Halo Orbit Integration Into Horizon Simulation Framework: A Literature Review

Christian Fuller

November 2021

Contents

1 Introduction

It is a personal goal to develop knowledge of modeling and simulation particularly where orbital mechanics are concerned. To this end my thesis aims to intersect these two fields through the implementation of the halo orbit regime to the Horizon Simulation Framework. The motivation behind the study of halo orbits is a rising popularity in the aerospace community especially when concerning the upcoming return to the Moon mission Artemis. These types of orbits offer a few advantages, and additionally provide a unique challenge to simulate due to the chaotic nature of the governing equations [1]. The Horizon Simulation Framework offers a valuable platform for implementations of these orbits for a few reasons. First it has a developed scheduling algorithm to propagator forward 'a day in the life' for user defined spacecraft subsystems, which with the inclusion of halo orbits will enhance future projects' subsystem design. Second, it aims to be an open-source application with community driven model development which allows for further refinement of the halo orbit propagator. Finally because HSF is a model based system engineering tool with focus on mission requirement verification, the addition of this orbital family will add to the capabilities and enhance the development of future interplanetary missions.

2 Halo Orbits

Halo orbits result from the interaction between three gravitational bodies. Most commonly the Earth-Moon or Earth-Sun systems are considered [2]. Depending on mission objectives these orbit types can offer a few advantages; for example in the upcoming Lunar Gateway mission the selected halo orbit allows unimpeded communications with Earth ground stations, visibility of the lunar hidden surface, and relatively low cost of station keeping [2]. ADD MORE

The halo orbit family represents a unique challenge to solve given its highly coupled equations. The most common approach is by making the assumption of a circular restricted three body problem [3]. In this process the mass of the third body is assumed to have little effect in the system, and that its dynamics are principally governed by the other two bodies. Additionally the two large masses are presumed to be in a circular orbit around a common barycenter [3]. The barycenter is defined to be the center of mass of the two bodies. There are 5 equilibrium points that result from the equations of motions developed by the above restrictions. Three of which are co-linear and termed the L_1, L_2, and L_3 points, and the other two triangular and termed the L_4 and L_5 points. As diagrammed in Figure 1 where the view is from the positive z direction looking 'down' on the orbital plane of the two masses. Halo orbits are vertically stable periodic paths around these points, and in this case they would come into and out of the page.

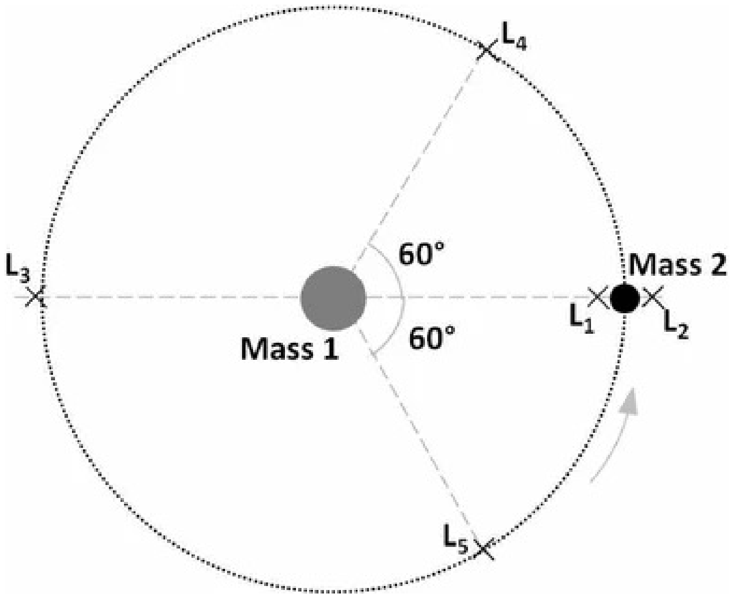

Figure 1: View of Lagrange Points of Two Mass System [4]

3 Model Based System Engineering

A simple definition of a system is a set of entities that interact or are interrelated [5]. Model based systems engineering is a way of expressing these entities in a graphical manner to aid in engineering tasks. Formally, Model based system engineering (MBSE) is the technique of using modeling to aid system requirement, design, analysis, verification, and validation of a given complex problem [6]. This technique arose out of increasing demand for multifaceted highly engineered systems in which previous approaches to complexity management were deemed inadequate [7]. The Systems engineering process usually has the following components: problem definition, deriving system requirements, performing analysis, and then compiling candidate solutions [7]. Model based systems engineering enhances all aspects of the aforementioned process by providing a way to represent all aspects of the system in a centralized diagram, using a common language to standardize the approach, and allowing for an iterative approach to problem solving [7]. There are a multitude of ways to implement MBSE in a contemporary engineering environment [6].

