75th International Astronautical Congress (IAC), Milan, Italy, 14-18 October 2024.

IAC-24,B6,1,4,x84822

CUBENAV: AN OPERATIONAL FLIGHT DYNAMICS TOOL TO SUPPORT GUIDANCE AND NAVIGATION OF DEEP-SPACE CUBESATS

Alessandro Morselli[a,*], **Marco Lombardo**[b], **Julia Muylle**[a], **Luis Gomez Casajus**[b], **Alfredo Locarini**[b], **Marco Maggi**[b], **Marco Zannoni**[b], **Valeria Cottini**[c], **Simone Ciabuschi**[c], **Silvia Natalucci**[c]

[a] *Dept. of Aerospace Science and Technology, Politecnico di Milano, Via La Masa 34, 20156 Milano, Italy*, {alessandro.morselli, julialisa.muylle}@polimi.it

[b] *Dept. of Industrial Engineering, Università di Bologna, Via Fontanelle, 47121 Forlì, Italy*, {marco.lombardo14, alfredo.locarini, luis.gomezcasajus, marco.maggi7, m.zannoni}@unibo.it

[c] *Agenzia Spaziale Italiana, Via del Politecnico snc, 00133 Roma, Italy*, {valeria.cottini, simone.ciabuschi, silvia.natalucci}@asi.it

CubeNav, a project funded by ASI within the ALCOR program, aims to develop a tailored Flight Dynamics infrastructure to support CubeSats' GNC operations in deep-space missions. Led by the Radio Science and Planetary Exploration Lab at the University of Bologna (Unibo) in collaboration with the DART Lab at Politecnico di Milano, CubeNav leverages the expertise of both teams, with Unibo focusing on navigation analysis and DART Lab on guidance tools. Both teams were involved in many deep-space CubeSat missions: Unibo navigated LICIACube and ArgoMoon, while DART Lab led the phase B study of LUMIO and contributed to the GNC development of HERA's Milani CubeSat. CubeNav aims to streamline the planning, engineering, and validation of Flight Dynamics operations, reducing human intervention. Eliminating repetitive tasks performed by the operators could substantially contribute in reducing the learning curve, human error, and time required for scenario setting and data analysis. This is pivotal to achieve cost-effective ground-based navigation of deep-space CubeSats.

The software architecture is outlined, focusing on its modular design, also detailing the external dependencies like ESA's GODOT. Three functional blocks constitute CubeNav: interface and data format conversion, flight dynamics, and the Graphical User Interface (GUI) The first layer standardizes communication between Unibo and DART proprietary tools. The flight dynamics layer, coded in C++ and Python, encompasses navigation and guidance tools, enabling orbit determination and trajectory analysis. The GUI serves as a user-friendly interface, facilitating parameter input, output selection, and data visualization, thereby aiding operator interaction and task automation. The GUI includes a tool dedicated to radiometric data processing and residual computations, pivotal for handling and analyzing mission-collected radiometric data, and for assessing navigation solution accuracy. Additionally, it offers visualization features to interpret processed data effectively. A timeline visualization is also available in the GUI, which is crucial for efficient mission planning and execution as it can display mission events such as station passes and maneuver times. The integration of all plots and tools via cursors provides operators a global overview on the current scenario, allowing prompt response to deviations from the planned trajectory, hence augmenting operations efficiency.

CubeNav's integrated tools enhance operational efficiency and effectiveness, emphasizing automation, adaptability, and task automation, thereby refining CubeSat navigation. Ongoing development focuses on fortifying the automation layer, underscoring CubeNav's significance in deep-space missions.

Keywords: CubeSats; Ground Systems; Flight Dynamics Software

1 Introduction

Space exploration is entering a new era by utilizing low-cost miniaturized platforms, specifically interplanetary CubeSats. The main reason for this shift is that modularizing and miniaturizing spacecraft components significantly reduces production, integration, and launch costs. CubeSats are small and cost-effective, yet they can still complete a wide range of mission tasks [1], as demonstrated in near-Earth orbits. However, the current approach may hinder this progress: while the cost of system development increases with size, the expense of flight dynamics operations does not, and these are still costly to perform from the ground, requiring personnel and assets that will soon reach capacity. Using automated and advanced flight dynamics software could help to reduce the need for human effort and, consequently, the cost of operations. To address this, the CubeNav project was developed as an integrated flight dynamics infrastructure to support navigation operations for deep-space CubeSats. This project is funded by the Italian Space Agency (ASI) as part

of the ALCOR program [2] and is 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. Specifically, Unibo performed the navigation of LICIACube [3] and ArgoMoon [4], while the DART Lab is leading the phase B study of LUMIO [5] and has performed the mission analysis and Guidance, Navigation, and Control (GNC) development of HERA's Milani CubeSat [6].

This article will delve into the design of the CubeNav software and its accompanying benefits. In Section 2, an overview of the software's objectives and functionalities will be presented, along with the third-party software and external dependencies that are utilized. In Section 3, the overall architecture of the software will be described, focusing on the individual modules and the level of interaction with the user and other software interfaces. Specifically, the architecture is divided into three layers: Mission Analysis, Interface, and Flight Dynamics. The . An overview of the developed Flight Dynamics tools is provided in Section 4, followed by a description of the Graphical User Interface (GUI) with a specific focus Timeline application 5. Lastly, the article will conclude with some concluding remarks in Section 6.

