is noted for comparison.

A. Physical implementation

The baseline C-MAPSS40k is designed to be simulated on a single computer and therefore is implemented in a single Simulink model file. For the HIL simulation system, the model was modified so that each block indicated in Fig. 1 resides in its own file. This allows the system to take advantage of semi-parallel simulations by running each model on a separate computer.[4] The three components (EPM, CSP, and UI) communicate via the exchange of six data arrays at each time-step; all communication is completed by a model before beginning its calculations. Because of its low overhead and fast transfer speeds, the User Datagram Protocol (UDP) was chosen as the means for this communication; more detail on the setup of the HIL simulation system can be found in Refs. 4, 5.

Further decomposition of the CSP is performed through integration of microcontroller boards with the HIL simulation system. Each board contains an Ethernet port that allows it to communicate directly with the CSP Simulink model. This is illustrated in Fig. 2 for a single board; other boards are connected in a similar fashion. For sensor models, data from the EPM are routed by the CSP to the appropriate board, where they are processed and routed back; for actuator models, the controller command from the CSP is routed appropriately and the data returned are packaged to be sent to the EPM. When a controller network is integrated, the communication scheme will be altered slightly. This is due to the fact that a master node will be responsible for routing the sensor and controller output data appropriately. However, input data to the sensors and output data from the actuators will still be transferred to the EPM over UDP, as this represents the physical interface between the hardware and engine.

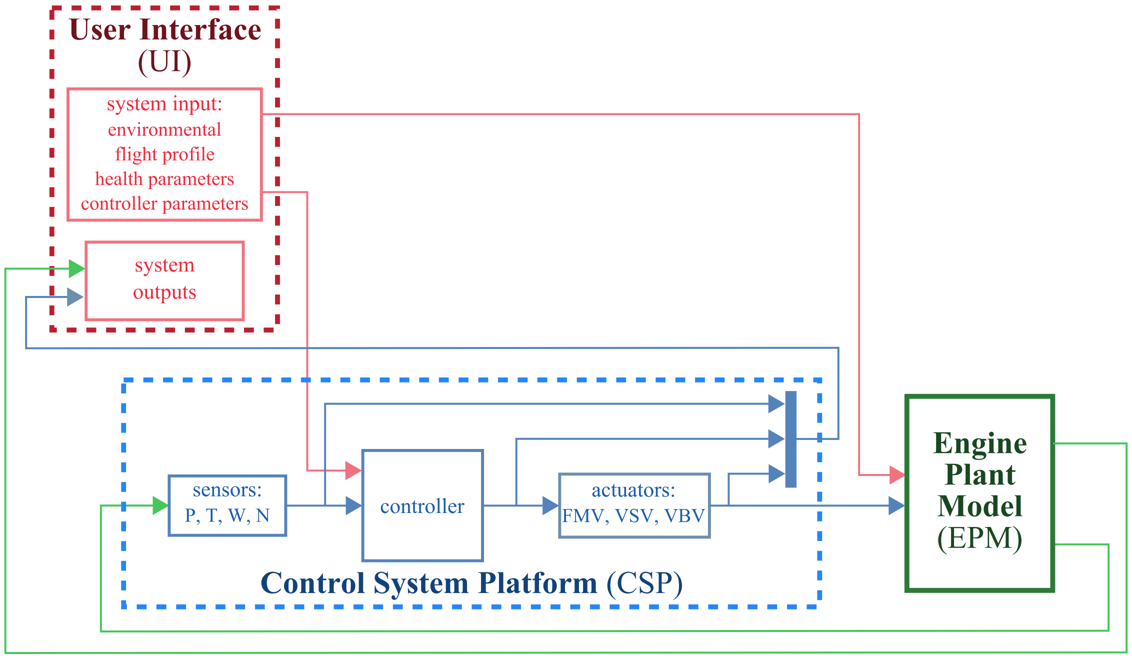

Figure 1. Block diagram illustrating the HIL simulation system framework. The sensed values (Pressure, P, Temperature, T, Flow Rates, W, and Shaft Speeds, N) and actuators (Fuel Metering Valve (FMV), Variable Stator Vane (VSV), and Variable Bleed Valve (VBV)) are specific to the C-MAPSS40k engine and controller models.

B. Smart transducer models

C-MAPSS40k was designed with a centralized control system. By default it contains simple linear models for the sensors in the system, and similar models of the actuators, which are cascaded with nonlinearities such as saturation and, in some cases, local loop closure. Additional effects, such as quantization related to the conversion from the analog to digital domains, are not yet captured by these models. Furthermore, the controller network present in a distributed architecture, which introduces delays and possible packet loss to the system, was not part of the design criteria for the baseline C-MAPSS40k simulation. To improve hardware simulation capability of the C-MAPSS40k system, a framework based around the IEEE 1451

