(Preprint) AAS 18-091

MODULAR SOFTWARE ARCHITECTURE FOR FULLY-COUPLED SPACECRAFT SIMULATIONS

Cody Allard[*], Manuel Diaz Ramos[†], Patrick Kenneally[‡], Hanspeter Schaub[§] and Scott Piggott[¶]

Computer simulations of spacecraft dynamics are widely used in industry and academia to predict how spacecraft will behave during proposed mission concepts. Current technology and performance requirements have placed pressure on simulations to be increasingly more representative of the environment and the physics that spacecraft will encounter. This results in increasingly complex computer simulations. Designing the software architecture in a modular way is a crucial step to allow for ease of testing, maintaining, and scaling of the software code base. However, for complex spacecraft modeling including flexible or multi-body dynamics, modularizing the software is not a trivial task because the resulting equations of motion are fully-coupled nonlinear equations. This requires manipulation of the equations of motion to adhere to a modular form. In this paper, a software architecture is presented for creating complex fully-coupled spacecraft simulations with a modular framework. The architecture provides a solution to these common issues seen in dynamics modeling. The modularization of the fully-coupled equations of motion is completed by solving the complex equations analytically such that the spacecraft rigid body translational and rotational accelerations are solved for first, and the other second order state derivatives are found later. This architecture is implemented in the Basilisk astrodynamics software package and is a fully tested example of the proposed software architecture.

INTRODUCTION

An important aspect when considering software design is the scalability, maintainability, and testability of the software.[1] If the software is not designed well, adding complexity (scalability), maintaining functionality amidst a changing code base (maintainability) and the ease of verifying functionality (testability) can become extremely laborsome.[2] For complex simulations of spacecraft, this methodology needs to be considered to avoid these complications. However, multi-body dynamics poses a difficult problem because of the coupled nature of the system through the non-diagonal system mass matrix.[3] This mass matrix relates the dynamical effect of the second order state variables between all of the interconnected bodies.

Although multi-body dynamics is a complex challenge not only from an equation of motion (EOM) development but from a software implementation perspective, there is an abundant amount

[*]Graduate Student, Aerospace Engineering Sciences, University of Colorado Boulder.
[†]Graduate Student, Aerospace Engineering Sciences, University of Colorado Boulder.
[‡]Graduate Student, Aerospace Engineering Sciences, University of Colorado Boulder.
[§]Professor, Glenn L. Murphy Chair, Department of Aerospace Engineering Sciences, University of Colorado, 431 UCB, Colorado Center for Astrodynamics Research, Boulder, CO 80309-0431. AAS Fellow.
[¶]ADCS Integrated Simulation Software Lead, Laboratory for Atmospheric and Space Physics, University of Colorado Boulder.

of open source software packages simulating multi-body dynamics. Bullet[4] is an open source multi-body dynamics software package that utilizes the Gauss-Seidel Method to solve the system mass inverse for diagonally dominant matrices.[5] Project CHRONO is an open source multi-physics software package that utilizes parallel computing to solve multi-body dynamics with a large number of degree's of freedom.[6] Rigid Body Dynamics Library is an open source multi-body dynamics software package that utilizes the Articulated Body Algorithm and Composite Rigid Body Algorithm for solving the dynamics.[7] Adding spacecraft specific environmental factors and incorporating flight software into these open source packages can be laborsome and is not the intended use of these software packages.

In contrast to the open source packages, there are commercial software packages that are solving multi-body dynamics problems. MathWorks Simscape Multibody[8] can generate EOMs to be integrated and can output simulation code to Matlab or C code. Motion Genesis uses Kanes method[9,10] to output simulation code to Matlab, C, or Fortran and includes energy and momentum validation.[11] One downfall of these equation of motion generators is that the equations are specific to that system which introduces scalability, maintainability, and testability issues with software architecture. Additionally, these software packages, although general, present issues with computational efficiency.

In contrast to general multi-body dynamics software, there are software packages that focus only on spacecraft simulation because of the unique environment that spacecraft encounter, and the specific challenges that modeling spacecraft dynamics entails. STK SOLIS is a software package for modeling spacecraft with both translational and attitude dynamics but does not model disturbances that can change the center of mass of the spacecraft, for example flexing solar arrays.[12] The Jet Propulsion Laboratory has a software package, Dynamics Algorithms for Real-Time Simulation (DARTS).[13] This simulation software package utilizes spacial operator algebra for the development of the multibody dynamics[14] to create the system mass matrix in a form that can be solved efficiently with a recursive algorithm.[15] NASA's open source software package named 42,[16] allows for spacecraft composed of multiple rigid or flexible bodies using tree-topology[17] to formulate the dynamics resulting in a system mass matrix inversion solution. Orekit is an open source software package for spacecraft simulations and flight software and models the spacecraft as a rigid body and the dynamics are primarily focused on defining perturbations as external forces and torques.[18]

The software packages described that involve multi-body dynamics have to populate a system mass matrix and either have to find the inverse of the matrix or use other linear algebra techniques.[14,15] Populating the system mass matrix while retaining a modular software architecture is difficult because the system needs to know the locations of the states in the system mass matrix and also know the relative locations of other coupled states. Additionally, inverting the system mass matrix can be computationally expensive because the calculation scales with N^3. To combat these common problems, this paper introduces a method to generalize the EOMs which will be applicable to a wide range of spacecraft configurations, utilizes a back-substitution method to modularize the EOMs, and develops a software architecture that will leverage the modularized equations. While the prior methods allow for general multi-body setups, this method is specifically developed for common spacecraft configurations where there is a single rigid spacecraft hub onto which additional bodies are attached. This assumption is a key enabler that leads to an elegant modular framework that can be implemented in numerical simulations without dropping any dynamical coupling between the components. This allows for the underlying physics to be retained which enables energy and momentum conservation checks to be completed.