2 The CubeNav project

The CubeNav initiative is designed to create a Flight Dynamics software to facilitate deep-space mission operations. The development of this software relies on the combined expertise of the two research groups involved to provide a comprehensive infrastructure for supporting GNC operations. Particularly, the researchers at Unibo are focused on developing navigation analysis tools, while the DART Lab is responsible for implementing guidance ones. The purpose of CubeNav is to automate and simplify flight dynamics operations and procedures. The software first version was delivered in February 2024, at the formal end of the project. This includes the navigation and guidance analysis modules and the integrated user interface. All modules and tools were tested with unit and regression tests, while the graphical user interface has been tested via functional tests.

The use of CubeNav could offer numerous benefits, such as reducing the cost and time spent on manually performed operations, increasing the precision of results, especially for small satellites, and significantly lowering the learning curve for performing GNC analyses. This tool also reduces the need for operator support, thereby decreasing the cost of operations and the likelihood of human error. Additionally, CubeNav enables faster scenario setting and data analysis.

The software has been developed with code maintainability and modularity as its primary drivers, in order to facilitate its development and validation. The software's modular organization groups similar functionalities together in libraries, making it easier for tools and applications to use these libraries. Additionally, the software minimizes its dependence on external software to only the necessary open-source components. Finally, the software's coding is standardized, ensuring that the architecture of the software is coherently organized between the two research groups and that unit tests are implemented and performed independently on each library.

2.1 External dependencies

The CubeNav software must guarantee high standards in code quality, reliability, and ease of maintenance, given its intended use during satellite operations. It is therefore essential that the software functions without any disruptions, particularly during critical phases. Furthermore, the software will require ongoing maintenance for an extended period, given the typically lengthy mission preparation phases and mission duration associated with deep space missions.

It is of the utmost importance to exercise great care in the selection of external software to be employed within the project, with particular attention paid to the C++ modules. Although numerous external libraries with advanced features might exists, it is necessary to make use of stable, well-established libraries that can guarantee backward compatibility across versions and API stability. This is key to avoid the burden of constantly update the project's source code to track alterations in external dependencies or being forced to stick with an older version of such libraries (a significant challenge for astrodynamics software, with an anticipated lifespan exceeding 10 years). Therefore, the selection of third party software has been performed according to the following criteria:

- minimize external dependencies, maintaining only those that are 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]).

[1] tclap.sourceforge.net. Accessed: 20/08/2024.
[2] github.com/jbeder/yaml-cpp. Accessed: 20/08/2024.

The second category groups the software dedicated to the astrodynamics computations. These are ESA's GODOT[1] and NASA JPL's SPICE[2] 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[3] 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`[4] for the realization of the Graphical User Interface of CubeNav.

3 CubeNav architecture overview

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.

The primary layer in the CubeNav project is the Orbital Modeling Engine Layer, which is not directly implemented within the project. It is composed of third-party software that provides an extensive and comprehensive orbital and observation model for trajectory design and reconstruction. The two software options used by CubeNav are ESA's GODOT or NASA JPL's SPICE. The Mission Analysis layer is the second layer, which contains the actual low-level procedures to generate outputs that will be visualized in the upper layers. This layer is made up of several libraries that are owned by Unibo and Polimi, or third-party, and can be further divided into two main processing modules: the Navigation Module and the Guidance and Control (GC) Module. The third layer is the Interface layer, which serves as the interface between the outputs obtained from the lower layers and converts them into suitable formats for use in the flight dynamics computations in Layer 4. The final layer is the Flight Dynamics layer, which executes all flight dynamics procedures to obtain the desired outputs. All of these layers are managed by the Graphical User Interface (GUI), which is the tool that the user will interact with. The GUI has a twofold objective: on one hand, it allows the definition of user-defined inputs and the selection of desired outputs, and on the other hand, it provides graphical and text representation of the results obtained from the computations. The GUI is responsible for managing all of the layers and ensuring that the user receives the desired results in a clear and concise manner.

A schematic representation of the functional flow of the software is provided 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 and subsequently imported. The parsing of the environment file is then performed by the utilities within layer 1, where the universe and ephemeris necessary for computations are loaded into memory and made available (Scenario set-up). This approach guarantees that all simulations and computations performed during one session make use of the same ephemeris and configuration sets, thus ensuring consistency among the different CubeNav tools and results. Through the GUI is then possible to prepare the input file for the requested analyses, for instance by providing the time window of interest or selecting the reference frame. The desired tools can be selected and the corresponding input can be inserted by means of dedicated dialog boxes. At this stage, the selected navigation and guidance utilities (Solver) can be run to perform the desired computations. The Flight Dynamics layer processes the inputs and orbit file and generates the user-selected outputs. Since the tools are run as separated instances and the GUI works as an *orchestrator*, it is possible to run them in parallel. Once all the computations are completed and the output files are available, it is possible to generate the corresponding plots which are then visualized by the GUI. Additionally, the events can be loaded into the Timeline tool, which allows to graphically represent them in chronological order.

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 Navigation

The navigation module gathers a set of University of Bologna proprietary tools that aid the orbit determi-

[1] godot.io.esa.int/docs/1.2.0/. Accessed: 20/08/2024.
[2] naif.jpl.nasa.gov/naif/toolkit.html. Accessed: 20/08/2024.
[3] godot.io.esa.int/godotpy. Last access: 20/08/2024
[4] doc.qt.io/qtforpython-6/. Accessed: 20/08/2024.

