Multi-spacecraft Autonomous Positioning System: LEO Demo Development

Evan Anzalone, Chris Becker, Danielle Crump, Daniel Heater
NASA Marshall Space Flight Center
Huntsville, AL 35812; 256-544-5492
Evan.J.Anzalone@nasa.gov

ABSTRACT

The Multi-spacecraft Autonomous Positioning System enables a solar-system wide navigation capability. This architecture takes advantages of a growing interplanetary communication network. The spacecraft, as well as ground-based assets, share best estimated position and velocity, through embedded navigation packets. Combined with timing observations, this supports onboard autonomous state estimation. Through simulation-based analysis, this architecture has been shown to maintain state estimation accuracy during a Martian cruise while reducing the need for Earth-based state updates. The current stage of this research is development of a hardware-in-the-loop (HIL) simulation to further analyze this architecture. This allows for embedding of space-capable (and space-flown) components into a multi-agent hardware simulation. This demonstration of the architecture is used to both verify the simulation results, as well as to provide a proving ground and experimental test-bed for flight software development, evaluation, and algorithm optimization. By using COTS components, and in-house developed flight software libraries, a multi-spacecraft hardware-in-the-loop simulation has been developed. A Low Earth Orbit mission has been designed to demonstrate the architecture's performance. The HIL simulation of the flight software and architecture captures the first steps towards the further development of a platform to demonstrate this navigation technology, laying the groundwork for further architecture expansion.

OVERVIEW

As the number of spacecraft in simultaneous operation continues to grow, there is an increased dependency on ground-based navigation support. The current baseline for deep space navigation uses Earth-based radiometric tracking, requiring long duration observations to perform orbit determination and generate state updates. The age, complexity, and high utilization of the ground assets pose a risk to spacecraft navigation performance. In order to perform complex operations at large distances from Earth such as landing and proximity operations, autonomous systems are required. With increasingly complex mission operations, the need for frequent and Earth-independent navigation capabilities is further reinforced.

The Multi-spacecraft Autonomous Positioning System (MAPS) takes advantage of the growing inter-spacecraft communication network and infrastructure to allow for Earth-autonomous state measurements. This type of network is already being implemented and routinely used in Martian communications, through the use of the Mars Reconnaissance Orbiter and Mars Odyssey spacecraft. The growth of this architecture is continued through MAVEN, and future commercial Mars telecom orbiters. This growing network provides an initial Mars-local capability for inter-spacecraft communication and implementation of navigation capability. These navigation updates are enabled by cross-communication between assets in the network, coupled with onboard navigation estimation routines to integrate packet travel time to generate ranging measurements. Inter-spacecraft communication allows for frequent state broadcasts and time updates from trusted references. The architecture is a software –based solution, enabling its implementation on a wide variety of current assets, with the operational constraints and measurement accuracy determined by onboard systems (processing, pointing, power, and timing capabilities). The Martian communication network, along with DSN support, provides an initial architecture for simulation and analysis of MAPS, providing a notional deep space implementation.

A Low Earth Orbit demonstration mission concept is also being developed and analyzed. This mission scenario focuses on capturing the in-flight accuracy of available spacecraft clocks and demonstrating in-flight packet transmission and processing, among a limited number of assets. The simulation architecture allows for analysis of link budgets and estimated performance as a function of individual asset orbits and simulated errors. To capture the effects of real hardware, a hardware-in-the-loop system is being developed to integrate flight quality radio and clock hardware to capture receiver packet delays and clock uncertainty to directly model

spacecraft behavior. This system is designed to allow for high-accuracy timing measurements and delay modeling, through the use of high stability reference timing sources and software approaches to minimize latency. This architecture is being modularly developed to allow for a large number of independent spacecraft agents, and to support deep space architecture design.

This paper provides an overview of the MAPS architecture, from concept design and simulation to hardware-in-the-loop testing. Current hardware development plans for a cubesat-based in-space demonstration will be presented. MAPS provides a unique architecture for inter-spacecraft autonomous navigation capability, providing a software-centric approach to state estimation and providing several opportunities for future capability growth through technology infusion and alignment with published roadmaps.

ARCHITECTURE OVERVIEW

In order to provide a more detailed discussion of MAPS to foster insight into its technical capabilities and growth, a brief discussion follows. First, a summary of need and importance of deep space navigation is presented with a limited survey of the state of the art. Following sections focus on the initial formulation of the conceptual architecture, analysis results, and discussion of the next step in development and demonstration of the architecture.

Problem Formulation

With the continual advancement of spacecraft technology, missions venture farther out from Earth to our planetary neighbors, such as the Messenger mission to Mercury, and distant locales, such as the New Horizons mission to Pluto and Charon. As the missions and spacecraft have become more complex, the requirements on spacecraft navigation become more stringent.[1] Advances in navigation techniques also enable new missions.[2] The main performance drivers are a function of the spacecraft's trajectory from orbital entry and cruise, its onboard hardware capabilities, planetary orbit and entry error requirements, and scientific observation requirements.

Even with advanced celestial dynamics models and navigation measurements, it is not possible to perfectly predict spacecraft ephemeris. This is due to the complexity of the sys-tem being modeled, and assumptions used in the modeling of dynamic effects in deep space. There is also inherent error in predicting and propagating a spacecraft's trajectory. Additional effects are due to uncertainty in the dynamic models, physical spacecraft limitations, constrained onboard capability, and the measurement process itself.

Current navigation methods utilize a range of signals and observation techniques. Positioning techniques utilize ranging signals from Earth-based assets, onboard optical observations of planetary bodies, and internal tracking of the spacecraft's state via inertial measurements. The processing of large amounts of observation data through complex orbit de-termination algorithms provides state of the art state estimate and prediction. To meet the increasing demands on spacecraft positioning and the drive towards autonomous spacecraft, several methods are being investigated.[3] These techniques include performing Doppler and ranging between spacecraft in local orbits[4], moving optical navigation analysis methods onboard[5], utilization of observations of high energy pulsars[6], and integration of navigation measurements into communication signals[7].

The Multi-spacecraft Autonomous Positioning System (MAPS)

MAPS takes advantage of a solar system-wide network of space data relays, spacecraft, and ground assets to provide a navigation capability. This capability is designed to be scalable and expandable, with performance growing as the number of active agents increases and with the integration of dedicated high accuracy navigation nodes. The underlying communication systems that form the basis of this architecture have been studied in[8] and can be equated to the concept of an Interplanetary Internet[9,10]. The Interplanetary Internet envisions a network between planetary bodies and spacecraft similar to the architecture here on Earth, with local networks operating in geographic areas and high-bandwidth communication relays linking these various subnets.

The Martian Network[11] is a first step towards this concept and an inter-connected solar system. The primary functionality of spacecraft as data relays is implemented by the multiple orbiters in Martian orbit, Mars Odyssey, Mars Reconnaissance Orbiter, and MAVEN. In addition to providing detailed observations of the Martian atmosphere and geology, these spacecraft also serve roles as data collectors and forwarders for the Mars rovers Curiosity and Mars Science Laboratory. Development and implementation of a dedicated local data relay has also been considered and analyzed, such as for the Mars Telecom Orbiter[12].

This system is envisioned to take advantage of the increasing bandwidth of developing communications and increasing number of in-space data relays to provide an autonomous navigation capability. This uses the communications signals as observables in an online filter to provide updates to the spacecraft's estimated position. By embedding the required navigation directly

into the command and control packets, navigation fixes are generated as the spacecraft is processing the data. The data required can be programmed as a standard part of the navigation messages. The MAPS protocols integrate known state information at the transmission location into the communication packets that are being telemetered to the spacecraft in real-time. This reduces any reliance on human operators to process the data and the long passes currently required to generate sufficient data in order to perform high accuracy orbit determination on the ground.

