ASTROSIM: A MINOR CELESTIAL BODY ENVIRONMENTS SIMULATION SUITE.

Pelayo Peñarroya[(1)], **Pablo Hermosín**[(1)], **Simone Centuori**[(1)], **Lars Hinüber**[(1)]

[(1)] *Deimos Space S.L.U., Ronda de Poniente 19, Tres Cantos (Madrid), 28760, Spain*

ABSTRACT

Astrodynamics Simulator (AstroSim) is a specialized software tool created by Deimos Space S.L.U. for the purpose of mission analysis and navigation in small-body environments. Through extensive validation against Commercial Off-The-Shelf (COTS) tools, the tool has demonstrated its reliability. It encompasses a wide array of capabilities, including high-fidelity trajectory propagation, navigation analysis, Image Processing (IP), image rendering, landing simulation with contact dynamics, Hazard Detection and Avoidance (HDA), and event detection.

The objective of this paper is to provide an overview of the core functionalities and architectural framework of AstroSim. The tool is designed as a modular suite in Python, leveraging the language's versatility and Object-Oriented Programming Language (OOP) style to facilitate the seamless integration of additional functional blocks. The paper also presents the outcomes of a validation campaign, highlighting the attained levels of accuracy achieved by the tool.

Numerous modules within AstroSim are exemplified in the paper. Among them is the *astroHarm* module, which enables the extraction of spherical harmonic coefficients from polyhedral shape models. Another notable module, *astroHda*, employs Artificial Intelligence (AI) techniques utilizing Convolutional Neural Network (CNN) to identify hazards during landing simulations, such as shadows, features, and steep slopes. Additionally, the *astroRender* module harnesses the capabilities of Blender for image rendering, enabling optical navigation and facilitating the simulation of landings while accounting for the impact's contact dynamics.

1 INTRODUCTION

In the last decades, missions to minor celestial bodies have gained importance, brought forward by missions like Rosetta, OSIRIS-Rex [1], Hayabusa [2], or more recently DART [3]. These missions are proof that the interest of the space sector in these bodies is growing and that they are becoming more accessible, aided by the evolution of the technology and autonomous methodologies required.

In order to assist with the analyses needed for the design of these missions, an effort needs to be made to improve the models used in simulations in such environments. Introducing AstroSim, a cutting-edge tool designed to accurately model and predict the trajectories of space objects. This software is developed to meet the demanding needs of mission planners, researchers, and space agencies, offering exceptional precision and reliability in orbital analysis.

AstroSim provides an extensive range of capabilities. Users can perform orbital propagation for diverse scenarios, including Earth orbits, interplanetary missions, lunar missions, and more. It offers flexibility in defining orbital elements, initial conditions, and time frames, allowing for customized analyses tailored to specific mission requirements.

In addition to precise trajectory predictions, AstroSim enables users to visualize and analyze the resulting orbital paths in 2D and 3D, facilitating a deeper understanding of mission dynamics. It also

offers advanced tools for assessing orbital stability, evaluating collision risks, and optimizing mission planning.
With a focus on accuracy, efficiency, and user-friendliness, AstroSim empowers space professionals to make informed decisions, optimize mission designs, and ensure the safety and success of space operations. Whether it's for satellite deployment, space exploration missions, or scientific research, the software provides a reliable and indispensable solution for orbital analysis and planning.

2 ARCHITECTURE

AstroSim is a software tool developed at Deimos Space S.L.U. meant to be used for mission analysis and navigation around small bodies. The tool started as an orbital propagator including the typical central gravity forces and perturbations to be encountered in small body environments. Gravitational forces can be modelled with many different methodologies, from simpler implementations, such as point-mass modelling, to more complex ones like spherical harmonics or polyhedron-based models. On the other hand, the perturbations included in AstroSim include third-body gravity (unlimited bodies), and Solar Radiation Pressure (SRP). Third-body gravity models can assign any type of gravitational model to each third body in the simulation, which is particularly useful when dealing with binary systems where the bodies involved are very close to each other. Ephemeris are obtained from SPICE, which is internally linked to AstroSim by means of SPICE's Python module [4], but fictitious bodies can also be defined for specific analyses. SRP models use a conical shadowing model that considers umbra, penumbra, and even the rare case of antumbra (annular eclipse).
Additionally, to high-fidelity trajectory propagation, AstroSim offers a wide range of capabilities, such as navigation analysis, IP, image rendering, landing simulation including contact dynamics, HDA, or event detection. To integrate these functionalities, different Python libraries and third-party software have been used.
The design of AstroSim follows a modular architecture, allowing the integration of different functionalities as required. The core of the suite is written in Python, chosen due to the extensive community resources available and its license-free nature, making it highly adaptable to various environments where other licenses may not be accessible. Another key factor in selecting Python was its OOP nature, which aligns well with the envisioned modular design of the suite.
The concept of this tool differs slightly from conventional mission analysis approaches, as all data regarding propagation, including the propagation itself, is handled within the *Spacecraft* object class. This class encompasses the methods used to propagate the spacecraft's position and attitude. Upon declaring the spacecraft, a separate object of the *DynamicEnvironment* class is defined, where dynamics flags associated with the spacecraft are set, along with the corresponding ephemeris information retrieved from SpiceyPy [4].
Additionally, various instruments, actuators, noise models, and hyper-parameters for performance configuration can be attached to the spacecraft.
At the core of AstroSim lies an orbital propagator, implemented as a method within the *Spacecraft* class. This propagator utilizes Scipy's numerical integrators as the engine for the propagation. The default DOP853 numerical integrator employs an explicit Runge-Kutta method of order 8 (5, 3) with variable time-step propagation and dense output generation for interpolation, ensuring minimal loss of accuracy (more detailed information can be found at [5]).
Naturally, a dynamics library is necessary to provide the state derivatives for the numerical integrator. In AstroSim, the implemented dynamics are specifically tailored for small bodies, as they are the most relevant in such contexts:

