IWSCFF 19-88

SMALL SATELLITE FORMATION FLYING APPLICATION USING THE BASILISK ASTRODYNAMICS SOFTWARE ARCHITECTURE

Simon van Overeem[*] and Hanspeter Schaub[†]

The miniaturization of satellite technology has renewed interest in the use of a large number of equal-type satellites flying in a formation to achieve a single goal. This work aims to integrate a formation flying simulation application into the modular Basilisk Astrodynamics Simulation Software Architecture. This framework uses a message passing interface to communicate states between discrete modules, and a state and dynamics engine to propagate the spacecraft states. Prior work only shows how to use Basilisk to simulate a single satellite. This paper illustrates how the framework can be used to simulate a set of spacecraft. First, a simulation is developed that simulates multiple uncoupled spacecraft in Basilisk. After that, a simulation consisting of two spacecraft using an attitude control algorithm to allow for formation flying is created. The performance of this simulation is evaluated by means of the norm of the attitude error.

INTRODUCTION

Due to their small size and modular nature, small satellites enable multi-satellite missions that are otherwise not possible.[1] These multi-satellite missions require the spacecraft to fly in a formation. Formation flying is defined as a set of two or more spacecraft whose states are coupled through a common control law.[2] In particular, at least one spacecraft of the set must track the desired state relative to another spacecraft and the control law has to depend upon the state of this other spacecraft.[3] These small satellites (of equal-type-and-build) flying in a formation can be used for distributed sensing missions such as signal intelligence missions where multiple satellites can be used for the analysis of radio frequency (RF) signals and localization of RF emitters.[4,5] Furthermore, these small satellites can be used for Earth science missions in order to understand the Earth system and its response to natural and human-induced changes as well as planetary science missions to understand the planets and small bodies that inhabit the solar system.[1]

Major formation flying mission design challenges are centered on formations consisting of a large number of equal-type spacecraft. This type of formation requires a tool that is able to simulate the dynamics and control of spacecraft formation flying in a modular way. In the field of formation flying simulation, tools such as the Formation Algorithm and Control Testbed (FACT),[6] the Control & Analysis Simulation Testbed (CAST),[6] or the Formation Algorithms and Simulation Testbed (FAST),[7] all designed by the Jet Propulsion Laboratory, can be used for simulation of multiple spacecraft operating in a formation. However, these simulation tools only allow the user to simulate a limited number of spacecraft and/or formation types. Both FACT and CAST only allow for the simulation of two spacecraft in a Leader/Follower formation.[6] Furthermore, the FAST testbed is capable to simulate a formation that consists of no more than five spacecraft.[7] Finally, Princeton

[*]Visiting Scholar, Autonomous Vehicle Systems Laboratory

[†]Professor, Glenn L. Murphy Endowed Chair, Smead Aerospace Engineering Sciences Department

Satellite Systems has developed a formation flying simulation module. This tool is able to simulate an arbitrary number of spacecraft and also consists of a Graphical User Interface for a user-friendly environment. However, this tool requires a MATLAB license.[8] For this reason, there is the necessity for an open-source modular formation flying simulation application that allows for the simulation of equal-type-and-build small satellites. The simulation application has to allow for easy expandability of the number of spacecraft such that it is also possible to simulate formations consisting of a large number of spacecraft.

This paper demonstrates that two spacecraft, each with its own characteristics such as spacecraft layout, orbital elements, and flight software can successfully perform formation flying within a single Basilisk simulation instance. The main challenge that entails the integration of a formation flying simulation application into Basilisk are that datastreams between different modules should not be confused. Besides that, the simulation has to make sure that a deputy spacecraft is able to read the state data from the chief spacecraft and adjust its state accordingly. First of all, this paper elaborates on the architecture of Basilisk. Furthermore, the architectural layout of a formation flying simulation is discussed together with the characteristics of this architectural layout. Finally, to validate that the architecture allows for a formation flight simulation, a deputy spacecraft pointing control module is developed. With the knowledge of its relative position with respect to the chief spacecraft, this module allows the deputy spacecraft to point to the chief spacecraft.

METHODOLOGY

This section starts with a brief explanation on the architecture of Basilisk while being used for the simulation of a single spacecraft. After that, the architectural layout for the simulation of multiple spacecraft in a single simulation instance is discussed. Furthermore, an explanation is given on the expandability of the multi-spacecraft architectural layout to allow for the simulation of a large number of spacecraft. Finally, the formation flying control algorithm is discussed.

Basilisk Astrodynamics Software Architecture

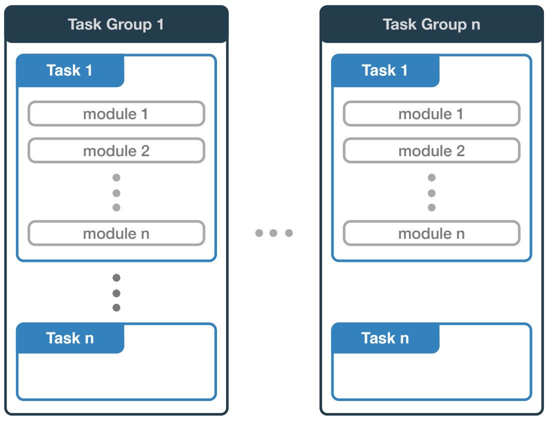
Basilisk is an astrodynamics framework that is able to simulate complex spacecraft systems in the space environment. The type of space missions that Basilisk can be used for ranges from Earth-orbiting CubeSats to interplanetary probes. Basilisk is unique in its capability to simulate coupled attitude and orbital motion to high fidelity with high computational efficiency using a C++ based dynamics engine, its use of message passing interfaces to provide modularity, and its embrace of the open source community.[9] Besides that, Flight Software modules are written in C to allow these modules to be easily ported directly to flight targets. Basilisk users interact and develop scenarios using Python. The Python bindings are auto-generated at build time using the Software Interface Generator (SWIG) tool.[10,11]

The Basilisk framework consists of three components: modules, tasks, and task groups. A module in Basilisk is stand-alone code that typically implements a specific model (e.g. an actuator, sensor, or dynamics model) or self-contained logic (e.g. translating a control torque to a reaction wheel command voltage). Modules publish output data and receive input data using the publish-subscribe message passing interface. Modules can receive data as a message by subscribing to available messages in the Messaging System. Furthermore, modules publish output data as a message to the Messaging System.[10] To elaborate a little more on this Messaging System an example is given. Consider a module that outputs a control torque. This control torque data is required by a module that translates the control torque to a reaction wheel command voltage. In this case, a data flow is made possible by the control torque module publishing the data to the Messaging System with a specific message name. After this is done the command voltage module looks for the same message name in the Messaging System and uses the linked data as an input.

