AIAA SciTech Forum
5-9 January 2015, Kissimmee, Florida
AIAA Guidance, Navigation, and Control Conference

CubeSat Autonomous Rendezvous Simulation

E. Glenn Lightsey* and Andrew J. Fear†
The University of Texas at Austin, Austin, Texas, 78712, USA

An autonomous mission manager is being developed for use on small satellites, including CubeSats, in proximity operations applications where one satellite is near another cooperating spacecraft. The mission manager performs mission event sequencing/resequencing and coordination between the autonomous rendezvous and docking algorithm and the maneuvering satellite while also providing guidance, navigation, and control automation, contingency diagnosis and response, and abort condition determination and execution. In the case of small satellites, limited sensing, actuation, and computing resources require special consideration when creating a mission manager for these vehicles. A detailed simulation tool was created that allows existing guidance, navigation, and control laws to be incorporated into an overall mission manager structure. A representative approach trajectory for a spacecraft from 1 km to 1 m to a cooperating vehicle is used to demonstrate performance. Spacecraft sensor and actuator hardware is simulated so that imperfect knowledge and control may exercise the mission manager algorithms. The system is designed to run in real-time on a standard low power microprocessor that could be used on a CubeSat or similar small satellite.

Nomenclature

a	Acceleration, m/s^2
ϵ	Random walk noise
η	Gaussian zero-mean noise
σ	Standard deviation
τ	Gauss-Markov process time constant, s
t	Time, s
$\mathbf{r}$	Satellite position vector, m
r	Satellite position magnitude, m
$\boldsymbol{b}$	Accelerometer bias, m/s^2
$\boldsymbol{u}$	Satellite control, m/s^2
$\boldsymbol{w}$	Process noise
$\boldsymbol{v}$	Measurement noise
μ	Gravitational constant, m^3/s^2
$\boldsymbol{x}$	State vector
$\mathbf{P}$	State covariance matrix
$\boldsymbol{y}$	Measurement vector
ω	Orbit mean motion, s^{-1}

I. Introduction

Within the past decade, CubeSats have progressed from Sputnik-like radio beacons to more fully featured spacecraft which are capable of performing important science, commercial, and operations-related objectives. A key aspect of this development has been the creation of integrated CubeSat pointing and translational control systems which are now being demonstrated in orbit. Prior research, for example, has led to arc-minute

*Professor, Aerospace Engineering, 210 E 24th St, AIAA Associate Fellow.
†Graduate Research Assistant, Aerospace Engineering, 210 E 24th St, AIAA Student Member.

capable attitude control systems and 3D-printed cold-gas thrusters which will be flown on 3U CubeSats within the next year (2015).[1]

Of particular interest is the case of a CubeSat operating in proximity (within 1 km) of another vehicle, including the challenging case of autonomous rendezvous & docking (AR&D) of two cooperating vehicles. The goal of this research is to enable a maneuverable CubeSat to autonomously (i.e., without direct human operator intervention) sense and control its orientation and position relative to another vehicle in a proximity rendezvous and docking scenario.

The ability of small satellites to safely maneuver relative to other nearby vehicles is a critical enabling capability that has been publicly recognized by the NASA Technology Roadmap (TA04 and TA05),[2] the Office of the Chief Technologist, the Space Technology Mission Directorate, and the Human Exploration and Operations Directorate. Operational capabilities that would be enabled by CubeSat AR&D include on-orbit satellite inspection, servicing, and repair, and eventually, assembly of structures in space using specialized CubeSats as functional building blocks. Once safety considerations have been addressed, maneuverable CubeSats could support human exploration activities by providing enhanced on-orbit situational awareness, and performing tasks through teleoperation, potentially reducing the need for dangerous and costly human Extra-Vehicular Activities (EVAs).

Scientifically, multi-vehicle formations have many applications. There are many cases of multi-point physical measurements that could be used to improve our understanding of the Earth's environment and the Sun-Earth connection. Measuring the temporal and spatial response of the magnetosphere to solar forcing activity is just one of many potential applications that are enabled by low cost multi-point measurements. Performing such a mission with a formation of low cost CubeSats is encompassed within the capability of autonomous maneuverability that is presented in this study.

While many of the pieces for such a mission are being developed separately, an integrated solution is currently lacking. The goal of this research is to create a software defined onboard mission manager which plans, executes, monitors, and updates the rotational and translational maneuvers and resulting trajectories that are needed to accomplish the desired AR&D objectives. Such a process must incorporate the relative dynamics of vehicles in Low Earth Orbit (LEO), ingest absolute and relative sensor measurements to estimate a dynamic state vector, and manage limited actuation resources such as thruster propellant.

In a specific example, the task must be capable of maneuvering the vehicle from an initial standoff distance of 1 km to a relative separation of 1 m (for example) with matching velocities, at which point docking of the vehicles may be accomplished. The software must be able to self-monitor the safety of the projected trajectory throughout the maneuver with regard to collision risk and to put the vehicle in a safe holding trajectory if there is an unacceptable deviation from the intended path. The process must accept high level go/no-go commands and potentially be halted or re-started at any point by a human overseer monitoring the AR&D process from afar. The software must be designed to operate in the presence of single event upsets (SEU) and software resets. The software must execute in real-time in an onboard microprocessor that is used in a commercial-off-the-shelf (COTS) implementation of a low cost CubeSat.

Fortunately, this task may be partitioned using an approach that is technically manageable, leverages current activities, and takes advantage of previously developed resources. The University of Texas at Austin's (UT-Austin's) Texas Spacecraft Lab (TSL) has partnered with NASA's Johnson Space Center (JSC), incorporating their experience in vehicle proximity operations, rendezvous and docking.

This research is specifically tailored to the software mission manager portion of the AR&D task. A software tool known as CubeSat Autonomous Rendezvous & Docking Software (CARDS) is being developed to provide an on-board mission manager for autonomous spacecraft proximity operations. The design approach is to select and implement, rather than re-invent, established algorithms that are suitable for the application. Using separately available algorithms and code, an integrated executable software program is being created and tested that meets the stated AR&D requirements and could run in real-time on a CubeSat embedded microprocessor system. The software is modular and reusable, and will become part of the AR&D Warehouse for future small satellite flight opportunities.[3]

II. System Description

II.A. Overview

CARDS consists of two parts, an environmental simulation and the mission manager software. The purpose of the environmental simulation is to create an analytical setting for the mission manager to be tested. The

mission manager can be considered the core of the CARDS project, as this is where the actual autonomy and guidance portion of the system resides. Details of the environmental simulation and mission manager designs are discussed in this paper.

