SSC24-X-08

OPTASAT: An Open-Source, Flexible Software Framework for Small Satellite Operations

Thomas Murphy, Kerri Cahoy
Department of Aeronautics and Astronautics, Massachusetts Institute of Technology
77 Massachusetts Avenue, Cambridge, MA, 02139
tjmurphy@mit.edu

ABSTRACT

This paper introduces the Open-source Python Tool for Awareness of Spacecraft and Analysis of Telemetry, or OPTASAT, software for spacecraft mission operations and planning. OPTASAT is modular and extensible, and incorporates multiple tools to contextualize spacecraft data. The increasing access to space is also growing the pool of spacecraft operators. There is a need for spacecraft operations tools that are open source and accessible. The best tools for modeling and understanding the situation of a satellite in space currently have high barriers to entry including cost, learning curves, and complex interfaces. Missions that cost over $100M and last for years justify hundreds of thousands of dollars for operating software and months of training for operators, but few-million-dollar, two-year missions (as commonly found in smallsats) do not. OPTASAT includes tools to assist with planning for beta angles, astronomical target selection, and Earth-observing sensor simulation. These tools can be used for end-to-end planning of mission actions, from concept to test and operation of the spacecraft. Visualization of data is a priority in the display of data in order to reduce learning curves. OPTASAT is fully open-source and adaptable, as operators need the ability to adjust their software to tailor it to the particular parameters of their missions to suport the wide variety of spacecraft and missions that exist today. In particular, this paper will present examples of astronomy missions, Earth-observing missions, and multi-spacecraft coordination missions that can be analyzed through OPTASAT.

INTRODUCTION

Spaceflight missions are becoming smaller, cheaper, and more approachable for a wide variety of groups. Teams operating spacecraft with limited resources need to have software which will track the state of their spacecraft and help visualize the past, present, and future actions it may take.

The wide variety of the types of space missions being flown (Earth observation, technology demonstrations, astronomy, remote sensing, communications, and more) necessitates software which is flexible and can be tuned to suit the particular needs of the operators. The software should be approachable, allowing users and operators to make use of it with minimal training. Finally, the software should provide sufficient utilities to users, so that they can begin immediately applying it to their mission with minimal setup.

APPROACH

Establishing scope for satellite operational software of this nature is crucial. We can not aim to serve all needs for all space missions. Instead, this software aims to provide a wide variety of capabilities for the most common types of missions. Highly specialized missions will likely still need to use existing commercial software, but simpler missions, will be able to use OPTASAT to meet mission objectives. A goal of OPTASAT is to reduce duplicated work by providing teams with the basic tools they need in order to operate their missions, and minimize the amount of custom tools which need to be created in order to manage the day-to-day tasks of operating a small satellite mission.

OPTASAT has some intentional limitations. The first is that it assumes that missions will be operating in low-Earth orbit (LEO). While plotting mission trajectories to Mars is an essential capability for interplanetary missions, these missions are still rare. This allows many simplifications to be made. For one, the standard SGP4 propagator can be used for the evolution of orbits, and TLE data is readily available for tracking the parameters of the mission at any given time. Another limitation is that OPTASAT assumes missions will be operating without propulsion. Trajectory and maneuver planning are a crucial component for many missions, but most small satellites do not have propulsive capabilities[1], and therefore propulsion is not a built-in feature of OPTASAT. Applying these assumptions to the orbits in the missions simplifies the internal behavior of OPTASAT, without too heavily restricting the breadth of missions of interest which can be supported. This scope limitation makes development and use of OPTASAT more approachable.