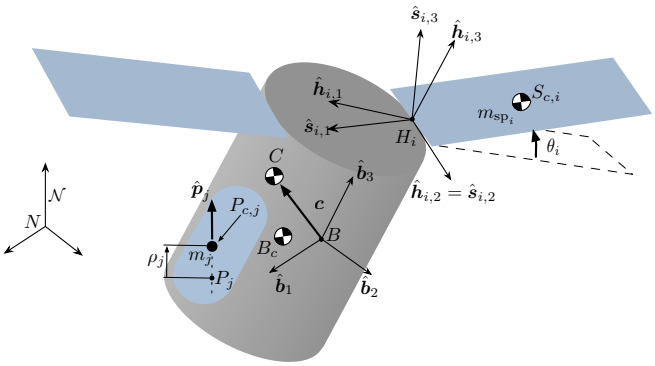

Figure 1. Complex spacecraft with multiple degrees of freedom

SPACECRAFT SPECIFIC COMPACT EQUATIONS OF MOTION FORM

An important consideration when first developing the EOMs are the associated assumptions because they will ultimately dictate how applicable the mathematical structure is to different dynamical systems. Figure 1 shows an example spacecraft with flexing solar arrays and lumped mass fuel slosh and will be the reference when discussing the assumptions.[19] Since both of these types of physical phenomena change the center of mass of the spacecraft, they are good examples for the multi-body spacecraft problems. The common aspect the majority of spacecraft share is that, at least, a small portion can be assumed to be rigid. In Figure 1 the rigid portion of the spacecraft is the gray cylinder. This portion is called the rigid body *hub*. The hub is assumed to have a non-zero mass, m_{hub}, a center of mass location, B_c and an inertia tensor defined about its center of mass, $[I_{\text{hub},B_c}]$.

The most important aspect of the rigid body hub is that it is the object that the body frame, $\mathcal{B} : \{\hat{\boldsymbol{b}}_1, \hat{\boldsymbol{b}}_2, \hat{\boldsymbol{b}}_3\}$, is attached to. To keep the formulation as general as possible, the body frame origin, point B, does not have to be coincident with the hub's center of mass, point B_c. It is very common to make the assumption that these two points are coincident, and makes the derivation of the EOMs simpler,[20,9] however, allowing point B to be located at any location fixed with respect to the rigid hub gives more generality. It is very common in spacecraft missions that a structure frame is defined by the structural engineering team where its origin is not coincident with the rigid body hub's center of mass. Therefore, it gives flexibility in where the body frame origin can be defined. An additional assumption that keeps the formulation as general as possible, is the inertia tensor $[I_{\text{hub},B_c}]$ does not need to be diagonal when defined in body frame components. The formulation would be simpler but less general if a diagonal matrix were used.[20,21,22]

Now that the rigid body hub is defined, the state variables that define the state of the hub at any given time are: the position of point B with respect to point N, $\boldsymbol{r}_{B/N}$, the inertial velocity of point B with respect to point N, $\dot{\boldsymbol{r}}_{B/N}$, the Modified Rodrigues Parameters (MRPs) representation of the body frame $\mathcal{B}$ with respect to the inertial frame $\mathcal{N}$, $\boldsymbol{\sigma}_{\mathcal{B}/\mathcal{N}}$ and the inertial angular velocity vector of the body frame $\mathcal{B}$ with respect to the inertial frame $\mathcal{N}$, $\boldsymbol{\omega}_{\mathcal{B}/\mathcal{N}}$. The MRPs are the chosen attitude parameterization set because its a minimal set of 3 parameters with elegant non-singular implementations.[21] However the dynamics are independent of the chosen attitude parameterization, therefore any attitude description can be used. These 4 variables represent the 6 degrees of freedom that the rigid body hub exhibits and represents the 12 state variables that are needed to implement a

second order differential equation in software. These, at a minimum, are the states required for the system. The additional degrees of freedom on the system will be referenced to the body frame, $\mathcal{B}$.

Now that the important parameters have been defined for the rigid body hub, other degrees of freedom need to be introduced and generalized. Figure 1 shows flexing solar panels and lumped mass fuel slosh as additional degree's of freedom as an example system. Each of those models are labeled as "effectors". Each effector is assumed to have a mass, m_{eff}, a center mass location, E_c, and a position vector from point B to E_c, $\boldsymbol{r}_{E_c/B}$. If the effector has inertia properties, the effector has a frame, $\mathcal{E} : \{\hat{\boldsymbol{e}}_1, \hat{\boldsymbol{e}}_2, \hat{\boldsymbol{e}}_3\}$, and an inertia tensor, $[I_{\text{eff},E_c}]$ that is defined about its center of mass.

With the hub parameters and the effector parameters defined, the general form proposed in this research for the hub's EOMs are shown in Eqs. (1) and (2). This general form was formalized by using the same systematic approach for multiple dynamics problem formulations including flexible solar arrays,[23] lumped mass fuel slosh,[19] imbalanced reaction wheels,[24] fully-coupled mass depletion,[25] and imbalanced variable speed control moment gyroscopes.[26,27] These references explain the derivations in detail and all result in a familiar form. The first equation proposed for this general form is the translational motion equation seen in Eq. (1).

$$m_{\text{sc}}\ddot{\boldsymbol{r}}_{B/N} - m_{\text{sc}}[\tilde{\boldsymbol{c}}]\dot{\boldsymbol{\omega}}_{\mathcal{B}/\mathcal{N}} + \sum_{i=1}^{N_{\text{eff}}}\sum_{j=1}^{N_{\text{DOF},i}} \boldsymbol{v}_{\text{Trans,LHS}_{ij}}\ddot{\alpha}_{ij} = \boldsymbol{F}_{\text{ext}} - 2m_{\text{sc}}[\tilde{\boldsymbol{\omega}}_{\mathcal{B}/\mathcal{N}}]\boldsymbol{c}' - m_{\text{sc}}[\tilde{\boldsymbol{\omega}}_{\mathcal{B}/\mathcal{N}}][\tilde{\boldsymbol{\omega}}_{\mathcal{B}/\mathcal{N}}]\boldsymbol{c} + \sum_{i=1}^{N_{\text{eff}}}\boldsymbol{v}_{\text{Trans,RHS}_i} \quad (1)$$

is in terms of the total mass of the spacecraft, m_{sc}, the vector from point B to the center of mass of the spacecraft, $\boldsymbol{c}$, and the time derivative with respect to the body frame of $\boldsymbol{c}$, $\boldsymbol{c}'$. N_{eff} is the number of effectors, and $N_{\text{DOF},i}$ is the i$^{\text{th}}$ effector's degrees of freedom, $\boldsymbol{v}_{\text{Trans,LHS}_{ij}}$ is a vector for the translational equation that corresponds with the j$^{\text{th}}$ degree of freedom of the i$^{\text{th}}$ effector's second order derivative of its state, α_{ij}, and $\boldsymbol{v}_{\text{Trans,RHS}_i}$ is the i$^{\text{th}}$ effector's vector contribution to the forces on the right hand side of Eq. (1). The rotational EOMs form are shown in Eq. (2).