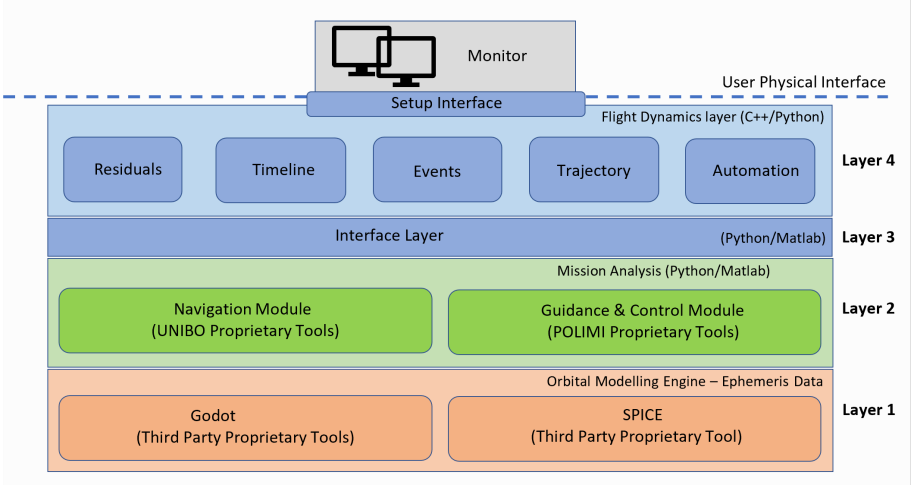


Fig. 1: CubeNav architecture organization in layers.

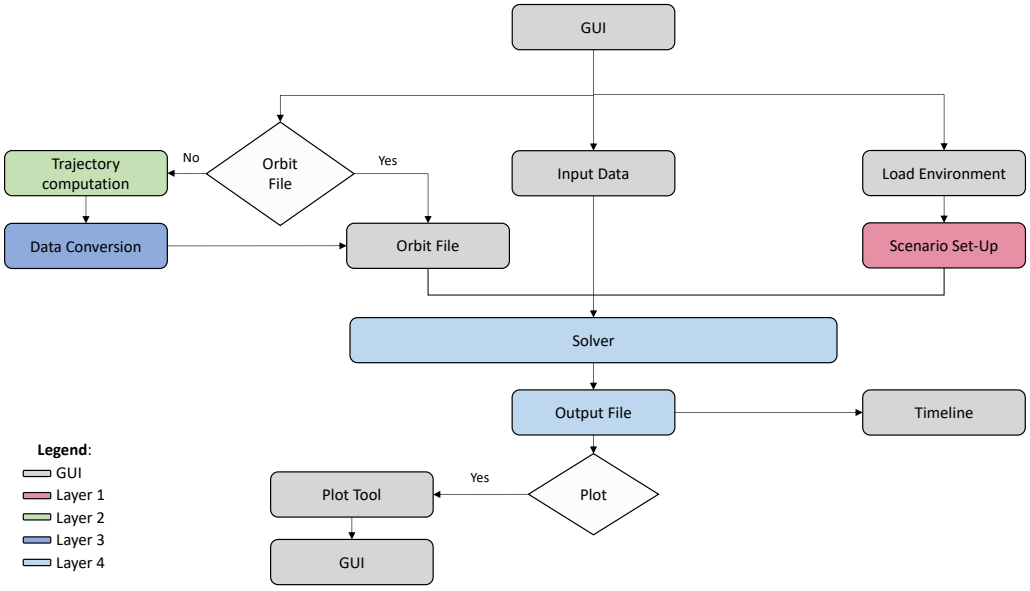


Fig. 2: CubeNav functional flow. The colors refer to the corresponding layer, as defined in Figure 1.

nation 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.

3.1.2 Guidance

The GC module is designed to gather Polimi's software for trajectory design and optimization. These four programs include 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 software [7] is intended for the analysis and simulation of low thrust interplanetary missions, and it is implemented in MATLAB while also being fully compatible with NASA's JPL SPICE Toolkit. It is geared towards performing trajectory optimization in highly nonlinear models where the design is restricted by very limited control authority. ULTIMAT has two primary modules: the Design module implements trajectory optimization techniques in a high-fidelity model, while the Assessment module evaluates the flyability of these trajectories. In particular, the program can find periodic and quasi-periodic orbits, optimize impulsive transfer trajectories with a multiple-burn multiple-shooting method, transform impulsive solutions into a finite-burn equivalent, and conduct station keeping analyses. The software's assessment of the trajectory is performed through a series of hierarchical tasks, including preliminary geometric checks, detailed navigation analyses, and the computation of vis-

ibility windows and radiometric measurements. It can also conduct sensitivity analyses against variations in low-thrust maneuver timing, duration, magnitude, and pointing angles, as well as Orbit Determination (OD) and Covariance Analysis (CA) and the quantification of Navigation Cost (NC). LT2O is a MATLAB-native software designed by Polimi to optimize low-thrust trajectories for multi-revolution transfers using indirect methods [7]. This tool handles time-, radiation-, energy-, and fuel-optimal problems while implementing simple dynamics models, such as the two-body model with J2 perturbation in cartesian coordinates, Modified Equinoctial Elements (MEE), the restricted three- and four-body models. LT2O employs advanced hybrid techniques for low thrust trajectory optimization, including continuation methods, smoothing techniques, analytical derivatives, and an accurate switching detection system that allows for end-to-end optimizations for transfers with up to 500 revolutions.

