SSC22-WKP1-15

Development of a Simulation Framework for CubeSat Performance Modeling

Mark A. Kurtz, Andrew S. Keys, Kirk W. Johnson
Air Force Institute of Technology
2950 Hobson Way, WPAFB, OH 45433
mark.kurtz.1@spaceforce.mil

ABSTRACT

Space systems are notoriously difficult to develop due to the nature of the environment in which they must operate. Designers have only a limited window to ensure systems will function as intended, placing a high importance on testing. This paper discussed the ongoing development of a simulation framework to support Hardware-in-the-Loop (HIL) testing of CubeSat subsystem hardware. This work is being conducted at the Air Force Institute of Technology (AFIT) in support of the institution's CubeSat program. The simulation framework is organized into the classic spacecraft subsystems. Each of these subsystems will support a software model and interfaces for the integration of flight hardware into the simulation framework. In demonstration of this concept, propulsion hardware has been successfully integrated into the model environment. Telemetry reception and command transmission within the simulation framework is functional and demonstrated. A loop containing the propulsion hardware, simple controller, and orbital motion propagator was developed to demonstrate the HIL test functionality of the simulation framework. This focus on the development of the propulsion HIL test configuration is a point of distinction from other HIL simulations, which typically focus on the Attitude Determination and Control System (ADCS). Presented results validate successful integration of propulsion subsystem hardware into the simulation framework. Future work will focus on the integration of CubeSat subsystem models into the framework.

INTRODUCTION

Designing space systems is an incredibly difficult task thanks largely to the demanding environment in which these systems operate. Space systems must work perfectly in a hostile environment which cannot be completely replicated on the ground. Mistakes and flaws typically cannot be corrected once a spacecraft is in its operational orbit. Consequently, a strong test campaign is vital to the success of any space system. This is even more so the case with CubeSat programs. Such programs typically have much smaller budgets and much faster schedules than their traditional counterparts. The CubeSat systems themselves usually lack redundant components and/or subsystems. Teams are often smaller and less experienced. These factors and limitations make a strong test campaign simultaneously more important and more challenging to achieve for CubeSat programs.[1]

Success under these circumstances means effectively utilizing limited resources. A common approach is to use hardware-in-the-loop simulations.[2] Such simulations replace a hardware component's operational environment with an emulated equivalent.[3] For example, magnetic field, sun and star locations, and other space environment characteristics can be modeled and fed to the Attitude Determination and Control System (ADCS) such that the ADCS believes it is in its operational environment. Such a HIL test setup enables development and testing of the system without requiring additional spacecraft hardware.[2] The Air Force Institute of Technology (AFIT) operates its own CubeSat program. Several missions are planned and under development, but AFIT has not yet had an operational system on orbit. Consequently, testing within AFIT's CubeSat program is in its early stages. Like most other CubeSat programs, AFIT's CubeSat program is constrained in terms of schedule and budget. Additionally, research efforts lack a high level of continuity as student and faculty researchers transition through the institution.[4]

As a result, there exists a clear necessity for research aimed at improving the AFIT's ability to test CubeSat hardware. Development of a Hardware-in-the-Loop simulation was selected as the best means of improving test capabilities. The objective here is to develop a simulation framework which can be used to test flight hardware in loop with its simulated environment. This approach provides a number of advantages compared to other possibilities. For example, engineering test units can be constructed out of flight or flight-like hardware. While this approach provides a highly accurate representation of the operational system, it necessitates the acquisition of additional copies of system components.[5] This can be expensive and time consuming for CubeSat program's like AFIT's. It also is typically not feasible to produce enough test units for everyone

who needs one, potentially resulting in wait times and schedule delays. HIL simulation can decrease dependency on hardware availability, enabling faster testing without unnecessary delays.

The second objective of this research is to demonstrate the integration of system hardware into the simulation architecture. The propulsion subsystem was selected as the subsystem of focus for this objective. There were several reasons for this selection. One of AFIT's missions in development utilizes a propulsion subsystem for maneuvers.[4] AFIT has not developed a propulsion-capable CubeSat before, so it is highly desirable to focus additional testing efforts on the propulsion subsystem. Additionally, propulsion subsystems are not very common on CubeSats, nor are they often the focus of HIL simulation efforts. Typically, CubeSat HIL simulation efforts focus on the ADCS as this system is present in every CubeSat and often contains the most complex hardware.[2] Thus, HIL simulation solutions are much less readily available for AFIT's propulsion subsystem. At the time of writing, no HIL simulation effort incorporating this specific propulsion subsystem could be identified, adding a degree of novelty to this chosen subsystem focus.