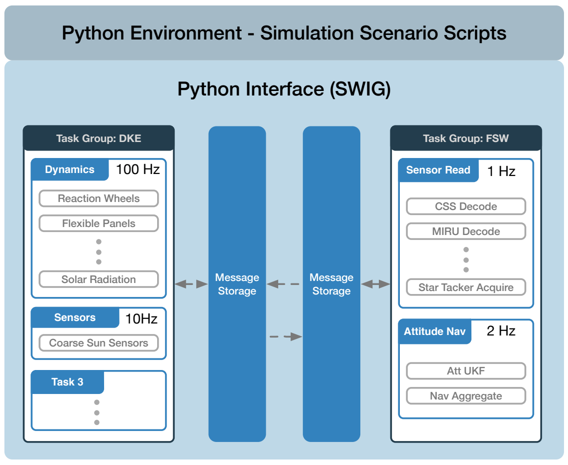
Tasks are groupings of modules. Each task has an integration rate set by the user which directs the update rate of all modules assigned to that task. This results in a framework that allows grouping modules with different integration rates according to the desired fidelity.[10]

Task groups are the highest level grouping in Basilisk. A task group acts as a silo for tasks and provides the mechanism for resolving message dependencies between modules (which is done by making use of the Messaging System).[10]

Figure 1a shows the general layout of a Basilisk simulation. As can be observed in the figure, one or multiple task groups can be created each consisting of one or multiple tasks. Each task consequently consists of one or more modules (each module corresponding to the specific update rate of the task).



(a) Task groups, tasks and modules[10]



(b) Example of Basilisk simulation[10]

Figure 1: Layout of Basilisk architecture

Figure 1b shows an example of a Basilisk simulation. These simulations generally consist of two task groups in order to make a clear distinction between the onboard Flight Software modules and the specific Dynamics modules. One task group is used to store modules such as actuators, sensors, and dynamics models. The name of this task group is the Dynamics task group (task group: DKE). A second task group is used to store Flight Software algorithms. The name of this task group is the Flight Software task group (task group: FSW). The messaging system makes sure that the published output data of each module is used as an input by the correct module. However, in order to simulate a spacecraft formation flying mission, it is necessary to design an architecture that allows multiple spacecraft to be set up in Basilisk.

Multiple Spacecraft Simulation Setup

Basilisk Layout Options for Simulating Multiple Spacecraft In order to allow Basilisk to perform simulations that involve formation flying, it is necessary to set up multiple spacecraft along with their associated messages. There are different architectural layouts for the Basilisk framework that allow the simulation of multiple spacecraft in a single simulation instance. The options considered for the multi-spacecraft simulation are listed below.

Architectural Layout 1: This layout can be observed in Figure 2. This figure shows one task group consisting of Dynamics modules and a second task group consisting of Flight Software modules. Resulting in two task groups consisting of the modules for all spacecraft in the formation. Making use of this architectural layout results in a clear separation between Dynamics modules and Flight Software modules which gives the user a clear overview. This separation is especially beneficial in case simulations are performed that include hardware-in-the-loop as making a clear separation between Dynamics- and Flight Software modules makes it possible to efficiently load the flight software on the onboard hardware while running the Dynamics modules remotely on a desktop. However, in case of the simulation of large amounts of spacecraft this architecture can become somewhat disorganized.

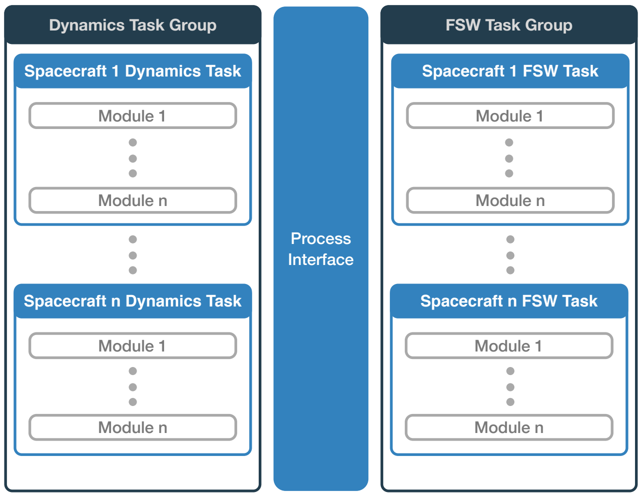


Figure 2: Architectural layout 1

Architectural Layout 2: This layout can be observed in Figure 3. This figure shows one task group consisting of the Dynamics modules and the Flight Software modules for one single spacecraft. Resulting in a task group for each spacecraft that needs to be simulated. A benefit of this architectural layout is that the structure is very organized as each task group represents a spacecraft. This helps the user to efficiently arrange the Dynamics- and Flight Software tasks in the simulation. However, this architecture shows one obvious disadvantage. In case, for instance, two spacecraft need to be simulated in a formation (two task groups) and the flight software of the first spacecraft has to read state data from the second spacecraft a problem occurs. Because the dynamics states of the second spacecraft are updated after the flight software of the first spacecraft is updated, this flight software reads state data from a previous timestep which is undesirable.

Architectural Layout 3: This layout can be observed in Figure 4. This figure shows that one task group consists of the Dynamics modules for a single spacecraft in the formation and that another task group consists of Flight Software modules for a single spacecraft in the formation. This results in a total of 2n task groups for n spacecraft. This architectural layout provides a clear separation between the Flight Software tasks and Dynamics tasks of each spacecraft without becoming disorganized. For this reason, this architecture also makes it possible to efficiently load the flight software on the onboard hardware. This architecture has the disadvantage that it requires very strict bookkeeping.

As it is undesirable to update the Flight Software before the dynamics states are updated, it is important to make sure that each Dynamics task group is updated before the Flight Software task groups are updated. This requires the user to continuously keep track of the update order of the task groups.

