Formation Flight Simulation Framework And Relative Motion Study

Pádraig S. Lysandrou*
University of Colorado Boulder, Boulder, Colorado, 80301
First Project for ASEN6014: Spacecraft Formation Flying

In this project, we create a 6DoF simulation framework which is used to explore relative motion for two spacecraft. We look at the addition of perturbations and their effect on the inaccuracy of simplified relative motion descriptions. Rendezvous and proximity operations are commonplace in the space industry, occurring on a monthly basis, and with new developments still being made in guidance and control. Therefore, the goal of this project was to create a modular 6DoF formation-flying simulation framework that will enable me to develop and employ these guidance and control strategies for the second project and for future research.

Nomenclature

${}^{\mathbf{N}}\mathbf{r}_i$	=	Position vector, inertial frame
${}^{\mathbf{N}}\mathbf{v}_i$	=	Velocity vector, inertial frame
G	=	Gravitational constant
$\boldsymbol{a}_d$	=	Perturbing acceleration
m_i	=	Mass of vehicle i
$[x\ y\ z]$	=	Position components, rectilinear frame
ρ	=	Atmospheric Density
C_D	=	Drag Coefficient
$\boldsymbol{\omega}$	=	Angular velocity vector, rad/s
$\boldsymbol{\sigma}_{B/N}$	=	Modified Rodrigues Parameter of Body relative to inertial
$[I]$	=	Inertia matrix
h	=	Specific angular momentum
$f,\ \dot{f}$	=	True anomaly, True anomaly rate
θ	=	True Latitude
n	=	Mean Motion

I. Introduction

This paper is about building a simulation framework which aims to support of a variety of interesting guidance and control schemes for formation spaceflight. This includes capsule docking to the ISS with keep-out state constraints and actuator timing and impulse constraints. Constrained, optimal, online guidance routines which acknowledge as much of the nonlinear dynamics and actuator physics as possible are attractive research topics. Spacecraft servicing, inspection, grappling, detumbling, and large refueling spacecraft simulations are equally important applications that should be supported with my simulation framework. I have written a modular 6DoF simulation tool that can take arbitrarily many spacecraft with customizable perturbations. I have tested many of these perturbations and also taken a closer look at their affect on relative orbits. I will discuss this in depth in the following pages.

*PhD Student, Aerospace Engineering, AIAA Student Member

II. Task 1: 3DoF Simulation with perturbations

In the first part of this project, we must apply the translational equations of motion for orbiting satellites. We will start with the simple two-body formulation and then add on perturbations of interest. Each of the bodies in our formation abide by their own differential equation:

$$\ddot{\boldsymbol{r}}_i = -\frac{Gm_E}{r_i^3}\boldsymbol{r}_i + \frac{\boldsymbol{f}_d}{m_i(t)} + \frac{\boldsymbol{f}_{u_i}(t)}{m_i(t)} \tag{1}$$

$$= -\frac{Gm_E}{r_i^3}\boldsymbol{r}_i + \boldsymbol{a}_d(t) + \boldsymbol{u}_i(t) \tag{2}$$

Where a_d represents external disturbance accelerations on spacecraft i and u_i represents the input control accelerations on the same vehicle. Numerically integrating these equations of motion with our formation initial conditions allows us to find the position and velocity of each spacecraft over time. We will need the position and velocity vectors for studying relative motion between chosen spacecraft later in this paper.

We include input forces so that in the future we can apply forces to the spacecraft as thrusters would. This will allow us to change our orbital elements and perform maneuvers for rendezvous or proximity operations. Additionally, our numerical integrator keeps track of mass with a simplified thruster formulation of $\dot{m} = -\frac{\sum_k \left\| F_{thrust_k} \right\|}{g_0 I_{sp}}$. This allows us to keep track of the wet and dry mass of the spacecraft over time, and how this affects the equations of motion.

There are a multitude of interesting perturbations on spacecraft that must be accounted for, but, for the sake of brevity, we shall focus on a few. For vehicles with smaller orbital altitudes, the Earth's oblate affect and atmospheric drag can be some of the largest contributing exogenous forces. We shall implement these into our 3DoF translational dynamics moving forward.

A. Perturbation: J2 Oblate Spheroid Acceleration

The gravity potential field of a planet can be modeled with a spherical harmonic series approximation. The dominant harmonic of the solution, the J_2 oblateness perturbation, causes a highly noticeable precession of near-Earth satellite orbits. The gradient of the Earth's gravitational perturbation function $R(\boldsymbol{r})$ produces our required perturbing acceleration in Cartesian coordinates [1]:

$$\boldsymbol{a}_{J_2} = -\frac{3}{2}\frac{\mu J_2 R_e^2}{r^4}\left\{\begin{array}{c} \left(1 - 5\frac{z^2}{r^2}\right)\frac{x}{r} \\ \left(1 - 5\frac{z^2}{r^2}\right)\frac{y}{r} \\ \left(3 - 5\frac{z^2}{r^2}\right)\frac{z}{r} \end{array}\right\} \tag{3}$$

where R_e is the radius of the Earth, r is the norm of the spacecraft position vector, and J_2 is zonal harmonic constant $1082.63E^{-6}$. This effect is apparent in figure 2.

B. Perturbation: Simplified Atmospheric Drag Model

Another large disturbance for LEO spacecraft is atmospheric drag. The International Space Station has to perform orbit-raise maneuvers regularly to combat the constant decrease in orbital energy due to atmospheric drag. For the sake of simplicity, we assume an even drag coefficient and therefore invariant with respect to attitude. Knowing that

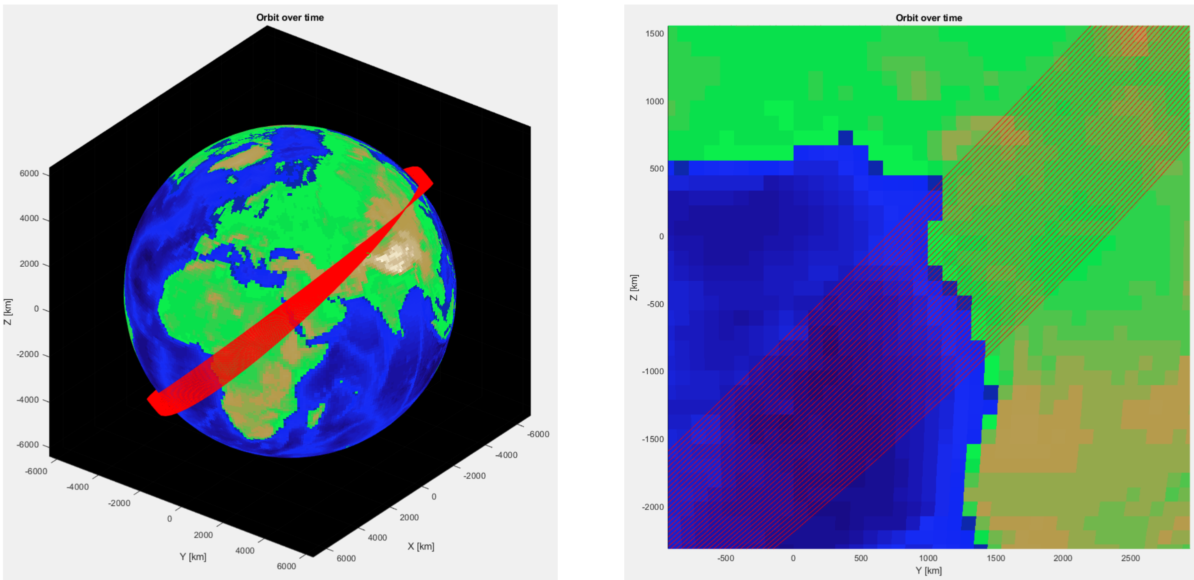
$$\boldsymbol{F}_{\mathrm{D}} = -\frac{1}{2}\mathrm{C_D}\frac{\mathrm{A}}{\mathrm{m}}\rho(t)\|\boldsymbol{V}\|\,\boldsymbol{V} \tag{4}$$

where $\boldsymbol{V}$ is the velocity of the satellite relative to the atmosphere. We assume an exponential atmospheric model of form $\rho(h) = \rho_0 e^{\frac{-h}{H}}$ where h represents our altitude. A is the reference area of the satellite, and C_D is the drag coefficient. Via Newtonian flow, and with our spherical assumption, this is 2. By the transport theorem, we can calculate the velocity of the atmosphere, assuming that it rotates with the planet. This assumption implies that the particles are coupled to the earth just as the particles close to the surface. We can modify this assumption for better accuracy [2] with the following equation:

$$V = \dot{r} - \omega_{A/N} \times r \tag{5}$$

$$= \dot{r} - \left(\omega_E \frac{R_E}{\|r\|}\right) \times r \tag{6}$$

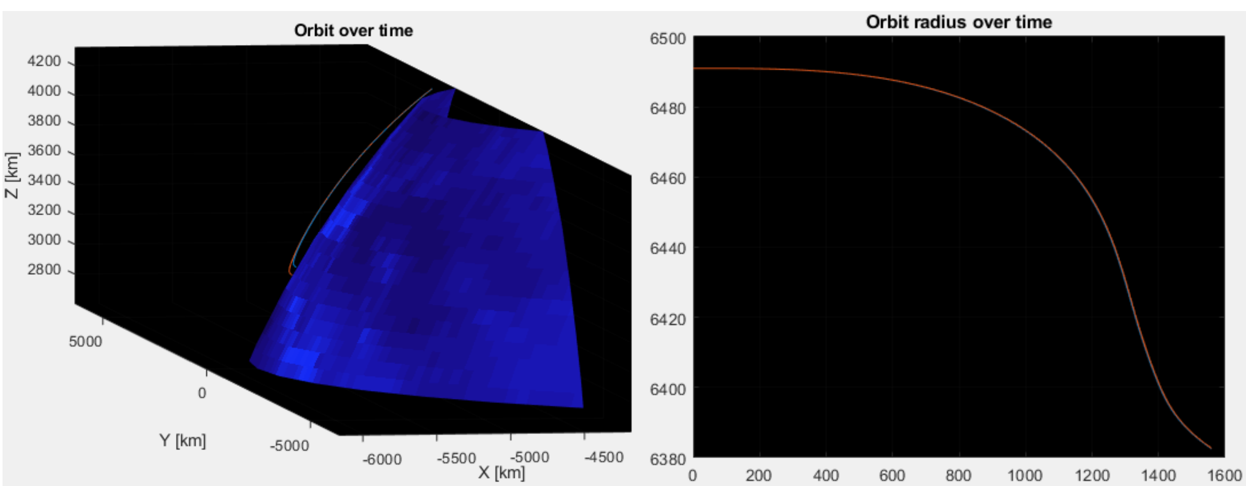
where ω_E is the rotation of the Earth in the inertial frame.



(a) View of the orbit precession history

(b) Closer YZ plane view, projected onto Earth

Fig. 1 Orbital effect from J2 oblate spheroid perturbing acceleration



(a) Chief and Deputy co-deorbiting

(b) Magnitude of radius vectors, kilometers versus seconds

Fig. 2 Atmospheric drag acting on two spacecraft with an initial altitude of 120km.