II.B. Tools

Two software tools were used for creating the simulations for CARDS. This section provides a description of these tools and how they were implemented to create the simulation.

II.B.1. Trick

All of the simulations for CARDS were created using a software package called Trick that was developed at NASA's Johnson Space Center (JSC) in Houston, Texas. Trick is a useful tool for creating and running dynamic simulations, which is why it was chosen for use in this project. An advantage that Trick brings is its ability to easily separate simulation and real-time execution for the user. Under the hood, Trick schedules the simulation functions to run at the appropriate time intervals while also monitoring the real-time clock, and will attempt to take action if it begins to fall behind schedule. Unfortunately, this may result in some skipped simulation execution frames as it will attempt to start the new appropriate execution frame immediately to catch up.

Functions and objects are written in C/C++ to be used by Trick. Simulation objects are defined in a syntax similar to C++ in a file named the S_define file. The S_define file is how Trick knows what objects should exist in the simulation as well as information about the functions that act on the objects. The functions and their function specifications are declared inside the definition of the simulation object. Function specifications are identifiers on functions that explain to Trick the purpose of that particular function, and therefore where and how it should be called during execution. There are numerous types of function specifications including initialization, default data, scheduled, derivative, and integration. The initialization specification means that the function should be called as soon as the simulation is started. The default data specification is typically responsible for setting any parameters that may have been included in a written simulation; this occurs after initialization so that a user can change the parameters of the simulation without worrying about a variable not being initialized. Unlike the initialization and default data functions that are usually run only once per simulation, scheduled functions are run by Trick at a user specified interval. A derivative function is used to calculate any derivatives needed for integration. Integration functions are used to perform Trick's integration. Both derivative and integration functions are called at every simulation time step.

Another advantage of Trick is that it utilizes user-made input files which define the parameters inside a simulation that are read into the simulation at runtime. This allows simulations with differing parameters to run without the need to recompile the whole simulation; only the input file needs to change. These input files are scripts written in the Python language that are processed at execution.[4]

II.B.2. JEOD

A module extension of Trick, known as the JSC Engineering Orbital Dynamics (JEOD) module, is also used. This module "is a collection of computational mathematical models that provide vehicle or vehicles trajectory generation by the solution of a set of dynamics models represented as differential equations."[5] It contains models that are useful for simulating planets and their gravitation, as well as orbit perturbations including atmospheric drag. The planet models have the option to be spherical or non-spherical bodies with spherical harmonics. The atmospheric drag effects are easily able to be turned on or off before running the simulation. JEOD also has the built-in capability for coordinate transformations, utilizing planet-centered inertial, planet-fixed, and local-vertical local-horizontal (LVLH) frames. It also includes relative frames that can be defined in relation to a specified target frame, such as the target satellite. This makes it easier to obtain relative navigation data for the chaser satellite. JEOD splits all of its models into four categories: Dynamics, Environment, Interactions, and Utilities. Fig. 1 shows a diagram for the JEOD classes that are used within Trick. More information regarding JEOD can be found in the JEOD User's Guide provided by NASA JSC.

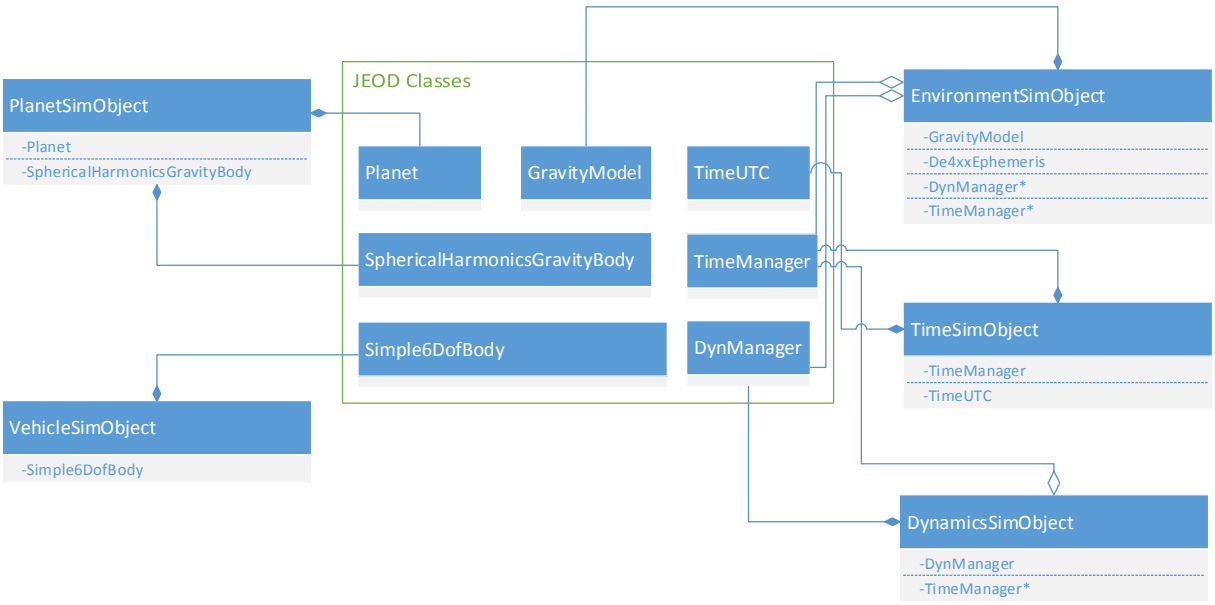

Figure 1. Environmental simulation class diagram showing basic JEOD class implementation

II.C. Sensor Models

As previously mentioned, the vehicle simulation objects possess child objects that model the various onboard sensors. The currently modeled sensors are accelerometers and a Global Positioning System (GPS) receiver. This sensor list is preliminary and will be updated as more aspects of the CubeSat hardware are included in the analysis. This section describes the models used to simulate these onboard sensors.