The concept of operations is given below in Figure 1. As shown, MAPS takes advantage of a growing communication infrastructure to integrate a positioning capability into the individual data packets. This enhancement will allow for increased navigation precision and more frequent navigation updates. The core of the method is utilizing all assets in the infrastructure to perform tracking duties to each other. Whenever two spacecraft are in range to communicate and operations allows for a contact, the two assets attempt a contact. As part of this contact, the receiving satellite updates its position based on the navigation header received and the measured time delay. The primary navigation nodes will be the data relays that form the basis of the expanding complex network. As the individual spacecraft communicate with each other and provide positioning functionality, the overall navigation accuracy and knowledge of individual position is increased as updated references are propagated throughout the network.

The ideal initial deep space demonstration of MAPS will be in service as an augmentation to traditional ground-based radiometric tracking. These measurements (in addition to GPS and other measurements in LEO prior to Earth-orbit ejection) will still be used to initialize the spacecraft's state to high accuracy. Navigation packets would initially serve as support to traditional methods, with their primary benefit being to maintain spacecraft position knowledge accuracy while reducing the need for long tracking passes during cruise (where downlink data requirements are not as pressing) reducing operational cost. An additional driver for a deep-space mission is the larger allowable errors in timing measurement. These large distances exhibit longer timing delays and minimize the effect of small timing uncertainties in individual measurements.

The simulation results[7,13] demonstrate the capability of the MAPS architecture for a Mars transfer trajectory using publicly available SPICE MSL ephemeris. The use of communication-based navigation information relaxed the need for Deep Space Network state updates. As the duration between ground-based updates increased, the MAPS architecture demonstrated much smaller navigation error growth. The simulation result provided much insight to the feasibility and capability of

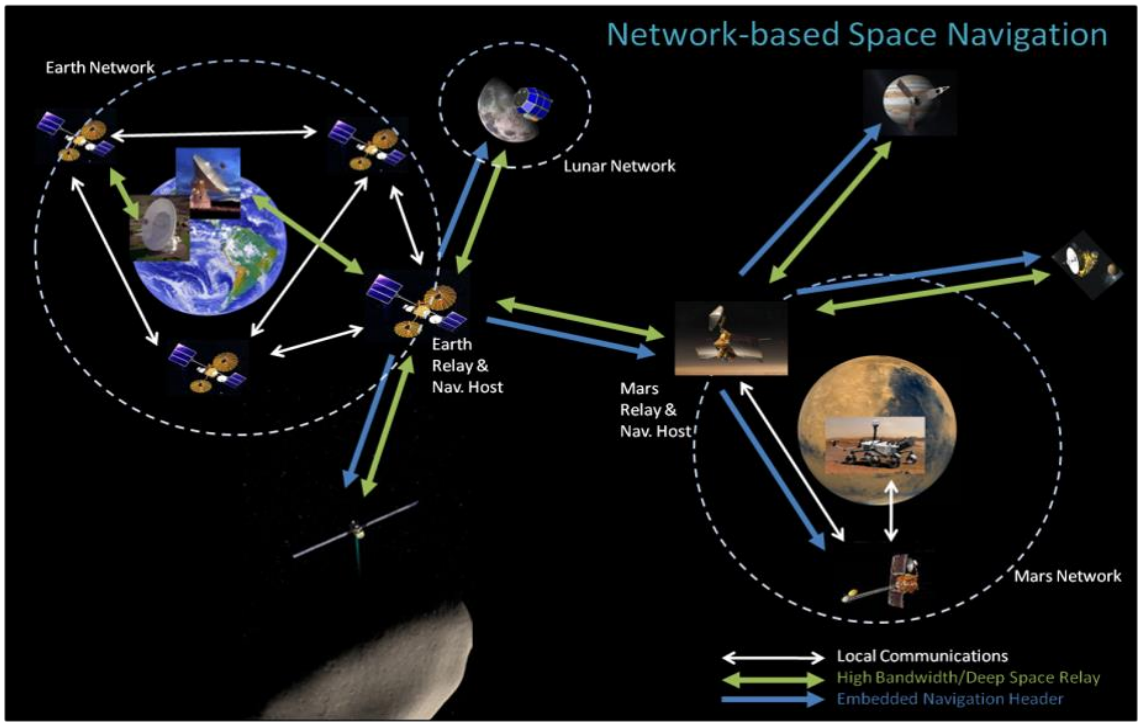

Figure 1: MAPS Architecture

this architecture even in a limited system.

Next step and current planning

In order to demonstrate the architecture's feasibility, it needs to be implemented on a notional flight-comparable computer, as well as flight-quality radios to capture the effects of processing latencies between signal reception and onboard processing and time-tagging. The primary observable of the architecture is embedded within the communication packets, particularly the timing delays between packet transmission and reception. These delays are difficult to exactly model in a laboratory, due to inherent latencies in processing, generation, and simulation of these delays to high precision during hardware validation.

With MAPS developed and explored via a modeling and simulation environment[14], the next step in development is to move towards an in-orbit demonstration of the technology. This is the primary path to maturation of the technology. An in-orbit demonstration is needed to validation the software simulation and analysis assumptions. Additionally, an in-orbit mission is required to capture the large distances and velocities involved in orbit, removing modeling uncertainties in packet travel times.

LEO DEMONSTRATION MISSION

As mentioned above, the ideal environment for testing of these algorithms would be an interplanetary cruise trajectory. While this was simulated for architecture evaluation, the opportunities for such a mission are very limited. Due to the software-centric nature of MAPS, the algorithms must be integrated early within the supporting cruise software design, in order to align with the software development and testing schedules. It can take years from mission proposal to flight on an interplanetary mission. An alternative path towards in-space demonstration is to focus on a limited implementation of the architecture in low Earth orbit (LEO). The availability of high-accuracy GPS observations provides both a best estimated trajectory as well as the capability to discipline the spacecraft's onboard oscillator, allowing for a high accuracy and highly stable timing reference. Once the architecture is shown to operate successfully in LEO, a deep space implementation can be developed which will require the use of highly stable onboard oscillators, such as an Ultra-Stable Oscillator or a Deep Space Atomic Clock.

Architecture and Spacecraft Design

A demonstration in LEO focuses on the utilization of and optimization of long-distance crosslinks between multiple spacecraft in orbit. There are two primary architecture selections which can be utilized for this study, dependent on funding and resource availability. The most direct implementation of the architecture involves placement of two assets in LEO. These two assets would then exercise cross-spacecraft communications and demonstrate the MAPS algorithms. The two spacecraft can be implemented on as a small a platform as a 3U cubesat, with identical hardware.

These can be deployed from the ISS to form a network for MAPS testing. Once deployed from ISS, the cubesats would employ differential drag techniques[15, 16] to establish and maintain the desired spacing between assets for MAPS testing. While use of drag will reduce the orbital lifetime of the assets, it allows for changing the initial orbital configuration of each after ISS deployment without requiring onboard propulsive capabilities

To increase the phasing between them, Cubesat 1 is oriented at a maximum drag attitude while Cubesat 2 is oriented to induce minimum drag. The maximum drag attitude has a long side of the cubesat (10x30 cm drag surface area) oriented perpendicular to the orbital velocity vector, while the minimum drag attitude has the long axis of the cubesat parallel to the velocity vector placing a small side (10x10 cm) in the ram direction. This results in lowering Cubesat 1's orbit more quickly than Cubesat 2, which in turn increases the inter-spacecraft drift rate. At some point, the orientations' of the two cubesats are switched so that Cubesat 1 has minimum drag while Cubesat 2 has maximum drag. The drift rate between the two spacecraft then begins to decrease and reaches zero once both cubesats are back in the same orbit, at which point they will maintain the same attitude to equalize the drag between them. Figure X shows the semi-major axis of each cubesat vs time since deployment for a given simulation run. Figure Y shows an example of how the inter-spacecraft distance evolves during and after this sequence if no further maintenance is performed.