- Central body gravity

1. as point mass,
2. as spherical harmonic model, and
3. as polyhedron model (with constant density).

- SRP
  1. ballistic modelling of the spacecraft, and
  2. including eclipses
- Third-Body Gravity (TBG)
  1. as point mass,
  2. as spherical harmonic model, and
  3. as polyhedron model (with constant density).

The spherical harmonics and SRP models implemented in AstroSim are based on the formulations presented in [6]. The polyhedron model, on the other hand, follows the approach described in [7], and it utilizes the implementation by [8].
In addition to environmental dynamics, AstroSim provides the capability to incorporate artificially-induced perturbations such as thrusters or Attitude Control System (ACS). Thrusters can be modelled as instantaneous, impulsive, or continuous accelerations, and they can follow user-defined profiles with various interpolation schemes. ACS components can also be included in the simulation as attitude control devices, allowing the spacecraft to be constrained to follow a given or user-defined attitude profile, such as nadir-pointing or inertially-fixed.
Event detection is integrated within the numerical integrator of AstroSim. This feature identifies cases where the propagation needs to be halted in order to reset the derivatives due to abrupt changes in the non-linearity of the dynamics they induce.
The eclipse detector determines the epoch at which the spacecraft transitions between eclipse, penumbra, and umbra states (and vice versa). The algorithm follows a classical conical shadow model, as described in section 3.4.1 of [6], shown in Figure 1.

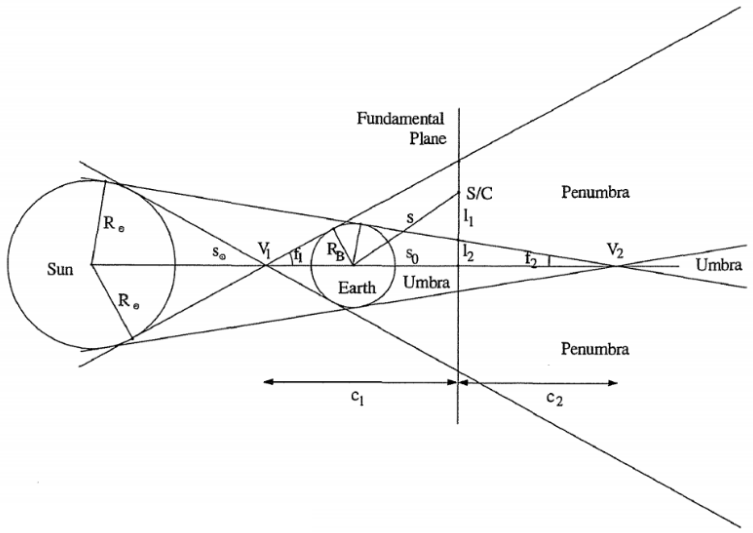


Figure 1: Schematics of the conic shadow model used in AstroSim, implemented following [6].

Another event included in the detector is the collision handler, which operates in two different modes: reference radius and shape model. In the absence of a shape model for the body causing the potential collision, a reference radius value is used to check if the spacecraft has reached a specified altitude relative to the body. If the condition is met, an error is raised to halt the propagation due to the collision. However, if a shape model is provided, the Laplacian of the polyhedron is computed based on the spacecraft's state to determine whether the spacecraft is inside (collision) or outside the shape model, following the approach explained in [9].
Both event detectors described above employ a refined bisection technique that leverages the intermediate steps of the numerical integrator and its dense output to efficiently determine the event epoch.
The remaining event detectors are dedicated to actuators and instruments that can be attached to the spacecraft object. However, the detection methods for these are relatively straightforward since manoeuvre commands or sensor activations are time-tagged sequences where only the epoch needs to be checked, thus allowing full knowledge of the event before it occurs. Consequently, no bisection is required to determine the event epoch.
All of the aforementioned functionalities have undergone extensive testing against established in-house tools used at Deimos Space S.L.U. over the past years, yielding successful results. The validation campaign involved 7-day propagations with each perturbation activated sequentially to prevent errors from one model from affecting the overall performance of the propagation. Event detection was also evaluated for eclipse scenarios. The achieved accuracies are presented in section 3.1.1.
The different modules developed for AstroSim are grouped into two categories: flight dynamics and utilities.

