1st Annual RPI Workshop on Image-Based Modeling and Navigation for Space Applications
Rensselaer Polytechnic Institute, Troy, NY—June 4-5, 2018

SOFTWARE ARCHITECTURE FOR CLOSED-LOOP AUTONOMOUS OPTICAL NAVIGATION SCENARIOS

Thibaud Teil[*], Hanspeter Schaub[†]

This paper develops an optical navigation software architecture in order to realistically develop, run, and test novel autonomous navigation methods. This architecture harnesses two main components: a high-fidelity, faster than real-time, astrodynamics simulation framework; and a sister software package to dynamically visualize the simulation environment. Maneuvers such as fly-bys and orbit insertions occur over short periods of time and must occur autonomously. Yet, there are no open-source software packages that provide fully coupled spacecraft dynamics and Flight Software (FSW) capabilities, especially for Optical Navigation (OpNav) mission scenarios. The presented tool consists of the *Basilisk* astrodynamics framework interfacing with a *Unity*-based visualization that provides a synthetic image stream of a camera sensor. This allows guidance navigation and control (GNC) algorithms to be run in a closed-loop format. The optical measurements are read and processed in the simulation, and used for control and decision making. This *Unity*-based software has the ability to import shape-models, planet maps, and change into an instrument point-of-view. Paired with open-source image processing libraries, these combined components have provided all the necessary pieces to fully simulate autonomous, closed-loop, OpNav scenarios. This allows for progress in the autonomy sector, as full-fledged Flight Software could be tested in a real flight context. Furthermore, this increases the reliability of GNC methods, allowing for sufficient state estimation with minimal instruments. This paper presents the *Basilisk* and *Unity* engine interface.

INTRODUCTION AND MOTIVATION

As future missions ambitiously take us farther into the Solar System, there will be a need for autonomous operations with minimal Earth contact. Furthermore, renewed interest in sending humans beyond low Earth orbit calls for enhanced safety, for which autonomy from ground teams plays a key role in case of communication failures. Whether it be for robotic exploration of the solar system, manned spaceflight, or small satellite development, autonomy opens the door to new mission concepts.

One key enabler for autonomy is on-board, optical navigation. This paper outlines a novel framework which seeks to combine navigation algorithms within a simulated environment. These algorithms require this reliable test-bed to be developed and improved. This software package will provide that capability in order to open new avenues in spacecraft autonomy. This work outlines the architectural decisions that were made and the tools that are being used to optimize functionality.

[*]Graduate Student, Aerospace Engineering Sciences, University of Colorado Boulder.
[†]Glenn L. Murphy Chair of Engineering, Department of Aerospace Engineering Sciences, University of Colorado, 431 UCB, Colorado Center for Astrodynamics Research, Boulder, CO 80309-0431. AAS Fellow.