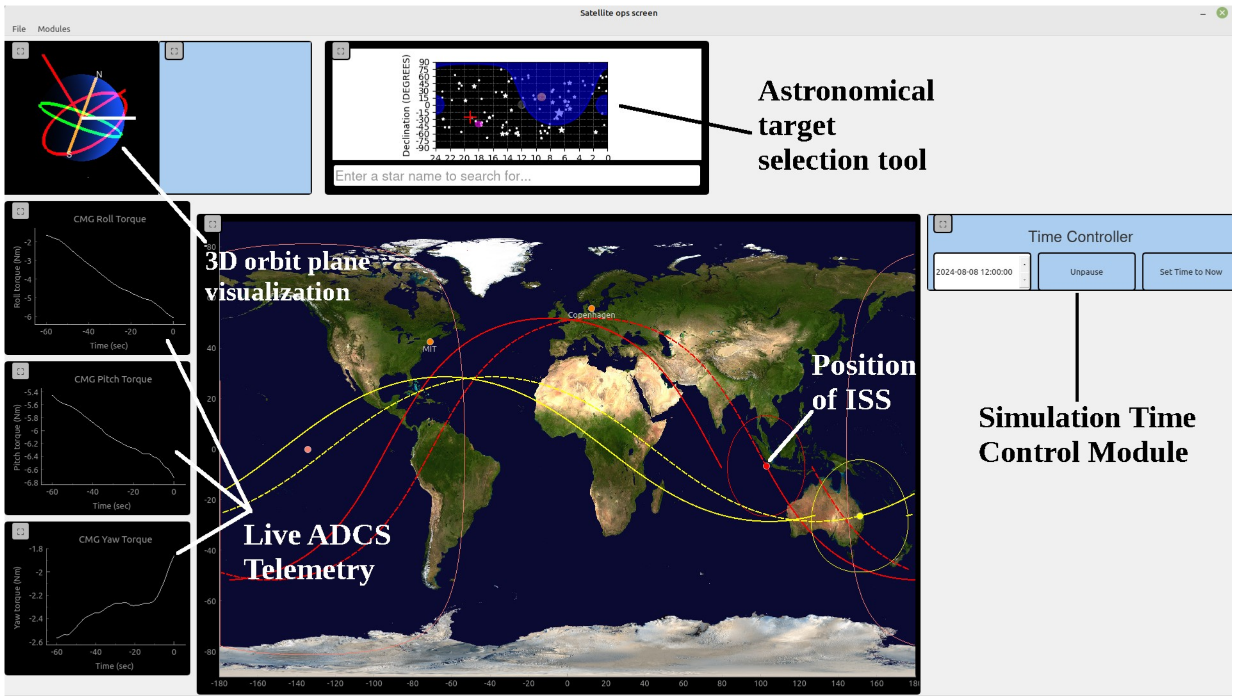


Figure 1: An example of an OPTASAT window in-use, with annotations for some of the visible elements of the interface

We also make some assumptions about the capabilities of the users of OPTASAT. OPTASAT is made to be reconfigurable, but this configuration is done in a manner that requires some level of familiarity and comfort in working with text-based software configuration management. Additionally, we assume that operators (as teams, if not as individuals) are comfortable modifying Python software, since they may need to write custom code to extend the capabilities of OPTASAT to fit their needs.

Using an interpreted language like Python means sacrificing performance when compared to C/C++ or modern alternatives like Rust. However, with modern computer hardware, performance is often not as precious a resource as previously, and the decision is based on prioritizing the needs and abilities of the users. Python is a language that students today are broadly familiar with, and one that a large number of aerospace engineers are capable of writing code in. Given that OPTASAT will be used on aerospace missions, we assume that the users are engineers who are comfortable writing software, as opposed to being software engineers who may have more experience with low-level, systems-oriented languages. Using Python expands our pool of users who will be able to understand and make changes to the software.

Customization by the user is supported on three levels. The first level is that the user interface is fully customizable. The user chooses which controls will be in which part of the screen, and can resize graphical elements according to their priorities. Options which are less often used can be placed behind menus, while key controls can be placed in permanent positions in the window. This level of configuration is done through modification of a JSON (JavaScript Object Notation) file[2]. The second level of customization is the fact that OPTASAT supports the creation of custom user interface modules, which will present mission information to the user in whatever format they find useful and intuitive. Third, customization is limitless due to OPTASAT being free and open-source software, meaning that users are able to adapt any portion of the software to fit their needs.

OPTASAT LIBRARIES AND INTERFACES

OPTASAT is written strictly in Python, with the assistance of some external libraries.

Running OPTASAT from a fresh installation of Ubuntu 24.04 requires simply cloning the source code repository, and installing seven Python libraries. These include PyQT5, PyQTGraph, and PyOpenGL (all for graphics), Skyfield[3] (an astronomy-focused library, with additional capabilities for satellite orbit propagation), Matplotlib and Pandas (for data management and plotting), and LightStreamer (for loading the ISS telemetry stream, to demonstrate live data display). The latest (at time of writing) versions of all of these libraries are supported.

