IAC-16-C1.1.4

SIMULATING ATTITUDE ACTUATION OPTIONS USING THE BASILISK ASTRODYNAMICS SOFTWARE ARCHITECTURE

John Alcorn[a]*, Hanspeter Schaub[b], Scott Piggott[c], Daniel Kubitschek[d]

[a] *Colorado Center for Astrodynamics Research, University of Colorado Boulder, 431 UCB, Boulder, Colorado, United States, 80309, john.alcorn@colorado.edu*
[b] *Colorado Center for Astrodynamics Research, University of Colorado Boulder, 431 UCB, Boulder, Colorado, United States, 80309, hanspeter.schaub@colorado.edu*
[c] *Laboratory for Atmospheric and Space Physics, University of Colorado Boulder, 1234 Innovation Drive, Boulder, Colorado, United States, 80303, scott.piggott@lasp.colorado.edu*
[d] *Laboratory for Atmospheric and Space Physics, University of Colorado Boulder, 1234 Innovation Drive, Boulder, Colorado, United States, 80303, daniel.kubitschek@lasp.colorado.edu*
* Corresponding Author

Abstract

The Basilisk astrodynamics software architecture is being designed to be capable of both faster-than realtime simulations, including repeatable Monte-Carlo simulation options, as well as providing real-time options for hardware-in-the-loop simulations. The Basilisk package is designed as a set of Python modules written in C/C++ which allows for the ease of scripting and reconfigurability of Python while still providing the execution speed of C/C++. The software is being developed jointly by the University of Colorado Autonomous Vehicle Systems laboratory and the Laboratory for Atmospheric and Space Physics. The resulting framework is targeted for both astrodynamics research as well as sophisticated mission-specific vehicle simulations that include hardware-in-the-loop scenarios. This paper discusses how Basilisk is used to simulate attitude actuation devices such as reaction wheels (RWs) and attitude control system (ACS) thrusters. A modular framework allows for the these devices to be readily included in the dynamic simulation, and thus test candidate Attitude Determination and Control System (ADCS) control strategies. The RWs are modeled to include misalignment and mass imbalance properties. The thruster model simulates a minimum thruster impulse, thruster on and off ramping, as well as a finite thruster servo frequency. The paper discusses how the RW and ACSs components are simulated and analyzed for maneuvers such as a deep space sun-pointing attitudes with RWs, or a Mars orbit insertion with the ACS thrusters.

1. Introduction

Analysis through simulation and hardware-in-the-loop (HWIL) testing have become essentials of modern spacecraft attitude determination and control system (ADCS) development. The primary benefits of using these tools are improvement in the quality of design and testing by reducing cost and duration of development. Factor of safety for a design may be fine-tuned as needed by virtually testing and verifying the spacecraft at failure conditions. In-flight anomalies and "what if" situations may be explored via Monte Carlo tools in pure simulation. HWIL allows evaluation of hardware in a controlled, low-burden environment to expose technical faults and spacecraft integration problems before millions of dollars are spent launching the system to space.[1–3]

Numerous spacecraft astrodynamics simulation softwares exist and have various features and strengths/weaknesses. State-of-the-art commerical-off-the-shelf (COTS) and government-off-the-shelf (GOTS) softwares each have unique strengths, but present limited capability to provide a complete physically realistic dynamical representation of a spacecraft capable of being used for research with the purpose of ADCS design analysis while allowing user-friendly, platform independent interaction. Additionally, many of these software solutions are prohibitively expensive for low-budget missions or student development. Open-source softwares/freeware may be poorly maintained and/or not user friendly, requiring more time to setup and learn how to use than is available for a particular mission. Most importantly, few COTS/GOTS softwares single-handedly allow for dually-integrated capabilities of realtime HWIL testing and standalone faster than realtime simulation with built-in repeatable Monte Carlo simulation tools in

