Development of Tactical and Strategic Operations Software for NASA's Lunar Flashlight Mission

Mason Starr
Graduate Research Assistant
Lunar Flashlight Operations Lead
Georgia Institute of Technology
766 Piedmont Ave NE, Atlanta GA 3038
mstarr32@gatech.edu

Faculty Advisor: Glenn Lightsey
Georgia Institute of Technology
glenn.lightsey@gatech.edu

ABSTRACT

Lunar Flashlight (LF) is an interplanetary CubeSat mission designed to demonstrate the use of a novel green monopropellant propulsion system and characterize lunar surface ice with a near-infrared laser array and reflectometer. LF is also the first Jet Propulsion Laboratory (JPL) mission to be operated entirely by students. While JPL provided baseline tools to Georgia Tech (GT), bespoke tools and software were developed by GT operators. Four tools developed by the author are discussed in this paper: (1) Downlink Helper is a Graphical User Interface (GUI) tool which improves the tactical downlink of recorded spacecraft telemetry. The tool automatically creates and sends downlink commands, displays an intuitive representation of telemetry onboard and downlinked from the spacecraft, and aids operator decision making with predicted downlink times for onboard files. (2) The SeqGen tool suite uses a Python-based object-oriented class structure to parse, generate, and manipulate LF command sequences from minimal input parameters. SeqGen pulls from a database of modular components, performs calculations to insert command parameters, and automatically version controls and archives sequences. SeqGen classes are flexible and are easily ported into other tools and applications, such as the Linter. (3) The Linter is a command line tool that parses LF command sequences and checks them against a database of mission flight rules. Flight rule violations and warnings are automatically detected and displayed for the operator. (4) SMARTS is a GUI tool that enables operator-in-the-loop propulsive burns on LF's highly anomalous propulsion system. Thruster performance is variable and unpredictable, preventing deterministic command sequences from being used to fire the thrusters, and threatening to saturate LF's reaction wheels. To manage spacecraft momentum, the spacecraft is rotated about a thruster's force vector while firing. SMARTS enables operators to tactically calculate, queue, and send command modules such that they execute onboard at precise phases in the rotation. Lessons learned from the development process are condensed and can be used to inform the operations of other student-led interplanetary small satellite missions.

INTRODUCTION

Mission Objectives and Concept of Operations

Theme 1 of NASA's Strategic Knowledge Gaps indicates the importance of understanding and characterizing lunar resources, most importantly water. Permanently Shadowed Regions (PSRs) on the lunar poles are known sources of water-ice, but the distribution of the water-ice is unknown. Lunar Flashlight seeks to map the distribution of surface water-ice in PSRs using active infrared reflectance spectroscopy[1]. By mapping the distribution of lunar water, LF can play a key role in the acquisition of valuable lunar resources. However, to accomplish this science, LF first must reach the Moon.

Entering an orbit around the Moon is not easy: it requires a significant change in velocity (ΔV) imparted in order change a spacecraft's ballistic trajectory such that it is captured by the Moon's gravity. Many existing CubeSat propulsion technologies cannot achieve the ΔV required to enter a lunar orbit, but LF possesses an experimental "green" monopropellant propulsion system that provides the required ΔV, allowing for Lunar Orbital Insertion (LOI). LF's pre-launch Concept of Operations (ConOps) reflected this capability, with a high intensity post-launch phase and low intensity cruise phase leading up to LOI. After LOI, LF would enter a Near Rectilinear Halo Orbit (NRHO) and start a two-month high intensity science campaign phase with low-altitude perilune science passes every 6 days. In-flight propulsion anomalies prompted a significant change in ConOps, and LF is now targeting a trajectory of lunar flybys that has a lower ΔV requirement while still enabling science.

Throughout all mission phases, operators use NASA's Deep Space Network (DSN) to communicate with spacecraft in two-way contacts. Operators monitor spacecraft health, respond to anomalies, and command the spacecraft to execute activities. The most important activity that operators perform are Trajectory Correction Maneuvers (TCMs), in which the spacecraft fires its thrusters to impart ΔV and change trajectories.

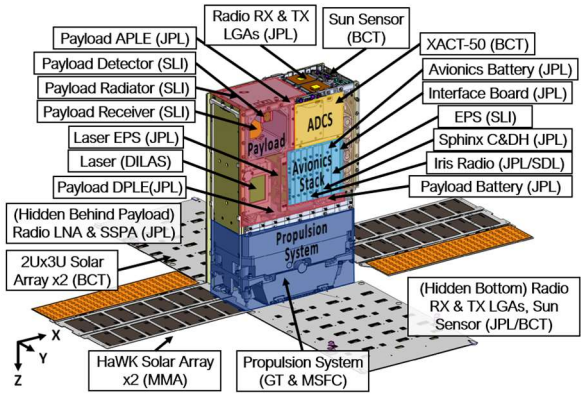

Figure 1: Lunar Flashlight Spacecraft and Subsystems

Spacecraft Overview

Lunar Flashlight, shown in Figure 1, is a 6U CubeSat classified as a technology demonstration mission under NASA's Science and Technology Mission Directorate. As a tech demo, LF utilizes a wide range of experimental technologies that have never flown before, which are indicated by $_{TD}$ subscripts in the subsystem breakdown[2]. The LF subsystems are:

Payload$_{TD}$: A short wave infrared laser reflectometer with four 72W lasers, a detector, and a dedicated battery and power system. The payload produces a significant amount of heat when operated, which is distributed to a radiator via a novel phase change material.

Attitude Determination and Control System (ADCS): A Blue Canyon Technologies XACT-50 attitude control system with three reaction wheels, four sun sensor arrays, an inertial measurement unit, a star tracker, and independent firmware. The reaction wheels are used to control the spacecraft's attitude and store angular momentum imparted on the spacecraft through external torques such as solar radiation pressure or thruster firings. If the reaction wheels saturate due to excessive angular momentum buildup, such as that imparted by a faulty propulsive burn, the spacecraft would spin out of control and the mission would likely be lost.

Lunar Flashlight Propulsion System (LFPS)$_{TD}$: A novel propulsion system created in a collaboration between Georgia Tech's Space Systems Design Laboratory, JPL, and NASA's Marshall Space Flight Center. A pump feeds ASCENT monopropellant, a less toxic alternative to hydrazine, through a 3D-printed titanium manifold's fuel lines to four thrusters. Unlike hydrazine, ASCENT propellant must pass over heated catalyst beds in the thrusters to react fully. Each thruster is placed off the spacecraft center of mass and therefore exerts a torque on the spacecraft when fired. The torques can be used to perform desaturation maneuvers (desats) to dump momentum out of the XACT's reaction wheels. To nominally perform TCMs and achieve ΔV, the XACT would command the four thrusters to pulse in unison, exerting a translation force on the spacecraft while autonomously balancing torques to avoid reaction wheel saturation.