- Flight Dynamics
    1. *astroCelMec*: computes celestial mechanics parameters such as orbital period, or eclipse state.
    2. *astroDyn*: all the dynamics modelling functions are included here.
    3. *astroGuid*: based on Pyomo [10], configures optimisers to design the guidance trajectory that the spacecraft will need to follow to get to the chosen landing spot.
    4. *astroNav*: based on filterPy ([11]), collects all the functions needed to implement a navigation filter. Default configuration includes an Unscented Kalman Filter (UKF), observation generation and modelling (for range, position, velocity, azimuth and elevation, and Line of Sight (LoS) observations), or uncertainty propagation, for example.
    5. *astroTransf*: functions needed to perform reference frame and time system transformations. Supported by SPICE [4].
- Utilities
    1. *astroEvents*: a collection of event handlers used as input to the numeric integrator to detect an event.
    2. *astroFile*: all the functions that deal with input/output or file modification.
    3. *astroHarm*: independent suite developed following the method by [9] is used to obtain spherical harmonic coefficients from a given shape model. In [12], the advantages and limitations of this module are explored.
    4. *astroHda*: this module includes HDA algorithms used for on-board trajectory corrections based on new information collected as the spacecraft approaches the landing spot. Includes CNN-based techniques for shadow detection, feature detection, and slope estimation.

5. *astroImage*: functions having to do with image processing capabilities. OpenCV's Python library (more information at [13]) was used to implement some of the capabilities present in this suite, such as cross-correlation evaluation.
6. *astroInit*: small module that collects the basic libraries and modules needed within AstroSim.
7. *astroMath*: add-on with some specific math functions.
8. *astroPerf*: design-oriented module that helps to profile the newly included functionalities to detect memory leaks or inefficiencies.
9. *astroPlot*: Using [**?**], this module is meant to help with problem and analysis representation.
10. *astroRender*: module that exploits the capabilities that Blender ([**?**]) can provide from its own Python environment to graphically simulate small body environments. This module can be used to render images as if they were taken by an optical instrument mounted on the spacecraft at any point of the simulation or even to simulate the contact dynamics that would take place upon landing. The work on [14] shows the results obtained using this module for the Milani CubeSat ([15]).
11. *astroVar*: module that includes functionalities involving variable manipulation.
12. *astroVec*: module that includes specific vectorial operations.

The simulation script allows for independent module imports based on specific requirements. However, certain functions within these modules may have internal dependencies that are automatically imported when the functions are called. Users only need to import the higher-level functionalities they require, and the AstroSim system handles the internal dependencies seamlessly.

3 RELEVANT MODULES

3.1 Orbital Propagator

The orbital propagator is one of the core functionalities of AstroSim. It is integrated in the *Spacecraft* class and is fully configurable by the user. The orbital propagator leverages the OOP design of the entire software suite, as it is implemented as a method of the *Spacecraft* class and fetches configuration parameters from the *Spacecraft* object. This includes essential spacecraft parameters such as spacecraft mass, coefficient of reflectivity, and the cross-sectional area. The configuration of the environment is stored in the *DynamicEnvironment* class and passed to the *Spacecraft* object, to be accessed by the orbital propagator. Lastly, the integrator itself is configurable by the user. The integration method can be selected from any method available in Scipy. Depending on the requirements of the user, both fixed step-size and variable step-size can be configured, as well as the relative and absolute tolerances. If no integrator configuration is conducted by the user, the integrator properties are automatically set depending on the initial state and dynamical environment of the spacecraft.
The propagation method can then be called with a time span to be propagated. During the first call of the propagation method, the event trackers are initialised depending on the initial state and dynamical environment. Additionally, the event trackers can be configured to terminate the propagation for certain conditions, update the derivatives, and restart the propagation to avoid discontinuities. If a terminal condition occurs, the event data is stored in the *Spacecraft* object, the derivatives and event trackers are updated, and the propagation is restarted. The stored data can then be accessed from outside the propagation method, and further analysis and reports can be generated using the

event data. On termination of the propagation, the *denseOutput* object and derivates are stored in the *Spacecraft* object.