a cross-platform environment. Mathworks's Simulink[1] and National Instruments LabVIEW[2] provide excellent platforms for building quick models of control systems for faster than realtime simulation and HWIL simulations. Each of these softwares can even be used to produce C code directly from their drag and drop environment with the use of add-on packages. Downsides of these softwares are lack of a built-in visualization, cost, and the necessity of a fairly experienced user. Softwares such as AGI's Systems Toolkit[3], a.i. solutions's Freeflyer[4], and Applied Defense Solutions's Spacecraft Design Tool[5] all provide beautiful visualizations and highly efficient standalone simulation tools, but are geared mainly towards analysis and development of orbits, mission design, and complex situational analysis. STK does provide a six-degree-of-freedom option by providing for state propagation of spacecraft attitude dynamics, but relies on external torque generation and has no way of implementing accurate sensor or actuator models during its dynamic simulation. Furthermore, none of these three software packages are conditioned for HWIL testing. Softwares such as such as NASA's Trick[6] and CS's Orekit/Rugged[7] are both specifically made for spacecraft attitude dynamics simulation. These softwares fall short of providing a fully-coupled dynamic representation of a spacecraft with complex actuators such as reaction wheels. Various other open-source softwares exist for attitude dynamics simulation but are either poorly documented, unmaintained, or lack critical features for all-in-one ADCS development software.[4]

The Basilisk astrodynamics software is being developed by the University of Colorado Boulder's Autonomous Vehicle Systems (AVS) Laboratory and the Laboratory for Atmospheric and Space Physics (LASP). Basilisk provides deterministic, integrated faster than realtime simulation while at the same time providing HWIL simulation capabilities using a modular and fast C/C++ architecture. This source code is then wrapped in Python allowing the convenience of a fully scriptable Python user interface. The modular architecture and fully-coupled dynamical representation allows for complex actuators such as imbalanced reaction wheels to be simulated without sacrificing accuracy. Basilisk has been used internally by the University of Colorado/LASP for simulation of flexible dynamics,[5] fuel slosh,[6] reaction wheel jitter,[7] thrust pulsing algorithm evaluation,[8] guidance algorithm development,[9] and for analysis and support of ADCS subsystem developments.

This paper includes an overview of the Basilisk architecture and discussions of unique features. Actuator modules of Basilisk, including reaction wheels and thrusters, are discussed within the context of both pure simulation and FSW. An example of the Basilisk visualization and a discussion of visualization options are included.

2. Basilisk Overview

The Basilisk framework has been designed from inception to support several different (often competing) requirements.

- **Speed:** Even though the system is operated through a Python interface, the underlying simulation executes entirely in C/C++ which allows for maximum execution speed. The requirement for the Mars mission are sufficiently accurate vehicle simulation with at least a 365x realtime speeds ("*a year in a day*").
- **Reconfiguration:** The user interface executes natively in Python which allows the user to change task-rates, model/algorithm parameters, and output options dynamically on the fly.
- **Analysis:** Python-standard analysis products like numpy[8] and matplotlib[9] are actively used to facilitate rapid and complex analysis of data obtained in a simulation run without having to stop and export to an external tool. This capability also applies to the Monte-Carlo engine available natively in the Basilisk framework.
- **HWIL:** Basilisk provides synchronization to realtime via clock tracking modules. This allows the package to synchronize itself to one or more timing frames in order to provide deterministic behavior in a realtime environment. External communication is handled via the Boost library[10] with ethernet currently available and serial planned in the near future.

Figure 1 shows the Basilisk logo. The name Basilisk was chosen to reflect both the reptilian (Python) nature of the product-design as well as a nod to the speed requirements as the South American common basilisk runs so fast that it can even run across water.

Figure 2 shows a diagram of the Basilisk software architecture. The Python user interface layer allows the simulation to be easily reconfigured which allows the user complete freedom in creating their own simulation modules and FSW modules. Scenario scripts utilizing the user-defined simulation can be used to configure spacecraft properties, initial conditions, events, and various simulation parameters such as timing. The Python user

[1] http://www.mathworks.com/products/simulink/
[2] http://www.ni.com/labview/
[3] http://www.agi.com/products/stk/
[4] https://ai-solutions.com/freeflyer/
[5] http://www.applieddefense.com/products-and-services/sdt/
[6] https://github.com/nasa/trick
[7] https://www.orekit.org/
[8] http://www.numpy.org/
[9] http://matplotlib.org/
[10] http://www.boost.org

