AAE 490 – MODELING AND SIMULATION

Blue Team Final Report

Low Earth Orbit Rendezvous

Victor Gandarillas, Nick Kowalczyk, Mitch Sangalis, Collin York, Dan Zhou

5/3/2013

Nomenclature

x_{ref}	= inertial x-position of Reference Spacecraft in n_1 direction
y_{ref}	= inertial y-position of Reference Spacecraft in n_2 direction
z_{ref}	= inertial z-position of Reference Spacecraft in n_3 direction
$\ddot{x}_{ref}$	= x component of Reference Spacecraft acceleration relative to Earth
$\ddot{y}_{ref}$	= y component of Reference Spacecraft acceleration relative to Earth
$\ddot{z}_{ref}$	= z component of Reference Spacecraft acceleration relative to Earth
$\overline{r}$	= reference orbit position vector from center of Earth to Reference Spacecraft
μ	= gravitational parameter of Earth
x	= x-position of Displaced Spacecraft relative to Reference Spacecraft
y	= y-position of Displaced Spacecraft relative to Reference Spacecraft
z	= z-position of Displaced Spacecraft relative to Reference Spacecraft
$\dot{x}$	= x component of Displaced Spacecraft velocity relative to Reference Spacecraft
$\dot{y}$	= y component of Displaced Spacecraft velocity relative to Reference Spacecraft
$\dot{z}$	= z component of Displaced Spacecraft velocity relative to Reference Spacecraft
$\ddot{x}$	= x component of Displaced Spacecraft acceleration relative to Reference Spacecraft
$\ddot{y}$	= y component of Displaced Spacecraft acceleration relative to Reference Spacecraft
$\ddot{z}$	= z component of Displaced Spacecraft acceleration relative to Reference Spacecraft
f_x	= x component of external force on Displaced Spacecraft
f_y	= y component of external force on Displaced Spacecraft
f_z	= z component of external force on Displaced Spacecraft
n	= mean motion of Earth and Reference Spacecraft
$x_{inertial,disp}$	= inertial x-position of Displaced Spacecraft in n_1 direction
$y_{inertial,disp}$	= inertial y-position of Displaced Spacecraft in n_2 direction
$z_{inertial,disp}$	= inertial z-position of Displaced Spacecraft in n_3 direction
$\hat{b}_i$	= orthogonal, dextral set of unit vectors in the principal axes of the body frame B
$\hat{a}_i$	= orthogonal, dextral set of unit vectors in the orbit frame A
$\hat{n}_i$	= orthogonal, dextral set of unit vectors in the inertial frame N
ε_i	= Euler parameters relating orbit frame A to body frame B
ε_i^d	= desired Euler parameter
ω_i	= angular velocities relating inertial frame N to body frame B, rad/s
I	= transverse moment of inertia, kg*m^2
J	= axial moment of inertia, kg*m^2
Ω	= angular velocity of orbit frame A in inertial frame N, rad/s
m_i	= angular acceleration input in the $\hat{b}_i$ directions
$\overline{x_0}$	= initial relative position vector, m
$\overline{v_0}$	= initial relative velocity vector, m
$\overline{x_f}$	= final relative position vector, m/s
$\overline{v_f}$	= final relative velocity vector, m/s
R	= radius of Earth, km
h	= altitude of spacecraft above Earth's surface, km

Introduction

The motivation for this project was to simulate the rendezvous of two spacecraft in Low Earth Orbit (LEO). The two spacecraft simulated are referred to as the Reference Spacecraft and the Displaced Spacecraft. The software used to simulate the motion of the two spacecraft is Trick v.7.23.1, provided by Metecs and NASA Johnson Space Center. The Avizo software package from VSG is used for visualization and an Apache subversion repository (SVN) used for revision control. A detailed description of the dynamics model, vehicle controllers, other software development, and lessons learned follows.

General Assumptions of the Model

The Reference Spacecraft is modeled as an axisymmetric rigid body orbiting Earth at an altitude of 200 km. The Earth is modeled as a point mass, fixed in the inertial frame. The orbit of the Reference Spacecraft was modeled using the inverse square law and the relative motion of the Displaced Spacecraft with respect to the Reference Spacecraft was modeled using the Hill-Clohessy-Wiltshire (HCW) equations. The Displaced Spacecraft is also modeled as an axisymmetric rigid body.

The force of gravity is the only natural force included. This means that other forces such as drag and solar radiation pressure are not incorporated. This governs the natural motion of the spacecraft (orbit and attitude). Although gravity is not modeled directly for the Displaced Spacecraft, the HCW equations serve as a linearized model of the same force. The Reference and Displaced Spacecraft are in coplanar circular orbits about Earth. The simulation allows for the Reference Spacecraft to move in any orbit, but that is not used for any analysis in this project, since HCW is limited in the cases where the Reference Spacecraft is not in circular orbit.

Gravity moments are also incorporated in the simulation. It is important to note however, that although gravity affects the orientation of the spacecraft, the change in orientation does not affect their orbits.

Maneuvering the spacecraft is also simplified. While actual spacecraft are limited in their translational motion by their orientation, the translational thrusters act orthogonal to each other in the inertial frame and the rotational thrusters produce moments about the inertial (body?) axes no matter the orientation of the spacecraft. The Reference Spacecraft does not change its orbit. Instead the Displaced Spacecraft is set into motion relative to the Reference Spacecraft and the LQR changes the orbit accordingly.

Both spacecraft have attitude controllers that use instantaneous moments that last for each timestep in the simulation. There is no spin stabilization of any kind. This would be very expensive for a real spacecraft, but in this model, mass is assumed to be constant.

Dynamics of the Model

Inverse Square Model

The reference spacecraft is modeled using the inverse square law for gravity.

$$\overline{F} = \frac{\mu m}{r^3}\overline{r} \tag{1}$$

The investigation is carried out in Low Earth Orbit at an altitude of 200km and a planetary radius of 6371km and a gravitational parameter of 3.896004418e5km^3/s^2.

The mass of the spacecraft is not included, so Eq.1 becomes

$$\ddot{r_i} = \frac{\mu}{r_i^3}\overline{r_i} \tag{2}$$

Equation 2 is broken into components in the source code.

$$\ddot{x}_{ref} = \frac{\mu}{x_{ref}^3} x_{ref} \tag{3}$$

$$\ddot{y}_{ref} = \frac{\mu}{y_{ref}^3} y_{ref} \tag{4}$$

$$\ddot{y}_{ref} = \frac{\mu}{z_{ref}^3} z_{ref} \tag{5}$$

Initial position and velocity are chosen so that the Reference Spacecraft is in a circular orbit about Earth. The orbit remains on the x-y plane for all tests described here. All z components of the initial conditions are set to zero. The location of the Reference Spacecraft in inertial coordinates is shown below in Figure 1:

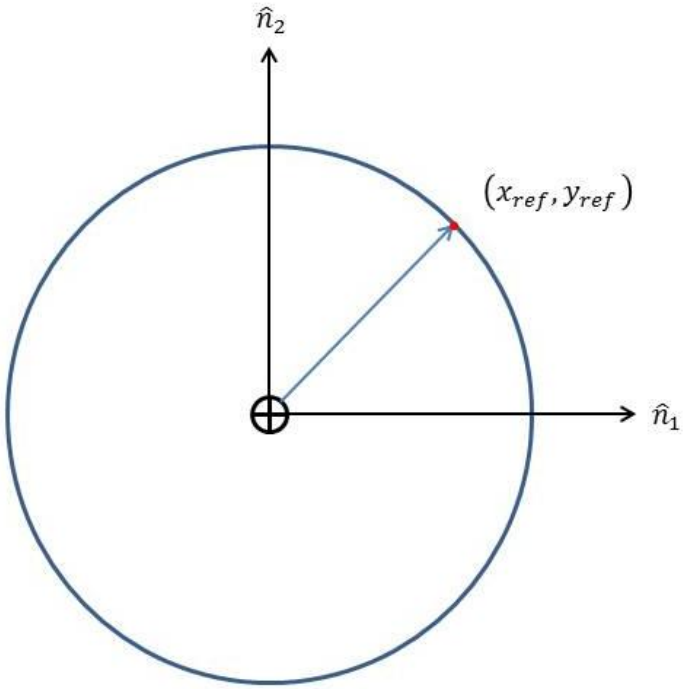


Figure 1: Reference Spacecraft in inertial frame orbiting Earth; $\hat{n}_3$points out of the page

The coordinates for the Reference Spacecraft do not need any conversion for Avizo since they are already inertial coordinates.

Hill-Clohessy-Wiltshire Model

The Hill-Clohessy-Wiltshire (HCW) equations are incorporated into the simulation to model the motion of the Displaced Spacecraft relative to the Reference Spacecraft. Non-linear equations of formation flight are linearized about a circular reference orbit to obtain the HCW equations. The orbit of the Reference Spacecraft is assumed to be circular (the initial conditions are actually chosen for a circular orbit, so in this case, it is not an assumption, but for more eccentric orbits, the simulation accuracy will worsen), which reduces the equations of motion of the Displaced Spacecraft to the Hill-Clohessy-Wiltshire model.

Derivation of the HCW equations begins with describing the mean motion of the orbit, shown in Equation 6 below:

$$n = \sqrt{\frac{\mu}{r^3}} \tag{6}$$

The Reference Spacecraft is placed in Low Earth Orbit at an altitude of 200km, using a planetary radius of 6371km and a gravitational parameter of 3.896004418e5km^3/s^2. When all the derivation is complete, we are left with three linear differential equations in the x, y, and z directions:

$$\ddot{x} = 2n\dot{y} + 3n^2x + f_x \quad (7)$$

$$\ddot{y} = -2n\dot{x} + f_y \quad (8)$$

$$\ddot{z} = -2n^2z + f_z \quad (9)$$

Given the proper initial velocity, based on the initial displacement, these equations may be used to produce rendezvous between our reference spacecraft and our Displaced Spacecraft. For these simulations, the Displaced Spacecraft position has no component in the z-direction, so from here on, it is assumed that this is a two-dimensional problem in x and y. In these equations, the reference spacecraft is always located at position (0,0), which means that the orbit reference frame is fixed in the orbit of the Reference Spacecraft. In order to extend the investigation to motion relative to the inertial frame, a reference frame conversion is necessary for the Displaced Spacecraft. The two reference frames are shown below in Figure 2:

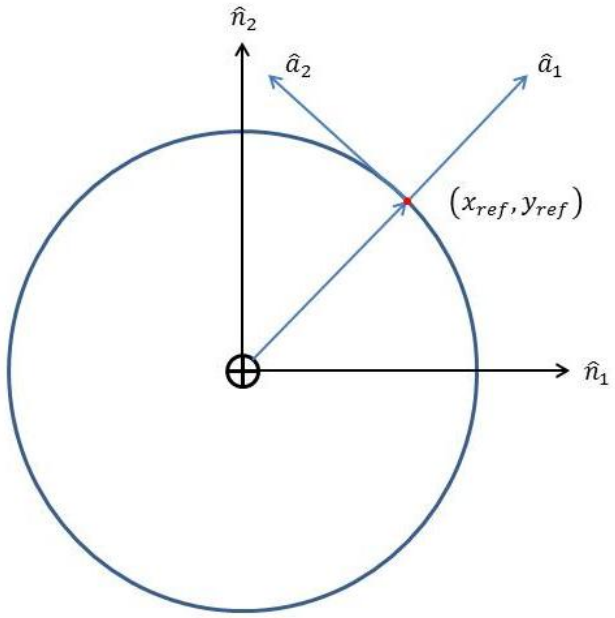


Figure 2: Inertial frame is shown as the N frame, while the HCW frame is shown using x and y

Notice that the orbital frame is centered at the reference spacecraft, which is depicted as the red dot. The $\hat{a}_1$-direction is radially outward, while the $\hat{a}_2$-direction is along the velocity vector.

The motion of the Displaced Spacecraft in the inertial frame is necessary for proper visualization. Information about the orbiting spacecraft is sent to Avizo for visualization. Avizo reads the position coordinates as inertial coordinates. The issue with that is, as mentioned above, the HCW equations hold the reference orbit fixed at (0,0). If the position of the Displaced Spacecraft is (0,0), it will appear in the center of the Earth in the visualization. To fix this problem, the Reference Spacecraft's position vector and the position vector of the Displaced Spacecraft must be added together before being sent to Avizo. The simple addition is shown in Equations 10 and 11:

$$x_{inertial,disp} = x_{ref} + \frac{x_{inertial}x_{ref}}{\sqrt{x_{ref}^2 + y_{ref}^2}} - \frac{y_{inertial}y_{ref}}{\sqrt{x_{ref}^2 + y_{ref}^2}} \quad (10)$$

$$y_{inertial,disp} = y_{ref} + \frac{x_{inertial}y_{ref}}{\sqrt{x_{ref}^2 + y_{ref}^2}} + \frac{y_{inertial}x_{ref}}{\sqrt{x_{ref}^2 + y_{ref}^2}} \quad (11)$$

Once these conversions have been applied, $x_{inertial,disp}$ and $y_{inertial,disp}$ are sent to Avizo as the position of the Displaced Spacecraft with respect to the inertial frame and both spacecraft show up orbiting Earth as the come together in rendezvous.

Attitude Model

The attitude model governing the orientation of the spacecraft describes an axisymmetric vehicle, symmetric about the 1-axis and subject to external gravity torque. Inertia properties were arbitrarily chosen to be I = 20kg-m^2, and J = 5kg-m^2, to give the Reference and Displaced Spacecraft prolate inertia ellipsoids.

The kinematic differential equations for each spacecraft are as follows

$$\dot{\varepsilon}_1 = \frac{1}{2}(\varepsilon_4\omega_1 - \varepsilon_3\omega_2 + \varepsilon_2(\omega_3 + \Omega)) \quad (12)$$

$$\dot{\varepsilon}_2 = \frac{1}{2}(\varepsilon_3\omega_1 + \varepsilon_4\omega_2 - \varepsilon_1(\omega_3 + \Omega)) \quad (13)$$

$$\dot{\varepsilon}_3 = \frac{1}{2}(-\varepsilon_2\omega_1 + \varepsilon_1\omega_2 + \varepsilon_4(\omega_3 + \Omega)) \quad (14)$$

$$\dot{\varepsilon}_4 = -\frac{1}{2}(\varepsilon_1\omega_1 + \varepsilon_2\omega_2 + \varepsilon_3(\omega_3 + \Omega)) \quad (15)$$

where ε_i are the Euler parameters of that vehicle's orientation. Equations 12-15 describe the relationship between the orbiting frame A and the body frame B. For reference, a depiction of the relevant frames is shown below:

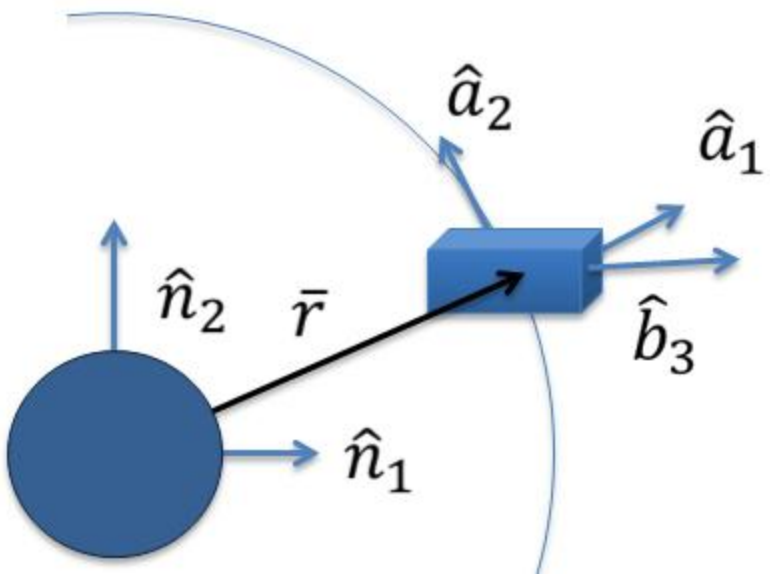


Figure 3: Reference frame configuration sketch

In addition to the kinematic equations of motion, dynamic equations of motion are required to fully describe the motion. The angular velocities used are measured in the B frame, as seen from an inertial observer in the N frame. Under the influence of gravity torque, the dynamic equations are derived to those shown below:

$$\dot{\omega}_1 = 12\Omega^2 X(\varepsilon_1\varepsilon_2 - \varepsilon_3\varepsilon_4)(\varepsilon_1\varepsilon_3 + \varepsilon_2\varepsilon_4) - X\omega_2\omega_3 + m_1 \quad (16)$$

$$\dot{\omega}_2 = -6\Omega^2 X(1 - 2\varepsilon_2^2 - 2\varepsilon_3^2)(\varepsilon_1\varepsilon_3 + \varepsilon_2\varepsilon_4) + X\omega_3\omega_1 + m_2 \quad (17)$$

$$\dot{\omega}_3 = m_3 \quad (18)$$

Where is *X* is the shape factor defined as follows:

$$X = 1 - \frac{J}{I} \quad (19)$$

The seven differential equations presented above describe the motion of a spacecraft in circular orbit subject to gravity torque, spinning at a rate such that the spacecraft would be fixed in the orbiting frame without any disturbances. In order to make this information useful in Avizo, the Euler axis and Euler angle must be calculated.

Using the kinematic variables above, the Euler axis and Euler angles can be calculated using Equations 20-23 below. This step is required for visualization purposes because Avizo uses Euler axis and Euler angle to determine orientation.

$$\lambda_1 = \frac{\varepsilon_1}{\sqrt{1 - \varepsilon_4^2}} \quad (20)$$

$$\lambda_2 = \frac{\varepsilon_2}{\sqrt{1 - \varepsilon_4^2}} \quad (21)$$

$$\lambda_3 = \frac{\varepsilon_3}{\sqrt{1-\varepsilon_4^2}} \tag{22}$$

$$\phi = 2\cos^{-1}(\varepsilon_4) \tag{23}$$

Controller Theory

HCW Controller Theory and Background

We derive the Hill-Clohessy-Wiltshire equations with an orbital reference frame $\hat{a}$, which rotates with the Reference Spacecraft. The $\hat{a}_1$, $\hat{a}_2$ and, $\hat{a}_3$ unit vectors represent the radially outward (*x*), tangential (*y*), and orbit normal directions (*z*), respectively. The scalar HCW equations are shown below [1]:

$$\begin{aligned} \ddot{x} - 2n\dot{y} - 3n^2x &= F_x \\ \ddot{y} + 2nx &= F_y \\ \ddot{z} + n^2z &= F_z \end{aligned} \tag{24}$$

Since the equation to describe motion in the $\hat{a}_3$ direction is independent of the other directions, it is removed from the controller, and only the $\hat{a}_1$-$\hat{a}_2$ plane is analyzed. Converted to state space form, the equations below show the matrix form of the equations [2].

$$\dot{u} = Au + Bv$$

$$\dot{u} = \begin{bmatrix} \dot{u}_1 \\ \dot{u}_2 \\ \dot{u}_3 \\ \dot{u}_4 \end{bmatrix} = \begin{bmatrix} \dot{x} \\ \dot{y} \\ \ddot{x} \\ \ddot{y} \end{bmatrix}$$

$$A = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 3n^2 & 0 & 0 & 2n \\ 0 & 0 & -2n & 0 \end{bmatrix}$$

$$u = \begin{bmatrix} u_1 \\ u_2 \\ u_3 \\ u_4 \end{bmatrix} = \begin{bmatrix} x \\ y \\ \dot{x} \\ \dot{y} \end{bmatrix} \tag{25}$$

$$B = \begin{bmatrix} 0 & 0 \\ 0 & 0 \\ 1 & 0 \\ 0 & 1 \end{bmatrix}$$

$$v = \begin{bmatrix} F_x \\ F_y \end{bmatrix}$$

An integral controller will allow for the command of the Displaced Spacecraft to a desired position and have it travel there with no steady state error. By incorporating the integral term for the *x* and *y* states compared to reference values x_d and y_d, the state space equations change to the following [2]:

$$\dot{u} = A_i u + B_i v - W u_d$$

$$\dot{u} = \begin{bmatrix} \dot{u}_1 \\ \dot{u}_2 \\ \dot{u}_3 \\ \dot{u}_4 \\ \dot{u}_5 \\ \dot{u}_6 \end{bmatrix} = \begin{bmatrix} \dot{x} \\ \dot{y} \\ \ddot{x} \\ \ddot{y} \\ x - x_d \\ y - y_d \end{bmatrix}$$

$$A_i = \begin{bmatrix} & A & & 0 & 0 \\ & & & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \end{bmatrix}$$

$$u = \begin{bmatrix} u_1 \\ u_2 \\ u_3 \\ u_4 \\ u_5 \\ u_6 \end{bmatrix} = \begin{bmatrix} x \\ y \\ x \\ y \\ \int (x - x_d) dt \\ \int (y - y_d) dt \end{bmatrix} \quad \textbf{(26)}$$

$$B_i = \begin{bmatrix} & B & \\ 0 & & 0 \\ 0 & & 0 \end{bmatrix}$$

$$W = \begin{bmatrix} 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ 1 & 0 \\ 0 & 1 \end{bmatrix}$$

$$u_d = \begin{bmatrix} x_d \\ y_d \end{bmatrix}$$

In order to change the performance characteristics of the HCW equations, state feedback control is employed. To accomplish this, we make the force inputs *v* a function of the state variables:

$$v = -Kx$$

where *K* is a 2-by-6 matrix of gains to relate the force inputs with the state variables. The new state space form of the controlled HCW equations are given below:

$$\dot{u} = (A_i - B_i K)u - W u_d \tag{27}$$

By expanding these matrices back into a scalar equation form, the following is derived:

$$\begin{aligned}
\dot{u}_1 &= \dot{x} \\
\dot{u}_2 &= \dot{y} \\
\dot{u}_3 &= (3n^2 - K_{11})x - K_{12}y - K_{13}\dot{x} + (2n - K_{14})\dot{y} - K_{15}\int (x - x_d)dt - K_{16}\int (y - y_d)dt \\
\dot{u}_4 &= -K_{21}x - K_{22}y + (-2n - K_{23})\dot{x} - K_{24}\dot{y} - K_{25}\int (x - x_d)dt - K_{26}\int (y - y_d)dt \\
\dot{u}_5 &= x - x_d \\
\dot{u}_6 &= y - y_d
\end{aligned} \tag{28}$$

The reason for converting to this form is that Trick does not have built-in functions to handle 6-by-6 matrices. The expanded form allows for easier implementation in Trick.