DIRETTO is a MATLAB software that uses the direct transcription method to optimize low-thrust trajectories [7]. It converts the optimal control problem into a nonlinear programming problem (NLP) and offers flexibility to accommodate both satellite and operational system constraints. The software implements three different collocation methods, including Hermite-Simpson, Gauss-Lobatto, and Pseudospectral, to satisfy the state and control variables and fulfill the dynamics constraints. DIRETTO is capable of deriving solutions for short and medium duration transfer phases and has been successfully used in solving low-thrust Earth-Mars transfers with ballistic capture. The Milani Mission Analysis Pipeline [6] 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, landing design, orbital event computation, knowledge analysis, dispersion analysis, and contingency analysis.

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. The tools available in the Flight Dynamics layer are described in detail in Section 4.

3.3.1 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.

3.4 CubeNav Graphical User Interface

In astrodynamics software designed for orbit determination, event computation, and navigation analysis, the Graphical User Interface (GUI) plays a crucial role in facilitating complex tasks. The GUI provides users with intuitive access to various tools and functionalities, allowing them to visualize trajectories, configure simulation parameters, and analyze results efficiently. Dialog boxes are integral to this process, offering structured interfaces where users can input mission-specific data, define orbital parameters, and set up analyses scenarios. These dialog boxes ensure that all necessary inputs are captured accurately, guiding the user through and removing the burden of setting up via a text editor the input files, reducing the chances of errors. The CubeNav GUI is developed in Python and makes use of PySide6 and Qt, hence it can be run in Linux, Windows (via WSL), and Mac platforms. Section 5 provides a detailed overview of the GUI characteristics and design.

4 Flight Dynamics tools

This section provides an overview of the different flight dynamics tools that have been developed within the CubeNav project. These include:

- a residual viewer, to perform pass-through analysis of radiometric observations;
- an orbit propagator;
- orbital events generators and plotters;
- an orbit merger;
- an orbit visualization tool;
- an orbit comparison tool.

All these tool are standalone executables which can be run from terminal providing a configuration file and an input file in JSON format. In the following a description of the above listed applications is provided.

4.1 Residual Viewer

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 3. 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 radiotracking 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 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 (see 5.1), synchronizing the time cursor of the timeline represented as a vertical line on the residual viewer.

The residual viewer window presents by default two types graphs: one which is dedicated to the residual vi-

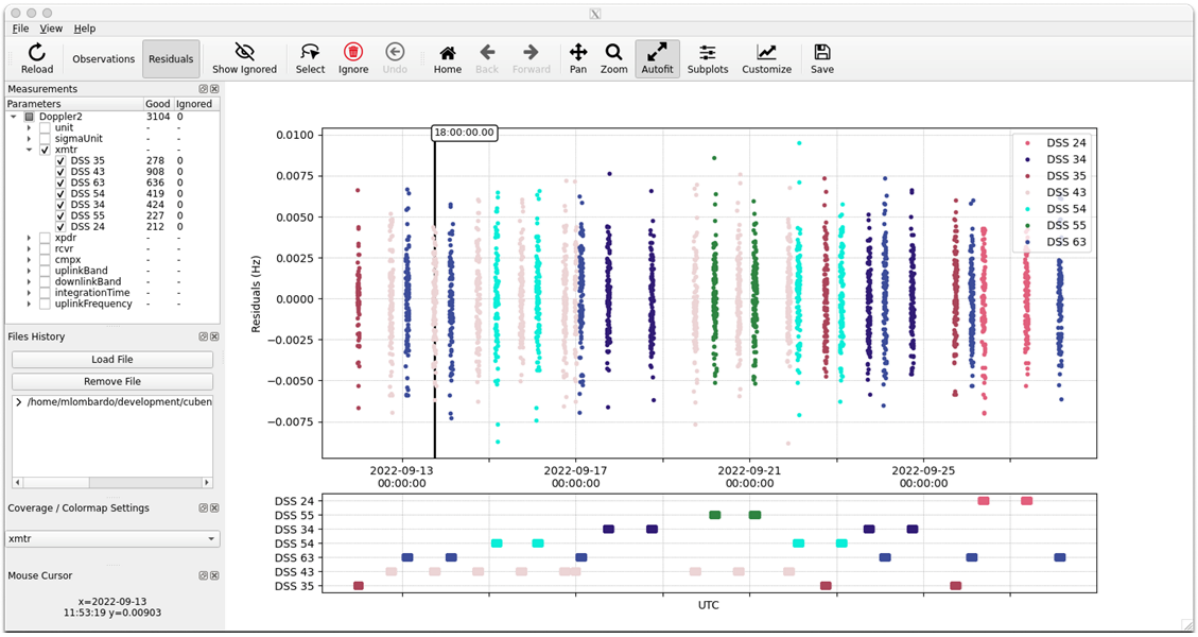


Fig. 3: Residual data visualization using the residual viewer tool (the vertical gray line identifies the time cursor of the Timeline tool). On the bottom the coverage of the selected stations.

sualization itself and the other which provides the coverage from each ground station. Note that the *lazo* selection allows the user to select outliers or specific data points which can be then included or ignored for the analysis. Multiple observation files can be loaded into memory and, once processed, will appear in the dropdown menu on the left where they could be selected for being displayed.

4.2 Orbit propagator

The Orbit Propagator is a convenient standalone tool that allows to perform a propagation using ESA's GODOT routines without the need of writing a Python script. The objective of this software application is to compute the trajectory of satellites and spacecraft in orbit, enabling accurate propagation of orbital states over time, taking into account various perturbative forces such as gravitational attraction from celestial bodies, atmospheric drag, solar radiation pressure, and spacecraft-specific parameters. The orbit propagator requires two input files: one to define the initial state (or an orbit file from which it can be extracted) and the dynamics file, a JSON file containing the definition of the dynamical model and perturbations.