Power: Four solar arrays, an electrical power systems board, and a 3s2p Li-Ion battery.

Command and Data Handling (C&DH)$_{TD}$: A JPL-provided radiation-hardened Sphinx flight computer running flight software (FSW) written in JPL's F Prime framework. LF FSW allows for command sequences to be uplinked and executed onboard the spacecraft with precision timing, but notably, these command sequences are linear and do not allow for conditional branching (i.e., no "if" statements).

Communications: An Iris radio with two pairs of patch antennas, designed by JPL and built by Utah State University's Space Dynamics Laboratory. Operators communicate with LF via the DSN. The Iris supports multiple data rates for both uplink and downlink, broadcasts real-time telemetry, and allows for recorded telemetry files to be downlinked.

The Mission Operations System

The Lunar Flashlight Mission Operations System (MOS) is an interconnected network of teams, individuals, tools, hardware, processes, data, and other resources. The team-level interfaces of the MOS are shown in Figure 2, as well as the entities composing the LF Project.

In the summer of 2021, JPL contracted Georgia Tech (GT) as Lunar Flashlight's primary Mission Operations Center and Ground Data System (MOC/GDS), hereby referred to as Ops. In doing so, GT became the first university to operate a JPL mission. A team of four GT grad students formed the initial Ops team, which grew to 13 operators over the next year: seven graduate students and six undergraduate students.

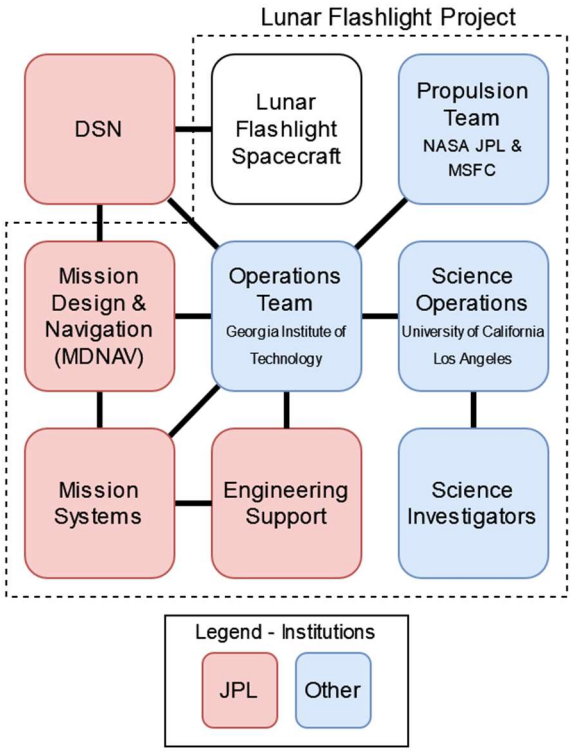

Figure 2: The Lunar Flashlight High-Level Mission Operations System

The LF operations team is ultimately tasked with commanding the spacecraft to achieve mission objectives while maintaining spacecraft health and safety. The responsibilities of Ops are divided into two main categories: strategic and tactical. Tactical operations are conducted while operators are in two-way communication with the spacecraft via the DSN, and strategic operations are conducted during the time in between contacts.

Strategic operations include mission planning, DSN scheduling, activity development, documentation writing, software and tool development, telemetry analysis, spacecraft modeling, and more. Mission planning is the process in which future spacecraft activities are formulated. Typically, mission planning is broad in scope and has a long event horizon, meaning that activities are planned out well into the future. Factors such as key trajectory-related events, operator availability, DSN scheduling, and activity priority must all be considered during mission planning. During mission planning meetings, activities are slotted into a Mission Events Timeline (MET). The LF MET is a Microsoft Excel file saved in cloud storage that details activity planning, tactical contact times, meetings, staffing, and parameters for the Sequence Generation tool described later in this paper. A sub-section of the MET is shown in Figure 3.

Overview					DSN
Activity	Num	Start (EST)	End (EST)	Description	Station
Meeting		4/3/2023 17:00	4/3/2023 18:00		
Contact	217	4/3/2023 21:45	4/3/2023 23:25	PN DDOR	Station 36/25
Meeting		4/4/2023 11:00	4/4/2023 12:00	GITL Timing Test Part 2 (Test C) Activity Review	
Contact	218	4/4/2023 13:00	4/4/2023 17:00	c218: GITL Timing Test Part 2 (Test C)	Station 55
Meeting		4/4/2023 17:00	4/4/2023 18:00		
Contact	219	4/4/2023 20:25	4/5/2023 0:25	c219: Autonomous	Station 26
Meeting		4/5/2023 11:00	4/5/2023 12:00	GITL Delta V Kickoff & Quick Review of GITL module TT (Test D)	
Contact	220	4/5/2023 11:55	4/5/2023 15:15	c220: GITL module TT (test D)	Station 55
Meeting		4/5/2023 17:00	4/5/2023 18:00		
Contact	221	4/5/2023 20:25	4/6/2023 0:25	c221: Autonomous	Station 26
Meeting		4/6/2023 11:00	4/6/2023 12:00	GITL Module while rotating TT Activity Review (test E)	
Contact	222	4/6/2023 12:20	4/6/2023 16:20	c222: GITL Module while rotating TT (test E)	Station 56
Meeting		4/6/2023 17:00	4/6/2023 18:00		
Contact	223	4/6/2023 20:20	4/7/2023 0:00	c223: Autonomous	Station 26
Meeting		4/7/2023 11:00	4/7/2023 12:00	GITL Delta V Pre-Activity Review Meeting & Flow Diagram Review	

Figure 3: Lunar Flashlight Mission Events Timeline

During tactical operations, operators are in two-way contact with the spacecraft using the DSN. Contacts are typically one to two hours long, with an additional hour of setup and half an hour of teardown, making a tactical shift up to 4 hours long. Since LF operates in the Earth-Moon system, the light time delay for radio communications is on the order of single digit seconds. Because of this, LF operators can manually "joystick" the spacecraft during two-way contacts: sending a command, observing a response, and using the response to inform the next command. Since LF command sequences do not allow for conditional checks or branching logic, complicated activities with dependencies are performed Human in the Loop (HitL).

During a contact, the tactical operations team executes spacecraft activities such as propulsive maneuvers, laser firings, and subsystem testing by following pre-approved procedures. Operators may also tactically respond to spacecraft anomalies. Three tools described later in this paper are used for tactical operations: command line scripts, Downlink Helper, and SMARTS.