OPTASAT consists of an interface which collects individual modules, each of which is a dedicated Python file for performing a given task. For example, one Module exists which shows the location of the satellite on a map. This is known as the "Mapdot" module. The modules structure is described in further detail in Section 4.

OPTASAT is able to take input from any arbitrary data source. An example is provided which will load the public Lightstreamer-based data feed of telemetry data points from the International Space Station. Equivalently, users could use a ground station radio over a software serial port to receive data directly from their spacecraft, or use any other type of data feed. OPTASAT also automatically fetches TLE data for any spacecraft that the users are interested in, along with caching these files and updating them as they become stale.

Configuration of an OPTASAT installation is performed through the use of JSON text files. The text file encompasses all of the variables needed to describe the behavior of OPTASAT. This means that users can exchange these files with each other in order to share their interfaces. The JSON files describe the layout of every module in the window, as well as any parameters such as what satellite to plot on a given graph, or the field of view of a satellite's sensors. In normal operation, a user will never have to write Python code, and can use the JSON interface to adjust the parameters of any modules they have. Python programming is only needed if the underlying functionality of modules is to be changed, or if the user wishes to create their own modules.

MODULARITY

A wide selection of modules are provided, to provide a basic set of functions for a generic space mission. The provided set of basic modules can support a large number of examples of what can be done within the OPTASAT framework, such that anyone who wants or needs to create a new module can pull portions of working code into their new module. For example, we provide a module which demonstrates three-dimensional plotting through PyQTGraph's OpenGL interface to model an orbit around the Earth, with the equator and axis drawn. We also demonstrate how to embed Matplotlib plots within a module, allowing users who may already have tools producing Matplotlib-based output to easily integrate these into the OPTASAT framework. Table 1 shows a full list of all modules included in the current (at time of writing) version of OPTASAT. Further sections will discuss a selection of modules in detail.

Table 1: Modules included with OPTASAT

Rectangle: The most basic module. Simply displays a colored rectangle within the interface. Primarily useful as a starting point for creating new modules.
Time Controller: Allows the user to set the internal simulation to a different day and time
Telemetry Grapher: Displays live telemetry data from spacecraft in a live-updating, auto-scaling graph
Space Weather Overview: Displays the latest space weather data pulled from NOAA Space Weather Prediction Center
Mapdot: Displays the location of arbitrary number of satellites on a map of the Earth, with their footprints (area of Earth that can see the satellite) drawn, along with the next and previous orbits. Also labels ground stations, which are listed in JSON. Can also simulate the area of Earth imaged by a defined satellite-mounted imager.
Passfinder: For any ground station and satellite, generates a list of times that the satellite will pass over the ground station, and visually displays each as a circle sized according to the maximum elevation of the pass above the horizon.
Pass Polar Plot: Displays the current locations of any satellite in the sky from the point of view of a ground station.
3D Orbit Renderer: Displays a 3D view of the orbital plane of a satellite, oriented against the sun, to view the physical Beta Angle for contextualization
Follower Satellite: Generates a virtual satellite which follows an existing satellite at a specified spacing. Useful for planning of small constellation missions.
Ground View: Simulates the view of the Earth through an Earth imaging sensor with a given field of view
Starmap: Simulates the view of celestial objects for astronomy missions
Eclipse Plot: Generates a timeline for a satellite, indicating at all times whether it is eclipsed by Earth.

The use of modules allows the functionality for a given module to be contained within a single Python file, meaning that any given module can be integrated into an OPTASAT configuration setup, and adjusted to the user's preferences. If a module is no longer used, it is simple to remove the code for that module, and OPTASAT will continue running. If users want to extend the capabilities of a given module, they know immediately where to find the code for that module, since it will simply be in the Python file which shares the name of the module.

Modules also aid in the onboarding process for new users. A user does not necessarily need to know how to operate every portion of an OPTASAT configuration in order to be up and running. If one user is training another, they can isolate their focus on one module at a time, and learn each module when they are ready.

OPTASAT includes several ready-made modules to serve purposes that are considered to be common across many space missions. Here, we discuss several modules to highlight the types of modules which exist in OPTASAT.