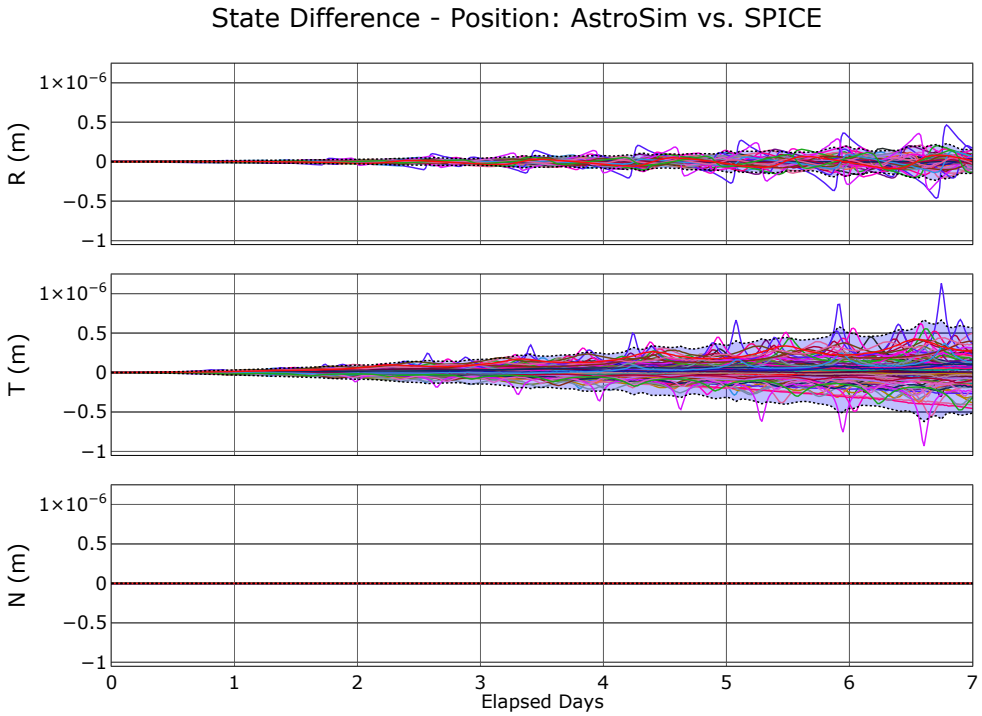
3.1.1 Validation

AstroSim has recently been validated against COTS mission analysis tools. SPICE [4] and General Mission Analysis Tool (GMAT) [16] were selected as reference tools due to their extensive validation and flight heritage [17, 18, 19]. An exhaustive validation campaign was performed, where the different dynamical models were independently tested using 100 random initial states and propagating them for seven days under different perturbations.
In order to automate the validation process, an additional module, *astroValidate*, has been implemented. It ensures that both AstroSim and the baseline tool use the same initial states and dynamical configuration. The dynamical environment, integrator configuration, and the baseline tool are defined by the user. Initial states can either be set explicitly, or generated pseudo-randomly from user-defined ranges of orbital elements. The configuration of dynamical environment, integrator properties and initial state are then processed by *astroValidate* and passed to the tools. After the propagation, *astroValidate* reads the generated outputs and prepares the validation reports and analysis.
Errors after a week of propagation are in the order of micrometers for the Point Mass (PM) gravity model, in the order of meters for the spherical harmonics and polyhedron-based gravity, and in the order of millimetres and centimetres for TBG and SRP perturbation models respectively. Table 1 lists the mean and standard deviation for different dynamical model combinations between AstroSim and the baseline tool. A fixed step-size of $30\,\mathrm{s}$ has been used in all propagations to minimize differences resulting from the integrator.

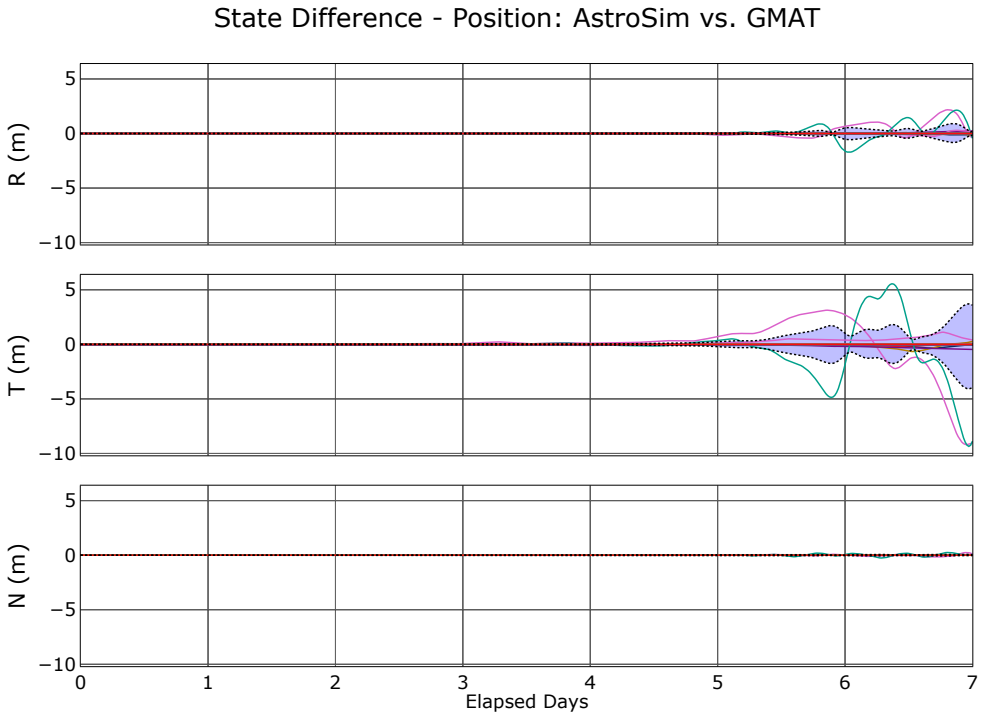
Table 1: Final position difference statistics for 100 random initial states after seven days of propagation.