II.C.1. Accelerometer

The model for each accelerometer sensor measurement is given by

$$\tilde{a} = a + \epsilon_a + \eta_{w,a} \quad (1)$$

$$\dot{\epsilon}_a = -\epsilon_a/\tau_a + \eta_{d,a} \quad (2)$$

where tilde represents a measured sensor value. In other words, the measured acceleration is the truth with an added Gauss-Markov noise term, ϵ_a, and white noise measurement error, $\eta_{w,a}$, with standard deviation, $\sigma_{w,a}$. Eq. (2) represents a first-order Gauss-Markov noise model for the random walk measurement error. The values for the error correlation time constant, τ_a, and the standard deviation, $\sigma_{d,a}$, for the zero-mean Gaussian distribution terms are parameters associated with a physical accelerometer. A manufacturer may give the accelerometer random walk noise, $\sigma_{r,a}$, in units of $\mathrm{m/s^2}/\sqrt{\mathrm{Hz}}$. This is related to the standard deviation for the Gauss-Markov white noise error, $\sigma_{d,a}$, by[6]

$$\sigma_{d,a} = \sigma_{r,a}\sqrt{2/\tau_a} \quad (3)$$

A C++ class to model an accelerometer was defined for use with Trick. The accelerometer class has variables pertaining to the white noise standard deviation, $\sigma_{w,a}$, the time constant, τ_a, and the drift bias, $\sigma_{r,a}$. At every simulation time step interval, the time-correlated noise term ϵ_a is integrated. The accelerometer class has an update function that calculates new measurement values. The rate for the update function to be called is set in the S_define file. At every time step where the update function is called, the accelerometer class object obtains the true acceleration for the vehicle body to which it is attached and adds in the current value for the noise, ϵ_a, as well as a value from the distribution, $\eta_{w,a}$; the result is stored in a variable for the current acceleration measurement.

The parameters for each accelerometer ($\sigma_{w,a}$, τ_a, $\sigma_{r,a}$) are set in the simulations input file. It should be noted that the user inputs $\sigma_{r,a}$ and the model uses Eq. (3) to calculate the value for $\sigma_{d,a}$. Thus, it is easy to perform simulations for modeling different accelerometer devices. The accelerometer noise can also be turned off in the input file by setting the standard deviations for the noise terms to zero and a flag can be set so that the integration of ϵ_a does not occur. This allows for testing using true acceleration values if desired.

II.C.2. GPS

The GPS sensor model uses the same measurement and noise format as the accelerometer sensor model.

$$\tilde{\mathbf{r}} = \mathbf{r} + \boldsymbol{\epsilon}_p + \boldsymbol{\eta}_{w,p} \tag{4}$$

$$\dot{\boldsymbol{\epsilon}}_p = -\boldsymbol{\epsilon}_p/\tau_p + \boldsymbol{\eta}_{d,p} \tag{5}$$

Similar to the accelerometer model, the GPS sensor was written as a C++ class. The GPS class has variables that store the current measured GPS position in the Earth-Centered Inertial reference frame (ECI), as that is the default reference frame for the Trick simulation. However, the reference frame for the GPS measurements can be easily changed to store the Earth-Centered Earth-Fixed (ECEF) position using the built in JEOD reference frame transformations for a more realistic simulation. Like the accelerometer model, the GPS class has an update function that is set to be called at a regular time interval in the S_define file. When the simulation calls a GPS object's update function, the measurement for the current position is calculated. First, the GPS object obtains from JEOD the actual satellite position in the simulation (in the default ECI frame). Then, the value for the drift and random walk noise, $\boldsymbol{\epsilon}_p$, at the current simulation time is added to the actual position, as well as a generated random white noise value from the $\boldsymbol{\eta}_{w,p}$ distribution. The drift and random walk noise is integrated at the simulation's set integration rate and occurs before the GPS update function is called so that the most up-to-date noise value is used in the measurement calculation. In addition, it is assumed that the GPS receiver is also estimating a solution for the spacecraft velocity. The model for the velocity is comparable to the position model, with different values for the noise and time constant variables.

II.D. Kalman Filter

A Kalman filter class was created to perform an extended Kalman filter (EKF) technique to estimate a satellite's position, velocity, and accelerometer bias using the accelerometer and GPS position and velocity measurements. Additionally, the covariance of the state is estimated. First, the standard procedure for finding the estimate of a state and covariance from a process is shown. Then, the model for the chaser and target vehicle state estimation is described.

A standard process has the form shown by Eq. (6) where $\boldsymbol{X}$ represents the $n \times 1$ state vector with n states, $\boldsymbol{u}$ is the $q \times 1$ control vector, and $\mathbf{f}$ is an $n \times 1$ vector-valued function representing the dynamics of the system.[7] The second term, $\mathbf{G}\boldsymbol{w}(t)$, represents uncertainty in the process model, where $\boldsymbol{w}$ is an $n \times 1$ vector of white noise whose standard deviation is a result of the process model and must be tuned.

$$\dot{\boldsymbol{X}}(t) = \mathbf{f}(\boldsymbol{X}, \boldsymbol{u}, t) + \mathbf{G}\boldsymbol{w}(t) \tag{6}$$

$$\boldsymbol{Y}(t) = \mathbf{g}(\boldsymbol{X}, t) + \boldsymbol{v}(t) \tag{7}$$

Equation (7) is the measurement model, where $\boldsymbol{Y}(t)$ is an $m \times 1$ measurement vector and $\boldsymbol{v}(t)$ is an $m \times 1$ white noise vector. The process and measurement noise terms, $\boldsymbol{w}$ and $\boldsymbol{v}$, have zero correlation with covariances $\mathbf{Q}$ and $\mathbf{R}$, respectively.

$$E[\boldsymbol{w}\boldsymbol{v}^T] = \mathbf{0} \tag{8}$$

$$E[\boldsymbol{w}\boldsymbol{w}^T] = \mathbf{Q} \tag{9}$$

$$E[\boldsymbol{v}\boldsymbol{v}^T] = \mathbf{R} \tag{10}$$

It is assumed that there is no cross-correlation between the elements in each noise vector and as a result $\mathbf{Q}$ and $\mathbf{R}$ are diagonal matrices. This model can be linearized by taking the first-order Taylor series expansion of Eq. (6) about the current state estimate $\hat{\boldsymbol{x}}$ with $\boldsymbol{X}(t) = \hat{\boldsymbol{x}}(t) + \boldsymbol{x}(t)$ and $\boldsymbol{Y}(t) = \mathbf{g}(\hat{\boldsymbol{x}}, t) + \boldsymbol{y}(t)$.