3.1 SysML™ Based Approaches to MBSE

Many modern applications of MBSE implement the Systems Modeling Language (SysML™) which is based on the Unified Modeling Language(UML) the former developed by the Object Modeling Group(OMG) in 2007 [8][9]. SysML enables development of a descriptive hierarchy in which aspects of the system are modeled as blocks. These blocks are descriptions in themselves and interconnected to show relationships and dependencies between elements [7]. However, SysML is limited by the inability to carry model transformation natively, and instead typically implements other scripting languages [7]. For a proprietary solution some model analysis is done through Matlab ® and for high-fidelity space environments, Systems Tool Kit ®[6]. Notable limitations to this approach is that these tools require specialized knowledge, usually acquired through proprietary training, that each implementation typically requires a custom model be developed, and there is significant monetary cost associated with each one [6]. To this end, an attempt has been made to incorporate SysML model diagrams into the open-source Horizon Simulation Framework [10]. However, the success of such implement depends heavily on the quality of SysML input models as HSF directly interprets models designed there [10].

3.2 Capella Based Approaches to MBSE

A recent addition to the open-source MBSE environment is Capella. This software was developed in 2015 by then PolarSys Organization [5]. A motivating factor in its development was the general difficulty in learning SysML® for system engineers, since SysML® stems from a software engineering background which has proven difficult for those unfamiliar with the field [5]. To help combat this, Capella has a user interface similar to Microsoft PowerPoint, with drag and drop elements [5]. A key distinction of Capella is that it is a combination of a systems engineering method and a supporting tool that provides syntax [5]. Whereas SysML® is method agnostic, insofar as it merely provides a place to represent whatever methodology chosen by a facilitating system engineer [5]. Capella utilizes the Architecture Analysis and Design Integrated Approach (ARCADIA) which is a MBSE method that aims to define and validate the architecture of complex systems, and allow for iterations in the definition phase of the problem [11]. Further ARCADIA has a few core principles that are enacted through Capella, namely: that all stakeholders share the same information, specialized subject knowledge is categorized as a 'viewpoint' in relation to the requirements from which the proposed architecture is then verified, rules for verification are established early in order to verify solution, and collaboration between levels is supported by interconnected elaboration of models and the models of the different levels are validated to one another[11].

3.3 Verification and Validation Model Based System Engineering

A subset of MBSE applications is in the form of validation and verification of a given systems solution. Verification is defined as a confirmation through objective evidence that the solution performs all intended functions [12]. Validation is confirmation through the application of objective evidence that the solution satisfies stakeholders needs [12]. MBSE satisfies the following definitions of verification and validation if the models developed are significantly high-fidelity enough to accurately model environments and constraints. Model based verification is carried out through three different methods, verification by semantic review, by testing, and by formal methods [13]. In the first method a semantic diagram which shows structures and relationships of the model is reviewed by an appropriate subject matter expert to judge correctness [13]. However this approach is limited to a static analytical model and thus can only judge if the model was built the right way, not that the correct phenomena is modeled [13]. To validate by testing is through propagation of the model, to do so it must be an executable for a dynamic study of how outputs vary over time from expected results [13]. This approach requires a verified test case that includes all inputs, initial conditions and constraints. This approach is limited since it only seeks to uncover errors and not to prove errors don't exist [13]. The final technique is by formal methods, this is the most rigorous form of verification as it requires mathematical proof [13]. This level of verification result holds for any combination of inputs or system states however because of the difficulty in performing this technique it is typically relegated to use on critical system properties [13]. An example of a formal model verification used in MBSE is the checking where numeric or symbolic computation is used to show that a system parameter holds for every possible system state, however this technique is limited since in some applications it is impossible to analyze every possible exception in a given state space [13].