The third objective is to effectively utilize parallel research efforts to the greatest extent possible. For a HIL simulation built to test CubeSat hardware, the simulated environment includes the CubeSat's other subsystems in addition to the physical space environment. It is therefore necessary to simulate these subsystems, or at least aspects of these subsystems. Several ongoing research efforts at AFIT are focused on modeling and simulating some of AFIT's CubeSat subsystems. Incorporating these efforts into the simulation architecture delivers several key advantages. The first is that duplication of effort is avoided. Given the time and manpower constraints of a CubeSat program, this is a more efficient utilization of limited resources. Additionally, incorporation of these parallel models provides a degree of continuity over the involved research. Given the fairly rapid turnover of students and faculty at AFIT, it is not uncommon for research to lose continuity. This is true even of the research efforts connected to AFIT's CubeSat programs. The simulation framework is intended to be further developed as the CubeSat program progresses, providing a degree of continuity for the integrated subsystem models. The simulation framework is being designed to allow for real subsystem hardware and its simulated software equivalent to run within the simulation. This will allow for subsystem models developed through other research efforts to be validated against the actual hardware they are attempting to model. Additionally, the framework will enable the included subsystem models to interact with each other, providing additional insight into each model's performance.

SIMULATION FRAMEWORK METHODOLOGY

The simulation framework is designed with the intent of supporting various subsystems as either hardware or software components within the simulation. The idea here is that the user will configure which subsystems are physically connected as hardware and which are modeled within the simulation depending on testing requirements. For example, a HIL test of the ADCS would see the simulation set to connect to ADCS hardware with the other subsystems selected to run as software models within the simulation framework.

Strucutre and Organization

The framework is organized into the classic CubeSat subsystems[1] as shown in Figure 1. Because CubeSats are typically developed, built and tested at the subsystem level, it made the most sense to apply this organizational scheme to the simulation framework. Additionally, the CubeSat's external environment is contained within a separate subsystem. This organization makes adjustment of the external environment model simple and mirrors the CubeSat's operational environment.

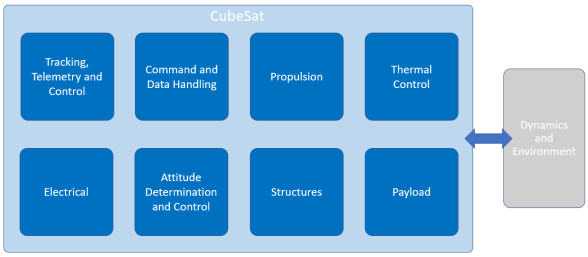

Figure 1: Simulation Framework High Level Architecture

Each subsystem block within the framework consists of two major components; the hardware interface and software-based model as shown in Figure 2, which depicts the propulsion subsystem. A switch enables the subsystem to either interface with the system's corresponding hardware, or to utilize a software-based model. The framework is built to enable this duality for each of the included subsystems. Currently, development has been limited to demonstrating this functionality within a single subsystem. As described previously, the propulsion subsystem was selected as the ideal candidate.

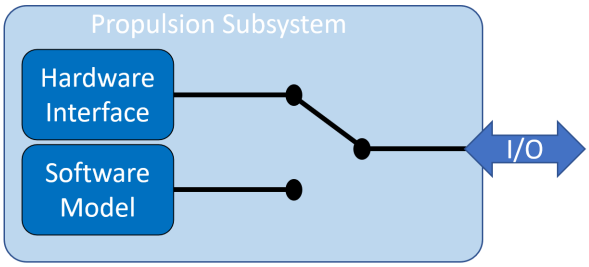

Figure 2: Subsystem Architecture for Hardware and Software Interfacing

Hardware Integration

The propulsion subsystem hardware integrated into the simulation framework is the SNIPE Micro Propulsion System (MiPS) produced by Vacco Industries. This system is a small form factor cold gas propulsion system designed for CubeSat missions and will be flown on an upcoming AFIT mission.[6]

The propulsion unit uses R-236fa as its cold gas propellant to provide 25 mN of thrust and a specific impulse of 40 seconds for each of four nozzles. The four nozzles point along the same axis and are aligned to within 0.1 degrees of each other. The Vacco MiPS contains 1.2 kilograms of propellant within a footprint of just 12x10x10 cm. Communication with this propulsion system occurs via an RS-422 serial port. This interface allows for serial communications with the propulsion subsystem at a data rate of up to 10 mbps.[7]

To facilitate the development and testing of CubeSat missions using their propulsion system, Vacco Industries furnishes an emulator in advance of delivery of the actual propulsion unit. The emulator is a physical piece of hardware consisting of a logic board and RS-422 serial interface encased within a metal and glass housing. The emulator runs the propulsion system software and mimics the functioning of physical components such as its heaters or thruster valves. This propulsion emulator was used to develop the simulation framework's ability to interface with the propulsion system. This was done because the emulator was available well in advance of the flight hardware yet utilizes identical communications protocol.

Integrating the propulsion subsystem into the simulation framework required several additional components. As shown in Figure 3, the hardware must be connected to a hardware interface. This is a physical component which adapts the hardware's interface to that of the system hosting the simulation framework. On the host system, drivers corresponding to the hardware interface component are needed in order for the system to recognize the CubeSat hardware. Finally, configuration blocks within the simulation software environment enable communication between the simulation framework and the propulsion system hardware.