Mission Timeline

Lunar Flashlight launched on December 11th, 2022, aboard a SpaceX Falcon-9 as a secondary payload to ispace's Hakuto-R lunar lander. At 4:43 AM Eastern Time, the Ops team successfully contacted LF for the first time.

Most subsystems have performed remarkably well in flight. Power, thermal, comms, C&DH, and ACS have all exceeded expectations despite experiencing occasional anomalies. Results from the Artemis launch shave shown how difficult and risky deep space CubeSat

missions can be[3], so the fact that LF is still largely functional is noteworthy. However, the propulsion system has remained anomalous since day two of the mission, when the first desat was attempted but resulted in net momentum gain rather than loss.

After weeks of extensive propulsion testing, the root cause of the propulsion anomaly was narrowed down to a probable cause: Foreign Object Debris (FOD) was blocking the fuel lines. The thrusters were operable, but with highly variable and unpredictable performance levels. Firing all thrusters simultaneously during a TCM was not feasible: if the thrust levels became unbalanced, the spacecraft would experience an unrecoverable torque. A dramatic change in ConOps was required. The LF project conceptualized a new mode of operation for TCMs, in which the spacecraft rotates about an axis while firing one thruster.

Rotating TCMs

Rotating TCMs (RTCMs) balance the spacecraft's momentum state such that the reaction wheels will not saturate while firing a single thruster. This novel mode of operation has required countless hours of testing, analysis, modeling, and development, along with a flight software update and ACS parameter updates. The complete derivation of RTCMs is out of scope for this paper, but an overview is given below.

When firing a thruster, the spacecraft is subject to a torque $\vec{\tau}$ equal to the cross product of the thruster location with respect to spacecraft center of mass and the thrust force vector. Within the spacecraft body frame, $\vec{\tau}$ is constant in direction for a given thruster and scales with thrust. Let $\vec{h}$ be the angular momentum of the spacecraft, including that stored in the spacecraft's reaction wheels. Note that torque is also the time derivative of angular momentum.

During an RTCM, the spacecraft is rotated about a single thrust vector. Figure 4 shows angular momentum $\vec{h}$ in the spacecraft body frame projected onto the x'-y' plane, which is normal to the thrust vector. As the spacecraft rotates, $\vec{h}$ traces a "momentum circle" circle on the plane. At a key phase $\theta = 180°$, $\vec{\tau}$ directly opposes $\frac{d}{dt}\vec{h}$. If the thrusters are fired at this key phase with the correct thrust magnitude, torque and change in angular momentum will cancel out, preventing angular momentum buildup. If the thrusters are fired when not at this key phase, the angular momentum will build up and threaten to saturate the reaction wheels.

Starting in late January, LF performed multiple sequenced RTCMs and seemed to be on track to reach the Moon. However, additional thruster degradation put an end to the sequenced RTCM campaign and required additional adaptation from the project, as time was slipping away.

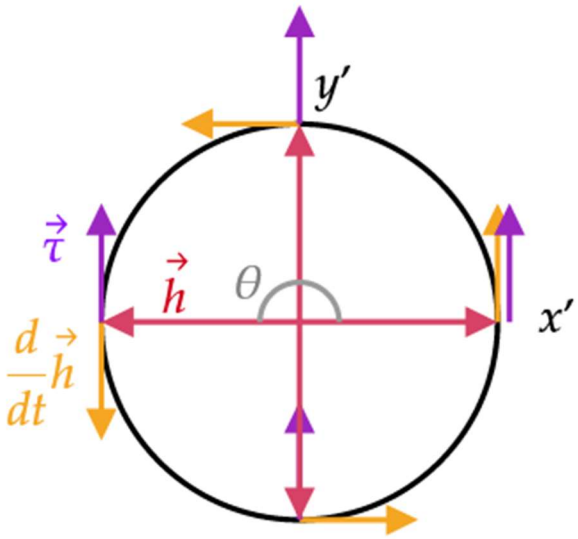

Figure 4: RTCM Momentum Circle

Progressively riskier thruster recovery activities were performed, such as running the pump in reverse in an attempt to dislodge FOD. Since FOD entering the fuel pump from downstream threatens to shred the impeller, the decision to try this approach was not taken lightly. However, the tests paid off: results indicated that FOD could be dislodged from the fuel lines with a reverse pump operation. However, during subsequent burns, cleared FOD would eventually settle in the fuel lines again, reducing or completely cutting off propellant flow after a variable amount of time. The current approach of executing RTCMs with command sequences was no longer sufficient; without branching logic, sequenced commands required a certain level of predictability in thrust to balance the spacecraft momentum state. An entirely new mode of operation was conceived: propulsive burns would be commanded, with the required precision timing, by operator-developed ground software known as SMARTS.

RTCM development is still ongoing; the ΔV produced so far by the thrusters is much less than that required to reach the Moon. The original reference trajectory has been rejected in favor of a trajectory with lower ΔV requirements. Rather than enter an NRHO, LF now intends to perform multiple lunar flybys over a period far longer than the mission's intended lifespan. To achieve this trajectory, Lunar Flashlight must work around the FOD blockage and achieve significant ΔV in late April and mid-May of 2023. Without this ΔV, LF will be

ejected from the Earth-Moon system and enter a heliocentric orbit.

SOFTWARE DEVELOPMENT APPROACH

This section describes key tools and software developed by the author over the mission's duration. All tools were developed with the intention of decreasing operator workload, which frees up operator resources for other valuable work. This is particularly the case for tactical tools, which are used under time-limited, sometimes stressful conditions. Development of tools was iterative, using lessons learned from Operational Readiness Tests (ORTs), strategic testing, and flight events.

All software tools were version controlled in an export-controlled GitHub repository.

Operations Environment

As a non-JPL institution interfacing with JPL resources like the DSN, cybersecurity was of great concern when developing the ground data system. Tactical operations are conducted using Linux virtual machines (VMs) hosted on computers in the MOC. The VMs are segmented on their own network with no internet access, so that unauthorized users cannot access the DSN by tunneling through the GT network. Additionally, the VMs were shown to be operable for previous missions like MarCO. For these reasons, VMs are provided "as-is" and are not updateable. Tactical tools were developed in Python 2.7 to be compatible with VMs and strategic tools, which can be run from any computer, were developed in Python 3, but were written to be backwards-compatible with Python 2.