3.4 Validation solutions as applied to Halo Orbit regime

The value in developing MBSE verification tools that include orbital mechanics is inclusive design. Subsystems that depend on specific orbital parameters can then be designed and iterated upon in a holistic fashion. This is necessarily true even in an MBSE environment in which the system diagram and modeling tools are divorced. The validation/verification of high-fidelity orbital dynamics models involves comparing them to externally generated "truths", often this means comparing to other verified tools such as GMAT, or Systems Tool Kit [14]. In practice there are limited validation solutions in specific cases of Halo orbits that allow for integration with MBSE environments, especially when stipulating open-source solutions. Tools like GMAT and STK require significant effort to learn and implement within desired environments.

4 Open-Source Software

Modern software projects usually are either projects that are open-source in themselves or include libraries from open-source efforts. To better understand why this approach to software development has become an integral part of the field it is useful to formally define the term. Insofar as it applies, open-source is analogous to the revolutionary production line for the Ford motor company in that the novelty of it is the production process, and not the product itself [15]. The process then, is the development of software in an environment that is: first, free and readily available to the public or a privy section, and second presented with some source code to which willing participants can either debug, add capability to, write documentation for, or otherwise contribute to [15]. Whatever the contribution may be, there ought to be a set of guidelines or standards to which additions are checked for quality, less one might have a 'too many cooks in the kitchen' scenario [15]. In other words, without well-defined rules for contribution to the project, it may be pulled along in a tangential direction to its purpose, or may not benefit at all from open-source as no value added work is being done [15]. The main motivation of open-source, then, is the discretization of work into independent contributors with a common or loosely common goal that is accomplished through varied approaches, verified by a set of rules [15]. One might see similarities between open-source and MBSE, since both are efforts to combat the ever imposing rise of complexity in engineering problems [7]. A difference in them is the disjointed nature of contribution, in MBSE there is a defined structure of *who*, namely qualified individuals who are subject area experts, and in open-source one merely needs an internet connection. With this in mind, the quality of an open-source project depends heavily on the constraints it applies to its contributors [15]. The best projects out there have well defined scopes and goals to guide its contributors [15]. In one sense the source code hopes to provide an architecture that allows for and guides implementation through its community[15]. Since the Horizon Simulation Framework aims to be an open-source software it must have a rigorous set of standards of contribution, and well defined scope to be successful in this field.

4.1 Meditations on The Implementation of Novel Capability to HSF

The foundation of building a successful open-source community depends on the definition of scope, and well developed outline for further contribution from the community [15]. In the former context the Horizon Simulation Framework is well defined in objective as it desires to be a MBSE verification tool developed with a holistic modeling of the spacecraft and its requirements through a breadth-first scheduling algorithm. In the latter, and subsequently as an open-source project it is deficient due to its lack of rigorous standards for contribution from its community. For example, say a community member develops an attitude determination and control subsystem model for a novel control-law algorithm, currently there is no published pathway for which this model to be verified and implemented. Part of this work has been carried by Cal Poly masters student, Jack Balfour. This thesis implemented unit tests to many of the utilities within HSF. Once a new subsystem has been included in the project, the unit tests will run a sample mission for each one of the utilities with the inclusion of the new subsystem [Cite Pending] . The purpose of this is to ensure that HSF is still able to compile with the addition. However there is no check for whether this is a value added addition, in other words, it is taken on faith that this new subsystem will be a useful addition to the goal of HSF. In what follows is a preliminary meditation on the standards for implementations of novel contributions to HSF.

In HSF models are either written in a native language of C# or in Python and then interpreted by IronPython. For simplicity sake this distinction will be ignored and instead will focus on a general 'value added' component. In order to give guidance to those who wish to contribute HSF should publish a current state document. This document would outline current issues with the code like known bugs, missing functionality, and any updating efforts [16]. Additionally it would include the capabilities yet to be implemented, short and long term goals, overall vision and guidance on how to ask questions [16][17]. Next HSF should include a code of ethics on how to contribute to the project. The first of the HSF standards should require a description of the added component, i.e. what utilities does it affect, what is its goal, how does one implement it, and what value does it bring to HSF [17]. This could be in the form of a readme or commented code Ṡecond if applicable, the contributor has to run a test on the added code to verify its functionality. Third, a review of the added model's code by a current manager of the source code. This review would include an exception of the code, review of provided documentation, and eventual inclusion to source code.