$$\dot{\boldsymbol{x}}(t) = \mathbf{A}\boldsymbol{x}(t) + \mathbf{B}\boldsymbol{u}(\boldsymbol{x}, t) \tag{11}$$
$$\boldsymbol{y}(t) = \mathbf{H}\boldsymbol{x}(t) \tag{12}$$

where the matrices $\mathbf{A}$, $\mathbf{B}$, and $\mathbf{H}$ have sizes $n \times n$, $n \times q$, $m \times n$, respectively, and are defined as

$$\mathbf{A} \equiv \partial\mathbf{f}/\partial\boldsymbol{x}|_{\boldsymbol{x}=\hat{\boldsymbol{x}}} \tag{13}$$
$$\mathbf{B} \equiv \partial\mathbf{f}/\partial\boldsymbol{u}|_{\boldsymbol{x}=\hat{\boldsymbol{x}}} \tag{14}$$
$$\mathbf{H} \equiv \partial\mathbf{g}/\partial\boldsymbol{x}|_{\boldsymbol{x}=\hat{\boldsymbol{x}}} \tag{15}$$

The covariance propagation is performed with the following model.

$$\mathbf{P}(t) = E[\boldsymbol{x}(t)\boldsymbol{x}(t)^T] \tag{16}$$
$$\dot{\mathbf{P}}(t) = \mathbf{A}\mathbf{P} + \mathbf{P}\mathbf{A}^T - \mathbf{P}\mathbf{H}^T\mathbf{R}^{-1}\mathbf{H}\mathbf{P} + \mathbf{G}\mathbf{Q}\mathbf{G}^T \tag{17}$$

Assuming an a priori estimate for the state estimate and covariance, it is now possible to update these estimates using the models above at each measurement epoch. If there is no a priori estimate, $\bar{\boldsymbol{x}}(0) = \mathbf{0}$ and $\bar{\mathbf{P}}(0) = \boldsymbol{I}_{n\times n}$ are used. The procedure for updating the state estimate is to first propagate the a priori estimate and covariance to the next measurement epoch. Then, the measurement model is calculated at the current state estimate. The Kalman gain is then found and used to find the new estimated state $\hat{\boldsymbol{x}}$ and covariance $\mathbf{P}$. This procedure is outlined below.

1. Propagate a priori estimates $\bar{\boldsymbol{x}}$ and $\bar{\mathbf{P}}$ using eqs. (18) and (17) to next measurement epoch to obtain $\hat{\boldsymbol{x}}$ and $\mathbf{P}$.
2. Calculate the measurement residual, $\boldsymbol{y}(t) = \boldsymbol{Y}(t) - \mathbf{g}(\hat{\boldsymbol{x}}, t)$
3. Find the Kalman gain $K = (\mathbf{P}\mathbf{H})^{-1}(\mathbf{H}\mathbf{P}\mathbf{H}^T + \mathbf{R}^{-1})$
4. Calculate new best estimate $\hat{\boldsymbol{x}}_{new} = K\boldsymbol{y}$
5. Calculate new covariance $\mathbf{P}_{new} = (\boldsymbol{I}_{n\times n} - K\mathbf{H})\mathbf{P}$
6. Update a priori estimate $\bar{\boldsymbol{x}} = \hat{\boldsymbol{x}}_{new}$, $\bar{\mathbf{P}} = \mathbf{P}_{new}$

The state for a satellite in orbit around Earth is given in terms of position and velocity of the satellite in the ECI reference frame, as well as the accelerometer bias. The vehicle has some 3 degree-of-freedom acceleration control (through thrusters or another actuator). Therefore, the number of states in this case is $n = 9$ with $q = 3$ controls. The resulting state space model is

$$\dot{\boldsymbol{x}}(t) = \begin{bmatrix} \dot{\mathbf{r}} \\ \ddot{\mathbf{r}} \\ \boldsymbol{b} \end{bmatrix} = \begin{bmatrix} \dot{\mathbf{r}} \\ -\mu\mathbf{r}/r^3 + \boldsymbol{u} \\ \mathbf{0} \end{bmatrix} \tag{18}$$

$$\mathbf{A} = \begin{bmatrix} \mathbf{0}_{3\times3} & \boldsymbol{I}_{3\times3} & \mathbf{0}_{3\times3} \\ \mathbf{J} & \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} \\ \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} \end{bmatrix} \tag{19}$$

$$\mathbf{B} = \begin{bmatrix} \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} \\ \mathbf{0}_{3\times3} & \boldsymbol{I}_{3\times3} & \mathbf{0}_{3\times3} \\ \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} \end{bmatrix} \tag{20}$$

$$\mathbf{G} = \boldsymbol{I}_{9\times9} \tag{21}$$

Where

$$\mathbf{J} \equiv \frac{\partial \ddot{\mathbf{r}}}{\partial \mathbf{r}} = -\frac{\mu}{r^3}\boldsymbol{I}_{3\times 3} + \frac{3\mu}{r^5}\mathbf{r}\mathbf{r}^T \tag{22}$$

The measurements being made are the position, velocity, and acceleration of the satellite. In order to account for the bias in the accelerometer measurement the accelerometer bias term is added into the model for the acceleration.

$$\boldsymbol{y}(t) = \begin{bmatrix} \mathbf{r} \\ \dot{\mathbf{r}} \\ -\mu\mathbf{r}/r^3 + \boldsymbol{b} \end{bmatrix} \tag{23}$$

$$\mathbf{H} = \begin{bmatrix} \boldsymbol{I}_{3\times 3} & \mathbf{0}_{3\times 3} & \mathbf{0}_{3\times 3} \\ \mathbf{0}_{3\times 3} & \boldsymbol{I}_{3\times 3} & \mathbf{0}_{3\times 3} \\ \mathbf{J} & \mathbf{0}_{3\times 3} & \boldsymbol{I}_{3\times 3} \end{bmatrix} \tag{24}$$

II.E. Vehicle Models

The CARDS environment is specified for a rendezvous guidance situation. Therefore, there are two vehicles that need to be simulated, the target and the chaser. For preliminary results, the target and chaser have been assumed to be identical in mass and instrumentation. Each vehicle is a separate object of a vehicle simulation class defined in the S_define file. The vehicle simulation class contains instances of JEOD classes that model the mass properties as well as the translational and rotational state of the vehicle. The vehicle class also has objects of the accelerometer and GPS sensor classes. An object of the Kalman filter class is also defined in the vehicle object. The Kalman filter object's update function is called at a regularly scheduled interval to estimate the state of the vehicle object.