III. Task 2: 6DoF Simulation with perturbations

The goal of this task was to incorporate the extra rotational degrees of freedom into this simulation framework so that I can understand and simulate attitude dependent control laws later on. In this framework, we shall use Modified Rodrigues Parameters as our attitude formalism. Let us define our rotational state vector as the following:

$$\boldsymbol{X} = \begin{bmatrix} \boldsymbol{\sigma}_{BN} \\ {}^{B}\boldsymbol{\omega}_{BR} \end{bmatrix} \tag{7}$$

The $^{\times}$ symbol denotes the skew symmetric matrix form of a cross product, for $\boldsymbol{u}$ control torque vector, we know the the rigid body dynamics obey the following:

$$[I]\dot{\omega}_{B/N} = -[\omega^{\times}_{B/N}][I]\omega_{B/N} + \boldsymbol{u} + \boldsymbol{L} \tag{8}$$

This can be solved for purely $\dot{\boldsymbol{\omega}}_{BN}$ on the left hand side. Without going into the derivation, we also know the following to be the kinematic differential equation for the MRP of the vehicle [1]:

$$\dot{\boldsymbol{\sigma}}_{BN} = \frac{1}{4}[(1-\sigma^2)\mathbb{I}_3 + 2[\boldsymbol{\sigma}^{\times}] + 2\boldsymbol{\sigma}\boldsymbol{\sigma}^T]^{\mathcal{B}}\boldsymbol{\omega}_{BN} \tag{9}$$

Given that the attitude must be constrained to within the unit sphere, and the MRP needs to be switched to the shadow set, we must write our own integrator where we can have control over this function. We use a fourth order Runge-Kutte algorithm for integration (RK4). Using the nonlinear dynamics function $\dot{\boldsymbol{X}} = f(t, \boldsymbol{X})$, the integration is Algorithm 1. Each point $i \in [1 : N]$ is 1 integration time step, and therefore the full simulation time dtN.

Algorithm 1 Fourth Order Runge Kutte Integrator

for i = 1:N-1 **do**
- $k_1 = \dot{\boldsymbol{X}}(t(i),\ \boldsymbol{X}(:,i))$
- $k_2 = \dot{\boldsymbol{X}}(t(i) + \frac{dt}{2},\ \boldsymbol{X}(:,i) + \frac{dt}{2}k_1)$
- $k_3 = \dot{\boldsymbol{X}}(t(i) + \frac{dt}{2},\ \boldsymbol{X}(:,i) + \frac{dt}{2}k_2)$
- $k_4 = \dot{\boldsymbol{X}}(t(i) + dt,\ \boldsymbol{X}(:,i) + dtk_3)$
- $\boldsymbol{X}(:,i+1) = \boldsymbol{X}(:,i) + \frac{dt}{6}(k_1 + 2k_2 + 2k_3 + k_4)$

end for

A. Perturbation: Gravity Gradient Torques

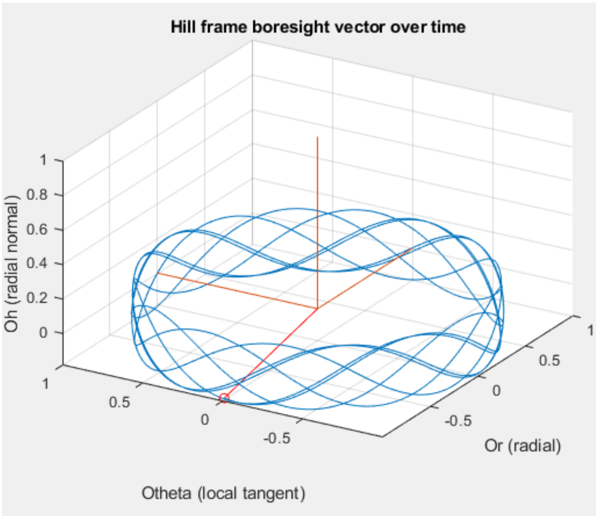
In addition to translational perturbations, there are also rotational perturbations. For the sake of this project, I have chosen only to implement gravity gradient torques. These torques become large the closer you are to the planet: they scale as a $\frac{1}{R^3}$ law. The following gravity gradient torque, from [1]:

$$\boldsymbol{L}_G = \frac{-3GM_E}{r_c^5}\boldsymbol{r}_c \times [\boldsymbol{I}]\boldsymbol{r}_c \tag{10}$$

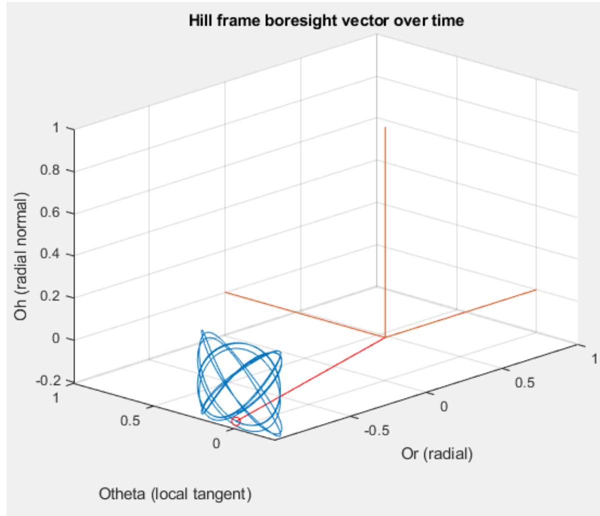
Where $[\boldsymbol{I}]$ is the spacecraft inertia matrix, GM_E is the gravitational parameter of Earth, and $\boldsymbol{r}_c$ is the vector to the center of mass of each spacecraft in our formation. The most stable initial condition is with the body vector along the most elongated inertia axis pointing down in the negative radial direction. Additionally, the spacecraft should have an angular rate congruent with that of the true anomaly rate and in the $\hat{\boldsymbol{h}}$ direction. In Figure 3 we see one spacecraft not under torques, and another under gradient torques. These spacecraft have an off-boresight initial attitude, in an 500km circular orbit, to show the motion. The boresight sweep is passively bounded by a haversine-box which maps out an angular displacement of a sphere.

B. Perturbation: Atmospheric Torques

Although I did not get a chance to implement this feature, I can at least spend some time about the mathematics and details behind how this would go into my simulation framework. Recall that I derived the velocity of the atmosphere



(a) Spacecraft rotation without gravity gradient



(b) Stabilizing effect of gravity gradient

Fig. 3 The gravity gradient stabilizing effect is shown on the right. Both plots show spacecraft dynamics with the same initial conditions and in the Hill frame. The bases vectors are shown in the middle, pointing radial, along-track, and normal. The initial attitude MRPs are $\sigma_{B/N} = [0.5065 \ \ -0.4623 \ \ 0.0784]^T$ for a spacecraft with $[I_c] = diag([33334\ 33334\ 1])$.

relative to the spacecraft in the inertial frame. We can then look at this in the body frame to extract torques, a method taken from the Markley and Crassidis text [3].

$$\boldsymbol{V}_{relB} = [BN(\boldsymbol{\sigma}_{B/N})]\boldsymbol{V} \tag{11}$$

We can assume a design of the spacecraft made up of flat plates, where the inclination of each of the plates with respect to the relative velocity is the following:

$$\cos\theta^i_{\text{aero}} = \frac{\mathbf{n}^i_B \cdot \boldsymbol{V}_{relB}}{\|\boldsymbol{V}_{relB}\|} \tag{12}$$

where $\mathbf{n}^i_B$ is a normal vector to each of the plates. We can then show that the proper aerodynamic force to pay attention to is simply

$$\boldsymbol{F}^i_{\text{aero}} = -\frac{\rho(h)C_D}{2}\|\boldsymbol{V}_{relB}\|\,\boldsymbol{V}_{relB}A_i \max\left(\cos\theta^i_{\text{aero}}, 0\right) \tag{13}$$

In this case, if the reported cosine value is negative, it is now behind the flow and unimportant. The drag coefficient for a flat plate using the Newtonian flow assumption is simply $C_D = 2\sin^3\alpha$ where α is the total angle of attack, or in this case just θ^i_{aero}. Moving forward we can then formulate the total amount of aerodynamic torque, for a convex shaped spacecraft, as the following:

$$\mathbf{L}^i_{\text{aero}} = \sum_{i=1}^{N} \boldsymbol{r}^i \times \mathbf{F}^i_{\text{aero}} \tag{14}$$

where $\boldsymbol{r}^i$ is the position from the center of mass of the vehicle to the center of pressure of that particular plate.

IV. Task 3: Relative Motion Description Study

In the formation case of two spacecraft, one chief and one deputy, with $\mathbf{r}_c$ being the Cartesian position vector of the chief and $\mathbf{r}_d$ being the Cartesian position vector of the deputy, we say that:

$$\boldsymbol{r}_d = \boldsymbol{r}_c + \boldsymbol{\rho} = (r_c + x)\hat{\boldsymbol{o}}_r + y\hat{\boldsymbol{o}}_\theta + z\hat{\boldsymbol{o}}_h \tag{15}$$

Where the unit vectors listed are elements of the Hill frame coordinate system, where the DCM transforming an inertial frame vector to a Hill frame one is described as such: $[HN] = [\hat{\boldsymbol{o}}_r^T\ \hat{\boldsymbol{o}}_\theta^T\ \hat{\boldsymbol{o}}_h^T]^T$. Instantaneously each vector can be found with $\hat{\boldsymbol{o}} = \frac{\boldsymbol{r}}{\|\boldsymbol{r}\|}$, $\hat{\boldsymbol{o}}_h = \frac{\boldsymbol{r}\times\boldsymbol{v}}{\|\boldsymbol{r}\times\boldsymbol{v}\|}$, and $\hat{\boldsymbol{o}}_\theta = \hat{\boldsymbol{o}}_h \times \hat{\boldsymbol{o}}_r$. Using the true anomaly rate, for any Keplerian orbit, $\boldsymbol{\omega}_{H/N} = \dot{f}\hat{\boldsymbol{o}}_h$ in the hill frame, we can take the inertial frame transport of the position vector:

$$\dot{\boldsymbol{r}}_d = (\dot{r}_c + \dot{x} - \dot{f}y)\hat{\boldsymbol{o}}_r + (\dot{y} + \dot{f}(r_c + x))\hat{\boldsymbol{o}}_\theta + \dot{z}\hat{\boldsymbol{o}}_h \tag{16}$$

$$\ddot{\boldsymbol{r}}_d = (\ddot{r}_c + \ddot{x} - 2\dot{y}\dot{f} - \ddot{f}y - \dot{f}^2(r_c + x))\hat{\boldsymbol{o}}_r + (\ddot{y} + 2\dot{f}(\dot{r}_c + \dot{x}) + \ddot{f}(r_c + x) - \dot{f}^2 y)\hat{\boldsymbol{o}}_\theta + \ddot{z}\hat{\boldsymbol{o}}_h \tag{17}$$