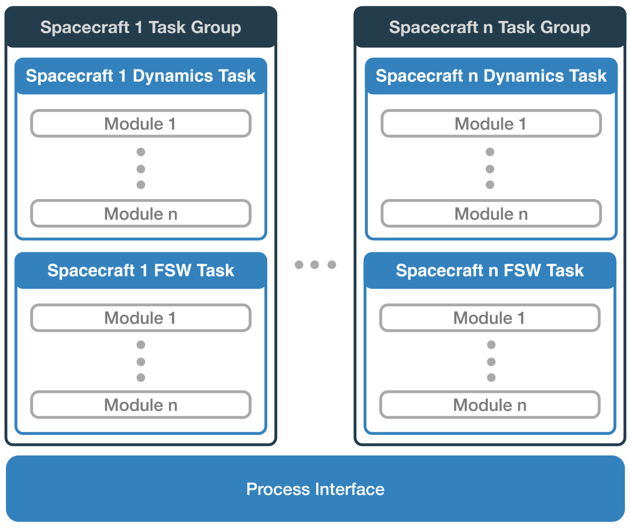
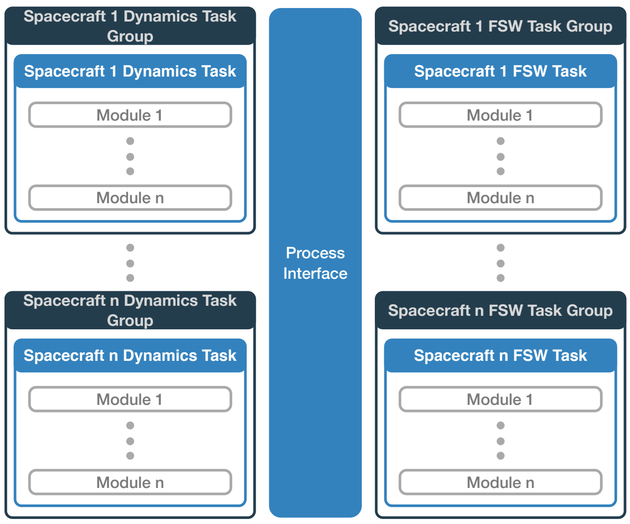


Figure 3: Architectural layout 2



Spacecraft 1 Dynamics Task Group
Spacecraft 1 Dynamics Task
Module 1
Module n
Spacecraft n Dynamics Task Group
Spacecraft n Dynamics Task
Module 1
Module n
Process Interface
Spacecraft 1 FSW Task Group
Spacecraft 1 FSW Task
Module 1
Module n
Spacecraft n FSW Task Group
Spacecraft n FSW Task
Module 1
Module n

Figure 4: Architectural layout 3

Mainly due to the fact that Architectural Layout 2 can lead to data reading timing issues, it seems to be undesirable to perform formation flight simulations using this architecture. As a matter of fact, both Architectural Layout 1 and 3 seem to be reasonable options to perform formation flight simulations. However, the architectural layout of Basilisk for the simulation of a single spacecraft is most comparable to Architectural Layout 1. Because this paper shows the first progress in formation flying simulations using Basilisk it makes sense to hold on to this existing architecture as much as possible. For this reason, it is decided to proceed with layout 1 to create a working formation flight simulation for this paper.

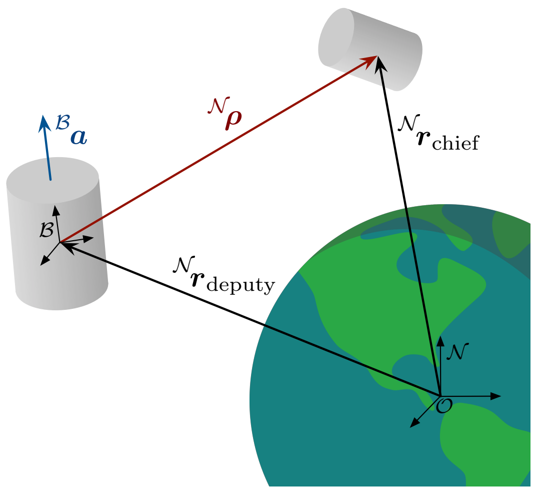
Messaging System Modules can receive data as a message by subscribing to available messages in the Messaging System. The output data of each module in this Messaging System has a specific message name attached to it. A module that requires input data consequently subscribes to the corresponding message name and uses the attached data as an input. In Basilisk each module has a default input- and output message name for the data. In case multiple spacecraft are simulated at the same time it is unavoidable to duplicate certain modules. This results in a situation where two modules (of two different spacecraft) write a message to the Messaging System with the same message name. After that two modules (again of two different spacecraft) subscribe to these messages to use the data as an input. However, because both message names are identical it is unknown which data belongs to which spacecraft. In order to make sure that modules receive the correct input data it is necessary to make them subscribe to the correct message names. For this reason, the multiple spacecraft simulation architecture requires the user to manually overwrite the default message names of the input and output data.

Gravity Body Setup Most Basilisk simulations make use of a built-in module that uses NASA's SPICE kernels for time-varying planet ephemeris messages.[12] However, in case SPICE is used to set up a celestial body for a simulation that consists of multiple spacecraft with the exact same initial conditions an offset exists between the inertial position of the spacecraft and the expected (and correct) inertial position after the simulation is done. This problem is not solved yet, however, it is assumed that the SPICE module makes use of different ephemeris data for solving the equations of motion of each spacecraft. For this reason, the gravity body factory class called gravFactory is used to add gravity bodies to the simulation. The difference between this module and the built-in SPICE kernel module is that the gravFactory module does not take into account that the ephemeris of the planets is time-varying, while SPICE does take this into account. Due to the implementation of the gravFactory module, this problem is solved.

Expandability Due to the modularity of Basilisk and the simplicity of the Basilisk multiple spacecraft simulation architecture, it also allows for easy expansion of the number of spacecraft that needs to be simulated. Making use of Architectural Layout 1, a spacecraft in Basilisk is essentially defined as a set of Dynamics- and Flight Software modules. These modules are sorted in tasks and these tasks are added to task groups. In order to add a spacecraft to the simulation that is used to simulate spacecraft of equal-type-and-build, all the user has to do is duplicate the tasks in both the Dynamics task group as well as the Flight Software task group. Finally, it is necessary to manually overwrite the message names of required input data and published output data in order to prevent modules reading arbitrary data (as is discussed in the section about the Messaging System for multiple spacecraft). This results in the possibility to simulate an arbitrary number of equal-type-and-build spacecraft with each spacecraft having its own characteristics, orbital elements, and flight software.