Time Controller

The Time Controller module is a simple module, which features a readout of the date and time of the internal OPTASAT simulation. This module, shown in Figure 2, provides an example of cross-module functionality, since the user's interaction with this module affects the global time variable. Most importantly, the time entered into this module is passed to the Skyfield-based satellite propagator, and is used to propagate the TLE for the user's spacecraft to the specified date. The most important use-case for this functionality is that of future data collection planning. For example, in an Earth-imaging mission, the Time Controller can be used to predict the locations that the spacecraft will fly over in the near future, and thus allow the users to plan out their next ground targets of interest. The Time Controller can also be used to view times in the past, which is useful for evaluating events like telemetry anomalies to trace their causes. This may include determining whether a sudden temperature increase is coincident with the spacecraft exiting eclipse, or whether a spacecraft reset occurred while it was passing through the South Atlantic Anomaly. The Time Controller simply propagates the spacecraft's current TLE to the chosen time. Due to the limitations of TLE files, propagations far from the current date will not be reliable[4]; future development of OPTASAT will include automatic fetching of historical TLE data, to use the most contemporary TLE for any chosen date in the past. All times in the Time Controller operate on Universal Coordinated Time (UTC).

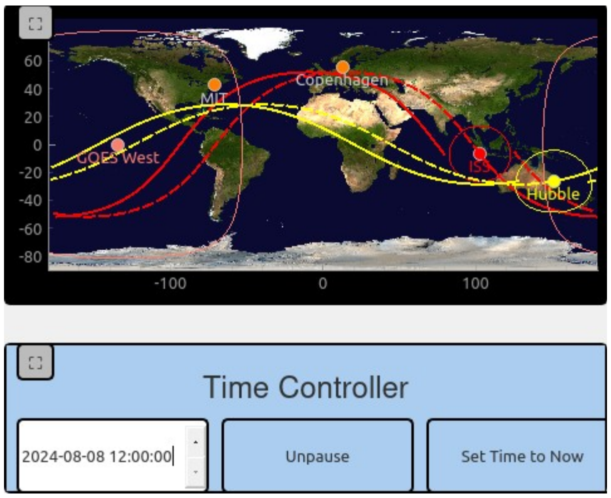


Figure 2: MapDot and Time Controller modules

MapDot

The MapDot module, shown in Figure 2, shows the location of the spacecraft as a dot on a map of the world. This module allows the user to plot the location of one or more spacecraft, and integrates with the Time Controller to show the locations at any chosen time. The MapDot also allows the user to specify in the JSON configuration any ground sites of interest. These could include communication ground stations, imaging sites of interest, or any other location on Earth. The MapDot also allows the user to specify sensors on their spacecraft, which have a user-defined field-of-view (FOV), that can then be plotted on the map. For example, if the spacecraft has a camera with a 40 degree field of view, MapDot will simulate a 40-degree cone being projected from the spacecraft toward Earth, and will identify the region of Earth's surface which is contained within that cone by plotting points on the map where the cone intersects. Additionally, the user can control the orientation of the cone. By default, the sensor is pointed in the nadir direction (directly at Earth), but during runtime (rather than in the JSON) the user is able to move the sensor off-nadir by whatever angle is desired. This is presented in two axes, where one axis is the size of the angle between the camera's boresight vector and the nadir direction, and the other axis is the direction of that deviation. For example, we can be 10 degrees off-nadir, to the southeast (at an angle of 135 degrees). This allows mission operators to plan their opportunities for imaging and evaluate whether a given ground target will be accessible from a given orbit.

StarMap

While the MapDot is useful for planning of Earth imaging missions, the StarMap module is intended for astronomy missions.

The StarMap module (Figure 3) is derived from a tool which was developed for the DeMi (Deformable Mirror Demonstration Mission) spacecraft[5]. This was a 6U cubesat operated by MIT which featured a miniaturized space telescope for testing adaptive optics in space. This necessitated tools which would inform the operators of the celestial objects available for imaging. DeMi had several priorities in object selection. These included the preference for bright objects, the restrictions of Sun keepout zones, the awareness of which objects were obscured by the Earth, and more. The initial approach for solving this problem was to develop a weighting-based function to evaluate each celestial object and choose the one with the highest score, but this made it difficult to validate that the selection was correct. Therefore, the StarMap was developed, and later, with the development of OPTASAT, was converted into a module rather than being a standalone application. The StarMap is an embedded MatPlotLib plot, meaning that users who are familiar with MatPlotLib plotting in Python will be able to understand its internal structure. It also provides a strong example for any user who wishes to construct a module consisting of a MatPlotLib plot.