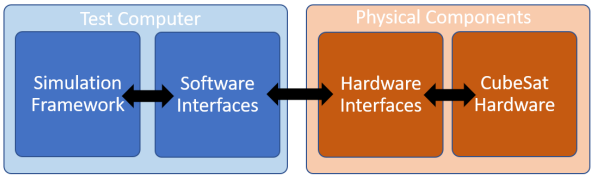

Figure 3: Hardware to Simulation Framework Structure

The hardware interface used here is a Ulinx serial converter. This device adapts a serial cable connected to the RS-422 port to USB as shown in Figure 4, ensuring compatibility with most computers. The system used to build and run the simulation framework is a laptop running a standard installation of Windows 10.

Figure 4: UlinxRS-422/485 to USB Adapter

Drivers must be installed for the host computer to detect hardware connected via the Ulinx adapter. The drivers are the software interface between hardware and simulation. Once installed, connected hardware will show up in Device Manager as a COM port. Device Manager will specify which com port is connected. The port is often labeled "COM3" but can contain other numbers depending on the host computer's specific configuration.

Simulation Software

Because modeling and simulation is not not a new concept, many software environments exist which can be used to create powerful simulations . Mathwork's Simulink was selected as the software environment for the simulation framework. Simulink provides strong support for interfacing with external devices. Additionally, the majority of parallel subsystem modeling efforts were also being built within Matlab and Simulink. Thus, building the simulation framework within Simulink ensures a high degree of compatibility with the models that have been identified for integration into the framework. Additionally, Simulink enables the organization of simulation components into subsystems,

providing a strong visual organization to the simulation framework.

The simulation framework runs on Simulink 2021a. To complete the interfacing of the simulation with the propulsion system, several specific Simulink blocks were utilized. The first of these is the Real-Time Sync block. Without this block, the simulation will attempt to iterate as fast as possible. While this can be useful for an entirely virtual CubeSat model, the hardware-interfaced simulation framework must run in real time. The Real-Time Sync block accomplishes this by coordinate computation at each step iteration with accurate time-keeping.

Along with this block, input and output blocks are needed. The "Packet Input" and "Packet Output" blocks from the Simulink Real-Time Desktop Library were selected to provide input from and output to the CubeSat propulsion hardware. Though other blocks are available in Simulink's libraries, only blocks within the Real-Time Desktop Library are compatible with real time operation. Both blocks must be configured to utilize an installed data acquisition board. Serial ports are treated as boards within these blocks, so "Serial Port"[3] was installed and selected for both blocks as shown in Figure 5. This figure depicts the Packet Input block and its configuration options. The "board" was configured with a Baud rate of 115200, eight data bits, one stop bit and no parity in accordance with the Vacco MiPS Interface Control Document (ICD).[6]

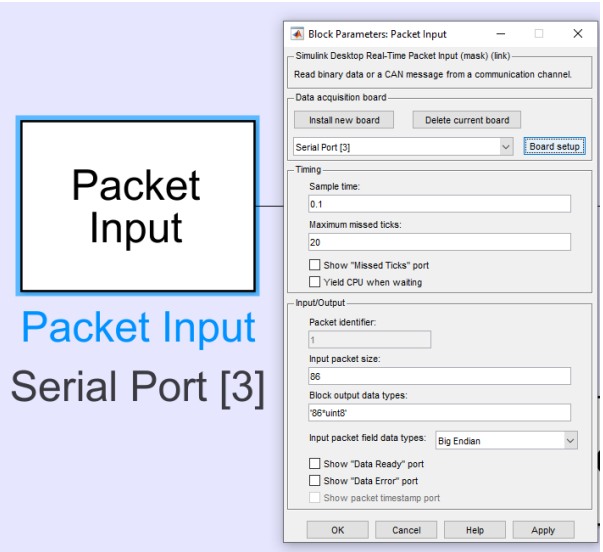

Figure 5: Packet Input Block and Configuration Options

The Packet Input block was configured to interpret input from hardware as packets of 86 bytes in length. This corresponds to the length of the telemetry message produced and transmitted by the propulsion system. Additionally, the input block was configured to interpret incoming data as Big-Endian unsigned eight-bit integers. This corresponds to the specifications detailed in the propulsion ICD.[6]

Similarly, the Packet Output block is configured to output packets ten bytes in length, also as Big-Endian unsigned eight-bit integers. The block and its configuration options are shown in Figure 6. All commands accepted by the propulsion system follow this structure. The inclusion and configuration of these blocks within the simulation framework completes the chain of components, both hardware and software, needed to interface the propulsion subsystem hardware with the simulation framework. Multiple input and output blocks can be utilized within the simulation as long as they are properly configured.