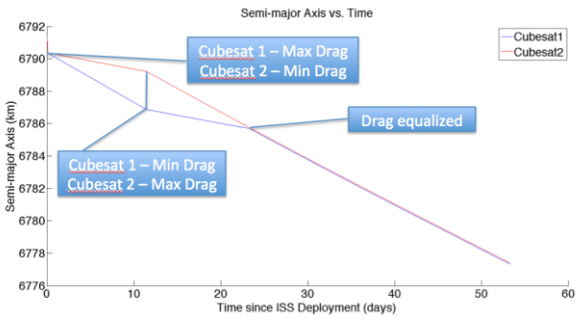

Figure 2: Spacecraft Drag Operational Sequence

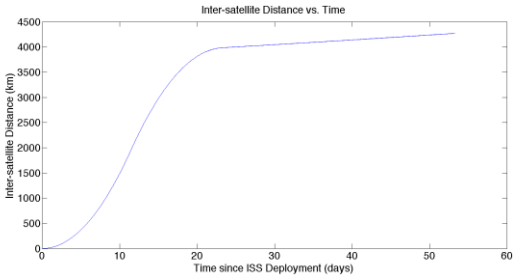

Figure 3: Inter-Spacecraft Distance

An alternate approach to demonstration of this hardware takes advantage of existing space platforms to act as navigation nodes. For this scenario, only one spacecraft would be launched into LEO. Over the course of operations, the ISS's SCaN SDR payload[17] would operate in place of one of the spacecraft. This scenario trades the cost and resources required to build an additional cubesat vs. integration and interfacing with an external payload.

Both scenarios would exercise a ground-based navigation node. This asset would operate as part of the ground station communication infrastructure. This will allow direct simulation of navigation packets being embedded in ground-based communications. The primary drawback on relying on Earth-based observations is the uncertainty due to timing delays induced by the dynamics of the Earth's upper atmosphere.

Concept of operations

Figure 4 captures the concept of operations for the orbital demonstration mission. Each of the assets will operate on a predefined schedule of communication passes between each other. The need for scheduling these ahead of time captures and emulates deep space scheduled communication passes. It also allows for the

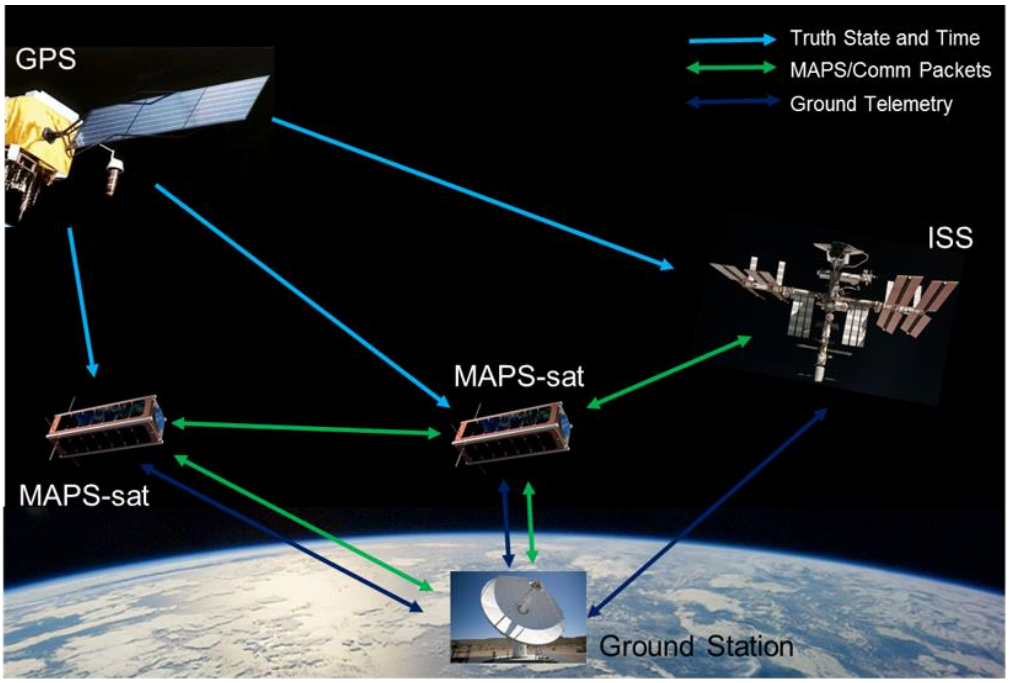

Figure 4: MAPS LEO Concept of Operations

spacecraft to ensure they have adequate pointing between assets to achieve a closed communication link, and support early ground support and checkout. Due to the pointing accuracy required to receive adequate power and signal to noise levels to enable link closure initial coordination between assets will consist of a pre-planned schedule of pointing commands. As the on-orbit demonstration continues, scheduling of passes will be eased off, allowing the spacecraft to exercise onboard algorithms for attempting cross-spacecraft links.

Onboard GPS receivers will initially discipline the local oscillators using onboard software, enabling high-fidelity timing information. After initial validation, the spacecraft will disable clock disciplining, allowing for performance comparisons under a worst case scenario.

Spacecraft Design

To support this architecture design, an initial down-selection of COTS cubesat components have been chosen to form the basis of the orbital experimental platform. The primary requirements of the spacecraft are to exercise long-distance line-of-sight communications with other assets, both on the ground and in orbit, observe the true state of the vehicle to support post-processed high fidelity orbit determination, and maintain knowledge and control of attitude in order to achieve cross-asset links.

To minimize development time, the platform components have been chosen to be close-to or flight-proven components. During preliminary investigations, the components in Table 1 were selected for the MAPS cubesat architecture. These were chosen based upon prior experiences with the sensors, current integration efforts, and commercial availability in order to minimize time to flight. The power requirements for these components are being tracked and will be fed into detailed power system sizing of batteries and solar panels.

Table 1: Initial Component Selection

Component Description	Manufacturer
Flight Computer Q7; 17 krad tolerance	Xiphos Technologies
S-Band Radios; SWIFT-SLX Transceiver	Tethers Unlimited
3G Flex EPS	Clyde-Space
Digital Fine Sun Sensor/SS-411	SSBV
SSBV Magnetometer	SSBV
GPS12-V1	SpaceQuest

The volume and mass of the potential components was also captured to feed into vehicle sizing and component placement within the bus. Initial layouts showed that with these selected components, a 3-unit (3U) cubesat will be sufficient. A notional design is given in Figure 5. This sizing assumed an onboard 30Watt-hour power source. As part of the solar panel sizing, a detailed power analysis will provide input to the size constraints of the chosen electronics architecture.