$$m_{\text{sc}}[\tilde{\boldsymbol{c}}]\ddot{\boldsymbol{r}}_{B/N} + [I_{\text{sc},B}]\dot{\boldsymbol{\omega}}_{\mathcal{B}/\mathcal{N}} + \sum_{i=1}^{N_{\text{eff}}}\sum_{j=1}^{N_{\text{DOF},i}} \boldsymbol{v}_{\text{Rot,LHS}_{ij}}\ddot{\alpha}_{ij} = \boldsymbol{L}_B - [\tilde{\boldsymbol{\omega}}_{\mathcal{B}/\mathcal{N}}][I_{\text{sc},B}]\boldsymbol{\omega}_{\mathcal{B}/\mathcal{N}} - [I'_{\text{sc},B}]\boldsymbol{\omega}_{\mathcal{B}/\mathcal{N}} + \sum_{i=1}^{N_{\text{eff}}}\boldsymbol{v}_{\text{Rot,RHS}_i} \quad (2)$$

$[I_{\text{sc},B}]$ is the inertia tensor of the spacecraft about point B, $[I'_{\text{sc},B}]$ is the time derivative with respect to the body frame of $[I_{\text{sc},B}]$, $\boldsymbol{v}_{\text{Rot,LHS}_{ij}}$ is a vector for the rotational equation that corresponds with the j$^{\text{th}}$ degree of freedom of i$^{\text{th}}$ effector's second order derivative of its state, α_{ij}, and $\boldsymbol{v}_{\text{Rot,RHS}_i}$ is the i$^{\text{th}}$ effector's vector contribution to the torques on the right hand side of Eq. (2).

Finally, the individual effector degree of freedom EOMs are to fit the following form:

$$a_{jj_i}\ddot{\alpha}_{ij} + \sum_{k=1;k\neq j}^{N_{\text{DOF},i}} a_{jk_i}\ddot{\alpha}_{ik} = \boldsymbol{a}_{\alpha_{ij}}^T\ddot{\boldsymbol{r}}_{B/N} + \boldsymbol{b}_{\alpha_{ij}}^T\dot{\boldsymbol{\omega}}_{\mathcal{B}/\mathcal{N}} + c_{\alpha_{ij}} \quad (3)$$

where each effector has $N_{\text{DOF},i}$ EOMs to fully describe the motion of that effector. If $N_{\text{DOF},i} = 1$ for an effector, Eq. (3) simplifies to:

$$\ddot{\alpha}_i = \boldsymbol{a}_{\alpha_i}^T \ddot{\boldsymbol{r}}_{B/N} + \boldsymbol{b}_{\alpha_i}^T \dot{\boldsymbol{\omega}}_{\mathcal{B}/\mathcal{N}} + c_{\alpha_i} \tag{4}$$

Eqs. (1)-(4) are the generalized EOMs that can apply to a wide-variety of spacecraft. Utilizing this common form yields consistent EOMs that enable the modular software architecture. While looking at Eqs. (3) and (4), it should be pointed out that the i$^{\text{th}}$ effector EOM do not include the second order state variables from other effectors, only the individual effectors and their corresponding degree's of freedom. This is a key assumption that will allow for modularity between all of the effectors attached to the rigid body hub.

BACK-SUBSTITUTION METHOD

A product of multi-body dynamics is the dynamic coupling between the second order state variables that results in a non-diagonal system mass matrix.[20] This can be troublesome in attempting to integrate the EOMs in software. When integrating EOMs the form that is beneficial is $\dot{\boldsymbol{X}} = \boldsymbol{f}(\boldsymbol{X}, t)$ where $\boldsymbol{X}$ is the state vector, $\dot{\boldsymbol{X}}$ is the time derivative of $\boldsymbol{X}$, and $\boldsymbol{f}(\boldsymbol{X}, t)$ is a function of the current state and time, t. When there is a system mass matrix, $[M]$, present, the form changes to $[M]\dot{\boldsymbol{X}} = \boldsymbol{f}(\boldsymbol{X}, t)$. Therefore, a system mass matrix needs to be inverted to solve this complex problem. This results in two problems with inverting a system mass matrix. Firstly, inverting a matrix can be computational inefficient because the calculation scales with N^3. Secondly, the modularization of the dynamics from software implementation perspective is a difficult task because the system needs to know the location of each effector within the system mass matrix and locations relative to the other effectors. A back-substitution method is developed to solve this problem.

To visualize the impact of the EOMs generalized form, the spacecraft seen in Figure 2 is used as an example. The spacecraft has panels modeled as as two interconnected rigid bodies with a single rotational degree of freedom each. Figure 2 only shows 2 sets of dual-connected solar panels, but the example is generalized to N number of sets. If the EOMs were put into the generalized form from the previous section, the dynamical coupling of this complex system can be visualized in the following equation:

$$\begin{bmatrix}
3\times3 & 3\times3 & 3\times1 & 3\times1 & 3\times1 & 3\times1 & . & 3\times1 & 3\times1 \\
3\times3 & 3\times3 & 3\times1 & 3\times1 & 3\times1 & 3\times1 & . & 3\times1 & 3\times1 \\
1\times3 & 1\times3 & 1\times1 & 1\times1 & 0 & 0 & . & 0 & 0 \\
1\times3 & 1\times3 & 1\times1 & 1\times1 & 0 & 0 & . & 0 & 0 \\
1\times3 & 1\times3 & 0 & 0 & 1\times1 & 1\times1 & . & 0 & 0 \\
1\times3 & 1\times3 & 0 & 0 & 1\times1 & 1\times1 & . & 0 & 0 \\
. & . & . & . & . & . & . & . & . \\
1\times3 & 1\times3 & 0 & 0 & 0 & 0 & . & 1\times1 & 1\times1 \\
1\times3 & 1\times3 & 0 & 0 & 0 & 0 & . & 1\times1 & 1\times1
\end{bmatrix}
\begin{bmatrix}
\ddot{\boldsymbol{r}}_{B/N} \\
\dot{\boldsymbol{\omega}}_{B/N} \\
\ddot{\theta}_{11} \\
\ddot{\theta}_{12} \\
\ddot{\theta}_{21} \\
\ddot{\theta}_{22} \\
. \\
\ddot{\theta}_{N1} \\
\ddot{\theta}_{N2}
\end{bmatrix}
=
\begin{bmatrix}
3\times1 \\
3\times1 \\
1\times1 \\
1\times1 \\
1\times1 \\
1\times1 \\
. \\
1\times1 \\
1\times1
\end{bmatrix} \tag{5}$$

Eq.(5) shows the form of the second order state variable coupling that results from this configuration. It confirms that the individual degrees of freedom for the sets of solar panels are coupled with each other, but do not directly coupled through second order state derivatives with the other sets of panels. This is a key insight and is exploited in the following back-substitution method.

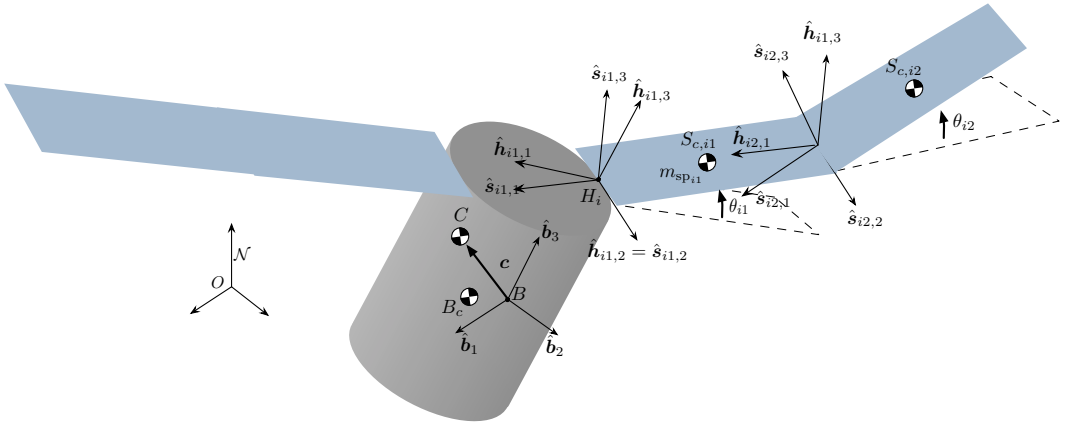

Figure 2. Dual-hinged rigid bodies frame and variable definitions

Looking further into Eq. (5), all of the solar panel second order state derivatives are present in the hub translational and rotational equations. On the other hand, the hub translational and rotational second order state variables are present in the individual solar panel EOMs. This dynamic coupling through the hub is another key insight that the back-substitution method will use to modularize the EOMs.

The back-substitution method is presented for effectors that have $N_{\text{DOF},i} = 1$. Therefore, the equation for each effector's motion can be seen in Eq. (4). The first step in the back substitution method is to substitute Eq. (4) into both the translational and rotational EOMs for the rigid body hub. First, the substitution into the translational motion for $N_{\text{DOF},i} = 1$ is shown in the following equation:

$$m_{\text{sc}}\ddot{\boldsymbol{r}}_{B/N} - m_{\text{sc}}[\tilde{\boldsymbol{c}}]\dot{\boldsymbol{\omega}}_{\mathcal{B}/\mathcal{N}} + \sum_{i=1}^{N_{\text{eff}}} \boldsymbol{v}_{\text{Trans,LHS}_i}\left[\boldsymbol{a}_{\alpha_i}^T\ddot{\boldsymbol{r}}_{B/N} + \boldsymbol{b}_{\alpha_i}^T\dot{\boldsymbol{\omega}}_{\mathcal{B}/\mathcal{N}} + c_{\alpha_i}\right] = \boldsymbol{F}_{\text{ext}}$$
$$- 2m_{\text{sc}}[\tilde{\boldsymbol{\omega}}_{\mathcal{B}/\mathcal{N}}]\boldsymbol{c}' - m_{\text{sc}}[\tilde{\boldsymbol{\omega}}_{\mathcal{B}/\mathcal{N}}][\tilde{\boldsymbol{\omega}}_{\mathcal{B}/\mathcal{N}}]\boldsymbol{c} + \sum_{i=1}^{N_{\text{eff}}} \boldsymbol{v}_{\text{Trans,RHS}_i} \quad (6)$$

Simplifying and combining like terms yields the translational EOM that has been decoupled from the other effector accelerations:

$$\left[m_{\text{sc}}[I_{3\times3}] + \sum_{i=1}^{N_{\text{eff}}} \boldsymbol{v}_{\text{Trans,LHS}_i}\boldsymbol{a}_{\alpha_i}^T\right]\ddot{\boldsymbol{r}}_{B/N} + \left[-m_{\text{sc}}[\tilde{\boldsymbol{c}}] + \sum_{i=1}^{N_{\text{eff}}} \boldsymbol{v}_{\text{Trans,LHS}_i}\boldsymbol{b}_{\alpha_i}^T\right]\dot{\boldsymbol{\omega}}_{\mathcal{B}/\mathcal{N}} = \boldsymbol{F}_{\text{ext}}$$
$$- 2m_{\text{sc}}[\tilde{\boldsymbol{\omega}}_{\mathcal{B}/\mathcal{N}}]\boldsymbol{c}' - m_{\text{sc}}[\tilde{\boldsymbol{\omega}}_{\mathcal{B}/\mathcal{N}}][\tilde{\boldsymbol{\omega}}_{\mathcal{B}/\mathcal{N}}]\boldsymbol{c} + \sum_{i=1}^{N_{\text{eff}}}\left[\boldsymbol{v}_{\text{Trans,RHS}_i} - \boldsymbol{v}_{\text{Trans,LHS}_i}c_{\alpha_i}\right] \quad (7)$$