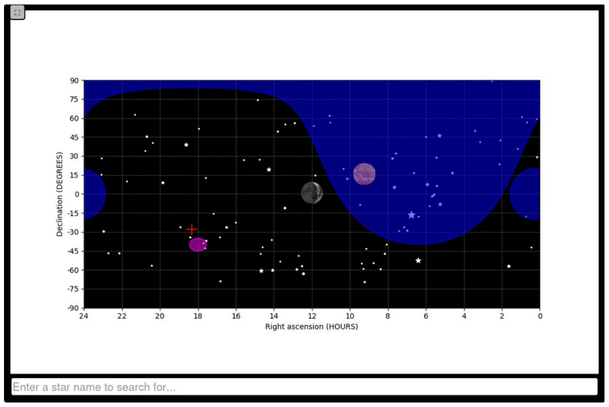


Figure 3: The StarMap module, displaying the stars, the Sun and Moon, and the region of sky obscured by Earth.

The core feature of the StarMap is that it displays a map of all the stars in the sky, over the full celestial sphere. Every star is plotted at its location, and every star is visible at all times. Familiar constellations like Orion are readily visible. The stars are plotted with an equirectangular projection[6], where the right ascension and declination are directly translated to the X and Y coordinates on the plot. Stars are drawn with their physical size proportional to their apparent magnitude, causing brighter stars to be more readily visible. In the configuration for the map, a threshold can be set, such that only stars brighter than a given magnitude are drawn. The star catalog is obtained from the Skyfield library, which contains the Hipparcos star catalog.

Besides the stars, the next key element of the StarMap is the drawing of the Earth. The Earth is rendered by calculating the distance from the satellite to the Earth, and then using that to derive the Earth's angular size. We then use the Haversine Formula to calculate points around a circle (in spherical coordinates) which are equidistant from the location which points from the spacecraft to the center of the Earth. By doing this, we map a circle onto the equirectangular projection, and can display the distorted shape of the Earth, with the distortion of the projection affecting the apparent shape. In the event that the satellite is at a high latitude in orbit, the Earth may be obscuring one of the two Celestial Poles, in which case the Earth will stretch across that edge of the StarMap.

The next elements shown in the StarMap are the Sun and Moon. These are useful for awareness, and can also have keepout zones associated with them, configured by JSON. Because the StarMap also responds to the Time Controller module, users can also put in the date of a Solar Eclipse to see the Moon cover the Sun, which is a useful confirmation of functionality. More importantly, the Sun and Moon are also useful for evaluating when either is being shadowed by the Earth, which is beneficial for avoiding stray light, and having extra assurance that the telescope will not point at the Sun, including inadvertently passing by the Sun when slewing attitude from one imaging target to another. The StarMap also calculates and renders the current phase of the Moon by applying a calculated shadow mask over the image used for the Moon.

The StarMap allows specification of arbitrary keepout zones, which may be configured to any desired angle, and can either be located around an object (especially the Sun or Moon), or located at an arbitrary stellar location if needed. This allows users to identify which stellar objects remain available for imaging.

Finally, the StarMap shows a cross at the location in the sky which is represented by the vector pointing from the center of the Earth to the spacecraft. Equivalently, this is the vector pointing from the spacecraft directly away from the Earth. Any stellar target near this cross will maximize the avoidance of any effects of Earthshine. This cross also heavily aids with keeping track of the location of the spacecraft as it moves through its orbit, and works as a useful anchoring point for the user to contextualize what is shown in the StarMap.

Orbit Renderer

The 3D orbit renderer displays a full 3D view of the Sun, Earth, and the spacecraft, and allows the user to move the virtual camera around arbitrarily. One of the most useful purposes of this module is to allow the viewing of solar beta angles. While the equations are straightforward to calculate a beta angle, telling an operator “Our current beta angle is 55 degrees” generally is not helpful for understanding the mission-level implications of that value. By displaying the beta angle on a physically viewable Earth, and especially by allowing the rotation of the camera for a full 3D view (as opposed to textbooks which show a 2D drawing and attempt to show the relationship between 3D vectors), it becomes more straightforward to understand the state of the illumination of the spacecraft. While OPTASAT is primarily intended for spacecraft operations, this module also helps to illustrate the educational value that may be obtained by transforming spacecraft state information into tangible, visualized interpretations for analysis by humans. Showing a real example may be highly useful in a classroom environment.