Next, the gain matrix *K* is selected. For the system to be stable, gain values are chosen such that the poles of the system are in the left half plane. This occurs when the eigenvalues of $A_i - B_i K$ are to the left of the imaginary axis in the s-plane. Since the equations form a 6th order system, we must pick 6 poles. The *place* command is utilized in MATLAB to perform this task.

Theoretically, the 6th order system can be approximated by a 2nd order system by giving 4 of the poles relatively larger real components. This approximation is valid because the responses due to the higher magnitude poles fade away much faster than the other two. For this experiment, the following poles are selected:

$$\begin{aligned}
\lambda_1 &= -0.04 + 0.02i \\
\lambda_2 &= -0.04 - 0.02i \\
\lambda_3 &= -0.15 \\
\lambda_4 &= -0.16 \\
\lambda_5 &= -0.18 \\
\lambda_6 &= -0.2
\end{aligned} \tag{29}$$

By enforcing these poles on the system, the gain matrix is calculated:

$$K = \begin{bmatrix} 0.0442 & -0.0163 & 0.3901 & -0.0399 & 0.0012 & -0.0015 \\ 0.0040 & 0.0420 & 0.0124 & 0.3799 & 0.0003 & 0.0011 \end{bmatrix} \tag{30}$$

From this pole placement, the responses from λ_1 and λ_2 should dominate. The damping ratio and natural frequency of the second order approximation is given below [3]:

$$s^2 + 2\zeta\omega_n s + \omega_n^2 = (s - \lambda_1)(s - \lambda_2) \tag{31}$$

The natural frequency ω_n for this system is 0.0447 rad/s, and the damping ratio is 0.8944. By inspection of the poles, the damped frequency ω_d is 0.02 rad/s.

The equations for settling time, rise time and percent overshoot of the second order system are given below:

$$t_{s,5\%} = \frac{3}{\zeta \omega_n}$$

$$t_r = \frac{1}{\omega_d} \tan^{-1} \frac{\omega_d}{\zeta \omega_n} \quad \textbf{(32)}$$

$$PO = 100\% \times e^{\left(-\frac{\zeta \pi}{\sqrt{1-\zeta^2}}\right)}$$

From these equations and the chosen poles of the second order system, the following table of theoretical performance characteristics is produced:

Table 1: Second Order Performance

settling time, s	75
rise time, s	23.18
percent overshoot, %	0.1867

Attitude Controller Theory and Background

The mathematical model for the attitude of each spacecraft is given in Eq. 1 below. The body frame B has an orthogonal, dextral set of unit vectors $\hat{b}_i$ fixed in the principal axes of the spacecraft. The inertial frame N is fixed in the Earth with unit vectors $\hat{n}_i$. Additionally, orbital frame A defines a set of unit vectors $\hat{a}_i$ rotating around Earth with a constant angular velocity. The Euler parameters ε relate the orientation of B in A. The angular velocities ω relate the rotation of B in N.

$$\dot{\varepsilon}_1 = \frac{1}{2}(\varepsilon_4\omega_1 + \varepsilon_2(\omega_3 + \Omega) - \varepsilon_3\omega_2)$$
$$\dot{\varepsilon}_2 = \frac{1}{2}(\varepsilon_3\omega_1 - \varepsilon_1(\omega_3 + \Omega) + \varepsilon_4\omega_2)$$
$$\dot{\varepsilon}_3 = \frac{1}{2}(-\varepsilon_2\omega_1 + \varepsilon_4(\omega_3 - \Omega) + \varepsilon_1\omega_2) \quad \textbf{(33)}$$
$$\dot{\varepsilon}_4 = -\frac{1}{2}(\varepsilon_1\omega_1 + \varepsilon_3(\omega_3 - \Omega) + \varepsilon_2\omega_2)$$

$$\dot{\omega}_1 = 12\Omega^2 X(\varepsilon_1\varepsilon_2 - \varepsilon_3\varepsilon_4)(\varepsilon_1\varepsilon_3 + \varepsilon_2\varepsilon_4) - X\omega_2\omega_3 + m_1$$

$$\dot{\omega}_2 = -6\Omega^2 X(1 - 2\varepsilon_2^2 - 2\varepsilon_3^2)(\varepsilon_1\varepsilon_3 + \varepsilon_2\varepsilon_4) + X\omega_3\omega_1 + m_2$$

$$\dot{\omega}_3 = m_3$$

$$\text{where } X = 1 - \frac{J}{I}$$

The linearization of Eq. 33 allows for the simple implementation of a linear state feedback controller. The particular solution of interest is when the axis of symmetry $\hat{b}_3$ aligns with the orbit radial direction $\hat{a}_1$. This orientation is chosen, because it allows the spacecraft to dock in coplanar orbits. In addition, experiments with the Hill-Clohessy-Wiltshire model show easier docking in the orbit radial direction than the velocity direction. Using perturbation theory, the following equations (Eq. 34) substitute into the attitude model (Eq. 33) for linearization:

$$\begin{aligned}
\varepsilon_1 &= 0 + \tilde{\varepsilon}_1 \\
\varepsilon_2 &= \frac{\sqrt{2}}{2} + \tilde{\varepsilon}_1 \\
\varepsilon_3 &= 0 + \tilde{\varepsilon}_3 \\
\varepsilon_4 &= \frac{\sqrt{2}}{2} + \tilde{\varepsilon}_4 \\
\omega_1 &= -\Omega + \tilde{\omega}_1 \\
\omega_2 &= 0 + \tilde{\omega}_2 \\
\omega_3 &= 0 + \tilde{\omega}_3 \\
m_1 &= 0 + \tilde{m}_1 \\
m_2 &= 0 + \tilde{m}_2 \\
m_3 &= 0 + \tilde{m}_3
\end{aligned} \tag{34}$$

The following linear model is achieved after substituting the perturbed state variables from Eq. 34 into Eq. 33 and neglecting higher order terms:

$$\begin{aligned}
\dot{\tilde{\varepsilon}}_1 &= \frac{\Omega}{2}\tilde{\varepsilon}_2 - \frac{\Omega}{2}\tilde{\varepsilon}_4 + \frac{\sqrt{2}}{4}\tilde{\omega}_1 + \frac{\sqrt{2}}{4}\tilde{\omega}_3 \\
\dot{\tilde{\varepsilon}}_2 &= -\frac{\Omega}{2}\tilde{\varepsilon}_1 - \frac{\Omega}{2}\tilde{\varepsilon}_3 + \frac{\sqrt{2}}{4}\tilde{\omega}_2 \\
\dot{\tilde{\varepsilon}}_3 &= \frac{\Omega}{2}\tilde{\varepsilon}_2 - \frac{\Omega}{2}\tilde{\varepsilon}_4 - \frac{\sqrt{2}}{4}\tilde{\omega}_1 + \frac{\sqrt{2}}{4}\tilde{\omega}_3 \\
\dot{\tilde{\varepsilon}}_4 &= \frac{\Omega}{2}\tilde{\varepsilon}_1 + \frac{\Omega}{2}\tilde{\varepsilon}_3 - \frac{\sqrt{2}}{4}\tilde{\omega}_2
\end{aligned} \tag{35}$$

$$\dot{\tilde{\omega}}_1 = Y\tilde{\varepsilon}_1 - Y\tilde{\varepsilon}_3 + \tilde{m}_1$$

$$\dot{\tilde{\omega}}_2 = -Y\tilde{\varepsilon}_2 - \Omega X\tilde{\omega}_3 + \tilde{m}_2$$

$$\dot{\tilde{\omega}}_3 = \tilde{m}_3$$

where $Y = 3\sqrt{2}\Omega^2 X$

By putting the linear model in state space form, Equation 36 describes the model in terms of matrices with state variable vector x, state matrix A, input vector u, and input matrix B.

$$\dot{x} = Ax + Bu$$

$$\begin{bmatrix} \dot{\tilde{\varepsilon}}_1 \\ \dot{\tilde{\varepsilon}}_2 \\ \dot{\tilde{\varepsilon}}_3 \\ \dot{\tilde{\varepsilon}}_4 \\ \dot{\tilde{\omega}}_1 \\ \dot{\tilde{\omega}}_2 \\ \dot{\tilde{\omega}}_3 \end{bmatrix} = \begin{bmatrix} 0 & \frac{\Omega}{2} & 0 & -\frac{\Omega}{2} & \frac{\sqrt{2}}{4} & 0 & \frac{\sqrt{2}}{4} \\ -\frac{\Omega}{2} & 0 & -\frac{\Omega}{2} & 0 & 0 & \frac{\sqrt{2}}{4} & 0 \\ 0 & \frac{\Omega}{2} & 0 & -\frac{\Omega}{2} & -\frac{\sqrt{2}}{4} & 0 & \frac{\sqrt{2}}{4} \\ \frac{\Omega}{2} & 0 & \frac{\Omega}{2} & 0 & 0 & -\frac{\sqrt{2}}{4} & 0 \\ Y & 0 & -Y & 0 & 0 & 0 & 0 \\ 0 & -Y & 0 & 0 & 0 & 0 & -\Omega X \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} \tilde{\varepsilon}_1 \\ \tilde{\varepsilon}_2 \\ \tilde{\varepsilon}_3 \\ \tilde{\varepsilon}_4 \\ \tilde{\omega}_1 \\ \tilde{\omega}_2 \\ \tilde{\omega}_3 \end{bmatrix} + \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \tilde{m}_1 \\ \tilde{m}_2 \\ \tilde{m}_3 \end{bmatrix} \quad \textbf{(36)}$$

By implementing integral terms for the Euler parameters, the state space equations become Eq. 37. The integral terms allow the system to move to different desired orientations.

$$\dot{x}_i = A_i x_i + B_i u - W x_d \quad \textbf{(37)}$$

$$\begin{bmatrix} \dot{\tilde{\varepsilon}}_1 \\ \dot{\tilde{\varepsilon}}_2 \\ \dot{\tilde{\varepsilon}}_3 \\ \dot{\tilde{\varepsilon}}_4 \\ \dot{\tilde{\omega}}_1 \\ \dot{\tilde{\omega}}_2 \\ \dot{\tilde{\omega}}_3 \\ \dot{e}_1 \\ \dot{e}_2 \\ \dot{e}_3 \\ \dot{e}_4 \end{bmatrix} = \begin{bmatrix} 0 & \frac{\Omega}{2} & 0 & -\frac{\Omega}{2} & \frac{\sqrt{2}}{4} & 0 & \frac{\sqrt{2}}{4} & 0 & 0 & 0 & 0 \\ -\frac{\Omega}{2} & 0 & -\frac{\Omega}{2} & 0 & 0 & \frac{\sqrt{2}}{4} & 0 & 0 & 0 & 0 & 0 \\ 0 & \frac{\Omega}{2} & 0 & -\frac{\Omega}{2} & -\frac{\sqrt{2}}{4} & 0 & \frac{\sqrt{2}}{4} & 0 & 0 & 0 & 0 \\ \frac{\Omega}{2} & 0 & \frac{\Omega}{2} & 0 & 0 & -\frac{\sqrt{2}}{4} & 0 & 0 & 0 & 0 & 0 \\ Y & 0 & -Y & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & -Y & 0 & 0 & 0 & 0 & -\Omega X & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} \tilde{\varepsilon}_1 \\ \tilde{\varepsilon}_2 \\ \tilde{\varepsilon}_3 \\ \tilde{\varepsilon}_4 \\ \tilde{\omega}_1 \\ \tilde{\omega}_2 \\ \tilde{\omega}_3 \\ e_1 \\ e_2 \\ e_3 \\ e_4 \end{bmatrix}$$

$$+ \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} \tilde{m}_1 \\ \tilde{m}_2 \\ \tilde{m}_3 \end{bmatrix} - \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \varepsilon_1^d \\ \varepsilon_2^d \\ \varepsilon_3^d \\ \varepsilon_4^d \end{bmatrix}$$

where $e_i = \int (\varepsilon_i - \varepsilon_i^d) dt$

By incorporating state feedback control, the input vector becomes a function of the state variables and a time-invariant gain matrix. This relationship is shown in Eq. 38.

$$u = -Kx_i \tag{38}$$

This relationship between the input variables and the state variables gives the new state space equation below:

$$\dot{x}_i = (A_i - B_i K)x_i - Wx_d \tag{39}$$

Next, values of the gain matrix K are chosen to produce a stable system with eigenvalues in the left half plane. To perform this task, the LQR function in MATLAB calculates gains for the linear system.

Finally, the gains from the linear system must be applied to the nonlinear attitude model. Numerical integration in Trick is utilized to incorporate the state feedback controller. At each time step, Eq. 38 converts the linear system gains and the instantaneous state variables to angular acceleration inputs to the system. This process effectively controls the nonlinear attitude model when the state variables are

Simulation Capability

Currently, the simulation is set up to model the dynamics of two spacecraft with an LQR guiding Displaced Spacecraft to Reference Spacecraft or some other location defined by the user in real time. The two spacecraft are initially in coplanar orbits. The LQR only operates in the X and Y directions, which have defined the orbital plane in all tests up to this point. Thrusters are modeled as effectors for the LQR. The thrusters operate in the X, Y, and Z directions, regardless of the attitudes or relative positions of the two spacecraft.

The simulation also controls attitude with thrusters, which apply a moment to the spacecraft about x, y, and z.

The simulation is linked with Avizo 7.0. The two spacecraft are visualized as gray boxes in orbit about a rotating Earth. Two views are available: inertially fixed, nonrotating view showing the spacecraft positions relative to Earth, and a local nonrotating view fixed relative to the Reference Spacecraft. Neither view follows a camera path relative to the body to which it is fixed.

Running in Real Time

```
#include "S_default.dat"
#include "Modified_data/orb_rmot_rot.dr"
#include "Modified_data/rotations.dr"
#include "Modified_data/avizo_sim_stuff.dr"
#include "Modified_data/realtime.dr"

sys.exec.sim_com.panel_size = Panel_Size_Lite ;
sys.exec.in.stripchart_input_file = "Modified_data/spacecraft.sc" ;

sys.exec.in.tv = Off ;
sys.exec.in.tv_input_file = "spacecraft.tv" ;

stop = 1*800;
```

The input file is interpreted and uses the C syntax. All file paths within the #include directives start from the TRICK_USER_HOME directory is required and the other files define what data is recorded when the simulation runs.

The GUI that comes up as a result of the code above is shown below.

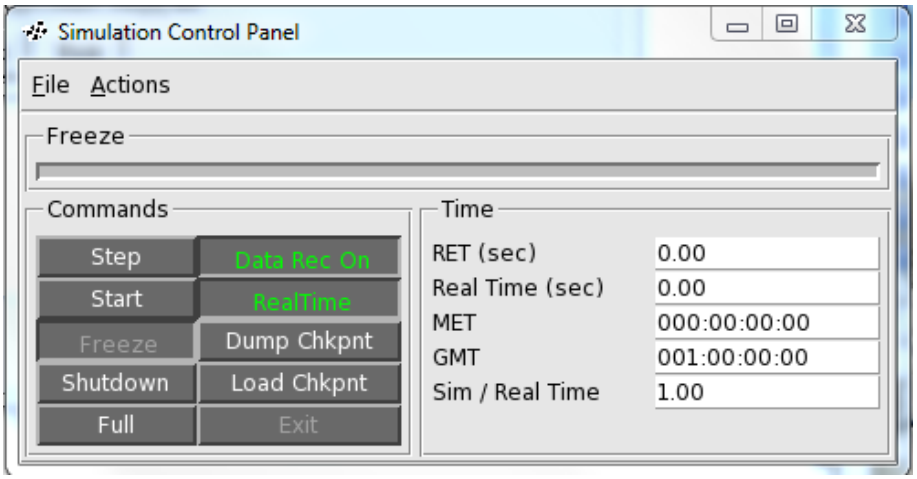


Figure 4: Simulation control Panel Lite View

Changing sys.exec.sim_com.panel_size = Panel_Size_Lite ; to sys.exec.sim_com.panel_size = Panel_Size_Ultralite ; will show the following GUI:

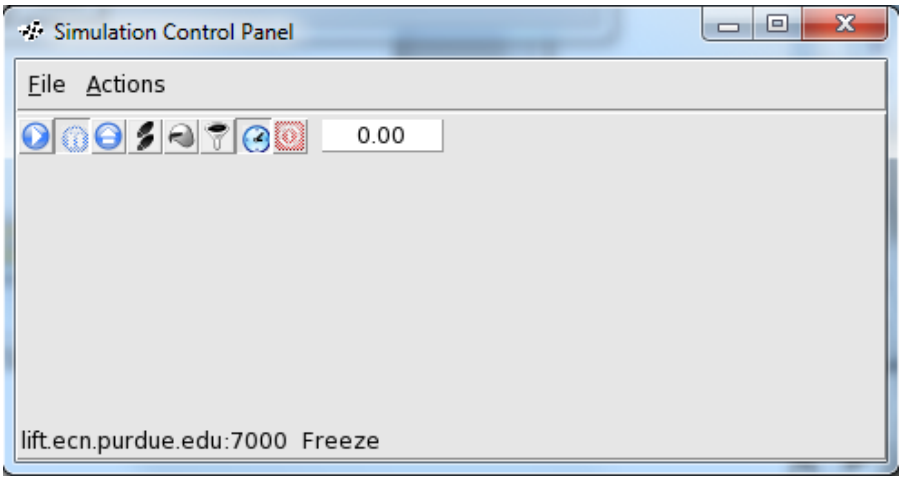


Figure 5: Simulation control Panel Ultralite View

The port number, necessary to run TCL files for the human in the loop GUI, is visible in both windows at the bottom (the first window needs to expand because the default size is too small). It doesn't matter which one is used.

In order to run in real time, the realtime.dr is required.

```
// Data file for running real time simulation
sys.exec.in.frame_log = Yes ;
sys.exec.in.rt_software_frame {s} = 0.01 ;
sys.exec.in.rt_itimer = No ;
sys.exec.in.rt_itimer_pause = No ;
sys.exec.in.rt_itimer_frame {s} = 0.01 ;
sys.exec.in.enable_freeze = Yes ;
sys.exec.work.freeze_command = Yes ;
sys.exec.sim_com.monitor_on = Yes ;
```

The simulation can be throttled from the Actions menu in the Simulation Control Panel when sys.exec.in.rt_itimer = No ; and sys.exec.in.rt_itimer_pause = No ;

Experiment 1 – Evaluation of HCW Error Accumulation

Introduction

Since HCW is an approximation of the motion of the Displaced Spacecraft, some position error will build up as the equations are integrated over time. The inverse square model calculates the reference position of the Displaced Spacecraft, and the HCW model calculates an approximate position. The distance between these two positions at each point in time is evaluated for Experiment 1. This distance (the error between the two models) and the position vectors are shown in Figure 6.