Following the same pattern for the rotational hub EOM, Eq. (2), yields:

$$\left[m_{\text{sc}}[\tilde{\boldsymbol{c}}] + \sum_{i=1}^{N_{\text{eff}}} \boldsymbol{v}_{\text{Rot,LHS}_i}\boldsymbol{a}_{\alpha_i}^T\right]\ddot{\boldsymbol{r}}_{B/N} + \left[[I_{\text{sc},B}] + \sum_{i=1}^{N_{\text{eff}}} \boldsymbol{v}_{\text{Rot,LHS}_i}\boldsymbol{b}_{\alpha_i}^T\right]\dot{\boldsymbol{\omega}}_{\mathcal{B}/\mathcal{N}} = \boldsymbol{L}_B$$

$$- [\tilde{\boldsymbol{\omega}}_{\mathcal{B}/\mathcal{N}}][I_{\text{sc},B}]\boldsymbol{\omega}_{\mathcal{B}/\mathcal{N}} - [I'_{\text{sc},B}]\boldsymbol{\omega}_{\mathcal{B}/\mathcal{N}} + \sum_{i=1}^{N_{\text{eff}}} \left[\boldsymbol{v}_{\text{Rot,RHS}_i} - \boldsymbol{v}_{\text{Rot,LHS}_i} c_{\alpha_i}\right] \quad (8)$$

The following matrices are defined to yield a more compact notation:

$$[A] = m_{\text{sc}}[I_{3\times 3}] + \sum_{i=1}^{N_{\text{eff}}} \boldsymbol{v}_{\text{Trans,LHS}_i} \boldsymbol{a}_{\alpha_i}^T \quad (9)$$

$$[B] = -m_{\text{sc}}[\tilde{\boldsymbol{c}}] + \sum_{i=1}^{N_{\text{eff}}} \boldsymbol{v}_{\text{Trans,LHS}_i} \boldsymbol{b}_{\alpha_i}^T \quad (10)$$

$$[C] = m_{\text{sc}}[\tilde{\boldsymbol{c}}] + \sum_{i=1}^{N_{\text{eff}}} \boldsymbol{v}_{\text{Rot,LHS}_i} \boldsymbol{a}_{\alpha_i}^T \quad (11)$$

$$[D] = [I_{\text{sc},B}] + \sum_{i=1}^{N_{\text{eff}}} \boldsymbol{v}_{\text{Rot,LHS}_i} \boldsymbol{b}_{\alpha_i}^T \quad (12)$$

$$\boldsymbol{v}_{\text{Trans}} = \boldsymbol{F}_{\text{ext}} - 2m_{\text{sc}}[\tilde{\boldsymbol{\omega}}_{\mathcal{B}/\mathcal{N}}]\boldsymbol{c}' - m_{\text{sc}}[\tilde{\boldsymbol{\omega}}_{\mathcal{B}/\mathcal{N}}][\tilde{\boldsymbol{\omega}}_{\mathcal{B}/\mathcal{N}}]\boldsymbol{c} + \sum_{i=1}^{N_{\text{eff}}} \left[\boldsymbol{v}_{\text{Trans,RHS}_i} - \boldsymbol{v}_{\text{Trans,LHS}_i} c_{\alpha_i}\right] \quad (13)$$

$$\boldsymbol{v}_{\text{Rot}} = \boldsymbol{L}_B - [\tilde{\boldsymbol{\omega}}_{\mathcal{B}/\mathcal{N}}][I_{\text{sc},B}]\boldsymbol{\omega}_{\mathcal{B}/\mathcal{N}} - [I'_{\text{sc},B}]\boldsymbol{\omega}_{\mathcal{B}/\mathcal{N}} + \sum_{i=1}^{N_{\text{eff}}} \left[\boldsymbol{v}_{\text{Rot,RHS}_i} - \boldsymbol{v}_{\text{Rot,LHS}_i} c_{\alpha_i}\right] \quad (14)$$

Using these definitions, the coupled translation and rotation hub EOMs are written compactly as

$$\begin{bmatrix} [A] & [B] \\ [C] & [D] \end{bmatrix} \begin{bmatrix} \ddot{\boldsymbol{r}}_{B/N} \\ \dot{\boldsymbol{\omega}}_{\mathcal{B}/\mathcal{N}} \end{bmatrix} = \begin{bmatrix} \boldsymbol{v}_{\text{Trans}} \\ \boldsymbol{v}_{\text{Rot}} \end{bmatrix} \quad (15)$$

Equation (15) represents a system of 6 equations, that can be solved using the Schur complement matrix formulation[28] for the partitioned form of the hub system mass matrix:

$$\dot{\boldsymbol{\omega}}_{\mathcal{B}/\mathcal{N}} = \Big([D] - [C][A]^{-1}[B]\Big)^{-1} (\boldsymbol{v}_{\text{Rot}} - [C][A]^{-1}\boldsymbol{v}_{\text{Trans}}) \quad (16)$$

$$\ddot{\boldsymbol{r}}_{B/N} = [A]^{-1}(\boldsymbol{v}_{\text{Trans}} - [B]\dot{\boldsymbol{\omega}}_{\mathcal{B}/\mathcal{N}}) \quad (17)$$

This shows that the back-substitution method only requires two 3×3 matrix inverses. The additional degree of freedom state derivatives are found by back-substituting these solutions into Eqs. (3) and (4).

MODULAR SOFTWARE ARCHITECTURE

Figure 3 shows the Unified Modeling Language (UML) class diagram for object oriented computer programming languages proposed in this paper. This is the design that will allow for complex fully-coupled dynamics to be implemented in software while retaining a modular architecture. Additionally, it aims to solve the issues of testability, maintainability, and scalability that fully-coupled dynamics problems pose.

