CUBENAV: A FLIGHT DYNAMICS TOOL TO SUPPORT GUIDANCE AND NAVIGATION OPERATIONS OF DEEP-SPACE CUBESATS

Julia Muylle[(1)], Alessandro Morselli[(1)], Marco Lombardo[(2)], Alfredo Locarini[(2)], Luis Gomez Casajus[(2)], Marco Maggi[(2)], Marco Zannoni[(2)], Francesco Topputo[(1)], Valeria Cottini[(3)], Simone Ciabuschi[(3)], Silvia Natalucci[(3)]

[(1)]*Politecnico di Milano, Via La Masa 34, 20156 Milano, Italy, +39-02-2399-7157, {julialisa.muylle, alessandro.morselli, francesco.topputo}@polimi.it*
[(2)]*Università di Bologna, Via Fontanelle 40, 47121 Forlì, Italy, +39-0543-374437, {marco.lombardo14, alfredo.locarini, luis.gomezcasajus, marco.maggi7, m.zannoni}@unibo.it*
[(3)]*Italian Space Agency (ASI), Via del Politecnico snc, 00133 Roma, Italy, +39 06 8567917, {valeria.cottini, simone.ciabuschi, silvia.natalucci}@asi.it*

ABSTRACT

On ground satellite operations represent a significant part of mission development, both in terms of cost and time. This is particularly true for CubeSats, which are intended as low-budget satellites. The need of tools that could minimize the involvement of operators is therefore evident. CubeNav is a project aimed at developing a Flight Dynamics infrastructure tailored for supporting GNC operations of interplanetary CubeSats. The use of such a tool should ease the planning, development, engineering, and validation of Flight Dynamics operations and procedures, thus reducing the need of intervention by satellite operators. This work presents the overall architecture of CubeNav, with a focus on the modules being implemented and on the level of interaction between the user and the software blocks interfaces and outputs. The software is based on ESA's GODOT software and is developed in a modular fashion, to ease the automation of procedures to perform operations and analyses. CubeNav includes a Graphical User Interface, which allows the user to manage the execution of tools, provide input data as well as visualizing and interacting with outputs. The GUI provides a practical and fast way of performing necessary flight dynamics computations and visualizing the results in effective graphs and a timeline with all relevant events.

1 INTRODUCTION

The space exploration is moving towards a new paradigm, by exploiting the lower cost of miniaturized platforms: the interplanetary CubeSats. The reason behind this interest is that modularization and miniaturization of spacecraft components lead to a significant reduction of production, integration and launch costs. CubeSats are small and economical, but still capable of performing many different mission tasks [3], as already proven for near-Earth orbits. Nevertheless, the current modus operandi can hamper this momentum: while the system development costs scale with its size, the same is not true for flight dynamics operations, which are still expensively performed from ground, so requiring personnel and ground assets, which - at this pace - will soon saturate. The use of automated and advanced flight dynamics software could help in reducing the need of human effort and therefore the cost of operations. In this frame, the CubeNav project has been developed as an integrated flight dynamics infrastructure to support navigation operations for deep-space CubeSats. The project is

funded by the Italian Space Agency (ASI), as part of the ALCOR program [6], and developed as a joint collaboration between the Radio Science and Planetary exploration Lab of the University of Bologna (Unibo) and the Deep-Space Astrodynamics Research and Technology (DART) Lab of the Politecnico di Milano. Both teams have been involved in different deep-space CubeSats missions. Namely, Unibo performed the navigation of LICIACube [5] and ArgoMoon [7] while the DART Lab is leading the phase B study of LUMIO [1] and has performed the mission analysis and Guidance, Navigation and Control (GNC) development of HERA's Milani CubeSat [4].
This paper will present the architecture of the CubeNav software and the associated benefits that can be derived. In Section 2, an overview of the software's objectives and utilities is presented, together with the underlying third-party software and external dependencies exploited. In Section 3, the overall software architecture is described, focusing on the implemented modules and on the level of interaction with the user and with the interfaces to other software. In particular, the architecture is developed in three layers: Mission Analysis, Interface and Flight Dynamics. Additionally, the Graphical User Interface (GUI) is presented as the envelop which allows the user to interact with the software. In section 4, the current status of development of the software and the planned developments are presented. Finally, concluding remarks are provided in Section 5.

2 THE CUBENAV PROJECT

The CubeNav project is aimed at developing a Flight Dynamics software to support operations of deep-space missions. The software development leverages the expertise of the two involved research groups to provide an integrated infrastructure to support GNC operations: in particular, the development of the navigation analysis tools is performed by researchers of Unibo, while the DART Lab is in charge of the implementation of the guidance ones. CubeNav is therefore intended as a tool that will ease and automatize flight dynamics operations and procedures. The tool is expected to be fully implemented and tested by January 2024. All the underlying codes for guidance and navigation analysis have already been developed, while the user interface is still in development, in particular for integration of guidance and navigation solutions. The existing tools are currently being tested with the use of unit tests and regression tests, while the GUI will be tested through functional tests.
The benefits that the use of such a tool could bring regard the cost of operations performed manually, the time spent by the operators and the precision of the results, especially for small satellites. The CubeNav software generates as outputs all the parameters and mission details relevant to navigation operations, which could bring to a reduction of the learning curve to perform GNC analyses. This also contributes to requiring a lower need of support from operators, thus reducing the cost of operations and the chances of human error, as well as enabling a faster scenario setting and data analysis.
The software has been designed considering as key drivers the code maintainability and modularity in order to ease its development and validation. The modular organization of the software is performed such that similar functionalities are grouped together or collected in libraries. The tools and applications shall then make use of developed libraries. Furthermore, the dependencies from external software are limited to the minimum necessary and restricted to open source software. Finally, the software coding is standardized such that the architecture of the software is coherently organized between the two research groups and unit tests are implemented in parallel with the libraries and performed on each of them independently.