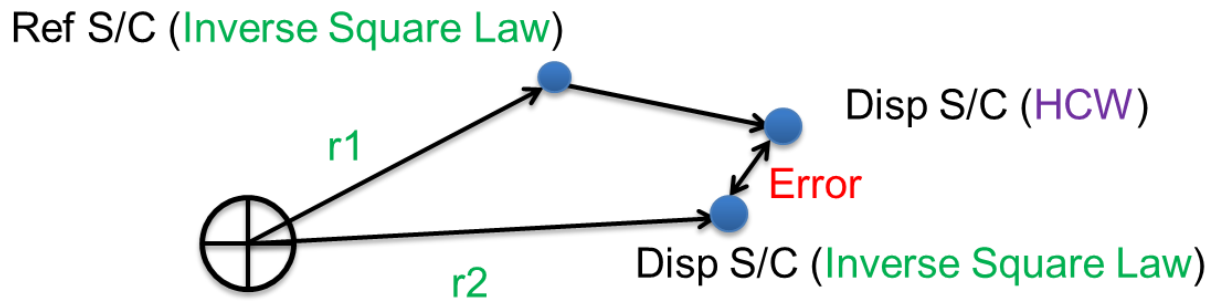


Figure 6: Experiment 1 Schematic

The time until the error reaches a prescribed threshold determines if the HCW model produces reasonably accurate results when compared to the inverse square model. The threshold is chosen to be 1 km and the simulation is run until the error reaches that threshold. Since the time until the threshold is reached is not known *a priori*, the simulation is written in Trick and uses Trick's dynamic events job to end the simulation. The time until the dynamic events job is triggered is output and results are discussed in the following sections.

Formal Statement of Hypothesis

All else equal, as initial distance between the Reference Spacecraft and Displaced Spacecraft in circular orbits increases, the time until the error between the HCW and inverse square models reaches 1 km decreases.

Details of the Experiment

In order to test the hypothesis, a series of Trick simulations varies the initial position vectors in the inertial frame of \the two spacecraft in LEO. The error between the models can be described by comparing the relative motion of the Displaced Spacecraft produced by the inverse square law and HCW models. Analysis of this error as a function of time produces a critical time for each set of initial conditions when the error reaches 1 km. The independent variable is the initial displacement of the Displaced Spacecraft relative to the Reference Spacecraft. The dependent variable is the time for the error to reach the 1 km threshold.

The motion of Reference Spacecraft is modeled using Equation 1. For one simulation, the motion Displaced Spacecraft is also modeled using Equation 1. In the second simulation, the motion of the Displaced Spacecraft is modeled using the HCW equations given in Equation 2. Both simulations run simultaneously in this experiment.

$$\ddot{x} = 2n\dot{y} + 3n^2x + f_x$$

$$\ddot{y} = -2n\dot{x} + f_y$$

$$\ddot{z} = -n^2z + f_z \tag{40}$$

In Eq. 40, x, y and z are components of position in a frame fixed in the Reference Spacecraft (frame $\hat{a}$, Figure 2). The force components f_x, f_y and f_z are set to 0 for this experiment. That is, drag and other natural forces are ignored and the spacecraft is in free motion.

The Earth is fixed at the origin of the inertial frame. The Reference Spacecraft is described by the inverse square model and initial conditions are chosen for a circular orbit. The Displaced Spacecraft is also described by the inverse square model in a circular orbit for one simulation. The position of the Displaced Spacecraft is given by the summation of the Reference Spacecraft position vector and the position of the Displaced Spacecraft relative to the Reference Spacecraft calculated from HCW. From Eq. 41, the error is the magnitude of the difference between the positions of the Displaced Spacecraft from each simulation.

$$Error = \sqrt{\left(r2_x - (r1_x + HCW_x)\right)^2 + \left(r2_y - \left(r1_y + HCW_y\right)\right)^2} \tag{41}$$

Table 2: Constants

Relevant Constants	
Earth Radius [km]	**6371.0**
Gravitational Parameter [km^3/s^2]	**398600.4418**

Results from the Trick simulation for 5 trials are used for analysis, each with a different displacement between Reference Spacecraft and Displaced Spacecraft. The initial velocity vectors are chosen to produce circular orbits for both spacecraft. These trials allow the testing of 1, 10, 100, 500, and 1000 m displacements between the two spacecraft. The total altitudes of the spacecraft from the Earth's surface are given in Table 3. The displacement values were chosen to be arbitrarily close to the Reference Spacecraft.

Table 3: Initial Conditions

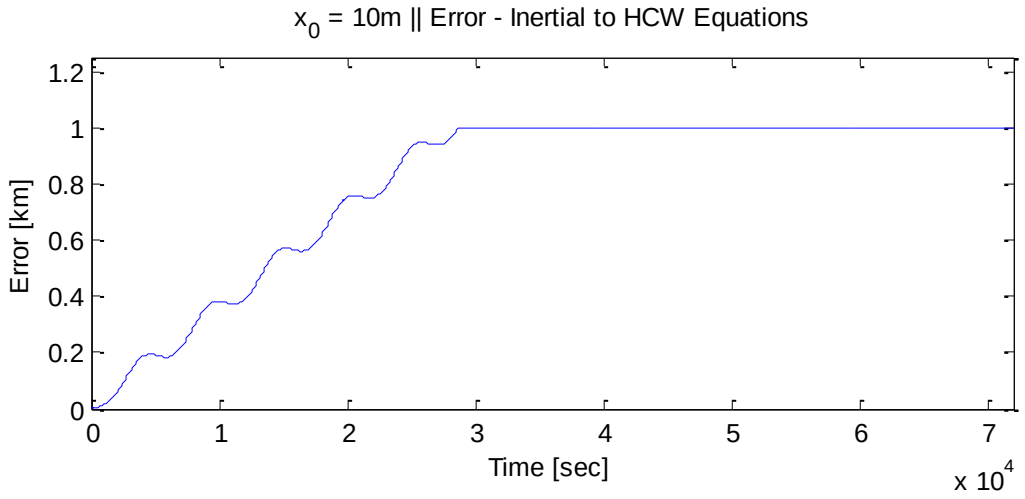
Trial #	Initial Reference Spacecraft	Initial Displaced Spacecraft

	Orbital Radius [km]	Orbital Radius [km]
1	200	200.001
2	200	200.01
3	200	200.1
4	200	200.5
5	200	201

The Trick simulation is run with a time step of 0.1 s. The Runge-Kutta 4th Order method was used for all numerical integration in the Trick simulation. A dynamic event is in place to stop the simulation when the error magnitude reaches 1 km. The error tolerance for the dynamic event is 10^{-5}.

Analysis

The time it takes for the error to reach the prescribed threshold of 1 km by looking at the error time history plots. The plots below show error time history plots for initial conditions of 10m, 100m, and 1000m. As discussed previously, once the threshold is reached, the dynamic events job sets the accelerations to zero yielding the flat lines shown in the figures.



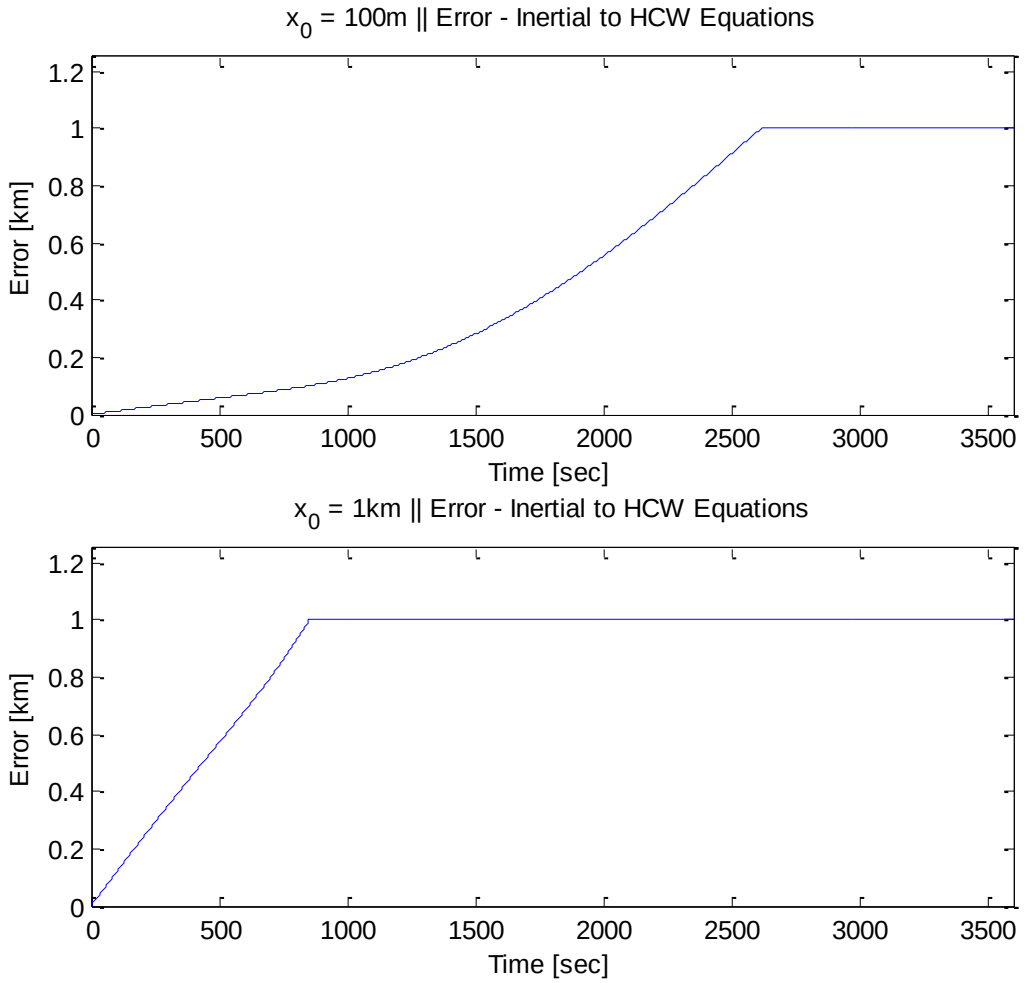


Figure 7: Error Time History Plots for 10m (top), 100m (middle), and 1000m (bottom)

As expected, the error starts at zero since any possible modeling error has not yet entered the system. However, the error seems to grow exponentially at first. For longer periods, the error can be shown to change as a sinusoidal curve that oscillates about a line with an upward trend. For each case we see that the error does in fact reach the prescribed maximum allowable error in a finite time. This relationship between the time it takes to reach our error threshold and the initial conditions is important to know when eventually running the final rendezvous and docking simulation.

Plot the times to reach threshold against their respective initial displacements, the relationship between the accuracy of the HCW equations and the input conditions becomes apparent (Figure 8).

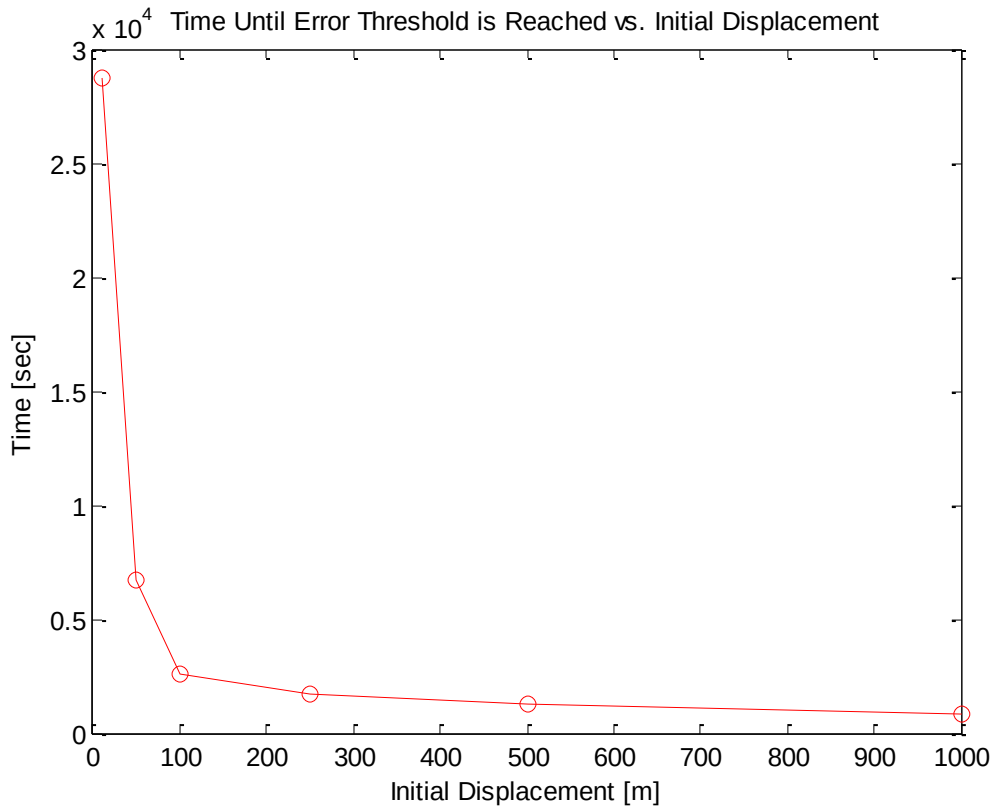


Figure 8: Time to reach error threshold for varying initial displacement

As expected, the maximum amount of time to run a simulation using the HCW model until the error becomes unacceptable is inversely related to the magnitude of the initial displacement. It is also important to note that if the error threshold is changed to either a lower or a higher value, this same trend will still be observed; only the final times will vary.

Conclusion

It has been demonstrated that a longer integration time and a large initial relative position between two spacecraft increase the error between the inverse square model and the HCW model. The problem has been treated as a two body problem for each spacecraft, with the Earth acting as the attracting body for each spacecraft. That is, there is no gravitational interaction between the two spacecraft in the simulation. In this setting, the time it takes for the error in the HCW model to reach a predefined threshold is inversely proportional to the initial relative positions of two spacecraft in circular orbit. Here, Earth is the central body and the orbits of the two Displaced Spacecraft from both models propagated simultaneously until the error reaches the prescribed threshold of 1 km. The motion of the Reference Spacecraft is modeled with the inverse square law in both scenarios. The Displaced Spacecraft uses the inverse square law in one model and HCW in the other. The difference in inertial position of the Displaced Spacecraft between the two models is the amount of error in the HCW model.

The error grows as the combination of a linear function and a sinusoidal function. In the inertial frame, the actual orbit of the Displaced Spacecraft is circular and the center of the orbit modeled with the HCW equations of motion shifts away from the Earth as time goes on.

As expected, the error for the second spacecraft between the standard inverse square model and HCW model increases with time, regardless of initial conditions. Increasing the initial separation of the two spacecraft causes the error to reach a prescribed threshold in less time than with a larger initial separation. The relationship holds for larger permissible errors.

Experiment 2 – Controlling Spacecraft to Rendezvous

Hypothesis

The response of a 6^{th} order HCW model with an integral and state feedback controller can be modeled as a 2^{nd} order system if the other poles are placed relatively far away.

If the four poles farthest from the origin are moved far away, the second order system comprised of the two remaining poles will produce a step -response with rise time, settling time, and maximum overshoot that are within 10% of the 6^{th} order system.

Procedure

1) Choose controller gains
2) Input gains to Trick in the "input" file
3) Set all initial conditions for relative motion to 0 in the "rmot.d" file
4) Set the desired position to be at (0.02, 0). This makes the vehicle move 20m in the x-direction while trying to hold y at 0
5) Compile and run the simulation for 300s and record the data
6) Using trick_dp, plot the response of rho(x) and rho(y) against time, as well as rho(x) against rho(y)
7) Use the *trk2csv* command to create a CSV file for MATLAB to read
8) In MATLAB, use the *stepinfo* command to calculate response characteristics
9) Set all initial conditions for relative motion to 0 in the "rmot.d" file
10) Set the desired position to be at (0, 0.02). This makes the vehicle move 20m in the y-direction while trying to hold x at 0
11) Compile and run the simulation for 300s and record the data
12) Using trick_dp, plot the response of rho(x) and rho(y) against time, as well as rho(x) against rho(y)
13) Use the *trk2csv* command to create a CSV file for MATLAB to read
14) In MATLAB, use the *stepinfo* command to calculate response characteristics

Analysis and Discussion

With initial conditions equal to zero, Figure 9 shows the response of the system as the $\hat{a}_1$ component of relative motion is commanded to 20 m. The other components are commanded to a constant zero.

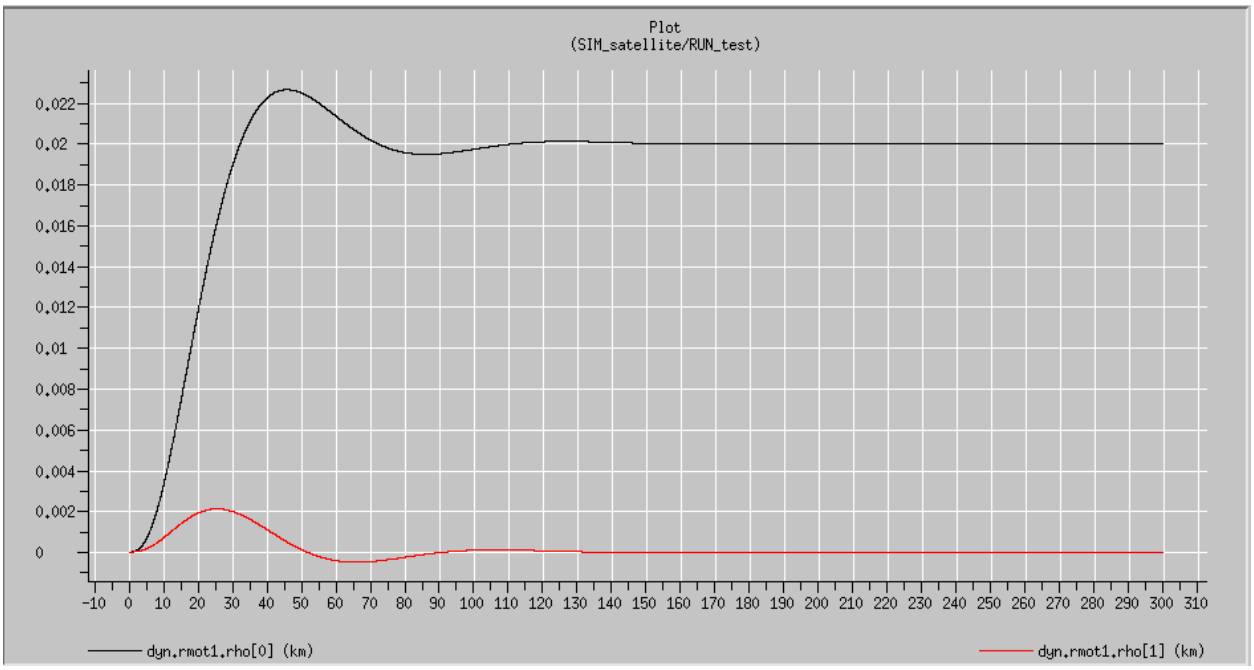


Figure 9: The system responds in both directions as the radial component of relative motion is sent to 20 m away from the Reference Spacecraft.

As the Displaced Spacecraft moves to the desired location, there is overshoot in the position. Also, the Displaced Spacecraft shows oscillations as it settles around 20 m. Table 4 shows the performance parameters of the system as calculated by MATLAB's *stepinfo* function.

Table 4: *x* Step Response

settling time, s	95.30
rise time, s	20.34
percent overshoot, %	13.21

The predicted rise time of 23.18 s from the second order system is very close to the experimental rise time of 20.34 s. The settling time of the second order approximation has a 21.3% error when compared to the settling time of the 6^{th} order system. The overshoot percent is off by two orders of magnitude between the second order approximation and the actual system.
With all other initial conditions equal to zero, Figure 2 shows the response of the system as the $\hat{a}_2$ component of relative motion is commanded to 20 m. The other components are commanded to a constant zero.