We can also take advantage of the following relationship of the specific angular momentum of the chief:

$$h = r_c^2\dot{f} \tag{18}$$

$$\dot{h} = 2r_c\dot{r}_c\dot{f} + r_c^2\ddot{f} = 0 \tag{19}$$

$$\ddot{f} = -2\frac{\dot{r}_c}{r_c}\dot{f} \tag{20}$$

The angular momentum rate is zero for Keplerian orbits, but of course non-zero for orbits under perturbations. To simplify the equations of motion, we can develop the dynamics of the chief by taking successive time derivatives with respect to the intertial frame:

$$\boldsymbol{r}_c = r_c\hat{\boldsymbol{o}}_r \tag{21}$$

$$\dot{\boldsymbol{r}}_c = \dot{r}_c\hat{\boldsymbol{o}}_r + \boldsymbol{\omega}_{H/N} \times r_c\hat{\boldsymbol{o}}_r \tag{22}$$

$$= \dot{r}_c\hat{\boldsymbol{o}}_r + \dot{f}r_c\hat{\boldsymbol{o}}_\theta \tag{23}$$

$$\ddot{\boldsymbol{r}}_c = \ddot{r}_c\hat{\boldsymbol{o}}_r + (\ddot{f}r_c + \dot{f}\dot{r}_c)\hat{\boldsymbol{o}}_\theta + (\dot{f}\hat{\boldsymbol{o}}_h) \times (\dot{r}_c\hat{\boldsymbol{o}}_r + \dot{f}r_c\hat{\boldsymbol{o}}_\theta) \tag{24}$$

$$= (\ddot{r}_c - \dot{f}^2 r_c)\hat{\boldsymbol{o}}_r + (\ddot{f}r_c + 2\dot{f}\dot{r}_c)\hat{\boldsymbol{o}}_\theta \tag{25}$$

$$= (\ddot{r}_c - \dot{f}^2 r_c)\hat{\boldsymbol{o}}_r + (-2\frac{\dot{r}_c}{r_c}\dot{f}r_c + 2\dot{f}\dot{r}_c)\hat{\boldsymbol{o}}_\theta \tag{26}$$

$$= (\ddot{r}_c - \dot{f}^2 r_c)\hat{\boldsymbol{o}}_r \tag{27}$$

And by the two-body orbital equations of motion, we know that $\ddot{\boldsymbol{r}}_c = (\ddot{r}_c - \dot{f}^2 r_c)\hat{\boldsymbol{o}}_r = -\frac{\mu}{r_c^2}\hat{\boldsymbol{o}}_r$. We can also see, that by equating the vectorial components, we get $\ddot{r}_c = \dot{f}^2 r_c - \frac{\mu}{r_c^2} = \dot{f}^2 r_c(1 - \frac{r_c}{p})$ knowing that the semiparameter is $p = \frac{h^2}{\mu} = \frac{r_c^4\dot{f}^2}{\mu}$. Now we can simplify the deputy equations of motion via substitution of our developed relationships.

$$\ddot{\boldsymbol{r}}_d = ((\dot{f}^2 r_c(1 - \frac{r_c}{p})) + \ddot{x} - 2\dot{y}\dot{f} - (-2\frac{\dot{r}_c}{r_c}\dot{f})y - \dot{f}^2(r_c + x))\hat{\boldsymbol{o}}_r + (\ddot{y} + 2\dot{f}(\dot{r}_c + \dot{x}) + (-2\frac{\dot{r}_c}{r_c}\dot{f})(r_c + x) - \dot{f}^2 y)\hat{\boldsymbol{o}}_\theta + \ddot{z}\hat{\boldsymbol{o}}_h \tag{28}$$

$$= (-\frac{\mu}{r_c^2} + \ddot{x} - 2\dot{y}\dot{f} + 2\frac{\dot{r}_c}{r_c}\dot{f}y - \dot{f}^2 x)\hat{\boldsymbol{o}}_r + (\ddot{y} + 2\dot{f}(\dot{x} - \frac{\dot{r}_c}{r_c}x) - \dot{f}^2 y)\hat{\boldsymbol{o}}_\theta + \ddot{z}\hat{\boldsymbol{o}}_h \tag{29}$$

$$= (\ddot{x} - 2\dot{f}(\dot{y} - y\frac{\dot{r}_c}{r_c}) - \dot{f}^2 x - \frac{\mu}{r_c^2})\hat{\boldsymbol{o}}_r + (\ddot{y} + 2\dot{f}(\dot{x} - x\frac{\dot{r}_c}{r_c}) - \dot{f}^2 y)\hat{\boldsymbol{o}}_\theta + \ddot{z}\hat{\boldsymbol{o}}_h \tag{30}$$

And writing the orbital equation of motion, with the fact that the norm of 15 is $r_d = \sqrt{(r_c + x)^2 + y^2 + z^2}$:

$$ {}^{H}\ddot{\boldsymbol{r}}_d = -\frac{\mu}{r_d^3}\begin{bmatrix} r_c + x \\ y \\ z \end{bmatrix} \tag{31} $$

We can equate the scalar components of the vectors to get the exact nonlinear relative equations of motion, making no assumptions other than Keplerian motion:

$$ \ddot{x} - 2\dot{f}(\dot{y} - y\frac{\dot{r}_c}{r_c}) - \dot{f}^2 x - \frac{\mu}{r_c^2} = -\frac{\mu}{r_d^3}(r_c + x) \tag{32} $$

$$ \ddot{y} + 2\dot{f}(\dot{x} - x\frac{\dot{r}_c}{r_c}) - \dot{f}^2 y = -\frac{\mu}{r_d^3} y \tag{33} $$

$$ \ddot{z} = -\frac{\mu}{r_d^3} z \tag{34} $$

A. Considering Perturbed Motion in the Chief Frame

To consider perturbed motion, we must reformulate the chief equations such that they are perturbed by general acceleration $\boldsymbol{a}_d$ where this could be environmental perturbation or a control acceleration like a thruster. Our chief orbit frame angular velocity expression used initially is now naive. By examining the specific orbital angular momentum vector, we gain insight:

$$ \boldsymbol{h} = \boldsymbol{r}_c \times \dot{\boldsymbol{r}}_c = \boldsymbol{r}_c \times (\frac{{}^{H}d}{dt}\boldsymbol{r}_c + \boldsymbol{\omega}_{H/N} \times \boldsymbol{r}_c) \tag{35} $$

$$ = \boldsymbol{r}_c \times (\boldsymbol{\omega}_{H/N} \times \boldsymbol{r}_c) \tag{36} $$

We are able to make a simplification due to $\boldsymbol{r}_c = r\hat{\boldsymbol{o}}_r$ and ${}^{H}\dot{\boldsymbol{r}}_c = \dot{r}\hat{\boldsymbol{o}}_r$ being colinear. We can further solve, using the triple product identity, to find that $\boldsymbol{h} = r_c^2\boldsymbol{\omega}_{H/N} - (\boldsymbol{\omega}_{H/N} \cdot \boldsymbol{r}_c)\boldsymbol{r}_c$. This brings us to see that

$$ \boldsymbol{\omega}_{H/N} = \frac{\boldsymbol{h}}{r_c^2} + \omega_r\hat{\boldsymbol{o}}_r \tag{37} $$

This may seem useless at first, but will prove to be helpful soon. We see that this angular velocity only has components in the radial and out-of-plane axes. An angular rate in the theta direction would instantaneously misalign the orbit (and radial vector). To move forward with looking at perturbations in the chief frame, we must take the time derivative of the specific angular momentum vector:

$$ \dot{\boldsymbol{h}} = \boldsymbol{r}_c \times \boldsymbol{a}_d = \frac{{}^{H}d}{dt}\boldsymbol{r}_c + \boldsymbol{\omega}_{HN} \times \boldsymbol{h} \tag{38} $$

$$ = \dot{h}\hat{\boldsymbol{o}}_h + \left(\frac{\boldsymbol{h}}{r_c^2} + \omega_r\hat{\boldsymbol{o}}_r\right) \times \boldsymbol{h} \tag{39} $$

$$ = \dot{h}\hat{\boldsymbol{o}}_h - \omega_r h\hat{\boldsymbol{o}}_\theta \tag{40} $$

Of course, the vectorial decomposition of the perturbing acceleration is just $\boldsymbol{a}_d = a_r\hat{\boldsymbol{o}}_r + a_\theta\hat{\boldsymbol{o}}_\theta + a_h\hat{\boldsymbol{o}}_h$. This allows us to write $\boldsymbol{r} \times \boldsymbol{a}_d = ra_\theta\hat{\boldsymbol{o}}_h - ra_h\hat{\boldsymbol{o}}_\theta$. Therefore, we have the following formulation:

$$ra_\theta \hat{\boldsymbol{o}}_h - ra_h \hat{\boldsymbol{o}}_\theta = \dot{h}\hat{\boldsymbol{o}}_h - \omega_r h \hat{\boldsymbol{o}}_\theta \tag{41}$$

$$\omega_r = \frac{ra_h}{h} \tag{42}$$

$$\dot{h} = ra_\theta \tag{43}$$

Substituting these values into 37, we have that

$$\boldsymbol{\omega}_{H/N} = \frac{\boldsymbol{r}_c \times \dot{\boldsymbol{r}}_c}{r_c^2} + \frac{\boldsymbol{r}_c a_h}{h} \tag{44}$$

These will be useful later when we need to perform mappings from perturbed Cartesian coordinates to Hill frame coordinates to understand inaccuracies of some the assumptions that will be made.

B. Simplifications for Derivation of the Clohessy-Wiltshire Equations

Continuing with our nonlinear, unperturbed, equations of motion 32, we can further simplify for a circular chief orbit. This will produce a set of linear dynamics which can be useful for analytical insight and computationally simplified algorithm development. We can first approximate r_d by dropping quadratic or higher terms, making it $r_d \approx \sqrt{1 + \frac{2x}{r_c}}$. Looking at how this affects our two body dynamics, we can perform the expansion about x:

$$\frac{\mu}{r_d^3} \approx \frac{\mu}{r_c^3}\left(1 - \frac{3x}{r_c} + \frac{15x^2}{2r_c^2} - \frac{35x^3}{2r_c^3} + \frac{315x^4}{8r_c^4} + O\left(x^5\right)\right) \tag{45}$$

$$\approx \frac{\mu}{r_c^3}\left(1 - \frac{3x}{r_c}\right) \tag{46}$$

$$\therefore \quad {}^H\ddot{\boldsymbol{r}}_d \approx -\frac{\mu}{r_c^3}\left(1 - \frac{3x}{r_c}\right)\begin{bmatrix} r_c + x \\ y \\ z \end{bmatrix} \tag{47}$$

$$\approx -\frac{\mu}{r_c^3}\begin{bmatrix} r_c - 2x \\ y \\ z \end{bmatrix} \tag{48}$$

We truncate quadratic and higher terms as done before. Similarly, we eliminate crossterms in the approximation. We know that $p = \frac{h^2}{\mu}$ and $h = r^2\dot{f}$. Plugging these into the expression for the chief orbit equations we have that $\frac{\mu}{r_c^3} = \frac{r_c\dot{f}^2}{p}$. Furthermore, knowing that $r = \frac{p}{1+e\cos f}$, we find that $\frac{\mu}{r_c^3} = \frac{\dot{f}^2}{1+e\cos f}$. Using these simplifications, we can substitute into 32 to create a new set of equations

$$\begin{aligned} \ddot{x} - x\dot{f}^2\left(1 + 2\frac{r_c}{p}\right) - 2\dot{f}\left(\dot{y} - y\frac{\dot{r}_c}{r_c}\right) &= 0 \\ \ddot{y} + 2\dot{f}\left(\dot{x} - x\frac{\dot{r}_c}{r_c}\right) - y\dot{f}^2\left(1 - \frac{r_c}{p}\right) &= 0 \\ \ddot{z} + \frac{r_c}{p}\dot{f}^2 z &= 0 \end{aligned} \tag{49}$$

Simplifying further with true latitude $\theta = \omega + f$ we can write them again as the following:

$$
\begin{aligned}
\ddot{x} - x\left(\dot{\theta}^2 + 2\frac{\mu}{r_c^3}\right) - y\ddot{\theta} - 2\dot{y}\dot{\theta} &= 0 \\
\ddot{y} + x\ddot{\theta} + 2\dot{x}\dot{\theta} - y\left(\dot{\theta}^2 - \frac{\mu}{r_c^3}\right) &= 0 \\
\ddot{z} + \frac{\mu}{r_c^3}z &= 0
\end{aligned}
\tag{50}
$$