2.1 External dependencies

The CubeNav software is intended for use during astrodynamics and small satellites operations. This means that the software shall guarantee high standards in code quality, reliability, and ease of maintenance. To this aim, it is extremely important to carefully select the external software to be used

within the project, in particular for its C++ modules. While many external libraries with advanced features might exists, it is necessary to make use of stable, mature libraries which can therefore guarantee backward compatibility among versions and API stability. This is key to avoid the burden of constantly update the project's source code to keep track of changes in external dependencies or being forced to stick with an older version of such libraries (a non trivial issue for an astrodynamics software, whose expected lifespan can easily exceed 10 years). Therefore, the selection of third party software has been performed according to the following criteria:

- minimize external dependencies, allowing only those that are actually critical for the software.
- avoid the use of proprietary or closed-source software, favoring instead the adoption of those libraries which are released under open-source license such as the Lesser General Public License (LGPL) or more permissive license.
- favor the use of validated libraries for critical computations.

The third party software used in CubeNav can be divided in two categories. The first set includes libraries for arguments parsing (TCLAP[1]), parsing and generation of input and output files (yaml-cpp[2]).
The second category groups the software dedicated to the astrodynamics computations. These are ESA's GODOT[3] and NASA JPL's SPICE[4] which are employed for the generation of orbital models and observations model. GODOT is a flight dynamics software developed by ESA/ESOC that performs computations for estimation, optimisation and analysis of spacecraft orbits. GODOT is composed by many libraries, developed in C++ and Python, which can be used by the user to create their own solutions. In CubeNav, GODOT is used in its C++ version for analyses of spacecraft orbits through computation of astrodynamical events, preliminary Orbit Determination and residuals processing. The SPICE system is instead used for retrieving celestial bodies ephemeris in trajectory optimization tools and as an alternative to GODOT for orbital events computations to allow the maximum flexibility for the user.
For what concerns the Python modules, the same guidelines have been followed. The use of the GODOT Python interface[5] already brings in as transitive dependencies a comprehensive set of common Python modules, (e.g. `numpy`). In addition, the following modules are direct dependencies of CubeNav Python modules: `matplotlib`, which is used for graphical representations of orbits, and `PySide6`[6] for the realization of the Graphical User Interface of CubeNav.

3 CUBENAV ARCHITECTURE

The CubeNav software is organized in four layers, which represent the four main functional blocks of the software. A schematic representation of CubeNav layers organization is presented in Figure 1.

[1] https://tclap.sourceforge.net/. Last visited on: 17/05/2023.
[2] https://github.com/jbeder/yaml-cpp. Last visited on: 17/05/2023.
[3] https://godot.io.esa.int/docs/1.2.0/. Last visited on: 17/05/2023.
[4] https://naif.jpl.nasa.gov/naif/toolkit.html. Last visited on: 17/05/2023.
[5] https://godot.io.esa.int/godotpy/. Last visited on: 17/05/2023
[6] https://doc.qt.io/qtforpython-6/. Last visited on: 17/05/2023.

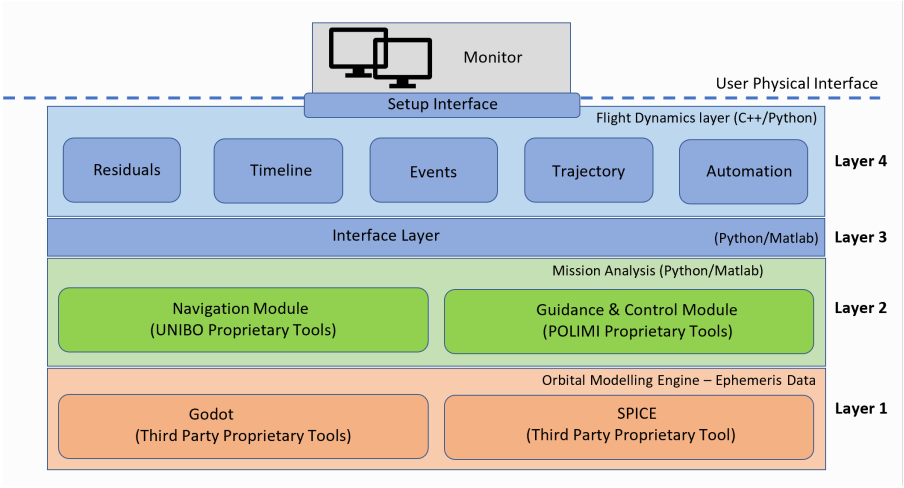


Figure 1: CubeNav architecture organization in layers.

The first layer is the Orbital Modelling Engine Layer and it is the only layer which is not directly implemented within the CubeNav project. This layer is composed by the third party software that provide an extensive and comprehensive orbital and observation model to perform trajectory design and reconstruction. The two software used by CubeNav could be ESA's GODOT or NASA JPL's SPICE. The second layer is the Mission Analysis layer, which contains the actual low-level procedure to generate the outputs to be visualized in the upper layers. This layer is composed of several libraries that are property of Unibo and Polimi, or third party, and it can be further subdivided in two main processing modules: the Navigation Module and the Guidance and Control (GC) Module. The third layer is the Interface layer, which represents the interface with the outputs obtained from the lower layer to convert them in suitable formats, to be used for the flight dynamics computations in layer 4. The last layer is the Flight Dynamics layer, which represents the layer that actually executes all flight dynamics procedures to get the desired outputs. All these layers are managed by the Graphical User Interface, which represents the tool which the user will interact with. The GUI has a twofold objective: on one hand it allows the definition of user-defined inputs and selection of desired outputs, on the other hand it provides graphical and text representation of the results obtained from computations.
The architectural flow followed by the software is described in the following and graphically represented in Figure 2. The definition of input data, orbit file and environment set-up by the user is performed through the GUI. In particular, if the orbit file is not available, this can be generated through layers 2 and 3 with the available optimizer, or with other external software. The selection of the environment file is then transferred to layer 1, where the universe and ephemeris necessary for computations are uploaded and set up. The defined inputs and orbit file are then used by the Flight Dynamics layer to retrieve the user-selected outputs. Here, the code segments that are run depend on the desired output selected by the user. Different solvers might be used, in which case they are run in parallel. Once all the computations are completed, the output file is created, and, if selected by the user, the graphical representation is shown in the GUI.