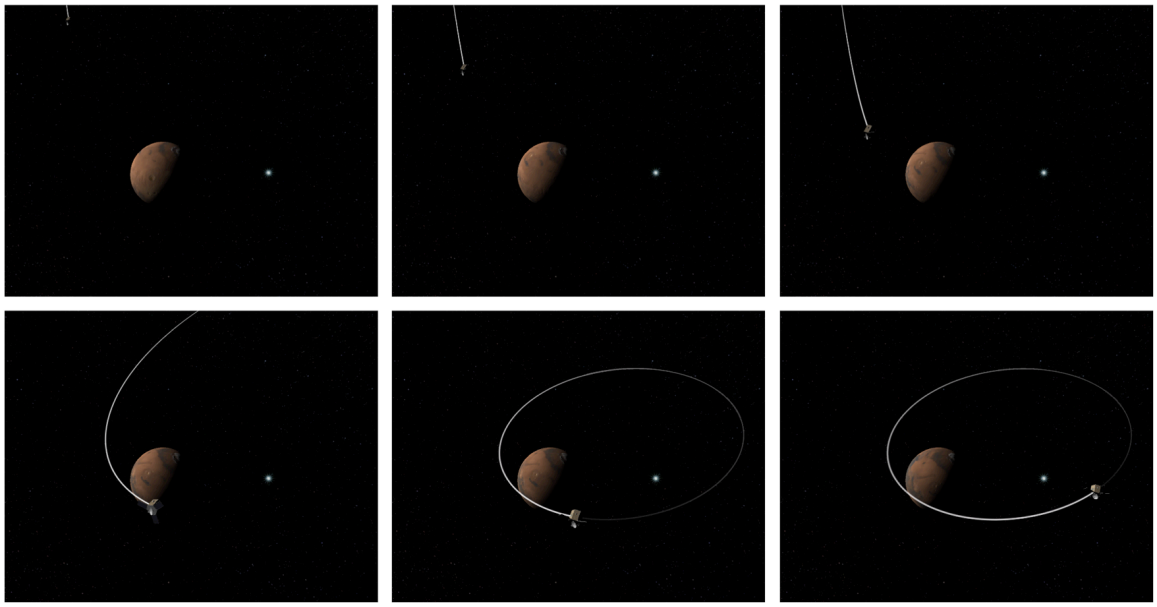

Figure 1. Mars Orbit Insertion Scenario[8]

OpNav simulations have in the past focused either on the image processing component[1] , or on the estimation component,[2,3] or on using mission data.[4] These provide valuable insight on both many facets of the problem. Yet no common open-source software package exists, which provides modularity and repeatability all the while bringing together contributions from many actors. Furthermore, current simulations do not couple the spacecraft dynamics and control into the OpNav measurements.[5] Camera models are linked to the image processing and filter performances,[6] but this does not loop back to the spacecraft control algorithms.

One major scenario of motivation is an orbit entry maneuver. This maneuver occurs in dangerous proximity to the body of interest, during a short time span (often too short for human intervention), and is paramount to mission success. A spacecraft's sole use of optical measurements provides assurance of mission success, all the while showcasing the OpNav chain linked with attitude control and trajectory modifications. Figure 1 shows a Mars Orbit Insertion performed in *Basilisk*.[7] In future work, this will be the baseline test for the software architecture.

In the future, this software will also help develop many other scenarios. For entry, decent, and landing (EDL) and asteroid missions safety, these developments could add an important element of reliability by providing a testbed for autonomy and quantifying performance. SLAM and cross-correlation methods could also be implemented and tested in a realistic spacecraft environment. These algorithms are currently being developed for these purposes notably within NASA and ESA.[9]

SOFTWARE ARCHITECTURE

In this section, the components of the software framework are described.

Basilisk and its Visualization Software

Basilisk is an open-source astrodynamics framework being developed by the University of Colorado Autonomous Vehicle Systems (AVS) lab and the Laboratory for Atmospheric and Space

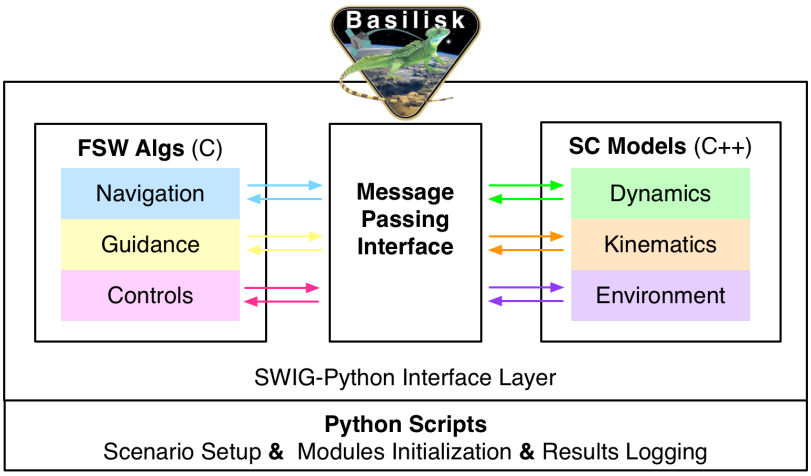

Figure 2. The *Basilisk* architecture[7,10]

Physics (LASP). Through high-fidelity, faster-than-real-time dynamics, it allows one to simulate spacecrafts in realistic flight conditions. The inherent speed of the framework and its multithreaded Monte-Carlo capability provides high-end analysis tools. In this simulation, FSW and spacecraft models are separated. By communicating through the Message Passing Interface (MPI), blocks of code can be added and contribute to the simulation without necessary knowledge of other blocks. This interface allows for closed loop control and simulation.

Alongside this effort, a visualization software[8] package receives *Basilisk* state messages and dynamically displays these states using the *Unity*[*] rendering engine. This companion software has the ability to import shape-models and planet maps and display and instrument point-of-view. Paired with open-source image processing libraries, these combined components provide all the necessary pieces to fully simulate autonomous, closed-loop, OpNav scenarios. This research develops the software architecture that allows to combine these components and demonstrates its capabilities. The open-source nature of the project is critical as it allows for any user to contribute to the project, and therefore centralizes progress in astrodynamics.