Finally, we shall make the assumption that the chief is on a circular, non-eccentric orbit. This means that $\dot{r_c} = 0,\ e = 0,\ \dot{f} = n$. Using these assumptions, our equations drop down to:

$$\ddot{x} - 2n\dot{y} - 3n^2x = 0 \tag{51}$$

$$\ddot{y} + 2n\dot{x} = 0 \tag{52}$$

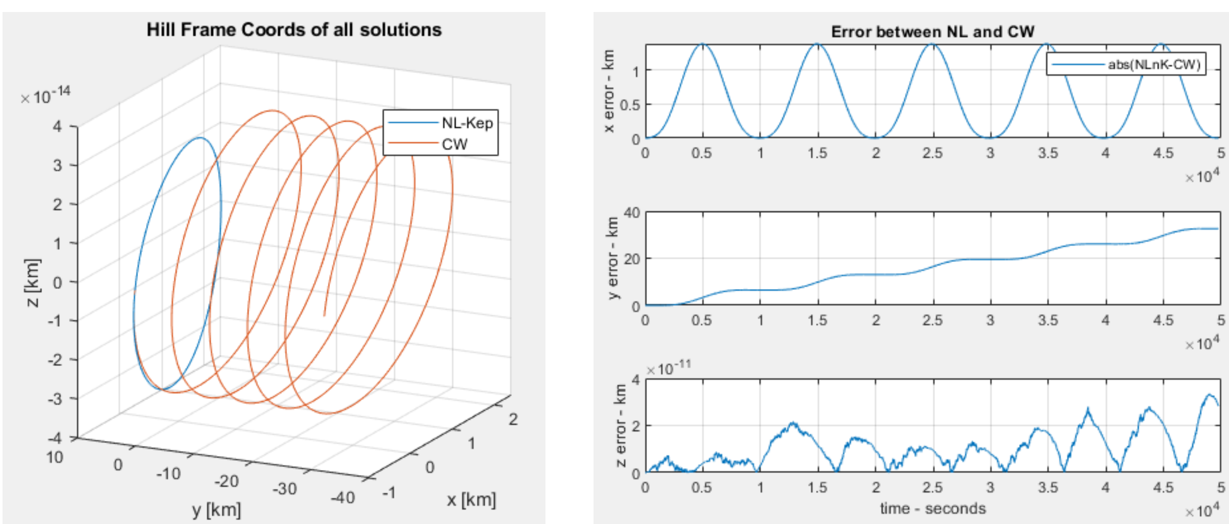
$$\ddot{z} + n^2z = 0 \tag{53}$$

V. Task 4: Comparison of Keplerian Relative Motion and Non-Keplerian Descriptions

Given that we have just derived the Clohessy-Wiltshire (CW) equations, we are intimately familiar with the assumptions made about the chief orbit. It must be circular and Keplerian. Additionally, with the truncations and approximations made, the relative distance between deputy and chief should be much smaller than the position vector to the chief itself. This gives us room to compare accuracy of the CW and NL methods with and without perturbations. Clearly, the most interesting things to vary here are eccentricity and any orbital element which will increase the relative distance or subject the vehicles to higher disturbances sufficiently.

A. Varying Eccentricity - Keplerian Case for CW and Nonlinear (NL) Relative Equations

Here, let us vary the eccentricity. The chief has an initial state vector of $[a\ e\ i\ \Omega\ \omega\ M_0] = [10000km\ \ 0.1\ \ 51.6\ \ 0\ \ 0\ \ 0]$ where the deputy has $\delta oe = oe_d - oe_c = [0\ \ 0.0001\ \ 0\ \ 0\ \ 0\ \ 0.00001]$. This case represents an eccentric lead-follower situation.

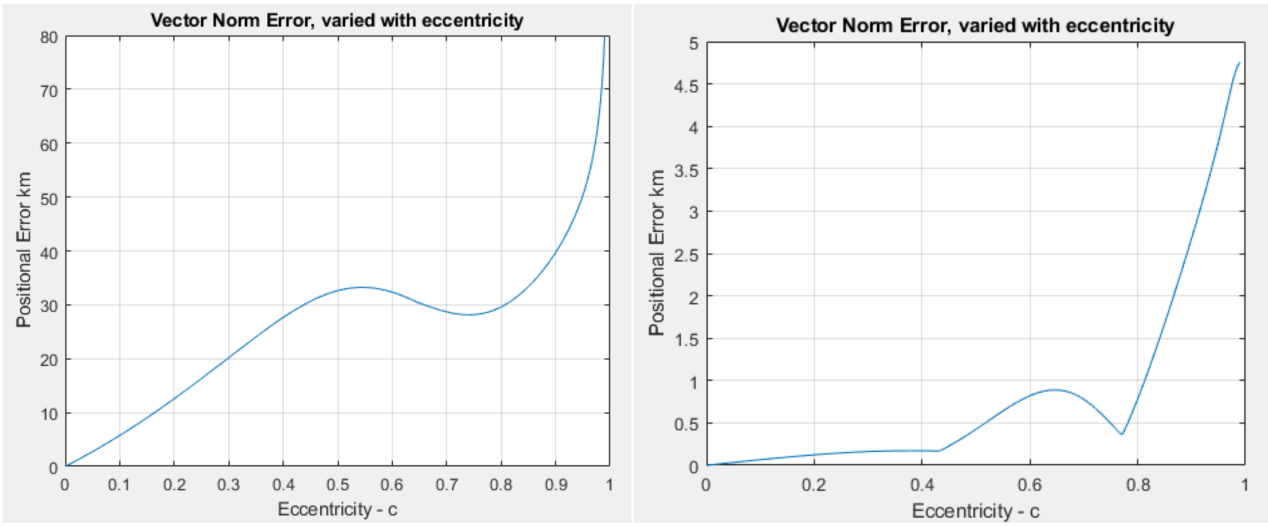


(a) Hill frame of NL and CW motion

(b) CW error wrt the Nonlinear equations.

Fig. 4 Notice the z error is small, but the y error is nearly 40km by the end of the integration.

Clearly the y error in 4 gros quite large very quickly. Obviously this is due to the violation of our circular chief assumption. But let's look closer. We see that CW error is highly sensitive, not only to eccentric chiefs, but to eccentricity differences between chief and deputy as seen in 5. Curiously, we also see that the error is not necessarily monotonic. The plots shown in figure 5 differ from 4 in that their $[i\ \Omega\ \omega\ M_0] = [45\ \ 20\ \ 30\ \ 20]deg$ to observe this behaviour.



(a) Chief eccentricity sweep with deputy 0.0001 ahead

(b) Eccentricities of deputy and chief equally swept

Fig. 5 The left most plot is the error of CW equations when the chief eccentricity was sweeped from 0 to 1. The deputy was always off by 0.0001 in eccentricity. The right most plot is the same sweep, but with both at equal eccentricities.

Clearly, in the event that the CW equations are used in an eccentric case, we must make sure that the difference in eccentricity is as close to zero. When the two spacecraft are offset in eccentricities by a small amount, we see an order of magnitude higher error.

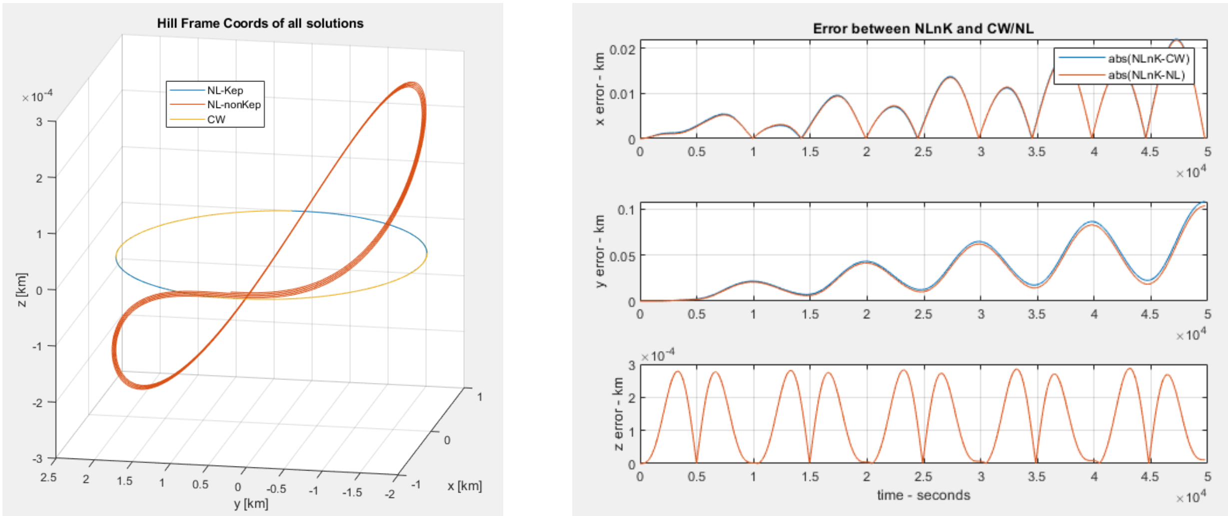
B. Circular Chief with Perturbations

In the case where the chief is circular, the nonlinear and CW equations of motion have essentially the same error related to the nonlinear EOM with perturbing forces. The figures 8 show the Hill frame motion of the CW, nonlinear, and nonlinear with perturbations as well as their errors compared to truth. So we can expect to be as accurate as our nonlinear hillframe EOMs when the chief is circular. Our out of plane disturbances (J2 in this case) account for 0.1 meters of z error, which is likely small enough for most guidance applications. Usually in rendezvous operations, we would have a proximity sensor regardless. The initial state vector for the chief was $[a\ e\ i\ \Omega\ \omega\ M_0] = [10000km\ \ 0\ \ 51.6\ \ 0\ \ 0\ \ 0]$ where the deputy has $\delta oe = oe_d - oe_c = [0\ \ 0.0001\ \ 0\ \ 0\ \ 0\ \ 0.00001]$.

Recall that we must use our previous derivation 44 to reconstruct the Hill frame coordinates from our nonlinear perturbed rototranslational equations of motion. We use algorithm 2 for this.