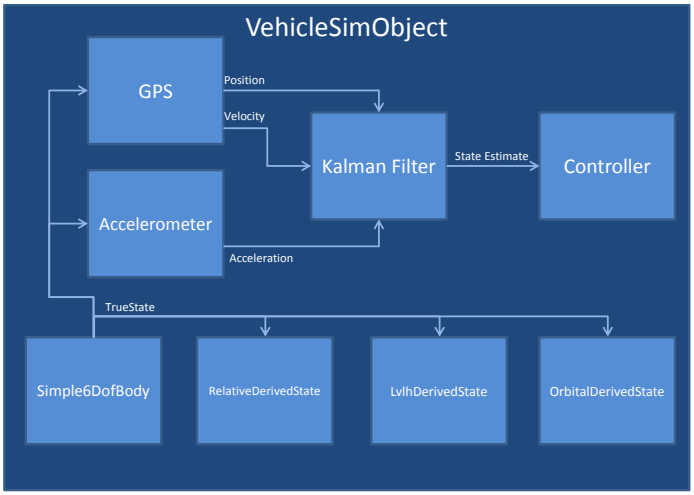

Figure 2. Vehicle simulation block diagram

II.F. Controller

A simple guidance law was implemented to control the chaser satellite to a 1 meter distance from the target satellite. The law implemented is described in a paper by D'Souza[8] using Hill's equations for satellites in near-circular orbits.

$$\dot{\boldsymbol{X}}(t) = \boldsymbol{A}\boldsymbol{X}(t) + \boldsymbol{B}\boldsymbol{u}(t) \tag{25}$$

$$\boldsymbol{X}(t) = [x \quad y \quad z \quad \dot{x} \quad \dot{y} \quad \dot{z}] \tag{26}$$

$$\boldsymbol{A} = \begin{bmatrix} 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 2\omega \\ 0 & -\omega^2 & 0 & 0 & 0 & 0 \\ 0 & 0 & 3\omega^2 & -2\omega & 0 & 0 \end{bmatrix} \tag{27}$$

$$\boldsymbol{B} = \begin{bmatrix} \mathbf{0}_{3\times 3} & \boldsymbol{I}_{3\times 3} \end{bmatrix}^T \tag{28}$$

$$\boldsymbol{u}(t) = -\boldsymbol{B}^{\boldsymbol{T}}\boldsymbol{R}(t)\boldsymbol{Q}^{-1}(t)\left[\boldsymbol{\psi} - \boldsymbol{R}^T(t)\boldsymbol{X}(t)\right] \tag{29}$$

In Eq. (29) the variable $\boldsymbol{\psi} = [x_f \quad y_f \quad z_f \quad \dot{x}_f \quad \dot{y}_f \quad \dot{z}_f]^T$ is the desired state of the controlled vehicle at the given final time, t_f. It can easily be seen that the y direction can be decoupled from the x and z, and as such the control can be split into two different problems to be solved separately. The matrices $\boldsymbol{R}$ and $\boldsymbol{Q}$ are the guidance and controllability matrices whose elements will be defined for each problem. The controllability matrix, $\boldsymbol{Q}$, has the property of symmetry, $\boldsymbol{Q} = \boldsymbol{Q}^T$.

As shown by D'Souza, the coupled control for x and z can be solved using

$$
\begin{aligned}
R_{11} &= 1 && (30)\\
R_{12} &= 0 && (31)\\
R_{13} &= 0 && (32)\\
R_{14} &= 0 && (33)\\
R_{21} &= 6(\omega t_{go} - \sin\omega t_{go}) && (34)\\
R_{22} &= 4 - 3\cos\omega t_{go} && (35)\\
R_{23} &= 6\omega(1 - \cos\omega t_{go}) && (36)\\
R_{24} &= 3\omega\sin\omega t_{go} && (37)\\
R_{31} &= (4\sin\omega t_{go} - 3\omega t_{go})/\omega && (38)\\
R_{32} &= 2(\cos\omega t_{go} - 1)/\omega && (39)\\
R_{33} &= 4\cos\omega t_{go} - 3 && (40)\\
R_{34} &= -2\sin\omega t_{go} && (41)\\
R_{41} &= 2(1 - \cos\omega t_{go})/\omega && (42)\\
R_{42} &= \sin\omega t_{go}/\omega && (43)\\
R_{43} &= 2\sin\omega t_{go} && (44)\\
R_{44} &= \cos\omega t_{go} && (45)
\end{aligned}
$$

$$
\begin{aligned}
Q_{11} &= (3\sin 2\omega t_{go} + 32\sin\omega t_{go} - 24\omega t_{go}\cos\omega t_{go} - 3(\omega t_{go})^3 - 14\omega t_{go})/\omega^3 && (46)\\
Q_{12} &= -3(\sin\omega t_{go} - \omega t_{go})^2/\omega^3 && (47)\\
Q_{13} &= (6\cos 2\omega t_{go} + 8\cos\omega t_{go} + 24\omega t_{go}\sin\omega t_{go} - 9(\omega t_{go})^2 - 14)/2\omega^2 && (48)\\
Q_{14} &= -(3\sin 2\omega t_{go} + 16\sin\omega t_{go} - 12\omega t_{go}\sin\omega t_{go} - 10\omega t_{go})/2\omega^2 && (49)\\
Q_{22} &= -(3\sin 2\omega t_{go} - 32\sin\omega t_{go} + 26\omega t_{go})/4\omega^3 && (50)\\
Q_{23} &= -(3\sin 2\omega t_{go} - 28\sin\omega t_{go} + 22\omega t_{go})/2\omega^2 && (51)\\
Q_{24} &= -(3\cos 2\omega t_{go} - 16\cos\omega t_{go} + 13)/4\omega^2 && (52)\\
Q_{33} &= -(3\sin 2\omega t_{go} - 24\sin\omega t_{go} - 19\omega t_{go})/\omega && (53)\\
Q_{34} &= -(12\sin\frac{\omega t_{go}}{2})^4/\omega && (54)\\
Q_{44} &= (3\sin 2\omega t_{go} - 10\omega t_{go})/4\omega && (55)
\end{aligned}
$$