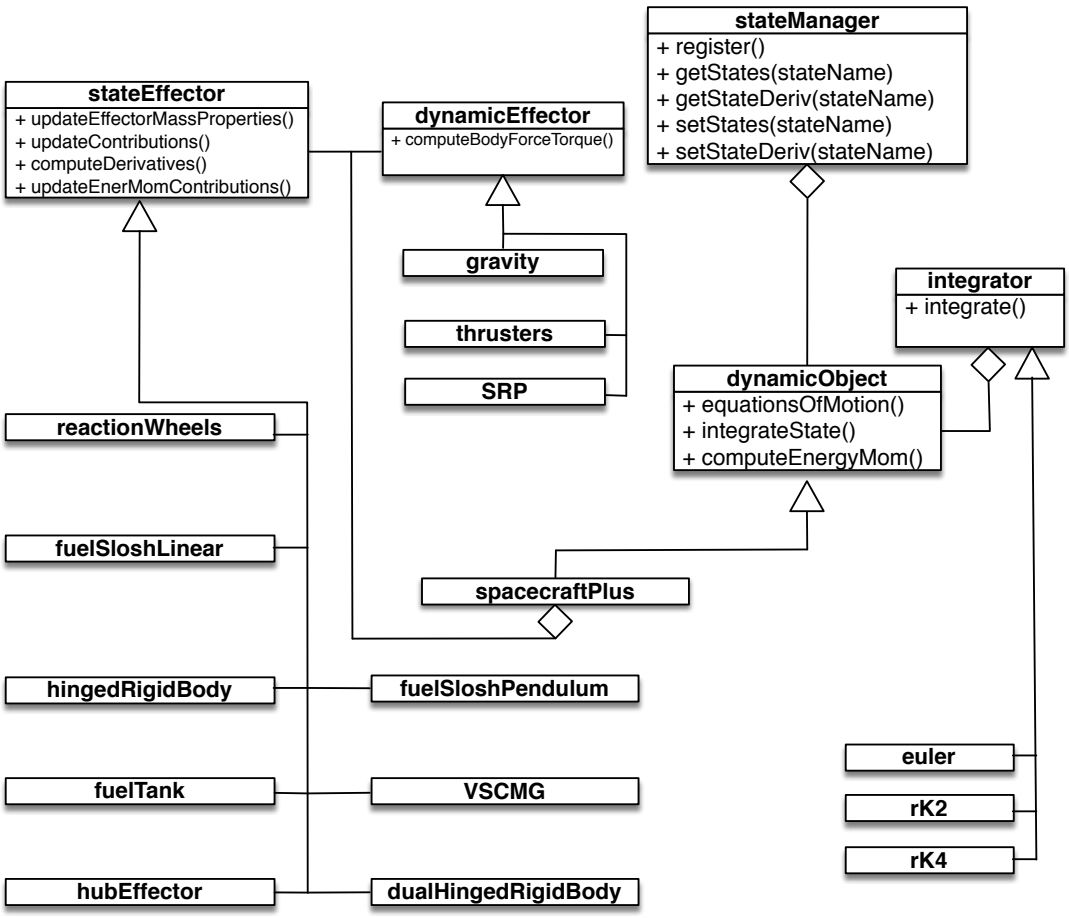

Figure 3. UML Diagram for Modular Architecture.

The *dynamicObject* seen in Figure 3 is a parent class or abstract class (depending on preference) that defines the base functionality of the object that will control the calculation of the system EOMs and essentially solve for the well-known state derivative vector $\dot{\boldsymbol{X}} = \boldsymbol{f}(\boldsymbol{X}, t)$. However, the term state vector is used loosely here because the `stateManager` organizes, stores, and controls all states of the system. The `dynamicObject` is an abstract or parent class because this would allow for different types of systems to be implemented in the future which are not necessarily using the proposed back-substitution method in this paper. Therefore, the `spacecraftPlus` is an instantiation of the `dynamicObject` and is the class that is implementing the back-substitution method.

In the generalized EOMs introduced earlier in this paper, the term "effectors" was used to defined objects that are attached to the spacecraft and have dynamic states that need to be integrated and some examples are: reaction wheels, flexing solar arrays, fuel slosh, etc. In this modular software architecture, those effectors are called `stateEffector`s and are illustrated in Figure 3. In contrast, `dynamicEffector`s are phenomena that result in an external forces or torques being applied to the spacecraft. Examples of these include: gravity, thrusters, SRP, etc.

The `stateEffector` abstract or parent class is the class that defines the necessary methods (and variables) needed for each effector to provide contributions to the spacecrafts mass properties

(m_{sc}, $[I_{\text{sc},B}]$, $\boldsymbol{c}$, etc.) using the method `updateEffectorMassProperties` and contributions to the back substitution matrices ($[A]$, $[B]$...$\boldsymbol{v}_{\text{Trans}}$, etc.) using the method `updateContributions`. Each effector needs to be able to compute their own state derivatives and uses the method `computeDerivatives()`. Finally, the method `computeEnerMomContributions` is the method that enables effectors add their contributions to the energy and momentum of the system for validation purposed. Additionally, it should be noted that in Figure 3 it shows that `stateEffectors` are aggregated in `spacecraftPlus`. This allows for the modularity of the dynamics because `spacecraftPlus` does not know the type of effectors attached to it, but rather has an array of `stateEffectors` which makes it general.

Another important aspect of the software architecture is the `hubEffector` instantiation of `stateEffector`. The `hubEffector` is representing the rigid body hub defined in the generalized EOM form and has translational and attitude states associated with it. The `hubEffector` is unique to all of the `stateEffectors` because it is not included in the array of `stateEffectors` that are looped over in `spacecraftPlus` but rather defined as an object in `spacecraftPlus` and its methods are always called in `equationsOfMotion()`. This is because the assumption for the back-substitution method and the generalized EOM form is that the spacecraft will always have a rigid body hub with a body frame, $\mathcal{B}$, attached and with the corresponding states: $\boldsymbol{r}_{B/N}$, $\dot{\boldsymbol{r}}_{B/N}$, $\boldsymbol{\sigma}_{\mathcal{B}/\mathcal{N}}$ and $\boldsymbol{\omega}_{\mathcal{B}/\mathcal{N}}$.