Several tools utilize a JPL-developed tactical operations software called the AMMORS Mission Data Processing and Control System (AMPCS). AMPCS is highly customizable and provides Graphical User Interfaces (GUIs) for spacecraft commanding, file uplink, telemetry visualization, and more. AMPCS is the primary tool used for tactical operations; however, the tools described in this paper satisfy operator needs that AMPCS does not. To aid software development, AMPCS has a backend Python-based API called MTAK that allows Python scripts and applications to issue commands to the spacecraft and parse incoming telemetry. MTAK is utilized by tactical scripts, Downlink Helper, and SMARTS.

Linux Command Line Scripts

Command line scripts written in Python or bash were the first pieces of automation developed by the Ops team. They are quick to develop but typically limited in scope, performing a singular operation with all input parameters provided at runtime.

Tactical scripts command the spacecraft or parse telemetry, while strategic scripts are used for product generation or mission planning. Strategic scripts include:

- MDNAV product generation scripts that facilitate exchange of information between Ops and MDNAV in a structured way. Ops scripts parse spacecraft telemetry to generate and deliver spacecraft attitude SPICE files, time kernels, and other data products to the MDNAV team. Additionally, data products and ephemeris files provided by MDNAV are automatically ingested and processed into Ops resources, such as testbed setup scripts.
- Mission planning scripts that parse the MET and send out virtual meeting invites to the LF project for each tactical contact.
- QuickLook generation scripts that parse spacecraft telemetry to generate standardized plots, allowing LF project members to assess telemetry quickly. The templates governing plot creation are stored in simple plain text files, which allows operators with little technical experience to customize QuickLooks without reading code.

Tactical scripts include:

- A propulsion readout script that rapidly sends commands to the LFPS and parses the resulting bytes in real-time telemetry to read out prop system parameters. The creation of this script more than halved the time of LFPS parameter readback.
- A spacecraft filesystem listing script that lists files in a directory on the spacecraft, writes the results to a file, then downlinks and parses the file.
- A script that reports momentum magnitude by calculating the root sum squared of incoming momentum telemetry.

Learning to use the Linux command line proved to be a challenge for most new operators, so command line training was included in the operator training plan. Even after training, most operators were able to work much more efficiently with GUIs for complex operations, which motivated the development of the Downlink Helper and SMARTS.

DOWNLINK HELPER

Motivation

During a contact, Lunar Flashlight transmits real-time telemetry that is indicative of the spacecraft's current state. However, Ops needs to reconstruct the spacecraft state during the time between contacts to look for anomalies and assess spacecraft health and safety. To

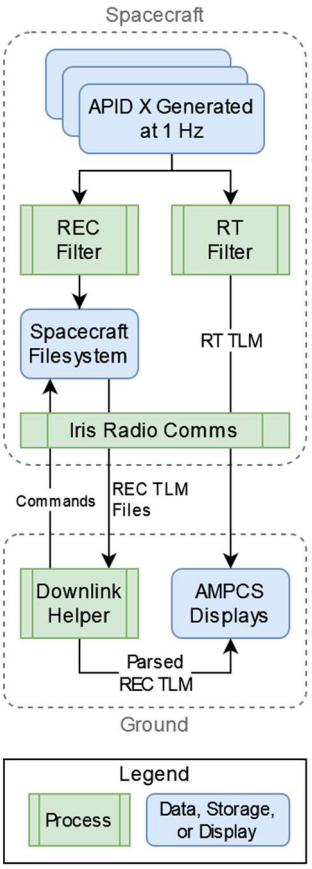

Figure 5: Telemetry Downlink Data Flow

accomplish this, operators downlink recorded telemetry files during contacts, in addition to receiving real-time telemetry. The process of telemetry downlink and Downlink Helper's involvement is shown in Figure 5.

In compliance with CCSDS blue book standards[4], LF telemetry is separated into categories identified by Application Process Identifiers (APIDs). Each APID contains a group of telemetry, for example Event Verification Records (EVRs) or channelized telemetry for power, ACS, C&DH, etc. As per project flight rules, operators first query the APID and time range to assess APID file size and downlink feasibility before initializing downlink. When downlinking recorded telemetry, operators specify an APID and a time range measured in spacecraft clock (SCLK) seconds. The raw FSW commands for querying and downlinking telemetry are not easily human readable: they require referencing an APID table and converting a SCLK argument to datetime format to understand.

During the first ORT, the processes for downlinking recorded telemetry were wholly ineffective. Ops had developed a system involving spreadsheets for tracking what telemetry had been downlinked, and manually generated commands for commanding the downlink. Unsurprisingly, several commanding errors were made, and downlinking telemetry became a significant blocker in tactical procedures.

Implementation

To alleviate these issues, Downlink Helper was developed. Downlink Helper, shown in Figure 6,