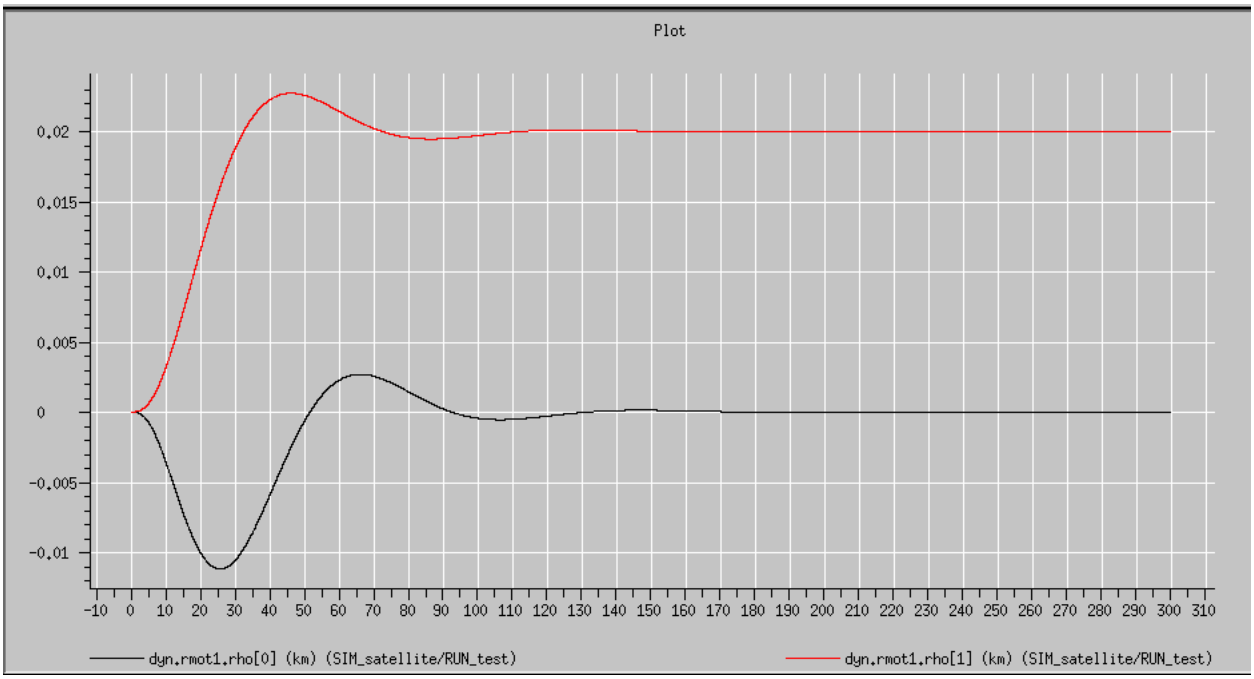


Figure 10: The system responds in both directions as the prograde component of relative motion is sent to 20 m away from the Reference Spacecraft.

As the Displaced Spacecraft moves to the desired location, there is similar behavior to the radial displacement. The Displaced Spacecraft shows oscillations as it settles around 20 m. Table 5 shows the performance parameters of the system as calculated by MATLAB's *stepinfo* function.

Table 5: *y* Step Response

settling time, s	96.21
rise time, s	20.45
percent overshoot, %	13.59

Like in the previous case, the predicted rise time of 23.18 s from the second order system is very close to the experimental rise time of 20.45 s. The settling time of the second order approximation has a 22% error when compared to the settling time of the 6th order system. The overshoot percent is off by two orders of magnitude between the second order approximation and the actual system.

Though the rise time of the approximated system gives reasonable results, the overshoot percent estimation by the second order system has an unacceptable level of error. The rise time calculations from the second order system give reasonable estimations for order of magnitude and general behavior of the more complicated system. Overall, the second order approximation only offers useful information in some performance parameters.

Conclusion

Because the 2nd order approximation has more than 10% error relative to the 6th order controller, the team rejects the hypothesis as false. A 2nd order approximation to a state-feedback integral

controller of the Hill-Clohessy-Wiltshire relative motion equations does not produce sufficient accuracy.

controller of the Hill-Clohessy-Wiltshire relative motion equations does not produce sufficient accuracy.

Graphical User Interface (GUI) for HCW controller

In order to bring a human into the loop, a graphical user interface (GUI) was created that would allow the user to change the final location of the Displaced Spacecraft relative to the Reference Spacecraft. Figure 1 below provides a screenshot of the GUI itself.

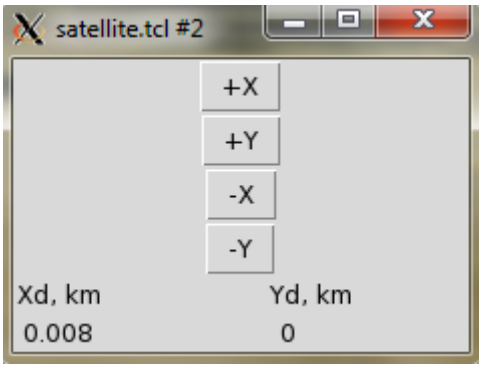


Figure 11: GUI used to change the desired location of the Displaced Spacecraft relative to the Reference Spacecraft.

Initially, the controller is set up to take a spacecraft with any set of initial conditions and bring it to rendezvous with the Reference Spacecraft in circular orbit. With the orbiting frame used in the HCW equations, this is equivalent to moving the second spacecraft to the origin. The GUI allows the user to change this final location, so instead of leading to rendezvous, the spacecraft could end up 8 meters away, for example, as illustrated in Figure 11.

When one of the buttons is clicked, the final location changes in the corresponding direction by 8 meters. This value is hardcoded in the program itself and can be changed if desired. Later, more functionality will be included so that the user can specify how much the location will change with each click. The label, currently showing [0.008, 0] in Figure 11, updates as the final location is changed. This allows the user to know what further input is required to get to their desired location.

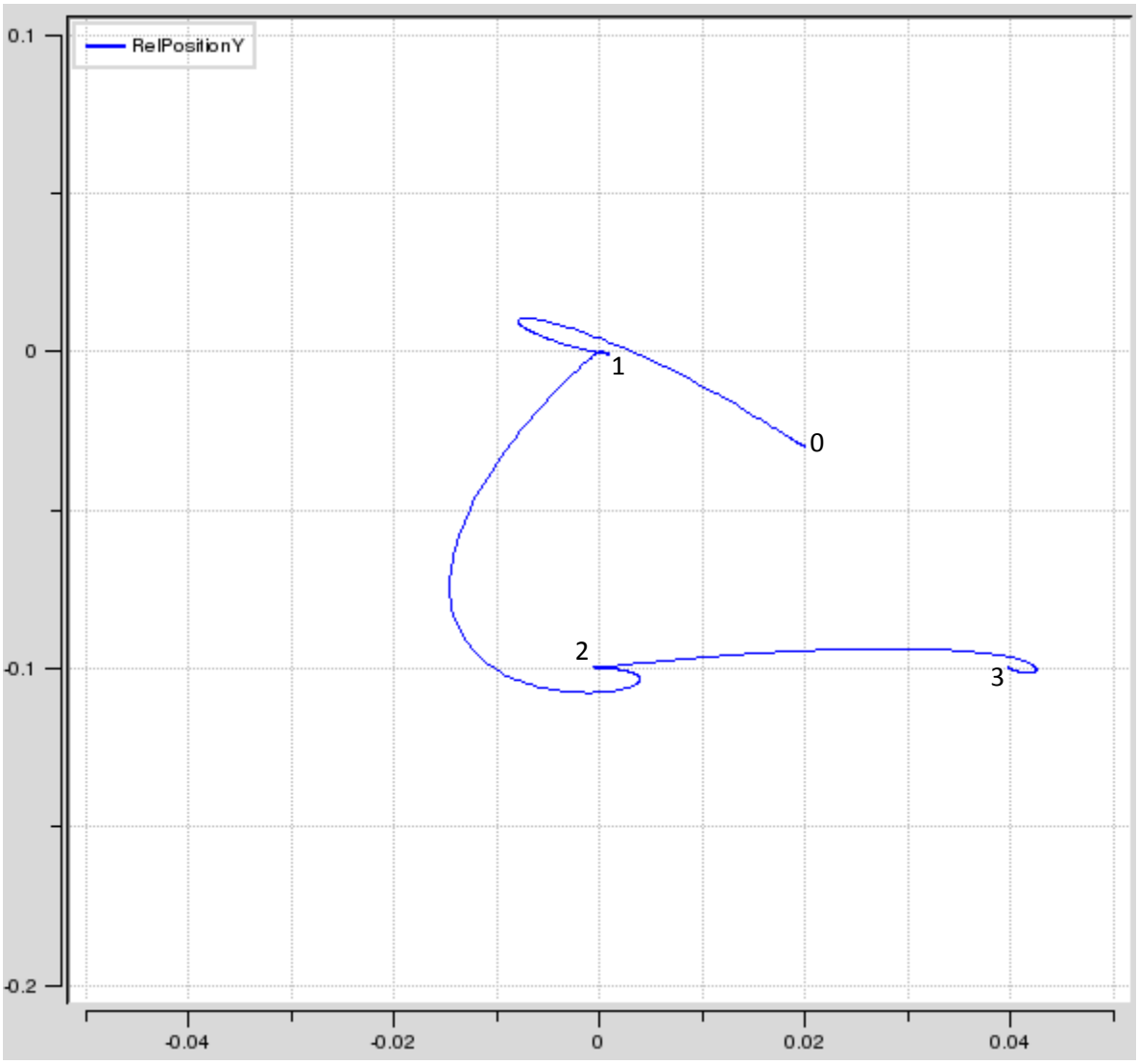


Figure 12: Trajectory history as the user changes the desired final location.

Figure 12 above shows the trajectory history while the desired location is being continuously changed. Initially, the Displaced Spacecraft begins at point 0 with random initial velocities and then meets with the first spacecraft at the origin, point 1, with motion governed by the controller. In this demonstration, upon reaching the origin, the final location was changed to point 2 located at [x, y] = [0, -0.1]. Similarly, the desired location was then moved to point 3 at [x, y] = [0.04, -0.1].

As shown in the figure, the controller is able to adjust the motion of the spacecraft in order to move to the desired location. Further insight into this motion is described in the experiment itself.

The GUI itself is created using the Tool Command Language (Tcl). The first task in creating the label and buttons is relatively straightforward with a code excerpt being presented in Figure 13.

```
proc createX { } {
        global rmot1
        button .bX -text "+X" -command "fire_jetX $rmot1(socket)"
        label .lX -textvariable rmot1(flags)
        pack .bX .lX
        }
```

Figure 13: Tcl code segment for creating a button and label.

This segment of code creates the first button in the GUI labeled "+X" as well as the label underneath which displays the desired location as it is updated. The [-command "fire_jetX $rmot1(socket)"] portion calls the procedure that actually modifies the final location of the Displaced Spacecraft. This procedure is shown in Figure 14 below.

```
proc fire_jetX { sock } {
        Simcom::send_cmd $sock "dyn.rmot1.Xd\[0\] += .02 ;"
        }
```

Figure 14: Tcl code segment for changing the final location when a button is pushed.

Since the final location is a variable within the equations of motion, the value can be modified directly from the .tcl script as shown. Thus, there is no need to modify the model or simulation code as is required when adding an additional force or perturbation such as in the cannon example presented in the appendix.

Staged controller

A GUI was also created to provide clearer visualizations and to closer represent how controllers would be enacted in the real world. The concept behind this consists of a series of stages. First, the Displaced Spacecraft is set to approach the Reference Spacecraft. Upon reaching the vicinity of the Reference Spacecraft, the attitude controller then comes into play to orient the vehicles for rendezvous. Finally, after the attitude is stabilized, the HCW controller comes back into play to bring the two spacecraft together.
The GUI consists of three simple buttons that will enact each stage described above and is shown in Figure 15 below.

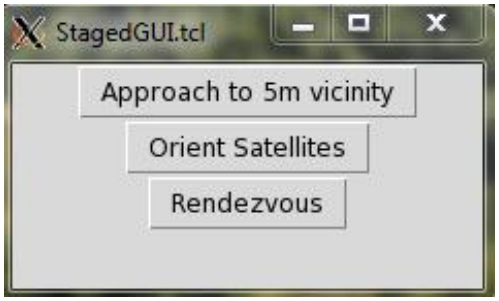


Figure 15: GUI that performs staged rendezvous.

Upon running this simulation, the relative position time history is obtained and presented in Figure 16 below.

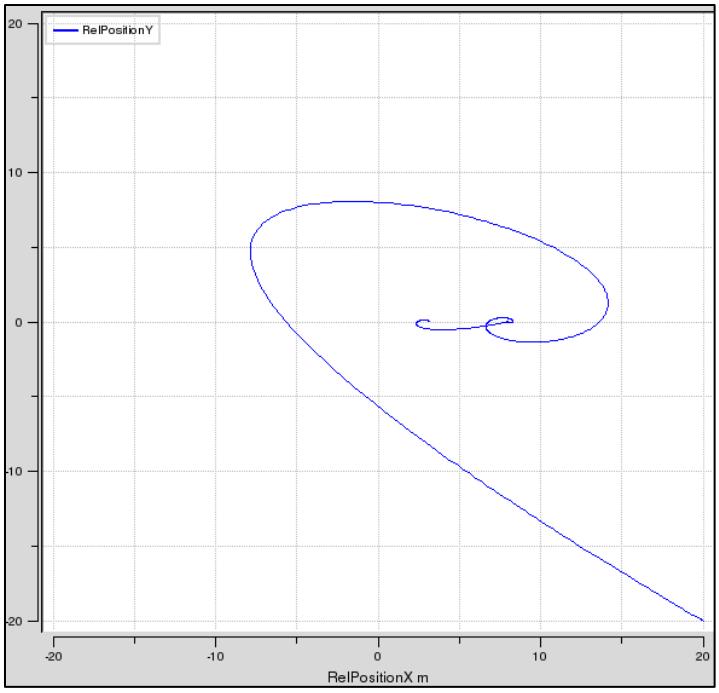


Figure 16: Relative Position Time History

The Displaced Spacecraft begins in the bottom right corner. During the first stage (approach), the spacecraft moves to a location 5m from the Reference Spacecraft. It moves to 8m in the x direction in order to account for the dimensions of the spacecraft. During the second stage, the position does not change as only the attitude is being controlled. Lastly, the spacecraft rendezvous as indicated by the 3m separation between the spacecraft, again counting for the dimensions of the spacecraft.

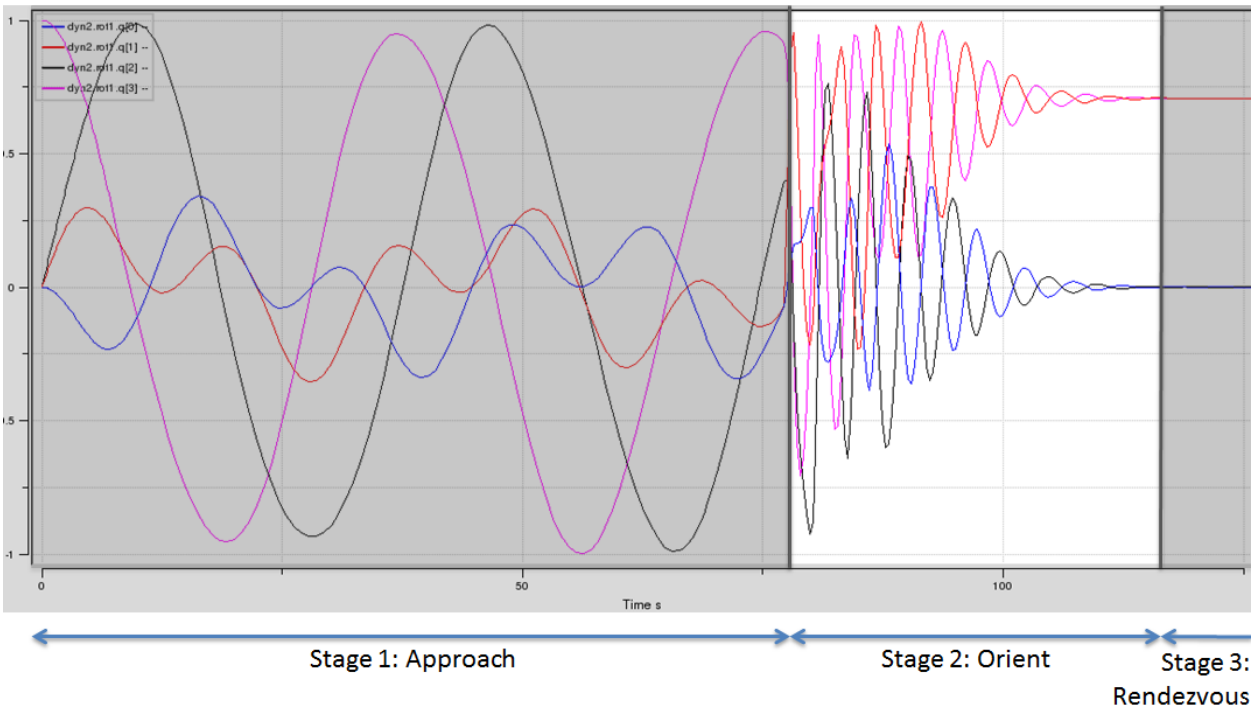


Figure 17: Euler Parameter Time History

Figure 3 above presents the orientation history of the spacecraft for each of the stages. During the first stage, the orientation represents the natural motion influenced mainly by the initial conditions. During this time, the controller is not in play. During the second stage, the attitude controller comes into play and stabilizes the orientation. The desired values of the Euler parameters were chosen as to orient the vehicles in a way that is ideal for the spacecraft to approach in the radial direction. Finally, the orientation in stage 3 shows the stabilized spacecraft which is required for actual rendezvous.

It is important to note that the attitude controller is very sensitive to when it is first enacted. If the controller begins when the Euler parameters are nowhere near their desired solutions, the controller will not be able to stabilize the spacecraft. Ideally, some sort of check would be implemented to prevent the controller from beginning while far from the solution.

Visualization

Overview

One major component of the simulation is the link between the simulation and Avizo for visualization. A TCL script named serverconnect.scro is written in order to send data from Trick, which runs on the UNIX server, to Avizo, which runs on the client server (Figure N). Data can be sent in real time – as Trick generates the data, it immediately goes to Avizo. The data is calculated, sent, and visualized in real time and the simulation is controlled by the user. The user can also throttle the speed of the simulation to reach a desired test point more quickly.

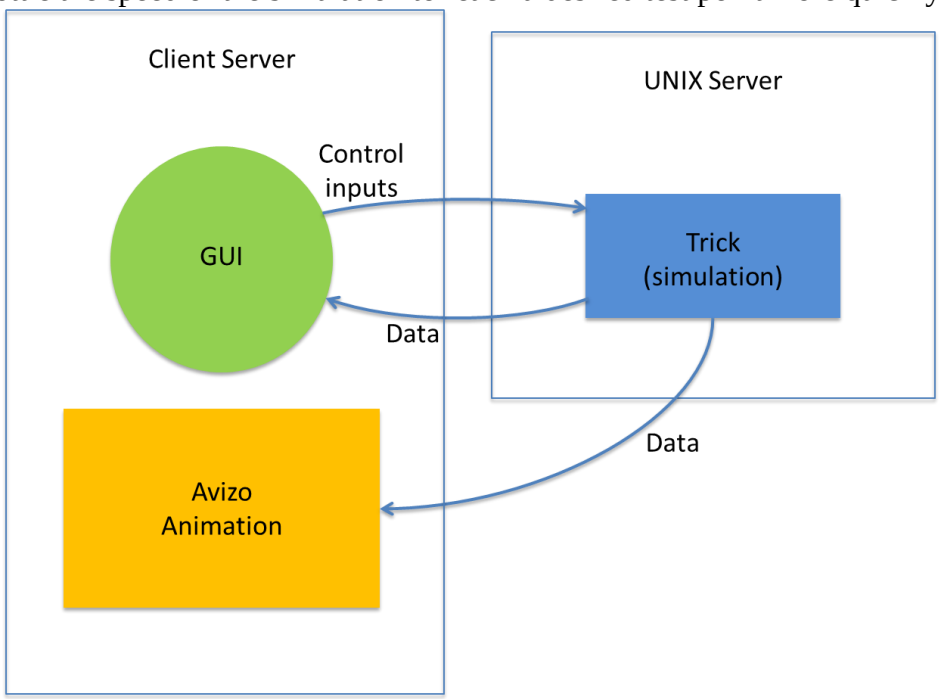


Figure 18: Simulation hardware setup and data flow

Development

Initially, the visualization used .wrl 3D models that were readily available. Two windows were open in Avizo, side by side. One window showed one object orbiting another, representing the Reference Spacecraft and the Earth. The positions of both vehicles in the inertial reference frame were sent to Avizo. Attitude was not incorporated initially into the visualization.

The other window showed one object representing the Displaced Spacecraft moving around and eventually meeting up with the another object, representing the Reference Spacecraft. The positions sent to Avizo were the relative positions in the Relative Spacecraft reference frame ([0, 0, 0] for the Reference Spacecraft).

The final TCL scripts send data of both spacecraft to Avizo. Position data are all in the inertial frame. Avizo displays the Displaced Spacecraft and the Reference Spacecraft in two views. One view is inertially fixed and displays a rotating Earth with both spacecraft in orbit. Another view is fixed to the Reference Spacecraft and shows the Displaced Spacecraft following its trajectory relative to the Reference Spacecraft. Neither view rotates about the point to which it is fixed or moved relative to the central object (Earth in the first view, Reference Spacecraft in the second view).