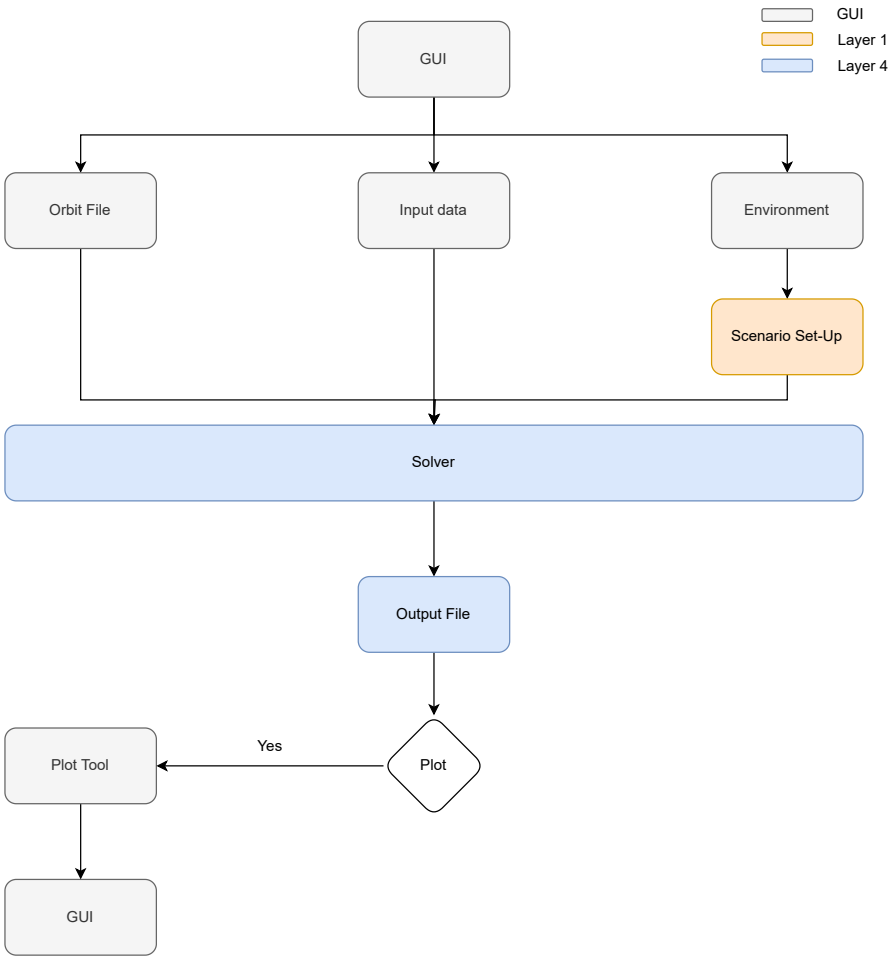


Figure 2: CubeNav architecture flow. The colors refer to the corresponding layer, as defined in Figure 1.

3.1 Mission Analysis layer

The mission analysis layer represents the layer where the outputs necessary for flight dynamics analyses are computed. This is divided into guidance and navigation. In the guidance module the trajectory is computed and optimized, while the navigation module performs orbit determination and flight path control. This layer is based on third party software, mainly Polimi and Unibo proprietary software.

3.1.1 Guidance

The GC module collects the Polimi's software to perform trajectory design and optimization. The four software available are ULTIMAT (Ultra Low Thrust Interplanetary Mission Analysis Tool), LT2O (Low-Thrust Trajectory Optimizer), DIRETTO (DIREct collocation Tool for Trajectory Optimization) and the Milani Mission Analysis Pipeline. The first [2] is a software meant to provide an analysis and simulation suite for low thrust interplanetary scenarios, implemented in MATLAB and fully integrated with the NASA's JPL SPICE Toolkit. The software is aimed at performing impulsive trajectory optimization into highly nonlinear models, where the design is constrained from very limited control authority. ULTIMAT is composed by two main modules: the Design module implements trajectory optimization techniques in a high fidelity model, while the Assessment module assesses the flyability of such trajectories. In particular, the trajectory design embeds the possibility of finding periodic and quasi-periodic orbits, optimize impulsive transfer trajectories with a multiple-burn

multiple-shooting technique, transform impulsive solutions into a finite-burn equivalent, and perform station keeping analyses. The assessment of the trajectory is instead performed through a number of hierarchical tasks ranging from preliminary geometrical checks to detailed navigation analyses: in particular, the outputs that the software can provide are a pre-processing of the solution evaluating mission requirements against celestial events and linking constraints; visibility windows and related radiometric measurements computation; sensitivity analysis against variations of low-thrust maneuver timing, duration, magnitude, and pointing angles; Orbit Determination (OD) and Covariance Analysis (CA); quantification of Navigation Cost (NC). LT2O [2] is a software developed for low-thrust trajectory optimization of transfer with multiple revolutions with indirect methods. LT20 handles time-, radiation-, energy-, and fuel-optimal problems in a MATLAB-native environment, and it implements simple dynamics models, such as the two-body model with J2 perturbation in cartesian coordinates and Modified Equinoctial Elements (MEE) and the restricted three- and four- body models. The tool implements sophisticated hybrid techniques for low thrust trajectory optimization, such as continuation methods, smoothing techniques, analytical derivatives and accurate switching detection system that allows to conduct end-to-end optimizations for transfers with up to 500 revolutions. DIRETTO [2] is a MATLAB software for optimization of low thrust trajectories using direct transcription method to solve the optimal control problem: the state and control variables are discretized and the optimal control problem is converted to a nonlinear programming problem (NLP). In particular, three different collocation methods are implemented, thus embedding different ways the state and control variables are satisfied and how the dynamics constraints are fulfilled: these are Hermite-Simpson, Gauss-Lobatto and Pseudospectral methods. DIRETTO offers considerable flexibility to accommodate both satellite and operational system constraints. It can be used with high fidelity models to derive solutions for short and medium duration transfer phases. For example, it has been successfully used in solving low-thrust Earth-Mars transfers with ballistic capture [2], where the optimization involves both the heliocentric transfer and ballistic capture point targeting. The Milani Mission Analysis Pipeline [4] is a tool written in MATLAB for trajectory design and mission analysis around small celestial bodies, such as asteroids. The tool is highly automated and can perform trajectory design using waypoints strategy and landing design, as well as analyses such as orbital event computation, knowledge analysis, dispersion analysis and contingency analysis.

3.1.2 Navigation

The navigation module gathers a set of University of Bologna proprietary tools that aid the orbit determination and the flight path control processes. The orbit determination allows the estimation of a spacecraft trajectory given a set of observables and a dynamical model. If the estimated trajectory differs from the reference one, flight path control process optimizes and computes orbital maneuvers to correct these differences.