Fig. 1: The Basilisk logo.

interface layer abstracts logging/analysis which allows a single compilation of the source code to support completely different simulations. Most simulation modules are written in C++ to allow for object-oriented development while the FSW modules are written in C to allow for easy portability to flight targets. Simulation modules and FSW modules communicate through the message passing interface (MPI), which is a singleton pattern. The MPI allows data traceability and ease of test. Figure 3 shows a MPI data map, which allows the user to visualize data flow between modules. Modules are limited in their ability to subscribe to messages and write messages, thus setting limitations on the flow of information and the power of modules to control data generation. The messaging system is also instrumented to track data exchange, allowing the user to visualize exactly what data movement occurred in a given simulation run. The Python interface to the C/C++ layer relies on the Simplified Wrapper and Interface Generator (SWIG) software[1], a cross-platform, open-source software tasked solely with interfacing C/C++ with scripting languages. Basilisk is inherently cross-platform in nature, currently used on Mac, Windows, and Linux systems.

Basilisk has an accompanying visualization that uses Qt[2]/OpenGL[3] to visualize the spacecraft, planets, and various qualitative data and indicators for sensors and actuators. Simulation events and device faults may be triggered directly from the visualization.

3. Basilisk Actuator Simulation

This section discusses the actuator modules within Basilisk to provide examples of core capabilities. Discussed are reaction wheel simulation model options, including the simplified jitter model and fully-coupled jitter model. Simulation module of discrete on/off thrusters

[1] http://swig.org/
[2] https://www.qt.io/
[3] https://www.opengl.org/

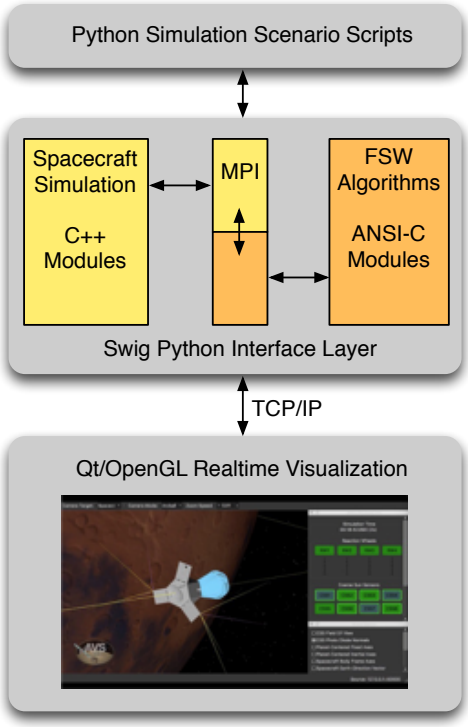


Fig. 2: Basilisk architecture diagram.

3.1 Simulating Reaction Wheels

The Basilisk software hosts multiple reaction wheel models of wheel mass imbalance induced angular jitter. The imbalance models may be used mainly as an analysis tool specifically for validating the requirements of imaging, jitter sensitive instruments, and structural dynamics of the spacecraft. A simplified imbalance model may be enabled to obtain a quick estimate of the attitude disturbances caused by jitter. The simplified model applies forces and torques to the spacecraft as external disturbances.[10–12] This model, although sufficient for a rough estimate, does not allow angular momentum and energy to be conserved as basic physics says they must be.[13] Basilisk has the option of enabling a fully-coupled reaction wheel imbalance model that provides for physically realistic analysis of wheel jitter. The fully-coupled jitter model is a full dynamical representation of static and dynamic imbalance and incorporates the full inertia tensor of the reaction wheel and center of mass location in order to let jitter act as internal forces and torques, thus allowing energy and momentum to be conserved.[7] Figures 4-5 show sample data produced using Basilisk's fully-coupled reaction wheel jitter model. Figure 4 depicts the overall effect of jitter from three reaction wheels by plotting principle angle versus time. Figure 5 shows the effect of jitter on the spacecraft body rates. Additionally, unlike the simplified jitter model the dynamics also cause the reaction wheels to be back-effected by jitter causing realistic fluctuations in wheel speed to be modeled. Reaction wheel