Avizo

Working with Avizo presented many challenges, especially when attempting to incorporate a real-time connection with Trick. The team was presented with two files that work together to create a visualization in Avizo of one body moving in three directions. From these two files, the team was able to determine a great amount about how the connection between Trick and Avizo works, how to write in tcl, and how to modify the files so that Avizo looks as desired as soon as the project is opened up.
Initially, when trying to add translational motion for a second body, the team thought that the procedure named *get_sim_data* would have to be duplicated. The team also thought to open up a second socket and create a separate array for the new data instead of reusing *positions*. As it turns out, none of that is necessary. All of the data can be sent through the same socket, as shown in Figure 19 below:

```
simcom_send_cmd $sat(socket) "var_cycle = $cycle_rate;"
simcom_send_cmd $sat(socket) "var_add dyn.rmot1.rho\[0\];"
simcom_send_cmd $sat(socket) "var_add dyn.rmot1.rho\[1\];"
simcom_send_cmd $sat(socket) "var_add dyn.rmot1.rho\[2\];"
simcom_send_cmd $sat(socket) "var_add dyn.orb1.r\[0\];"
simcom_send_cmd $sat(socket) "var_add dyn.orb1.r\[1\];"
simcom_send_cmd $sat(socket) "var_add dyn.orb1.r\[2\];"
simcom_send_cmd $sat(socket) "var_add dyn2.rot1.ehat\[0\];"
simcom_send_cmd $sat(socket) "var_add dyn2.rot1.ehat\[1\];"
simcom_send_cmd $sat(socket) "var_add dyn2.rot1.ehat\[2\];"
simcom_send_cmd $sat(socket) "var_add dyn2.rot2.ehat\[0\];"
simcom_send_cmd $sat(socket) "var_add dyn2.rot2.ehat\[1\];"
simcom_send_cmd $sat(socket) "var_add dyn2.rot2.ehat\[2\];"
simcom_send_cmd $sat(socket) "var_add dyn2.rot1.Phi;"
simcom_send_cmd $sat(socket) "var_add dyn2.rot2.Phi;"
```

Figure 19: This block of code in the get_sim_data procedure collectes all of the variables of interest in through a single socket.

Once the team learned how to hear all of the data coming out of Trick, the team had to figure out how to store it. After some research, the team found out that the commands *lappend* and *lindex* may be used in the way shown below in Figure 20 to accomplish the task:

```
# Appends data to lists
lappend xPositionT1 "[lindex $sat(positions) 0]"
lappend yPositionT1 "[lindex $sat(positions) 1]"
lappend zPositionT1 "[lindex $sat(positions) 2]"
lappend xPositionT2 "[lindex $sat(positions) 3]"
lappend yPositionT2 "[lindex $sat(positions) 4]"
lappend zPositionT2 "[lindex $sat(positions) 5]"
lappend xAxisT1 "[lindex $sat(positions) 6]"
lappend yAxisT1 "[lindex $sat(positions) 7]"
lappend zAxisT1 "[lindex $sat(positions) 8]"
lappend xAxisT2 "[lindex $sat(positions) 9]"
lappend yAxisT2 "[lindex $sat(positions) 10]"
lappend zAxisT2 "[lindex $sat(positions) 11]"
lappend ThetaT1 "[lindex $sat(positions) 12]"
lappend ThetaT2 "[lindex $sat(positions) 13]"
```

Figure 20: This block of code appends the data coming out of Trick (positions) to a time-based array called xPositionT1, for example.

The code in Figure 21 is within a loop that runs each iteration. What happens is the *lindex* command is used to access the value of *positions*, coming thought the socket named *sat*, located at the 0^{th} element of *positions*. The value returned by the call of *lindex* is then appended to the time-based array of choice using the *lappend* command. Once the data is stored, the objects in Avizo must be updated so that they rotate and translate as they should. This part of the file is fairly straight forward, and is depicted below in Figure 21:

```
update
region1sc_Spring2.wrl setTranslation $HCWInertiaX $HCWInertiaY $HCWInertiaZ
region1sc_Spring2.wrl touch 1
region1sc_Spring2.wrl fire
viewer 1 redraw
viewer 0 redraw

update
region1sc_Spring2.wrl setRotation [lindex $sat(positions) 6] [lindex $sat(positions) 7] [lindex $sat(positions) 8] [lindex $sat(positions) 12]
region1sc_Spring2.wrl touch 1
region1sc_Spring2.wrl fire
viewer 1 redraw
viewer 0 redraw
```

Figure 21: This code shows how to update the position and orientation of one of our objects.

To change position, use the command *setTranslation*, and give the inertial x, y, and z positions respectively. These must be inertial positions, with (0,0) being the center of your attracting body. Also shown is the command to change orientation, *setRotation*. This command needs the Euler Axis x, y, and z components, followed by the Euler Angle. Given this information in that order, your objects will rotate properly.

Another important lesson learned is how to add a second object. The code that performs this task is shown below in Figure 22:

```
# Creates the text field for entering the object names
# region1sc_Spring2.wrl
$this newPortText targName1
$this targName1 setLabel "Name of HCW SC:"
$this targName1 setValue "region1sc_Spring2.wrl"

# Creates the text field for entering the object names
# region1sc_Spring09.wrl
$this newPortText targName2
$this targName2 setLabel "Name of Orbiting SC:"
$this targName2 setValue "region1sc_Spring09.wrl"
```

Figure 22: Two objects are being read into the tcl file, and can now be used by Avizo.

These lines set the value of the file *region1sc_Spring2.wrl*, to the name *targName1*. *targName1* is then given the label *Name of HCW SC*. The same process is used for importing the Reference Spacecraft. These names are then passed into the procedure *get_sim_data*, where the objects are assigned their respective translational and rotational states.

The final key lesson learned when working with this file was the ins and outs of how the "Retrieve Data" button works. When this button is clicked within the Avizo window, the following loop is executed:

```
# Retrieve Data Button
$this newPortButtonList openSocket 1
$this openSocket setLabel "Connection:"
$this openSocket setLabel 0 "Roll Train"
$this openSocket setCmd 0 {

    set timecall [$this cycle_rate getValue]
    global sat
    global timeInt
    global xPositionT1
    global yPositionT1
    global zPositionT1
    global xPositionT2
    global yPositionT2
    global zPositionT2
    global xAxisT1
    global yAxisT1
    global zAxisT1
    global xAxisT2
    global yAxisT2
    global zAxisT2
    global ThetaT1
    global ThetaT2

    # connect to server
    set sat(socket) [simcom_connect [$this ipAddress getValue] [$this portNumber getValue]]

    get_sim_data $timecall [$this targName1 getValue] [$this targName2 getValue]

    $this numberSteps setValue $timeInt(max)

    $this time setMinMax 1 $timeInt(max)
}
```

Figure 23: This code is executed when the button "Roll Train" is clicked.

The important lines of this procedure are the parts where Avizo connects to the Trick server, outlined in red. The first line creates the socket called *sat*. It does this by using the *simcom_connect* procedure and inputting the *ipAddress* and *portNumber* listed earlier in the code. The variable *sat* is then set to be a socket listening on the specified channel. Once the connection is established, *get_sim_data* is called. To used this procedure, send in the two objects named *targName1* and *targName2*. At this point, the code within *get_sim_data*, which is outlined earlier in this Appendix, does its job to retrieve specific values from Trick and assign them to the correct objects in Avizo.

Other miscellaneous lessons were learned as well. For example, in the '.hx' file, or the Avizo project file, it is impossible to scale objects to be any smaller than 1e-3. If this is attempted, Avizo will give an error and truncate the scale factor at 1e-3. However, it is possible to bypass this issue by going into the object file, or '.wrl' file itself and modifying the scale factors within the file. They are not easy to find, but a search for the word "scale" usually leads to the correct location in the file. For example, the team modified the *Earth.wrl* file so that instead of having a radius of 6371, the object has a radius of 0.006371. Such a change is equivalent to scaling the object by 1e-6 in the project file.

Summary

Experiment 1 was set up to measure the difference in position between the position of Displaced Spacecraft using the inverse square model and the position of Displaced Spacecraft using the HCW model. After that, only the HCW equations model the translational motion of the Displaced Spacecraft. Gravity moments are also incorporated which affect the attitudes of both spacecraft, making the transition from a system of particles to ta system of rigid bodies, enabling us to analyze the stability of the spacecraft. The thrust effectors have also been implemented with an LQR guiding the Displaced Spacecraft to rendezvous with the Reference Spacecraft. Rendezvous is the default final state, but the user can modify the final state in real time with a GUI.

Graphics are added in Avizo with two views; one showing the positions of the spacecraft relative to Earth, and another showing Displaced Spacecraft relative to Reference Spacecraft. The simulation runs with graphics in real time, giving an accurate visual of the simulation's representation of the motion.

User Guide: Staged Controller Rendezvous

To run the staged rendezvous simulation, make sure Xming is running and have a UNIX window open. After obtaining the model and simulation files from SVN, compile the code with the "CP" command while in the *SIM_satellite* folder. Next follow the proceeding steps if viewing the results with stripcharts:

- While in the *SIM_satellite* folder, type **S_main_Linux_4.4_212_x86_64.exe RUN_test/input2 &** to run the simulation.
- While in the same folder, type **./StagedGUI.tcl 7002 &** to open the GUI. "7002" corresponds to the port number found from the simulation control panel.
- After starting the simulation, enact the first stage by clicking the first button.
- Use the stripchart to see when the Displaced Spacecraft reaches the vicinity of the first and then orient the spacecraft.
- After Stabilizing, click the final button to start the final stage towards rendezvous.

If running the demonstration with Avizo, follow these steps:

- While in the *SIM_satellite* folder, type **S_main_Linux_4.4_212_x86_64.exe RUN_test/input &** to run the simulation.
- While in the same folder, type **./StagedGUI.tcl 7002 &** to open the GUI. "7002" corresponds to the port number found from the simulation control panel.
- In Avizo, click the restart button and input the correct IP address and port number.

    - The port number is found from the simulation control panel and the IP address can be found by typing **nslookup gus** where "gus" corresponds to the server you are running on.
- In Avizo, click the "Connect to Train" button to connect Trick with Avizo.
- After starting the simulation, enact the first stage by clicking the first button.
- Use the stripchart to see when the Displaced Spacecraft reaches the vicinity of the first and then orient the spacecraft.
- After Stabilizing, click the final button to start the final stage towards rendezvous.

Future Work

Future tasks tentatively include improving the human-computer interaction with the simulation, optimizing the LQR to minimize Δv, and completing the thrusters model for the spacecraft.

In order to improve the human-computer interaction, the team will develop the graphics further to have a view fixed in the Displaced Spacecraft as the user guides it to the desired target and orientation in space. The ability to control attitude will also be added to the LQR. Incorporating the 3dconnexion Space Navigator 6DOF mouse with Trick will allow the user to more intuitively and fluidly change the target state.

Appendix 2: Finding initial conditions required for rendezvous

One of our first tests with the Hill-Clohessy-Wiltshire (HCW) model was to find the initial conditions that lead to rendezvous. The reason for this was to see if the equations operate how we expect them to as well as to provide a useful trajectory for visualization in Avizo.

While the equations of motion (EOMs) are propagated forward in time within Trick, the easiest way to find the required initial conditions that lead to rendezvous was to use MATLAB as one simply must use a decreasing time span to integrate backwards.

The first step in finding the initial conditions that lead to rendezvous is defining exactly what rendezvous entails. First, the final relative position vector, $\overline{x_f}$, is set equal to zero so that the two spacecraft physically meet up. If the final relative velocity vector is also set to zero; however, we will be unable to propagate the equations as the acceleration would remain constant at zero. Also, since a slow approach is desired instead of a hard collision, the final relative velocity vector must consist of small values. A 10 cm/s approach in the inertial x direction was found to be sufficiently small for a slow approach that also allows the user to see the trajectory progress in

a reasonable amount of time. Lastly, the time of flight was set to one hour to give enough time for the small final conditions to grow for more useful visualizations.

After setting up the initial conditions and relative motion equations discussed previously, the required initial conditions are easily found by running time backwards. As an example, with the initial and final conditions described above and a radial distance defined by Equation 1 below with h = 1000 km,

$$r = R + h \tag{42}$$

the initial conditions that lead to rendezvous are:

$$X_f = 43.59\hat{x} + 381.0\hat{y} \tag{43}$$

$$V_f = -0.09005\hat{x} - 0.08699\hat{y} \tag{44}$$

Where $\hat{x}$, is defined along the radius vector and $\hat{y}$ is orthogonal to $\hat{x}$.

These initial conditions can then be input into the Trick model in order to produce rendezvous. Figures 1 and 2 show the relative position trajectory and the relative velocity time history, respectively.

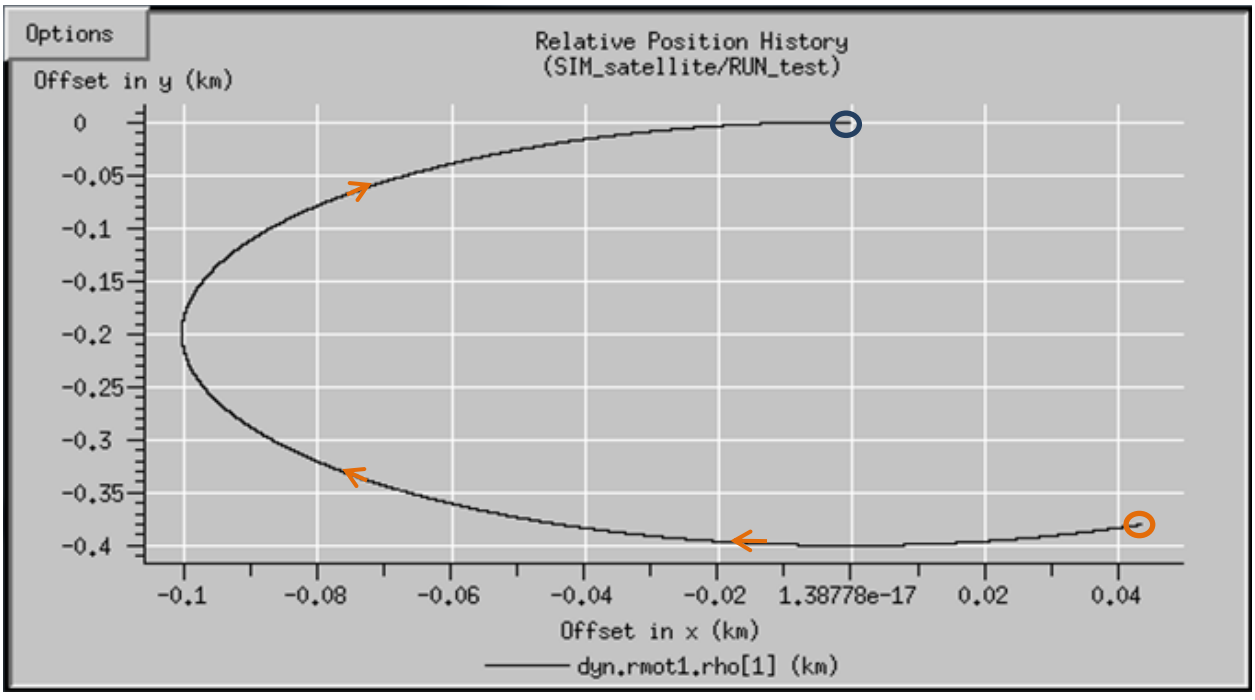


Figure 24: Trajectory of Displaced Spacecraft leading to rendezvous.

Figure 1 above shows the rendezvous with the conditions described previously. The HCW equations use a reference frame defined in the orbit of the Reference Spacecraft in circular orbit with the origin fixed to the position of the Reference Spacecraft. Therefore, the blue circle at the origin indicates the position of said spacecraft. The orange circle then indicates the initial location of the Displaced Spacecraft. The black line then represents the path the Displaced Spacecraft takes over the course of an hour. As expected, the second spacecraft moves in a path that leads to rendezvous without any outside forces guiding it.

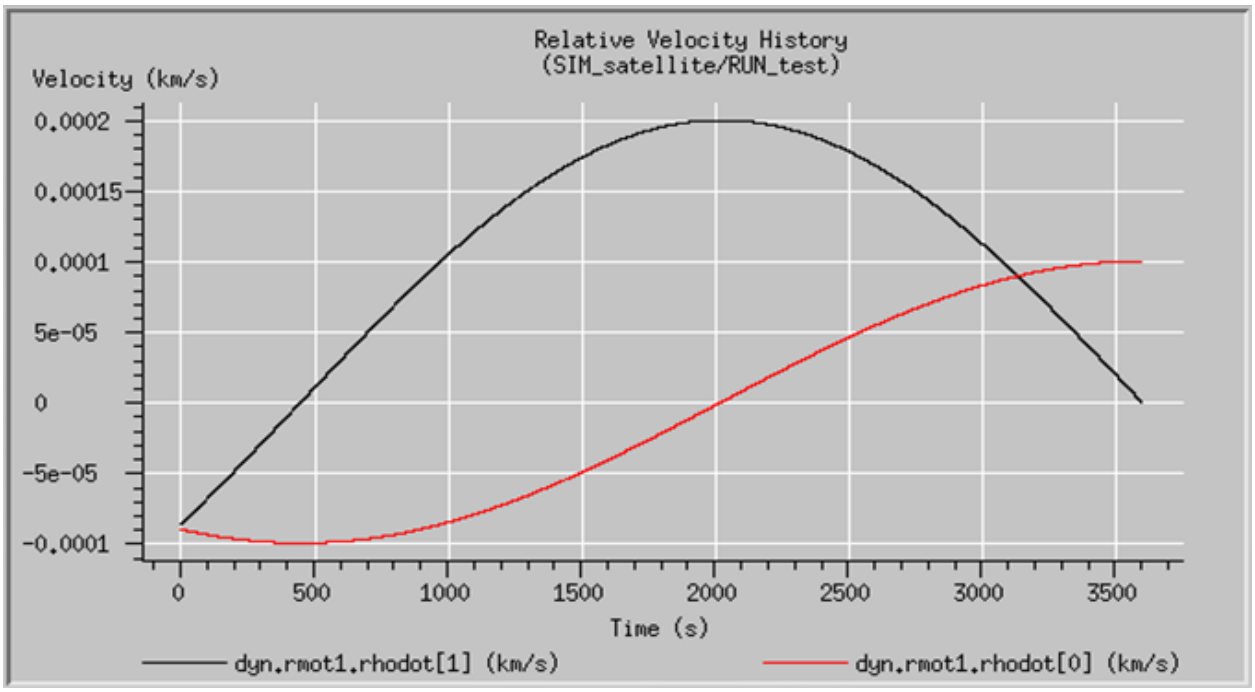


Figure 25: Relative velocity time history leading to rendezvous between two spacecraft. The red line represents the velocity in the radial direction of the first orbit while the black line represents the orthogonal velocity.

Figure 2 above shows the relative velocity time history with the red line showing the velocity in the $\hat{x}$ direction and the black line showing the velocity in the $\hat{y}$ direction. The initial velocities are found on the left endpoints and then change according to the HCW equations. Upon rendezvous, the velocities are at the right endpoints and are equal to the values defined previously of 0 cm/s and 10 cm/s, respectively.

As a whole, this experiment demonstrates that the HCW equations can be used to model rendezvous. In addition, the initial conditions that are required for such a feat if no control is available and there are no outside forces acting on the spacecraft are presented.

Appendix 3: Human in the loop as an external force – Cannon Example

In the controller discussed previously, a variable used in the EOMs was modified directly. Because of this, only a Tcl file was needed while no modifications to the model or simulation files were required. However, if external forces are added to the system, a different method is required which requires modifications to the original files.

As a demonstration of this method, a cannonball being shot from the ground with some initial conditions was modeled. After the initial shot, there are at most two forces acting on the ball: the force of gravity and a force applied from a thruster controlled by the user.

To start, a modular approach is taken in that each force is found in its own file. The code segments which calculate the acceleration due to gravity and the acceleration due to an external thruster are presented in Figures 1 and 2, respectively.

```
if ( ! C->impact ) {
    /* Still above ground and flying */
    C->acc_grav[0] = 0.0;
    C->acc_grav[1] = -9.81;
}
```

Figure 26: Code segment that sets the acceleration due to gravity. Found within cannon_deriv_gravity_impact.c.

Since gravity is always acting on the body, setting its impact on the cannonball is done by simply setting the x-component of gravity equal to 0 m/s^2 and the y-component equal to -9.81 m/s^2.

```
C->acc_jet[0] = 0.0;
C->acc_jet[1] = 0.0;

if ( ! C->impact ) {
    if (C->jet_on[0]) {
        C->acc_jet[0] = C->acc_jet_pos[0];
        C->jet_on[0] = 0;
    }

    if (C->jet_on[1]) {
            •
            •
            •
```

Figure 27: Code segment that sets the acceleration from the thruster if the button has been clicked. This is for the "+X" thruster only; the other buttons are nearly identical and are found within cannon_acc_jet.c.

The acceleration due to the thruster being on or off is slightly more complicated. First, the acceleration is set equal to 0 m/s^2 since the thruster is on for a limited amount of time when the

button is clicked. After using dynamic events to check if the ball has hit the ground or not, the program checks if the button has been pressed. The jet_on[4] variable acts as a Boolean in that it equals 0 when the thruster is off and 1 when the thruster is on. This is done in the Tcl file similar to how variables were changed directly in the other GUI. If the thruster is on, it sets the acceleration equal to the acceleration that the thruster can provide, as defined in the definitions file, and then turns the Boolean back to false. Otherwise, the acceleration remains equal to 0 m/s^2 and the simulation progresses.