Spacecraft Pointing Flight Software Module

Formation flying is defined as a set of two or more spacecraft whose states are coupled through a common control law. In order to make two spacecraft fly in a formation, it is necessary for one spacecraft to read state data from another spacecraft and use this state data to control the spacecraft. In order to demonstrate that a spacecraft in Basilisk has the capability to read data from another spacecraft and use this data for control purposes a relative pointing control module is created. This module is intended for controlled formation flying applications where it is necessary to set up at least a chief spacecraft and a deputy spacecraft. The notations and derivations used for the design of this module are taken from the book Analytical Mechanics of Space Systems.[13] The goal of this module is to align a vector defined by the user in the deputy body frame (${}^{\mathcal{B}}\boldsymbol{a}$) with a vector that points from the deputy- to the chief spacecraft (${}^{\mathcal{N}}\boldsymbol{\rho}$). Both vectors are displayed in Figure 5, where the origin of the inertial frame ($\mathcal{N}$) is set to the center of the Earth. The ${}^{\mathcal{B}}\boldsymbol{a}$ vector defined by the user can, for instance, be the location of an antenna or a camera within the body frame. Possible applications of the pointing of these objects to a different spacecraft lie in the field of communication as well as vision-based relative navigation.[14] Besides the ${}^{\mathcal{B}}\boldsymbol{a}$ vector, the module uses the location of the chief and deputy spacecraft in the inertial reference frame as an input. The outputs of this module consist of the orientation, the angular velocity, and the angular acceleration of a reference frame with respect to the inertial reference frame. This reference frame is discussed in more detail later in this section. The outputs of this module, as well as the spacecraft's orientation with respect to the inertial frame, are fed into the attitude tracking error module which calculates the error between the reference frame and the spacecraft body frame to make sure that the deputy spacecraft body frame eventually aligns with this reference frame. The alignment of the deputy spacecraft body frame with the reference frame has to result in an alignment of the ${}^{\mathcal{B}}\boldsymbol{a}$ vector with the ${}^{\mathcal{N}}\boldsymbol{\rho}$ vector.



${}^{\mathcal{B}}\boldsymbol{a}$
${}^{\mathcal{N}}\boldsymbol{\rho}$
${}^{\mathcal{N}}\boldsymbol{r}_{\text{chief}}$
$\mathcal{B}$
${}^{\mathcal{N}}\boldsymbol{r}_{\text{deputy}}$
$\mathcal{N}$
$\mathcal{O}$

Figure 5: Vector illustrations used for spacecraft pointing Flight Software module.

As is discussed earlier the inputs of this module are the position of the chief and the deputy spacecraft with respect to the inertial reference frame (${}^{\mathcal{N}}\boldsymbol{r}_{\text{chief}}$ and ${}^{\mathcal{N}}\boldsymbol{r}_{\text{deputy}}$ respectively). In order to find the unit vector that points from the deputy to the chief Eq. (1) can be used.

$$ {}^{\mathcal{N}}\hat{\boldsymbol{\rho}} = \frac{{}^{\mathcal{N}}\boldsymbol{r}_{\text{chief}} - {}^{\mathcal{N}}\boldsymbol{r}_{\text{deputy}}}{||{}^{\mathcal{N}}\boldsymbol{r}_{\text{chief}} - {}^{\mathcal{N}}\boldsymbol{r}_{\text{deputy}}||} \tag{1} $$

Consequently, a coordinate system is built around the ${}^{\mathcal{N}}\hat{\boldsymbol{\rho}}$ vector as can be seen in Eq. (2). In this equation, ${}^{\mathcal{N}}\hat{\boldsymbol{x}}$, ${}^{\mathcal{N}}\hat{\boldsymbol{y}}$, and ${}^{\mathcal{N}}\hat{\boldsymbol{z}}$ are the normalized x-, y-, and z-components of the $\mathcal{R}$ frame written in $\mathcal{N}$ frame components. Furthermore, ${}^{\mathcal{N}}\hat{\boldsymbol{z}}_{\text{inertial}}$ represents the z-component in the $\mathcal{N}$ frame ($[0, 0, 1]^T$). In case ${}^{\mathcal{N}}\hat{\boldsymbol{\rho}}$ aligns with ${}^{\mathcal{N}}\hat{\boldsymbol{z}}_{\text{inertial}}$ different cross products are taken to avoid undefined axes. The entries in the direction cosine matrix can be observed in Eq. (3). This direction cosine matrix can consequently be converted to the Modified Rodrigues Parameter (MRP) vector $\boldsymbol{\sigma}_{RN}$. This MRP vector thus describes the orientation of the $\mathcal{R}$ frame that is built around the vector that points from the deputy to the chief (${}^{\mathcal{N}}\hat{\boldsymbol{\rho}}$) with respect to the $\mathcal{N}$ frame.

$$\begin{aligned} {}^{\mathcal{N}}\hat{\boldsymbol{x}} &= {}^{\mathcal{N}}\hat{\boldsymbol{\rho}} \\ {}^{\mathcal{N}}\hat{\boldsymbol{y}} &= {}^{\mathcal{N}}\hat{\boldsymbol{z}}_{\text{inertial}} \times {}^{\mathcal{N}}\hat{\boldsymbol{\rho}} \\ {}^{\mathcal{N}}\hat{\boldsymbol{z}} &= {}^{\mathcal{N}}\hat{\boldsymbol{x}} \times {}^{\mathcal{N}}\hat{\boldsymbol{y}} \end{aligned} \tag{2}$$

$$[RN] = \begin{bmatrix} {}^{\mathcal{N}}\hat{\boldsymbol{x}}^T \\ {}^{\mathcal{N}}\hat{\boldsymbol{y}}^T \\ {}^{\mathcal{N}}\hat{\boldsymbol{z}}^T \end{bmatrix} \tag{3}$$

After that, it is possible to calculate the angular velocity of the $\mathcal{R}$ frame with respect to the $\mathcal{N}$ frame in $\mathcal{N}$ frame components. In order to find the change in $\boldsymbol{\sigma}_{RN}$ a numerical approximation is used. To find the change in $\boldsymbol{\sigma}_{RN}$ over one single timestep, $\boldsymbol{\sigma}_{RN}(t_{k-1})$ is subtracted from $\boldsymbol{\sigma}_{RN}(t_k)$ and divided by the timestep (Δt) as can be seen in Eq. (4).

$$\dot{\boldsymbol{\sigma}}_{RN} = \frac{\boldsymbol{\sigma}_{RN}(t_k) - \boldsymbol{\sigma}_{RN}(t_{k-1})}{\Delta \text{t}} \tag{4}$$

After that, the angular velocity of the $\mathcal{R}$ frame with respect to the $\mathcal{N}$ frame in $\mathcal{R}$ frame components can be computed using Eq. (5) (equation 3.164 in[13]). All σ components (also the vector ones) used in this equation are the average between σ at t_{k-1} and σ at t_k. This is done due to the fact that using the σ components at solely time t_k results in incorrect values for ω. The flaw was discovered due to the fact that for a circular relative motion of the deputy spacecraft with respect to the chief spacecraft with a known angular rate it turns out that the results from the numerical method fluctuate around the true angular velocity. Taking the average of σ results in an angular velocity that is in coherence with values calculated analytically for circular as well as elliptical relative orbits.

$${}^{\mathcal{R}}\boldsymbol{\omega}_{RN} = \frac{4}{(1+\sigma^2)^2}[(1-\sigma^2)[I_{3\times3}] - 2[\tilde{\boldsymbol{\sigma}}] + 2\boldsymbol{\sigma}\boldsymbol{\sigma}^T]\dot{\boldsymbol{\sigma}}_{RN} \tag{5}$$