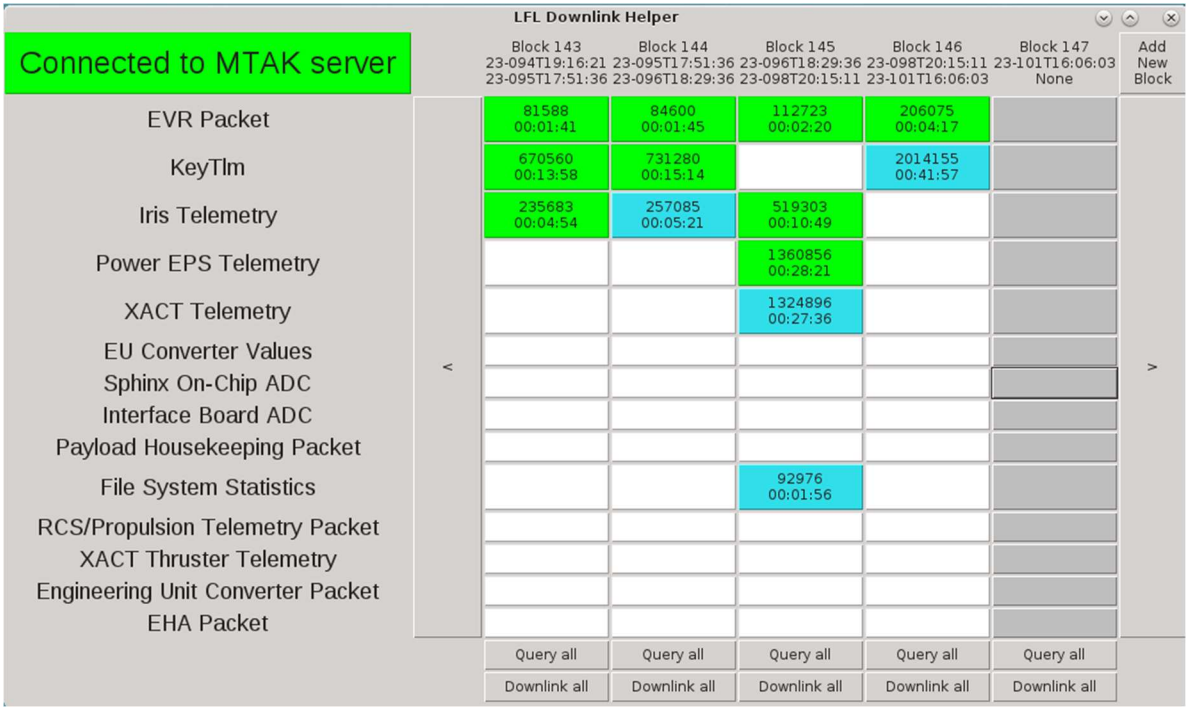

Figure 6: Downlink Helper User Interface

provides a visual representation of recorded telemetry that has been queried, downlinked, or left onboard the spacecraft. Each row represents an APID, labeled by human readable description rather than a number. Each column, referred to as a "block", represents a timespan over which telemetry is queried and downlinked. The timespan SCLK range has been converted to UTC and displayed in a human-readable format. White boxes indicate APIDs that are not queried or downlinked, blue boxes represent queried telemetry that has not been downlinked, and green boxes indicate telemetry that has been downlinked. To downlink past telemetry, operators create a new block if needed, then simply click the APID buttons to issue query and downlink commands. All command parameters and time ranges are automatically handled by the tool. Downlink Helper is intuitive and usable by operators with limited experience, which has dramatically sped up the telemetry downlink process during contacts.

To enable rapid response to any anomalies that occurred between contacts, it is essential to parse and review recorded telemetry as it is downlinked. The version of AMPCS provided to Ops was not able to parse recorded telemetry files as they were downlinked; they had to be run in a separate instance of AMPCS. Because of this, recorded telemetry could not be reviewed until after a contract had been completed, which increased response time to potential anomalies. To enable this rapid response, the downlink helper was modified to detect downlinked data product files, open them, and stream the bytes to a parallel AMPCS session connected to the primary AMPCS session via Java Messaging Service. With this solution, operators were able to parse, view, and assess recorded telemetry as soon as it was downlinked, which enabled rapid anomaly response that proved essential in flight.

After launch, updated models of spacecraft downlink speeds were implemented into the downlink helper. Using the file size of the APID returned by the query command, the downlink helper calculated and displayed the expected downlink time of each APID block. This feature aided operator decision-making during contacts, allowing them to quickly determine what APIDs they had time to downlink when time in the contact was running out.

Future Work

Currently, Downlink Helper does not handle real-time telemetry, but it could be modified to do so. Real-time telemetry transmission shares the total downlink budget with recorded telemetry downlink, so the real-time telemetry transmission rates affect file downlink times. The downlink helper could automatically detect and display the transmit rates of real time APIDs and allow operators to quickly change the rates by sending the appropriate commands. Using the packet size of each APID, the total downlink budget being used by real-time telemetry could be calculated, displayed, and used to update predicted recorded-telemetry downlink times to aid operator decision-making.

APID blocks are inflexible in Downlink Helper's current configuration. It would be preferable if APID blocks could be combined, split, changed in size, or be separate for each APID rather than each APID sharing a block. An option to visually scale each block proportional to the time the block contains would provide an intuitive sense of a block's time range, rather than the operator having to look at the time ranges manually and compare them to the MET.

When the spacecraft filesystem is relatively full, tactically querying APID file sizes takes longer, which can be a blocker to tactical activities. To save time, queries could be conducted in the period between contacts. The recorded EVRs that read out the file size could be downlinked during the following contact, and Downlink Helper could automatically parse the EVRs to create blocks and populate file sizes.

SEQGEN

Motivation

Operators command LF by using 1086 FSW commands defined in an XML command dictionary. Commands are executed in one of two ways: sent in real-time by an operator on the ground or executed onboard the spacecraft by a command sequence. Like commands sent in real-time, sequence commands have a command stem and arguments, but unlike real-time commands, sequenced commands each have a specified timing. Commands can have relative timing, in which they are executed after a specified length of time has passed from the previous command, or absolute timing, in which the command executes once a particular SCLK has passed. Due to their timing functionality, sequences are used to configure the spacecraft outside of contacts, perform activities with precise timing constraints, or act as timeout commands in the event of loss of signal.

Notably, LF sequences are linear and do not have branching: they cannot have "if" statements or other conditional logic. Sequences are written as ASCII text then converted to a smaller-sized binary file that is

uplinked to and executed on the spacecraft. Sequences often contain hundreds of commands, each with carefully chosen timing parameters and arguments. Ops quickly learned that creating them manually is both time consuming and prone to error. To alleviate these issues, LF SeqGen tools were created to automatically generate sequences with minimal input. LF SeqGen is independent of the JPL Seqgen tool[5].

Implementation

SeqGen is a Python-based tool that runs on an object-oriented structure of Sequence and Command classes. The object-oriented backend provides flexibility; sequences are more easily manipulatable once parsed into classes from a text file. The backend has been used for sequence generation and tools like the Linter. The overarching SeqGen data flow is shown in Figure 7.

SeqGen assembles sequences out of individual components. A library of over 40 sequence components was created, with each component representing a specific series of commands within a sequence. Components range in complexity, from configuring the Iris radio for two-way comms to executing a full RTCM. Each component is stored as a version-controlled text file so that they are easily editable by operators with less programming experience. Components often have parameters that are calculated and filled in by automated scripts and processes. Figure 8 shows a component that charges the payload battery. Parameters are indicated by brackets: the times at which to start and end the charging. Other commands in the sequence are standard sequenced commands with no parameters.

```
# charge payload battery          ← Comment
# Power on payload
<startTime> PAYLOAD_POWER_ON      ← Parameter
R00:00:05 PAYLOAD_SOFT_RESET
# Disable thermal control to comply with FR PL-06
R00:00:05 PAYLOAD_DISABLE_BATTERY_THERMAL_CONTROL
# Charge payload
R00:00:05 PAYLOAD_SET_EPS_STATE CHARGING
# Stop charging, nominally 3 hours after charge start
<endTime> PAYLOAD_SET_EPS_STATE IDLE   ← Argument
# Enable thermal control
R00:00:05 PAYLOAD_ENABLE_BATTERY_THERMAL_CONTROL
# Turn off payload
R00:05:00 PAYLOAD_POWER_OFF       ← Stem
          ← Timing
```

Figure 8: Payload Battery Charge Component

Component text files are read in by SeqGen and parsed into an object-oriented library that is easily importable to other Python scripts. A script using SeqGen accepts an input, uses the input to choose components, calculates component parameters, inserts parameters into components, and ultimately assembles the components to write out a command sequence. Separate command files are generated for operators to easily uplink, validate, and execute sequences onboard LF.