Basilisk is a highly modular simulation framework in which all modules exchange data via messages. This modularity comes from the fact that *Basilisk* modules publish and subscribe without requiring knowledge of other existing modules. The separation of FSW and spacecraft models managed via the MPI naturally welcomes another actor: the Visualization. The interface between FSW and spacecraft models allow for closed loop simulation and control. In the same way, the combined interfaces between FSW, spacecraft models, and the Visualization can allow for closed-loop OpNav guidance navigation and control.

Real-time interfacing

The software architecture allows the Visualization to capture information from the spacecraft's environment and communicate it to *Basilisk* while running. In turn, *Basilisk* is able to process this information and use it for decision making and for navigation. The spacecraft's attitude and position coupled with the environment changes are critical to a realistic simulation.

[*] https://unity3d.com

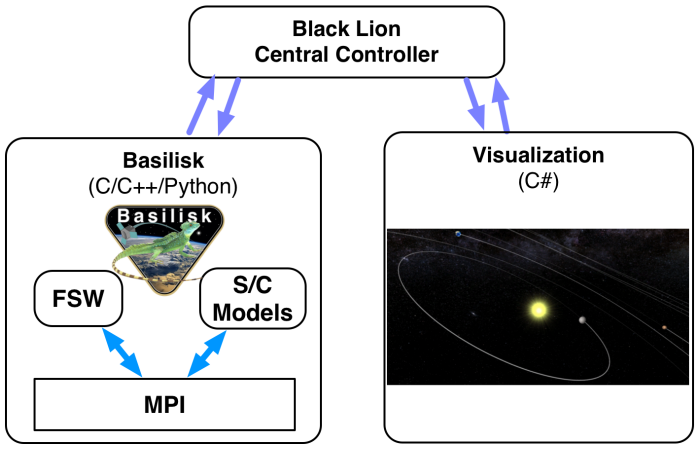

Figure 3. Interaction between *Basilisk*, *Black Lion*, and the Visualization

Implementation of the interface produces two challenges. The first is making two heterogenous software entities communicate and the second is to have the complete simulation execute faster than real time. Each software package is written in different programming languages, wrap their internal data using different structures or packet types, and execute at different speeds. The challenge is to integrate these heterogenous components while maintaining synchronous operation of the modules. The *Black Lion*[10,11] package developed in the AVS lab is middleware that ensures this proper interfacing between nodes. In this case, the nodes or applications are *Basilisk* and the Visualization. *Black Lion* ensures:

- The transport of binary data via a transport layer (TCP).
- The marshaling (or translation) of binary data. Each node must know how to convert the received bytes into structures that can be managed internally.
- The synchronization of nodes to keep all the nodes in lock-step during the simulation run.

The central controller acts as a master in the synchronization of the nodes, and a broker in the data exchanges.

The last component is the translation layer (or marshaling) of the data. For that, Google Protobuffers* are used. They provide a platform and language agnostic translation layer library to facilitate marshaling and unmarshaling of data between the two simulation applications. By creating these Protobuffer structures, both the C++ code in *Basilisk* and the C# code in *Unity* can read in and write out the necessary content.

Figure 3 shows the interaction between the major nodes. *Black Lion* allows for the communication to occur correctly and in a synchronized manner. Figure 4 then shows the details of the interfaces. As stated previously, *Basilisk* modules write and subscribe to messages via the Message Passing Interface without any knowledge of other existing modules. *Basilisk* contains a C++ module which reads the required *Basilisk* messages, writes them as protobuffers, and gives them to *Black Lion* (symbolized by the bold black double arrow). The Visualization interface must then unpack the protobuffers in order for the game controller to use the data.