3.2 Interface layer

The Interface layer is the connection between the Mission Analysis layer, where trajectory and navigation optimizations are performed and the Flight Dynamics layer, where the outputs of the Mission Analysis layer are used to perform main flight dynamics computations. The outputs of the tools used in the Mission Analysis layer are provided in different formats depending on the tool used. In order to use them as inputs to the Flight Dynamics layer, it is necessary to convert them in a set of standardize interface files, which can be used within GODOT and SPICE modules or used to exchange information with external partners. The use of standardized files is also useful in order to avoid proliferation of implementation-dependant and customized output, as the standards defined in CubeNav will be taken as reference for any trajectory optimization tool developed in the future. As a result, the

software will be modular and it will be possible to plug-and-play new software or to interface other third-party tools. The codes to perform this conversion are written in MATLAB for the conversion of outputs from the guidance module and Python for the conversion of outputs from the navigation module. For the conversion of outputs from the trajectory optimizer, these are first saved into two MATLAB structures, which have the same configuration for all different optimization tools. The first structure contains the data of trajectory, namely the spacecraft name and ID, the time span, the spacecraft state (position and velocity vectors) and mass at all time epochs and the control vector in magnitude and direction, while the second one contains the settings, in particular the reference frame name and center. The data and settings structures are then used to create spk binary kernels, containing the spacecraft ephemeris. The same two structures can also be used to create an *OEM* orbit file with the metadata containing the spacecraft characteristics and the data section with the trajectory epochs and ephemeris, and a timeline file containing the trend for thrust ignition: in particular, the initial and final epoch for each thrust ignition are reported. Alternatively, the interface layer for the navigation module exploits JSON files to exchange the desired outputs. This general exchange format supports all the outputs of the navigation layer that mainly are conformed by residuals and time series. The routines to perform the data exchange are generated with a series of Python scripts.

3.3 Flight Dynamics layer

The Flight Dynamics layer represents the set of codes that provide all the relevant outputs necessary for generating and displaying the processing results. In particular, the layer can be subdivided in five main modules, designed to facilitate real-time insights on the navigation results and on critical parameters that affect operations. The modules implemented are: astrodynamical events, trajectory representation and comparison, navigation residuals, timeline for relevant events visualization, and automation.

3.3.1 Event tool

The Event tool collects all the codes necessary to compute different astrodynamical events, given a reference trajectory and a reference time span. The events considered are seven and are referred to different celestial bodies or stations. The first tool is the eclipse generation tool, which generates the list of eclipses with respect to one or more defined celestial bodies during the trajectory. The tool is able to distinguish between total, partial and annular eclipses: for each of them the initial and final epoch of the eclipse are provided as output. A similar tool is the bodies occultation tool, which identifies the time epochs where a celestial body is occulted by one or more other defined bodies with respect to the reference trajectory. The third tool is the altitude tool, which computes the altitude of the spacecraft with respect to the surface of a reference body and compares it with a user-selected value. The outputs are therefore the time instants when the spacecraft is at a given altitude with respect to the reference body. Similarly, the range tool computes the distance between the spacecraft and the center of a reference body and identifies the local maxima and minima, therefore the time instant when the range rate is null. The elevation tool computes the elevation with respect to a defined ground station and reports the instants when the elevation crosses the minimum value for the spacecraft to be visible from the ground station. The conjunction tool computes the phase angle with respect to one or more reference bodies to identify possible conjunctions windows, when the phase angle is below a user-defined minimum. Finally, the illumination tool computes the illumination factor defined by a selected source body and an occulting body. The event tool is identified by the epoch when the illumination factor is null, which correspond to total eclipse. The graphical representation of the events can be of two main types. The first is represented in Figure 3, where an example is provided for the case of the JUICE spacecraft eclipses with respect to Jupiter and Ganymede. The different colors and

height of the columns identify the different eclipses types. The plot has similar configuration also for occultations, where only one type of occultation is contemplated.

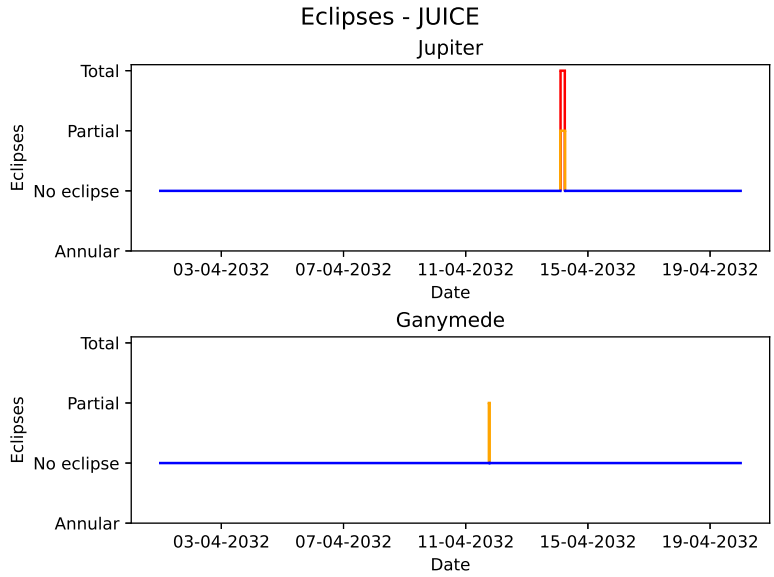


Figure 3: Graphical representation of the eclipses events of JUICE trajectory test case with respect to Jupiter and Ganymede.

A second kind of graphical representation is the one reported in Figure 4, where the altitude with respect to Jupiter and Europa is represented for JUICE trajectory. In this case the altitude is computed for every epoch in the time span and the intersection with the fixed input values represented by red lines is evaluated. Similar plots are also identified for the elevation, range and conjunction events, while the illumination fraction and range events plots only represent the values of the phenomenon under consideration, without the fixed value lines.