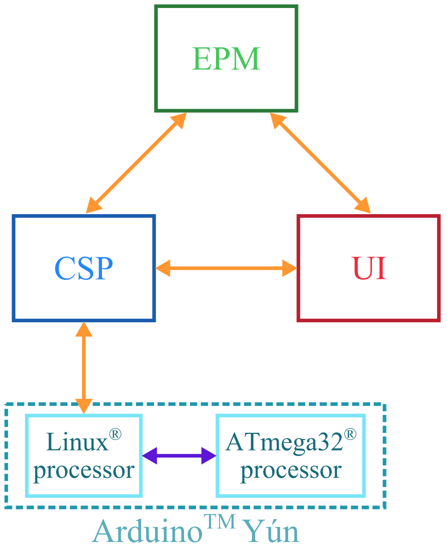

Figure 2. Block diagram showing how a single microcontroller integrates with the CSP in the HIL simulation system. Orange arrows indicate User Datagram Protocol (UDP) communication links, while the purple arrow represents serial communication between the two processors on the microcontroller, a feature that will be used in future research.

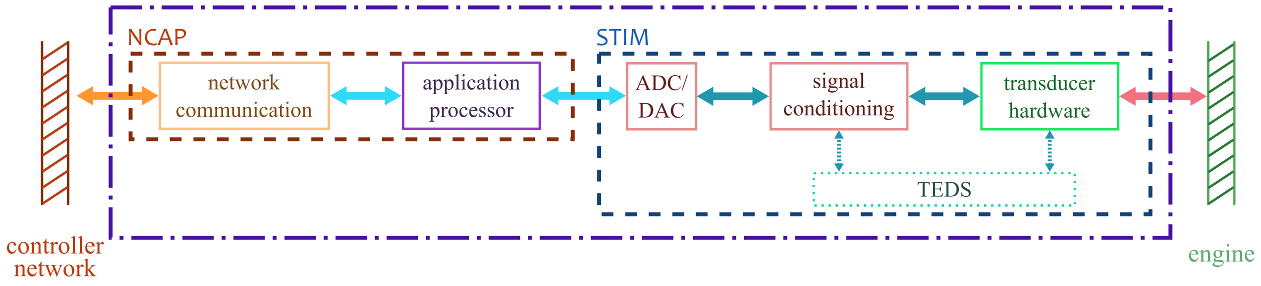

Figure 3. Block diagram of a SXD complying with the IEEE 1451 specifications.

specifications for a SXD has been implemented.[16]

An IEEE 1451-compliant SXD may be represented by the block diagram in Fig. 3, where the two main components are the Smart Transducer Interface Module (STIM) and Network-Capable Application Processor (NCAP), which interface with the engine and controller network, respectively. The STIM contains the sensor or actuator hardware, along with any signal conditioning circuitry, and an Analog-to-Digital (ADC) or Digital-to-Analog (DAC) converter for interfacing with the application processor residing on the NCAP. The Transducer Electronic Data Sheet (TEDS) is also a part of the STIM. The TEDS contains transducer manufacturing and calibration information in a digital format that can be accessed by the application processor. The block labeled 'network communication' in Fig. 3 indicates the adapter, and any drivers, required for the SXD to interface with the controller network.

For implementation of such a model in Simulink, a block library was created that contains sublibraries of components based on the blocks in Fig. 3.[16] In addition to blocks implementing signal conditioning filters, level change compensators, and signal conversions, the library includes generic sensor and actuator models, an electro-mechanical servo motor model, and various processing functions. A generic SXD model for the ten sensors in C-MAPSS40k was constructed by connecting blocks for the hardware, signal conditioning, ADC, and processing (averaging) components. Similar models were constructed for the three actuators: a generic open-loop model for the Variable Bleed Valve (VBV), a generic closed-loop model for the Fuel Metering Valve (FMV), and a closed-loop servo motor model for the Variable Stator Vanes (VSV). In addition to these more complex hardware models, a simple model of a network, capturing transmission delay and the possibility of lost packets, was introduced.

In order to implement models of these SXDs on the microcontroller,* the Simulink library was expanded to the Python programming language. Each block was rewritten as a Python module that could be combined with other modules to create the SXD models in a similar fashion to their Simulink counterparts; functionality of these models was verified against those constructed in Simulink. Models of the sensors and actuators were implemented on the Linux® processor in anticipation of using the ATmega32® processor (Fig. 2) for implementation of a dedicated control network in future work.

C. Controller models

Five versions of the C-MAPSS40k controller model are considered for benchmarking. Four models are implemented entirely in the MATLAB/Simulink environment: the 'baseline' controller model, the 'unstructured' controller model, the 'distributed' controller model, and the 'networked' controller model. The fifth configuration, the 'processor-in-the-loop,' includes microcontrollers integrated with the system. The baseline model is an unmodified copy of C-MAPSS40k, which runs on a single computer, while the other four controllers are implemented in the architecture represented in Fig. 1, with the slight modification that the UI and CSP were executed in two different instances of Simulink on the same computer. The unstructured configuration is the same as the baseline configuration, except that it is implemented in this three-part architecture, while the distributed configuration additionally contains models of SXDs, as described in Section II.B. The networked configuration is similar to that of the distributed, and also contains a statistical network model that has a small probability to drop or delay data packets. In the processor-in-the-loop configuration, the SXD models were implemented on individual microcontrollers, as illustrated in Fig. 2. These communicate directly with the CSP via Ethernet. Each subsequent model made the system more complex, as seen through an increase in run time, a necessary trade-off when introducing fidelity to a model. How-

*The microcontroller, which is an Arduino Yún, contains two processors: an Atmel ATmega32® processor that can be programmed in C, and a Linux® processor that can run Python scripts.