The S_define file is responsible for keeping track of the separate forces and then calling the collector used to join the accelerations together. Figure 3 below shows the parts of the S_define that perform this action.

```
  (derivative) cannon_jet/gravity:
    cannon_deriv_gravity_impact( CANNON* C = &dyn.cannon ) ;

  (0.5, effector) cannon_jet/gravity:
          cannon_acc_jet( CANNON* C = &dyn.cannon) ;

        (derivative) cannon_jet/gravity:
      cannon_collect_acc(CANNON* C = &dyn.cannon );

  (integration) cannon_jet/gravity:
    cannon_integ(
      INTEGRATOR* I = &dyn.integ,
      CANNON* C = &dyn.cannon ) ;
  (dynamic_event) cannon_jet/gravity:
      cannon_impact(
          CANNON* C       = &dyn.cannon,
          double* time        = &sys.exec.work.integ_time,
          int *event_evaluate_tgo = &sys.exec.work.event_evaluate_tgo ) ;
} dyn ;

collect dyn.cannon.acc_collect = {
        dyn.cannon.acc_grav[0],
        dyn.cannon.acc_jet[0]} ;

integrate (0.01) dyn ;
```

Figure 28: Modifications to the S_define file that allow for additional forces to the system.

The red box contains the acceleration due to gravity while the blue box contains the acceleration from the thrusters. An effector class job is used for the thruster because it is not part of the physical system. Instead it is a force controlled by the user and therefore a job must be scheduled to account for it appropriately. This can be done with a derivative class job, but is more intuitive this way. The “0.5” here indicates that the thruster will fire for half second intervals.

The acceleration components are then joined using the collect statement shown in the green boxes. The acc_collect variable within the cannon structure is a multidimensional array. Each row of the array consists of the acceleration components from an individual source with each column containing the accelerations for each respective direction. These are sent to the collect file shown in Figure 4.

```
#include "../include/cannon.h"
#include "sim_services/include/collect_macros.h"

int cannon_collect_acc(CANNON* C)
{

    double **collected_acc;
    int ii;

    collected_acc = (double**)(C->acc_collect);
    C->acc[0] = 0.0;
    C->acc[1] = 0.0;
    for(ii=0; ii<NUM_COLLECT(collected_acc); ii++) {
        C->acc[0] += collected_acc[ii][0];
        C->acc[1] += collected_acc[ii][1];
    }
    return 0;
}
```

Figure 29: Code segment that joins the separate sources of acceleration into one variable. Found in cannon_collect_acc.c.

The multidimensional array, 2-dimensional in this example, is sent to this collect file which uses the collect macros. In the for-loop inside the red box, the individual components are extracted from the multidimensional array, added together, and then stored in the acc variable used in integration.

If additional forces are to be added, such as drag, one simply needs to create another file that sets its magnitude and direction along with an additional line to the collect statement. The code is modularized in this way which makes it great for expansion and testing the effects of individual parts of the model.

Appendix 4: Code

Attitude Header File

satellite_sim/satellite/attitude/include/attitude.h

```
/*****************************************************************************
PURPOSE:        (Attitude for Satellite Attitude Equations Header)
*****************************************************************************/

#ifndef _attitude_h_
#define _attitude_h_
#define PI 3.14159

typedef struct
{
        double DCM[3][3];       /* *i -- Direction Cosine Matrix */
        double q[4];            /* -- Quaternion */
        double qdot[4];         /* -- Quaternion rate */
        double w0[3];           /* *i r/s Initial angular velocity */
        double w[3];            /* r/s Angular velocity */
        double wdot[3];         /* r/s2 Angular velocity rate */
        double J  ;             /* *i kg*M2 Mass-moment of inertia about symmetric axis*/
        double I  ;             /* *i kg*M2 Mass-moment of inertia about nonsymmetric axes*/
        double u0[3];           /* *i N*M Applied torques */
        double u[3];            /* N*M Applied torques */
        double k;               /* -- Magnitude of quat vec */
        double ehat[3];         /* -- Euler axis */
        double Phi;                     /* d Euler anlge */
        double emag;            /* -- Magnitude of Euler axis */
        double s;               /* r/s Angular velocity of B frame in C frame */
        double G[3][11];        /* -- Gain */
        double Gvals[3][11];            /* -- Gain Values */
        double m[3];                    /* -- Control vector */
        double Int[4];                  /* -- adsf */
        double qd[4];                   /* -- Desired quaternion */
        double Espin;                   /* -- Earth Rotation */
        double S2flag;          /* -- Stage 2 Flag */
} ROT;

#endif
```

Attitude Default Data File 1

satellite_sim/satellite/attitude/include/attitude1.d

```
/* Default data for Rotational Equation */

ROT.ehat[0] = 1;
ROT.ehat[1] = 0;
ROT.ehat[2] = 0;
ROT.Phi = 0; //degrees

/* Initial angular velocity */
ROT.w0[0] {r/s} = -.1;
ROT.w0[1] {r/s} = .2;
ROT.w0[2] {r/s} = .3;

/* Moments of inertial about body axes (assumed as principle axes) */
ROT.I {kg*M2} = 6;
ROT.J {kg*M2} = 1.5;

/* Applied Torques (currently torque-free) */
ROT.u0[0] {N*M} = 0.0;
ROT.u0[1] {N*M} = 0.0;
ROT.u0[2] {N*M} = 0.0;

/* Rate of spin of B in C frame */
ROT.s {r/s} = 0;

/* Desired Quaternions */
ROT.qd[0] = 0;
ROT.qd[1] = sqrt(2)/2;
ROT.qd[2] = 0;
ROT.qd[3] = sqrt(2)/2;

ROT.S2flag = 0;

ROT.G[0][0] {--} = 0;
ROT.G[0][1] {--} = 0;
ROT.G[0][2] {--} = 0;
ROT.G[0][3] {--} = 0;
ROT.G[0][4] {--} = 0;
ROT.G[0][5] {--} = 0;
ROT.G[0][6] {--} = 0;
ROT.G[0][7] {--} = 0;
ROT.G[0][8] {--} = 0;
ROT.G[0][9] {--} = 0;
ROT.G[0][10] {--} = 0;
ROT.G[1][0] {--} = 0;
ROT.G[1][1] {--} = 0;
ROT.G[1][2] {--} = 0;
ROT.G[1][3] {--} = 0;
ROT.G[1][4] {--} = 0;
ROT.G[1][5] {--} = 0;
ROT.G[1][6] {--} = 0;
ROT.G[1][7] {--} = 0;
ROT.G[1][8] {--} = 0;
ROT.G[1][9] {--} = 0;
ROT.G[1][10] {--} = 0;
ROT.G[2][0] {--} = 0;
ROT.G[2][1] {--} = 0;
```

```
ROT.G[2][2] {--} = 0;
ROT.G[2][3] {--} = 0;
ROT.G[2][4] {--} = 0;
ROT.G[2][5] {--} = 0;
ROT.G[2][6] {--} = 0;
ROT.G[2][7] {--} = 0;
ROT.G[2][8] {--} = 0;
ROT.G[2][9] {--} = 0;
ROT.G[2][10] {--} = 0;

ROT.Gvals[0][0] {--} = 4.153;
ROT.Gvals[0][1] {--} = -.001;
ROT.Gvals[0][2] {--} = -4.153;
ROT.Gvals[0][3] {--} = -.001;
ROT.Gvals[0][4] {--} = 3.297;
ROT.Gvals[0][5] {--} = 0;
ROT.Gvals[0][6] {--} = 0;
ROT.Gvals[0][7] {--} = 1.581;
ROT.Gvals[0][8] {--} = 0;
ROT.Gvals[0][9] {--} = -1.581;
ROT.Gvals[0][10] {--} = 0;
ROT.Gvals[1][0] {--} = -.0016;
ROT.Gvals[1][1] {--} = 4.153;
ROT.Gvals[1][2] {--} = -.0016;
ROT.Gvals[1][3] {--} = -4.153;
ROT.Gvals[1][4] {--} = 0;
ROT.Gvals[1][5] {--} = 3.297;
ROT.Gvals[1][6] {--} = .0004;
ROT.Gvals[1][7] {--} = .0015;
ROT.Gvals[1][8] {--} = 1.581;
ROT.Gvals[1][9] {--} = .0015;
ROT.Gvals[1][10] {--} = -1.581;
ROT.Gvals[2][0] {--} = 4.153;
ROT.Gvals[2][1] {--} = .001;
ROT.Gvals[2][2] {--} = 4.153;
ROT.Gvals[2][3] {--} = -.0024;
ROT.Gvals[2][4] {--} = 0;
ROT.Gvals[2][5] {--} = .0004;
ROT.Gvals[2][6] {--} = 3.297;
ROT.Gvals[2][7] {--} = 1.581;
ROT.Gvals[2][8] {--} = -.0015;
ROT.Gvals[2][9] {--} = 1.581;
ROT.Gvals[2][10] {--} = .0015;
```

Attitude Default Data File 2

satellite_sim/satellite/attitude/include/attitude2.d

```
/* Default data for Rotational Equation */

ROT.ehat[0] = 1;
ROT.ehat[1] = 0;
ROT.ehat[2] = 0;
ROT.Phi = 0; //degrees

/* Initial angular velocity */
ROT.w0[0] {r/s} = .2;
ROT.w0[1] {r/s} = -.1;
ROT.w0[2] {r/s} = .1;

/* Moments of inertial about body axes (assumed as principle axes) */
ROT.I {kg*M2} = 6;
ROT.J {kg*M2} = 1.5;

/* Applied Torques (currently torque-free) */
ROT.u0[0] {N*M} = 0.0;
ROT.u0[1] {N*M} = 0.0;
ROT.u0[2] {N*M} = 0.0;

/* Rate of spin of B in C frame */
ROT.s {r/s} = 0;

/* Desired Quaternions */
ROT.qd[0] = 0;
ROT.qd[1] = sqrt(2)/2;
ROT.qd[2] = 0;
ROT.qd[3] = sqrt(2)/2;

ROT.G[0][0] {--} = 0;
ROT.G[0][1] {--} = 0;
ROT.G[0][2] {--} = 0;
ROT.G[0][3] {--} = 0;
ROT.G[0][4] {--} = 0;
ROT.G[0][5] {--} = 0;
ROT.G[0][6] {--} = 0;
ROT.G[0][7] {--} = 0;
ROT.G[0][8] {--} = 0;
ROT.G[0][9] {--} = 0;
ROT.G[0][10] {--} = 0;
ROT.G[1][0] {--} = 0;
ROT.G[1][1] {--} = 0;
ROT.G[1][2] {--} = 0;
ROT.G[1][3] {--} = 0;
ROT.G[1][4] {--} = 0;
ROT.G[1][5] {--} = 0;
ROT.G[1][6] {--} = 0;
ROT.G[1][7] {--} = 0;
ROT.G[1][8] {--} = 0;
ROT.G[1][9] {--} = 0;
ROT.G[1][10] {--} = 0;
ROT.G[2][0] {--} = 0;
ROT.G[2][1] {--} = 0;
ROT.G[2][2] {--} = 0;
ROT.G[2][3] {--} = 0;
```

```
ROT.G[2][4] {--} = 0;
ROT.G[2][5] {--} = 0;
ROT.G[2][6] {--} = 0;
ROT.G[2][7] {--} = 0;
ROT.G[2][8] {--} = 0;
ROT.G[2][9] {--} = 0;
ROT.G[2][10] {--} = 0;

ROT.Gvals[0][0] {--} = 4.153;
ROT.Gvals[0][1] {--} = -.001;
ROT.Gvals[0][2] {--} = -4.153;
ROT.Gvals[0][3] {--} = -.001;
ROT.Gvals[0][4] {--} = 3.297;
ROT.Gvals[0][5] {--} = 0;
ROT.Gvals[0][6] {--} = 0;
ROT.Gvals[0][7] {--} = 1.581;
ROT.Gvals[0][8] {--} = 0;
ROT.Gvals[0][9] {--} = -1.581;
ROT.Gvals[0][10] {--} = 0;
ROT.Gvals[1][0] {--} = -.0016;
ROT.Gvals[1][1] {--} = 4.153;
ROT.Gvals[1][2] {--} = -.0016;
ROT.Gvals[1][3] {--} = -4.153;
ROT.Gvals[1][4] {--} = 0;
ROT.Gvals[1][5] {--} = 3.297;
ROT.Gvals[1][6] {--} = .0004;
ROT.Gvals[1][7] {--} = .0015;
ROT.Gvals[1][8] {--} = 1.581;
ROT.Gvals[1][9] {--} = .0015;
ROT.Gvals[1][10] {--} = -1.581;
ROT.Gvals[2][0] {--} = 4.153;
ROT.Gvals[2][1] {--} = .001;
ROT.Gvals[2][2] {--} = 4.153;
ROT.Gvals[2][3] {--} = -.0024;
ROT.Gvals[2][4] {--} = 0;
ROT.Gvals[2][5] {--} = .0004;
ROT.Gvals[2][6] {--} = 3.297;
ROT.Gvals[2][7] {--} = 1.581;
ROT.Gvals[2][8] {--} = -.0015;
ROT.Gvals[2][9] {--} = 1.581;
ROT.Gvals[2][10] {--} = .0015;
```

Attitude Integration Default Data File

satellite_sim/satellite/attitude/include/attitude_integ.d

```
#define NUM_STEP 12
#define NUM_VARIABLES 7

INTEGRATOR.state = alloc(NUM_VARIABLES) ;
INTEGRATOR.deriv = alloc(NUM_STEP) ;
INTEGRATOR.state_ws = alloc(NUM_STEP) ;

for (int kk = 0 ; kk < NUM_STEP ; kk++ ) {
        INTEGRATOR.deriv[kk] = alloc(NUM_VARIABLES) ;
        INTEGRATOR.state_ws[kk] = alloc(NUM_VARIABLES) ;
}

INTEGRATOR.stored_data = alloc(8) ;
for (int kk = 0 ; kk < 8 ; kk++) {
        INTEGRATOR.stored_data[kk] = alloc(NUM_VARIABLES) ;
}

INTEGRATOR.option = Runge_Kutta_4 ;
INTEGRATOR.init = True ;
INTEGRATOR.first_step_deriv = Yes ;
INTEGRATOR.num_state = NUM_VARIABLES ;

#undef NUM_STEP
#undef NUM_VARIABLES
```

Attitude Initialization File

satellite_sim/satellite/attitude/src/attitude_init.c

```
/*********************************************************
PURPOSE:      (Initialize Attitude)
*********************************************************/

#include <stdio.h>
#include <math.h>
#include "../include/attitude.h"
#include "../../orbit/include/orbit.h"
#include "trick_utils/math/include/trick_math.h"

int attitude_init( ROT* R, double rmag )
{
double q0[4];
double OMEGA = sqrt(MU/pow(rmag,3));

R->emag = sqrt(pow(R->ehat[0],2) + pow(R->ehat[1],2) + pow(R->ehat[2],2));

// Calculate initial quaternion (q0)
q0[3] = cos(0.5*R->Phi);
q0[0] = R->ehat[0]*sin(0.5*R->Phi);
q0[1] = R->ehat[1]*sin(0.5*R->Phi);
q0[2] = R->ehat[2]*sin(0.5*R->Phi);

/* Set initial quaternion */
R->q[0] = q0[0];
R->q[1] = q0[1];
R->q[2] = q0[2];
R->q[3] = q0[3];
R->k = 1;

/* Set initial angular velocity */
R->w[0] = -OMEGA;
R->w[1] = R->w0[1];
R->w[2] = R->w0[2];

/* Set applied constant torques */
R->u[0] = R->u0[0];
R->u[1] = R->u0[1];
R->u[2] = R->u0[2];

R->m[0] = 0;
R->m[1] = 0;
R->m[2] = 0;

R->Int[0] = 0;
R->Int[1] = 0;
R->Int[2] = 0;
R->Int[3] = 0;

R->Espin = 245.0;

return (0);
}
```

Attitude Derivative File

satellite_sim/satellite/attitude/src/attitude_deriv.c

```
/*************************************************************
PURPOSE:      (Attitude equations of motion)
*************************************************************/
#include "../include/attitude.h"
#include "../../orbit/include/orbit.h"
#include <math.h>
#include <stdio.h>
#include "trick_utils/math/include/trick_math.h"

int attitude_deriv(  ROT* R, double rmag)
{
        double OMEGA = sqrt(MU/pow(rmag,3));
        double X = 1 - R->J/R->I;

        if (R->S2flag) {
                R->G[0][0] = R->Gvals[0][0];
                R->G[0][1] = R->Gvals[0][1];
                R->G[0][2] = R->Gvals[0][2];
                R->G[0][3] = R->Gvals[0][3];
                R->G[0][4] = R->Gvals[0][4];
                R->G[0][5] = R->Gvals[0][5];
                R->G[0][6] = R->Gvals[0][6];
                R->G[0][7] = R->Gvals[0][7];
                R->G[0][8] = R->Gvals[0][8];
                R->G[0][9] = R->Gvals[0][9];
                R->G[0][10] = R->Gvals[0][10];
                R->G[1][0] = R->Gvals[1][0];
                R->G[1][1] = R->Gvals[1][1];
                R->G[1][2] = R->Gvals[1][2];
                R->G[1][3] = R->Gvals[1][3];
                R->G[1][4] = R->Gvals[1][4];
                R->G[1][5] = R->Gvals[1][5];
                R->G[1][6] = R->Gvals[1][6];
                R->G[1][7] = R->Gvals[1][7];
                R->G[1][8] = R->Gvals[1][8];
                R->G[1][9] = R->Gvals[1][9];
                R->G[1][10] = R->Gvals[1][10];
                R->G[2][0] = R->Gvals[2][0];
                R->G[2][1] = R->Gvals[2][1];
                R->G[2][2] = R->Gvals[2][2];
                R->G[2][3] = R->Gvals[2][3];
                R->G[2][4] = R->Gvals[2][4];
                R->G[2][5] = R->Gvals[2][5];
                R->G[2][6] = R->Gvals[2][6];
                R->G[2][7] = R->Gvals[2][7];
                R->G[2][8] = R->Gvals[2][8];
                R->G[2][9] = R->Gvals[2][9];
                R->G[2][10] = R->Gvals[2][10];
        }

        double dt = .01;
        R->Int[0] += (R->q[0] - R->qd[0])*dt;
        R->Int[1] += (R->q[1] - R->qd[1])*dt;
        R->Int[2] += (R->q[2] - R->qd[2])*dt;
        R->Int[3] += (R->q[3] - R->qd[3])*dt;
```

```
	R->m[0] = -R->G[0][0]*R->q[0] - R->G[0][1]*R->q[1] - R->G[0][2]*R->q[2] - R->G[0][3]*R->q[3] - R->G[0][4]*R->w[0]
					- R->G[0][5]*R->w[1] - R->G[0][6]*R->w[2] - R->G[0][7]*R->Int[0] - R->G[0][8]*R->Int[1] - R->G[0][9]*R->Int[2] -
						R->G[0][10]*R->Int[3];
	R->m[1] = -R->G[1][0]*R->q[0] - R->G[1][1]*R->q[1] - R->G[1][2]*R->q[2] - R->G[1][3]*R->q[3] - R->G[1][4]*R->w[0]
					- R->G[1][5]*R->w[1] - R->G[1][6]*R->w[2] - R->G[1][7]*R->Int[0] - R->G[1][8]*R->Int[1] - R->G[1][9]*R->Int[2] -
						R->G[1][10]*R->Int[3];
	R->m[2] = -R->G[2][0]*R->q[0] - R->G[2][1]*R->q[1] - R->G[2][2]*R->q[2] - R->G[2][3]*R->q[3] - R->G[2][4]*R->w[0]
					- R->G[2][5]*R->w[1] - R->G[2][6]*R->w[2] - R->G[2][7]*R->Int[0] - R->G[2][8]*R->Int[1] - R->G[2][9]*R->Int[2] -
						R->G[2][10]*R->Int[3];

	R->wdot[0] = 12*pow(OMEGA,2)*X*(R->q[0]*R->q[1] - R->q[2]*R->q[3])*(R->q[0]*R->q[2] + R->q[1]*R->q[3])
			- X*R->w[1]*R->w[2] + R->m[0];
	R->wdot[1] = -6*pow(OMEGA,2)*X*(1 - 2*pow(R->q[1],2) - 2*pow(R->q[2],2))*(R->q[0]*R->q[2] + R->q[1]*R->q[3])
			+ X*R->w[2]*R->w[0] + R->m[1];
	R->wdot[2] = R->m[2];

	R->qdot[0] = 0.5*(R->q[3]*R->w[0] + R->q[1]*(R->w[2] + OMEGA) - R->q[2]*R->w[1]);
	R->qdot[1] = 0.5*(R->q[2]*R->w[0] + R->q[3]*R->w[1] - R->q[0]*(R->w[2] + OMEGA));
	R->qdot[2] = 0.5*(-R->q[1]*R->w[0] + R->q[0]*R->w[1] + R->q[3]*(R->w[2] - OMEGA));
	R->qdot[3] = 0.5*(-R->q[0]*R->w[0] - R->q[1]*R->w[1] - R->q[2]*(R->w[2] - OMEGA));

	R->k = 1 - sqrt(pow(R->q[0],2) + pow(R->q[1],2) + pow(R->q[2],2) + pow(R->q[3],2));

	/* Set euler angles */
	R->ehat[0] = R->q[0]/sqrt(1-pow(R->q[3],2));
	R->ehat[1] = R->q[1]/sqrt(1-pow(R->q[3],2));
	R->ehat[2] = R->q[2]/sqrt(1-pow(R->q[3],2));
	R->emag = sqrt(pow(R->ehat[0],2) + pow(R->ehat[1],2) + pow(R->ehat[2],2));
	R->Phi = 2*acos(R->q[3])*180/PI; //degrees

	R->Espin += 360/3600/24*dt;

	return (0);
}
```