Model Baseline	Point Mass (PM) SPICE	Spherical Harmonics GMAT	Polyhedral GMAT	PM + TBG GMAT	PM + SRP GMAT
Mean [m]	1.52×10^{-7}	0.409	0.197	6.08×10^{-4}	1.17×10^{-2}
Std. Dev. [m]	1.12×10^{-7}	2.81	1.26	5.45×10^{-4}	3.88×10^{-2}

As detailed by Vallado [20], it is not sufficient to assess the performance of the propagator using a single state vector after the propagation. This becomes especially apparent when plotting the differences between flight dynamics tools over time.
Figure 2a shows the difference between the `prop2b` function of SPICE and AstroSim in the Hill reference frame [6]. No distinct outliers are apparent.
The behaviour is different for the polyhedron-based gravity model, depicted in Figure 2b. There are two distinct outliers over the remaining evolutions. As expected, the standard deviation is one order of magnitude larger than the mean difference between GMAT and AstroSim. Similar behaviour was found for the spherical harmonics gravity model. The initial states associated to the outliers were studied in more detail and found to be particularly unstable, due to the relative geometry of the orbit with respect to the shape of the body. Over the course of seven days of propagation, the orbits significantly changed in their shape and orientation relative to the central body. Small state differences are exacerbated by the chaotic nature of the small-body environment.
Overall, a good agreement between AstroSim and other State of the Art (SoA) mission analysis tools was found. The OOP design and extensive configurability of AstroSim allowed to adapt it and match it with other tools.



(a) Point-mass.



(b) Polyhedral.

Figure 2: Position differences for the two different gravity models. The 3-σ environment is plotted in blue in the background and delimited by the dotted line.

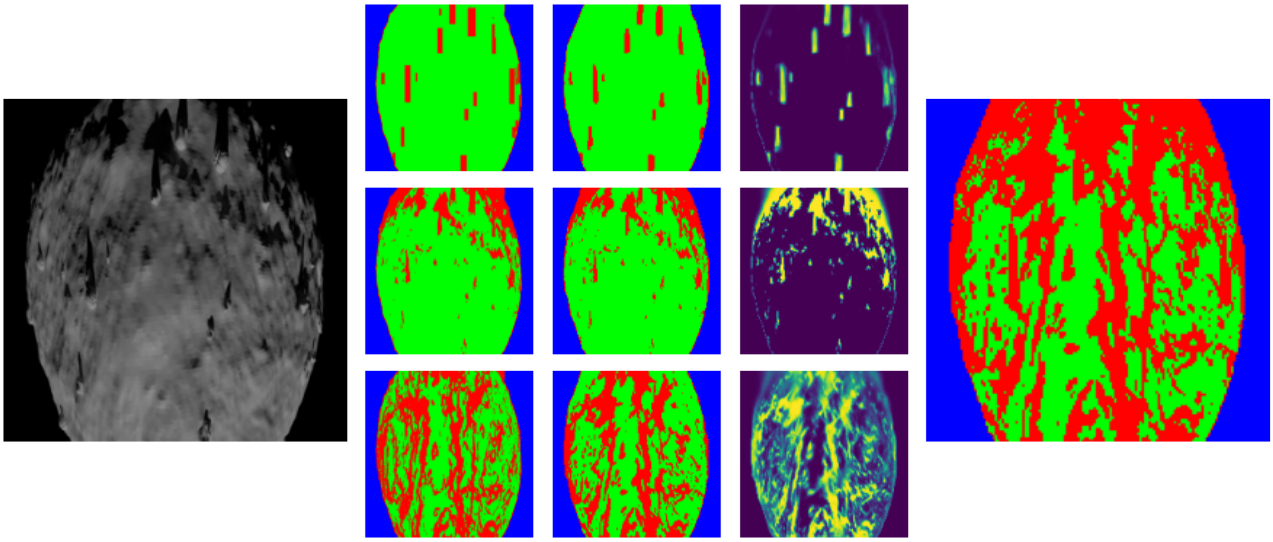


Figure 3: Example of an arbitrary image, rendered using *astroRender*, on the left. Intermediate layers of *astroHda* in the centre, namely, from left to right columns: target, prediction, and probability mask. From top to bottom, feature prediction, shadow prediction, and slope estimation. On the right side, the composed safety map where all hazards are combined to provide a conservative landing map.

3.2 HDA

One of the additional functionalities offered by AstroSim is the incorporation of HDA algorithms found in the *astroHda* module. These algorithms utilize CNNs and leverage FastAI and PyTorch frameworks to construct and train the required networks for three distinct layers: shadow detection, feature detection, and slope estimation. This three-layered system operates passively, relying solely on optical observations as input. In contrast, active systems employ instruments like Light Detecting And Rangings (LIDARs) to estimate surface characteristics such as steepness and roughness. Despite the inherent challenges of estimating slope solely based on optical observations, the networks within *astroHda* demonstrate hazard prediction capabilities (identifying high slopes) with true positive accuracies exceeding 70%.