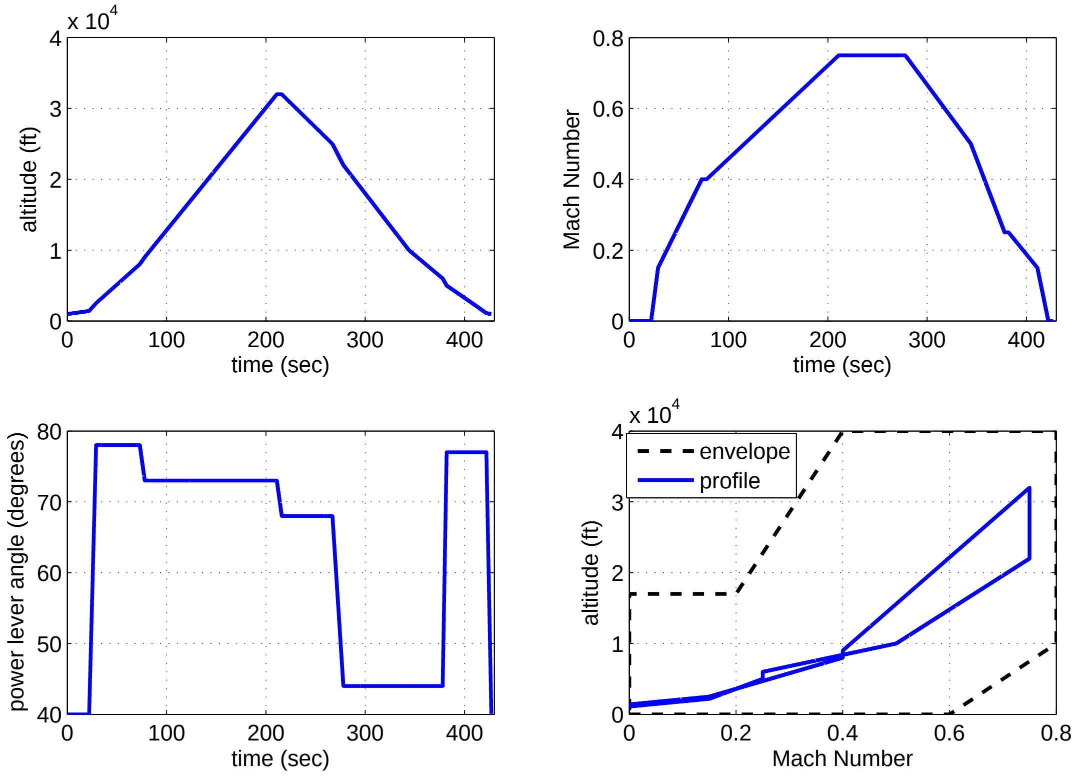

Figure 4. Altitude, Mach number, and PLA profiles for the test simulations used to benchmark the different CSP configurations. The plot in the lower right illustrates how the flight profile lies inside the C-MAPSS40k flight envelope.

ever, despite significant increases in run time when the SXD models were introduced (in the distributed and networked models) and again when the microcontrollers were integrated (the processor-in-the-loop model), simulations still completed faster than real-time.

III. Benchmarking results

The HIL simulation system, with each of the five controller models defined in Section II.C, was benchmarked through simulation with the test flight profile shown in Fig. 4. This profile was derived from the example profile that captures a complete flight from takeoff to landing, and allows for testing the models at conditions throughout the flight envelope.[17] This profile defines a flight compressed to 427 seconds.

Simulations were run using the Ordinary Differential Equation (ODE) four solver in Simulink with a fixed step size: for the baseline and unstructured models, a step size of 0.015 seconds was used, while for the other models this step size was decreased to 0.005 seconds. A reduced step size was required for the distributed and networked configurations to enable successful simulation with the SXD models. In particular, the time constant of the servo motor in the VSV model was such that numerical errors were encountered when larger step sizes were used. Additionally, the smaller step size provided the flexibility to assign different sampling times to components of the controller; for example, sensors provided data to the controller at each time step (a sampling time of 0.005 seconds) while the controller provided data to the actuators at 0.015 second intervals.