4.3 Orbital events generators and plotters

The Event generators are standalone tools that can compute different astrodynamical events. All tools in this category must receive as inputs a reference trajectory, a reference time span, and time step. Additional inputs are tool-specific. The events that can be computed with CubeNav are:

- altitude;
- conjunctions (phase angle);
- eclipses;
- illumination;
- elevation (visibility);
- occultations;
- range (distance);
- three points angle;

Each tool provides as output an event file which contains in form of a JSON array the computed events, which are characterized by an epoch and a set of parameters (e.g., the elevation and azimuth for the visibility events).

The *altitude* events generator 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. This tool is useful for checking the minimum and maximum altitude of a spacecraft with respect to a body or to determine the intervals at which a payload can be activated. The graphical representation associated to this tool is reported in Figure 4, where the altitude with respect to Jupiter and Europa is represented for JUICE trajectory. Note that the intersection between the altitude curve and each target altitude, represented by red lines, corresponds to the computed events in the output file.

The *conjunction* generator tool computes the Sun 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. This can be used to determine the opposition or superior and inferior conjunctions events.

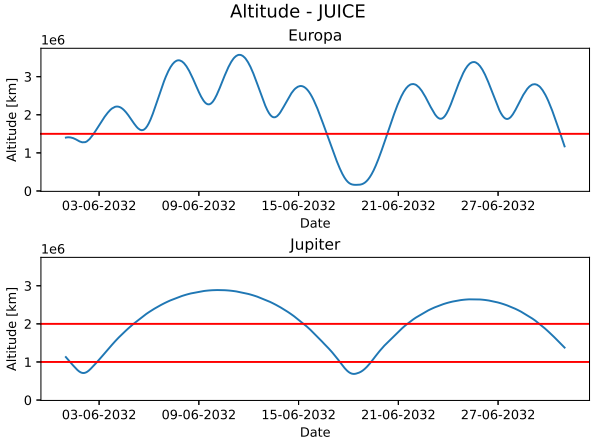


Fig. 4: Graphical representation of the fixed altitude events of JUICE trajectory test case with respect to Jupiter and Ganymede.

The *eclipse* generator returns the list of eclipses with respect to one or more celestial bodies specified in the input file. 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. Figure 5 provides an example of the graphical representation of the eclipse events. Here, the Jupiter and Ganymede eclipses for the JUICE spacecraft are identified in each subplot, with colors and height of the columns that identify the different eclipses types.

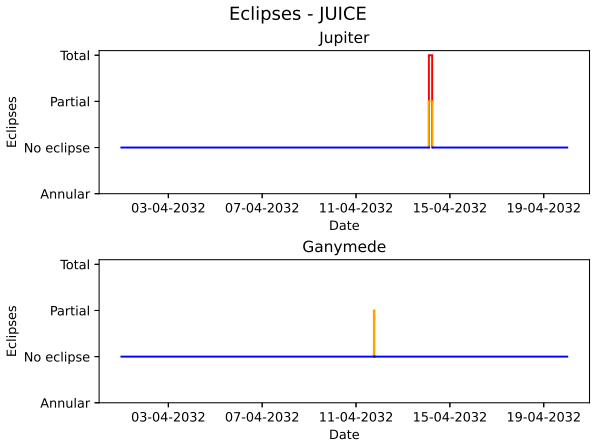


Fig. 5: Graphical representation of the eclipses events of JUICE trajectory test case with respect to Jupiter and Ganymede.

The *illumination* events are linked to the eclipse events, since this generator 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 minimum or null, correspond to total eclipse. The plot associated to this event is simply the value of the illumination function as a function of time.

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. It is possible to select a constant elevation value (fixed elevation) or consider the uplink or downlink horizon mask associated to the ground station. The elevation crossing events with positive or negative derivative correspond to an AOS and LOS event respectively. The plot associated to this event generation is simply the elevation value, together with the fixed elevation values, if present. An example of elevation plot obtainable with CubeNav is given in Figure 6.

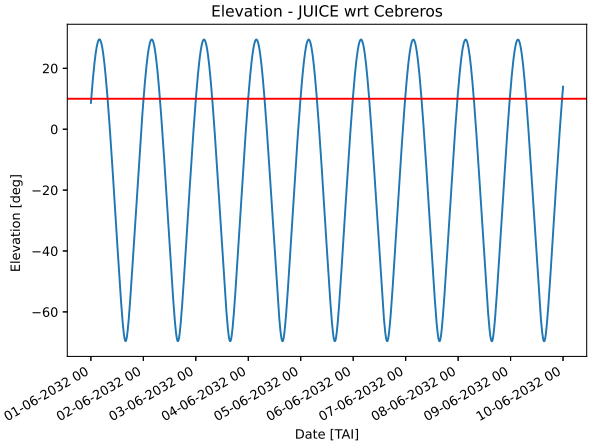
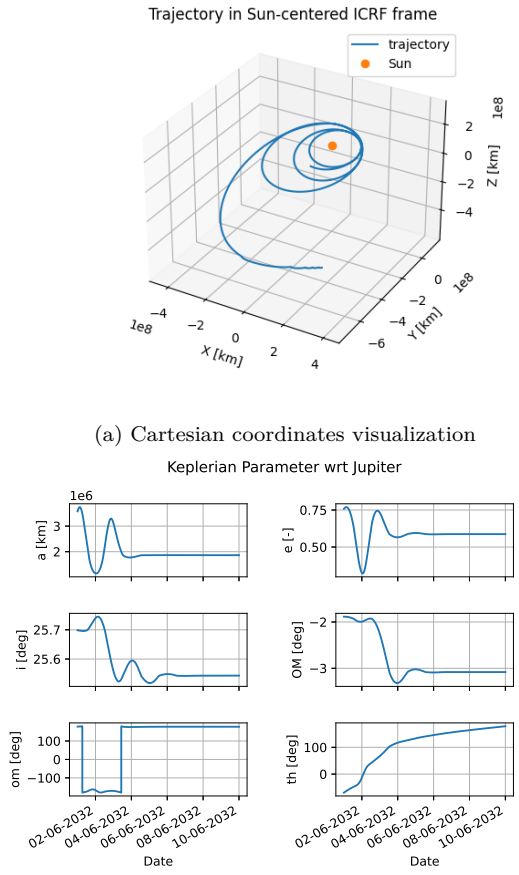