Since `spacecraftPlus` is an instantiation of `dynamicObject`, it inherits the methods that are defined in Figure 3. The method `equationsOfMotion()` is the method that solves for all of the state derivatives of the spacecraft system. To explain this method in more detail, Figure 4 is included to show the flow in pseudo code. The spacecraft mass properties need to be calculated first because in Eqs. (1) and (2) the total spacecraft mass, m_{sc}, inertia, $[I_{\text{sc},B}]$ and other parameters are needed. Next, the `gravityEffector` class is called to compute the gravity acting on the spacecraft. This is done at this location because some `stateEffectors` might need to know the gravitational force. Following this step, the `stateEffectors` are looped over to find their contributions to the back-substitution matrices. Now, all of the values have been computed for the hub state derivatives to be calculated using Eq. (15), which is computed in the `hubEffector`'s `computeDerivatives`. Finally, the `stateEffectors` are looped over to compute their derivatives using $\ddot{\boldsymbol{r}}_{B/N}$ and $\dot{\boldsymbol{\omega}}_{\mathcal{B}/\mathcal{N}}$.

Since the `hubEffector`'s derivative calculation is so vital in this structure, Figure 5 is shown to explain the calculations needed for this step. Again, this is shown using pseudo code. Additionally, this method shows the interaction between the `stateManager` and the rest of the system. The `stateManager` stores the states of the system in individual objects. These objects can be accessed using a string and once the object has been accessed, the methods seen in Figure 3 under the `stateManager` class are available. For example, the `getState` method delivers the current value of the state stored in that state object. In Figure 5, the hub effector uses those methods to retrieve the desired information from the `stateManager`. Ultimately setting the derivative values for the `hubEffector` is the goal of the `computeDerivatives` method and does so by using `setStateDeriv` for both $\ddot{\boldsymbol{r}}_{B/N}$ and $\dot{\boldsymbol{\omega}}_{\mathcal{B}/\mathcal{N}}$.

Another important method in this architecture is the `computeDerivatives` method for a generic `stateEffector`. To highlight this method, an effector representing a hinged rigid body is used and more details about this formulation can be seen in Reference 23. Figure 6 shows the pseudo code for the `computeDerivatives` method of a hinged rigid body effector. When this method is being computed, $\ddot{\boldsymbol{r}}_{B/N}$ and $\dot{\boldsymbol{\omega}}_{\mathcal{B}/\mathcal{N}}$ have already been calculated, therefore the hinged rigid

spacecraftPlus

```
equationsOfMotion()

    hubEffector.updateEffectorMassProperties()
    for(effector in stateEffectors)
        effector.updateEffectorMassProperties()
    end

    gravityEffector.computeGravField()

    for(effector in stateEffectors)
        effector.updateContributions()
    end

    for(effector in dynEffectors)
        effector.computeBodyForceTorque()
    end

    hubEffector.computeDerivatives()
    for(effector in stateEffectors)
        effector.computeDerivatives()
    end

end
```

Figure 4. Pseudo code for the `equationsOfMotion()` method within `spacecraftPlus`

hubEffector

```
computeDerivatives()

    rBN_NState = stateManager.getStateObject('hubPosition')
    rBNDot_NState = stateManager.getStateObject('hubVelocity')
    sigmaBN_State = stateManager.getStateObject('hubRotPosition')
    omegaBN_BState = stateManager.getStateObject('hubRotVelocity')

    rBNDot_N = rBNDot_NState.getState()
    rBN_NState.setStateDeriv(rBNDot_N)

    sigmaBNDot = omegaToSigmaDot(omegaBN_BState.getState())
    sigmaBN_State.setStateDeriv(sigmaBNDot)

        \dot{\boldsymbol{\omega}}_{\mathcal{B}/\mathcal{N}} = \Big([D] - [C][A]^{-1}[B]\Big)^{-1}(\boldsymbol{v}_2 - [C][A]^{-1}\boldsymbol{v}_1)

                \ddot{\boldsymbol{r}}_{B/N} = [A]^{-1}(\boldsymbol{v}_1 - [B]\dot{\boldsymbol{\omega}}_{\mathcal{B}/\mathcal{N}})

    omegaBN_BState.setStateDeriv(omegaBN_Dot)
    rBNDot_NState.setStateDeriv(rBNDDot_N)

end
```

Figure 5. Pseudo code for `hubEffector` `computeDerivatives()` method

hingedRigidBody

computeDerivatives()

theta_State = stateManager.getStateObject('panelTheta')
thetaDot_State = stateManager.getStateObject('panelThetaDot')
hubVelocity_State = stateManager.getStateObject('hubVelocity')
hubRotVelocity_State = stateManager.getStateObject('hubRotVelocity')

thetaDot = thetaDot_State.getState()
theta_State.setStateDeriv(thetaDot)

rBNDDot_N = hubVelocity_State.getStateDeriv()
omegaDotBN_B = hubRotVelocity_State.getStateDeriv()

$$\ddot{\theta}_i = \boldsymbol{a}_{\theta_i}^T \ddot{\boldsymbol{r}}_{B/N} + \boldsymbol{b}_{\theta_i}^T \dot{\boldsymbol{\omega}}_{\mathcal{B}/N} + c_{\theta_i}$$

thetaDot_State.setStateDeriv(thetaDDot)

end

Figure 6. Pseudo code for `hingedRigidBody` `computeDerivatives()` **method**

body effector can use the state manager's method called `getStateDeriv` which gives access to those pre-computed values. Looking at Eq. (4), the hinged rigid body effector utilizes $\ddot{\boldsymbol{r}}_{B/N}$ and $\dot{\boldsymbol{\omega}}_{\mathcal{B}/\mathcal{N}}$ in its calculation, and utilizes saved variables for faster results.

The power of this design is that `stateEffectors` can just be attached to the spacecraft in no particular order and the scalability of this design is unconstrained. Adding another effector does not depend on any other effectors even though the fully-coupled nature is still retained. All of the coupling is through the rigid body hub and the analytical form of the back-substitution method allows for this modularity. Additionally, a fixed size system mass matrix is inverted as opposed to a variably sized matrix which is common in fully-coupled dynamics simulations.