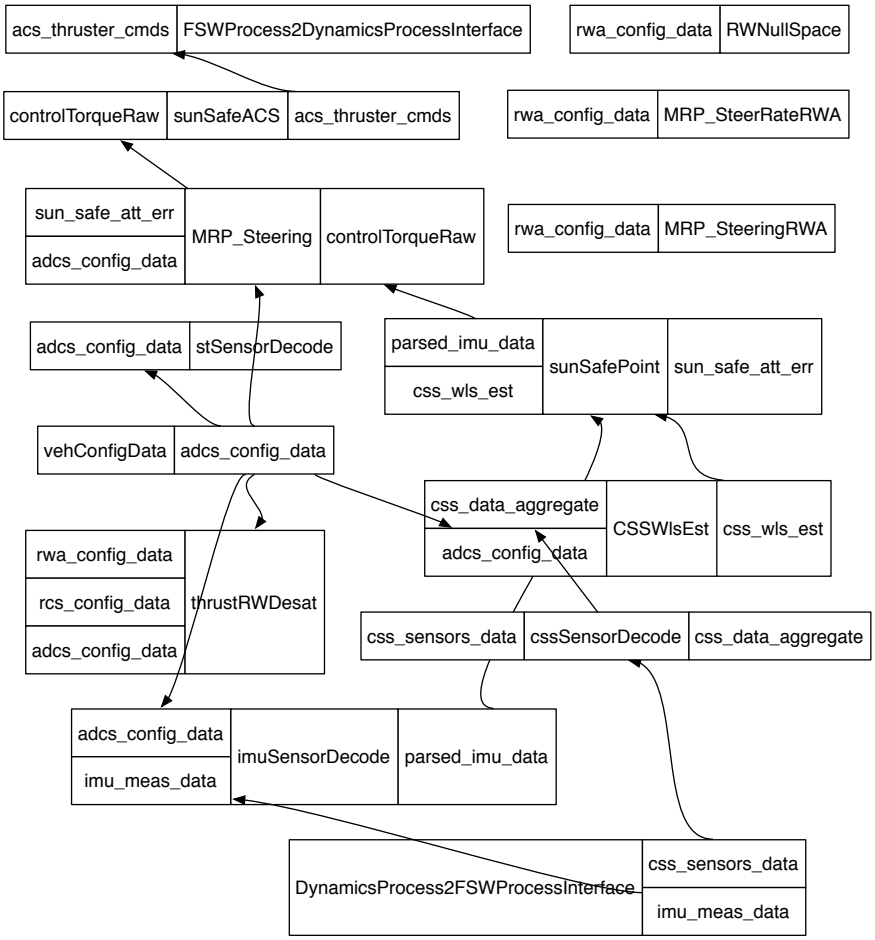


Fig. 3: Basilisk Message Passing Interface data map.

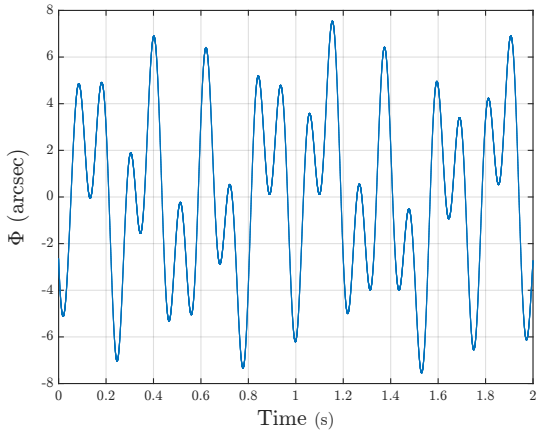


Fig. 4: Angular jitter of the spacecraft for the fully-coupled reaction wheel jitter simulation

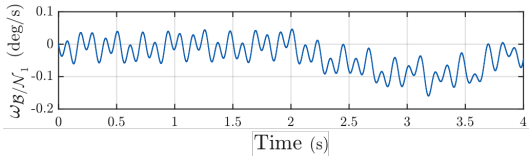
jitter options are fully scriptable from the Python layer of Basilisk.