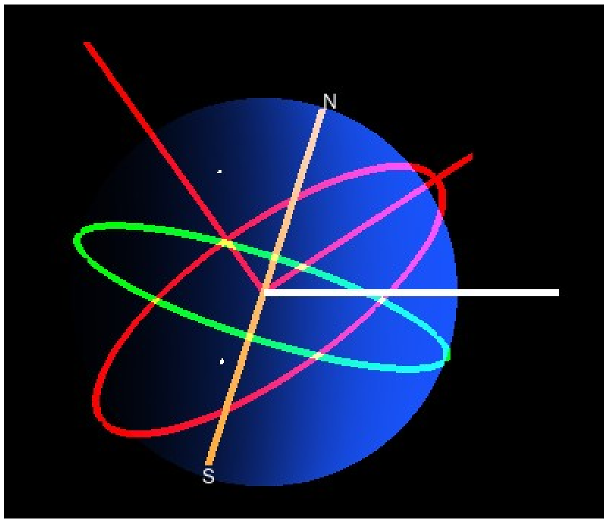


Figure 4: 3D orbit renderer, showing:
Orange - the axis of the Earth, with labeled poles
Green - the Equator
White - the sun-pointing vector
Red - the ISS orbit, its normal vector (to the upper left), and the sun vector projected onto its plane (to the upper right). The beta angle is the angle between this vector and the sun vector.

Pass Finder

The Pass Finder module (Figure 5) identifies when a particular satellite will be flying over the user’s specified location. The primary use case for this module is for identifying opportunities for communication with the satellite from the location of a given ground station. The Pass Finder will calculate the overflight opportunities, and importantly, will classify them in terms of their quality. It is rare for a satellite to fly directly overhead of any ground station location, and therefore some passes are of higher quality than others. A pass which consists of the satellite rising 5 degrees above the horizon is much less desirable than one that rises 80 degrees above the horizon. The maximum elevation of a pass acts as a metric for the quality of the pass.

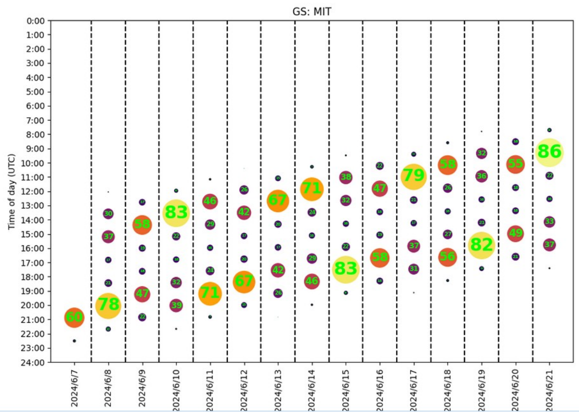


Figure 5: An example of the output of the Pass Finder module

In the Pass Finder, each pass is displayed as a circle. The location of each circle corresponds to the time and day of the pass. Each day is a vertical strip of time, labeled along the X axis. Along the Y axis, each day progresses from top to bottom, with noon directly across the middle. This arrangement resembles many popular calendar software tools. Each circle is displayed to indicate the maximum elevation of the pass, which is indicated in three different ways. First, the numerical value of the maximum elevation is displayed on each circle. Second, each circle is sized proportionally to the maximum elevation. Third, each circle is colored (using the Inferno color map) to indicate the quality of the pass, with the highest quality being light yellow. This way, it is easy to identify the highest-quality passes at a glance. This display also indicates trends in passes. In the Pass Finder results of Figure 5, we can see that passes get earlier in the day as time goes on, which can be a useful point of context for pass planning. We can also see that, for this ground station (MIT), in this orbit (the International Space Station), there are usually a total of about 6 passes per day, of which one or two are of high quality. Knowing these properties of an orbit is useful for mission planning prior to launch, as it allows operators to form an expectation for what level of access will be available to their satellite.

More modules are available with OPTASAT (see Table 1), and the wide variety of provided modules is intended to provide enough material for users to create their own modules and have enough examples to confidently understand how to interface with the rest of OPTASAT. Development of an explicit Application Programming Interface (API) is a crucial element of future work.