Algorithm 2 Converting output of Nonlinear Perturbed Equations to Hill-frame Coordinates

for i = 1:N **do**
  $[HN] = Cart2Hill([r_{c_i}\ \ v_{c_i}])$
  $h = \left\| [r_{c_i}^{\times}] v_{c_i} \right\|$
  $ah = [HN] * \boldsymbol{a}_{d_i}$
  ${}^{H}\boldsymbol{\omega}_{H/N} = [HN]\frac{[r_{c_i}^{\times}]\boldsymbol{v}_{c_i}}{\left\|\boldsymbol{r}_{\boldsymbol{c}_i}\right\|^2} + \frac{[HN]\boldsymbol{r}_{\boldsymbol{c}_i}}{h} a_d(3)$
  $\boldsymbol{\rho}(:, i) = [HN](r_{d_i} - r_{c_i})$
  $\dot{\boldsymbol{\rho}}(:, i) = [HN](v_{d_i} - v_{c_i}) - [{}^{H}\boldsymbol{\omega}_{H/N}^{\times}]\boldsymbol{\rho}(:, i)$
end for



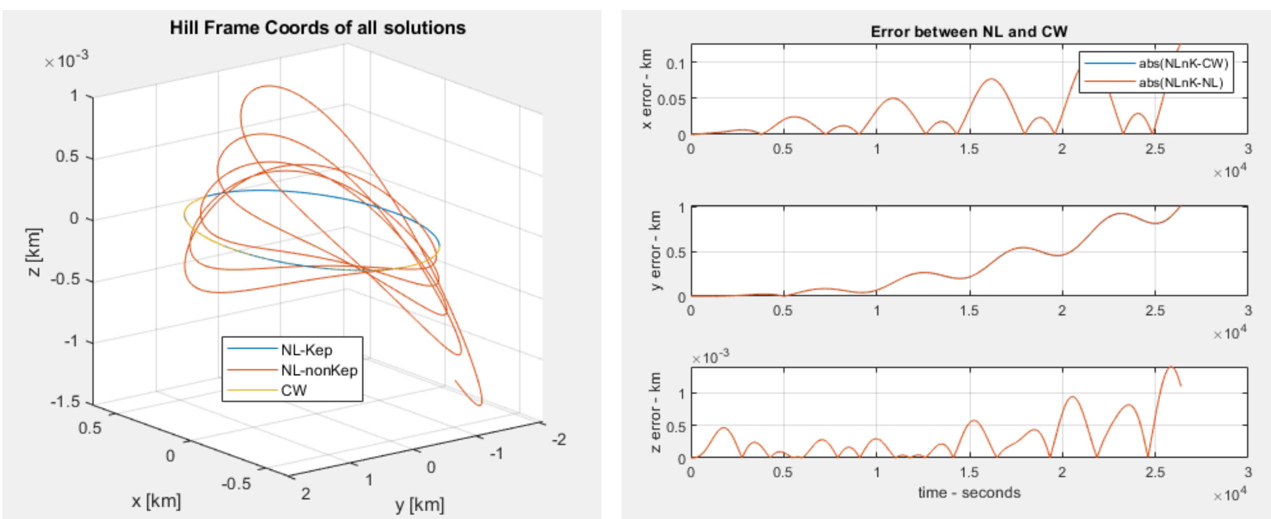
(a) Hill frame coordinates of the NL perturbed, NL unperturbed, and CW equation solutions.

(b) Error of the NL unperturbed and CW solutions wrt the NL perturbed solutions.

Fig. 6 Notice in this case that the error function of the CW linear dynamics does not depart from the nonlinear unperturbed dynamics, making it an attractive candidate for optimization in these scenarios.

C. Circular Chief with Perturbations - higher atmospheric drag effect

Given that we just saw some of the affect of J2, let us get a close look at how the atmospheric drag disturbance affects the Hill frame motion. We see, unsurprisingly, that the x and y Hill frame coordinates of the formation get spread out as the atmospheric drag spreads the satellites out. Here we change our altitude to 180km with the other orbital elements remaining the same from the previous example.



(a) Hill frame coordinates of the NL perturbed, NL unperturbed, and CW equation solutions.

(b) Error of the NL unperturbed and CW solutions wrt the NL perturbed solutions.

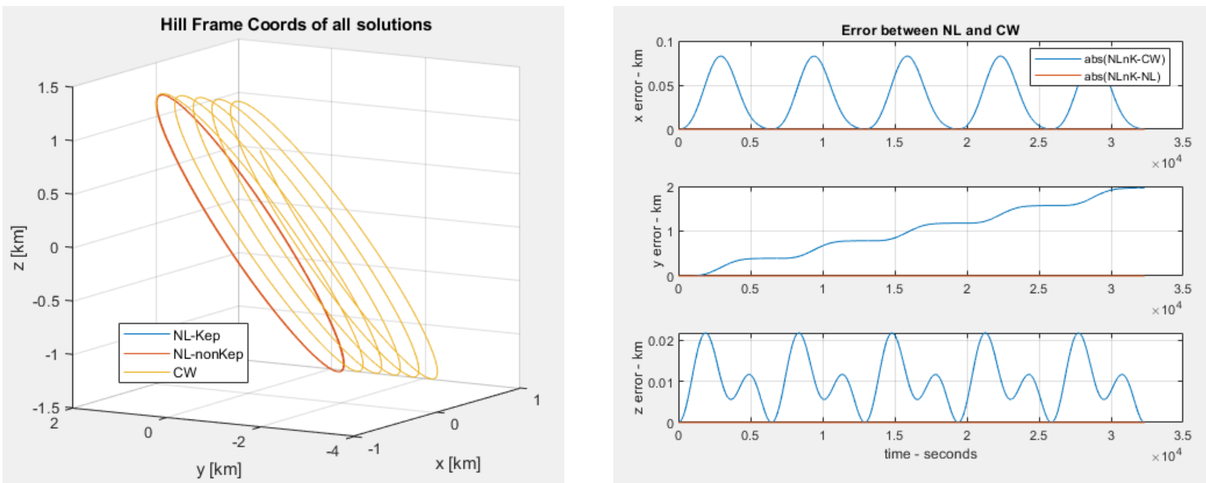
Fig. 7 Error grows as the atmospheric drag disturbance spreads out each satellite.

VI. Task 5: Passive Inspection Criteria for Formations

In order for a simple task like passive inspection to occur, we need a closed Hill frame relative orbit. We shall use the results of the derivations found in [1] and apply them to the nonlinear system with perturbations to see what kind of station keeping will be required if those methods of design were chosen. This answers the robustness question.

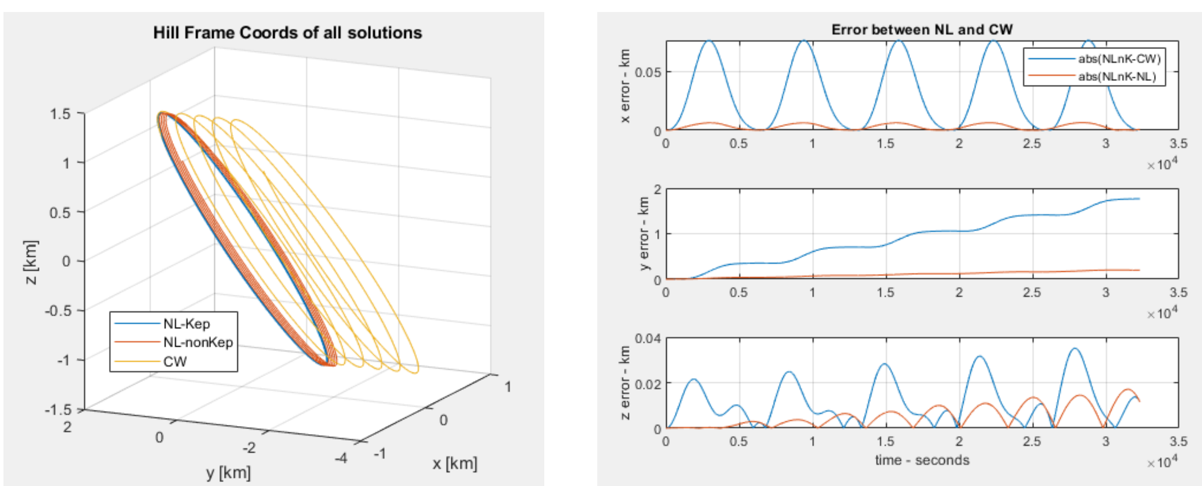
A bounded relative orbit is an orbit that repeats after each chief orbit. Their period needs to be the same, otherwise they will slowly drift apart over time and the orbit will be unbounded. We can start by using the result of a derivation in [1] where

$$\frac{\dot{y}_0}{x_0} = \frac{-n(2+e)}{\sqrt{(1+e)(1-e)^3}} \tag{54}$$



(a) Hill frame of solutions. NL and NL-nonKep are on top of each other, no perturbations added. **(b) Error of the CW case versus the unperturbed NL case.**

Fig. 8 These plots show the implemented bounded law under no perturbation. The nonlinear dynamics show the Hill frame to be closed and bounded, despite the CW equations drifting.



(a) Hill frame of solutions with NL perturbed. **(b) Error of the CW and NL case versus the perturbed NL truth dynamics.**

Fig. 9 Notice how the true perturbed NL dynamics drifts with time.

We see in 9 that despite this closed orbit derivation working for the nonlinear dynamics, it starts to drift under perturbations. Therefore station keeping maneuvers are needed to remain on this closed orbit, but likely less compared to other initial conditions for the orbit. The initial conditions used for this experiment can be found in the MATLAB code attached at the bottom. This drift is a product of the J2 oblate affect of the Earth. The altitude used in these initial conditions was too high for atmospheric drag to appreciably affect the Hill frame trajectories.

VII. Conclusions and Future Work

Overall, I learned a lot from this project in performing extra derivations not included in our textbook as well as performing Hill frame simulations to give me a feel for their behaviour under exogenous affects. In the future, much more work needs to go into the class structure for the spacecraft and formation classes. This includes the ability to hook up actuator physics and other relevant systems. I would also like to make an interactive simulator with live playback of trajectories. I think this is a good starting point for what I would like to work on for the second project of this course.

References

[1] Schaub, H., and Junkins, J. L., *Analytical mechanics of space systems*, 4^{th} ed., American Institute of Aeronautics and Astronautics, Inc, Reston, Virginia, 2018.

[2] Khalil, I., and Samwel, S., "Effect of Air Drag Force on Low Earth Orbit Satellites during Maximum and Minimum Solar Activity," *Space Research Journal*, Vol. 9, No. 1, 2016, pp. 1–9.

[3] Markley, F. L., and Crassidis, J. L., *Fundamentals of spacecraft attitude determination and control*, Vol. 33, Springer, 2014.