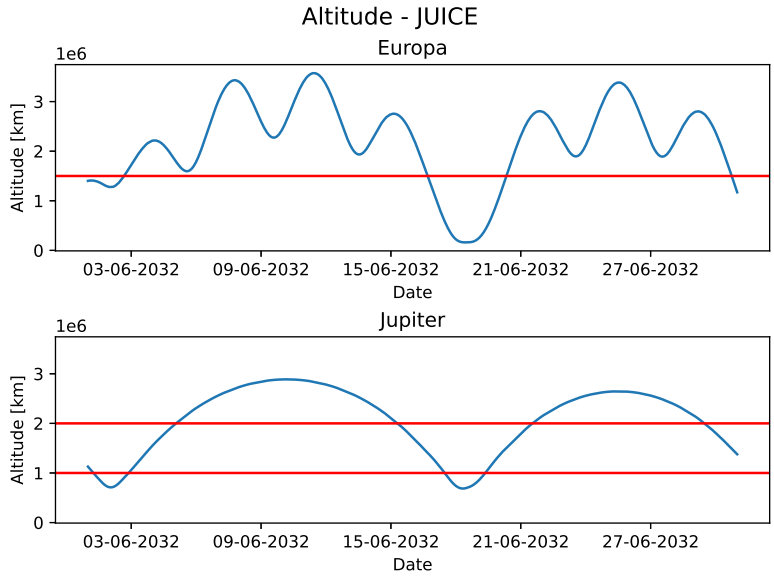
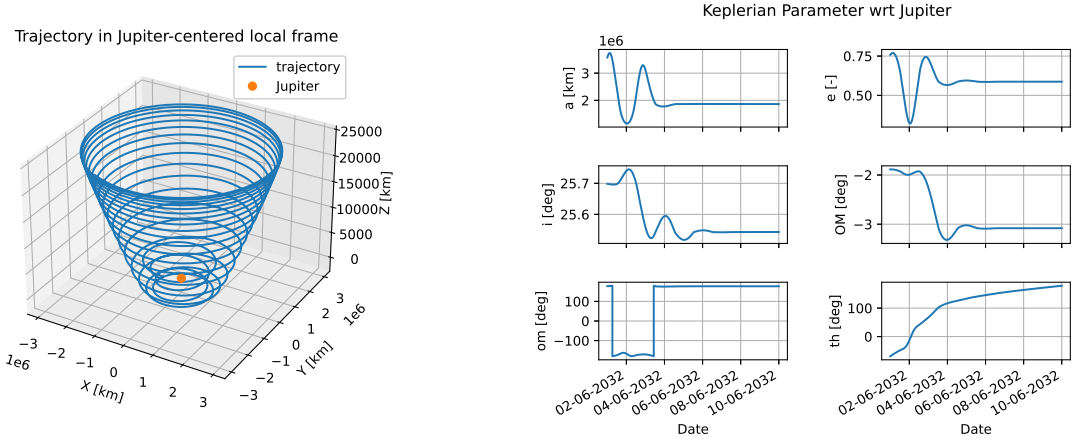


Figure 4: Graphical representation of the fixed altitude events of JUICE trajectory test case with respect to Jupiter and Ganymede.

3.3.2 Trajectory

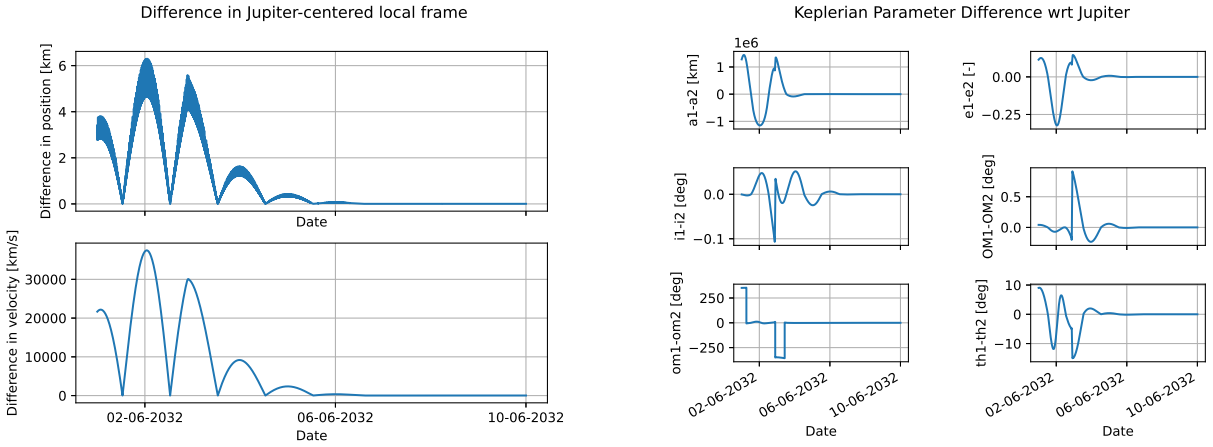
The Trajectory tool contains the tools necessary to handle the reconstructed and predicted spacecraft state. In particular, the tool embeds the graphical representation of the trajectory and the trajectory comparison tool. The first is the tool to generate the plot of the trajectory, given a certain orbit file. This can be represented in a user-defined reference frame and both in keplerian and cartesian parameters. The trajectory comparison tool instead requires two orbit files, representing the trajectories to compare. The comparison is made as an absolute difference between the parameters of the two trajectories. In particular, if cartesian coordinates are selected as orbital parameters, the norm of the difference in position and velocity is shown, while if keplerian parameters are selected, the difference for each keplerian parameter is represented. An example of a portion of JUICE trajectory in local frame with respect to Jupiter is reported in Figure 5 in cartesian and keplerian coordinates, while in Figure 6, the comparison between two different JUICE orbit file is represented.



(a) Cartesian coordinates

(b) Keplerian elements

Figure 5: Trajectory representation in Cartesian and Keplerian coordinates. A portion of JUICE trajectory in the local reference frame is used as an example.



(a) Cartesian parameters

(b) Keplerian parameters

Figure 6: Trajectory comparison in Cartesian and Keplerian coordinates. A portion of JUICE trajectory from two different orbit file is used as an example.