Using a direction cosine matrix ${}^{\mathcal{N}}\boldsymbol{\omega}_{RN}$ is calculated from ${}^{\mathcal{R}}\boldsymbol{\omega}_{RN}$. Finally, the angular acceleration (${}^{\mathcal{N}}\dot{\boldsymbol{\omega}}_{RN}$) can be calculated using Eq. (6).

$${}^{\mathcal{N}}\dot{\boldsymbol{\omega}}_{RN} = \frac{{}^{\mathcal{N}}\boldsymbol{\omega}_{RN}(t_k) - {}^{\mathcal{N}}\boldsymbol{\omega}_{RN}(t_{k-1})}{\Delta \text{t}} \tag{6}$$

The attitude tracking error module calculates the error of the deputy spacecraft's attitude with respect to the attitude of a reference frame. For this reason, using the $\mathcal{R}$ frame as a reference frame would result in an alignment of the x-axis of the deputy spacecraft's body frame with the ${}^{\mathcal{N}}\hat{\boldsymbol{\rho}}$ vector instead of an alignment of the vector specified by the user (${}^{\mathcal{B}}\boldsymbol{a}$) with the ${}^{\mathcal{N}}\hat{\boldsymbol{\rho}}$ vector. For this reason, an $\mathcal{A}$ frame is built around the ${}^{\mathcal{B}}\boldsymbol{a}$ vector. The vector components of this coordinate system can be seen in Eq. (7). In this equation, ${}^{\mathcal{B}}\hat{\boldsymbol{z}}_{\text{deputy}}$ represents the z-component of the deputy spacecraft body frame. Also for this coordinate system yields that different cross products are taken in case the ${}^{\mathcal{B}}\boldsymbol{a}$ vector aligns with the z-component of the deputy spacecraft body frame (${}^{\mathcal{B}}\hat{\boldsymbol{z}}_{\text{deputy}}$) in order to avoid undefined axes.

$$\begin{aligned} {}^{\mathcal{B}}\hat{\boldsymbol{x}} &= {}^{\mathcal{B}}\hat{\boldsymbol{a}} \\ {}^{\mathcal{B}}\hat{\boldsymbol{y}} &= {}^{\mathcal{B}}\hat{\boldsymbol{z}}_{\text{deputy}} \times {}^{\mathcal{B}}\hat{\boldsymbol{a}} \\ {}^{\mathcal{B}}\hat{\boldsymbol{z}} &= {}^{\mathcal{B}}\hat{\boldsymbol{x}} \times {}^{\mathcal{B}}\hat{\boldsymbol{y}} \end{aligned} \tag{7}$$

The direction cosine matrix that consequently maps the $\mathcal{B}$ frame to the $\mathcal{A}$ frame can be found in Eq. (8) and is converted to the MRP vector $\boldsymbol{\sigma}_{AB}$.

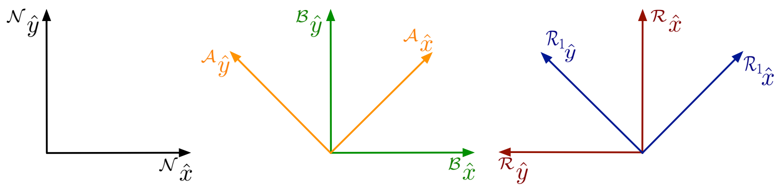
$$[AB] = \begin{bmatrix} {}^{\mathcal{B}}\hat{\boldsymbol{x}}^T \\ {}^{\mathcal{B}}\hat{\boldsymbol{y}}^T \\ {}^{\mathcal{B}}\hat{\boldsymbol{z}}^T \end{bmatrix} \tag{8}$$

Thus, up until now, two coordinate frames have been built around a specific vector. The $\mathcal{R}$ frame around the ${}^{\mathcal{N}}\hat{\boldsymbol{\rho}}$ vector and the $\mathcal{A}$ frame around the ${}^{\mathcal{B}}\boldsymbol{a}$ vector. So, in order to make sure that the ${}^{\mathcal{B}}\boldsymbol{a}$ vector aligns with the ${}^{\mathcal{N}}\hat{\boldsymbol{\rho}}$ vector, the $\mathcal{A}$ frame needs to align with the $\mathcal{R}$ frame. However, one of the inputs of the attitude tracking error module is the deputy spacecraft body frame ($\mathcal{B}$). For this reason the following computations are performed and illustrated using a 2D example. Note that this example is given in 2D in order to improve understandability. The calculations performed in the simulation are done in 3D and are equal to the computations shown in the example (solely an extra dimension is added to the vectors).

Looking at Eq. (6) the orientation of the $\mathcal{N}$ frame can be observed. Besides that, it is possible to see the $\mathcal{B}$ frame and the $\mathcal{A}$ frame relative to the $\mathcal{B}$ frame. Furthermore, Figure 6 shows the orientation of the $\mathcal{R}$ frame and the $\mathcal{R}_1$ frame (noting that $\boldsymbol{\sigma}_{AB} = \boldsymbol{\sigma}_{RR_1}$ by definition). Up until this point $\boldsymbol{\sigma}_{RN}$ and $\boldsymbol{\sigma}_{AB}$ are calculated. The goal is to align the $\mathcal{A}$ frame with the $\mathcal{R}$ frame. This is the same as aligning the $\mathcal{B}$ frame with the $\mathcal{R}_1$ frame (because $\boldsymbol{\sigma}_{AB} = \boldsymbol{\sigma}_{RR_1}$). For this reason, the reference MRP that will be used as an output is $\boldsymbol{\sigma}_{R_1N}$ and can be calculated using Eq. (9) (because $\boldsymbol{\sigma}_{BA} = -\boldsymbol{\sigma}_{AB}$).[13]

$$\boldsymbol{\sigma}_{R_1N} = \boldsymbol{\sigma}_{RN} + \boldsymbol{\sigma}_{BA} \tag{9}$$

The angular velocity and angular acceleration stay the same because the $\mathcal{A}$ frame is stationary with respect to the $\mathcal{B}$ frame.



${}^{\mathcal{N}}\hat{y}$
${}^{\mathcal{N}}\hat{x}$
${}^{\mathcal{A}}\hat{y}$
${}^{\mathcal{B}}\hat{y}$
${}^{\mathcal{A}}\hat{x}$
${}^{\mathcal{B}}\hat{x}$
${}^{\mathcal{R}_1}\hat{y}$
${}^{\mathcal{R}}\hat{x}$
${}^{\mathcal{R}_1}\hat{x}$
${}^{\mathcal{R}}\hat{y}$

Figure 6: Aligning coordinate systems example.