Project Code

Listing 1 Main Script (Simulation Framework)

```
%% Padraig Lysandrou ASEN6014 Project 1
clc; close all; clear all;

%% Begin the beguine
mu =  3.986004415e+05;

% orbital elements of chief
oe_c.a       = 6371 + 170;
oe_c.e       = 0;
oe_c.inc     = deg2rad(51.7);
oe_c.Omega   = deg2rad(0);
oe_c.omega   = deg2rad(0);
oe_c.Mo      = deg2rad(0);

% orbital elements of deputy
oe_d.a       = oe_c.a;
oe_d.e       = oe_c.e + 0.0001;
oe_d.inc     = oe_c.inc;
oe_d.Omega   = oe_c.Omega;
oe_d.omega   = oe_c.omega;
oe_d.Mo      = oe_c.Mo  + 0.00001;

% Plan out the timing
orbits = 5;
T = 2*pi*sqrt((oe_c.a^3)/mu);
T_tot = orbits*T;
dt= 10;
time = 0:dt:T_tot;
npoints = length(time);

% Insantiate a spacecraft object, set some parameters
chief = spacecraft(oe_c, 'oe');
deputy= spacecraft(oe_d, 'oe');

% Turn on our pertubations
chief.p.perturb_flag    = 1;
deputy.p.perturb_flag   = 1;

% determine other initial conditions
sigma_0_c = C2MRP(angle2dcm(0,-pi/1.9,0)*Cart2Hill(chief.rv'));
sigma_0_d = C2MRP(angle2dcm(0,-pi/1.9,0)*Cart2Hill(deputy.rv'));
omega_0 = [2*pi/T+0.0005 -0.0001 0.00].';
m_0 = 100;
r= 0.5; h= 200;
Ic = diag([(1/12)*(m_0*(3*r*r + h*h)) (1/12)*(m_0*(3*r*r + h*h)) (m_0*r*r)/2]);
chief.Ic = Ic;
deputy.Ic= Ic;

% initial condition vectors for both spacecraft
state_initial_c = [m_0 chief.rv.' sigma_0_c.' omega_0.'].';
% chief.state_vector = state_initial_c;
state_initial_d = [m_0 deputy.rv.' sigma_0_d.' omega_0.'].';
% deputy.state_vector = state_initial_d;


% compute the stability constants
Kr = (chief.Ic(2,2) - chief.Ic(1,1))/chief.Ic(3,3);
Ky = (chief.Ic(2,2) - chief.Ic(3,3))/chief.Ic(1,1);


%% INTEGRATE THE 6DOF MODEL
%
% Integrate chief and deputy nonlinear rototranslational dynamics
```

```
[timerk4, chiefstate] = chief.integrate_dynamics_rk4(time, state_initial_c);
[timerk4, deputystate] = deputy.integrate_dynamics_rk4(time, state_initial_d);

% plot_all_the_things(timerk4, chiefstate.');
% plot_all_the_things(timerk4, deputystate.');
plot_two_sats(timerk4, chiefstate.', deputystate.')


% ode45 integration scheme
% [time3, state_out3] = chief.integrate_dynamics(time, state_initial_c);
% plot_all_the_things(time3, state_out3);
% [time3, state_out3] = deputy.integrate_dynamics(time, state_initial_d);
% plot_all_the_things(time3, state_out3);
%}


%% COMPARING NL AND CW DYNAMICS WITH NO PERTURBATIONS
% get the chief orbital elements for the cw equations
p = chief.p;
p.oe_c = chief.oe;
rc = chief.rv(1:3);
vc = chief.rv(4:6);
rd = deputy.rv(1:3);
vd = deputy.rv(4:6);

% calculate integration conditions for relative orbits
f_c = (norm(chief.oe.h))/(norm(rc)^2);
N_omega_ON = [0; 0; f_c];
HN = Cart2Hill(chief.rv.');
rho_Hill_CW = HN*(rd - rc);
rhod_Hill_CW= HN*(vd - vc) - skew(N_omega_ON)*rho_Hill_CW;
relstate_0 = [rho_Hill_CW.' rhod_Hill_CW.'].';

%
rc0 = norm(rc);
rcd0 = (rc.'*vc)/rc0;       % dot the v vector from chief with or_hat
init_conds = [rho_Hill_CW.' rhod_Hill_CW.' rc0 rcd0];


% Clohessy Wiltshire Equations
f_dot = @(t_in, state_in, p) cw_hill_ode(t_in, state_in, p);
[timeCW, state_out1] = general_rk4(f_dot, timerk4, relstate_0, p);


% Nonlinear relative dynamics
f_dot = @(t_in, state_in, p) nl_rel_ode(t_in, state_in, p);
[timeNL, state_out2] = general_rk4(f_dot, timerk4, init_conds, p);

% plot dudes one top of each other
% figure; plot3(state_out2(1,:), state_out2(2,:), state_out2(3,:));
% grid on; axis square; hold on;
% plot3(state_out1(1,:), state_out1(2,:), state_out1(3,:));
% grid on; axis square
% title('Hill Frame Coords of CW and NL EOM (both Keplerian)')
% legend('NL','CW')
%}

% figure;
% subplot(3,1,1)
% plot(timerk4, state_out2(1,:)-state_out1(1,:));grid on;
% title('error between CW and NL relative ODEs (both Keplerian)')
% subplot(3,1,2)
% plot(timerk4, state_out2(2,:)-state_out1(2,:));grid on;
% subplot(3,1,3)
% plot(timerk4, state_out2(3,:)-state_out1(3,:));
% grid on;


```

```
%% PLOT THE BORESIGHT VECTOR OVER TIME
%{
state_out = chiefstate.';
sigma_BN =  state_out(:, 8:10);
x_N =       state_out(:, 2:4);
v_N =       state_out(:, 5:7);

b = zeros(3,length(time));
for i = 1:length(time)
    NB = MRP2C(sigma_BN(i,:).').';
    HN = Cart2Hill([x_N(i,:) v_N(i,:)]);
    b(:,i) = HN*NB*([0 0 1].');
end

figure;
plot3(0:.01:1,zeros(1,101),zeros(1,101),'Color',[0.8500 0.3250 0.0980]);
hold on; grid on;
plot3(zeros(1,101),0:.01:1,zeros(1,101),'Color',[0.8500 0.3250 0.0980]);
hold on;
plot3(zeros(1,101),zeros(1,101),0:.01:1,'Color',[0.8500 0.3250 0.0980]);
hold on;
plot3(b(1,1),b(2,1),b(3,1),'ro');
hold on; axis equal
for i = 1:100:length(time)
    plot3(b(1,i),b(2,i),b(3,i),'-ob','MarkerSize',1,'MarkerFaceColor','r');
    axis([-1.1 1.1 -1.1 1.1 -1.1 1.1]);
    hold on;
%     pause(2000    00000000000*eps)
    pause(eps)
end
xlabel('Or (radial)');
ylabel('Otheta (local tangent)');
zlabel('Oh (radial normal)');
title('Hill frame boresight vector over time')
%}



%% Compare CW equations (keplerian) to the perturbed rototranslational
% in the hill frame!

% compute the whole time history for perturbations one the chief
p_exo = chief.perturbing(timerk4, chiefstate);
rho_Hill = zeros(3, length(timerk4));
rhod_Hill= zeros(3, length(timerk4));

for k = 1:length(timerk4)
    % pull out the current position vectors
    rc = chiefstate(2:4, k);
    vc = chiefstate(5:7, k);
    rd = deputystate(2:4, k);
    vd = deputystate(5:7, k);

    %  Must calculate a customized, perturbed omegaH/N vector here:
    HN = Cart2Hill([rc.' vc.']);
    h = norm(skew(rc)*vc);
    ah = HN*p_exo.a(:,k);
    H_omega_ON = HN*((skew(rc)*vc)/(norm(rc)^2)) + (HN*rc/h)*ah(3);
    rho_Hill(:,k) = HN*(rd - rc);
    rhod_Hill(:,k)= HN*(vd - vc) - skew(H_omega_ON)*rho_Hill(:,k);
end


figure;
subplot(3,1,1)
plot(timerk4, abs(rho_Hill(1,:)- state_out1(1,:))); grid on; hold on;
plot(timerk4, abs(rho_Hill(1,:)- state_out2(1,:))); grid on; hold on;
```

```
title('Error between NL and CW')
% legend('abs(NLnK-CW)')
legend('abs(NLnK-CW)','abs(NLnK-NL)')
ylabel('x error - km')
subplot(3,1,2)
plot(timerk4, abs(rho_Hill(2,:)- state_out1(2,:))); grid on; hold on;
plot(timerk4, abs(rho_Hill(2,:)- state_out2(2,:))); grid on; hold on;
ylabel('y error - km')
subplot(3,1,3)
plot(timerk4, abs(rho_Hill(3,:)- state_out1(3,:))); grid on; hold on;
plot(timerk4, abs(rho_Hill(3,:)- state_out2(3,:))); grid on; hold on;
ylabel('z error - km')
xlabel('time - seconds')
grid on;



% figure; plot3(rho_Hill(1,:), rho_Hill(2,:), rho_Hill(3,:));
% grid on; axis square; hold on;
% plot3(state_out1(1,:), state_out1(2,:), state_out1(3,:));
% grid on; axis square
% title('Hill Frame Coords of CW and NL (non-Kep)')
% legend('NL','CW')


figure; plot3(state_out2(1,:), state_out2(2,:), state_out2(3,:));
grid on; axis square; hold on;
plot3(rho_Hill(1,:), rho_Hill(2,:), rho_Hill(3,:));
plot3(state_out1(1,:), state_out1(2,:), state_out1(3,:));
grid on; axis square
title('Hill Frame Coords of all solutions')
legend('NL-Kep','NL-nonKep','CW')
xlabel('x [km]')
ylabel('y [km]')
zlabel('z [km]')
```