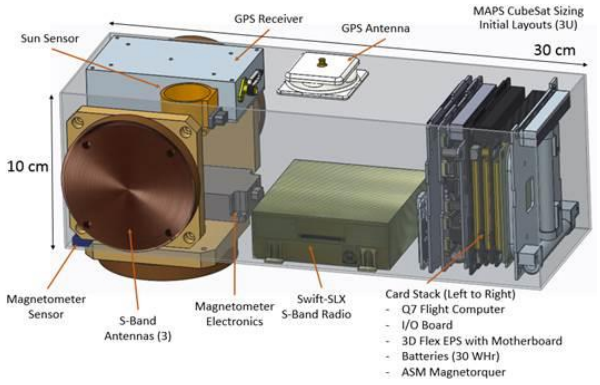

Figure 5: Initial Component Layout for 3U Cubesat

HARDWARE-IN-THE-LOOP DESIGN

To capture performance of the algorithm with actual spacecraft components, a hardware-in-the-loop (HIL) system is being designed and implemented to characterize flight components and integrated flight software systems. In addition to providing architecture performance analysis on an integrated hardware system, it will also allow for verification of the software simulation results.

This HIL architecture is being designed around 3 primary elements: a simulation coordinator to track the truth state of the simulation agents, a timing coordinator to maintain and control timing delays within the active simulation scenario, and a series of independent flat satellites (or flatsats) to simulate the hardware and software of an orbital or ground asset. Figure 6 provides an overview of the as-designed architecture, as well as the primary functions of each component. The various agents are connected to the simulation coordinator via an Ethernet bus to allow for high-rate data transmission for sensor emulation, truth inputs, as well as high-rate performance data collection and flatsat status. Each of these assets, their implementation, and functionality are addressed in the following sections.

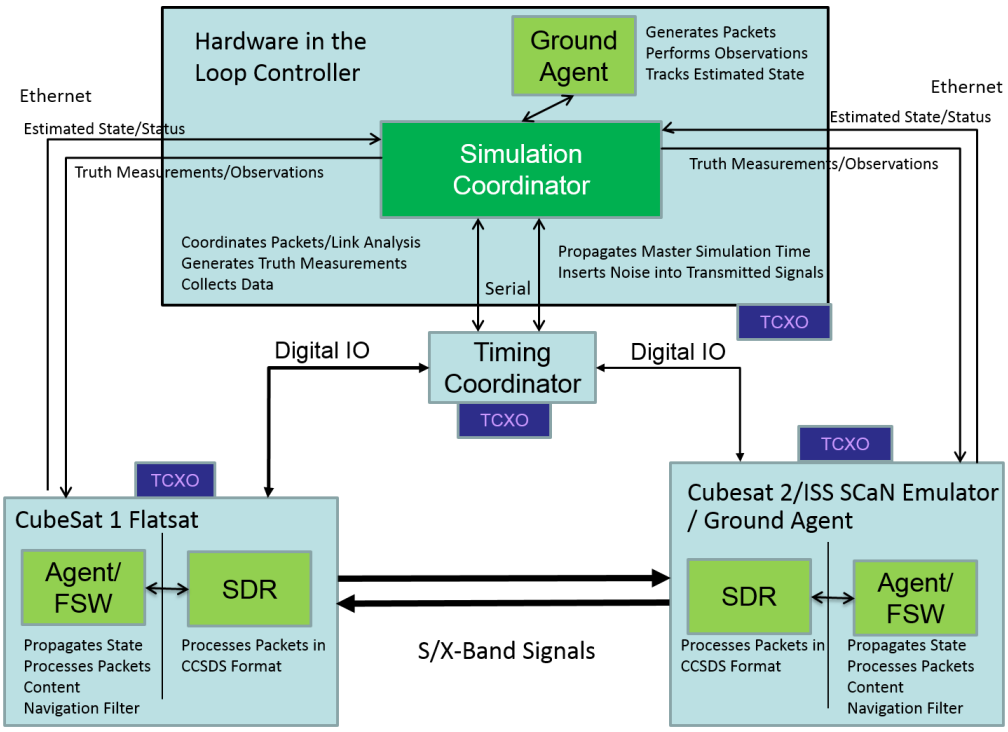

Figure 6: Conceptual HIL Architecture

Timing Sources

Due to the sensitivity of the algorithms to the onboard timing solution, the architecture is being designed with the capability to include a variety of clock inputs primarily via a 1 pulse-per-second (PPS) signal. This allows direct emulation of the same processes that are used on an orbital platform, especially deep space missions, where a high stability oscillator is used to maintain onboard timing. These 1PPS inputs discipline the local clock driver through the use of Network Time Protocol (NTP) software onboard each flatsat. These drivers use the input signal to generate an interrupt at the kernel-level to discipline the onboard clock, by helping to maintain clock stability and reduce clock drift.

Several clock sources will be used to capture the sensitivities of the MAPS algorithms to the clock references. These range from the built-in crystal oscillator included on the flight boards, to an XPRO Rubidium reference, and a Chip Scale Atomic Clock (CSAC). Although space-qualified components are not being used, these components are sufficient for testing of algorithms. These clock sources will also serve to verify the timing stability models being used in the software simulation. The inputs of the various timing sources and their integration is shown in Figure 7.

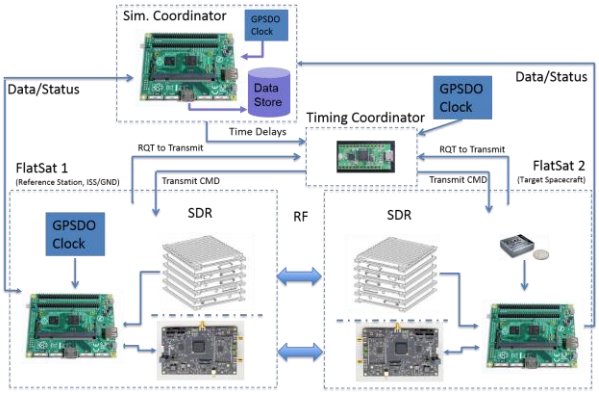

Figure 7: Implemented HIL Architecture

To capture best-case timing observations, GPS is also being interfaced with the NTP software to allow for both tracking of the time offset (to correct for time bias), as well as a disciplined PPS signal (to correct for frequency jitter). By including GPS as a potential timing source for the LEO demonstration mission, the MAPS architecture can first be exercised with these high accuracy observations, minimizing the sensitivity of the onboard estimation algorithms to timing uncertainty. As the algorithms are tested in orbit, the spacecraft will be reconfigured to use alternate PPS signals (i.e. remove GPS NMEA timing outputs or other onboard oscillators).

The clocks are included to allow for direct simulation of the short time frame drift and uncertainty over an inter-spacecraft communication pass. As such, the simulation will run in real-time during such period. To enable long-term dynamics simulation over multiple orbits (and to enable deep space mission analysis), time jumps will be needed to allow the assets to fast-forward through time. This can be accomplished by either increasing the dt of the simulation or by an external command to allow the assets to re-sync to an updated time.

Flatsats Implementation

The Xiphos Q7 platform identified above was chosen to serve as the flight processor due to its flight heritage and performance capabilities. These are needed to meet the demands of running multiple complex 6 or 8 state estimation filters (at either 1 Hz or 10Hz). A flight development unit of the planned platform is expected to be available for testing and integration in the current fiscal year. In order to match the capability of this system, several constraints were used to limit the flight processor emulators in use for initial testing and integration. These include requiring a platform with similar processing speed, memory, and platform.

Due to their low cost, flexibility, and similar specs, the Raspberry Pi platform was selected for the architecture. A photograph of the implemented architecture is shown in Figure 8, mirroring the implementation given in Figure 7. The team selected to use the Compute Modules for each spacecraft due to the potential for building a spacecraft platform around the module, and proceeding directly to flight development. All of the Raspberry Pi boards feature a good selection of ports and digital input/outputs to allow for simulation timing input, coordinator, and sensor integration.