3.3.3 Timeline

Timeline is a tool that provides an interactive graphical representation of a chronological sequence of events selected by the user. Timeline supports the user in planning and controlling the flight activities, in particular those related to navigation. For example, events that can be represented by the timeline are tracking passes, orbital maneuvers, Data Cut-Offs (DCOs), flight dynamics operations shift, occultations, scientific events. At the current state of works, the events set can be provided by the user using JSON formatted files or they are automatically retrieved from the Event tool described in Section 3.3.1. In addition, a dedicated function has been implemented to plot the actual tracking passes of Deep Space Network (DSN) through a query of the Service Preparation Subsystem (SPS[7]) web interface. The Timeline graphical representation in Figure 7, foresees a grid where each row is a set of events of same origin (i.e., orbital maneuvers) while the columns identify the timestep of the timeline. Also, an interactive time ruler is drawn on top of the grid showing the date and time. Furthermore, by clicking the middle mouse button on the grid, a time cursor is drawn with the aim of accurately showing the time in a selected location of the timeline. The grid covers the time interval configured by the user on a dedicated control panel where the timestep can also be chosen. Timeline rows are added by the user on a panel on the left side of the grid, while the events set to be drawn on the row is selected via a drop-down list. The events are drawn as rectangles that can be selected and clicked by the user with the aim of displaying the details of the events or calling up an associated function of the Event tool (i.e., plot the elevation of a spacecraft during a tracking pass).

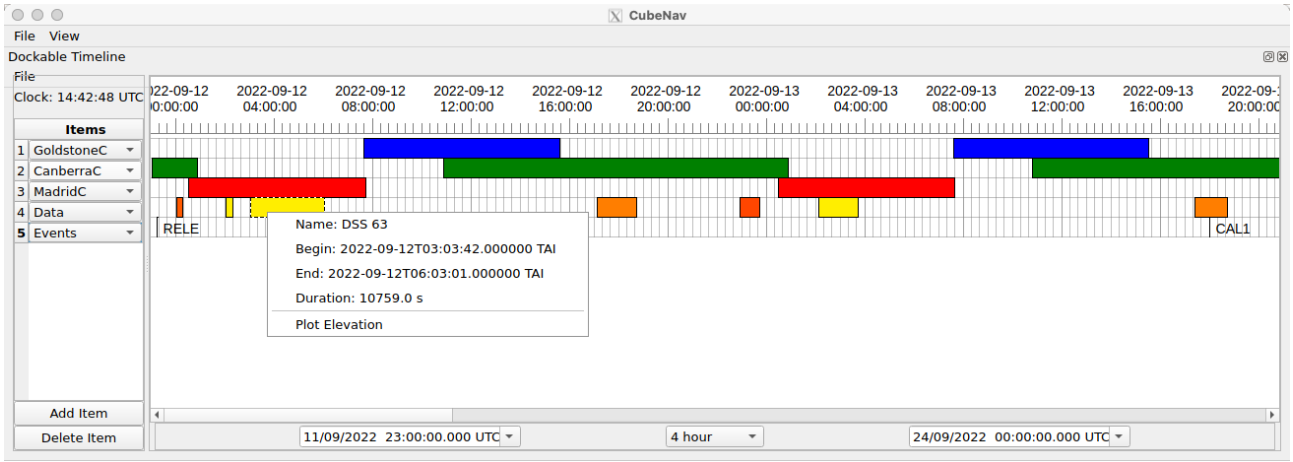


Figure 7: Graphical representation of the events schedule provided by the Timeline tool using LICIACube data (coverage from Goldstone, Canberra, and Madrid on rows 1 to 3, actual tracking passes on row 4, and several mission events on row 5).

3.3.4 Residuals

The residuals tool allows to perform a quick validation of the estimated trajectory, by providing a graphical representation of the retrieved residuals as represented in Figure 8. Pre-computed residuals with the Navigation layer tools can be directly loaded into the GUI using JSON formatted files. Alternatively, the tool allows to perform a quick pass-through using real radio-tracking measurements, namely range and range-rate observables. The pass-through is performed by using the GODOTPy library that retrieves the computed observables given a trajectory provided by the user. The user can select and edit the displayed residuals when necessary, for example in case of outliers. The

[7] https://spsweb.fltops.jpl.nasa.gov/rest/Wrapper/Schedule/schedview.html

editing activity can be then exported as a JSON file containing the removed residuals. The interface allows the user to zoom on data as well as have a representation of the data coverage by complex or station. Thanks to the modular architecture of the software, Residual tool works in symbiosis with the Timeline tool, synchronizing the time cursor of the timeline represented as a vertical line on the residual viewer.

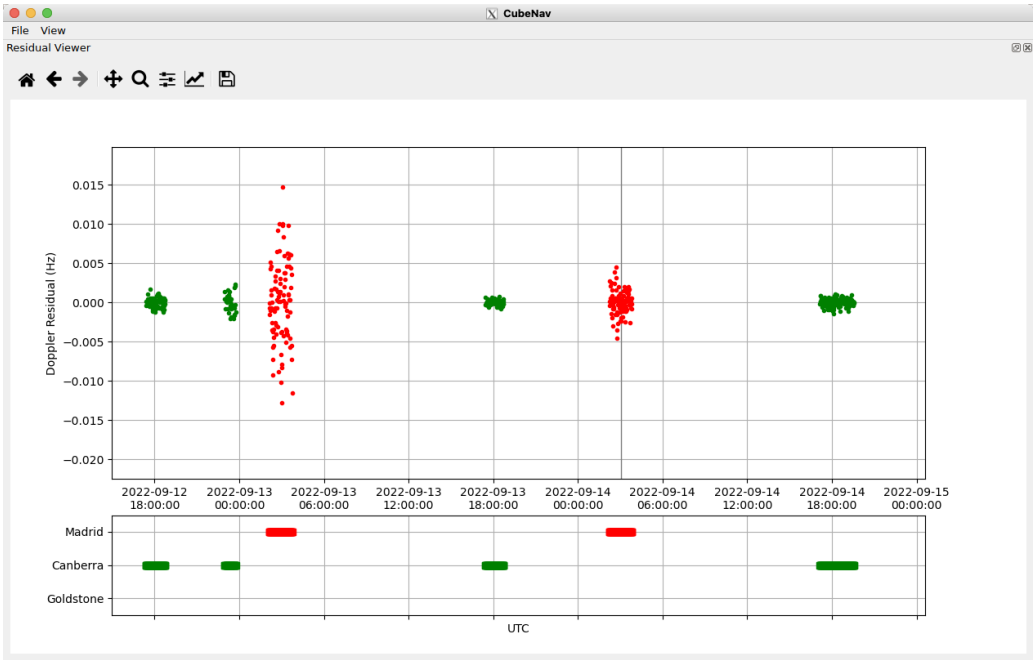


Figure 8: LICIACube residual data visualization using the residual viewer tool (the vertical gray line identifies the time cursor of the Timeline tool).