5 Trajectory Design Tools

Modern methods of orbital design typically use proprietary or in-house developed software, or costly prepackaged tools. One prominent tool in trajectory simulation and design is AGI's Systems Tool Kit ®. This tool has built in orbital models for many different flight cases, including Earth centric, heliocentric, halo orbits, and interplanetary to name a few. Additionally it is able to model spacecraft dynamics and subsystems along the mission timeline, with further capabilities enabled through Matlab®integration [18]. However, it should be noted that STK is not by definition a mission design tool, rather it is considered an orbital propagator and mission analysis tool [18]. The distinction is important as in MBSE the phases of design versus analysis are well defined and mark milestones in the MBSE process. That being said, STK still offers many tools geared to mission design such as its orbital optimization tools, Lambert trajectory tools, and many targeting algorithms [19].

Another framework commonly used is the NASA developed General Mission Analysis Tool (GMAT). [18]. Like STK®, the General Mission Analysis Tool has functionality for many different mission types ranging from LEO to distant retrograde orbits, libration points, and deep space missions [20]. This tool has been used to design trajectories for the ARTEMIS and the Lunar Reconnaissance Orbiter, as well as extensively as a preliminary design tool for various missions [20]. Through internal efforts GMAT has been validated and verified as a high-fidelity design tool through systemic evaluation and regression testing, which is allowed to be more widely adopted as an open-source trajectory design tool with further subsystem modeling capabilities extensible through Matlab ®integration [20].

The Adaptive Trajectory Design is another tool in the realm of trajectory design. It was developed by NASA Goddard Space Flight Center in partnership with Perdue University [21]. The main objective of this software was to provide a GUI based trajectory building tool with pre-developed circular restricted three body paths [21]. A notable feature of this software is the drag and drop sections to create outlines of courses, and the software's ability to interpolate and connect these sections. For a higher fidelity space flight model and ephemeris considerations it is recommended to use a software like GMAT, and as such relegates this tool to early design phases [21].

5.1 Novel development

There exists within the intersection of specialized trajectories and verification through model based system engineering, a lack of an open-source tool which allows and enhances both. Insofar as a tool which offers capabilities to propagate orbital dynamics beyond Earth centric schemes and a holistic approach to system verification and validation. There is value in having an open-source MBSE verification focused framework that has prebuilt models for halo trajectories. Since subsystems design benefits from knowledge of parameters which are specific to the orbital dynamics like temperature, radiation environment, and position relative to ground stations, to name a few, a tool that incorporates these two realms would allow for an enhanced design experience. To this end there has been some effort to develop this kind of tool using HSF. For example Earth based orbital propagators that include major perturbation effects have been implemented in a past thesis [22]. However, the gap in frameworks still exists for halo orbits, which underpins the motivation of my thesis.

6 Conclusion

The goal of this literature review was to lay out the state of the art of model based systems engineering, modeling of halo orbits, open-source software and general trajectory design tools. The thesis statement then follows in context of the aforementioned as: the implementation of halo orbits using circular restricted three body assumptions to add capabilities to the open-source Horizon Simulation Framework which is a model based systems engineering system level verification tool that allows the holistic modeling and further design of spacecraft subsystems. A distinction is made that HSF is a mission analysis and verification tool, but not a mission design tool like STK, GMAT, or The Adaptive Trajectory Design Tool, such that its goal is primarily to accurately propagate the mission, but not provide tools to design it.

References

[1] David A. Vallado and Wayne D. McClain. *Fundamentals of astrodynamics and applications*. 2nd ed. Space technology library ; v. 12. Dordrecht ; Kluwer Academic Publishers, 2001. ISBN: 0792369033.