Figure 3 shows an example of the capabilities that *astroHda* offers. Using semantic segmentation techniques, three CNNs were trained to detect features such as boulders or craters, detect shadows, and estimate slopes. Features and shadows are detected on a yes-no basis, but for slopes, a threshold is defined to initially classify them as dangerous or safe slopes. This classification is based on the spacecraft's design, and its capacity to land under certain terrain conditions. For instance, for the case exemplified in Figure 3, a threshold of 15 °was used.

In the middle section of the figure, three rows are provided, one for each of the HDA layers: feature detection, shadow detection, and slope estimation (from top to bottom). The three columns correspond to (from left to right) the target (or true mask), the prediction given by the network, and the certainty with which the network flags the hazards present in the image. Finally, all these layers are composed to generate what is called a safety map, on the right-most plot of Figure 3. There, a final representation of the safe and dangerous landing areas on the surface of the body in the image is given, for the Guidance, Navigation, and Control (GNC) system to select a final landing spot that satisfies the safety requirements of the mission.

3.3 Small Celestial Body Landing Simulations

The process of image rendering relies on the utilization of Blender and pyrender. Blender, an open-use licensed Video Effects (VFX) suite, is widely supported by the community and offers a diverse range of features. These features encompass 3D modelling, UV unwrapping, texturing, raster graphics editing, fluid and smoke simulation, particle simulation, soft body simulation, sculpting, rendering, motion graphics, and compositing, among others. This comprehensive toolset makes Blender highly appealing for image rendering, as it can generate images that closely resemble the outcomes achieved by optical payloads.
Pyrender, on the other hand, is a Python module specifically designed for image rendering, often with significantly faster performance compared to Blender. However, Blender provides additional advantages that can be directly harnessed within a small-body environment simulator. Notably, Blender incorporates a contact dynamics engine capable of facilitating rigid-body simulations that account for collisions and deformations. This capability is exploited in AstroSim to conduct contact dynamics simulations for landing sequences [14].

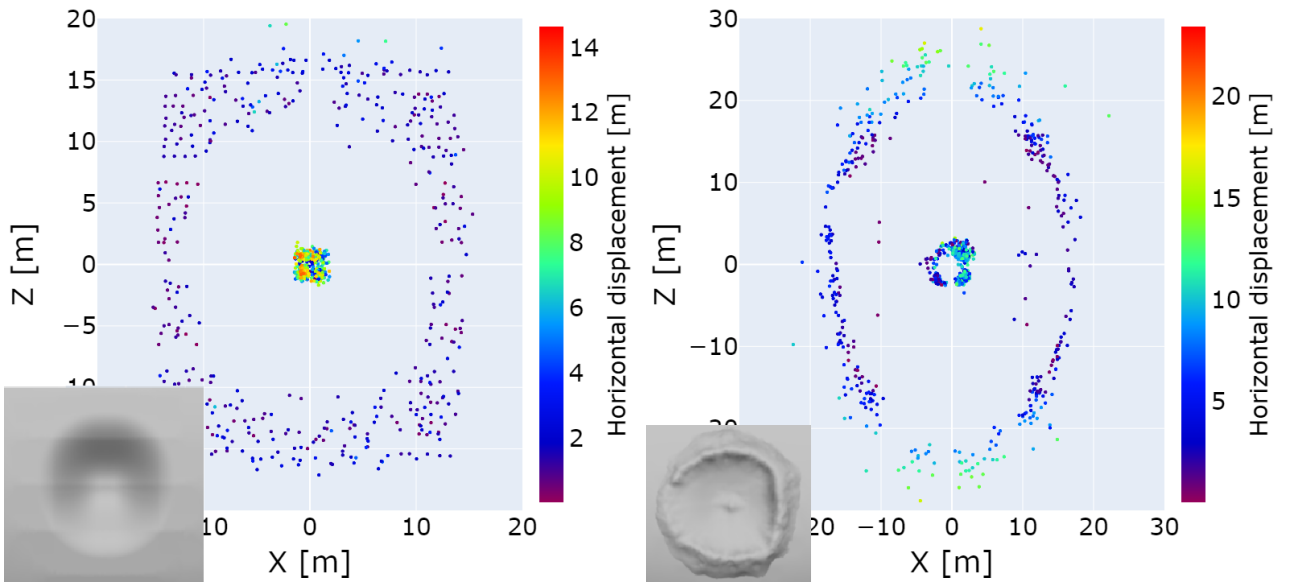


Figure 4: Horizontal displacement analyses plots for different types of craters [14]. Displacement corresponds to the horizontal distance travelled by the spacecraft from its initial position on the grid to the settling position at the end.

Figure 4 shows one of the analyses that can be obtained from the module developed. In it, the horizontal displacement of a particle (spacecraft) from its position on a grid placed above the landing spot to be studied (in this case a crater). The tool helps understand how different crater morphologies affect the way a certain spacecraft bounces after the initial touchdown when landing and how the features of the crater favour certain landing spots as final settling-down areas.
The tool provides a very valuable resource for landing planning, in particular when it comes to uncontrolled landings, as they are typical for CubeSats.