Fig. 6: Graphical representation of the elevation events of JUICE trajectory test case with respect to Cebreros 10 deg fixed horizon.

The *occultation* generator identifies the time epochs where a celestial body is occulted by one or more other defined bodies with respect to the reference trajectory. Since the occulting body (usually a planet or a moon) is located on the line-of-sight between the observer (usually the Earth, a ground-station or a planetary lander) and the target object (the spacecraft), during an occultation it is not possible to receive telemetry or send telecommands.

The *range* events generator computes the distance between the spacecraft and the center of a celestial body and identifies the local maxima and minima (i.e., the time instant when the range rate is null). When the body corresponds to the orbit main attractor, the minimum and maximum range correspond to the pericenter and apocenter of the orbit. The plot of the range event is similar to that of the altitude events.

The last generator is for *three points angle* events, which are defined as the time instants at which the angle between three points (planets, spacecraft, moons, landers) cross a given value. These events are useful to determine the relative visibility of objects in some configurations and could be useful for handling some payloads, especially when the visibility is constrained within some margins (e.g., the FOV of an instrument which is fixed on-board).



(a) Cartesian coordinates visualization

(b) Keplerian elements visualization

Fig. 7: JUICE trajectory representation in Cartesian and Keplerian coordinates.

4.4 Orbit Visualization

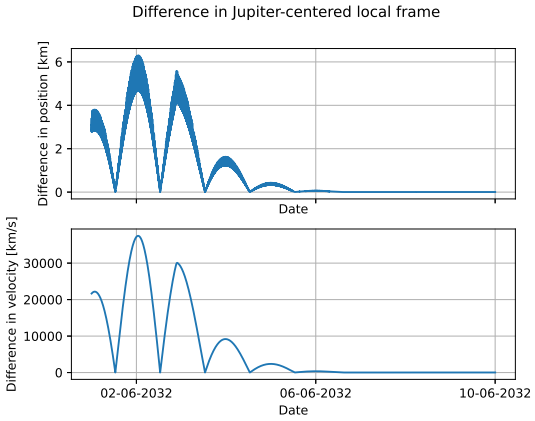
The Trajectory tool contains the tools necessary to handle the reconstructed and predicted spacecraft state. In particular, the tool embeds the graphical representation of the trajectory and the trajectory comparison tool. The first is the tool to generate the plot of the trajectory, given a certain orbit file. This can be represented in a user-defined reference frame and both in keplerian and cartesian parameters. An example of a portion of JUICE trajectory in local frame with respect to Jupiter is reported in Figure 7 in cartesian and keplerian coordinates.

4.5 Orbit merger

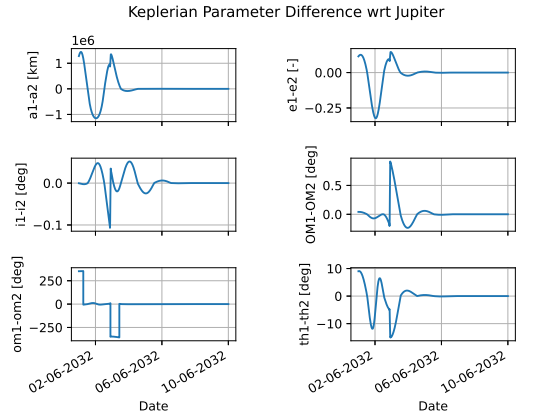
The main functionality of the orbit merger is to combine multiple orbit files, which contain information about the trajectory and position of a satellite or spacecraft, into a single, unified file. This tool is essential when managing data from multiple sources, time periods, or different mission phases. For instance, it can be used to combine orbit prediction corresponding to different orbit determination arcs or to combine an orbit prediction with an optimized orbit. The tool implemented in CubeNav can ingest and combine orbit files in different formats, like OEM, SPK, and GODOT IPF files.

4.6 Orbit comparison

The orbit comparison tool is essential in mission planning, orbit determination, and satellite operations where understanding the differences between predicted, actual, or various mission scenario orbits is critical. The trajectory comparison tool requires as input two orbit files, representing the trajectories to compare and the time interval and time step to use for the comparison. The orbit comparison can be performed considering different reference frames (inertial ecliptical or equatorial, local orbital, RTN). Additionally, different coordinate system can be used: besides the classical cartesian coordinates the keplerian elements are available as well since this representation is quite convenient to assess the performance of a manoeuvre. When selecting the cartesian coordinates it is also possible to compute the norm of the position and velocity difference. Figure 8 shows an example of orbit comparison between two different JUICE orbit files.