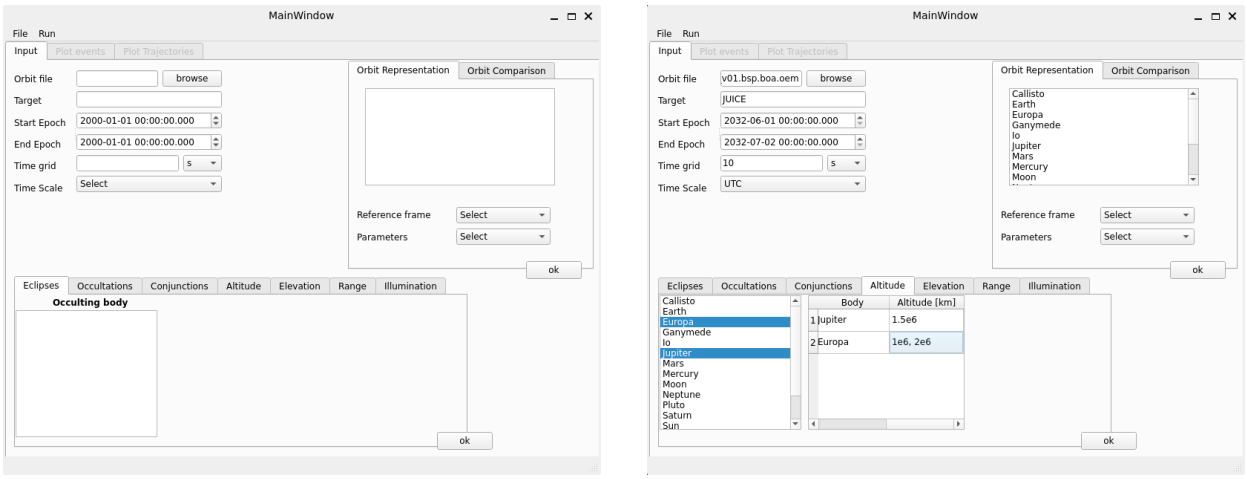
3.3.5 Automation

During the different mission phases, many analyses and checks are performed which largely involve repetitive tasks, starting from the analysis and processing of the radiometric data, the optimization of the spacecraft trajectory, and the subsequent Verification and Validation (V&V) of flight dynamics products and generation of official interfaces. This means that the generation of the products requires the execution of a predefined sequence of (complex) algorithms and tools, processing of results and extraction of relevant figures that are then compared against thresholds and bounds extracted by mission requirements and operative constraints.
The architecture of CubeNav is therefore designed having in mind the possibility to largely automate the use of such tools and routines for use during operations. The first key element is the presence of a single shared environment file where all the information concerning ephemerides, spacecraft characteristics, and GODOT setup are defined. The location of the file is defined within the the CubeNav configuration module, which accesses a predefined location or retrieves the environment file from a dedicated environment variable. This ensures that all tools will run with the same basic configuration, which is fundamental to guarantee a full compatibility among the results obtained with different tools.
The second key element to enable automation is the use of text input files in YAML or JSON format. This allows the possibility to use scripting languages (e.g., bash or Python) to manipulate such files according to mission-phase dependent parameters. For instance, the current date and time span needed for events generation can be directly substituted depending on the current date and time. Additionally,

the blocks structures are shared among different files (e.g., those defining the orbit file path or state vectors). The generation of input files can therefore be simplified and generalized, manipulating and assembling block templates through a scripting language. The same standardization applies to the output files. This enables common modules for parsing such files and, by exploiting GODOTPy event intervals computations, allows to easily define and assemble checks and tests as Python modules and programs. As an example, by combining the optimization outputs and visibility from ground stations, it is possible to check that manoeuvres are executed during passes.
Another important aspect that enables automation is the possibility to split and monitor the execution of all steps. CubeNav is designed in such way that generic and mission-specific computations are well defined and separated in programs and scripts, each providing its own output in a clear folder structure. By implementing a job scheduler it will be therefore possible to run tasks in a predefined sequence, prioritize critical tasks, handle dependencies, and running independent pipelines in parallel whenever possible. This rather flexible implementation has allows to recover the execution in case an intermediate step fails. Lastly, the key feature that will be added to the automation layer is a dashboard to summarize the execution status and outcome of the V&V checks.

3.4 CubeNav Graphical User Interface

The Graphical User Interface (GUI) is the direct interface available to the user. It has a twofold purpose: through this window the necessary inputs are provided by the user, as well as the desired outputs and the results are shown once all the computations have been performed. In Figure 9, a picture of the GUI is shown in its initial configuration and an example of input selection is reported.



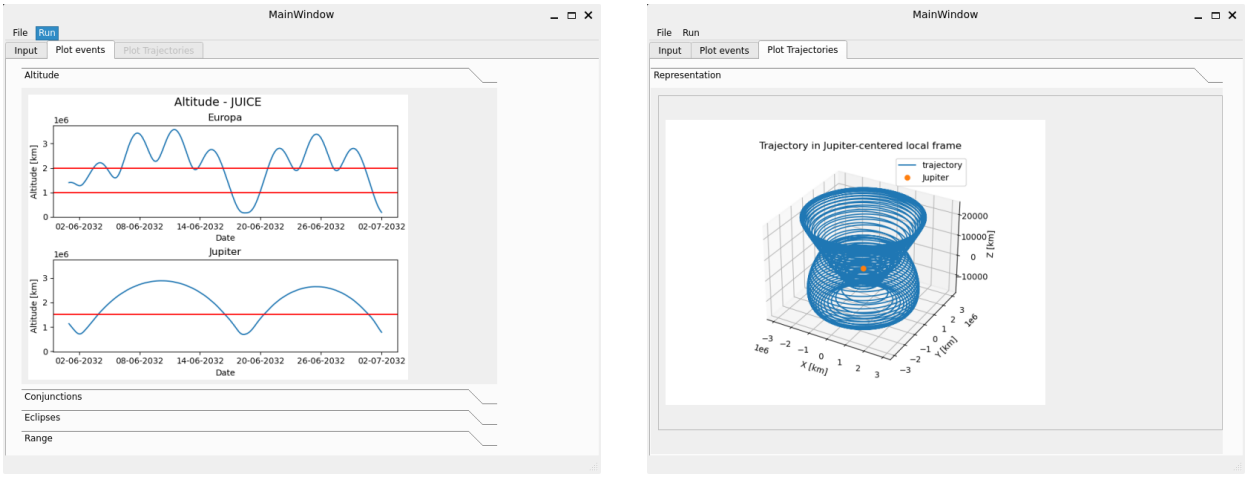
(a) Initial GUI layout

(b) GUI layout with inputs definition

Figure 9: GUI layout. On the left the initial user interface, on the right the interface after environment loading and input definition by the user.