Attitude Integration File

satellite_sim/satellite/attitude/src/attitude_integ.c

```
/*************************************************************
PURPOSE:      (Attitude state integration)
*************************************************************/
#include "sim_services/include/integrator.h"
extern void integrate(INTEGRATOR *I);
#include "../include/attitude.h"

int attitude_integ(
                    INTEGRATOR *I,
                    ROT* R )
{
        /* Load current states */
        I->state[0] = R->q[0];
        I->state[1] = R->q[1];
        I->state[2] = R->q[2];
        I->state[3] = R->q[3];
        I->state[4] = R->w[0];
        I->state[5] = R->w[1];
        I->state[6] = R->w[2];

        /* Load derivatives */
        I->deriv[I->intermediate_step][0] = R->qdot[0] ;
        I->deriv[I->intermediate_step][1] = R->qdot[1] ;
        I->deriv[I->intermediate_step][2] = R->qdot[2] ;
   I->deriv[I->intermediate_step][3] = R->qdot[3] ;
        I->deriv[I->intermediate_step][4] = R->wdot[0] ;
        I->deriv[I->intermediate_step][5] = R->wdot[1] ;
   I->deriv[I->intermediate_step][6] = R->wdot[2] ;

        /* Call the Trick integrator */
        integrate( I ) ;

        /* Unload new states */
        R->q[0] = I->state_ws[I->intermediate_step][0] ;
        R->q[1] = I->state_ws[I->intermediate_step][1] ;
        R->q[2] = I->state_ws[I->intermediate_step][2] ;
        R->q[3] = I->state_ws[I->intermediate_step][3] ;
        R->w[0] = I->state_ws[I->intermediate_step][4] ;
        R->w[1] = I->state_ws[I->intermediate_step][5] ;
        R->w[2] = I->state_ws[I->intermediate_step][6] ;

        /* Return step. Used for multi-pass integration */
        return ( I->intermediate_step ) ;
}
```

Orbit Header File

satellite_sim/satellite/orbit/include/orbit.h

```
/*****************************************************************************
PURPOSE:        (Orbit Equations Header)
*****************************************************************************/

#ifndef _orbit_h_
#define _orbit_h_
#define R_EARTH 6371.0 /* km */
#define MU 398600.4418 /* km3/s3 */

typedef struct ORB
{
        double r[3];                    /* km  Inertial Frame Position*/
        double rdot[3];                 /* km/s Velocity */
        double rddot[3];    /* km/s2 Acceleration */
        double r0[3];                   /* *i km Initial Position */
        double rdot0[3];    /* *i km/s Initial Velocity */
        double rmag;                    /* km Position Magnitude */
} ORB;

#endif
```

Orbit Default Data File

satellite_sim/satellite/orbit/include/orbit1.d

```
/* Default data for Satellite Equations */
#define R_EARTH 6371.0 /* km */
#define MU 398600.4418 /* km3/s3 */

/* Initial Position */
// Example: ROT.DCM[0][0]     {--}     = value;
ORB.r0[0] {km} = R_EARTH+200;
ORB.r0[1] {km} = 0.0;
ORB.r0[2] {km} = 0.0;

/* Initial velocity */
ORB.rdot0[0] {km/s} = 0.0;
ORB.rdot0[1] {km/s} = sqrt(MU/(R_EARTH+1000));
ORB.rdot0[2] {km/s} = 0.0;
```

Orbit Integration Default Data File

satellite_sim/satellite/orbit/include/orbit_integ.d

```
#define NUM_STEP 12
#define NUM_VARIABLES 6

INTEGRATOR.state = alloc(NUM_VARIABLES) ;
INTEGRATOR.deriv = alloc(NUM_STEP) ;
INTEGRATOR.state_ws = alloc(NUM_STEP) ;

for (int kk = 0 ; kk < NUM_STEP ; kk++ ) {
        INTEGRATOR.deriv[kk] = alloc(NUM_VARIABLES) ;
        INTEGRATOR.state_ws[kk] = alloc(NUM_VARIABLES) ;
}

INTEGRATOR.stored_data = alloc(8) ;
for (int kk = 0 ; kk < 8 ; kk++) {
        INTEGRATOR.stored_data[kk] = alloc(NUM_VARIABLES) ;
}

INTEGRATOR.option = Runge_Kutta_4 ;
INTEGRATOR.init = True ;
INTEGRATOR.first_step_deriv = Yes ;
INTEGRATOR.num_state = NUM_VARIABLES ;

#undef NUM_STEP
#undef NUM_VARIABLES
```

Orbit Initialization File

satellite_sim/satellite/orbit/src/orbit_init.c

```
/*****************************************************
PURPOSE:      (Initialize satellite orbit)
*****************************************************/

#include <stdio.h>
#include <math.h>
#include "trick_utils/math/include/trick_math.h"
#include "../include/orbit.h"

int orbit_init( ORB* O )
{

/* Initial orbital position */
O->r[0] = O->r0[0];
O->r[1] = O->r0[1];
O->r[2] = O->r0[2];

/* Initial orbital velocity */
O->rdot[0] = O->rdot0[0];
O->rdot[1] = O->rdot0[1];
O->rdot[2] = O->rdot0[2];

/* Initial orbital position magnitude */
O->rmag = V_MAG(O->r);

return (0);
}
```

Orbit Derivative File

satellite_sim/satellite/orbit/src/orbit_deriv.c

```
/*************************************************************
PURPOSE:      (Satellite orbit equations of motion)
*************************************************************/
#include "../include/orbit.h"
#include <math.h>
#include <stdio.h>
#include "trick_utils/math/include/trick_math.h"
//asdfasdf
int orbit_deriv( ORB* O )
{
                    static double time = 0.0;
                    O->rddot[0] = -MU*O->r[0]/(pow(O->rmag,3));
                    O->rddot[1] = -MU*O->r[1]/(pow(O->rmag,3));
                    O->rddot[2] = -MU*O->r[2]/(pow(O->rmag,3));

                    return (0);
}
```

Orbit Integration File

satellite_sim/satellite/orbit/src/orbit_integ.c

```
/*************************************************************
PURPOSE:      (Satellite state integration)
*************************************************************/
#include "sim_services/include/integrator.h"
extern void integrate(INTEGRATOR *I);
#include "../include/orbit.h"

int orbit_integ(
                    INTEGRATOR *I,
                    ORB* O )
{
        /* Load current states */
        I->state[0] = O->r[0];
        I->state[1] = O->r[1];
        I->state[2] = O->r[2];
        I->state[3] = O->rdot[0];
        I->state[4] = O->rdot[1];
        I->state[5] = O->rdot[2];

        /* Load derivatives */
        I->deriv[I->intermediate_step][0] = O->rdot[0] ;
        I->deriv[I->intermediate_step][1] = O->rdot[1] ;
        I->deriv[I->intermediate_step][2] = O->rdot[2] ;
   I->deriv[I->intermediate_step][3] = O->rddot[0] ;
        I->deriv[I->intermediate_step][4] = O->rddot[1] ;
        I->deriv[I->intermediate_step][5] = O->rddot[2] ;

        /* Call the Trick integrator */
        integrate( I ) ;

        /* Unload new states */
        O->r[0] = I->state_ws[I->intermediate_step][0] ;
        O->r[1] = I->state_ws[I->intermediate_step][1] ;
        O->r[2] = I->state_ws[I->intermediate_step][2] ;
        O->rdot[0] = I->state_ws[I->intermediate_step][3] ;
        O->rdot[1] = I->state_ws[I->intermediate_step][4] ;
        O->rdot[2] = I->state_ws[I->intermediate_step][5] ;

        /* Return step. Used for multi-pass integration */
        return ( I->intermediate_step ) ;
}
```

Relative Motion Header File

satellite_sim/satellite/rmot/include/rmot.h

```
/*****************************************************************************
PURPOSE:        (Relative Motion Equations Header)
*****************************************************************************/

#ifndef _rmot_h_
#define _rmot_h_
#define R_EARTH 6371.0 /* km */
#define MU 398600.4418 /* km3/s3 */

typedef struct RMOT
{
        double rho[3];              /* km  Relative Motion Frame Position*/
        double rhodot[3];  /* km/s Velocity */
        double rho0[3];             /* *i km Initial Relative Motion Frame Position*/
        double rhodot0[3];/* *i km/s Initial Velocity */
        double rhoddot[3];/* km/s2 Acceleration */
        double rhomag;              /* km Magnitude of rel pos */
        double force[3];    /* *i N Constant forcing function */
        double rmag;                /* km Inertial Position Magnitude */
        double rho_inert[3];        /* km Inertial position coordinates */
        double Xd[2];               /* km Desired Relative Positoin */
        double K[2][6];             /* -- Controller Gains */
        double Kvals[2][6];                 /* -- Controller Gain Values */
        double I[2];                /* km*s Integral Term */
        double dT;                          /* s Time Step */
        double Idot[2];             /* -- Intergral Term Derivative */
        double S1flag;              /* -- Stage 1 Flag */
        double S3flag;              /* -- Stage 2 Flag */
} RMOT;

#endif
```

Relative Motion Default Data File
satellite_sim/satellite/rmot/include/rmot.d

```
/* Initial relative pos and vel */
RMOT.rho0[0] {km} = .02;//0.043595;
RMOT.rho0[1] {km} = -.02;//-0.380988;
RMOT.rho0[2] {km} = 0.0;

/* Initial velocity */
RMOT.rhodot0[0] {km/s} = 0.0;//9.6985207*0.001;
RMOT.rhodot0[1] {km/s} = 0.0;//-8.6985207*0.001;
RMOT.rhodot0[2] {km/s} = 0.0;

/* Force */
RMOT.force[0] {N}       = 0.0;
RMOT.force[1] {N}       = 0.0;
RMOT.force[2] {N}       = 0.0;

/* Controller Gains
RMOT.K[0][0] {--}       = 0;
RMOT.K[0][1] {--}       = 0;
RMOT.K[0][2] {--}       = 0;
RMOT.K[0][3] {--}       = 0;
RMOT.K[0][4] {--}       = 0;
RMOT.K[0][5] {--}       = 0;
RMOT.K[1][0] {--}       = 0;
RMOT.K[1][1] {--}       = 0;
RMOT.K[1][2] {--}       = 0;
RMOT.K[1][3] {--}       = 0;
RMOT.K[1][4] {--}       = 0;
RMOT.K[1][5] {--}       = 0;

RMOT.Kvals[0][0] {--}   = 0.0387;
RMOT.Kvals[0][1] {--}   = 0.0112;
RMOT.Kvals[0][2] {--}   = 0.3497;
RMOT.Kvals[0][3] {--}   = 0.0478;
RMOT.Kvals[0][4] {--}   = 0.0013;
RMOT.Kvals[0][5] {--}   = 0.0006;
RMOT.Kvals[1][0] {--}   = 0.0108;
RMOT.Kvals[1][1] {--}   = 0.0414;
RMOT.Kvals[1][2] {--}   = 0.0415;
RMOT.Kvals[1][3] {--}   = 0.3603;
RMOT.Kvals[1][4] {--}   = 0.0006;
RMOT.Kvals[1][5] {--}   = 0.0015;
*/

/* Desired Position */
RMOT.Xd[0] {km}                 = 0.008;
RMOT.Xd[1] {km}                 = 0.0;

/* Integral */
RMOT.I[0] {km*s}        = 0.0;
RMOT.I[1] {km*s}        = 0.0;

/* Time Step */
RMOT.dT {s}                     = 0.001;
```

Relative Motion Integration Default Data File
satellite_sim/satellite/rmot/include/rmot_integ.d

```
#define NUM_STEP 12
#define NUM_VARIABLES 8

INTEGRATOR.state = alloc(NUM_VARIABLES) ;
INTEGRATOR.deriv = alloc(NUM_STEP) ;
INTEGRATOR.state_ws = alloc(NUM_STEP) ;

for (int kk = 0 ; kk < NUM_STEP ; kk++ ) {
        INTEGRATOR.deriv[kk] = alloc(NUM_VARIABLES) ;
        INTEGRATOR.state_ws[kk] = alloc(NUM_VARIABLES) ;
}

INTEGRATOR.stored_data = alloc(8) ;
for (int kk = 0 ; kk < 8 ; kk++) {
        INTEGRATOR.stored_data[kk] = alloc(NUM_VARIABLES) ;
}

INTEGRATOR.option = Runge_Kutta_4 ;
INTEGRATOR.init = True ;
INTEGRATOR.first_step_deriv = Yes ;
INTEGRATOR.num_state = NUM_VARIABLES ;

#undef NUM_STEP
#undef NUM_VARIABLES
```

Relative Motion Initialization File

satellite_sim/satellite/rmot/src/rmot_init.c

```
/*****************************************************
PURPOSE:      (Initialize relative motion orbit)
*****************************************************/

#include <stdio.h>
#include <math.h>
#include "trick_utils/math/include/trick_math.h"
#include "../include/rmot.h"
#include "../../orbit/include/orbit.h"

int rmot_init( RMOT* RM, ORB* O )
{

/* Initialize relative position based on sat1 and sat2 */
RM->rho[0] = RM->rho0[0];
RM->rho[1] = RM->rho0[1];
RM->rho[2] = RM->rho0[2];
RM->rhomag = V_MAG(RM->rho);

RM->rmag = 0;

/* Initialize relative velocity based on sat1 and sat2 */
RM->rhodot[0] = RM->rhodot0[0];
RM->rhodot[1] = RM->rhodot0[1];
RM->rhodot[2] = RM->rhodot0[2];

RM->force[0] = 0.0;
RM->force[1] = 0.0;
RM->force[2] = 0.0;

return (0);
}
```

Relative Motion Derivative File

satellite_sim/satellite/rmot/src/rmot_deriv.c

```
/*********************************************************
PURPOSE:      (Relative motion equations of motion)
*********************************************************/
#include "../include/rmot.h"
#include "../../orbit/include/orbit.h"
#include <math.h>
#include <stdio.h>
#include "trick_utils/math/include/trick_math.h"

int rmot_deriv( ORB* O, RMOT* RM )
{

        if (RM->S1flag) {
                RM->K[0][0] = RM->Kvals[0][0];
                RM->K[0][1] = RM->Kvals[0][1];
                RM->K[0][2] = RM->Kvals[0][2];
                RM->K[0][3] = RM->Kvals[0][3];
                RM->K[0][4] = RM->Kvals[0][4];
                RM->K[0][5] = RM->Kvals[0][5];
                RM->K[1][0] = RM->Kvals[1][0];
                RM->K[1][1] = RM->Kvals[1][1];
                RM->K[1][2] = RM->Kvals[1][2];
                RM->K[1][3] = RM->Kvals[1][3];
                RM->K[1][4] = RM->Kvals[1][4];
                RM->K[1][5] = RM->Kvals[1][5];
        }

        if (RM->S3flag) {
                RM->Xd[0] = 0.003;
                RM->Xd[1] = 0.000;
        }

        static double time = 0.0;

        double n = sqrt(MU/pow(O->rmag,3));

        RM->I[0] = (RM->rho[0]-RM->Xd[0])*RM->dT + RM->I[0];
        RM->I[1] = (RM->rho[1]-RM->Xd[1])*RM->dT + RM->I[1];

        double X[6] = {RM->rho[0],RM->rho[1],RM->rhodot[0],RM->rhodot[1], RM->I[0], RM->I[1]};
        double Xdot[6];

        Xdot[0] = RM->rhodot[0];
        Xdot[1] = RM->rhodot[1];
        Xdot[2] = ((3*n*n)-RM->K[0][0])*RM->rho[0] - RM->K[0][1]*RM->rho[1] - RM->K[0][2]*RM-
>rhodot[0] + (2*n - RM->K[0][3])*RM->rhodot[1]
                        - RM->K[0][4]*RM->I[0] - RM->K[0][5]*RM->I[1];
        Xdot[3] = -RM->K[1][0]*RM->rho[0] - RM->K[1][1]*RM->rho[1] + (-2*n - RM->K[1][2])*RM-
>rhodot[0] + - RM->K[1][3]*RM->rhodot[1]
                        - RM->K[1][4]*RM->I[0] - RM->K[1][5]*RM->I[1];
        Xdot[4] = RM->rho[0] - RM->Xd[0];
        Xdot[5] = RM->rho[1] - RM->Xd[1];

        RM->rhodot[0] = Xdot[0];
        RM->rhodot[1] = Xdot[1];
        RM->rhoddot[0] = Xdot[2];
```

```
		RM->rhoddot[1] = Xdot[3];
		RM->Idot[0] = Xdot[4];
		RM->Idot[1] = Xdot[5];
		RM->rhoddot[2] = -pow(n,2)*RM->rho[2] + RM->force[2];
		RM->rhomag = V_MAG(RM->rho);

		RM->rmag = sqrt(pow(O->r[0] + RM->rho[0]*O->r[0]/O->rmag - RM->rho[1]*O->r[1]/O->rmag,2) +
					pow(O->r[1] + RM->rho[0]*O->r[1]/O->rmag + RM->rho[1]*O->r[0]/O-
>rmag,2));
		double PI = 3.141596;
		double alpha = 0; //rad
		double beta = 0;

		if (RM->rho[0] >= 0 && RM->rho[1] >= 0) {
			alpha = atan(abs(RM->rho[1]/RM->rho[0])); }
		else if (RM->rho[0] < 0 && RM->rho[1] > 0) {
			alpha = PI - atan(abs(RM->rho[1]/RM->rho[0])); }
		else if (RM->rho[0] < 0 && RM->rho[1] < 0) {
			alpha = PI + atan(abs(RM->rho[1]/RM->rho[0])); }
		else if (RM->rho[0] > 0 && RM->rho[1] < 0) {
			alpha = 2*PI - atan(abs(RM->rho[1]/RM->rho[0])); }
		else if (RM->rho[0] == 0 && RM->rho[1] > 0) {
			alpha = 1/2*PI; }
		else if (RM->rho[0] == 0 && RM->rho[1] < 0) {
			alpha = 3/2*PI; }
		else if (RM->rho[0] > 0 && RM->rho[1] == 0) {
			alpha = 0; }
		else if (RM->rho[0] < 0 && RM->rho[1] == 0) {
			alpha = PI; }
		else {
			alpha = 0; }

		beta = PI - alpha;

		return (0);
}
```

Relative Motion Integration File

satellite_sim/satellite/rmot/src/rmot_integ.c

```
/*************************************************************
PURPOSE:      (Relative motion state integration)
*************************************************************/
#include "sim_services/include/integrator.h"
extern void integrate(INTEGRATOR *I);
#include "../include/rmot.h"

int rmot_integ(
                    INTEGRATOR *I,
                    RMOT* RM)
{
        /* Load current states */
        I->state[0] = RM->rho[0];
        I->state[1] = RM->rho[1];
        I->state[2] = RM->rho[2];
        I->state[3] = RM->rhodot[0];
        I->state[4] = RM->rhodot[1];
        I->state[5] = RM->rhodot[2];
        I->state[6] = RM->I[0];
        I->state[7] = RM->I[1];

        /* Load derivatives */
        I->deriv[I->intermediate_step][0] = RM->rhodot[0] ;
        I->deriv[I->intermediate_step][1] = RM->rhodot[1] ;
        I->deriv[I->intermediate_step][2] = RM->rhodot[2] ;
   I->deriv[I->intermediate_step][3] = RM->rhoddot[0] ;
        I->deriv[I->intermediate_step][4] = RM->rhoddot[1] ;
        I->deriv[I->intermediate_step][5] = RM->rhoddot[2] ;
        I->deriv[I->intermediate_step][6] = RM->Idot[0] ;
        I->deriv[I->intermediate_step][7] = RM->Idot[1] ;

        /* Call the Trick integrator */
        integrate( I ) ;

        /* Unload new states */
        RM->rho[0] = I->state_ws[I->intermediate_step][0] ;
        RM->rho[1] = I->state_ws[I->intermediate_step][1] ;
        RM->rho[2] = I->state_ws[I->intermediate_step][2] ;
        RM->rhodot[0] = I->state_ws[I->intermediate_step][3] ;
        RM->rhodot[1] = I->state_ws[I->intermediate_step][4] ;
        RM->rhodot[2] = I->state_ws[I->intermediate_step][5] ;
        RM->I[0] = I->state_ws[I->intermediate_step][6] ;
        RM->I[1] = I->state_ws[I->intermediate_step][7] ;

        /* Return step. Used for multi-pass integration */
        return ( I->intermediate_step ) ;
}
```