$$
t_{go} \equiv t_f - t \qquad (56)
$$

Similarly, for the decoupled y coordinate control:

$$
\begin{aligned}
\boldsymbol{R} &= \begin{bmatrix} \cos\omega t_{go} & -\omega\sin\omega t_{go} \\ \frac{\sin\omega t_{go}}{\omega} & \cos\omega t_{go} \end{bmatrix} && (57)\\
Q_{11} &= (\sin 2\omega t_{go} - 2\omega t_{go})/4\omega^3 && (58)\\
Q_{12} &= (\cos 2\omega t_{go} - 1)/4\omega^2 && (59)\\
Q_{22} &= -(\sin 2\omega t_{go} + 2\omega t_{go})/4\omega && (60)
\end{aligned}
$$

II.G. Mission Manager

The primary purpose of the mission manager is to provide a CubeSat with an autonomous guidance system. The system must be capable of monitoring the vehicle's rendezvous path and taking any necessary corrective action in case of maneuver deviation or failed hardware. Although execution of the mission manager is autonomous, human interaction is intended for higher level instructions such as go/no-go commands. The design of the mission manager software is intended to have a few modes to provide nominal functionality with guidance (path) and sensor health monitoring, failure correction, and standby for human input if necessary. The highest level of the mission manager software is a mode switcher that allows for easy transition from each mode to the next depending on events that trigger the mode switch. All of the mission manager software is written in the C++ programming language in preparation for porting to a spacecraft computer.

A base Mode class exists to provide a baseline for the more specific mode classes which inherit from the base Mode. Each mode class has a central function to perform the nominal processes for that mode. The ModeManager has a pointer to a base Mode object which keeps track of the current mode of the mission manager. A main loop performs the current mode's nominal processes indefinitely or until a transition event occurs. In the central function for each mode, checks exist for possible transition events. When an event is triggered, the ModeManager's main loop ends the functions of the current mode and enters the appropriate mode after the transition, beginning the processes for the next mode (which is now the current mode after transition). For instance, a sensor failure is a transition event that would exit the nominal guidance mode and enter standby for human input.

Another design of the mode classes is that each class is a singleton. Only one object of each mode may exist in the software. This is enforced by having a static pointer of each mode's own class type inside of its own definition. The constructor for each mode class is also private so that the mode cannot be created accidentally; a function belonging to each mode class must be called to create an instance of that mode. In addition, this creation function checks if an instance of the object exists by checking the static pointer member. If the static pointer is null, no instance of that mode has yet been created and the constructor is called and the static pointer is set to point to the newly created mode object. Furthermore, static variables such as these mode pointers must be defined at compile-time of the software, meaning that these mode class objects will always exist from start to finish.

A block diagram of the mission manager software is shown in Fig. 3. Classes and threads inside the mission manager software are represented by light blue and orange boxes, respectively. The green box shows human interaction with the mission manager software.

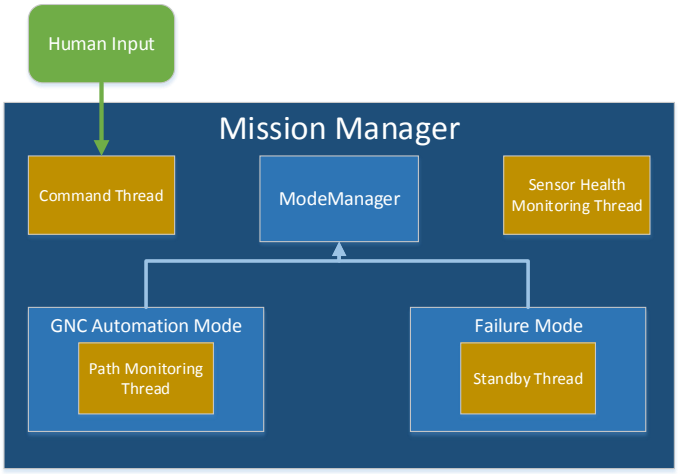

Figure 3. Mission manager block diagram

As stated, an important component of the mission manager software is to determine the guidance maneuvers for the CubeSat. It should be able to control the vehicle from 1 km to a 1 m distance to the target. The chaser satellite is assumed to be in communication with the target satellite and is receiving periodic state updates from the target. The control law in section II.F was implemented for testing, but this controller is intended as an example only in order to demonstrate the operation of the mission manger. The mission manager has a thread that is responsible for monitoring the current and planned path for the vehicle as part of the guidance, navigation, and control (GNC) mode. The monitoring thread constantly checks the chaser vehicle's trajectory for a possible maneuver deviation and will autonomously trigger a corrective action maneuver transition if needed. The mission manager calculates the nominal trajectory by propagating the equations of motion from the maneuver's starting state to the desired final state using the current state estimate as initial conditions. At each measurement epoch, the thread compares the updated state estimate to the calculated nominal state. A performance index is used to determine if the estimated satellite position is too far off from the intended trajectory and a corrective maneuver is needed.

Separately, another thread runs alongside the navigation thread to monitor sensor health. If a sensor stops reporting data, the mission manager will take note and take an appropriate action, which may include standing by for human input if a reboot is desired. Currently, there are plans to also check the case where a sensor is reporting data which is incorrect or "false positive." This is a much harder case to diagnose than if the sensor is no longer responding and more research into determining the validity of sensor data is needed for this part of the mission manager.

Another thread in the mission manager constantly listens for human input. The human input may be override or exit commands. When an instruction is received by the command thread, the message is parsed and the appropriate action is taken. For instance, if the user enters an override command to enter standby, the transition to standby event is triggered as it would have if a real failure had occurred.