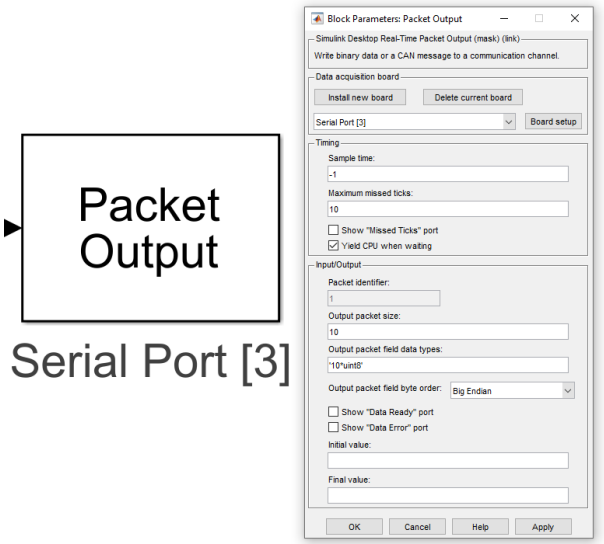

Figure 6: Packet Output Block and Configuration Options

PROPULSION HARDWARE COMMAND AND TELEMETRY INTEGRATION

With the communications link established, the simulation was then configured to interpret incoming data (telemetry) and properly format and transmit commands when needed. To this end, it is necessary to understand how the propulsion system transmits and receives data and to then develop what are essentially the drivers that enable the meaningful interfacing of the propulsion system with the simulation.

Telemetry Reception and Parsing

As specified previously, telemetry received from the Packet Input block is packaged into packets of 86 bytes in length. Each packet is a telemetry message transmitted

from the propulsion hardware and contains all available data pertaining to hardware status. Every packet begins with a start byte. and ends with a stop byte. Each byte within the telemetry message is an eight bit unsigned integer. In this state, each telemetry message packet is not useful to the simulation framework. Thus, the propulsion subsystem model must parse the packaged bytes into data that can be utilized by the simulation (such as hardware temperatures, pressures, etc). The parser was built as a custom Matlab function named ”parse_packets” within the propulsion subsystem model. It converts every byte in the telemetry message through a number of different operations.

Eight bit (one byte) integers can have a maximum value of 255, but the propulsion system needs to transmit larger values. This means that some data points are constructed from two or even three bytes. The propulsion system divides larger numbers into their Most Significant Byte (MSB), Least Significant Byte (LSB) and middle byte (when three bytes are needed). The MSB consists of the eight leftmost bits in the number and the LSB consists of the eight rightmost. To get the proper telemetry datapoint from the buffered telemetry message, the MSB and LSB must be reconstituted into a single value. Within the parse_packets Matlab function, a left logical bit shift is performed to shift the MSB eight bits to the left. The shifted MSB is then added to the LSB, creating a 16-bit integer. For many of the items in the telemetry message, a bit value must also be multiplied against this newly-created integer. Completing this process transforms the package of bytes into data that accurately reflects the current condition/configuration of the propulsion system hardware. For example, converting bytes two and three of the telemetry message into a single integer via this process returns a value which accurately reports the current pressure of the system’s propellant tank.

Some of the bytes in the telemetry message contain information on a bit-level. That is, each bit of the byte indicates something different about the propulsion system’s status. Because bits can only be one of two values (0 or 1), this bit-level telemetry usually indicates an on/off condition or whether an error condition is present or not. To obtain this information, the relevant bytes must first be converted to binary. The parser function then indexes the binary value, which MATLAB treats as a string, and assigns each bit to a descriptive variable name. Because Simulink usually cannot pass strings, each bit must then be converted into integers.

Development of the parse_packets function relied heavily on the propulsion system’s ICD which contained detailed information about each byte, including maximum and minimum values, bit values, and units. Though every byte in the telemetry message is converted by the parser, the block is configured such that telemetry items are not passed out to other blocks by default. Instead, the user adds or removes outputs by altering the parse_packets Matlab function. Consequently, parser function output can be scaled to include additional telemetry items as the simulation framework expands.

Commanding the Propulsion Hardware

In order for the simulation framework to properly interact with propulsion system, the simulation must be capable of transmitting commands to the hardware. The propulsion system requires specifically formatted commands and will reject all other inputs. Every command is exactly ten bytes long. The first byte is always the start byte and the last byte is always the stop byte. The same start and stop bytes (unsigned eight bit integers 123 and 125, respectively) that are used for the propulsion system’s telemetry message are used in every command.

The second byte of each command is an identifier byte, called the “Command Op Byte,” which is unique to every different type of command. For example, the Command Op Byte for the “commence thrusting” command is 0xA1. No other type of command uses this Command Op Byte, but it is always the same every time the commence thrusting command is issued. The third through eighth bytes of each command are specific to the contents of each command. These are the bytes that actually convey instructions to the propulsion system. Some commands require the transmission of less data than others, so some of these bytes may simply be “0” if unused. The ninth byte of each command is always a Cycle Redundancy Check (CRC) byte. This byte helps the propulsion hardware determine if a received command is valid and free of errors.