3.4 Spherical Harmonic Model Coefficients from Shape

Based on the work from [7, 9], a module was developed that computes the coefficients for the spherical harmonics gravitational model of a certain small celestial body from its shape model (usually a polyhedron). In [12], a more detailed overview is provided, with different analyses and validation sections for this module, named *astroHarm*. As it was shown in that work, spherical harmonics simulations offer a lighter computational burden, while keeping the accuracies of the propagation close to the polyhedral model, when the order and degree of the spherical model are high enough. This benefits on-board processes and helps mission planning strategies since a shape model is usually available (even if just a rough approximation) before the spacecraft arrives at the target body and orbits it for some time to estimate the coefficients needed for the spherical model.

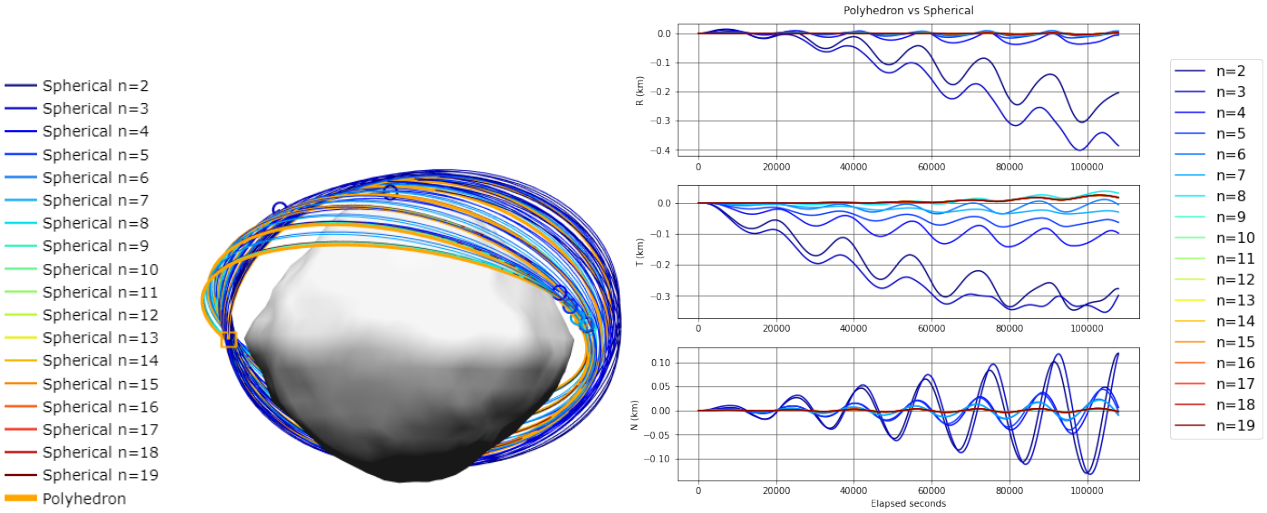


Figure 5: Trajectory propagation results for polyhedral and spherical harmonic models, on the left. On the right-hand side, a quantitative evaluation of the error that lower-order spherical harmonic models yield with respect to the ”ground truth” trajectory provided by the more accurate polyhedral model [12].

Figure 5 shows an example of a trajectory propagated using a polyhedral model and a set of trajectories propagated using increasingly higher-order spherical harmonics models. The right-hand side plot shows how when increasing the order and degree of the spherical harmonic model, propagation errors vanish, even if the computational effort becomes much lower. This tool is very helpful for on-board appliccations and can be used to create a catalogue of spherical harmonic coefficients for any observed body from which a shape model, even if rough, is available.

4 CONCLUSIONS

AstroSim was first presented at the AIAA/ASS SciTech 2022 in San Diego (CA), where a preliminary version of the suite was introduced. After that, it has been used to produce other conference papers and publications. In this paper, the main functionalities of the tool are described and its architecture is showcased.

Firstly, AstroSim is introduced as a modular suite developed in Python, to which different functional blocks can be easily added by exploiting the language’s high versatility and OOP style. Then, the results of the validation campaign are included, where the accuracies that the tool is capable of reaching

are shown in detail.
Examples of the different modules are provided, including *astroHarm*, a module to obtain the spherical harmonic coefficients from polyhedral shape models; *astroHda*, an AI-based module that uses CNN to detect hazards for a landing simulation (such as shadows, features, or high slopes); or *astroRender*, a module that uses Blender to render images that can be used for optical navigation and to simulate landings accounting for the contact dynamics of the impact.
AstroSim continues to be used within the Mission Analysis and Navigation team at Deimos Space S.L.U., and new features are added for specific project needs or following the improvements roadmap devised by the team. Some of these features include the generation of frozen orbits for asteroidal environments or visibility analysis capabilities, for instance.

5 REFERENCES

[1] R. Garner, "OSIRIS-REx TAGs Surface of Asteroid Bennu," Oct. 2020. [Online]. Available: http://www.nasa.gov/feature/goddard/2020/osiris-rex-tags-surface-of-asteroid-bennu

[2] S.-i. Watanabe, Y. Tsuda, M. Yoshikawa, S. Tanaka, T. Saiki, and S. Nakazawa, "Hayabusa2 Mission Overview," *Space Sci Rev*, vol. 208, no. 1, pp. 3–16, Jul. 2017.