Listing 2 Spacecraft Class

```
classdef spacecraft
    %spacecraft generalized spacecraft class
    %   Ideally can be used for launch vehicles and satellites

    properties
        p       % constants for dynamics, etc
        rv      % INITIAL position and velocity of vehicle
        oe      % INITIAL orbital elements of vehicle
        state_vector % udpated in integration
        Ic      % Inertia matrix of sysem, may change with stage seps.
        n = 13; % state vector dimension
    end

    methods
        function self = spacecraft(state, type, params)
            %spacecraft Constructor for this class
            %  state must be 1) [r;v], 2) an oe struct, 3) a full 13x1
            %  state vector [m r v mrp omega]

            if ¬exist('params', 'var')
                self.p = struct();
            else
                self.p = params;
            end

            % Check for required value
            if ¬isfield(self.p, 'mu')
                self.p.mu = 3.986004415e+05; %km3s?2
```

```
            end
            if ¬isfield(self.p, 'mu_m')
                self.p.mu_m = 3.986004418E14; %m3s?2
            end
            if ¬isfield(self.p, 'j2')
                self.p.j2 = 0.0010826269;
            end
            if ¬isfield(self.p, 'Re')
                self.p.Re = 6371; % km
            end
            if ¬isfield(self.p, 'control_flag')
                self.p.control_flag = 0;
            end
            if ¬isfield(self.p, 'perturb_flag')
                self.p.perturb_flag = 0;
            end


            % compute oe2rv and state vector as needed
            if strcmp(type, 'oe')
                p_out = schaub_elements(state, self.p,0,2,1);
                self.rv = [p_out.r; p_out.v];
                self.state_vector = zeros(13,1);
                self.state_vector(2:7) = self.rv;
                % computes more interesting values and replaces...
                p_out = schaub_elements(self.rv,self.p,0,1,2);
                self.oe = state;
                self.oe.h = p_out.h;
                self.oe.E = p_out.E;
                self.oe.nu= p_out.nu;
            elseif strcmp(type, 'rv')
                self.oe = schaub_elements(state,self.p,0,1,2);
                self.rv = state;
                self.state_vector = zeros(13,1);
                self.state_vector(2:7) = self.rv;
            elseif strcmp(type, 'statevector')
                self.rv = state(2:7);
                self.oe = schaub_elements(self.rv,self.p,0,1,2);
                self.state_vector = state;
            end

        end

        %% helper functions, etc
        function update_inertia(self, Ic_new)
            % other operations here.
            self.Ic = Ic_new*(self.p.mu/self.p.mu);
        end

        % just incase the internal state needs to be updated
        function update_state(self, state, type)
            if strcmp(type, 'oe')
                self.oe = state;
            elseif strcmp(type, 'rv')
                self.rv = state;
            end
        end

        % J2 Effect from oblate spheroid
        function a_J2 = J2(self, x_N)
            r1 = norm(x_N);
            j2_const = (-3/2)* ...
                ((self.p.mu*self.p.j2*(self.p.Re)^2)/(r1^4));
            x = x_N(1,:);
            y = x_N(2,:);
            z = x_N(3,:);
            a_J2 = j2_const* ...
               [(1-(5*((z^2)/(r1^2))))*(x/r1); ...
```

```
                (1-(5*((z^2)/(r1^2))))*(y/r1); ...
                (3-(5*((z^2)/(r1^2))))*(z/r1)];
            if r1-self.p.Re < 0
                a_J2 = [0 0 0].';
            end
        end

        function a_exo = atmo_drag_a(self, x_N, v_N, sigma_BN, m)
            % constants
            omega_E = [0 0 7.29211505392569e-05].';
                % equatorial rotation of earth
            H = 7.250;                      % km
            rho_naught = 1.225 * (1000^3);  % kg/km^3
            A = 1/(1000^2);                 % km^2

            % make this dependant upon the MRP at some point
            alpha = pi/2;
            r = norm(x_N);
            vatm = v_N - skew((self.p.Re/r)*omega_E)*x_N;
            % taking care if singular cases
            if r-self.p.Re ≥ 0
                rho = rho_naught*exp(-(r-self.p.Re)/H);
            else
                rho = 0;
            end
            Cd = 2*(sin(alpha)^3);
            v = norm(vatm);
            v_hat = vatm./v;
            q = (rho*v*v)/2;
            a_exo = -(q*Cd*A*v_hat)/m;
        end

        function tau_exo = atmo_drag_tau(self)
            tau_exo = [0 0 0].'.*self.p.mu;
        end

        % Gravity Gradient Torque, in body frame
        function tau_exo = grav_grad_tau(self, x_N, sigma_BN)
            BN = MRP2C(sigma_BN);
            x_B = BN*(x_N.*1000); % must convert to meters
            rc = norm(x_B);
            tau_exo = (((3*self.p.mu_m)/(rc^5))* ...
                (skew(x_B)*(self.Ic*x_B)));
        end

        % find the perturbing accelerations and torques for post procesing
        function p_exo = perturbing(self, time, state)
            p_exo.tau = zeros(3,length(time));
            p_exo.a   = zeros(3, length(time));
            if self.p.perturb_flag
                % Takes a time history of rototranslational state and time and
                % converts to a time history of expected perturbations
                % instantaneously at these times

                for i = 1: length(time)
                    state_in = state(:,i);
                    m =          state_in(1);
                    x_N =        state_in(2:4);
                    v_N =        state_in(5:7);
                    sigma_BN =   state_in(8:10);
                    p_exo.tau(:,i) = grav_grad_tau(self, x_N, sigma_BN);
                    p_exo.a(:,i) = J2(self, x_N) ...
                        + atmo_drag_a(self, x_N, v_N, sigma_BN, m);
                end
            else
                return
            end
        end
```

```matlab

        %% Dynamics function: can activate controls or peturbations
        function f_x = dynamics(self, t, state_in)

            % Should I make the control compute an external function?
            if self.p.control_flag
                u = compute_control(self, t, state_in);
            else
                u.tau = [0 0 0].';
                u.a = [0 0 0].';
                u.mdot = 0;
            end

            % distribute the states for use! these are row vectors.
            m =         state_in(1);
            x_N =       state_in(2:4);
            v_N =       state_in(5:7);
            sigma_BN =  state_in(8:10);
            omega_BN =  state_in(11:13);

            % Check for pertubation flag, and compute those
            if self.p.perturb_flag
                % Compute exogenous torques and accelerations
                % still need to add attitude torques from drag effects
                tau_exo = grav_grad_tau(self, x_N, sigma_BN);
                a_exo = J2(self, x_N) ...
                    + atmo_drag_a(self, x_N, v_N, sigma_BN, m);
            else
                tau_exo = [0 0 0].';
                a_exo = [0 0 0].';
            end

            % Mass depletion dynamics (kg/s)
            m_dot = -u.mdot;
            % Inertial velocity (km/s)
            x_dot = v_N;
            % Inertial acceleration with input acceleration and perturbs
            v_dot = -(self.p.mu/(norm(x_N)^3)).*x_N ...
                + a_exo ...
                + u.a;

            % MRP integration (rad/s for angular rates)
            sigma_dot = (0.25.*((1 -(sigma_BN.'*sigma_BN))*eye(3) ...
                + 2*skew(sigma_BN) ...
                +(2*sigma_BN*(sigma_BN.'))))*omega_BN;

            % Angular Rate Integration with input torques
            omega_dot = self.Ic\((-skew(omega_BN)*(self.Ic*omega_BN)) ...
                + tau_exo ...
                + u.tau);

            % output the derivative
            f_x = [m_dot x_dot.' v_dot.' sigma_dot.' omega_dot.'].';

        end


        %% ODE45 function call
        function [time, state_out] = integrate_dynamics(self, time, X_0, odesettings)
            if ¬exist('odesettings', 'var')
                odesettings = odeset('AbsTol', 1E-12, 'RelTol', 1E-12);
            end
            f_dot = @(t_in, state_in) self.dynamics(t_in, state_in);
            tic
            [time, state_out] = ode45(f_dot, time, X_0, odesettings);
            toc
        end

```

```
        %% Rk4 integrator call to the 6DoF dynamics with control
        function [time, state_out] = integrate_dynamics_rk4(self, time, X_0)
            f_dot = @(t_in, state_in) self.dynamics(t_in, state_in);
            dt = time(2) - time(1);
            npoints = length(time);
            state_out = zeros(self.n, npoints);
            state_out(:,1) = X_0;
            tic
            for i = 1:npoints-1
                k_1 = f_dot(time(i), state_out(:,i));
                k_2 = f_dot(time(i)+0.5*dt, state_out(:,i)+0.5*dt*k_1);
                k_3 = f_dot((time(i)+0.5*dt), (state_out(:,i)+0.5*dt*k_2));
                k_4 = f_dot((time(i)+dt), (state_out(:,i)+k_3*dt));
                state_out(:,i+1) = state_out(:,i) + (1/6)*(k_1+(2*k_2)+(2*k_3)+k_4)*dt;
                s = norm(state_out(8:10, i+1));
                if s > 1
                    state_out(8:10,i+1) = -(state_out(8:10,i+1) ./(s^2));
                end
            end
            toc
        end
    end
end
```

Listing 3 Nonlinear Relative ODEs

```
function dxdt = nl_rel_ode(¬, X, p)
%nl_rel_ode List of the noninear relative motion differential equations
%   State vector to be integrated to understand the relative motion of two
%   satellite (one chief and one deputy)

    % pull out constants
    mu = p.mu;
    oe = p.oe_c; % orbital elements of the chief
    rectum = oe.a*(1 - (oe.e)^2);
    h = sqrt(rectum * mu);

    % pull out states
    x = X(1);
    y = X(2);
    z = X(3);
    xd= X(4);
    yd= X(5);
    zd= X(6);
    rc= X(7);
    rcd=X(8);

    % intermediate variables
    fd = h/(rc^2);
    rd = norm([(rc+x), y, z]);

    % ODEs to solve
    xdd = 2*fd*(yd - (y*(rcd/rc))) + (x*fd*fd) + (mu/(rc*rc)) + ...
        - ((mu/(rd^3))*(rc + x));
    ydd = -2*fd*(xd - (x*(rcd/rc))) + (y*fd*fd) - ((mu/(rd^3))*y);
    zdd = -((mu/(rd^3))*z);
    rcdd = (rc*fd*fd)*(1-(rc/rectum));

    dxdt =[xd; yd; zd; xdd; ydd; zdd; rcd; rcdd];
end
```

Listing 4 CW ODEs

```
function dxdt = cw_hill_ode(¬, X, p)
%cw_hill_ode Linear Clohessy Wiltshire equations in the hill frame
```

```
% CW equations assume circular orbit for the chief
    % pull out constants
    oe = p.oe_c;
    n = sqrt(p.mu/(oe.a^3));

    % pull out states
    x   = X(1);
    y   = X(2);
    z   = X(3);
    xd  = X(4);
    yd  = X(5);
    zd  = X(6);


    % ODEs to solve (all linear...)
    xdd = (2*n*yd) + (3*n*n*x);
    ydd = (-2*n*xd);
    zdd = (-n*n*z);

    dxdt =[xd; yd; zd; xdd; ydd; zdd];
end
```

Listing 5 Cart2Hill

```
function [HN] = Cart2Hill(rv)
%Cart2Hill converts a set of R and V vectors into a hill frame DCM
%   hill frame order defined as [or otheta oh]

    % decompose
    rc = rv(1:3).';
    vc = rv(4:6).';

    hvec = skew(rc)*vc;
    hhat = hvec./norm(hvec);
    o_r = rc./norm(rc);
    o_h = hhat;
    o_theta = skew(o_h)*o_r;
    HN = [o_r.'; o_theta.'; o_h.'];
end
```

Listing 6 RV2OE and OE2RV Utility

```

function p_out = schaub_elements(state, p,Δ_t,inputFlag,outputFlag)
%schaub_elements performs operations: rv2oe and oe2rv for keplerian
%elements and other trashy routines
% state must be a [r v] for [1, 2] flags, and oe struct for [2,1] flags
% todo: [1,1] flags being integate rv from t1 to t2, same for [2,2]

if inputFlag == 1 && outputFlag == 2
    % compute the RV2OE transform
    % first extract the important parameters
    mu = p.mu;
    r = state(1:3);
    v = state(4:6);

    rhat = r./norm(r);
    h = cross(r,v);
    hhat = h./norm(h);
    e = (1/mu).*((v.'*v - (mu./norm(r))).*r - (r.'*v).*v);
    ehat = e./norm(e);
    ehat_p = cross([0 0 1],ehat).';
    p = norm(h)^2/mu;
    energy = (0.5*norm(v)^2) - (mu/norm(r));
    inc = acos(h(3)/norm(h));
```

```
    zhat = [0 0 1].';
    nhat_O = (cross(zhat,hhat))/(norm(cross(zhat,hhat)));
    Omega = atan2(dot([0 1 0].',nhat_O),dot([1 0 0].',nhat_O));
    if inc == 0
        Omega = 0;
    end
    nhat_Op = cross(hhat,nhat_O);
    omega = atan2(dot(e,nhat_Op),dot(e,nhat_O));
    a = p/(1-norm(e)^2);
    % calculate the initial true anomaly
    f0 = atan2(dot(r,ehat_p),dot(r,ehat));
    T = 2*pi*sqrt(abs(a^3)/mu);
    n = (2*pi)/T;

    % package up interesting parameters
    oe.h = h;
    oe.a = a;
    oe.e = norm(e);
    oe.inc = inc;
    oe.omega = omega;
    oe.Omega = Omega;
    oe.p = p;
    oe.f0 = f0;
    oe.T = T;
    oe.n = n;
    oe.energy = energy;
    % calculate the actual true anomaly
%     f0 = atan((cross(ehat, rhat).'*hhat)/(ehat.'*rhat));
    if norm(e) ≤ 1
       E = 2*atan(sqrt((1-norm(e))/(1+norm(e)))*tan(f0/2));
       M = E - (norm(e)*sin(E));
    elseif norm(e) > 1
       H = 2*atanh(sqrt((norm(e)-1)/(1+norm(e)))*tan(f0/2));
       M = (norm(e)*sinh(H) - H);
    end
    if M < 0
        M = M + 2*pi;
    elseif M > 2*pi
        M = mod(M, 2*pi);
    end
    oe.Mo = M - (sqrt(mu/abs(a^3))*Δ_t);
    [oe.nu, Ecc_anom] = findE(Δ_t,norm(e), sqrt(mu/abs(a^3)), oe.Mo);
    oe.E = Ecc_anom;
    p_out = oe;


elseif inputFlag == 2 && outputFlag == 1
    % compute the OE2RV transform
    mu = p.mu;
    oe = state;
    a = oe.a;
    e = norm(oe.e);
    h = sqrt(mu*a*(1-(e^2)));
    omega = oe.omega;
    Omega = oe.Omega;
    inc = oe.inc;
    n = sqrt(mu/abs(a^3));
    [nu, Ecc_anom] = findE(Δ_t, e, n, oe.Mo);

    rp = (h^2/mu)*(1/(1 + e*cos(nu)))*(cos(nu)*[1;0;0] + sin(nu)*[0;1;0]);
    vp = (mu/h)*(-sin(nu)*[1;0;0]+(e + cos(nu))*[0;1;0]);

    % R3_omega, DCM rotation about z, omega amount
    % R3_Omega, DCM rotation about the z, Omega amount
    % R1_inc,  DCM rotation about the x, inc amount
    % i wonder if anyone has written a quaternion form for this
    R3_omega = [ cos(omega)  sin(omega)   0
                -sin(omega)  cos(omega)   0
```

```
                  0        0      1];
    R3_Omega = [ cos(Omega)  sin(Omega)  0
                -sin(Omega)  cos(Omega)  0
                  0        0      1];
    R1_inc = [1        0          0
                  0    cos(inc)  sin(inc)
                  0   -sin(inc)  cos(inc)];

    % total rotations SO3 group
    perif_to_ECI = (R3_omega*R1_inc*R3_Omega).';
    r = perif_to_ECI*rp;
    v = perif_to_ECI*vp;

    % package up and send out
    p_out.E = Ecc_anom;
    p_out.r = r;
    p_out.v = v;
    p_out.nu = nu;
    p_out.h = h;
else

end


end
```