It should be noted that to achieve repeatable results, the Variable Bleed Valve (VBV) kicker block, which is part of the VBV control logic, was disabled. The VBV command is scheduled on corrected fan speed and Mach number, and is offset open when the core shaft speed and fuel flow rate drop significantly from one time step to the next. The VBV kicker triggers this offset, which is used during off-nominal VBV operation. The conditions under which this offset became active were adjusted to try to mitigate the effects of non-repeatable core shaft speed or fuel flow behavior during testing. It was not possible to fully account for these differences, so the offset became active during some of the test cases, raising the average VBV input at those

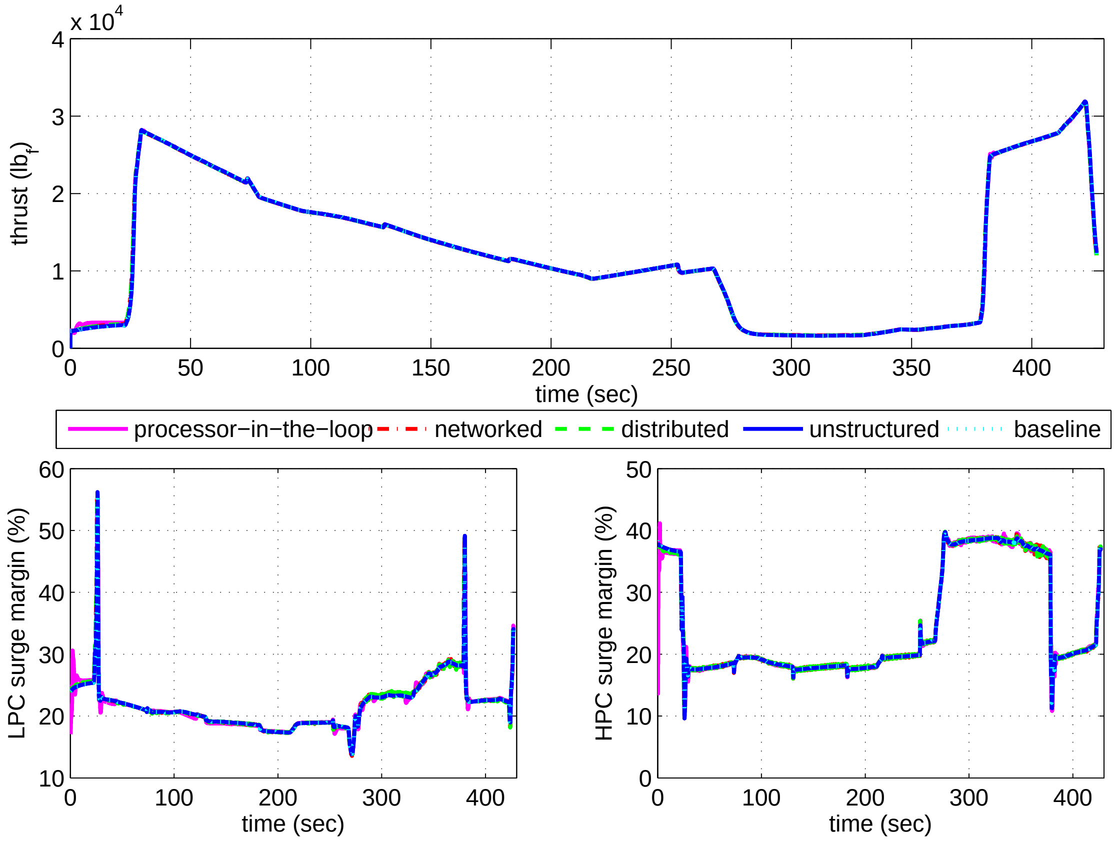

Figure 5. Plots comparing the thrust produced (top plot) and compressor surge margins (bottom row) in the engine models for benchmarking each of the controller configurations in Section II.C. Shown is a representative run of each configuration.

times in the simulation. Due to the current variability of the VBV kicker, it was disabled in all simulations.

To capture the run time variation due to processes executing simultaneously with the simulation, data were collected for five simulations of each model; run time data and model outputs for each iteration were collected. The figures in the paper show a representative run from each simulation whereas the the tables contain summary data of all runs from each simulation.

A. C-MAPSS40k model outputs

The results from benchmarking the controller configurations described in Section II.C, using the flight profile in Fig. 4, are presented in Fig. 5 to 7; these responses are representative of the engine, controller, and SXDs, respectively. Shown is a representative output from each test configuration from the start to the end of the profile. It should be noted that, generally, the responses did not differ significantly between simulations. In the figures, the dotted light blue lines represent output from the baseline case, against which the other results are compared. The solid dark blue lines are outputs from the unstructured model, the green dashed lines from the distributed model, the red dash-dot lines from the networked model, and the purple solid lines from the processor-in-the-loop model.

The plots in Fig. 5 compare the thrust response for each configuration, along with the Surge Margins (SMs) in the Low- and High-Pressure Compressors (LPC, HPC). These three values represent two important characteristics of the engine response: the force it produces, and how closely it approaches two of the operating limits. In general, all the models differed little from the baseline case, particularly in thrust production.