Additionally, SeqGen interfaces with a cloud-based sequence database to automatically assign sequence identification and version numbers using a templated

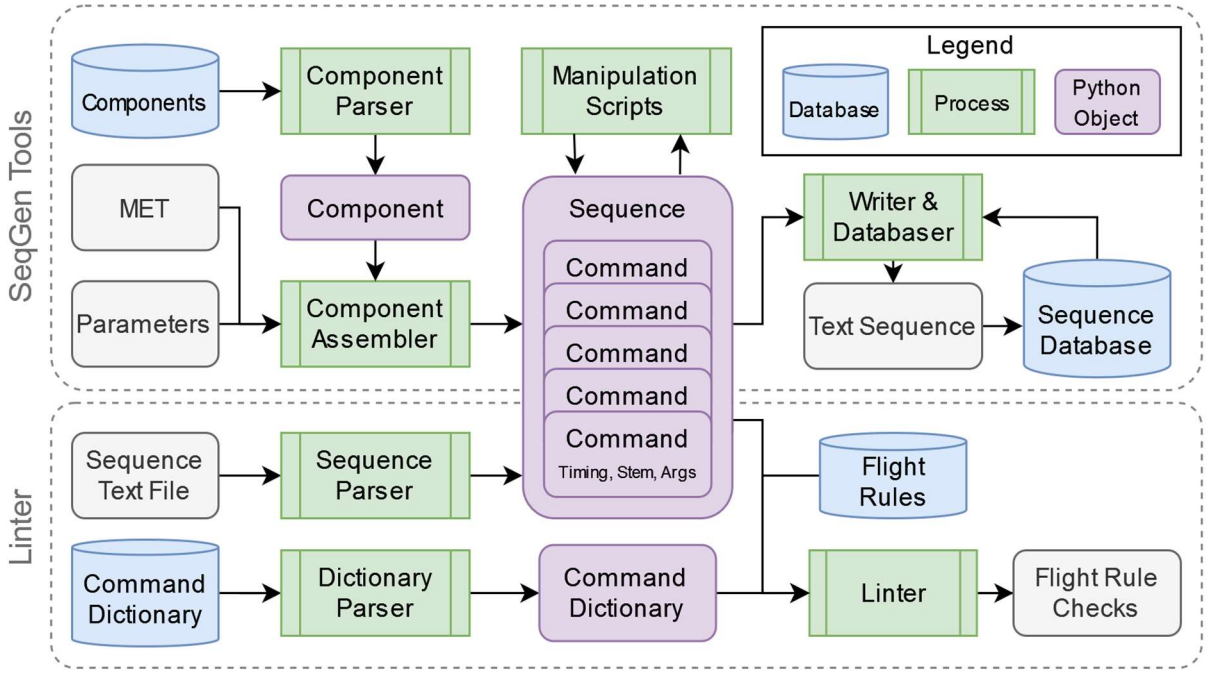

Figure 7: SeqGen and Linter Data Flow

sequence name scheme. SeqGen tools also automatically upload sequences to the database upon generation, providing a development history and additional version control.

Not all LF sequences are generated with SeqGen. Developing scripts that parse input and calculate parameters can be time-consuming, even though applying the calculations to components and producing sequences is trivial. Only sequences that are time-consuming to generate, require specific calculations and parameters, and will be used for multiple activities are implemented in SeqGen.

The first implementation of SeqGen was a script that ingests the MET and uses the specified contact timings and activities to create bridge sequences: sequences that "bridge" the gap from one contact to the next. After a contact, a bridge sequence executes multiple components that safely configure and power off subsystems to put the spacecraft into an "idle" state. Between contacts, components like payload battery charging or the autonomous APID queries described in the previous section can be executed. Before a contact, bridge sequences prepare the spacecraft for tactical operations by configuring the radio, slewing the spacecraft to a sun-pointed inertial attitude, queuing recorded telemetry for downlink, etc. Despite having hundreds of commands, bridge sequences are generated with a single command line input when using this script. The time saved has allowed the generation and V&V of bridge sequences to be handed off to undergraduate students, who would not normally have the time for these activities.

Sequence components are easily interchangeable, allowing operators to quickly adapt sequences to flight requirements. For example, during early flight operations, operators were staffing three 4-hour tactical contacts per day and struggling to keep pace with strategic operations. To free up more time for activity development, an "autonomous" contact component was developed that configured the spacecraft for DSN ranging, performed basic health and safety checkouts, and downlinked recorded telemetry without an operator on console. This autonomous component was completed on December 23rd and allowed the LF project to take a short but much-needed holiday vacation.

When the project moved into the RTCM campaign, SeqGen was updated to generate RTCM sequences. The RTCM component has 16 parameters to calculate and fill in, relating to desired burn direction, time of execution, thruster number, burn duration, etc. SeqGen automatically calculates all these parameters from an input of the MET, contact of burn execution, a propulsion parameter file, and the desired right ascension and declination of the burn as provided by MDNAV. Using an in-house ACS tool called GTball, SeqGen runs an internal ACS simulation: performing quaternion transformations, importing and utilizing SPICE kernels and spacecraft ephemeris, and modeling slews and rotations to check for instrument keep out zone violations. Calculations are displayed for the operator for reference. Complicated burn sequences can be generated in seconds, which vastly accelerates the activity development for propulsive burns. Sequenced RTCMs enabled by SeqGen were successful for several weeks before thruster performance became too variable for sequenced commanding. Now, SeqGen creates sequences that set up RTCMs, but the prop system commanding is handled by SMARTS.

Ultimately, SeqGen has saved the operations team hundreds of hours of work and has likely prevented dozens of command file errors that would have occurred if sequences were generated manually. The framework is flexible, and features like power or comms modeling could be implemented with more development time. The SeqGen framework has been used in other LF tools, such as the sequence linter.

Future Work

SeqGen will be updated with additional components and functionality as required by mission events.

LINTER

Motivation

Most spacecraft are operated in accordance with flight rules: documented constraints that determine what shall and shall not be done on the spacecraft. Flight rules can be documented as early as mission conceptualization and are expanded throughout a mission's lifetime. Many flight rules pertain to command sequences; for example, LF flight rule PROP-16 refers to the order in which two power rails must be enabled on the prop system: "The 5V power must always be turned on before the VBAT power. The VBAT power must always be turned off before the 5V power."