3.2 Simulating Thrusters

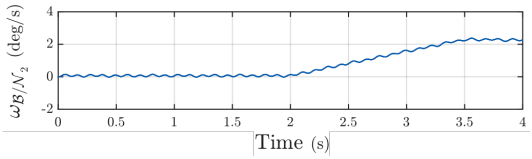
Basilisk's built-in thruster simulation module incorporates features such as on/off ramp, minimum impulse bit, thruster servo limitations, and saturation. These features are important for accurate simulation of temporal restrictions on thruster based actuation. On-ramping and off-ramping are implemented to simulate inherent transient behaviors in the initial firing and shutting off of the thrusters. Basilisk models these transients as first degree functions dependent on thruster test data, which very closely approximates the realistic behavior of thrusters. The implementation of the thruster's minimum impulse bit further restricts the initial firing characteristics. The minimum impulse bit is modeled as a minimum ON time for the thrusters so that they may not fire less than this amount of time.

4. Basilisk Actuator Control Evaluation

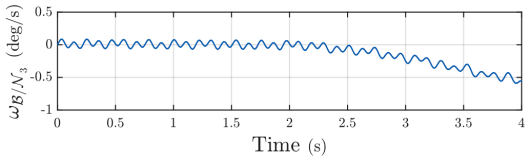
The FSW layer of Basilisk has a modular architecture that allows multiple modules to be included and configured at the Python user interface layer with the purpose of testing candidate ADCS control and actuation strate-



(a) x-axis angular velocity.



(b) y-axis angular velocity.



(c) z-axis angular velocity.

Fig. 5: Body rates of the spacecraft for the fully-coupled reaction wheel jitter simulation.

gies. Included in this section are discussions of discrete thruster firing modules and reaction wheel torque mapping schemes and signal conditioning/outlier rejection.

4.1 Thruster Pulsing Algorithms

A core feature of Basilisk is the ability to enable/disable modules from the Python-level. As applied to thruster pulsing algorithms, this allows the designer to trade different algorithms against each other by just setting flags in the Python scenario script. Figure 6 shows a diagram of thruster FSW modules, and how they connect to other FSW modules and the simulation aspect of Basilisk. The thrust mapping module is tasked with converting the control torque $\boldsymbol{L}_\mathrm{r}$ into individual thrust force $\boldsymbol{F}_i$ commands for each thruster. The thrust pulsing algorithms are purposed with mapping desired thruster force to ON time commands T_{ON_i} for each thruster. These algorithms are a necessity since many spacecraft operate using discrete ON/OFF thrusters that have operating characteristics such as minimum pulse time and pulse time resolution. The modularized architecture allows candidate ADCS control strategies to be readily tested using different torque mapping schemes and thruster pulsing algorithms. The availability and ease of implementation of these algorithms allows for optimization of propellant usage, thruster fire count, and steady state pointing accuracy. Current thruster pulsing modules implemented in Basilisk include multiple pulse rounding methods and the Schmitt trigger method.[8]

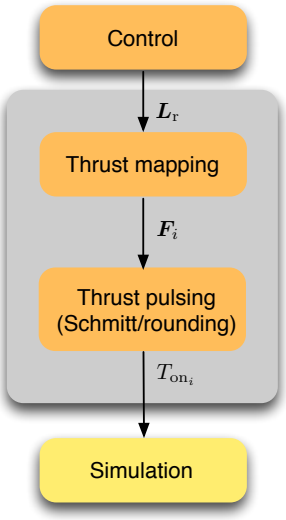


Fig. 6: Thruster FSW algorithms.

4.2 RW Torque Mapping and Signal Conditioning

Basilisk features multiple options for mapping attitude control law torque to reaction wheel motor torque and additional signal conditioning thereafter. Figure 7 shows a diagram of reaction wheel FSW modules, and how they connect to other FSW modules and the simulation aspect of Basilisk. Reaction wheel torque mapping typically involves using a minimum-norm inverse formulation to convert the control torque $\boldsymbol{L}_\mathrm{r}$ to individual reaction wheel motor torque commands u_{s_i}.[13] A power-optimal torque mapping formulation allows for reduction of the overall amount of energy and mechanical power required for a redundant set of reaction wheels.[14] An accompanying power-optimal control law for a redundant reaction wheel configuration may be implemented in cases where power is especially limited, such as with small satellites.[15] This control law allows for energy recuperation and reduced wheel speeds by applying reaction wheel torque null motion.