Looking more closely at the results, it can be observed that the initial condition of the simulations and subsequent startup transient differ slightly. Specifically in the processor-in-the-loop case, the SXD models were initialized with different initial conditions than all the other simulations. This caused a small discrepancy at the start of the simulation which then trimmed out. Minute quantization effects by the ADC

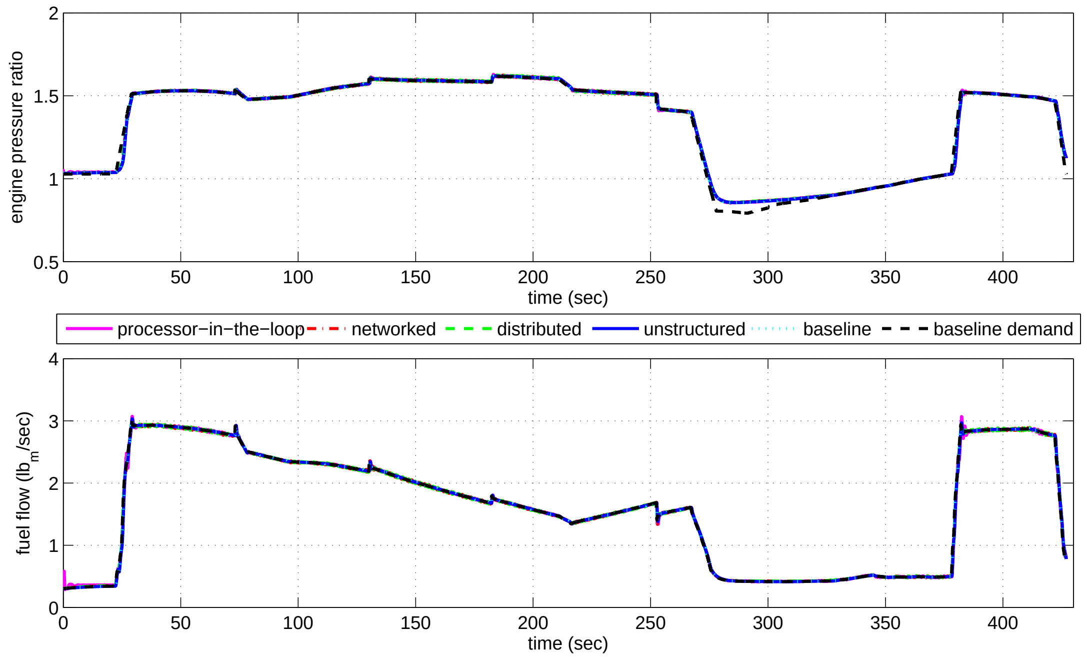

Figure 6. Plots comparing how well the system tracks the EPR demand (top plot) and the fuel flow demand (bottom plot) for benchmarking each of the controller configurations in Section II.C.

also contributed to discrepancies between simulation configurations with smart node models and the baseline model.

In general, many of the observations made for the results in Fig. 5 also apply to those in Fig. 6, which compares the tracking results for each controller. The black dashed line in Fig. 6 represents reference demands for EPR (the value tracked by the controller) and fuel flow (which is regulated by the FMV); the other lines show the response of the systems to these demands. EPR demand is mapped from PLA, so the reference signal provided to the controller was the same in each simulation, while fuel flow demand is calculated based on engine feedback and, therefore, differs between simulations. EPR tracking results match well between configurations. Small differences in how quickly the response tracks the demand are evident only upon very close inspection of the graphs. The distributed model has the fastest response, due to the decreased step size, while the slowest response was from the processor-in-the-loop case, and is related to the presence of hardware integrated with the Simulink models. Overall, little difference appeared between the demand tracking abilities of the models.

Figure 7 contains representative responses of the sensors and actuators in the system; the top row of plots shows the pressure and temperature measurements (not the truth values from the EPM) at station 2 in the engine (the entrance of the fan), and the bottom row shows the outputs of the variable geometry actuators. While the sensor outputs differed little between the models, noticeable differences in the VSV inputs were observed for the processor-in-the-loop configuration. This may be due, in part, to the VSV UDP receive function timing out and recycling old data. In general, the numerical integration methods implemented in the smart node models on the microcontrollers differ from those being executed by Simulink. Figures 5 to 7 show that outside of quantization effects, results from simulation of the distributed model differed little from those of the baseline case. This indicates that the multiple implementations of the C-MAPSS40k system perform equivalently, and that there are few noticeable implementation specific modeling errors.

B. Run time evaluation

In addition to comparing model outputs for each controller configuration in Section II.C, it was of interest to evaluate how quickly the simulations ran. Total run time data was collected for the five test simulations performed for each configuration; the distribution of these run times is listed in Table 1. The data indicate a trade-off in the system between complexity of the models (distributed and processor-in-the-loop configurations) and speed of simulation (baseline and unstructured configurations), as significant