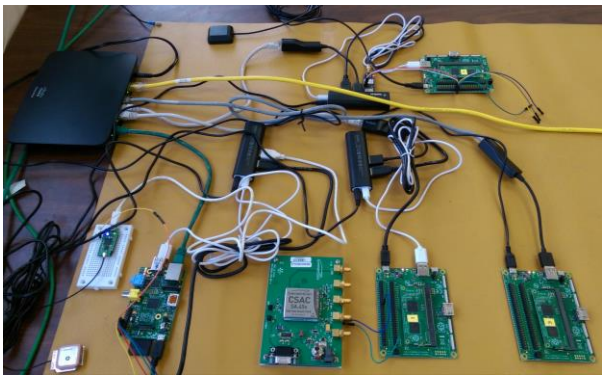

Figure 8: Initial HIL Testbed

To allow for integration of GPS-disciplined clock sources, several small receivers were selected to provide 1PPS reference data. Several platforms were selected for integration and will be compared in terms of timing stability. These includes both low-cost component such as the NavSpark (NS-T and standard versions), as well as a U-Blox NEO6-T module. Higher-end components such as a JAVAD TR-G2 and Novatel OEMV will also be used for comparison.

Timing Coordinator Development

The primary function of the timing coordinator is to apply and enforce the light travel times associated with packet transmissions. Due to the small distances involved in the demonstration architecture (as compared to the much larger distances for a deep space mission), the timing stability and latencies are extremely important. With the transmission times often being on the order of milliseconds, the latency effects will drive the accuracy of the simulation much more than the short term clock stability. As such, a platform was selected to provide high-accuracy hardware timers, with minimum overhead. To meet these requirements, the Arduino Due platform was selected. This platform consists of an ARM Cortex-M3 processor operating at 84Mhz. The Due contains 9 onboard hardware timers, and DMA drivers, allowing for high accuracy timing capability. To further limit the effect of latency, the processing of this platform is interrupt-based with minimal additional processing.

HIL SOFTWARE IMPLEMENTATION

Software-Based Simulation Tools

In addition to development of a spacecraft hardware emulation stack, software is needed to perform the actual simulation, capturing the dynamics of the assets, perform communications link analysis, provide inputs to various as-flown software models, as well as collection of post-flight simulation data for error analysis and visualization. To provide enhanced functionality within the software tools, standard software libraries are being utilized. These includes the Boost numerical analysis library for matrix inversions and navigation filter implementation[18], SPICE library for loading planetary and spacecraft ephemeris[19], the libSPRITE library for flight software generation, and the NTP servers for clock disciplining with external PPS sources[20]. The primary components of the software being developed include a suite of hardware emulators, and flight software surrogates.

libSPRITE Overview

libSPRITE is a NASA developed open-source platform for hosting flight software and simulations. The libSPRITE platform includes several libraries coded in C++. The most important for our purposes is the Simple RunTime eXectutive (SRTX). This library provides

task scheduling and data routing via a deterministic publish/subscribe (pub/sub) messaging system. Tasks are spawned and managed as threads from a user defined application. This system automatically assigns tasks to cores on multi-core computer systems. All tasks within a rategroup, i.e., running at the same periodic frequency, are run sequentially in priority order. This implementation ensures predictable and repeatable behavior from run to run, even when moving the software to a machine with a different number of processor cores. The deterministic pub/sub system is a unique feature. It guarantees predictable and repeatable delivery of messages between periodic tasks; even when the tasks are running at different frequencies.

The libSPRITE platform also includes the SRTX Configurability and Adaptability Language Extensions (SCALE) which allows developers to configure which tasks are run for a given session, their period, priority, etc. The exact same executable can be run with different configuration scripts to achieve different behavior. This flexibility is useful for running varying test cases, incremental development, or operating with different hardware configurations. Using SCALE, developers can also interact with the application while it is running though a debug console using the Lua scripting language. Tasks can be stopped and restarted, queried, and have parameters set from this console. The developer can expose however much or little functionality as they wish to the Lua console by defining the desired language bindings. Furthermore, SCALE allows Lua scripts to be run from within a SRTX task if one wishes to do so. This mix of compiled code paired with scripting gives developers tremendous flexibility.

Simulation Coordinator Software

The primary execution of the simulation and the coordination between the various hardware and software nodes is controlled by the coordinator. This piece of software both tracks the truth state of the assets in the simulation, stores data, and generates the data to feed into controlling the queuing and release of communication packets. An overview of the simulation coordinator inputs and outputs is shown in Figure 9.

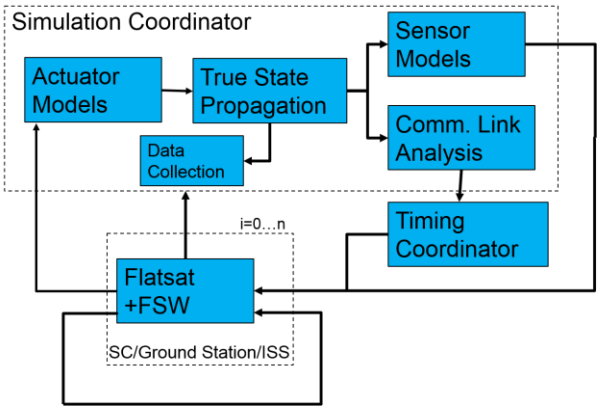

Figure 9: Coordinator Data Flow

A summary of the tasks that are included in the Simulation Coordinator are listed in Table 2. Each of these are implemented as a task within libSPRITE. These are currently set to operate at 10Hz, providing truth reference data at the next simulation tick. A brief discussion of the primary functionality of each task is given below.

Table 2: Simulation Coordinator Tasks

Task	Functionality
Clock	Tracks simulation truth time and generates simulation coordination tick.
Trajectory	Tracks truth translational state of spacecraft
GPS Driver	Provides truth data to GPS emulators
Attitude	Tracks true attitude of spacecraft
Communications	Checks for potential communication links between spacecraft and calculates light travel times
Mission Analysis	Calculates parameters to support vehicle sizing (i.e. power generation, and usage)
Data Collection	Stores truth data of simulation and reported data from assets to local database

Clock: The Clock task tracks the truth time in the simulation and acts as the master clock. In order to maximize clock stability, the timing is driven by a kernel-based timer, utilizing standard NTP algorithms with an external 1PPS GPSD reference signal. This task also captures the time in reference to the simulation epoch. The epoch time is needed to determine the truth states of the assets and state propagation via the dynamics planet and sensor models. This allows for setting the offset time to a future or past time. In addition to tracking the simulation truth time, this task also generates a 1Hz PPS signal that is sent to all nodes in the simulation. This signal serves to coordinate the sensor models as well as provide a reference to measure each asset's timing errors against.

Trajectory: The trajectory task tracks the true position and velocity of each spacecraft as a function of time. This task is currently implemented as an ephemeris look-up, although it can be easily expanded to include gravity models and state propagation. This data is used as an input to the attitude task as well as the GPS driver.

GPS Driver: This task provides the interface to the implemented GPS hardware emulators, which are discussed in the following section. The primary goal of this task is to provide a spacecraft's truth information to a specific asset's emulator.