The standby thread exists in case of system failure. While the standby thread is running, the mission manager waits until a continue command is received from the user. When the continue command is received by the command thread, the standby thread's exit event is triggered and the mission manager will proceed with its next function.

The mission manager software is currently in development. Thorough testing is planned to ensure all parts of the mission manager software perform as intended. The ModeManager and Mode classes will all undergo unit testing to validate the functionality of each class individually. After each class has been unit tested, full functional testing will be performed on the mission manager software by testing predetermined scenarios to simulate failures.

The sensor health thread will be tested and validated by changing the value of a variable acting as a flag in each sensor object mid-simulation. If a flag that controls whether or not the sensor object is collecting data is set to false, that object's update function does not obtain new values for the sensor's measurement variable. The data collection flag is controlled with a button on a Trick created graphical user interface (GUI). When the button is pressed by the user, the value of the data collection flag toggles between off and on values. Similarly, another flag variable owned by each sensor object is responsible for the false positive data case. When this flag is set to true, the sensor's update function will provide incorrect data measurements. Alternately, scheduled simulation events such as sensor failures can be scripted to occur at designated times through a user defined input file.

A similar testing scheme will be used for the GNC mode thread. Using the same GUI method, the mission manager may receive false state estimates or erroneous measurement data, simulating that the vehicle is on a path different than the nominal. The corrective action taken by the mission manager software is then fed back into the testing GUI, affecting the true vehicle state.

An issue to be resolved is how to determine if the vehicle's sensors have failed due to bad calibration (or other failure) or if the vehicle is off the nominal trajectory when the estimated vehicle state differs from the calculated nominal state. If only one sensor is providing measurements that are not within a to be determined tolerance as the other sensors, then there is a greater chance that a sensor failure has occurred rather than the vehicle being off the calculated path. However, if an unlikely situation presents itself where all sensors have failed in a similar bias, it is a possibility that the mission manager decides the vehicle is deviating from the nominal path yielding an undesired result. The likelihood of this scenario and possible solutions will be researched examined in future work.

III. Results

The environmental simulation was run for approximately 5 orbits (8 hours) with the chaser satellite in orbit around Earth at an altitude of 400 km in cases with and without the controller providing actuation. In this simulation, Earth was modeled as a spherical body and there were no atmospheric drag effects. Typical results from the simulation are presented.

III.A. Sensor Measurements

Values for the sensor noise parameters were chosen to demonstrate simulation functionality. Noise parameters may change depending on further research into acceptable real-world trends and capabilities. In the following sections, the error due to drift for each sensor is shown.

III.A.1. Accelerometer

The following parameters were used for the accelerometer sensor:

$$\begin{aligned} \sigma_{w,a} &= 3.162 \quad \text{cm/s}^2 \\ \sigma_{r,a} &= 3.162 \quad \text{cm/s}^2/\sqrt{\text{Hz}} \\ \tau_a &= 10 \quad \text{min} \end{aligned}$$

III.A.2. GPS

The parameters for the simulated GPS sensor are given below. A subscript v refers to the GPS velocity.

$$\begin{aligned} \sigma_{w,p} &= 1 \quad \text{m} & \qquad \sigma_{w,v} &= 1 \quad \text{km/hr} \\ \sigma_{r,p} &= 3.162 \quad \text{m}/\sqrt{\text{Hz}} & \qquad \sigma_{r,v} &= 2.24 \quad \text{m/s}/\sqrt{\text{Hz}} \\ \tau_p &= 10 \quad \text{min} & \qquad \tau_v &= 10 \quad \text{min} \end{aligned}$$

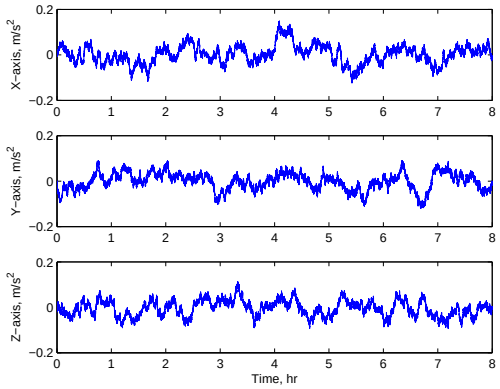

Figure 4. Accelerometer drift using $\sigma_{r,a} = 3.162$ cm/s²/√Hz and $\tau_a = 10$ min

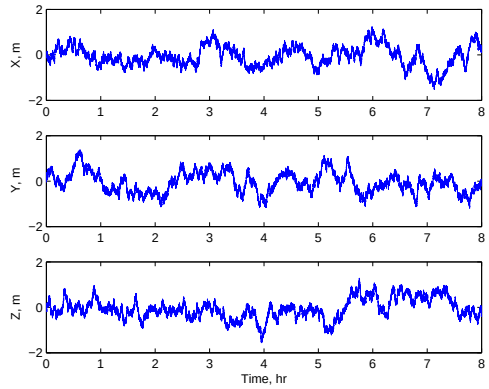

Figure 5. GPS position drift using $\sigma_{r,p} = 3.162$ m/√Hz and $\tau_p = 10$ min

III.B. Kalman Filter

The GPS and accelerometer sensors were modeled as measurements with the parameters described in the previous sections. The measurements were input into the Kalman filter object's update function to perform the state estimation. At each estimate epoch, the difference between the estimated and true states was calculated. A constant 2 m/s^2 bias was added into the accelerometer measurements to demonstrate that the filter correctly estimates and removes the accelerometer bias.

It can be seen that the estimated state has an error that stays near 2 m, which is due to the noise in the GPS position measurements shown in Fig. 5. In Fig. 7 the bias estimation is centered around 2 m/s^2 as expected.