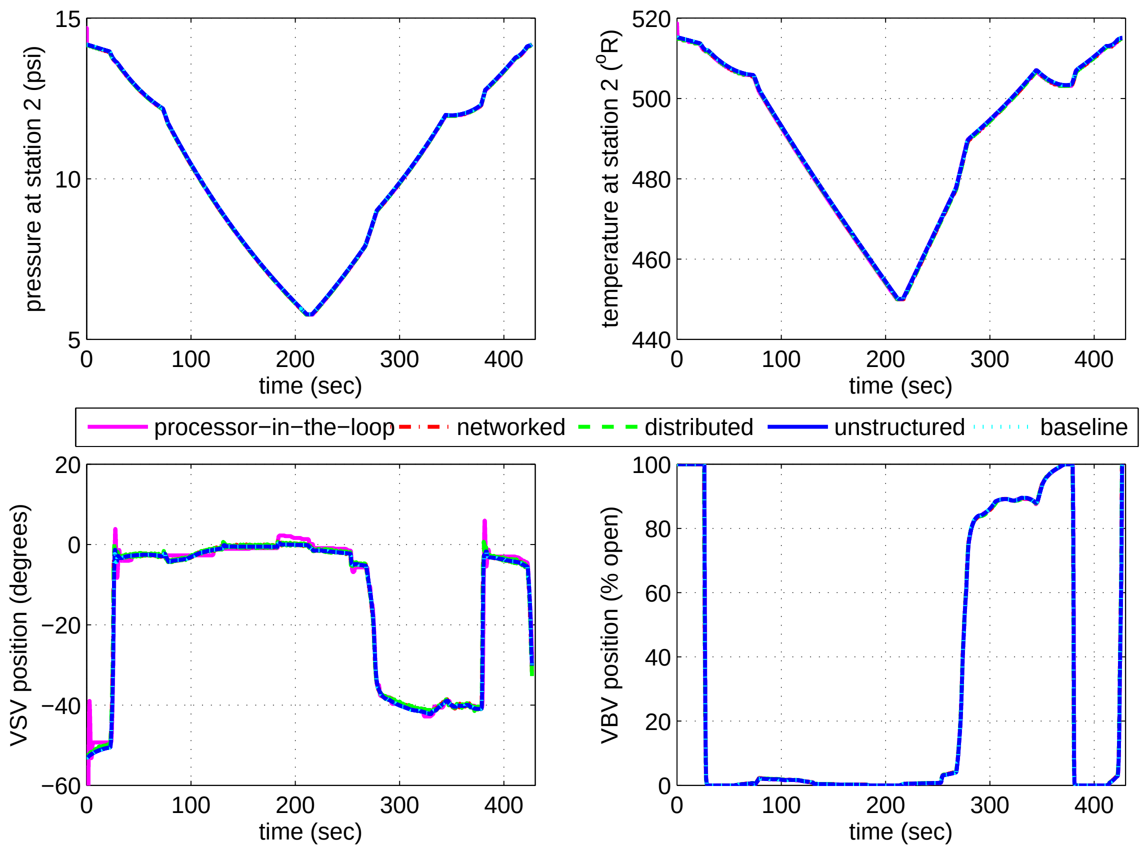

Figure 7. Plots comparing the output of the pressure and temperature sensors at station 2 in the engine (top row of plots, respectively) and of the variable geometry actuators (bottom row of plots) for benchmarking each controller configuration in Section II.C.

increases in run time are seen in the former. The increase for the distributed model is related both to the increased detail in the models of the sensors and actuators and to the increased number of time steps ($29,800$ for the baseline and unstructured configurations and $89,400$ for the others, due to the decreased step size). The run time increase for the processor-in-the-loop case is due to the additional communication required to route data between the CSP and SXD models during the simulation.

Also of note is the increased spread in the distribution of the run times, particularly for the distributed model. Some of this variance can be attributed to background tasks running at the same time as the simulation, which can briefly divert processing power away from Simulink in a random manner. Differences between the models themselves, as modifications and enhancements were made, affected how much run time variation was seen. The baseline and unstructured configurations differed only through the integration of the Ethernet network. While not significantly increasing the average run time, the presence of this hardware network in the model introduced uncertainty in the amount of time required for transferring data between components. In the baseline configuration, this communication occurred almost instantaneously, since a single model file was in use. When simulated on different computers (in the unstructured configuration and others), the total amount of time spent waiting to receive data in the network buffer differed for each simulation. This lead to the increased spread of run times. This effect was further magnified for the distributed configuration, since three times as many time steps were taken.

With the introduction of the networked controller, the packet drop percentage and delay was introduced to the simulation. A notable increase in average total run time was also observed. This is most likely due to increased computational complexity in the simulation, as the network models add additional software that must execute every simulation time step. Also, the packet losses created by the network models may cause the controller to issue commands that fluctuate more often than they would otherwise. These changing actuator commands may cause the engine state solver to run longer.

Table 1. Total average run times for benchmarking simulations of the configurations described in Section II.C. The 'real-time factor' estimates how many times faster than real-time the simulation ran, on average. The uncertainty noted is one-standard deviation of the data.