5. Simulation Example

5.1 Visualization

Basilisk has an accompanying visualization that allows the user to observe the simulated spacecraft's operations realtime. Figure 8 shows an example of the visualization with a spacecraft in Mars orbit. The control panel on the right hand side allows the user to view sensor and actuator data and trigger events. Reference frame axes may be enabled/disabled from the control panel. The visualization also demonstrates thruster plumes and the field of view of sensors such as star trackers and course sun sensors. Figure 9 is a demonstration of the Monte Carlo capabilities of Basilisk. The plot provided shows body rates of the spacecraft plotted versus time for a number of simulation runs. The individual runs of the simulation are plotted

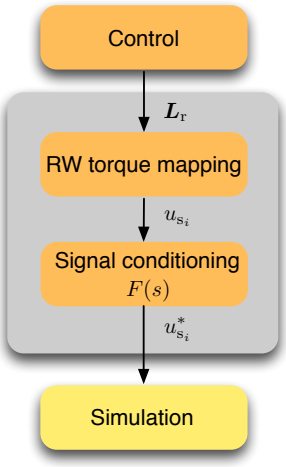


Fig. 7: Reaction wheel FSW algorithms.

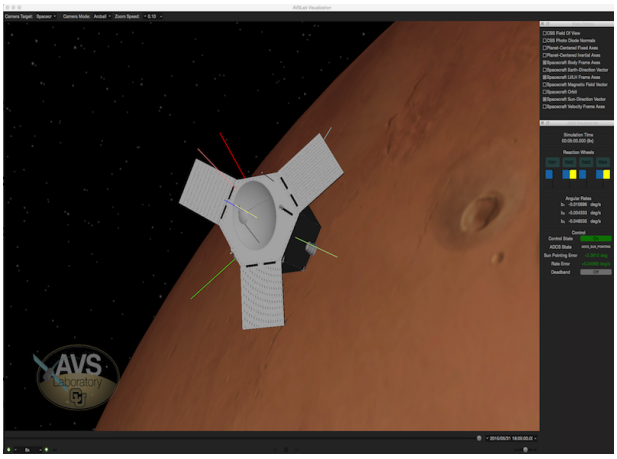
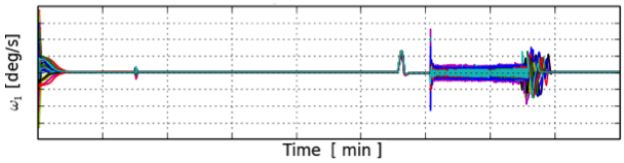


Fig. 8: Basilisk visualization example of a Mars orbiting spacecraft.

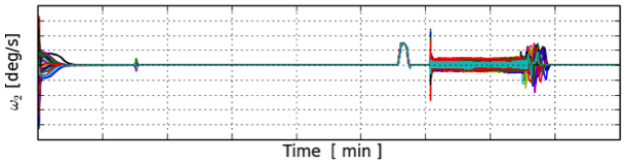
with different colors. This allows direct comparison of simulation runs with varying initial conditions and testing of worst case scenarios.

6. Conclusions

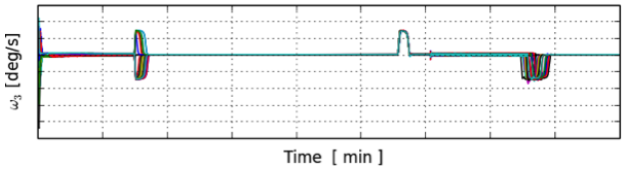
The general software architecture of the Basilisk astrodynamics software and specific modules of actuators are described. It is shown that Basilisk offers many of the same core benefits as Commercial-off-the-Shelf (COTS)/Government-off-the-Shelf (GOTS) softwares and is open-source, cross-platform, and has a fully-scriptable user interface using the common programming language Python. The discussion shows that Basilisk is able to simulate "*a year in a day*" due to its usage of C/C++ wrapped by the Python scenario scripting layer using SWIG. Additionally, Basilisk's handling of spacecraft dynamics allows for a fully-coupled, physically-realistic representation of the spacecraft's motion under the influence of complex dynamics such as imbalanced reaction wheels and fuel slosh. It is the authors' belief that the core-features



(a) x-axis angular velocity.