CONCLUSION

The results of this paper show that a modular software architecture is introduced for fully-coupled dynamics and solves the issue of maintainability, scalability, and testability for this problem. The elegant modularity is achieved by considering a specific spacecraft dynamical system where a range of dynamical sub-systems are attached to a central rigid hub. The proposed software architecture is shown to be relatively simple, allows for a fixed size system mass matrix to be inverted, allows effectors to be attached to the spacecraft in no particular order and does not have scaling limitations. Future work will be to extend this work multiple interconnected and independent spacecraft with effectors attached to them.

REFERENCES

[1] "IEEE Standard Glossary of Software Engineering Terminology," *IEEE Std 610.12-1990*, Dec 1990, pp. 1–84, 10.1109/IEEESTD.1990.101064.

[2] R. Filman, T. Elrad, S. Clarke, and M. Akşit, *Aspect-oriented Software Development*. Addison-Wesley Professional, first ed., 2004.

[3] J. Junkins and Y. Kim, *Introduction to Dynamics and Control of Flexible Structures*. AIAA education series, American Institute of Aeronautics & Astronautics, 1993.

[4] "Bullet: Real-Time Physics Simulation," October, 2017. http://bulletphysics.org/wordpress/.

[5] H. Jeffreys and B. Jeffreys, *Methods of Mathematical Physics*. Cambridge Mathematical Library, Cambridge University Press, 1999.

[6] "CHRONO: An Open Source Framework for the Physics-Based Simulation of Dynamic Systems," October, 2017. http://projectchrono.org.

[7] "RBDL: Rigid Body Dynamics Library," October, 2017. https://rbdl.bitbucket.io/.

[8] "Simscape Multibody - Model and simulate multibody mechanical systems," October, 2017. https://www.mathworks.com/products/simmechanics.html.

[9] T. R. Kane and D. A. Levinson, "Formulation of Equations of Motion for Complex Spacecraft," *Journal of Guidance, Control, and Dynamics*, Vol. 3, 2017/09/27 1980, pp. 99–112, 10.2514/3.55956.

[10] T. R. Kane and D. A. Levinson, "Multibody Dynamics," *Journal of Applied Mechanics*, Vol. 50, 12 1983, pp. 1071–1078.

[11] "Motion Genesis: Symbolic force, motion, and code-generation tools," October, 2017. www.MotionGenesis.com.

[12] "SOLIS: Commercial plug-in to the Analytical Graphics, Inc (AGI) Systems ToolKit (STK)," April, 2017. http://www.go-asi.com/solutions/stk-solis/.

[13] "DARTS: Dynamics Algorithms for Real-Time Simulation," June, 2017. https://dartslab.jpl.nasa.gov/DARTS/index.php.

[14] G. Rodriguez and A. Jain, "Spatial operator algebra for multibody system dynamics," *Journal of the Astronautical Sciences*, 1992.

[15] A. JAIN and G. RODRIGUEZ, "Recursive flexible multibody system dynamics using spatial operators," *Journal of Guidance, Control, and Dynamics*, Vol. 15, 2017/10/26 1992, pp. 1453–1466, 10.2514/3.11409.

[16] "42: A Comprehensive General-Purpose Simulation of Attitude and Trajectory Dynamics and Control of Multiple Spacecraft Composed of Multiple Rigid or Flexible Bodies," September, 2017. https://software.nasa.gov/software/GSC-16720-1.

[17] P. W. Likins, "Point-connected rigid bodies in a topological tree," *Celestial mechanics*, Vol. 11, May 1975, pp. 301–317, 10.1007/BF01228809.

[18] "OreKit: An accurate and efficient core layer for space flight dynamics applications," October, 2017. https://www.orekit.org/.

[19] C. Allard, M. Diaz-Ramos, and H. Schaub, "Spacecraft Dynamics Integrating Hinged Solar Panels and Lumped-Mass Fuel Slosh Model," *AIAA/AAS Astrodynamics Specialist Conference*, Long Beach, CA, Sept. 12–15 2016.

[20] T. R. Kane, P. W. Likins, and D. A. Levinson, *Spacecraft dynamics*. McGraw-Hill Book Co., 1983.

[21] H. Schaub and J. L. Junkins, *Analytical Mechanics of Space Systems*. Reston, VA: AIAA Education Series, 3rd ed., 2014, 10.2514/4.102400.

[22] M. Sidi, *Spacecraft Dynamics and Control: A Practical Engineering Approach*. Cambridge Aerospace Series, Cambridge University Press, 1997, 10.1017/CBO9780511815652.

[23] C. Allard, H. Schaub, and S. Piggott, "General Hinged Solar Panel Dynamics Approximating First-Order Spacecraft Flexing," *AAS Guidance and Control Conference*, Breckenridge, CO, Feb. 5–10 2016. Paper No. AAS-16-156.

[24] J. Alcorn, C. Allard, and H. Schaub, "Fully-Coupled Dynamical Modeling of a Rigid Spacecraft with Imbalanced Reaction Wheels," *AIAA/AAS Astrodynamics Specialist Conference*, Long Beach, CA, Sept. 12–15 2016.

[25] P. Panicucci, C. Allard, and H. Schaub, "Spacecraft Dynamics Employing a General Multi-tank and Multi-thruster Mass Depletion Formulation," *AAS Guidance, Navigation and Control Conference*, Breckenridge, CO, Feb. 2–8 2017. Paper AAS 17-011.

[26] J. Alcorn, C. Allard, and H. Schaub, "Fully-Coupled Dynamical Jitter Modeling Of Variable-Speed Control Moment Gyroscopes," *AAS/AIAA Astrodynamics Specialist Conference*, Stevenson, WA, Aug. 20–24 2017. Paper No. AAS-17-730.

[27] J. Alcorn, "Fully-Coupled Dynamical Jitter Modeling of Momentum Exchange Devices," Master's thesis, University of Colorado - Boulder, Boulder, CO, May 2017.

[28] J. L. Junkins and Y. Kim, *Introduction to Dynamics and Control of Flexible Structures*. Washington D.C.: AIAA Education Series, 1993.