Listing 7 Bounded Hill Orbit Test Script

```
%% Padraig Lysandrou ASEN6014 Project 1
clc; close all; clear all;

%% Begin the beguine
mu =  3.986004415e+05;

% orbital elements of chief
oe_c.a          = 7500;
oe_c.e          = 0.01;
oe_c.inc        = deg2rad(45);
oe_c.Omega      = deg2rad(20);
oe_c.omega      = deg2rad(30);
oe_c.Mo         = deg2rad(20);
oe_d.a          = 7500;
oe_d.e          = oe_c.e + 0.0001;
oe_d.inc        = deg2rad(45 + 0.01);
oe_d.Omega      = deg2rad(20);
oe_d.omega      = deg2rad(30 );
oe_d.Mo         = deg2rad(20);

n_c = sqrt(mu/(oe_c.a^3));
% constraint function for bounded relative orbits
yd0_over_x0 = (-n_c*(2+oe_c.e))/sqrt((1+oe_c.e)*((1-oe_c.e)^3));

% Plan out the timing
orbits = 5;
T = 2*pi*sqrt((oe_c.a^3)/mu);
T_tot = orbits*T;
dt= 10;
time = 0:dt:T_tot;
npoints = length(time);

% Insantiate a spacecraft object, set some parameters
chief = spacecraft(oe_c, 'oe');
deputy= spacecraft(oe_d, 'oe');

rc = chief.rv(1:3);
rd = deputy.rv(1:3);
```

```
vc = chief.rv(4:6);
vd = deputy.rv(4:6);
% calculate the rho and rhodots
HN = Cart2Hill(chief.rv.');
f_c = (norm(chief.oe.h))/((norm(rc))^2);
H_omega_ON = [0; 0; f_c];
rho_Hill = HN*(rd - rc);
rhod_Hill = HN*(vd-vc) - skew(H_omega_ON)*rho_Hill;

thing = rhod_Hill(2)/yd0_over_x0;
rho_Hill(1) =  thing;
N_rho_Hill = (HN.')*rho_Hill;

rd = rc + N_rho_Hill;

% now recreate the shit we needed
chief = spacecraft(chief.rv, 'rv');
deputy= spacecraft([rd.' vd.'].', 'rv');


% Turn on our pertubations
chief.p.perturb_flag    = 1;
deputy.p.perturb_flag   = 1;

% determine other initial conditions
sigma_0_c = C2MRP(angle2dcm(0,-pi/1.9,0)*Cart2Hill(chief.rv'));
sigma_0_d = C2MRP(angle2dcm(0,-pi/1.9,0)*Cart2Hill(deputy.rv'));
omega_0 = [2*pi/T+0.0005 -0.0001 0.00].';
m_0 = 100;
r= 0.5; h= 200;
Ic = diag([(1/12)*(m_0*(3*r*r + h*h)) (1/12)*(m_0*(3*r*r + h*h)) (m_0*r*r)/2]);
chief.Ic = Ic;
deputy.Ic= Ic;

% initial condition vectors for both spacecraft
state_initial_c = [m_0 chief.rv.' sigma_0_c.' omega_0.'].';
state_initial_d = [m_0 deputy.rv.' sigma_0_d.' omega_0.'].';




%% INTEGRATE THE 6DOF MODEL
%
[timerk4, chiefstate] = chief.integrate_dynamics_rk4(time, state_initial_c);
[timerk4, deputystate] = deputy.integrate_dynamics_rk4(time, state_initial_d);

% plot_all_the_things(timerk4, chiefstate.');
% plot_all_the_things(timerk4, deputystate.');
plot_two_sats(timerk4, chiefstate.', deputystate.')


%% COMPARING NL AND CW DYNAMICS WITH NO PERTURBATIONS
% get the chief orbital elements for the cw equations
p = chief.p;
p.oe_c = chief.oe;
rc = chief.rv(1:3);
vc = chief.rv(4:6);
rd = deputy.rv(1:3);
vd = deputy.rv(4:6);

% calculate integration conditions for relative orbits
f_c = (norm(chief.oe.h))/(norm(rc)^2);
N_omega_ON = [0; 0; f_c];
HN = Cart2Hill(chief.rv.');
rho_Hill_CW = HN*(rd - rc);
rhod_Hill_CW= HN*(vd - vc) - skew(N_omega_ON)*rho_Hill_CW;
relstate_0 = [rho_Hill_CW.' rhod_Hill_CW.'].';

```

```
%
rc0 = norm(rc);
rcd0 = (rc.'*vc)/rc0;        % dot the v vector from chief with or_hat
init_conds = [rho_Hill_CW.' rhod_Hill_CW.' rc0 rcd0];


% Clohessy Wiltshire Equations
f_dot = @(t_in, state_in, p) cw_hill_ode(t_in, state_in, p);
[timeCW, state_out1] = general_rk4(f_dot, timerk4, relstate_0, p);


% Nonlinear relative dynamics
f_dot = @(t_in, state_in, p) nl_rel_ode(t_in, state_in, p);
[timeNL, state_out2] = general_rk4(f_dot, timerk4, init_conds, p);

% plot dudes one top of each other
% figure; plot3(state_out2(1,:), state_out2(2,:), state_out2(3,:));
% grid on; axis square; hold on;
% plot3(state_out1(1,:), state_out1(2,:), state_out1(3,:));
% grid on; axis square
% title('Hill Frame Coords of CW and NL EOM (both Keplerian)')
% legend('NL','CW')
%}

% figure;
% subplot(3,1,1)
% plot(timerk4, state_out2(1,:)-state_out1(1,:));grid on;
% title('error between CW and NL relative ODEs (both Keplerian)')
% subplot(3,1,2)
% plot(timerk4, state_out2(2,:)-state_out1(2,:));grid on;
% subplot(3,1,3)
% plot(timerk4, state_out2(3,:)-state_out1(3,:));
% grid on;



%% PLOT THE BORESIGHT VECTOR OVER TIME
%{
state_out = chiefstate.';
sigma_BN =  state_out(:, 8:10);
x_N =       state_out(:, 2:4);
v_N =       state_out(:, 5:7);

b = zeros(3,length(time));
for i = 1:length(time)
    NB = MRP2C(sigma_BN(i,:).').';
    HN = Cart2Hill([x_N(i,:) v_N(i,:)]);
    b(:,i) = HN*NB*([0 0 1].');
end

figure;
plot3(0:.01:1,zeros(1,101),zeros(1,101),'Color',[0.8500 0.3250 0.0980]);
hold on; grid on;
plot3(zeros(1,101),0:.01:1,zeros(1,101),'Color',[0.8500 0.3250 0.0980]);
hold on;
plot3(zeros(1,101),zeros(1,101),0:.01:1,'Color',[0.8500 0.3250 0.0980]);
hold on;
plot3(b(1,1),b(2,1),b(3,1),'ro');
hold on; axis equal
for i = 1:100:length(time)
    plot3(b(1,i),b(2,i),b(3,i),'-ob','MarkerSize',1,'MarkerFaceColor','r');
    axis([-1.1 1.1 -1.1 1.1 -1.1 1.1]);
    hold on;
%     pause(2000    00000000000*eps)
    pause(eps)
end
xlabel('Or (radial)');
ylabel('Otheta (local tangent)');
```

```
zlabel('Oh (radial normal)');
title('Hill frame boresight vector over time')
%}



%% Compare CW equations (keplerian) to the perturbed rototranslational
% in the hill frame!

% compute the whole time history for perturbations one the chief
p_exo = chief.perturbing(timerk4, chiefstate);
rho_Hill = zeros(3, length(timerk4));
rhod_Hill= zeros(3, length(timerk4));

for k = 1:length(timerk4)
    % pull out the current position vectors
    rc = chiefstate(2:4, k);
    vc = chiefstate(5:7, k);
    rd = deputystate(2:4, k);
    vd = deputystate(5:7, k);

    %  Must calculate a customized, perturbed omegaH/N vector here:
    HN = Cart2Hill([rc.' vc.']);
    h = norm(skew(rc)*vc);
    ah = HN*p_exo.a(:,k);
    H_omega_ON = HN*((skew(rc)*vc)/(norm(rc)^2)) + (HN*rc/h)*ah(3);
    rho_Hill(:,k) = HN*(rd - rc);
    rhod_Hill(:,k)= HN*(vd - vc) - skew(H_omega_ON)*rho_Hill(:,k);
end


figure;
subplot(3,1,1)
plot(timerk4, abs(rho_Hill(1,:)- state_out1(1,:))); grid on; hold on;
plot(timerk4, abs(rho_Hill(1,:)- state_out2(1,:))); grid on; hold on;
title('Error between NL and CW')
% legend('abs(NLnK-CW)')
legend('abs(NLnK-CW)','abs(NLnK-NL)')
ylabel('x error - km')
subplot(3,1,2)
plot(timerk4, abs(rho_Hill(2,:)- state_out1(2,:))); grid on; hold on;
plot(timerk4, abs(rho_Hill(2,:)- state_out2(2,:))); grid on; hold on;
ylabel('y error - km')
subplot(3,1,3)
plot(timerk4, abs(rho_Hill(3,:)- state_out1(3,:))); grid on; hold on;
plot(timerk4, abs(rho_Hill(3,:)- state_out2(3,:))); grid on; hold on;
ylabel('z error - km')
xlabel('time - seconds')
grid on;



% figure; plot3(rho_Hill(1,:), rho_Hill(2,:), rho_Hill(3,:));
% grid on; axis square; hold on;
% plot3(state_out1(1,:), state_out1(2,:), state_out1(3,:));
% grid on; axis square
% title('Hill Frame Coords of CW and NL (non-Kep)')
% legend('NL','CW')


figure; plot3(state_out2(1,:), state_out2(2,:), state_out2(3,:));
grid on; axis square; hold on;
plot3(rho_Hill(1,:), rho_Hill(2,:), rho_Hill(3,:));
plot3(state_out1(1,:), state_out1(2,:), state_out1(3,:));
grid on; axis square
title('Hill Frame Coords of all solutions')
legend('NL-Kep','NL-nonKep','CW')
xlabel('x [km]')
```

```
ylabel('y [km]')
zlabel('z [km]')
```