(a) Cartesian coordinates visualization



(b) Keplerian elements visualization

Fig. 8: Trajectory comparison in Cartesian and Keplerian coordinates. A portion of JUICE trajectory from two different orbit file is used as an example.

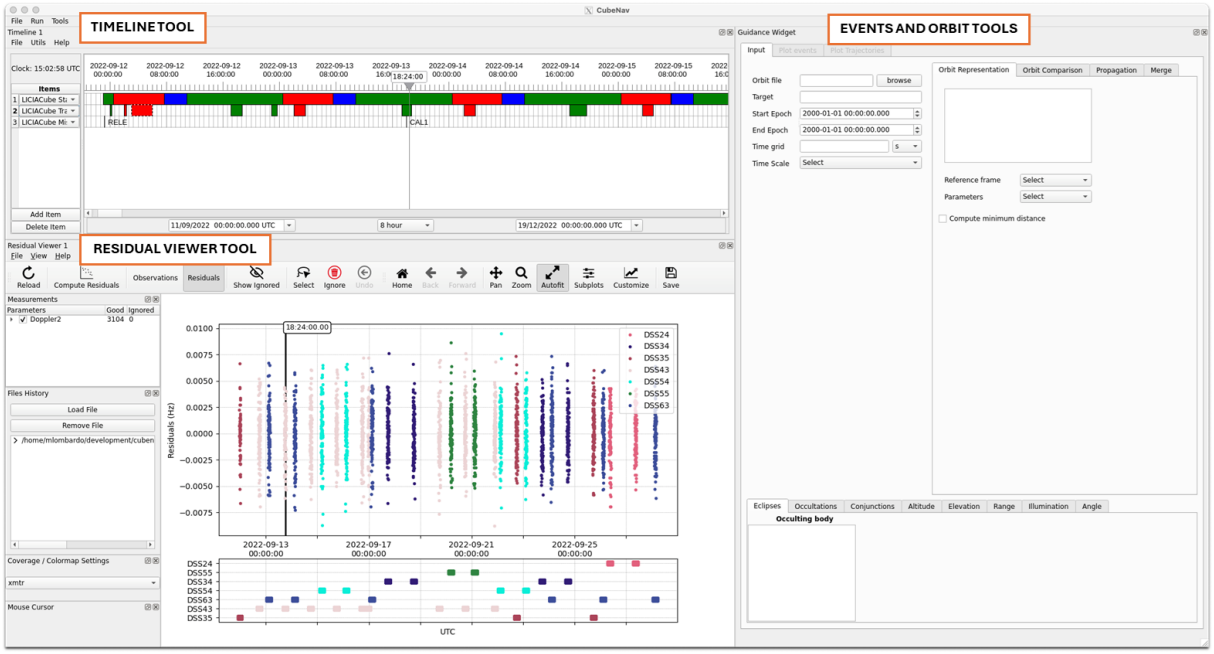


Fig. 9: CubeNav Graphical User Interface Layout.

5 Graphical User Interface

The graphical user interface of CubeNav is composed of three main dialog boxes:

- Events and orbits tools
- Timeline
- Residual viewer

At startup, by default, the three windows appear as represented in Figure 9. Each dialog box can be moved around in the main window, undocked to become a separate floating window. Multiple timeline and residual viewer windows can be opened in each session, while the event and orbit tool dialog box is unique. This choice is related to the fact that during operations, multiple residuals analysis might be carried out in parallel to test and compare different setups. Similarly, multiple timelines representing different subset of events might be useful to focus on separate aspects (e.g., operators shifts, station passes, instruments events).

The main menu of the GUI allows the selection of a working directory, where it is possible to save all configuration files. The configuration file is a JSON file which contains all parameters associated to the visualization and GUI layout. It can be edited to change the default colors, fonts and default time scale, enabling the use of the GUI also for color blind operators. Within the working directory the program will create three folders: one `inp` folder which collects all the JSON input files created by the GUI that are needed by the tools described in the previous sections, a `log` folder which contains each tool log files for the latest runs, and a `out` folder which collects all flight dynamics tools outputs and plots. The GUI also allows to save the current simulation input files and results in a separate folder.

In the following, descriptions of the Timeline tool and of the events and orbits tools GUI are provided. The residual viewer tool has been described previously in Section 4.1.

5.1 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 4.3. 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[1]) web interface

The Timeline graphical representation in Figure 10, 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