RESULTS

First, the results of a multiple spacecraft scenario are presented. These results consist of the exact architecture of the multi-spacecraft simulation as well as plots that show the orbits and angular velocity of both spacecraft. After that, the results of the spacecraft pointing flight software module are discussed.

Multiple Spacecraft Simulation Scenario

In order to verify the multiple spacecraft architecture in Basilisk, a scenario is created that sets up two uncoupled 6 degree of freedom spacecraft orbiting the Earth. This scenario shows future users how the setup of a multiple spacecraft simulation scenario is different from a single spacecraft simulation scenario and proves that it is possible to set up several spacecraft each using their own flight software algorithms. The main difference lies in the architecture of the simulation as can be observed in Figure 7. This figure shows the architectural layout of the two Earth-orbiting spacecraft scenario. It can be observed that according to the design decisions made two task groups are present, a Dynamics task group and a Flight Software task group. Furthermore, the Dynamics task group consists of a Chief Spacecraft task and a Deputy Spacecraft task respectively to store the modules. This scenario consists of two spacecraft (spacecraftObject) for which e.g. the location of the center of mass, the mass, and inertia tensor can be defined independently. Furthermore, the scenario includes a device that is able to determine the position, velocity, attitude, and attitude rate of the spacecraft (simpleNavObject), and a set of reaction wheels (rwStateEffector). Finally, the Earth is added to the scenario as a gravity body using the gravFactory module and affects both the chief- and deputy spacecraft. Furthermore, the Flight Software task group consists of four tasks, two tasks for the chief spacecraft and two tasks for the deputy spacecraft. The two tasks for each spacecraft form the flight software algorithm and the reason for dividing the modules over two tasks lies in the difference in integration rate for each module.

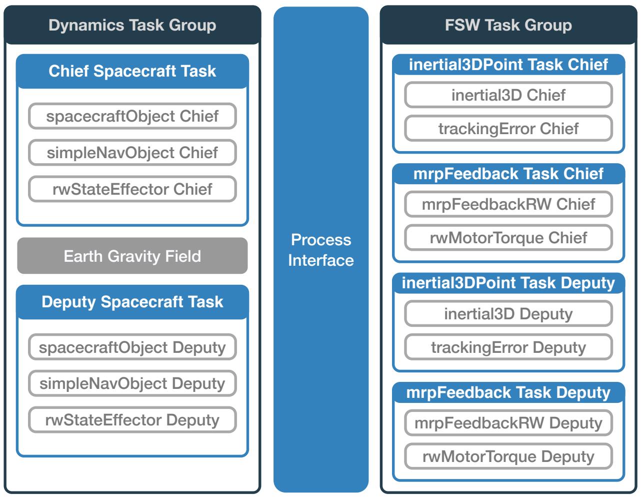


Figure 7: Basilisk multiple spacecraft simulation architecture

Figure 8 presents the simulation data flow diagram of the scenario. In this scenario, two spacecraft (each with a uniquely defined initial position, velocity, attitude, and attitude rate) have to align their body coordinate frame with a reference coordinate frame defined by the user (also uniquely defined for each spacecraft). The orientations of these reference frames are defined by the user through the inertial3D modules. After that, the trackingError modules receive the orientation and angular velocity of the spacecraft body frames as well as the reference frame orientation and calculates the error. This error is consequently converted to a required control torque using the mrpFeedbackRW module. This data is fed into the rwMotorTorque module to convert the control torque to the required reaction wheel speed. This speed is finally fed to the reaction wheels of each spacecraft (rwStateEffector). The change in angular velocity of the reaction wheels eventually has to make sure that both spacecraft come to rest and align with the user-defined coordinate frames.

Figure 9a shows the orbits of two spacecraft orbiting the Earth. In this figure, the dotted lines represent the shape of a full orbit, while the red lines show the propagation of a spacecraft over the simulation period. It can be observed that the orbital elements of both spacecraft are different.

Furthermore, in Figure 9b and Figure 9c it is possible to see the change in angular velocity for both spacecraft. It can be observed that both spacecraft have a unique set of curves due to the different initial attitude and attitude rate.

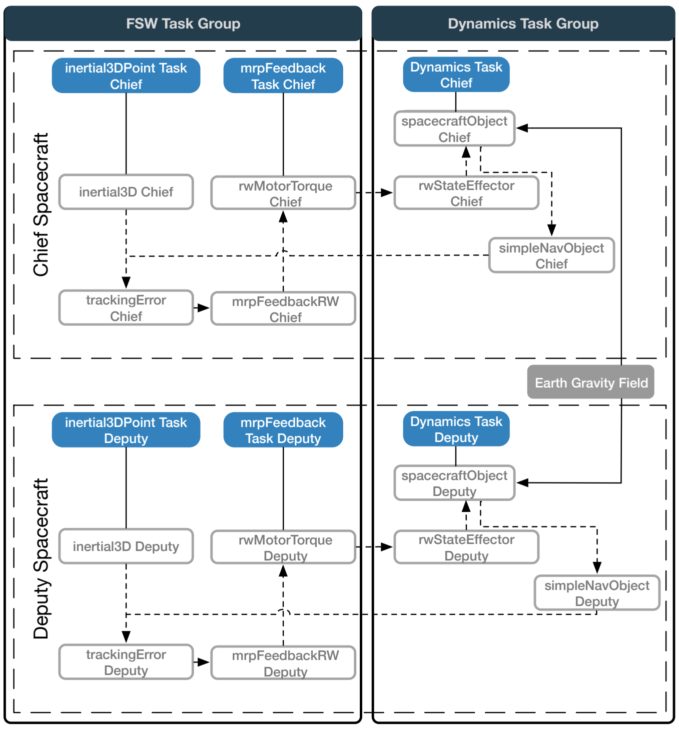
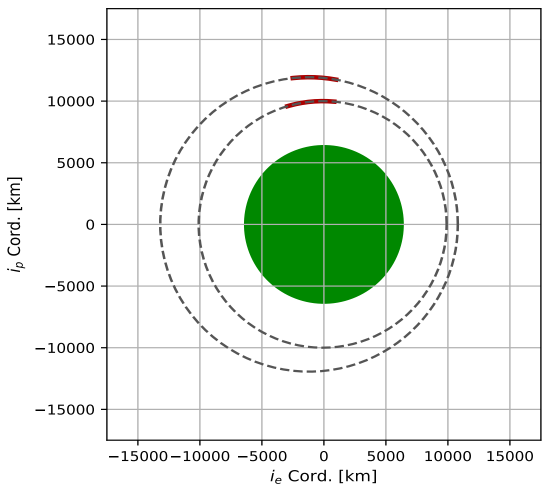
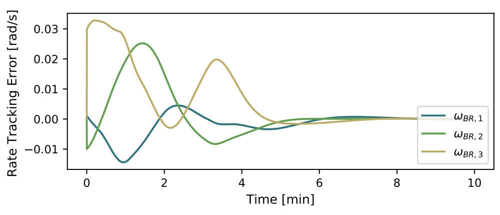