*https://developers.google.com/protocol-buffers/

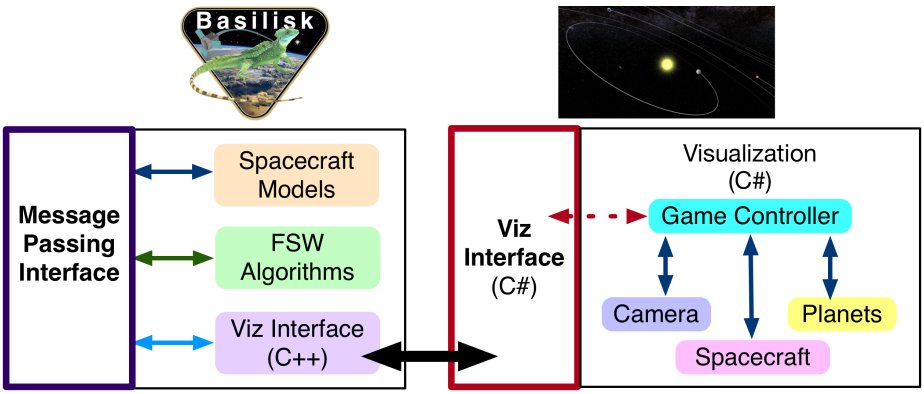

Figure 4. Different software interfaces

These design choices reflect the desire for a modular yet robust architecture. The use of Google Protobuffers allows for platform independent communication; *Unity* provides a user friendly and vast community for environment development; *Black Lion* is the middleware that connects and applications and synchronizes them. These tools provide the building blocks for the framework being implemented.

OPTICAL NAVIGATION COMPONENTS

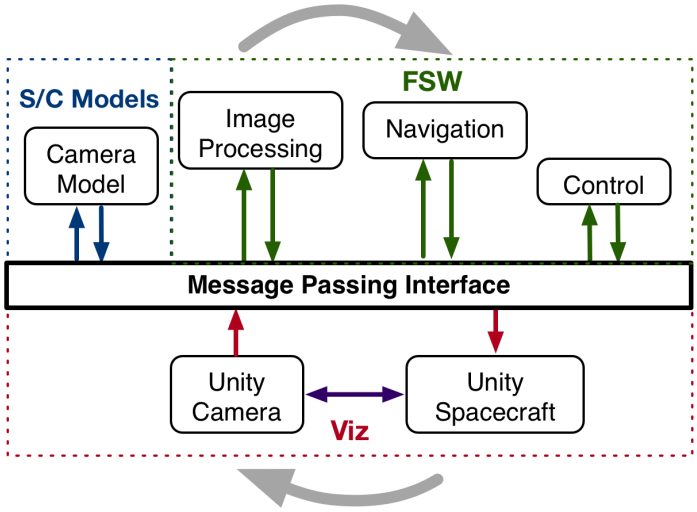

Figure 5. Information flow between the Visualization and the Simulation

Figure 5 shows the closed loop data flow between all the interfaces. As an image is snapped by the Visualization (or *Unity*) camera, it is written into a protobuffer and passed on to the simulation (red arrow). The camera model then corrupts the image, before the image processing module can extract features from it. These features are then digestible by the navigation filters. With updated state estimates, the control algorithms can therefore correct for errors. This reading and writing of messages is symbolized by blue arrows for the simulation, and green arrows for the FSW. As the spacecraft states and environment dynamics change per the control laws, the Visualization is notified via a protobuffer message, which can in turn change the images snapped by the on-board

camera.

Unity can save images to an external file. The message that it will provide to *Basilisk* will only provide: the time tag for the image, the camera head that it corresponds to, and the location of the image. Hence the camera model will read this message and have all the information required to retrieve the pixel data, and corrupt it. This prevents large data flows, by never creating messages with full image data.

Camera models and Navigation Filters

In order to realistically model OpNav scenarios, the images generated by the visualization software must be as realistic as possible. *Unity* provides a large set of lighting libraries that can simulate self-shadowing and model lighting on imported shape-models. This allows for the generation of complex lighting scenes. By extension, it allows for the reading in of partially lit planets, showing crescent lighting. This lighting is seen in the Visualization between Figures 6 and 7. Many more lighting models exist within *Unity* if more realistic functionality were needed.

These images are written out in their highest-definition form (Figure 6), and will be read by a camera model simulator. Depending on modifiable camera specifications (field of view, pixel size, dynamic range), the image can be corrupted. Furthermore, a blurred filter can be added as well to simulate spacecraft jitter or any other noise that could be desirable to model.

In order to facilitate this development, *OpenCV*[*] was chosen as a computer vision library. This will save development time and guaranteeing robustness through widely tested functions. The camera simulator (camera model) can then write its message which is read in by the image processing module in use. Depending on the OpNav scenario, this could be a centroid tracking scenario, or a feature tracking scenario. These options would lead to different output messages, which in turn would be picked up by the navigation filter.