As with the telemetry message, the Vacco ICD lists each command that can be sent to the propulsion system and details what information is sent and how it must be structured. For example, each thruster can be fired for a specified duration after a specified delay. For these thruster commands, bytes three through five contain the delay time, and bytes six through eight contain the fire duration time. Multiple bytes are used to transmit these numbers to allow for the transmission of larger values. If only single bytes were used, the largest value that could be transmitted would be 255. While it is technically possible for a user to memorize three-byte sequences or for the simulation to utilize only pre-programmed parameters, this is inherently limited and not desirable. Instead, the simulation is configured for a user to alter parameters directly. That is, if the user wants to change a delay time, the user alters that value and the simulation automatically converts it into a three-byte sequence.

The Vacco ICD also specifies the units of the data contained in each command if applicable. In the case of the thruster commands, delay and firing times are in milliseconds. The simulation is configured to accept user inputs in the ICD-specified units.

A significant source of challenge was in configuring commands to transmit only once. Simulink will compute over the entire model at every time step. This is desirable behavior for the rest of the model, but transmission of thruster commands must be single, conditional events. Additionally, the propulsion hardware requires at least 100 milliseconds between receiving commands. Repeated commanding transmission of thruster firing commands can reset and override existing commands even if the system is midway through firing. Thus, repeated transmission of commands at each time step would make most aspects of hardware interaction, such as programming and firing thrusters, impossible. This problem was solved through the use of triggered subsystems. A triggered subsystem only activates when it has received a conditional input. Typically, this input is a change of state, such as a rising or falling signal. Command transmission blocks were built entirely within each triggered subsystem, which were configured to activate only when receiving a rising signal. Within the propulsion subsystem model, conditional events which need to trigger transmission of a command are translated into a pulse with an integer value of one, lasting the duration of a single time step. This signal is fed into the trigger input of triggered subsystems, which otherwise receive a default value of negative one. The subsystems thus interpret each pulse as a rising signal, activate only at that time step, and transmit their respective commands. Figure 7 generically depicts the structure utilized for command transmission. This can be implemented as many times as needed within the simulation. For example, commands to change propellant tank temperature and to program and fire thrusters are contained within separate triggered subsystems which receive separate trigger pulses. Commands are computed initially at simulation start in the simulation framework's initialization file. This means that all possible commands must be preplanned prior to the start of the simulation. While this approach is somewhat limiting, it is more in-line with expected CubeSat operations. Commands are brought into the simulation as 10x1 arrays within constant blocks.

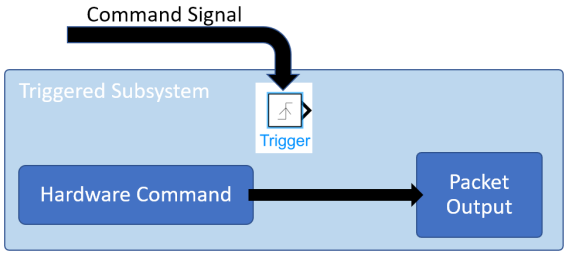

Figure 7: Command Transmission Structure

DEMONSTRATION AND VALIDATION

To complete the "loop" of a HIL simulation, the interfaced hardware must provide input to the simulation and be affected by the simulation outputs it receives. This is a key function of the simulation framework. To demonstrate this functionality, the simulation was configured to attempt to maneuver to a target. This functionality was chosen because maneuver is a key (and arguably most important) aspect of the propulsion system as it pertains to CubeSat performance. To complete this loop, two additional components were implemented; a dynamics model and a simple ADCS model. Figure 8 depicts the interaction between these components. The propulsion system provides a thrust input to the dynamics model. The dynamics model computes CubeSat position which is then provided to the simple ADCS model. The ADCS model then determines if additional thrust is needed and transmits the appropriate signal to the propulsion system.

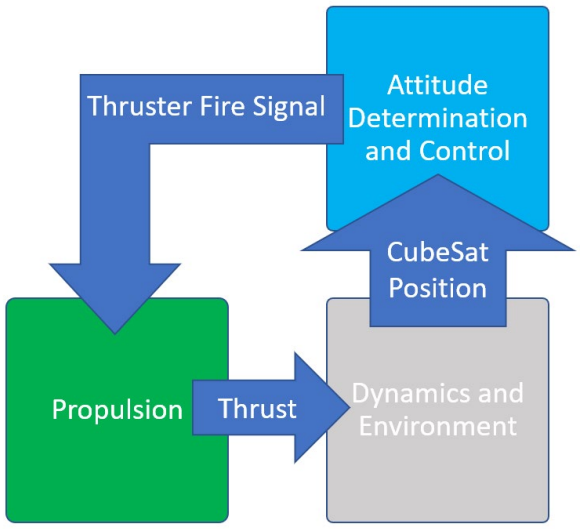

Figure 8: Hardware-in-the-Loop Configuration for Propulsion Subsystem

The dynamics model computes the CubeSat's position relative to a point in orbit around the Earth. This point is arbitrary and was selected for the purpose of