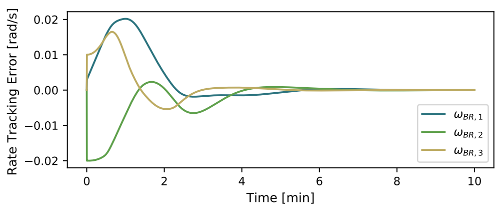
Figure 8: Simulation data flow diagram for multiple spacecraft simulation scenario



(a) Orbits of two spacecraft in Basilisk



(b) Chief spacecraft angular velocity components



(c) Deputy spacecraft angular velocity components

Figure 9: Multiple spacecraft simulation results

Spacecraft Pointing Flight Software module

This subsection describes the results of the Flight Software module developed to allow a deputy spacecraft to point to the chief spacecraft. To test this control module, a scenario is created in which two spacecraft fly in a Leader/Follower formation in an elliptical Earth-orbit. This subsection starts with the simulation data flow diagram, which shows how the modules are connected to each other. Furthermore, the pointing error is discussed in order to evaluate on the accuracy of the module.

Simulation Data Flow Diagram The simulation data flow diagram of the spacecraft pointing flight software module can be observed in Figure 10. This simulation data flow diagram shows the Flight Software task group on the left side and the Dynamics task group on the right side. Besides that, the tasks added to all task groups and the corresponding modules can be observed. For the chief spacecraft, there are no flight software tasks added to the Flight Software task group. However, there is a task added to the Dynamics task group for the chief spacecraft. This Dynamics task consists of the chief spacecraft (spacecraftObject chief) and the simpleNavObject which is able to determine the position, velocity, attitude, and attitude rate of the spacecraft. For the deputy spacecraft, two tasks are added to the Flight Software task group. The Spacecraft Pointing task consists of the spacecraftPointing module that is discussed previously. It can be observed that this module receives input data from both the chief- and deputy spacecraft simpleNavObject. The attitude, attitude rate, and angular acceleration of the reference frame as well as of the deputy body frame with respect to the inertial frame are used as an input for the TrackingError module of the deputy. The relative error of these two frames is consequently used as an input for the deputy mrpFeedbackControl module (which is a part of the MRP Feedback task). This module calculates the required control torque which is used as an input for the externalForceTorque module. This module can be seen as an invisible hand that is able to apply torques to the spacecraft in order to change the attitude.

Pointing Error In Figure 11a, Figure 11b, and Figure 11c the normalized attitude error of the deputy spacecraft body frame with respect to the reference frame can be observed for a simulation timestep of 1s, 0.1s, and 0.01s respectively. First of all, it can be observed that due to the initial state of the deputy spacecraft (a tumble), the error behavior shows an irregular pattern up to a simulation time of about 10 minutes for each simulation timestep. The error consequently decreases and plateaus. It can be observed that the error decreases with decreasing timestep. In case of a timestep of 1s, the maximum error after convergence is in the order of 10^{-6} while for a timestep of 0.1s and 0.01s the order of the error decreases to 10^{-7} and 10^{-8} respectively. Furthermore, it can be seen that the error in Figure 11a shows periodic behavior as the period of the elliptical orbit is equal to 97 minutes. However, there is a clear distinction between the three graphs observable. Where Figure 11a shows a smooth increase and decrease of the error and Figure 11c practically plateaus (even though a very small increase in error can be observed around 100 and 200 minutes), Figure 11b shows a capricious pattern after convergence. This essentially means that even though the orbit of the deputy spacecraft relative to the chief spacecraft is periodic, the error does not match this behavior.

Due to the fact that a numerical method is used to calculate the angular velocity as well as the angular acceleration of the chief spacecraft relative to the deputy spacecraft, the error does not decrease to the magnitude of the machine error. In case the angular velocity is calculated analytically or an improved numerical method is used, the error is expected to decrease as well. However, it needs to be noted that the scenario displayed is a scenario under perfect conditions. It is expected that factors such as reaction wheel jitter and sensor inaccuracy play a significant role and may overshadow the errors that come from the spacecraft pointing module.

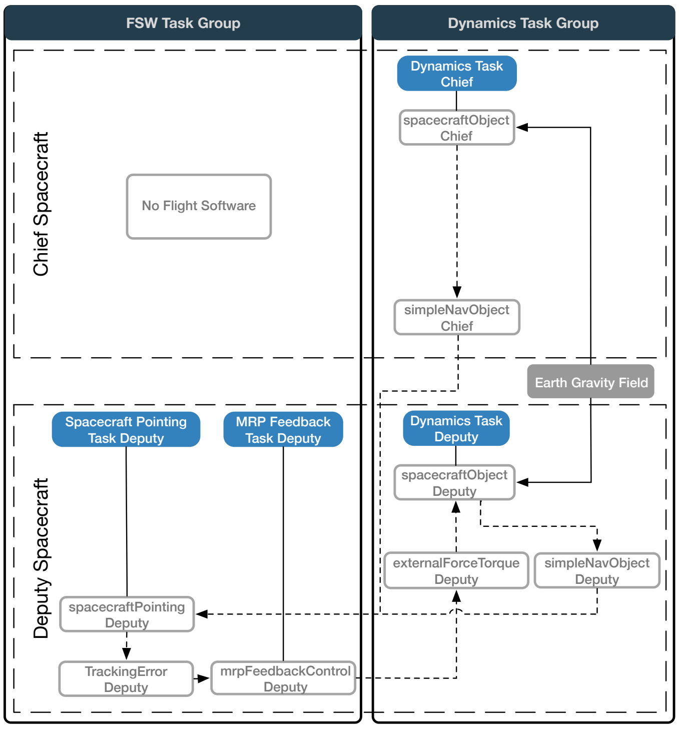
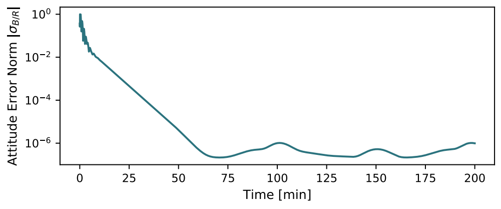
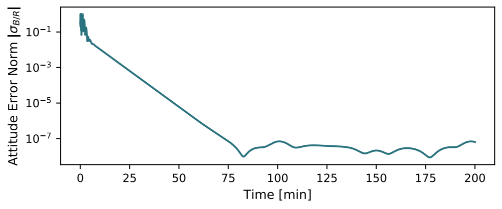


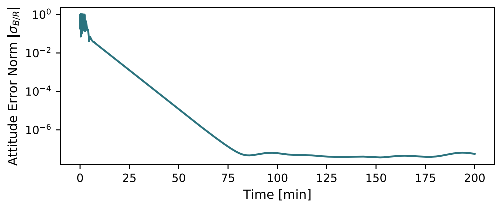
Figure 10: Simulation data flow diagram for deputy spacecraft pointing control module



(a) Coordinate system alignment attitude error norm for timestep of 1s



(b) Coordinate system alignment attitude error norm for timestep of 0.1s



(c) Coordinate system alignment attitude error norm for timestep of 0.01s

Figure 11: Relative pointing module coordinate system alignment attitude error norm.