(b) y-axis angular velocity.



(c) z-axis angular velocity.

Fig. 9: Basilisk Monte Carlo simulation example data.

of Basilisk match the aggregate benefits of all other maintained open-source spacecraft dynamics projects. Future work to be completed on Basilisk includes multiple spacecraft configurations with the purpose of formation flying research.

Acknowledgements

Special thanks go to the other members of the Basilisk development team supporting the presented developments, including Cody Allard, Patrick Kenneally, Mar Cols Margenet, and Manuel Diaz Ramos.

REFERENCES

[1] L. Slafer. The use of real-time, hardware-in-the-loop simulation in the design and development of the new hughes hs601 spacecraft attitude control system. 1989.

[2] Jesse Leitner. Space technology transition using hardware in the loop simulation. In *Aerospace Applications Conference, 1996. Proceedings., 1996 IEEE*, volume 2, pages 303–311. IEEE, 1996.

[3] A. Ptak and K. Foundy. Real-time spacecraft simulation and hardware-in-the-loop testing. In *Real-Time Technology and Applications Symposium, 1998. Proceedings. Fourth IEEE*, pages 230–236. IEEE, 1998.

[4] Andrew J Turner. *An open-source, extensible spacecraft simulation and modeling environment framework*. PhD thesis, Citeseer, 2003.

[5] C. Allard, H. Schaub, and S. Piggott. General hinged solar panel dynamics approximating first-order spacecraft flexing. In *AAS Guidance and Control Conference*, Breckenridge, CO, Feb. 5–10 2016. Paper No. AAS-16-156.

[6] C. Allard, M. Diaz Ramos, and H. Schaub. Spacecraft dynamics integrating hinged solar panels and lumped-mass fuel slosh model. In *AIAA SPACE 2016*, Long Beach, CA, Sep. 13–16 2016. Paper No. 2490836.

[7] J. Alcorn, C. Allard, and H. Schaub. Fully-coupled dynamical jitter modeling of a rigid spacecraft with imbalanced reaction wheels. In *AIAA SPACE 2016*, Long Beach, CA, Sep. 13–16 2016. Paper No. 2490836.

[8] John Alcorn, Hanspeter Schaub, and Scott Piggott. Attitude control performance analylsis using discretized thruster with resdidual tracking. In *AAS Guidance and Control Conference*, Breckenridge, CO, Feb. 5–10 2016. Paper No. AAS-16-038.

[9] M. Cols Margenet, H. Schaub, and S. Piggott. Modular attitude guidance development using the basilisk software framework. In *AIAA SPACE 2016*, Long Beach, CA, Sep. 13–16 2016. Paper No. 2490836.

[10] F. L. Markley and J. L. Crassidis. *Fundamentals Of Spacecraft Attitude Determination And Control*, volume 33. Springer, 2014.

[11] L. Liu. Jitter and basic requirements of the reaction wheel assembly in the attitude control system. 2007.

[12] D.K. Kim. Micro-vibration model and parameter estimation method of a reaction wheel assembly. *Journal of Sound and Vibration*, 333(18):4214–4231, 2014.

[13] H. Schaub and J. L. Junkins. *Analytical Mechanics of Space Systems*. AIAA Education Series, Reston, VA, 3rd edition, 2014.

[14] H. Schaub and V. J. Lappas. Redundant reaction wheel torque distribution yielding instantaneous l_2 power-optimal attitude control. *AIAA Journal of Guidance, Control, and Dynamics*, 32(4):1269–1276, July–Aug. 2009.

[15] R. Blenden and H. Schaub. Regenerative power-optimal reaction wheel attitude control. *AIAA Journal of Guidance, Control, and Dynamics*, 35(4):1208–1217, July–Aug. 2012.