demonstrating simulation functionality. The user specifies initial conditions for this point and the CubeSat. The point is specified using the six classical orbital elements. The CubeSat's initial conditions can be specified in the same manner or in terms of relative position and velocity to the selected point. The dynamics model uses a pair of two-body-problem orbit propagators to compute the relative position and velocity of the CubeSat during the simulation. Two orbits are computed and the difference in position and velocity between the two is calculated. The CubeSat's orbit propagator includes a force input to incorporate thruster outputs, but the propagators otherwise do not include perturbation effects. The model assumes that thrust occurs purely in-track.

The propulsion system hardware does not directly provide output thrust in the telemetry message. Thrust was necessarily derived from other telemetry data points. The propulsion subsystem built into the simulation framework provides for two possible methods of obtaining thrust values. The first simply assumes a nominal thrust at the thruster specification of 25 mN. The second utilizes plenum pressure telemetry to perform an evaluation of a manufacturer provided polynomial. The former approach is included in the model because the propulsion hardware emulator lacks the ability to dynamically model propellant tank dynamics. The emulator was used to build the simulation framework, so this option was necessary despite the pressure telemetry approach being a more accurate measure of output thrust. For both cases, the propulsion subsystem model evaluates thruster fire time telemetry to determine if that thruster is firing. If the firing time for a thruster is decreasing, that thruster is firing and model outputs its corresponding thrust. Constant firing time values indicate a non-firing thruster in which case the model does not output a thrust.

The ADCS model is a simple bang-bang controller constructed solely to demonstrate the ability of the simulation framework to command the payload hardware based on position data received from the dynamics model. The controller determines if the CubeSat is within a user-configurable tolerance of a target position and requests thruster fire if not within this tolerance. Only in-track position is considered and the CubeSat was placed in-track with its arbitrary target. This controller is not a permanent fixture of the simulation framework, but is necessary to demonstrate completion and operation of the hardware loop.

RESULTS

The simulation framework is capable of interfacing with propulsion system hardware. It can both accurately receive telemetry and transmit commands with the connected hardware. An example of telemetry reception is shown in Figure 9, which depicts plenum pressure telemetry received from the propulsion hardware emulator. For this simulation run, the emulator measured ambient pressure. The pressure data received by the simulation matches what was expected: about 743 torr, or 14.3 psi. This indicates that the simulation framework properly receives and parses hardware telemetry.

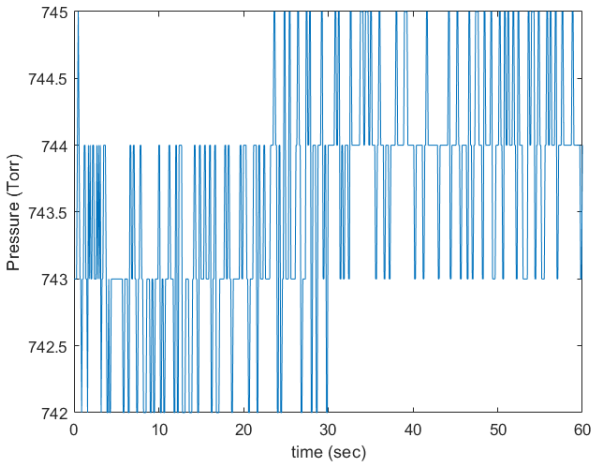

Figure 9: Plenum Pressure Telemetry from Propulsion Hardware Emulator

Additionally, the simulation framework successfully functions in a HIL configuration. The interfaced propulsion hardware inputs telemetry into the simulation. As shown previously in Figure 8, this telemetry (specifically thruster firing times) is converted into a thrust output which is fed into the Dynamics and Environment subsystem. This subsystem calculates the CubeSat's position, which is fed to a simple controller in the ADCS subsystem, which in turn sends a fire signal to trigger the transmission of thruster commands to the hardware, completing the loop. The expected result from this HIL demonstration configuration is twofold: First, the Propulsion subsystem telemetry will depict the execution of multiple thruster firing commands. Second, the relative in-track position of the CubeSat will change as the thrust input from the hardware is incorporated into orbit propagation. Figure 10 depicts thruster fire telemetry over the duration of a simulation run. The figure shows the expected behavior: namely that multiple thruster firings occur as a result of feedback generated by the simulation. As shown, the simulation was configured to produce thruster firings of a one second duration and was configured to assume 25 mN thrust output. Note that in Figure 9, all four thrusters are firing with identical firing regimes. The commands were preprogrammed to fire all thrusters.