During activity development, sequences and procedures must be checked for flight rule violations. Of the 106 LF flight rules, 52 are relevant to sequenced commanding. Without automation, this activity V&V involves manually filling out a checklist of flight rules, a process so time-consuming that it frequently was abandoned in favor of having expert operators review the sequence during activity review. Fortunately, automation could solve this problem.

A linter is a static analysis tool that automatically checks code for errors. Most modern development environments have a background linter that examines code for syntax errors, bad variables, or other issues as the code is being written. The LF Linter is Python based command line script that checks LF commands sequences for both syntax errors and flight rule violations.

Implementation

Using SeqGen classes, the Linter backend parses command sequences and stores them in an objected-oriented format. Flight rules have also been implemented in an object-oriented fashion. A main script parses a command sequence, iteratively compares it to all the flight rules, and displays the results in one of several possible formats to the operator. As shown in Figure 9, flight rules warnings, flight rule violations, and syntax errors are highlighted, with specific violating commands called out if applicable.

```
Rule COMM01 satisfied

Rule COMM02 satisfied

Rule COMM03 not satisfied
Send commands to configure Iris downlink in sequences in the following order:
        IRIS_MODE
        IRIS_DL_ANTENNA
        IRIS_DL_DATA_RATE
        IRIS_MOD_INDEX
        IRIS_SUBCARRIER_FREQ
        IRIS_DATA_ENC_MODE
        <class 'rules.RuleViolation'>: R00:00:01 IRIS_DL_ANTENNA "DL_ANT_J5"

Rule FSW01 satisfied

Rule FSW08 satisfied

Rule FSW16 not satisfied
No more than 10 consecutive relative timed commands should be used without an a
bsolute timed command interleaved in the command sequence to bound build-up in
execution time delays.
        <class 'rules.RuleWarning'>

Rule FSW19 satisfied
```

Figure 9: Example Linter Output

Future Work

Many flight rules were waived or modified after launch and have yet to be updated in the Linter. Other tools, such as SMARTS, have been a higher priority. Despite being slightly outdated, the linter has saved operators hours during the sequence generation and V&V process.

SMARTS

Motivation

After months of propulsion and ACS analysis, it was determined that performing human-in-the-loop RTCMs was the only way to achieve the ΔV required to reach the Moon. RTCMs require a delicate balance between thrust and momentum while the spacecraft is rotating to remain stable and execute an extended burn. LF's unpredictable thrusters upset this balance, which can result in dangerous momentum states. Unfortunately, LF has no onboard controller that can measure thrust, let alone control the spacecraft to account for it. LF's linear, non-branching command sequences were unable to account for variations in thrust, and RTCMs were repeatedly failing.

Implementation

To enable real-time response to thruster variations during RTCMs, the Semi-Autonomous MTAK Momentum Management And Reactive Time Script (SMARTS) is being developed. With SMARTS, operators can quickly adapt to thruster performance, performing thruster burns, pump reversals, and other propulsion operations with the click of a button. The GUI display of SMARTS is shown in Figure 10.

Before using SMARTS, a background sequence is executed on the spacecraft that initializes the propulsion system and heats the thrusters before slewing to a burn attitude and initializing a rotation. After an amount of time derived from power and thermal analysis, typically around 20 minutes, the sequence will end the rotation and return to a sun-pointing attitude. All commanding of propulsion maneuvers during the rotation are performed with SMARTS.

The timing of commands executed during RTCMs must be precise to properly manage the spacecraft's momentum state. As the spacecraft rotates about a thrust vector at 6 degrees per second, burns must be started within 6 degrees of a key phase angle. Therefore, operators must be able to send ground commands in real-time that execute on the spacecraft at a targeted time with single-second precision. Human operators are not capable of this level of precision, so automation is required. Additionally, propulsive burns drain the battery and rapidly heat the spacecraft. Flight models yield a maximum burn duration of around 20 minutes. With the low thrust provided by thrusters and the current ΔV requirements, no time can be wasted during a burn.

Before any propulsive maneuvers, a latency calibration is performed to calculate the time delay from telemetry reception on the ground to command execution on the spacecraft. Latency varies with light-time delay, spacecraft radio usage, ground server processing times, and other factors contributing to time delay. SMARTS accounts for all these factors when performing a latency calibration.

SMARTS internally models the spacecraft attitude, momentum state, and rotational phase by logging and transforming real-time ACS telemetry transmitted by LF. Using the internal model of spacecraft state and