[2] Giordana Bucchioni and Mario Innocenti. “Rendezvous in Cis-Lunar Space near Rectilinear Halo Orbit: Dynamics and Control Issues”. In: *Aerospace* 8.3 (2021). ISSN: 2226-4310. DOI: 10.3390/aerospace8030068. URL: https://www.mdpi.com/2226-4310/8/3/68.

[3] G. (Gerard) Gómez. *Dynamics and mission design near libration points. Vol. I, Fundamentals : the case of collinear libration points*. World scientific monograph series in mathematics ; vol. 2. Singapore ; World Scientific, 2001. ISBN: 1-281-95629-5.

[4] A Tartaglia et al. “How to use the Sun–Earth Lagrange points for fundamental physics and navigation”. In: *General relativity and gravitation* 50.1 (2017), pp. 1–21. DOI: 10.1007/s10714-017-2332-6.

[5] Shashank P. Alai. “Evaluating ARCADIA/Capella vs. OOSEM/SysML for System Architecture Development”. PhD thesis. Purdue University, 2019.

[6] David Kaslow et al. “Integrated model-based systems engineering (MBSE) applied to the Simulation of a CubeSat mission”. In: (2014), pp. 1–14. DOI: 10.1109/AERO.2014.6836317.

[7] et al. Friedenthal Sanford. *A Practical Guide to SysML : The Systems Modeling Language, Elsevier Science & Technology*. ProQuest Ebook Central, 2011.

[8] Aleksandr A. Kerzhner and Christiaan J. J. Paredis. “Model-Based System Verification: A Formal Framework for Relating Analyses, Requirements, and Tests”. In: (2011). Ed. by Juergen Dingel and Arnor Solberg, pp. 279–292.

[9] Jeff Estefan. “Survey of Model-Based Systems Engineering (MBSE) Methodologies”. In: *INCOSE MBSE Focus Group* 25 (Jan. 2008).

[10] Viren Kishor Patel. “SysML Output Interface and System-Level Requirement Analyzer for the Horizon Simulation Framework”. MA thesis. California Polytechnic State University, San Luis Obispo, 2018.

[11] P. Roques. *Systems Architecture Modeling with the Arcadia Method: A Practical Guide to Capella*. Elsevier Science, 2017. ISBN: 9780081017920. URL: https://books.google.com/books?id=Qj3jCwAAQBAJ.

[12] David Kaslow and Azad Madni. “Validation and Verification of MBSE-Compliant CubeSat Reference Model”. In: Jan. 2018, pp. 381–393. ISBN: 978-3-319-62216-3. DOI: 10.1007/978-3-319-62217-0_27.

[13] C Laing et al. “Questioning integration of verification in model-based systems engineering: an industrial perspective”. In: *Computers in industry* 114 (2020), p. 103163. DOI: 10.1016/j.compind.2019.103163.

[14] Mariana Poderico and Gianfranco Morani. “Validation of Tools for 3Dof Orbital Dynamics Simulation”. In: *American Journal of Engineering and Applied Sciences* 13 (Apr. 2020), pp. 649–657. DOI: `10.3844/ajeassp.2020.649.657`.

[15] Steve Weber. *The success of open source*. Cambridge, MA: Harvard University Press, 2004. ISBN: 0674012925.

[16] *Basic etiquette for open source projects*. Mozilla. 2021.

[17] B. Keepers, Wills S., and M. Linksvayer. *Building Welcoming Communities*. GitHub. 2021.

[18] Abolfazl Shirazi, Josu Ceberio, and Jose A. Lozano. “Simulation Framework for Orbit Propagation and Space Trajectory Visualization”. In: *IEEE Aerospace and Electronic Systems Magazine* 36.8 (2021), pp. 4–20. DOI: `10.1109/MAES.2021.3053121`.

[19] *Systems Tool Kit 12 Help*. AGI. 2021. URL: `https://help.agi.com/stk/index.htm`.

[20] Joel J. K Parker et al. “Verification and Validation of the General Mission Analysis Tool (GMAT)”. In: (2014).

[21] David Folta et al. “Trajectory Design Tools for Libration and Cislunar Environments”. In: Mar. 2016.

[22] Mitra Farahmand. “ORBITAL PROPAGATORS FOR HORIZON SIMULATION FRAMEWORK”. MA thesis. California Polytechnic State University, 2009.