CONCLUSION

From the analysis of the graphs, it can be concluded that it is possible to perform a formation flying simulation using the Basilisk Astrodynamics Simulation Software Architecture. Due to the fact that different tasks are used for different spacecraft as well as the use of unique message names between modules for each spacecraft it is possible to set up a multi-spacecraft simulation in a modular way. Besides that, in order to increase the number of spacecraft for a formation consisting of equal-type-and-build spacecraft it is only necessary to duplicate the tasks in both task groups and make use of unique message names. This can lead to the simulation of tens of spacecraft at the same time.

Furthermore, Basilisk allows the exchange of data between different spacecraft by designing a flight software control module that is able to read the spacecraft's own state data as well as data from different spacecraft. This leads to a control module that allows a deputy spacecraft to point a vector at the chief spacecraft with errors that decrease with decreasing timestep (after convergence). However, it needs to be noted that the spacecraft in the scenario are able to read the state data directly from a different spacecraft. In reality, the state data needs to be communicated by either making use of an inter-satellite link or a ground station, introducing another error source. The error behavior of this module also changes with decreasing timestep. As the orbital period for this formation is equal to 97 minutes and the chief and deputy spacecraft fly in a Leader/Follower formation, the distance between the two spacecraft increases and decreases periodically, resulting in a change in pointing error. Even though the error decreases for a timestep of 0.1s relative to the error for a timestep of 1s, the error behavior looks irregular. However, eventually decreasing the timestep even more results in an asymptotic decrease in error down to a factor of 10^{-8}.

Finally, it needs to be taken into account that for the evaluation of this formation flying control module ideal conditions are assumed. So in case external factors are taken into account, the error will increase.

RECOMMENDATIONS

With the goal to further improve on this research, future work may include the following set of items.

Even though the setup that is created for the simulation of multiple spacecraft allows for easy expansion of the number of spacecraft, it is required to keep good track of the names of the tasks and modules as well as the message names. For a large number of spacecraft, this can be a very time-consuming job. For this reason research into a method to more easily set up multiple spacecraft can be very beneficial for future users.

This research mainly focuses on the setup of multi-spacecraft simulations using Basilisk. However, in order to perform simulations using thousands of spacecraft, it may be very useful to do research on the increase in runtime with an increasing number of simulated spacecraft.

For the spacecraft pointing module, ideal conditions are assumed. Further research on the effect of adding external factors to this module increases the legitimacy of it.

The behavior of the relative pointing module for a timestep of 0.1s shows an irregular pattern. For this reason, further research should be conducted to test and improve the robustness of this pointing algorithm.

Finally, the type of formation flying that is simulated only allows for control of attitude. The addition of a formation flying station keeping control module is the next step to making the formation flying simulation application more complete.

ACKNOWLEDGEMENT

First of all, I would like to thank Dr. Hanspeter Schaub for giving me the opportunity to work in the Autonomous Vehicle Systems Laboratory for over five months and his support during this internship. Besides that, I would like to thank all Ph.D. students in the lab. Especially Scott Carnahan, Andrew Harris, Patrick Kenneally, John Martin, and Thibaud Teil for their continuous help and the many insights they have given me during my research.

REFERENCES

[1] F. Hadaegh, S. Chung, and H. Manohara, "On Development of 100-Gram-Class Spacecraft for Swarm Application," *IEEE Systems Journal Volume 10 Issue 2*, 2016.

[2] D. Scharf, F. Hadaegh, and S. Ploen, "A survey of spacecraft formation flying guidance and control. Part II: control," *American Control Conference*, 2004.

[3] R. Pirayesh *et al.*, "Formation Flying of a Two-CubeSat Virtual Telescope in a Highly Elliptical Orbit," *2018 SpaceOps Conference*, 2018.

[4] M. Brodecki and Z. Groot, de, "SIGINT: The Mission CubeSats are Made For," *32nd Annual AIAA/USU Conference on Small Satellites*, 2018.

[5] D. CaJacob *et al.*, "Geolocation of RF Emitters with a Formation-Flying Cluster of Three Microsatellites," *30th Annual AIAA/USU Conference on Small Satellites*, 2016.

[6] "Precision Formation Flying, Distributed Spacecraft Technology," `https://dst.jpl.nasa.gov/control/testbeds.htm`. Accessed: 02-18-2019.

[7] G. A. Sohl, S. Udomkesmalee, and J. L. Kellogg, "Distributed Simulation for Formation Flying Applications," *AIAA Modeling and Simulation Technologies Conference and Exhibit San Francisco, California*, 2005.

[8] L. Breger *et al.*, "Distributed Control of Formation Flying Spacecraft Built on OA," *AIAA Guidance, Navigation, and Control Conference and Exhibit*, 2003.

[9] C. Allard *et al.*, "Modular Software Architecture for Fully Coupled Spacecraft Simulations," *Journal of Aerospace Information Systems*, 2018.

[10] P. W. Kenneally, S. Piggott, and H. Schaub, "Basilisk: a flexible, scalable and modular astrodynamics simulation framework," *7th International Conference on Astrodynamics Tools and Techniques (ICATT), DLR Oberpfaffenhofen, Germany*, 2018.

[11] J. Alcorn, H. Schaub, S. Piggott, and D. Kubitschek, "Simulating Attitude Actuation Options Using the Basilisk Astrodynamics Software Architecture," *67th International Astronautical Congress*, 2016.

[12] T. Teil, "Basilisk Technical Memorandum: SPICE Data Interface Module," `http://hanspeterschaub.info/bskHtml/Basilisk-SPICE_INTERFACE20170712.pdf`. Accessed: 02-10-2019.

[13] H. Schaub and J. L. Junkins, *Analytical Mechanics of Space Systems*. AIAA, 4th ed., 2018.

[14] S. J. Carnahan, H. Schaub, and B. Lane, "Impact of Internal Heating on Spacecraft Thermal Signature," *AAS Spaceflight Mechanics Meeting*, 2019.