Configuration	Average total run time (sec)	Real-time factor
Baseline	0.7 ± 0.0	581.4
Unstructured	12.1 ± 0.5	35.2
Distributed	66.3 ± 8.2	6.4
Networked	112.1 ± 4.6	3.8
Processor-in-the-Loop	201.1 ± 7.6	2.1

When the sensor and actuator models were off-loaded to the microcontrollers for the processor-in-the-loop simulation, the total run time of the simulation significantly increased. This was due to the additional communication necessary to route the sensor and actuator data, which increased the time dedicated to UDP communication during each iteration of the simulation; i.e., instead of the CSP transmitting and receiving one packet each from both the EPM and UI in a given simulation time step, the CSP must receive one packet from each of the 13 smart transducer microcontrollers. Additionally, the microprocessors used to execute the SXD models had limited processing capability, which also contributed to increasing the time to process the simulation. These factors slowed the models down so that they ran only slightly faster than real time. One advantage gained from integrating the microcontrollers into the system was a reduction in run time variation, likely because the processing requirements of the CSP were reduced, decreasing the opportunity for background process to interfere.

The final column in Table 1 provides the 'real-time factor,' which indicates how many times faster than real time the model ran. As models with increasing sensor network fidelity are tested, this real-time factor decreases. This is specifically due to the fact that the higher fidelity models are more computationally intensive, and that smaller simulation step sizes are used. As indicated previously, the most significant changes in run time were due to replacing the sensor and actuator models with SXD models and decreasing the step size, which decreased the real-time factor from ≈ 35 in the unstructured case (the simplest configuration that implements a communication network between models) to just ≈ 6 times faster after implementing these changes to produce the distributed model. Integration of packet loss and delay modeling in the networked case, and the implementation of the microcontroller based processor-in-the-loop case further decreased the real-time factor to over three and two times faster than real time respectively. It should be noted that real-time factor was sensitive to the Simulink UDP block settings. Specifically, the number of retry attempts caused significant variations in the real-time factor. Retry attempts specifies the number of times the UDP block will check for new data inside of its memory buffer. If the retry attempts are exhausted and no new data is found, the data from the previous time step is recycled. The default retry attempts used for these simulations was $5,000$. For the SXD models implemented in the microcontrollers, the retry attempt was set to $3,500$.

C. Model Verification Results

In order to attempt to quantitatively verify these modeling approaches, the mean tracking error averaged over five runs (replicates) is given in Table 2. Note that the EPR data from the first half-second of the simulation were thrown out when computing these mean errors, as the simulation sometimes requires a very short trimming period to converge to a consistent operating state. EPR was chosen as the variable over which the simulation configurations are compared, as EPR is the controlled variable in each case, and the effects of digital networks on control performance is a primary focus of investigation in this work. The small standard deviation in Table 2 shows that the difference between runs of the same simulation was very constrained. This indicates high simulation repeatability.

As can be seen from Table 2, the Mean Absolute Error (MAE) generally increases as network fidelity is added. These results can be explained as follows: the control in the unstructured case performs differently than the baseline model due to the addition of a simulation network in which data being transferred between the different models are delayed by one time step and might be lost. The effect of this is only apparent during transients that significantly deviate from nominal. One can expect a control system to operate with a larger error when a delay is added in the loop. Further, the distributed case has slightly larger MAE than the unstructured case. This is expected because the SXD models were incorporated into this simulation, and these models contain effects such as sensor noise and quantization. The networked case performs similarly

Table 2. Mean absolute EPR tracking error expressed as a percentage of the total range of EPR command values (i.e., the EPR range is 0.8, as the EPR command signal varies from 0.8 to 1.6 as seen in Fig. 6). The value of one standard deviation is also shown for each configuration. Note that the standard deviation in the baseline simulation EPR tracking error is zero, because this simulation is deterministic.

Configuration	Percentage Mean Absolute EPR Tracking Error
Baseline	1.047 ± 0.00
Unstructured	1.091 ± 0.001
Distributed	1.197 ± 0.009
Networked	1.198 ± 0.008
Processor-in-the-Loop	1.227 ± 0.004

to the distributed case, even when models of packet delay and packet loss are added in addition to the SXD models. Finally, the processor-in-the-loop controller performs only slightly worse than the unstructured or distributed case. Subtracting the MAE of the processor-in-the-loop from the baseline case, the difference is only 0.151% which indicates they perform very similarly. Differences are likely due to simpler numerical integration techniques used in this controller; for instance, the outputs of the linear sensor and actuator dynamics are computed using the simple forward Euler method, whereas the sensor dynamics in the other simulations are computed using a Runge-Kutta solver. Overall, as can be seen in Table 2, the MAE values in EPR differ little from the baseline case, which indicates that these five implementation schemes effect only tiny changes in the control performance.

IV. Summary and future work

A Hardware-in-the-Loop (HIL) simulation system that offers the ability to test various engine control architectures and algorithms without the need for a physical engine prototype is under development at NASA Glenn Research Center. In order to demonstrate a path to a realistic test bed, a self contained closed-loop engine model was restructured to a processor-in-the-loop configuration. These models implemented in the HIL system were compared against the baseline engine and controller model from the Commercial Modular Aero-Propulsion System Simulation 40k (C-MAPSS40k). C-MAPSS40k was restructured to enable simulation on three separate computers, each containing one of three components (the Engine Plant Model, the Control System Platform (CSP), and the User Interface) communicating over a local area network. Additional modifications were made to the CSP to replace the sensor and actuator models in C-MAPSS40k with Smart Transducer models developed around the IEEE 1451 specifications. These models were implemented in Simulink as part of the CSP, and also implemented on microcontroller boards integrated with the HIL system to simulate an HIL application. To demonstrate that these modifications did not significantly affect the simulation results, and to evaluate the ability to run simulations in real-time, a test flight profile was defined for benchmarking the HIL system. Four controller configurations were considered for comparison to the baseline C-MAPSS40k. Results from benchmarking these alternate configurations have demonstrated that the extended models differed little from the baseline case. Possible topics for future work include modifying the initial condition creation process for the processor-in-the-loop system to reduce initialization transients or the implementation of a 4Mbps control network and the benchmarking of real-time simulations.