Attitude: This task receives the actuator commands sent from each node (for simulations without attitude analysis, simply the commanded attitude), and translational state of the node to calculate the rotational forces acting on the nodes and propagates the spacecraft attitude. This updated attitude information is provided to the attitude sensor emulators (and driver software) to aid in measurement generation.

Communications: The communications task determines if there are any potential links within the assets in the simulation. By tracking the attributes of each node's communication system(s) (type and frequency, gain, losses, etc.), the power and signal-to-noise ratios are computed for every combination of asset communication systems. For any links with adequate power and SNR, the light travel time is also calculated. The properties of these links, from, to, and travel time, are tabulated and forwarded to the timing coordinator.

Mission Analysis: This task is used to support the design of the spacecraft's auxiliary systems, in particular the power and thermal systems. By tracking the true attitude and position of each node, this function can capture the thermal flux onto the spacecraft and its associated solar panels to determine power generation. In addition this task is designed to track power usage, to provide feedback into vehicle sizing.

Timing Coordinator

The timing coordinator exists as a separate piece of hardware within the simulation architecture. The primary function of the timing coordinator is to track and implement the delays in packet transmission between assets. In order to minimize latencies, it is desired to minimize overhead of this asset by getting as close to bare metal programming as possible. Over the course of a simulation, the coordinator provides a list of potential communication links and their respective timing delays as a function of time. This data is updated every iteration of the communications function, and is fed to the timing coordinator. On a simulation sync pulse, this updated timing information is utilized.

The timing function tracks the current and last packet delay times in order to provide a 1st order interpolation as a function of the time a transmission request was made. The time of packet request is determined by a set of input interrupts, generated by each asset's radio driver. On interrupt, the timing coordinator calculate the packet delay, and starts an onboard high resolution timer. At the timer's completion an interrupt is generated and an external interrupt is sent to flatsat via its radio driver to physically transmit the packet over RF to the intended recipient.

Hardware Emulators

In order to support the development of drivers to the various hardware used as part of the demonstration spacecraft, a series of hardware emulators are being developed. These are in development for the onboard attitude and position sensors as well as actuators. The inputs to these are provided by the simulation coordinator, which tracks the truth state of each agent during the simulation. The emulators act as direct surrogates in place of actual hardware, simulating both the behavior and the communication protocols.

In the HIL simulation each flatsat will have a GPS Emulator that will create appropriate GPS NMEA and other manufacturer-specific sentences for consumption by the flight software. The Emulator receives the latest state and trajectory data for its flatsat from the Sim Coordinator. This data is parsed and output sentences are created and queued for transmission along the appropriate communication standard (i.e. serial TTL) in order to emulate the sensor's electrical interface.

For the purposes of the HIL simulation, the GPS Emulator executes on the flatsat, but separate from the flight software. This implementation helps to reduce latency between when the Emulator transmits sentences and when they are received and parsed by the flight software. The actual time of transmission of sentences by the GPS Emulator is controlled by a GPIO-pin interrupt sent by the Sim Coordinator. When this interrupt is triggered, the Emulator will transmit the previously created GPS sentences via a virtual serial port to the flight software. This implementation allows for more precise timing and the ability to introduce specific delays for the purposes of testing.

As the architecture simulation expands to include detailed attitude dynamics and estimation, further hardware emulators will be utilized. These will focus on implementation of models that capture the interfaces and functionality of common cubesat actuators and

sensors. This includes a digital sun sensor model and magnetometer. The primary actuator emulators being develop is for cubesat-sized magnetic torquers. Due to the modular nature of the HIL framework, future inclusion of micro star trackers and reaction wheels can be easily accomplished. This architecture can also be easily expanded to star field or sun simulators to capture full HIL capability and simulation.

Flatsat Software

The flatsats provide a mock satellite flight software running on representative hardware. The flatsat software includes expected functionality such as timekeeping, mission management, telemetry, and Attitude Determination and Control (ADAC). The software interfaces to hardware sensors and effectors are present but in place of actual sensors and effectors are the emulators, driven by the simulation coordinator's dynamics models. Table 3 provides a summary of the designed tasks. The MAPS algorithms run on the flatsats just as they would on an actual satellite.

Table 3: Flight Software Tasks

Task	Functionality
SC Clock	Tracks processor time and tracks offset to simulation time
MAPS Navigation	Uses MAPS algorithm to estimate current position, velocity, and time
Ops Navigations	Uses GPS to track best estimated state of the vehicle (operational state)
Attitude	Tracks current attitude of spacecraft
Estimation	Propagates state of other assets in network
Pointing	Autonomous pass attempt algorithms, and commands attitude of vehicle to meet operational constraints
Control	Generates outputs to actuators to maintain commanded attitude
Telemetry	Generates MAPS-compatible packets during communication passes, and stores operational data to downlink
Logger	Logs state of vehicle at high-rate and provides real-time data to coordinator
Command and Data Handling	Ingests, validates, and distributes commands from a ground system as well as telemetry data from other MAPS nodes.

SC (Spacecraft) Clock task – Provides a coordinated system time referenced to a common epoch for the MAPS network (currently GPS epoch). Also provides a free-running "time since boot" for tasks that do not require "wall clock" time. The SC clock system time reference is disciplined though the MAPS network. For simulation purposes, the time is referenced to system time. This is especially tricky when simulation time is accelerated or takes a leap forward, but these capabilities are necessary to support long duration mission studies.

MAPS Navigation – This task provides the estimated position/velocity vectors using onboard sensors and corrected via the MAPS network using the MAPS navigation estimation algorithms.

Ops Navigation – This task computes the best estimate of position/velocity state with onboard sensors but without MAPS navigation updates. This state can be considered the reference state; the state estimate we expect to see using current state of the art navigation.

Attitude – Computes the spacecraft attitude. This is performed separate from the position/velocity state since those values are estimated using two different mechanisms.

Estimation – The Estimation task is responsible for propagating the current state estimate of each MAPS asset and publishing those estimates for consumption by other flight software tasks. State propagation is facilitated by using gravitational models from NAIF SPICE-provided kernels. Onboard estimation of "other node" state is required to be able to point towards other nodes during inter-node communication passes.

Pointing – The Pointing task determines which asset the spacecraft will attempt to communicate with and generates the required attitude command quaternion. The primary inputs to this task are the estimated states of the other assets provided by the Estimation task. The Pointing task selects the desired asset to communicate with based on communication link quality (power, SNR, etc.), time since last communication with each other asset, and any previously scheduled communication passes. After selecting the desired asset, the required pointing attitude is determined and output to the Control flight software task.

Control – The control tasks commands spacecraft effectors to effect the necessary pointing.

Telemetry – The telemetry task both queues low-rate spacecraft data and generates the primary packet content to the radio driver. This driver then generates the MAPS header, and formats the packet into the correct CCSDS formatting prior to transmission. When in contact with a ground station, this task also downlinks the stored flight telemetry data from the internal queue. This task includes the capability to test onboard algorithms and dispersed network transmissions across relays to study telemetry scheduling and bandwidth.

Logger – The logger task records and provides high-rate data back to the simulation coordinator. This data is separate from ordinary telemetry data. It is used for more detailed analysis and visualization of system performance and operational status.

Command and Data Handling – The spacecraft flight software must be able to receive some commands. It must also receive and process MAPS packets from other nodes in the system. This task handles the receive side of the communications system and distributes data to the appropriate tasks within the flight software.

Each flatsat includes a series of hardware drivers that for communication and processing of data from attached sensor and actuator emulators. These include drivers for generating outgoing packets and transmitting telemetry, processing data transmitted from the GPS emulator, as well as communication with attitude sensors and actuators.