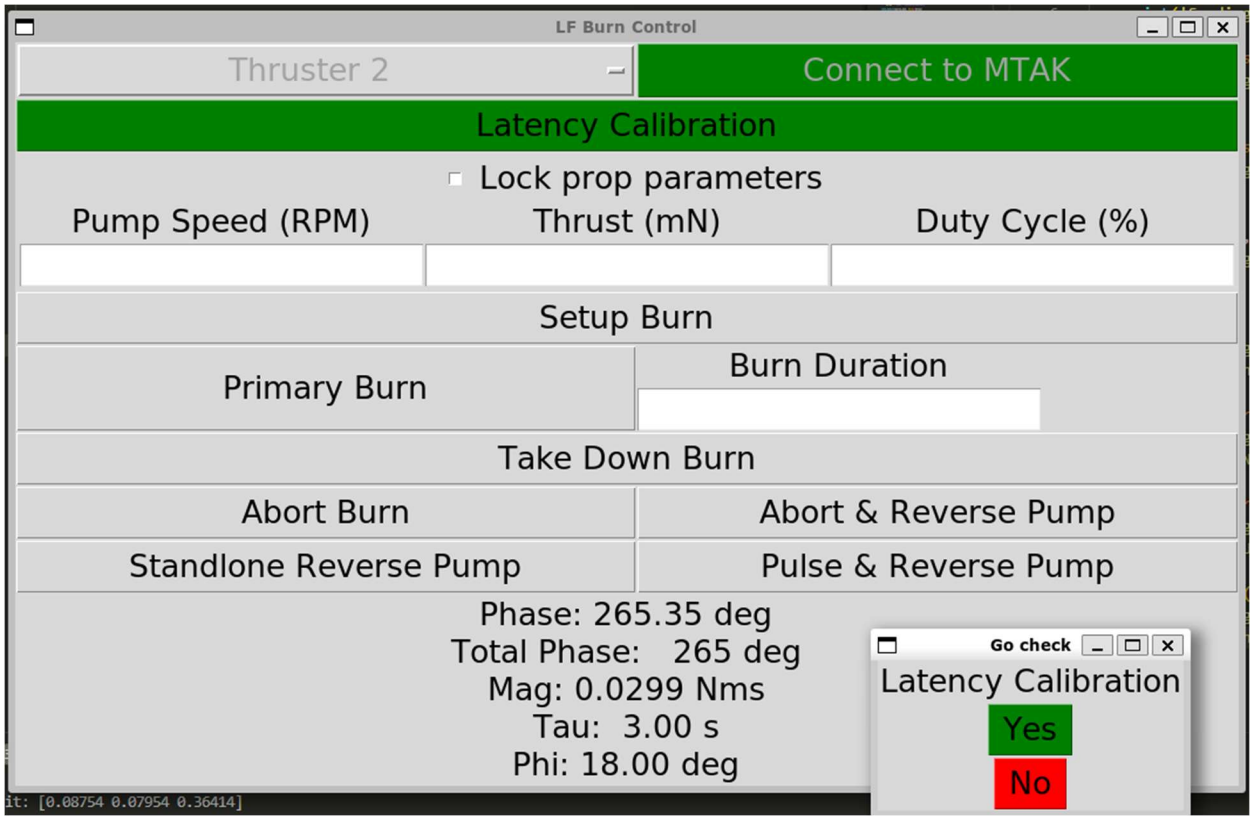

Figure 10: SMARTS Interface

calculated latency, SMARTS can reliably queue and dispatch commands for execution at a particular phase. Initial thread tests run on the spacecraft indicate a phase execution precision of ±1 degree, far outperforming activity requirements.

Depending on thruster performance and momentum state, operators execute command "modules" such as pump reversals, burn aborts, setup and takedown burns to change the spacecraft momentum magnitude, or extended primary burns. Modules are read in from text files using the SeqGen framework. Module parameters such as execution phase and pulse duration vary with momentum state, thrust, and thruster duty cycle, which can be manually set and locked by the operator. With the provided inputs, SMARTS performs parameters calculations internally and inserts them into modules, displaying the results to the operator before sending commands.

When using SMARTS, operators will follow a procedural flowchart to quickly determine which modules to send based on spacecraft state. However, commanding the spacecraft with modules is risky: if modules were to be sent in the wrong order or unintentionally overlap, the prop system could suffer damage. To prevent this, SMARTS also uses the internal spacecraft model in tandem with a procedural state machine to adaptively enable and disable the execution of modules based on spacecraft state. For example, while a burn is executing, the only modules available to send are those which explicitly cancel out of a burn.

Future Work

SMARTS is still in its early stages but is on a rapid development schedule to be completed by the ΔV deadline in late April. A plotting interface is currently under development.

Tactical thread test activities with increasing complexity are being developed and conducted to take incremental steps towards the critical upcoming burn campaigns in April and May of 2023. Ultimately, executing RTCMs using SMARTS is Lunar Flashlight's best chance of reaching the Moon.

CONCLUSION

Software developed by Ops has been essential for the ongoing success of Lunar Flashlight. Downlink Helper has allowed for faster, more intuitive downlinking of recorded telemetry. SeqGen has enabled operators to generate complex sequences with minimal input. The Linter has saved significant time during sequence V&V by automatically performing flight rule checks. Finally, SMARTS enables Ops to perform RTCMs, unique

propulsive activities that can be used to achieve ΔV without saturating LF's momentum wheels. These tools, in tandem with tactical and strategic scripts, serve to remove obstacles in the MOS, reduce operator cognitive load, and mitigate risk to the Lunar Flashlight project.

Lessons Learned

Ops greatly benefited from having the flexible, object-oriented classes developed for SeqGen and LF command sequences. The classes remain useful and can be used for a variety of scripts and tools. However, they were originally programmed in Python 3, which made using them on tactical VMs not possible. They have since been rewritten, and most Ops Python 3 tools are now backwards compatible with Python 2.7.

Operators perform better with GUI tools rather than command-line scripts for multi-step processes. The additional strategic development time it takes to develop a GUI can be worth the time it saves tactically.

Version controlling tools is an essential aspect of development. A simple Git repository with no addons is sufficient; if operators feel overwhelmed by Git complexity they are less likely to use it.

ACKNOWLEDGMENTS

The author would like to thank their advisor Dr. Glenn Lightsey, their fellow members of the Georgia Tech operations team, the management, navigation, and engineering support teams at JPL, the propulsion team, the science team, and all other members of the Lunar Flashlight project. Thank you to Emma Hansen and Nathan Cheek for development assistance with the linter and SMARTS. This work was conducted at the Georgia Institute of Technology under contract from the NASA Jet Propulsion Laboratory.

ACRONYMS

- ACS: Attitude Control System
- AMPCS: AMMOS Mission Data Processing and Control System
- APID: Application Process Identifies
- C&DH: Command and Data Handling
- ConOps: Concept of Operations
- DSN: Deep Space Network
- EVR: Event Verification Records
- FOD: Foreign Object Debris
- FSW: Flight Software
- GT: Georgia Tech
- GUI: Graphical User Interface
- HitL: Human in the Loop
- JPL: Jet Propulsion Laboratory
- LF: Lunar Flashlight
- LFPS: Lunar Flashlight propulsion System
- MDNAV: Mission Design and Navigation
- MET: Mission Events Timeline
- MOS: Mission Operations System
- MTAK: (A)MPCS Test Automation Toolkit
- NRHO: Near Rectilinear Halo Orbit
- PSR: Permanently Shadowed Region
- RTCM: Rotating TCM
- SCLK: Spacecraft Clock
- SMARTS: Semi-autonomous MTAK Momentum Management and Reactive Timing Script
- TCM: Trajectory Correction Maneuver
- V&V: Verification and Validation
- VM: Virtual Machine
- ΔV: Change in Velocity

REFERENCES

1. Vinckier, Quentin, Luke Hardy, Megan Gibson, Christopher Smith, Philip Putman, Paul Hayne, and R. Sellar. 2019. "Design and Characterization of the Multi-Band SWIR Receiver for the Lunar Flashlight CubeSat Mission." *Remote Sensing* 11(4):440. doi: 10.3390/rs11040440.
2. Cheek, Nathan, Collin Gonzalez, Phillippe Adell, John Baker, Chad Ryan, Shannon Statham, E. Glenn Lightsey, Celeste R. Smith, Conner Awald, and Jud Ready. n.d. "Systems Integration and Test of the Lunar Flashlight Spacecraft."
3. Zucherman, Aaron. 2023. "State of Interplanetary CubeSats."
4. Consultative Committee for Space Data Systems. 2020. "Space Packet Protocol."
5. Streiffert, Barbara, and Taifun O'Reilly. 2008. "The Evolution of Seqgen - A Spacecraft Sequence Simulator." in *SpaceOps 2008 Conference*. Heidelberg, Germany: American Institute of Aeronautics and Astronautics.