The information taken out of the visualization yields navigation data. Using the planet center direction, and with the knowledge of the ephemeris of said planet, both orbit determination and attitude control can be accomplished. It's important to note that observability can be difficult to achieve in certain configurations. For instance, the attitude problem suffers from an unobservable rate around the planet-center direction, as it does for sun heading estimation.[12] Similarly, as the spacecraft is still far from the celestial body of interest, it will only appear as a point on pixelized image. Only as it gets progressively closer will the planet's size become apparent, and the range from the object becomes observable. All these estimation challenges are fundamental to accurate GNC. Nonetheless, *Basilisk* already contains several implementations of filters, ranging from square-root Unscented Kalman Filters, to Extended Kalman Filters.

Vision-based navigation methods

Several optical navigation methods exist. Among these are star horizon,[13] centroid and apparent diameter,[14] star occultation,[15] and landmark tracking.[5] Each of these have their specific application scenarios depending on the object they are required to track.

In the context of the our software implementation for OpNav, a simple navigation method is desired for initial development. Centroid and apparent diameter measurements, for instance, find the

[*] https://opencv.org

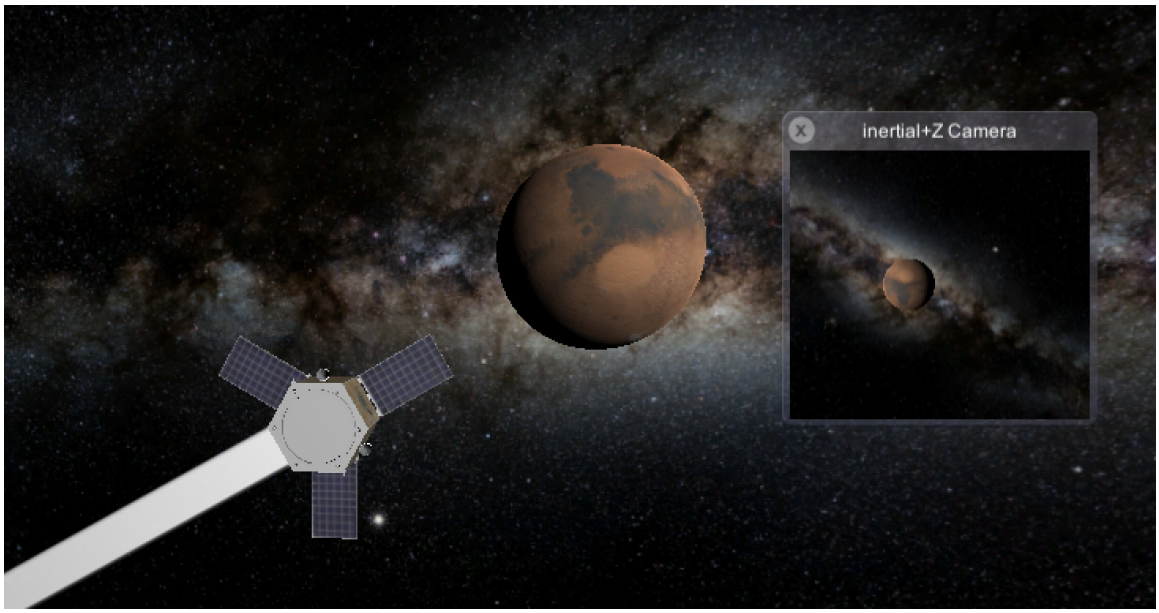

Figure 6. Spacecraft observing Mars in Visualization

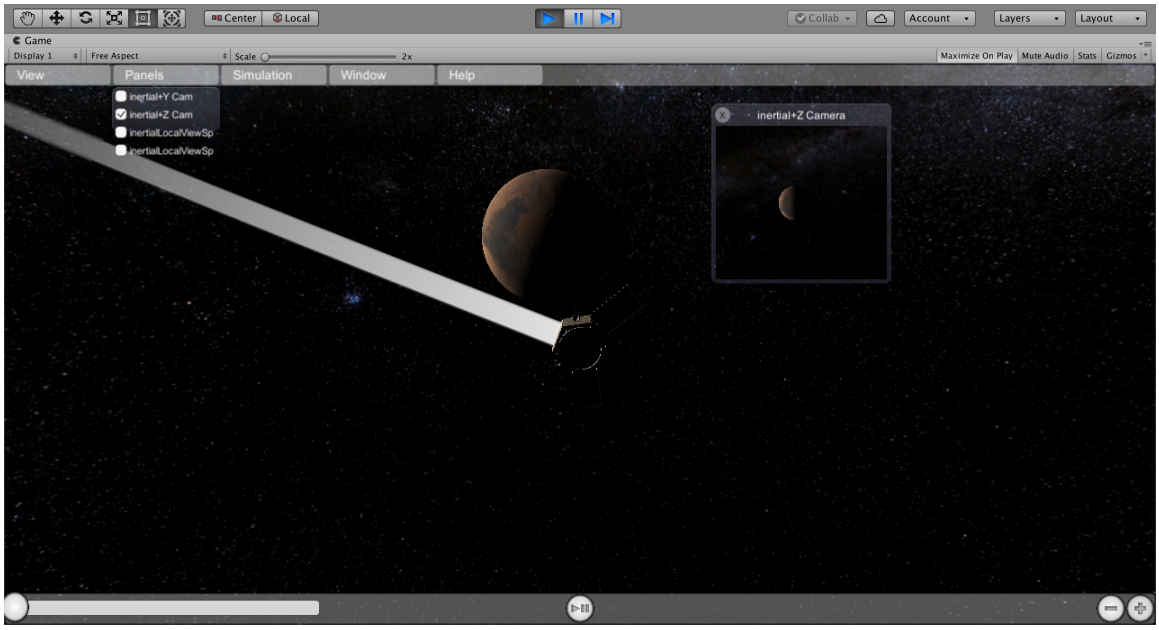

Figure 7. Camera views in the Visualization window

limb of a body and use the knowledge of its actual size and shape. By extracting direction and distance, the spacecraft's location relative to the body are observable.[6] For further development, using just a star tracker, the shape of the partially illuminated moon allows us to estimate the direction vector to the sun.[16] With the knowledge of the body in sight, its ephemeris, and its size, determination of both the spacecraft's orbit and attitude can be achieved. These methods require minimal image processing power, are relatively fast to implement, and provide a wealth of extractable information from images. The use of *OpenCV* has helped accelerate module design. For these reasons they will be the baseline methods used in simulating autonomous Optical Navigation. In the Visualization (Figure 7) planets can appear in the camera window with all of the background star field.