The *File* menu provides an interface to load the environment and save the results. As it can be noticed, the selection of celestial bodies or stations for events computation is done through a list, which is filled up once the environment file is loaded. The environment file contains the universe and the kernels necessary for the spacecraft trajectory under evaluation. The first tab *Input* collects the common inputs necessary to define the portion of the trajectory under investigation: this includes the spacecraft orbit file and name and the time span to be considered in the analysis, which includes initial and final epochs, time grid and time scale considered. Moreover, the definition of the desired outputs is also placed in the same window. In particular, two separated boxes are present for events computation and trajectory representation. In the event computation table, the configuration of each

event has a different setting, based on the inputs necessary, especially if fixed values of the quantity under investigation are needed. For all the events, the possibility of selecting different bodies and of specifying different fixed values for each body is included. The second box is the one related to trajectory representation and trajectory comparison. Here the inputs requires are the reference celestial body, the reference frame (local or inertial) and the desired orbital parameters (Keplerian or Cartesian). In the orbit comparison tab the second orbit file to be compared to the reference one is also required. When the common fields and the fields relative to at least one event or orbit representation type are correctly filled in, the *Run* button activates. The *Run* button embeds three possible options: the computation of the events, the computation and plot of the events and the plots of trajectory/trajectory comparison. When the input fields are all correctly filled in, the input files necessary for computations are created as JSON files. The *Run* button, then, starts the events or trajectory computations, which are performed all at once. Once the computations completed, other tabs are activated, depending on the run process selected. The results of the events computation are stored as JSON files in the result folder initially selected. The graphical representations are instead grouped in the *Plot events* and *Plot trajectories* tab, as represented in Figure 10. If more than one event was selected, the list of events is reported in the tab and the one to be visualized can be selected.



(a) Results for event

(b) Results for trajectory

Figure 10: GUI results layout.

4 ONGOING DEVELOPMENTS AND FUTURE WORK

The CubeNav software is still currently under development and testing. The project is a 18 months agreement, supposed to be concluded by January 2024. The current stage of development features a completed implementation of flight dynamics procedures. In particular, layer 2 is only composed of proprietary software, already developed by the two research teams and has been fully connected to the CubeNav project through the interface layer. Layer 4 has instead been fully implemented within CubeNav in the past months. This layer is currently under testing: unit tests have already been performed to validate event functions and configuration file, while regression tests are currently under development to validate all portions of codes implemented. The GUI implementation is still not concluded: two versions of it are currently present, one for navigation computations and one for guidance events computations, but they still need to be merged in a single user interface from which all computations can be performed. Once the final GUI is assembled, functional tests will be performed to assess its correct functioning.

5 CONCLUSION

This paper provides a description of the CubeNav software architecture and characteristics. The tool is organized in a modular fashion, by grouping together in different layers and modules segments of code with similar functionalities. This represents CubeNav strength in terms of adaptability of the software to different optimization tools such that it can be used in combination with other third-party software and for future automation of the operations tasks. Indeed, the CubeNav architecture embeds an interface layer which is able to convert outputs from different optimization tools to suitable inputs for flight dynamics computations. The tool represents an integrated flight dynamics software, which enables to obtain all mission details relevant to the navigation operations, thus raising the level of automation of flight dynamics operations. The segment of codes providing the necessary outputs for operations are all managed by a practical Graphical User Interface, which allows the user to easily define the necessary input parameters and select the desired outputs. The GUI also provides the interface where the results are shown to the user in text and graphical formats. Overall, the way of presenting and storing these outputs is beneficial both for the interaction with the operators and for allowing the generation of a schedule of automated activities and tasks for operation purposes. CubeNav is therefore an effective integrated tool, efficiently exploiting the knowledge and tools of two research groups and intended as a key tool for automation of flight dynamics operations for interplanetary CubeSats and consequent reduction of operations costs and errors.

6 ACKNOWLEDGEMENT

The work described in this paper was carried out as part of the ASI project F35F22000490005 for the development of the CubeNav software: operative navigation for deep-space CubeSat missions. The authors would like to acknowledge the support received by the Italian Space Agency (ASI).

REFERENCES

[1] A. Cipriano, D. Dei Tos, and F. Topputo, "Orbit Design for LUMIO: The Lunar Metoroid Impacts Observer," *Frontiers in Astronomy and Space Sciences*, vol. 5, p. 29, 2018.

[2] F. Topputo, D. Dei Tos, K. Mani, *et al.*, "Trajectory design in high-fidelity models," in *7th International Conference on Astrodynamics Tools and Techniques (ICATT)*, 2018, pp. 1–9.

[3] T. Villela, C. Costa, A. Brandao, F. Bueno, and R. Leonardi, "Towards the Thousandth CubeSat: A statistical Overview," *International Journal of Aerospace Engineering*, 2019.

[4] F. Ferrari, V. Franzese, M. Pugliatti, C. Giordano, and F. Topputo, "Preliminary mission profile of Hera's Milani CubeSat," *Advances in Space Research*, vol. 67, pp. 2010–2029, 6 2020.

[5] E. Dotto, V. Della Corte, M. Amoroso, *et al.*, "LICIACube-the Light Italian Cubesat for Imaging of Asteroids in support of the NASA DART mission towards asteroid (65803) Didymos," *Planetary and Space Science*, vol. 199, p. 105 185, 2021.

[6] G. Leccese, A. Fedele, and S. Natalucci, "Overview and Roadmap of Italian Space Agency Activities in the Micro- and Nano-Satellite Domain," in *73rd International Astronautical Congress (IAC)*, 2022.

[7] M. Lombardo, M. Zannoni, I. Gai, *et al.*, "Design and Analysis of the Cis-Lunar Navigation for the ArgoMoon CubeSat Mission," *Aerospace*, vol. 9, no. 11, p. 659, 2022.