Together these tasks and drivers form the basis of the flight software build for a MAPS demonstration mission. By including the emulation of actual hardware platforms and sensors to match an operational configuration, the inherent sensor delays and processing algorithms can be tested. These algorithms can also be implemented early in the development, feeding into hardware selections and software architecture design. The use of HIL analysis allows for a large amount of confidence in the developed software. With a tested and operational build, hardware selection and integration can be accelerated, reducing time to flight.

HIL CHARACTERIZATION

Upon implementation of the HIL architecture, the first vital task is to perform characterization of the interfaces alone, to identify the hardware latencies inherent in the architecture. The primary areas of characterization are system response latency, time delay accuracy (via timing coordinator), and clock stability. As the environment is still in development and integration testing as of this writing, limited results are available. The results of these initial capabilites are given below, as well as a description of the planned characterization tests.

Initial Clock Accuracy Results

Due to the dependence of the MAPS algorithms on tracking time, the first characterization test focuses on capturing the timing stability of several oscillators under a ground-operations scenario. This captures the accuracy of the clock under stable dynamics, while allowing a true error comparison to the global high-accuracy NTP time servers. This provides a baseline measurement of timing stability of the systems under test.

The initial testing sequence used the planned host platform with the oscillators tied to digital inputs to provide 1PPS signals. NTP servers compiled to allow operation at ATOM-level provided the error calculations and kernel-level clock disciplining capability. Four nodes were utilized to test a variety of parameters. The timing inputs to the nodes are given below in Table 4. Each of the nodes was running the same build of NTP (4.2.8.p1) and Debian Linux (Kernel 3.18.10+ non-RT). Each node was also attached to an external network to allow for access to the NTP global time servers for error characterization and time tracking.

Table 4: Timing Test Setup

Node	Timing Inputs
Node 0	NavSpark (GPS + 1PPS)
Node 1	CSAC (1PPS)
Node 2	UBlox NEA-6T (GPS + 1PPS)
Node 3	No external timing inputs (NTP only)

Timing information was initially captured via NTP's built-in logging capability, utilizing peerstats and loopstats files to capture local and comparative performance. The dynamics of the latency was captured for each system, as well as histograms of data over the observational period. Data was collected over four days. The results of this series of tests is shown in Figures 10 to 13.

These tests are planned to be repeated for additional clock sources (such as IMU-based high accuracy PPS, and a Rubidium oscillator).These tests will also be repeated with a full flight software stack in operation (at high load) in comparison to the limited load results shown below. Upon full simulation integration, the test will be repeated by synchronizing all nodes to a central local source, and track clock errors over the course of the simulation. This will capture the expected timing performance in the true environment. To support this mode, the simulation coordinator will also load synchronized timing data into each GPS emulator to simulate real-time timing messages over operation.