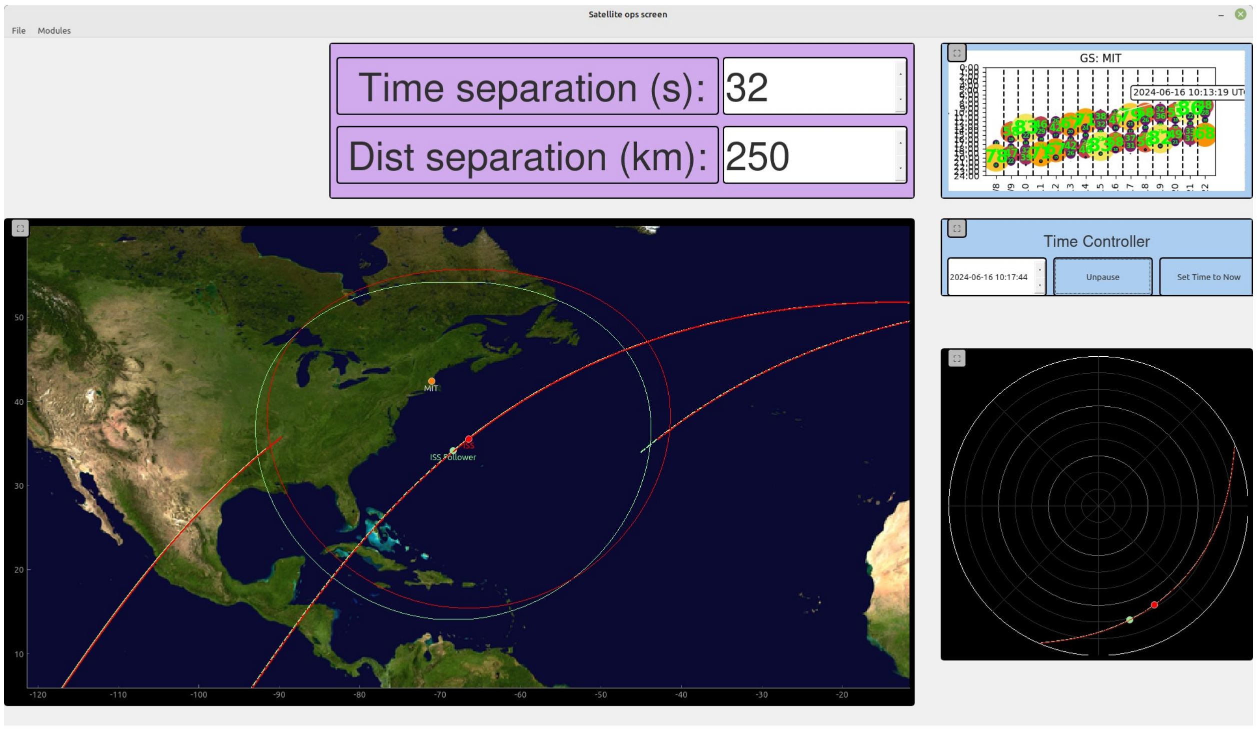
APPLICATION CASE STUDY: CLICK

CLICK[7] (CubeSat Laser Infrared CrossLinK Mission) is a mission to perform communications between two 3U CubeSats using an infrared laser link. CLICK-A flew in 2020 and demonstrated successful optical downlink from a single spacecraft to an optical ground station. The dual-spacecraft mission of CLICK-B and CLICK-C will be launching in 2025.

A key task in planning for operations of CLICK-B/C is that the operators need to determine the nature of ground radio communication with the two spacecraft. The problem can be summarized by asking "If the two spacecraft are some number of kilometers apart, what will our communication opportunities with the two of them look like?". Most importantly, the operators need to establish whether communication with the two spacecraft can happen simultaneously with a single antenna, or whether a ground antenna will have to pick one or the other to communicate with.

In order to visualize the dynamics of a pass with a pair of satellites, an OPTASAT configuration was built to simulate two spacecraft in LEO passing over a ground station. This configuration is shown in Figure 6. As a stand-in for CLICK-B and CLICK-C, we use the ISS as the "leader" satellite, and we use the Follower Satellite module to generate a satellite which follows the ISS, at whatever time or distance separation is needed (the time and distance are connected through the orbital velocity). In this screenshot, we have set the follower to lag 250 kilometers behind the ISS. We used the Pass Finder module (in the upper right) to select an arbitrary upcoming pass, and this moved the simulation time to the start of the pass. In the screenshot, the Pass Finder appears slightly cluttered, but this is because it is currently in a minimized format. Selecting the grey button in the upper left of the module will minimize the large map view, and will expand the Pass Finder to fill the large central area of the configuration.