However, other OpNav methods yield better navigation results. Measurements derived from landmark observations are to be processed onboard the spacecraft in an Extended Kalman Filter (EKF).[5] Uniform distribution of points via a "icosahedron pixelization" method,[17] amongst other feature tracking methods provide promising results. This comes at a computation cost. The real-time component of this framework creates a realistic environment to quantify and run more computationally extensive algorithms.

This software framework allows for rapid and high-fidelity testing, and can centralize progress from other fields within an astrodynamics framework. In the aerospace fields, this has been seen with ORB-SLAM development[18,19] and other cross-correlation methods.[9,20] These hold great promise for small body autonomous orbiting and have already proven to be useful on missions such as Rosetta. Further progress for these methods demands a high-fidelity simulation environment. Although advanced goals, this architecture allows for these developments. With exterior work on shape models construction, the goal is to centralize progress for small body identification and navigation within the same simulation. The *Basilisk* Visualization also allows one to upload shape models for any celestial body. A shape model of Vesta was added to the visualization in Figure 8, displaying the images made available by the simulation. This provides the opportunity to train and test shape model reconstruction methods by using fully coupled spacecraft attitude and orbital dynamics.

In conclusion, centroid tracking and apparent diameter measurements are the baseline OpNav methods in the design. In parallel, developments for feature tracking will be added in along with more image processing capabilities. The scene will be set for future improvements, notably using SLAM methods for small body navigation.

A TESTBED FOR NOVEL NAVIGATION METHODS

As seen in the previous subsection, this software's modularity and configurability can be applied to a wealth of missions. On a more fundamental level, it also changes the tools that can be used for mission design.

On-board Monte Carlo Analysis

Due to the highly non-linear, multi-parameter, and uncertain scenarios, Monte-Carlo analysis has become the de-facto method to prove a spacecraft design. While it provides invaluable information on the system, it is suboptimal when it comes to speed. Novel, smarter Monte-Carlo methods are in development and *Basilisk* provides the ability to run fast, parallelized simulation runs. If coupled with better Monte-Carlo techniques and better on-board processors, this could lead to the possibility of on-board Monte-Carlos. The spacecraft could therefore predict a dangerous decision or decide