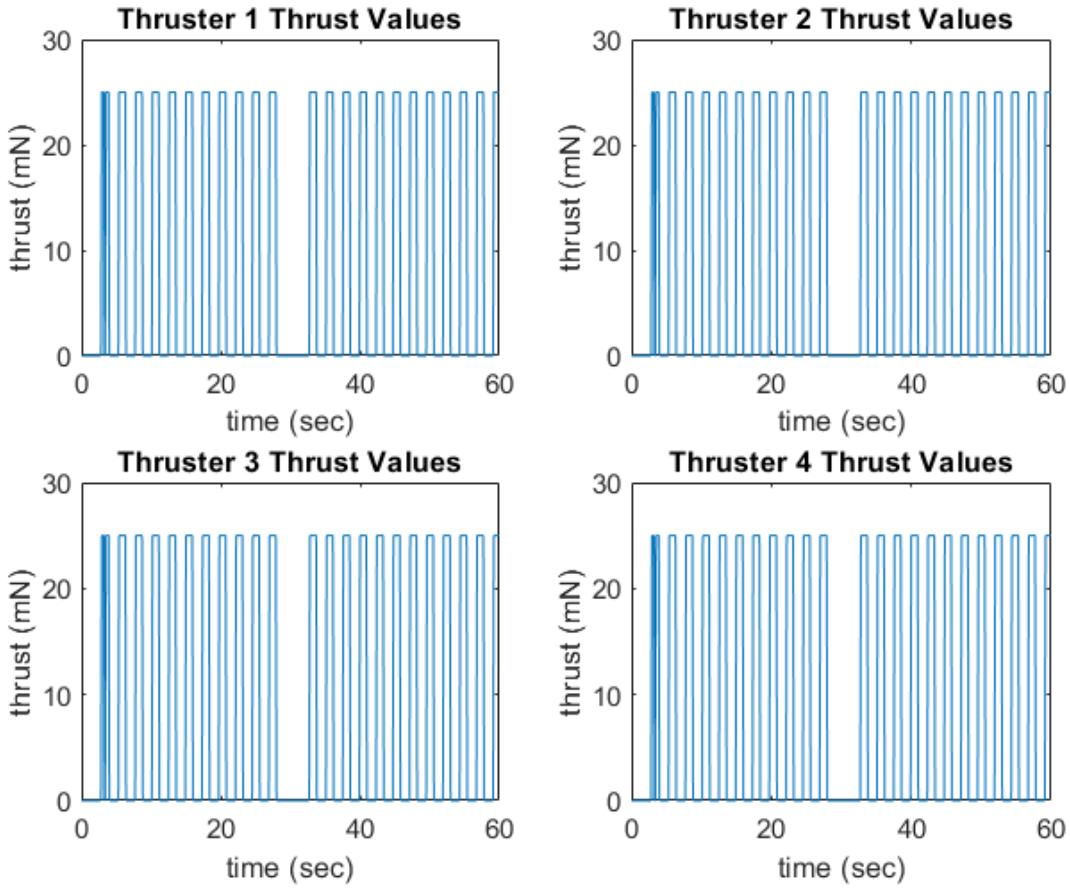

Figure 10: Thrust Output from Each Thruster

The thrust output produced by the propulsion subsystem is fed into the dynamics subsystem. This output produces the relative motion shown in Figure 11. The CubeSat starts initially at 25 meters relative to an arbitrary target (10 meters in-track). In-track thrust is applied to maneuver towards the target. At a distance of +/- 2 meters, the simple ADCS considers the CubeSat to have reached the target. The change in slope of this figure clearly indicates that the simulation changes its response based on calculated position relative to a target. Note that observing a change in response is the goal of this simulation run. A successful convergence near the arbitrary target is not of particular relevance for this goal.

Within the tolerance range of the target, no additional thrust is requested. This can be seen clearly in Figure 12, which plots thruster fire requests over time. Each request appears as a pulse of magnitude 1. Each request triggers the transmission of the pre-configured one-second thruster firing command. As this figure shows, the cessation of fire requests coincides precisely with the CubeSat's arrival within tolerance of its target location. The firing requests line up with their corresponding thruster firings depicted in the Figure 10 of the hardware telemetry. The gap in both figures indicates that the simple ADCS controller ceases to request thruster fire while the CubeSat is within tolerance of an arbitrarily-selected target (10 meters in-track). This demonstrates that commands are indeed triggered by simulation feedback which changes as a result of conditions within the simulation.

Thus, the simulation framework is capable of operating with hardware in the loop. Telemetry is received and utilized by the simulation to simulate a relevant environment and produce commands transmitted to the hardware. Although the simulated subsystems and environment are not to the fidelity of full-fledged CubeSat subsystems, they are sufficient to demonstrate this key functionality.