Once a pass is selected and being viewed, we can use the Time Controller module to shift time between the beginning and end of the pass, to see the motion of the satellites. Alternatively, we can simply let the Time Controller run, and watch the pass at real-life speed. In the black box in the lower right, we use the Pass Polar Plot module to view the positions of the two satellites (ISS in red, follower in green) as they appear from the perspective of an observer, and the track that they are following throughout the pass. The five modules in this configuration work closely together in order to give a cohesive understanding of the dynamics of a dual-satellite mission. This example illustrates some of the capabilities of OPTASAT and the useful ways modules can be combined.


Satellite ops screen

Time separation (s): 32

Dist separation (km): 250

GS: MIT

2024-06-16 10:13:19 UTC

Time Controller

2024-06-16 10:17:44

Figure 6: A configuration for modeling the dynamics of the CLICK mission

FUTURE WORK

While the groundwork has been laid for OPTASAT, it remains an ongoing software development project, and there have been several improvements identified which will be addressed in the future.

OPTASAT will need the capability to obtain the most relevant historical TLE for a given date in the past. While it is mathematically possible to propagate a current TLE to any date in the past, the accuracy quickly suffers, and therefore it would be very useful to be able to dynamically change which TLE to use for modeling based on the selected date. Because the focus for OPTASAT is active current operations, the current TLE has been sufficient so far for the primary use case.

OPTASAT will need an explicit Application Programming Interface (API) to be developed. To date, the program has been made by a single individual who is directly familiar with all the internal structures, but if the intent is for more users to be able to make modules which will integrate with the rest of OPTASAT, they will highly benefit from text-based descriptions of how to use the software and construct the desired interfaces. For the time being, it is hoped that the provided example modules will be enough for users to get their custom modules up and running.

User testing is a key future goal for OPTASAT. While the needs of users have been anticipated based on personal experience, assumptions have been made about the needs and abilities of the operators.

Using OPTASAT in operation of a flight mission is a major future goal. Using the software on a real mission, no matter how simple, will be beneficial for proving the usefulness of the software.

It may eventually be beneficial to extend OPTASAT's functionality to other planets, increasing scope to manage changing reference frames, propulsive orbit corrections, orbital captures, gravity assists, and everything else that comes with interplanetary travel. For the foreseeable future, these capabilities are out of scope, but are not outside any technical possibilities.

Finally, it may be prudent to include security features within OPTASAT, either to ensure that OPTASAT itself is secure, or to manage security of the spacecraft itself.

CONCLUSION

OPTASAT provides a flexible, approachable operations interface for small satellites, with many of the basic tools needed to operate a mission. Its functionality addresses a growing need for tools for new spacecraft operators. OPTASAT will continue to grow and be modified to suit the shifting priorities of these operators.

OPTASAT will be released through GitHub, under the MIT License, which allows for any person to modify, reproduce, and in all other ways re-distribute the software in whatever way they want, for any use. No attribution is required, though this paper may be cited if a user prefers.

Acknowledgements

The authors would like to thank Dr. Mary Knapp and Dr. Richard Linares for their help and guidance with the development of this software.

REFERENCES

1. Lemmer, Kristina. "Propulsion for CubeSats." *Acta Astronautica* 134 (2017): 231-243.
2. *Bray, T. (December 2017). Bray, T (ed.). "The JavaScript Object Notation (JSON) Data Interchange Format". IETF. doi:10.17487/RFC8259*
3. *Rhodes, Brandon. "Skyfield: High precision research-grade positions for planets and Earth satellites generator." Astrophysics Source Code Library (2019): ascl-1907.*
4. Kelso, T. S. "Validation of SGP4 and IS-GPS-200D against GPS precision ephemerides." (2007).
5. Morgan, Rachel E., et al. "On-orbit operations summary for the Deformable Mirror Demonstration Mission (DeMi) CubeSat." Adaptive Optics Systems VIII. Vol. 12185. SPIE, 2022.
6. Weisstein, Eric W. "Equirectangular Projection." From *MathWorld*--A Wolfram Web Resource. https://mathworld.wolfram.com/EquirectangularProjection.html
7. Serra, Paul C., et al. "CubeSat Laser Infrared CrosslinK Mission Status." *International Conference on Space Optics—ICSO 2020*. Vol. 11852. SPIE, 2021.