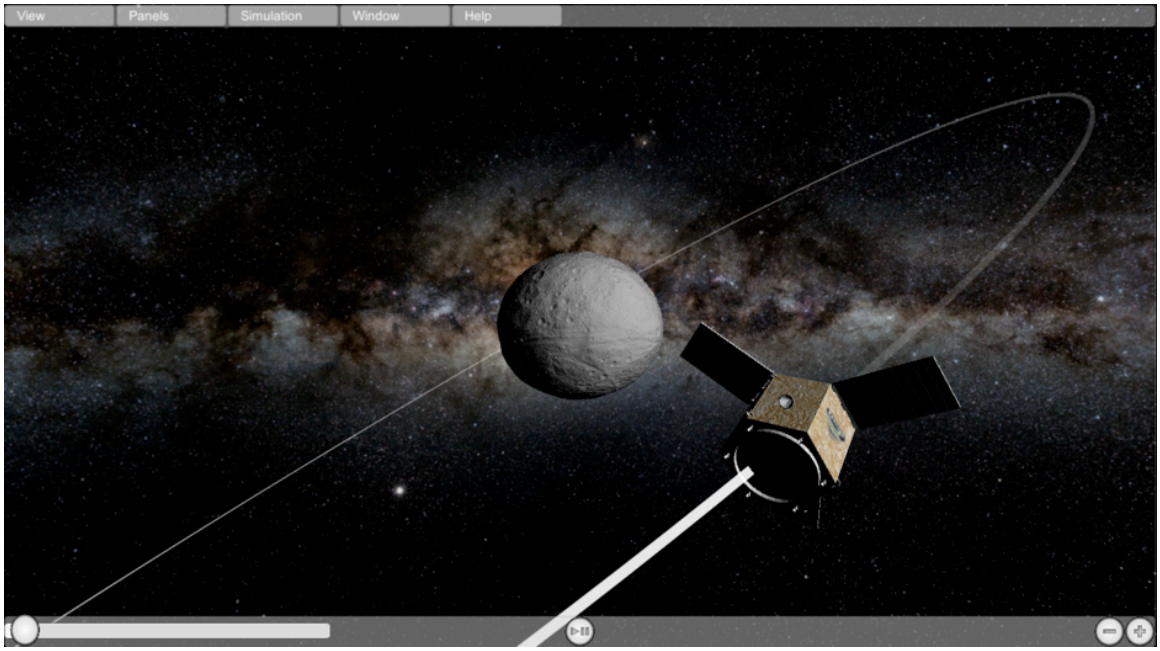

Figure 8. Vesta shape model uploaded into the Visualization

to pursue a maneuver that is predicted to be nominal given initial covariances and uncertainties.

Machine Learning

Machine learning has recently revolutionized data processing and handling. Reinforced learning allows for fault detection, and spacecraft operating mode changes in an autonomous way. This does require large amounts of training data, which is not a common resource in the aerospace field. This is where the OpNav simulation framework could come into play. Harnessing its speed and high-fidelity, Bayesian nets or neural nets could be trained over a large set of simulations. These trained nets could then be tested on unknown scenarios to see how they perform, notably in the face of anomaly. This is currently being developed within the AVS lab, along with a growing machine learning expertise.

Reinforcement Learning can also be paired with error classification. An agent can be trained in scenarios where sun-sensors, camera head units, wheels, or thrusters fail. This can lead to all-in-one fault detection and navigation algorithms where a learner will be resilient to faults and push autonomy forward.

CONCLUSIONS

This software architecture shall provide a testbed for modern Optical Navigation and novel navigation methods. Through the closed loop, coupled interaction between the simulation and the visualization, scenarios can provide high-fidelity data at fast rates. Amongst other improvements, future work in this architecture will open the door to Machine Learning techniques, and Monte Carlo analysis.

This framework hopes to push autonomy, notably for fly-by and orbit insertion maneuvers. Using a simulated camera, optical navigation methods, and the closed loop Visualization-Simulation

interaction, these scenarios can be tested using solely OpNav. In parallel to hardware testing and experiments, this software package will help push the envelope when implementing SLAM and shape reconstruction techniques as well.

ACKNOWLEDGMENT

The authors would like to acknowledge Mar Cols Margenet, Patrick Kenneally, and Jennifer Wood for *Black Lion* and *Unity* development.

REFERENCES

[1] S. Li, "Image Processing Algorithms For Deep-Space Autonomous Optical Navigation," *The Journal of Navigation*, Vol. 66, No. 605-623, 2013.