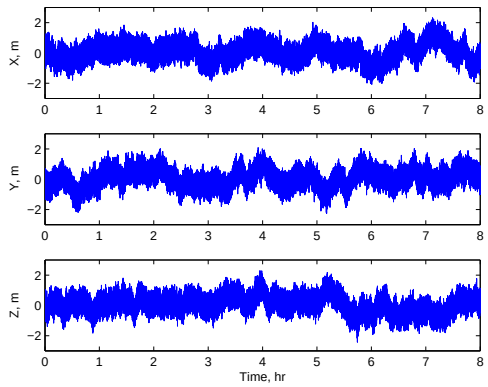

Figure 6. Estimated position error

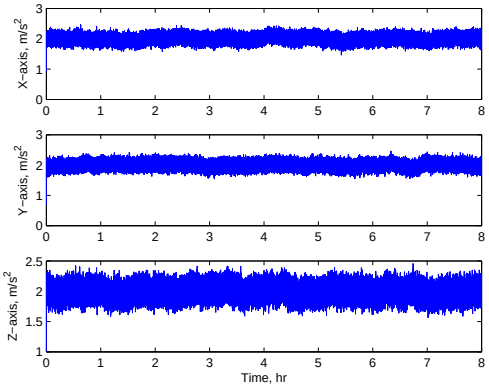

Figure 7. Estimated accelerometer bias

III.C. Controller

The second example by D'Souza was recreated to confirm the implementation of the controller. The chaser satellite was initially at a position of 2 km ahead of the target satellite with a desired final distance of 200 m. The transfer time was defined as 2400 seconds. The results of the maneuver can be seen in Figs. 8-10. It can be seen in Fig. 8 that the vehicle state starts at an initial distance of 2 km in the X-axis and successfully reaches the target distance within the 2400 second transfer time. The measurements given in the testing of the controller were true measurements without noise. This was done to show that the controller was implemented correctly. The controller will operate on noisy measurements when being used in conjunction with the mission manager.

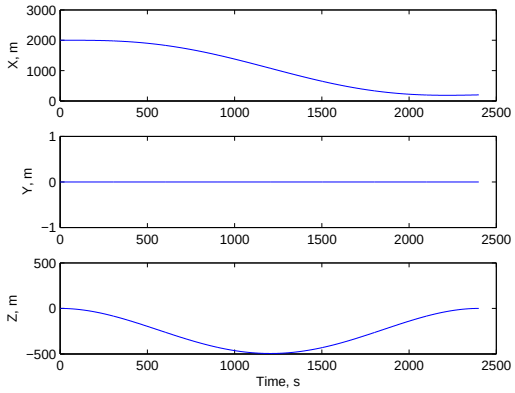

Figure 8. Chaser position

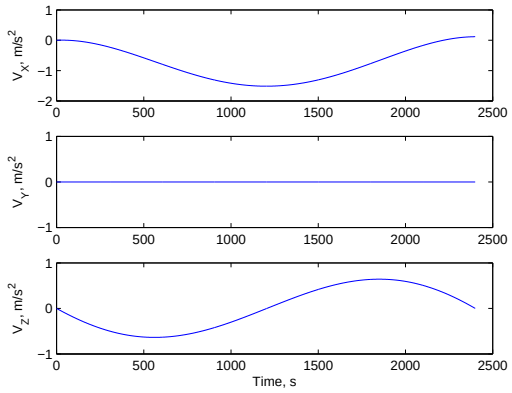

Figure 9. Chaser velocity

IV. Conclusion

A simulation for satellites performing an autonomous rendezvous maneuver was developed using the Trick simulation software with JEOD package. Satellite vehicles were simulated in orbit around Earth with GPS and accelerometer sensor models with noise error. A Kalman filter and control law were implemented for these vehicle objects for a baseline simulation.

A mission manager is currently in development to be used as an autonomous guidance manager for CubeSats. The mission manager software is intended to identify guidance and sensor failures and perform corrective action without human input. The preliminary design for this software was discussed.

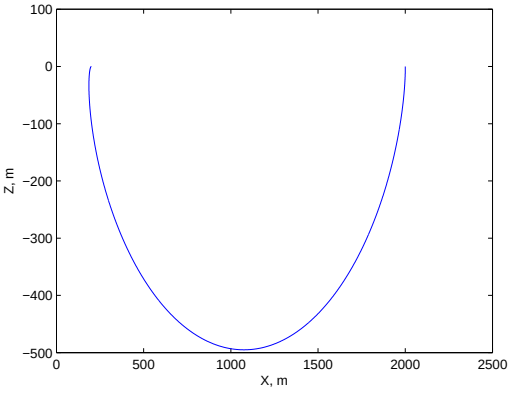

Figure 10. Chaser in-plane trajectory

Acknowledgments

This research was completed under the CubeSat Autonomous Rendezvous and Docking Software grant number NNX13AQ84A by the NASA Space Technology Mission Directorate. NASA JSC has provided access and technical support for the Trick and JEOD software. The authors would like to thank Terry Stevenson for taking the time to provide feedback and support.

References

[1]S. Arestie, E. G. Lightsey, and B. Hudson, "Development of a Modular, Cold Gas Propulsion System for Small Satellite Applications," Journal of Small Satellites, 1(2):63–74, 2012.

[2]"Space Technology Roadmaps: Technology Area Breakdown Structure," URL:http://www.nasa.gov/pdf/501627main_STR-

Int-Foldout_rev11-NRCupdated.pdf [cited May 29, 2014].

[3]"A Proposed Strategy for the U.S. to Develop and Maintain a Mainstream Capability Suite ("Warehouse") for Automated/Autonomous Rendezvous and Docking in Low Earth Orbit and Beyond," February 2012, URL:http://ntrs.nasa.gov/archive/nasa/casi.ntrs.nasa.gov/20120003191_2012002775.pdf [cited May 29, 2014].

[4]"Trick User's Guide," NASA Johnson Space Center, Version 13.1 dev, Houston, Texas, 2013.

[5]Jackson, A. A., and Thebeau, C. D., "JSC Engineering Orbital Dynamics (JEOD) Top Level Document," NASA Johnson Space Center, Rev. 1.3, Houston, Texas, 2013.

[6]Christian, J. A., and Lightsey, E. G., "Sequential Optimal Attitude Recursion Filter," Journal of Guidance, Control, and Dynamics, Vol. 33, No. 6, 2010.

[7]Brown, R., and Hwang, P.,Introduction to Random Signals and Applied Kalman Filtering: With MATLAB Exercises and Solutions, 3rd ed., Wiley, New York, 1997, Chaps. 2, 5, 10.

[8]D'Souza, C., "An Optimal Guidance Law for Formation Flying and Stationkeeping," AIAA Guidance, Navigation, and Control Conference, AIAA, Monterey, California, 2002.