[1] spsweb.fltops.jpl.nasa.gov/rest/Wrapper/Schedule/schedview

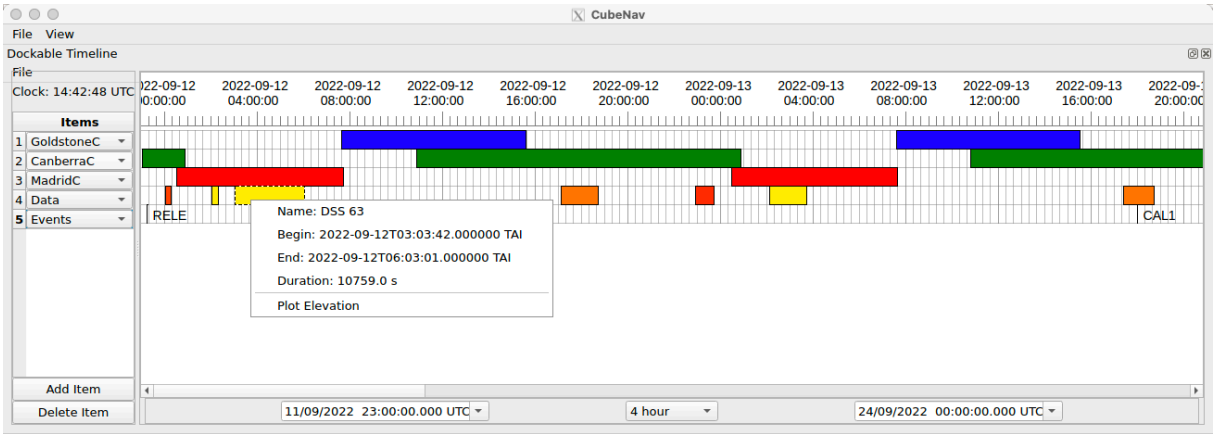


Fig. 10: Graphical representation of the events schedule provided by the Timeline tool using LICIACube data (coverage from Goldstone, Canberra, and Madrid on rows 1 to 3, actual tracking passes on row 4, and several mission events on row 5).

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).

5.2 Events and orbit tools GUI

This section provides an overview of the events and orbit tools GUI, which is the CubeNav element which allows the operator to setup and run the tools of interest which are related to orbit and events analysis. These comprises the tools listed in Sections 4.2 to 4.6.

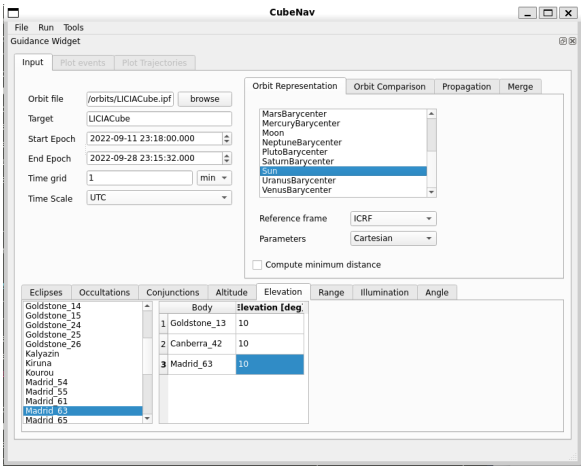


Fig. 11: Events and orbit tools layout after loading an environment file.

In Figure 11, the corresponding CubeNav dialog box is shown. The selection of celestial bodies or stations for events computation is done through dedicated lists. Besides this, the window contains a menu bar at the top and three separate tabs. The *File* menu provides an interface to load the environment and save the results. Note that at startup the fields in each list are not populated. These lists are filled up only once the environment file, which contains all GODOT universe files and SPICE kernels definition, is loaded into memory through the corresponding main menu option.

The first tab *Input* collects instead 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, namely the initial and final epochs, the time grid and time scale considered. Two separated boxes are also present for events computation and trajectory analysis (representation, comparison, propagation, and merge). 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, where applicable, is included. The second box is the one related to *trajectory analysis* tools. Here the inputs required 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. Similarly, the merge tab requires the selection of the orbit file to combine and the time span to consider for the merge operation. Finally, the propagation requires the selection of a dynamics configuration file to define the models for the propagation and the initial state to consider for the propagation.

Once 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 generated and opened as separate undocked windows, allowing the user to zoom in and interact with the generated plots via the typical matplotlib figure layout and options.

6 Conclusion

This article provides a detailed description of the CubeNav software architecture and its characteristics. The program is designed with a modular structure, dividing segments of the code with comparable functionalities into different layers and modules. This modular design allows CubeNav to be easily integrated with other third-party software for future automation of operational tasks. The software also boasts an interface layer that can convert outputs from various optimization tools to inputs suitable for flight dynamics calculations. CubeNav is a comprehensive flight dynamics software that enables users to access all mission-related information for navigation operations, thus streamlining the automation of flight dynamics tasks. The software's user-friendly Graphical User Interface (GUI) simplifies input parameter definition and output selection. Additionally, the GUI displays results in both textual and graphical formats, benefiting both operator interaction and the generation of a schedule for automated operations. Overall, CubeNav is a powerful integrated tool that effectively combines the expertise and resources of two research groups, serving as a key solution for automating flight dynamics operations for interplanetary CubeSats. By reducing operation costs and errors, CubeNav promises to be an efficient tool for supporting future exploration missions with CubeSats.

Acknowledgments

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).

Bibliography

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

[2] Giuseppe Leccese, Alberto Fedele, and Silvia Natalucci. Overview and Roadmap of Italian Space Agency Activities in the Micro- and Nano-Satellite Domain. In *73rd International Astronautical Congress (IAC)*, 2022.

[3] Elisabetta Dotto, Vincenzo Della Corte, Marilena Amoroso, I Bertini, JR Brucato, A Capannolo, B Cotugno, G Cremonese, V Di Tana, I Gai, 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*, 199:105185, 2021.

[4] Marco Lombardo, Marco Zannoni, Igor Gai, Luis Gomez Casajus, Edoardo Gramigna, Riccardo Lasagni Manghi, Paolo Tortora, Valerio Di Tana, Biagio Cotugno, Simone Simonetti, et al. Design and Analysis of the Cis-Lunar Navigation for the ArgoMoon CubeSat Mission. *Aerospace*, 9(11):659, 2022.

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

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

[7] F Topputo, DA Dei Tos, KV Mani, S Ceccherini, C Giordano, V Franzese, Y Wang, et al. Trajectory design in high-fidelity models. In *7th International Conference on Astrodynamics Tools and Techniques (ICATT)*, pages 1–9, 2018.