[3] A. F. Cheng, P. Michel, C. Reed, A. Galvez, and I. Carnelli, "DART: Double Asteroid Redirection Test," p. 2, 2012.

[4] A. Annex, B. Pearson, B. Seignovert, B. Carcich, H. Eichhorn, J. Mapel, J. von Forstner, J. McAuliffe, J. del Rio, K. Berry, K.-M. Aye, M. Stefko, M. de Val-Borro, S. Kulumani, and S.-y. Murakami, "SpiceyPy: a Pythonic Wrapper for the SPICE Toolkit," *JOSS*, vol. 5, no. 46, p. 2050, Feb. 2020. [Online]. Available: https://joss.theoj.org/papers/10.21105/joss.02050

[5] P. Virtanen, R. Gommers, T. E. Oliphant, M. Haberland, T. Reddy, D. Cournapeau, E. Burovski, P. Peterson, W. Weckesser, J. Bright, S. J. van der Walt, M. Brett, J. Wilson, K. J. Millman, N. Mayorov, A. R. J. Nelson, E. Jones, R. Kern, E. Larson, C. J. Carey, I. Polat, Y. Feng, E. W. Moore, J. VanderPlas, D. Laxalde, J. Perktold, R. Cimrman, I. Henriksen, E. A. Quintero, C. R. Harris, A. M. Archibald, A. H. Ribeiro, F. Pedregosa, and P. van Mulbregt, "SciPy 1.0: fundamental algorithms for scientific computing in Python," *Nature Methods*, vol. 17, no. 3, pp. 261–272, Mar. 2020. [Online]. Available: https://www.nature.com/articles/s41592-019-0686-2

[6] O. Montenbruck, E. Gill, and F. Lutze, "Satellite Orbits: Models, Methods, and Applications," *Appl. Mech. Rev.*, vol. 55, no. 2, p. B27, 2002. [Online]. Available: http://AppliedMechanicsReviews.asmedigitalcollection.asme.org/article.aspx?articleid=1396996

[7] R. Werner and D. Scheeres, "Exterior gravitation of a polyhedron derived and compared with harmonic and mascon gravitation representations of asteroid 4769 Castalia," *Celestial Mech Dyn Astr*, vol. 65, no. 3, 1997. [Online]. Available: http://link.springer.com/10.1007/BF00053511

[8] D. González, "DaniGlez/polygrav," Jun. 2020, original-date: 2020-06-18T18:16:52Z. [Online]. Available: https://github.com/DaniGlez/polygrav

[9] R. A. Werner, "Spherical harmonic coefficients for the potential of a constant-density polyhedron," *Computers & Geosciences*, vol. 23, no. 10, pp. 1071–1077, Dec. 1997. [Online]. Available: https://linkinghub.elsevier.com/retrieve/pii/S0098300497001106

[10] W. E. Hart, C. D. Laird, J.-P. Watson, D. L. Woodruff, G. A. Hackebeil, B. Nicholson, and J. D. Siirola, *Pyomo — Optimization Modeling in Python*, 2nd ed., ser. Springer Optimization and Its Applications. Springer International Publishing, 2017. [Online]. Available: https://www.springer.com/gp/book/9783319864822

[11] R. Labbe, "rlabbe/filterpy," May 2021, original-date: 2014-07-15T02:15:19Z. [Online]. Available: https://github.com/rlabbe/filterpy

[12] P. Peñarroya and R. Paoli, "Orbit Propagation Around Small Bodies Using Spherical Harmonic Coefficients Obtained From Polyhedron Shape Models," Apr. 2021.

[13] G. Bradski, "The OpenCV library," *Dr Dobb's J. Software Tools*, vol. 25, pp. 120–125, 2000. [Online]. Available: https://ci.nii.ac.jp/naid/10028167478/

[14] P. Peñarroya, M. Pugliatti, S. Centuori, and F. Topputo, "Using Blender As Contact Dynamics Engine For Cubesat Landing Simulations Within Impact Crater On Dimorphos," Apr. 2021.

[15] 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, no. 6, pp. 2010–2029, Mar. 2021. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0273117720309078

[16] S. P. Hughes, R. H. Qureshi, S. D. Cooley, and J. J. Parker, "Verification and validation of the general mission analysis tool (gmat)," in *AIAA/AAS Astrodynamics Specialist Conference*. American Institute of Aeronautics and Astronautics.

[17] NASA Navigation and Ancillary Information Facility. SPICE Users. [Online]. Available: https://naif.jpl.nasa.gov/naif/SPICE_Users.pdf

[18] GMAT Development Team. NASA Usage of GMAT. [Online]. Available: https://gmat.atlassian.net/wiki/spaces/GW/pages/380273244/NASA+Usage

[19] GMAT Development Team. Industry Usage of GMAT. [Online]. Available: https://gmat.atlassian.net/wiki/spaces/GW/pages/380273246/Industry+Usage

[20] D. Vallado, "An analysis of state vector propagation using differing flight dynamics programs."