[2] J. Christian, "An On-Board Image Processing Algorithm for a Spacecraft Optical Navigation Sensor System," AIAA Space Conference and Exposition, Anaheim, CA, AIAA, 2010.

[3] J. Christian, "Accurate Planetary Limb Localization for Image-Based Spacecraft Navigation," *Journal of Spacecraft and Rockets*, Vol. Vol. 54, No. No. 3, 2017, pp. pp 708–730.

[4] M. Dor and P. Tsiotras, *ORB-SLAM Applied to Spacecraft Non-Cooperative Rendezvous*. American Institute of Aeronautics and Astronautics, 2018/05/21 2018, doi:10.2514/6.2018-1963.

[5] A. Liounis, "Autonomous Navigation System Performance in the Earth-Moon System," AIAA Space Conference and Exposition, San Diego, CA, AIAA, September 2013.

[6] J. Christian, "Optical Navigation Using Planet's Centroid and Apparent Diameter in Image," *Journal of guidance, control, and dynamics*, Vol. 38, No. 2, 2015.

[7] J. Alcorn and H. Schaub, "Simulating Attitude Actuation Options Using the Basilisk Astrodynamics Software Architecture," *67th International Astronautical Congress*, Guadalajara, Mexico, Sept. 26–30 2016.

[8] J. Wood, M. C. Margenet, P. Kenneally, H. Schaub, and S. Piggott, "Flexible Basilisk Astrodynamics Visualization Software Using the Unity Rendering Engine," *AAS Guidance and Control Conference*, Breckenridge, CO, February 2–7 2018.

[9] J. M. Carson, C. Seubert, F. Amzajerdian, C. Bergh, A. Kourchians, C. Restrepo, C. Y. Villalpando, T. O'Neal, E. A. Robertson, D. F. Pierrottet, G. D. Hines, and R. Garcia, *COBALT: Development of a Platform to Flight Test Lander GN&C Technologies on Suborbital Rockets*. American Institute of Aeronautics and Astronautics, 2018/01/28 2017, doi:10.2514/6.2017-1496.

[10] M. C. Margenet, P. Kenneally, and H. Schaub, "Software Simulator for Heterogeneous Spacecraft and Mission Components," *AAS Guidance and Control Conference*, Breckenridge, CO, February 2–7 2018.

[11] M. Cols Margenet, P. W. Kenneally, H. Schaub, and S. Piggott, "Simulation Of Heterogeneous Spacecraft And Mission Components Through The Black Lion Framework," *John L. Junkins Dynamical Systems Symposium*, College Station, TX, May 20–21 2018. No. 7.

[12] T. Teil and H. Schaub, "Comparing Coarse Sun Sensor Based Sequential Sun-Heading Filters," *AAS Guidance and Control Conference*, Breckenridge, CO, February 2–7 2018.

[13] W. Owen, "Methods of Optical Navigation," 2011.

[14] J. Christian, "Onboard Image-Processing Algorithm for a Spacecraft Optical Navigation Sensor System," *Journal of spacecraft and rockets*, Vol. 49, No. 2, 2012.

[15] M. Psiaki, "Autonomous Lunar Orbit Determination using Star Occultation Measurements," Guidance, Navigation and Control Conference and Exhibit, Hilton Head, SC, AIAA, August 2007.

[16] J. Enright, "Moon-Tracking Modes for Star Trackers," *Journal of guidance, control, and dynamics*, Vol. 22, No. 1, 2010.

[17] M. Tegmark, "An Icosahedron-based Method for Pixelizing the Celestial Sphere," Vol. 470, The Astrophysical Journal Letters, 1996, pp. L81–L84.

[18] R. Mur-Artal, J. M. M. Montiel, and J. D. Tardós, "ORB-SLAM: A Versatile and Accurate Monocular SLAM System," *IEEE Transactions on Robotics*, Vol. 31, Oct 2015, pp. 1147 – 1163.

[19] R. Mur-Artal and J. D. Tardós, "ORB-SLAM2: an Open-Source SLAM System for Monocular, Stereo and RGB-D Cameras," *CoRR*, Vol. abs/1610.06475, 2016.

[20] A. M. S. Martin, D. S. Bayard, D. T. Conway, M. Mandic, and E. S. Bailey, "A Minimal state augmentation algorithm for vision-based navigation without using mapped landmarks," *GNC 2017: 10th International ESA Conference on GNC Systems*, Vol. 10, Salzburg, Austria, May 2017.