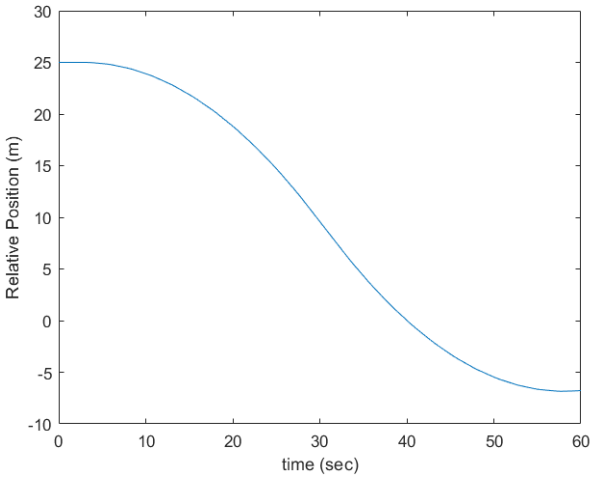

Figure 11: Computed In-Track Position Relative to an Arbitrary Target

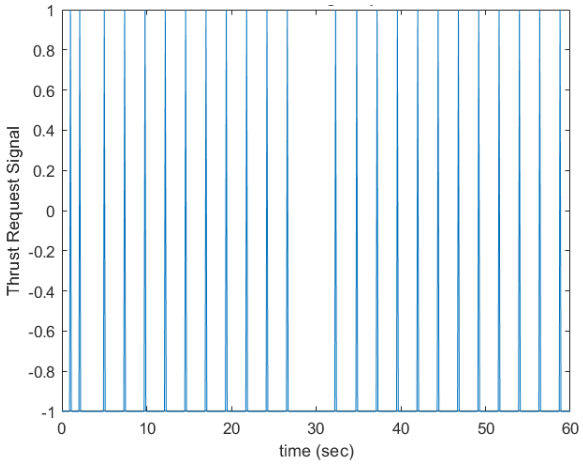

Figure 12: Firing Request Pulses Used to Trigger Transmission of Firing Commands

CONCLUSION

The work presented here is being developed to aid AFIT's CubeSat program through the creation of a framework design to bring hardware and software models of CubeSat subsystems together. The ultimate vision of this framework is to integrate hardware interfacing and software models for every subsystem on a given CubeSat. The simulation framework will become a flexible tool to assist its users with the testing and development of CubeSat programs.

Initial functionality and proof-of-concept has been successfully demonstrated. Integration of hardware (or at least its emulator stand-in) with subsystem and environmental models shows that the framework is capable of running in a HIL configuration and thus can support the testing and development of CubeSat hardware.

Development of the simulation framework is ongoing. Future work will focus on the integration of CubeSat subsystem models currently being developed in related research efforts. While availability is largely bound by the progress/pace of those other efforts, integration of subsystem models is an important piece of the simulation framework. In particular, a model of the propulsion system is currently being integrated. Successful integration of this model will allow for a number of possibilities. Chief among these is model validation. Performance of the software propulsion model could be evaluated against that of the actual hardware. This will also demonstrate the hardware/software configurability aspect of the simulation framework.

Additionally, the currently implemented models need to be refined and expanded. The ADCS controller built for validation/demonstration purposes needs to be replaced with a complete ADCS. The dynamics subsystem would benefit from the inclusion of rotational dynamics. Further in the future, other subsystem models will need to be integrated along with interfaces for their hardware counterparts. In time, this simulation framework can become a truly powerful test and validation tool.

REFERENCES

1. Wertz, J. R., Everett, D. F., and Puschell, J. J., Space Mission Engineering: the new SMAD, Microcosm Press, 2011.

2. Kiesbye, J., Messmann, D., Preisinger, M., Reina, G., Nagy, D., Schummer, F., Mostad, M., Kale, T., and Langer, M.,"Hardware-in-the-loop and software-in-the-loop testing of the MOVE-II CubeSat," Aerospace, Vol. 6, No. 12, 2019, pp. 1–25.

   https://doi.org/10.3390/aerospace6120130.

3. Kossiakoff, A., Sweet, W., Seymour, S., and Biemer, S., Systems Engineering Principles and, 2nd ed., Hoboken, NJ, 2011.

4. Keys, A., and Sheffield, C., "Grissom Project Management Plan," 2020.

5. Geletko, D. M., Grubb, M. D., Lucas, J. P., Morris, J. R., Spolaor, M., Suder, M. D., Yokum, S. C., and Zemerick, S. A., "NASA Operational Simulator for Small Satellites (NOS3): the STF-1 CubeSat case study," 2019.

   http://arxiv.org/abs/1901.07583.

6. Rezaei, R., Sorathia, J., and Bhandari, R., "Interface Control Guide for the Korea Astronomy and Space Science Institute SNIPE Micro Propulsion System," 2019.

7. Mach, D., Yengonian, D., and Bhandari, R., "SNIPE MiPS (Propellant Filled)," 2021.