References

[1]Culley, D. E., Thomas, R., and Saus, J., "Concepts for Distributed Engine Control," *Proceedings of the 43rd Joint Propulsion Conference and Exhibit*, AIAA-2007-5709, Cincinnati, OH, July 2007.

[2]Culley, D., "Transition in Gas Turbine Control System Architecture: Modular, Distributed, and Embedded," *ASME Turbo Expo2010: Power for Land, Sea, and Air*, Vol. 3, Glasgow, Scotland, United Kingdom, June 2010, pp. 287–297.

[3]Culley, D., Thomas, R., and Saus, J., "Integrated tools for future distributed engine control technologies," *Proceedings of the ASME Turbo Expo 2013*, GT2013-95118, San Antonio, TX, USA, June 2013.

[4]Culley, D., Zinnecker, A., and Aretskin-Hariton, E., "Developing an Integration Infrastructure for Distributed Engine Control Technologies," *AIAA Propulsion and Energy Forum and Exposition 2014: 50th AIAA/ASME/SAE/ASEE Joint Propulsion Conference*, AIAA-2014-3532, Cleveland, OH, July 2014.

[5]Aretskin-Hariton, E. D., Zinnecker, A. M., and Culley, D. E., "Extending the Capabilities of Closed-Loop Engine Simulation using LAN Communication," *AIAA Propulsion and Energy Forum and Exposition 2014: 50th AIAA/ASME/SAE/ASEE Joint Propulsion Conference*, AIAA-2014-3531, Cleveland, OH, July 2014.

[6]May, R. D., Csank, J., Litt, J. S., and Guo, T.-H., *Commercial Modular Aero-Propulsion System Simulation 40k (C-*

MAPSS40k) User's Guide, NASA/TM-2010-216831, September 2010.

[7]May, R. D., Csank, J., Lavelle, T. M., Litt, J. S., and Guo, T.-H., "A High-Fidelity Simulation of a Generic Commercial Aircraft Engine and Controller," *Proceedings of the 46th AIAA/ASME/SAE/ASEE Joint Propulsion Conference*, AIAA-2010-6630, Nashville, TN, July 2010.

[8]Csank, J., May, R. D., Litt, J. S., and Guo, T.-H., "Control Design for a Generic Commercial Aircraft Engine," *Proceedings of the 46th AIAA/ASME/SAE/ASEE Joint Propulsion Conference*, AIAA-2010-6629, Nashville, TN, July 2010.

[9]"IEEE Standard for a Smart Transducer Interface for Sensors and Actuators - Common Functions, Communication Protocols, and Transducer Electronic Data Sheet (TEDS) Formats," September 2007.

[10]"IEEE Standard for a Smart Transducer Interface for Sensors and Actuators - Network Capable Application Processor (NCAP) Information Model," 2000.

[11]"IEEE Standard for a Smart Transducer Interface for Sensors and Actuators - Transducer to Microprocessor Communication Protocols and Transducer Electronic Data Sheet (TEDS) Formats," 1998.

[12]"IEEE Standard for a Smart Transducer Interface for Sensors and Actuators - Digital Communication and Transducer Electronic Data Sheet (TEDS) Formats for Distributed Multidrop Systems," April 2004.

[13]"IEEE Standard for a Smart Transducer Interface for Sensors and Actuators - Mixed-Mode Communication Protocols and Transducer Electronic Data Sheet (TEDS) Formats," 2004.

[14]"IEEE Standard for a Smart Transducer Interface for Sensors and Actuators - Wireless Communication Protocols and Transducer Electronic Data Sheet (TEDS) Formats," October 2007.

[15]"IEEE Standard for a Smart Transducer Interface for Sensors and Actuators - Transducers to Radio Frequency Identification (RFID) Systems Communication Protocols and Transducer Electronic Data Sheet Formats," June 2010.

[16]Zinnecker, A. M., Culley, D. E., and Aretskin-Hariton, E. D., "A modular approach to modeling hardware elements in distributed engine control systems," *AIAA Propulsion and Energy Forum and Exposition 2014: 50th AIAA/ASME/SAE/ASEE Joint Propulsion Conference*, AIAA-2014-3530, Cleveland, OH, July 2014.

[17]Zaretsky, E. V., Litt, J., Hendricks, R. C., and Soditus, S. M., "Determination of Turbine Blade Life From Engine Field Data," NASA/TM-2013-2170308, April 2013.