Additional Planned Testing

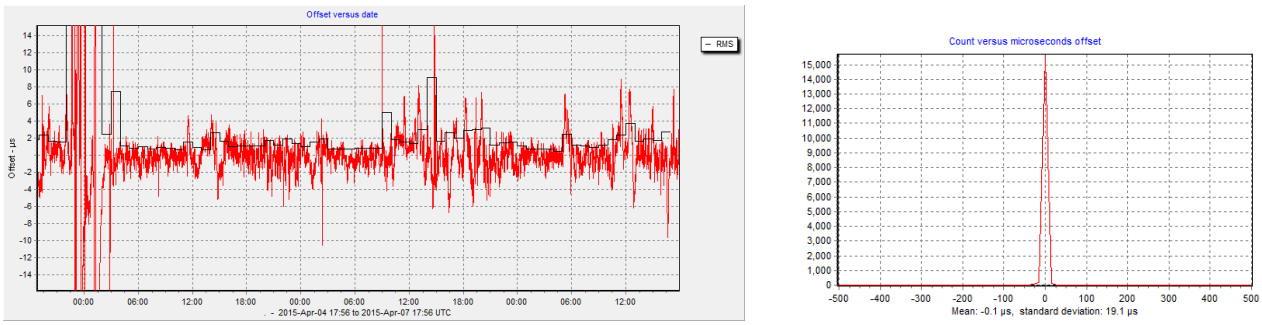

Figure 10: Node 0 Performance

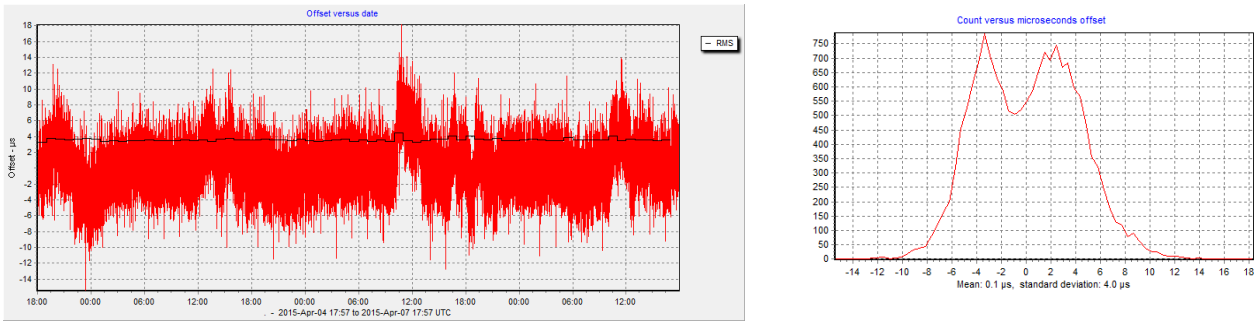

Figure 11: Node 1 Performance

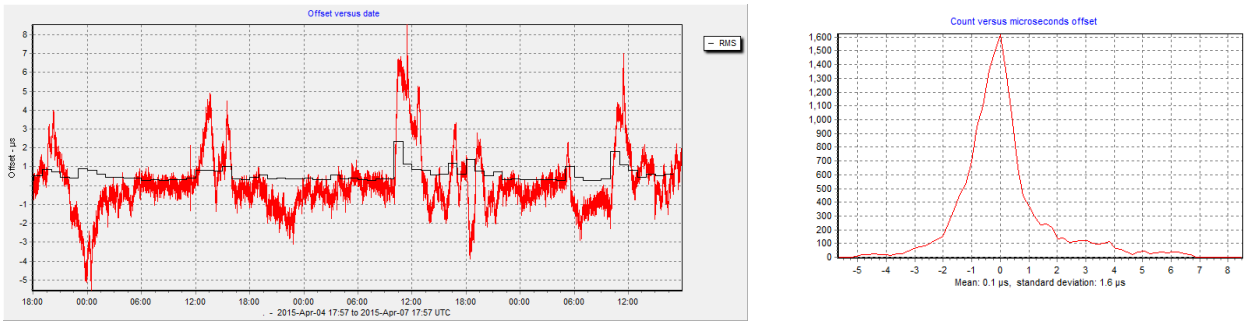

Figure 12: Node 2 Performance

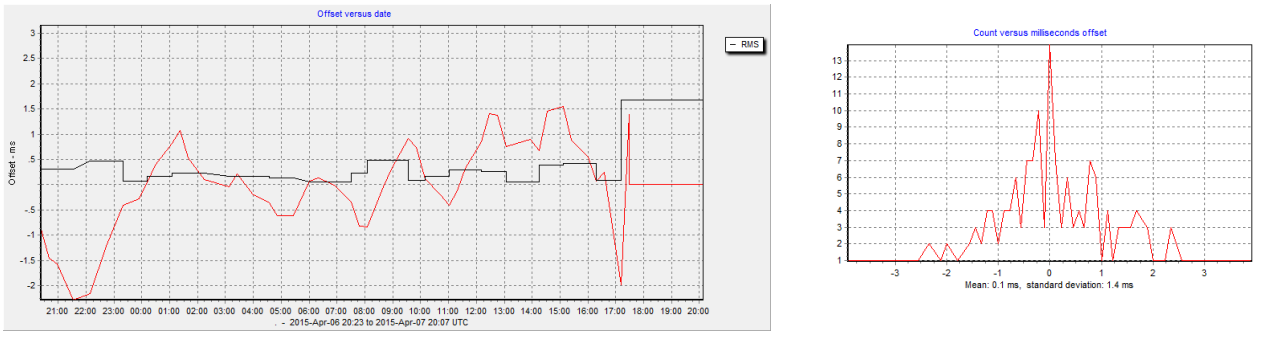

Figure 13: Node 3 Performance

A larger set of integration tests are planned to occur as the HIL architecture is finalized and moves towards integration for a full simulation for the LEO demonstration. The first series of tests will explore the need for a real-time operating system and its impact to timing stability. This will be completed through utilization of a custom Linux kernel with Pre-Empt RT[21] patch applied. These tests will analyze the

capabilities of the system in a manner similar to Brown and Martin[22], except for Raspberry Pi and Xiphos platforms with the inclusions of high stability external oscillators.

The latency of the architecture will be characterized at multiple levels. These simulation scenarios will capture latencies at the hardware driver, software task, and timing manager levels. This information will feed into correcting the time delay operations to correct for known architecture latency, as opposed to internal radio hardware latency. This test will be repeated for RT and non-RT Linux kernels, as well as for kernel and software level interrupts. This system architecture latency characterization will form the topic of an additional paper.

FORWARD DEVELOPMENT

The primary objective of the work at present is to complete the implementation and testing of the hardware in the loop environment. With an operational environment, it will be possible to further test, optimize, and demonstrate the MAPS architecture. These results will provide validation of the software simulation and iterate with mission sizing routines in order to develop a demonstration mission. Additional applications of the MAPS technology are being considered, including application to planetary surface navigation as well as LEO capability (as well as the inclusion of traditional ranging codes into the architecture)

CONCLUSIONS

This research expands the MAPS concept from architecture and software-based simulation and development to testing and integration on a hardware-based platform. This HIL testing allows for verification of simulation results using spacecraft-capable hardware to capture real system latency and timing effects on a notional spacecraft bus. The inclusion of a suite of flight software tasks allows for full emulation of a cubesat's onboard capabilities, computationally and functionally. With the completion of implementation and characterization of this architecture, continued developments can be made to support MAPS algorithms and support libraries to enable and drive implementation across a variety of mission and payloads to both demonstrate and implement a solar-system wide autonomous positioning system.

Acknowledgments

This work was made possible through funding from NASA/Space Technology Mission Directorate as a Futures task within the Game Changing Development program. The MAPS concept would not be growing and maturing without the support and help of Stacy Cook, Mike LaPointe, and Kevin Kempton. Our team is also very grateful for the exceptional guidance and backing of Dr. Carrie Olsen and Jeff Morton, as well as all of our department management and support.

References

1. Nelson, R. A., Key Issues for Navigation and Time Dissemination in NASA's Space Exploration Program. NASA, 2006.
2. Rush, J., Isreal, D., and Ramos, C., DRAFT Communication and Navigation Systems Roadmap, Technology Area 05. National Aeronautics and Space Administration, November 2010.
3. Christian, J. A. and Lightsey, E. G., "Review of Options for Autonomous Cislunar Navigation," Journal of Spacecraft and Rockets, vol. 46, September-October 2009.
4. Lightsey, E. G., Mogensen, A. E., Burkhart, P. D., Ely, T. A., and Duncan, C., "Real-time Navigation for Mars Mis-sions Using the Mars Network," Journal of Spacecraft and Rockets, vol. 45, May-June 2008.
5. Riedel, J., Bhaskaran, S., Eldred, D. B., Gaskell, R. A., Grasso, C. A., Kennedy, B., Kubitscheck, D., Mastrodemos, N., Synnott, S. P., Vaughan, A., and Werner, R. A., "Autonav Mark3: Engineering the Next Generation of Autonomous Onboard Navigation and Guidance," 2006.
6. Hanson, J. E., Principles of X-ray Navigation. PhD thesis, Stanford University, March 1996.
7. Anzalone, E. Agent and Model-Based Simulation Framework for Deep Space Navigation Design and Analysis, PhD thesis, Georgia Institute of Technology, August 2013
8. Scier, J.S., Rush, J.J., Williams, W.D., and Vrotsos, P., "Space Communication Architecture Supporting Exploration and Science: Plans and Studies for 2010-2030," 1st Space Exploration Conference¸ AIAA, Washginton, D.C., January 2005, AIAA-2005-2517.
9. Cerf, V. et al., "Delay-Tolerant Network Architecture: The Evolving Interplanetary Internet," Interplanetary Network Research Group, Internet Research Task Force, August 2002.
10. Burleigh, S. et al. "The Interplanetary Internet: A Communications Infrastructure for Mars Exploration," 53rd International Astronautical Congress, International Astronautical Federation, October 2002.

11. Lightsety, E.G, Mogensen, A.E. et al. “Real-Time Navigation for Mars Missions Using the Mars Network,” Journal of Spacecraft and Rockets, AIAA, Washington, D.C., Vol. 45, No. 3, May-June, 2008.

12. Franklin, S. F., John P. Slonski, J., Kerridge, S., Noreen, G., Riedel, J. E., Komarek, T., Stosic, D., Racho, C., Edwards, B., and Boroson, D., "The 2009 Mars Telecom Orbiter Mission," IEEE AC, October 2004.

13. Anzalone, E. J, and Chuang, J.C. H. "Conceptual Design of a Communication-Based Deep Space Navigation Network," AIAA Space 2012 Conference and Exposition, Pasadena, CA, September, 2012.

14. Anzalone, E. J., "Unified Simulation and Analysis Framework for Deep Space Navigation Design", Guidance, Navigation, and Control Conference. Volume 151, Advances in the Astronautical Sciences, Ed. May, A. Breckenridge, CO, February 2014.

15. Leonard et al., "Orbital Formationkeeping With Differential Drag", Journal of Guidance, Control, and Dynamics, Vol. 12, No. 1 (1989), pp. 108-113

16. Gengestad et al. "Operation, Orbit Determination, and Formation Contorl of the AeroCube-4 Cubesats", SSC13-X-4, Small Satellite Conference, 2013

17. Reinhart, R.C. et al. "Development of NASA' Space Communications and Navigation Test Bed aboard ISS to Investigate SDR, On-board Networking, and Navigation Techniques", ReSpace Conference, Albuquerque, NM, November, 2010.

18. http://www.boost.org

19. Acton, C., "Ancillary data services of nasa's navigation and ancillary information facility," Planetary and Space Science, vol. 44, no. 1, pp. 65-70, 1996.

20. Mills, David. "Internet Time Synchronization: The Network Time Protocol", IEEE Transactions on Communications, Vol. 39, No. 10, October 1991.

21. The real-time linux wiki. https://rt.wiki.kernel.org/.

22. Brown, J. and Martin, B. "How fast is fast enough? Choosing between Xenomai and Linux for real-time application", Twelfth Real-Time Linux Workshop, Nairobi, Kenya, October 25-27, 2010.