Trick S_Define File

satellite_sim/SIM_satellite/S_define

```
#define R_EARTH 6371.0
#define MU 398600.4418

sim_object
{
        sim_services/include:
          EXECUTIVE exec (sim_services/include/executive.d) ;

        (automatic) sim_services/input_processor:
          input_processor( INPUT_PROCESSOR * IP = &sys.exec.ip ) ;

        (automatic_last) sim_services/exec:
          var_server_sync( Inout EXECUTIVE * E = &sys.exec ) ;

} sys ;

sim_object
{
        satellite/orbit: ORB orb1 (satellite/orbit/include/orbit1.d) ;
        satellite/rmot: RMOT rmot1 (satellite/rmot/include/rmot.d) ;

        sim_services/include: INTEGRATOR integ1
          (satellite/orbit/include/orbit_integ.d) ;
        sim_services/include: INTEGRATOR integ2
          (satellite/rmot/include/rmot_integ.d) ;

        (initialization) satellite/orbit:
                orbit_init( ORB* O = &dyn.orb1 ) ;
        (initialization) satellite/rmot:
                rmot_init( RMOT* RM = &dyn.rmot1, ORB* O = &dyn.orb1 ) ;

        (derivative) satellite/orbit:
                orbit_deriv( ORB* O = &dyn.orb1 ) ;
        (derivative) satellite/rmot:
                rmot_deriv( ORB* O = &dyn.orb1, RMOT* RM = &dyn.rmot1  ) ;

        (integration) satellite/orbit:
                orbit_integ( INTEGRATOR* I1 = &dyn.integ1, ORB* O = &dyn.orb1 ) ;
        (integration) satellite/rmot:
                rmot_integ( INTEGRATOR* I2 = &dyn.integ2, RMOT* RM = &dyn.rmot1 ) ;
} dyn ;

integrate (0.001) dyn ;

sim_object
{
        satellite/attitude: ROT rot1 (satellite/attitude/include/attitude1.d) ;
        satellite/attitude: ROT rot2 (satellite/attitude/include/attitude2.d) ;

        sim_services/include: INTEGRATOR integ3
                (satellite/attitude/include/attitude_integ.d) ;
        sim_services/include: INTEGRATOR integ4
                (satellite/attitude/include/attitude_integ.d) ;

        (initialization) satellite/attitude:
                attitude_init( ROT* R1 = &dyn2.rot1, double rmag = dyn.orb1.rmag ) ;
```

```
        (initialization) satellite/attitude:
                attitude_init( ROT* R2 = &dyn2.rot2, double rmag = dyn.orb1.rmag ) ;

        (derivative) satellite/attitude:
                attitude_deriv( ROT* R1 = &dyn2.rot1, double rmag = dyn.orb1.rmag ) ;
        (derivative) satellite/attitude:
           attitude_deriv( ROT* R2 = &dyn2.rot2, double rmag = dyn.orb1.rmag ) ;

        (integration) satellite/attitude:
                attitude_integ( INTEGRATOR* I3 = &dyn2.integ3, ROT* R1 = &dyn2.rot1 ) ;
        (integration) satellite/attitude:
                attitude_integ( INTEGRATOR* I4 = &dyn2.integ4, ROT* R2 = &dyn2.rot2 ) ;
} dyn2 ;

integrate (.01) dyn2 ;
```

Trick Input File
satellite_sim/SIM_satellite/RUN_test/input

```
#include "S_default.dat"
//#include "Modified_data/orb_rmot_rot.dr"
//#include "Modified_data/rotations.dr"
//#include "Modified_data/avizo_sim_stuff.dr"
#include "Modified_data/realtime.dr"

sys.exec.sim_com.panel_size = Panel_Size_Lite ;
sys.exec.in.stripchart_input_file = "Modified_data/satellite.sc" ;

sys.exec.in.tv = Off ;
sys.exec.in.tv_input_file = "satellite.tv" ;

/* Controller Gains */
dyn.rmot1.Kvals[0][0] {--}  = 0.0442;
dyn.rmot1.Kvals[0][1] {--}  = -0.0163;
dyn.rmot1.Kvals[0][2] {--}  = 0.3901;
dyn.rmot1.Kvals[0][3] {--}  = -0.0399;
dyn.rmot1.Kvals[0][4] {--}  = 0.0012;
dyn.rmot1.Kvals[0][5] {--}  = -0.0015;
dyn.rmot1.Kvals[1][0] {--}  = .004;
dyn.rmot1.Kvals[1][1] {--}  = 0.042;
dyn.rmot1.Kvals[1][2] {--}  = 0.0124;
dyn.rmot1.Kvals[1][3] {--}  = 0.3799;
dyn.rmot1.Kvals[1][4] {--}  = 0.0003;
dyn.rmot1.Kvals[1][5] {--}  = 0.0011;

dyn2.rot1.Gvals[0][0] {--} = 4.153;
dyn2.rot1.Gvals[0][1] {--} = -.001;
dyn2.rot1.Gvals[0][2] {--} = -4.153;
dyn2.rot1.Gvals[0][3] {--} = -.001;
dyn2.rot1.Gvals[0][4] {--} = 3.297;
dyn2.rot1.Gvals[0][5] {--} = 0;
dyn2.rot1.Gvals[0][6] {--} = 0;
dyn2.rot1.Gvals[0][7] {--} = 1.581;
dyn2.rot1.Gvals[0][8] {--} = 0;
dyn2.rot1.Gvals[0][9] {--} = -1.581;
dyn2.rot1.Gvals[0][10] {--} = 0;
dyn2.rot1.Gvals[1][0] {--} = -.0016;
dyn2.rot1.Gvals[1][1] {--} = 4.153;
dyn2.rot1.Gvals[1][2] {--} = -.0016;
dyn2.rot1.Gvals[1][3] {--} = -4.153;
dyn2.rot1.Gvals[1][4] {--} = 0;
dyn2.rot1.Gvals[1][5] {--} = 3.297;
dyn2.rot1.Gvals[1][6] {--} = .0004;
dyn2.rot1.Gvals[1][7] {--} = .0015;
dyn2.rot1.Gvals[1][8] {--} = 1.581;
dyn2.rot1.Gvals[1][9] {--} = .0015;
dyn2.rot1.Gvals[1][10] {--} = -1.581;
dyn2.rot1.Gvals[2][0] {--} = 4.153;
dyn2.rot1.Gvals[2][1] {--} = .001;
dyn2.rot1.Gvals[2][2] {--} = 4.153;
dyn2.rot1.Gvals[2][3] {--} = -.0024;
dyn2.rot1.Gvals[2][4] {--} = 0;
dyn2.rot1.Gvals[2][5] {--} = .0004;
dyn2.rot1.Gvals[2][6] {--} = 3.297;
dyn2.rot1.Gvals[2][7] {--} = 1.581;
```

```
dyn2.rot1.Gvals[2][8] {--} = -.0015;
dyn2.rot1.Gvals[2][9] {--} = 1.581;
dyn2.rot1.Gvals[2][10] {--} = .0015;

dyn2.rot2.Gvals[0][0] {--} = 4.153;
dyn2.rot2.Gvals[0][1] {--} = -.001;
dyn2.rot2.Gvals[0][2] {--} = -4.153;
dyn2.rot2.Gvals[0][3] {--} = -.001;
dyn2.rot2.Gvals[0][4] {--} = 3.297;
dyn2.rot2.Gvals[0][5] {--} = 0;
dyn2.rot2.Gvals[0][6] {--} = 0;
dyn2.rot2.Gvals[0][7] {--} = 1.581;
dyn2.rot2.Gvals[0][8] {--} = 0;
dyn2.rot2.Gvals[0][9] {--} = -1.581;
dyn2.rot2.Gvals[0][10] {--} = 0;
dyn2.rot2.Gvals[1][0] {--} = -.0016;
dyn2.rot2.Gvals[1][1] {--} = 4.153;
dyn2.rot2.Gvals[1][2] {--} = -.0016;
dyn2.rot2.Gvals[1][3] {--} = -4.153;
dyn2.rot2.Gvals[1][4] {--} = 0;
dyn2.rot2.Gvals[1][5] {--} = 3.297;
dyn2.rot2.Gvals[1][6] {--} = .0004;
dyn2.rot2.Gvals[1][7] {--} = .0015;
dyn2.rot2.Gvals[1][8] {--} = 1.581;
dyn2.rot2.Gvals[1][9] {--} = .0015;
dyn2.rot2.Gvals[1][10] {--} = -1.581;
dyn2.rot2.Gvals[2][0] {--} = 4.153;
dyn2.rot2.Gvals[2][1] {--} = .001;
dyn2.rot2.Gvals[2][2] {--} = 4.153;
dyn2.rot2.Gvals[2][3] {--} = -.0024;
dyn2.rot2.Gvals[2][4] {--} = 0;
dyn2.rot2.Gvals[2][5] {--} = .0004;
dyn2.rot2.Gvals[2][6] {--} = 3.297;
dyn2.rot2.Gvals[2][7] {--} = 1.581;
dyn2.rot2.Gvals[2][8] {--} = -.0015;
dyn2.rot2.Gvals[2][9] {--} = 1.581;
dyn2.rot2.Gvals[2][10] {--} = .0015;

stop = 500;
```

Trick Real Time Simulation File

satellite_sim/SIM_satellite/Modified_data/realtime.dr

```
// Data file for running real time simulation
sys.exec.in.frame_log = Yes ;
sys.exec.in.rt_software_frame {s} = 0.01 ;
sys.exec.in.rt_itimer = No ;
sys.exec.in.rt_itimer_pause = No ;
sys.exec.in.rt_itimer_frame {s} = 0.01 ;
sys.exec.in.enable_freeze = Yes ;
sys.exec.work.freeze_command = Yes ;
sys.exec.sim_com.monitor_on = Yes ;
```

Trick Stripchart File

satellite_sim/SIM_satellite/Modified_data/satellite.sc

```
stripchart:
        title = "Satellite Relative Trajectory"
        geometry = 600x600+1000+100
        x_min = -20
        x_max = 20
        y_min = -20
        y_max = 20
        auto_shutdown = 0
        x_variable = dyn.rmot1.rho[0] --label=RelPositionX --unit=m
        dyn.rmot1.rho[1] --label=RelPositionY --unit=m

stripchart:
        title = "Satellite Orientation"
        geometry = 600x600+100+10
        x_min = 0
        x_max = 125
        y_min = -1
        y_max = 1
        auto_shutdown = 0
        x_variable = sys.exec.out.time --label=Time --unit=s
        dyn2.rot1.q[0]
        dyn2.rot1.q[1]
        dyn2.rot1.q[2]
        dyn2.rot1.q[3]
```

Avizo Server Connection File

satellite_sim/serverconnect.scro

```
#########################################################################
#----------------------constructor---------------------------------#
# Script used to connect Trick and Avizo #
#########################################################################$
$this proc constructor {} {

        clear

        echo "*************************************************************************"
        echo " Sat_connect program"
        echo "*************************************************************************"
        echo ""
        echo " To run simulation, start the Trick simulator first,"
        echo " but do not hit start until after the 'Retrieve Data'"
        echo " button in Avizo is pressed."

        # scro "ID"
        $this setVar scroTypeTranslateObject 1

        # Set global variables for access outside of constructor
        global sat
        global xPositionT1
        global yPositionT1
        global zPositionT1
        global xPositionT2
        global yPositionT2
        global zPositionT2
        global xAxisT1
        global yAxisT1
        global zAxisT1
        global xAxisT2
        global yAxisT2
        global zAxisT2
        global ThetaT1
        global ThetaT2

        # Empties variables
        set xPositionT1 "0"
        set yPositionT1 "0"
        set zPositionT1 "0"
        set xPositionT2 "0"
        set yPositionT2 "0"
        set zPositionT2 "0"
        set xAxisT1 "0"
        set yAxisT1 "0"
        set zAxisT1 "0"
        set xAxisT2 "0"
        set yAxisT2 "0"
        set zAxisT2 "0"
        set ThetaT1 "0"
        set ThetaT2 "0"

        # master time slider
        $this newPortTime time
        $this time setMinMax 1 300
        $this time setIncrement 1
```

```
# IP Address Box
$this newPortText ipAddress
$this ipAddress setLabel "Server IP Address: "
$this ipAddress setNumColumns 30
$this ipAddress setValue "128.46.118.212"

# Port Number Box
$this newPortText portNumber
$this portNumber setLabel "Port Number"
$this portNumber setNumColumns 6
$this portNumber setValue 7000

# Creates the text field for entering the object names
# region1sc_Spring2.wrl
$this newPortText targName1
$this targName1 setLabel "Name of HCW SC:"
$this targName1 setValue "region1sc_Spring2.wrl"

# Creates the text field for entering the object names
# region1sc_Spring09.wrl
$this newPortText targName2
$this targName2 setLabel "Name of Orbiting SC:"
$this targName2 setValue "region1sc_Spring09.wrl"

# Creates the text field for entering the object names
# earth2.wrl
$this newPortText targName3
$this targName3 setLabel "Name of Earth:"
$this targName3 setValue "earth2.wrl"

# rate to read data from server
$this newPortText cycle_rate
$this cycle_rate setLabel "Read time interval:"
$this cycle_rate setNumColumns 5
$this cycle_rate setValue 0.1

# Total Number of Steps
$this newPortText numberSteps
$this numberSteps setLabel "Total Iterations"
$this numberSteps setNumColumns 6
$this numberSteps setValue 100

# Retrieve Data Button
$this newPortButtonList openSocket 1
$this openSocket setLabel "Connection:"
$this openSocket setLabel 0 "Connect to Trick"
$this openSocket setCmd 0 {

        set timecall [$this cycle_rate getValue]
        global sat
        global timeInt
        global xPositionT1
        global yPositionT1
        global zPositionT1
        global xPositionT2
        global yPositionT2
        global zPositionT2
        global xAxisT1
```

```
                global yAxisT1
                global zAxisT1
                global xAxisT2
                global yAxisT2
                global zAxisT2
                global ThetaT1
                global ThetaT2

                # connect to server
                set sat(socket) [simcom_connect [$this ipAddress getValue] [$this portNumber getValue]]

                get_sim_data $timecall [$this targName1 getValue] [$this targName2 getValue] [$this targName3
getValue]

                $this numberSteps setValue $timeInt(max)

                $this time setMinMax 1 $timeInt(max)
        }

}

############################################################$
#---------------get_sim_data----------------------------#$
# collects data from the server and defines it #$
############################################################$
proc get_sim_data {cycle_rate region1sc_Spring2.wrl region1sc_Spring09.wrl earth2.wrl} {

        global sat
        global timeInt
        global xPositionT1
        global yPositionT1
        global zPositionT1
        global xPositionT2
        global yPositionT2
        global zPositionT2
        global xAxisT1
        global yAxisT1
        global zAxisT1
        global xAxisT2
        global yAxisT2
        global zAxisT2
        global ThetaT1
        global ThetaT2

        simcom_send_cmd $sat(socket) "var_cycle = $cycle_rate;"
        simcom_send_cmd $sat(socket) "var_add dyn.rmot1.rho\[0\];"
        simcom_send_cmd $sat(socket) "var_add dyn.rmot1.rho\[1\];"
        simcom_send_cmd $sat(socket) "var_add dyn.rmot1.rho\[2\];"
        simcom_send_cmd $sat(socket) "var_add dyn.orb1.r\[0\];"
        simcom_send_cmd $sat(socket) "var_add dyn.orb1.r\[1\];"
        simcom_send_cmd $sat(socket) "var_add dyn.orb1.r\[2\];"
        simcom_send_cmd $sat(socket) "var_add dyn2.rot1.ehat\[0\];"
        simcom_send_cmd $sat(socket) "var_add dyn2.rot1.ehat\[1\];"
        simcom_send_cmd $sat(socket) "var_add dyn2.rot1.ehat\[2\];"
        simcom_send_cmd $sat(socket) "var_add dyn2.rot2.ehat\[0\];"
        simcom_send_cmd $sat(socket) "var_add dyn2.rot2.ehat\[1\];"
        simcom_send_cmd $sat(socket) "var_add dyn2.rot2.ehat\[2\];"
        simcom_send_cmd $sat(socket) "var_add dyn2.rot1.Phi;"
        simcom_send_cmd $sat(socket) "var_add dyn2.rot2.Phi;"
```

```
simcom_send_cmd		$sat(socket) "var_add dyn2.rot1.Espin;"

set counter 0
set timeInt(min) 0
set timeInt(max) 0

if { [gets $sat(socket) sim_data] == -1 } { echo " Thats all folks..." }

while { [gets $sat(socket) sim_data] != -1 } {

	set sat(positions) $sim_data; # [format "%.0f" $sim_data]

	if {[lindex $sat(positions) 0]==0} {

		set timeInt(min) [expr $counter+1]
		echo "Waiting for Trick"
		update

	} else {

		# Appends data to lists
		lappend xPositionT1 "[lindex $sat(positions) 0]"
		lappend yPositionT1 "[lindex $sat(positions) 1]"
		lappend zPositionT1 "[lindex $sat(positions) 2]"
		lappend xPositionT2 "[lindex $sat(positions) 3]"
		lappend yPositionT2 "[lindex $sat(positions) 4]"
		lappend zPositionT2 "[lindex $sat(positions) 5]"
		lappend xAxisT1 "[lindex $sat(positions) 6]"
		lappend yAxisT1 "[lindex $sat(positions) 7]"
		lappend zAxisT1 "[lindex $sat(positions) 8]"
		lappend xAxisT2 "[lindex $sat(positions) 9]"
		lappend yAxisT2 "[lindex $sat(positions) 10]"
		lappend zAxisT2 "[lindex $sat(positions) 11]"
		lappend ThetaT1 "[lindex $sat(positions) 12]"
		lappend ThetaT2 "[lindex $sat(positions) 13]"
		lappend Espin "[lindex $sat(positions) 14]"

		set OrbitX [lindex $sat(positions) 3]
		set OrbitY [lindex $sat(positions) 4]
		set OrbitZ [lindex $sat(positions) 5]

		update
		region1sc_Spring09.wrl setTranslation $OrbitX $OrbitY $OrbitZ
		region1sc_Spring09.wrl touch 1
		region1sc_Spring09.wrl fire

		set HCWRelX [lindex $sat(positions) 0]
		set HCWRelY [lindex $sat(positions) 1]
		set HCWRelZ [lindex $sat(positions) 2]

		set HCWRelX [expr {$HCWRelX*1000}]
		set HCWRelY [expr {$HCWRelY*1000}]
		set HCWRelZ [expr {$HCWRelZ*1000}]

		set HCWInertiaX [expr {$OrbitX + ($HCWRelX*$OrbitX/sqrt($OrbitX**2+$OrbitY**2)) -
($HCWRelY*$OrbitY/sqrt($OrbitX**2+$OrbitY**2))}]
		set HCWInertiaY [expr {$OrbitY + ($HCWRelX*$OrbitY/sqrt($OrbitX**2+$OrbitY**2)) +
($HCWRelY*$OrbitX/sqrt($OrbitX**2+$OrbitY**2))}]
		set HCWInertiaZ $HCWRelZ
```

```
set EarthTheta [lindex $sat(positions) 14]

set CameraTheta [expr {3.14159*0.3}]

update
region1sc_Spring2.wrl setTranslation $HCWInertiaX $HCWInertiaY $HCWInertiaZ
region1sc_Spring2.wrl touch 1
region1sc_Spring2.wrl fire

set Rot1Ax [lindex $sat(positions) 6]
set Rot1Ay [lindex $sat(positions) 7]
set Rot1Az [lindex $sat(positions) 8]

set Rot2Ax [lindex $sat(positions) 9]
set Rot2Ay [lindex $sat(positions) 10]
set Rot2Az [lindex $sat(positions) 11]

update
region1sc_Spring2.wrl setRotation $Rot1Ax  $Rot1Ay $Rot1Az [lindex $sat(positions) 12]
region1sc_Spring2.wrl touch 1
region1sc_Spring2.wrl fire

update
region1sc_Spring09.wrl setRotation $Rot2Ax  $Rot2Ay $Rot2Az [lindex $sat(positions) 13]
region1sc_Spring09.wrl touch 1
region1sc_Spring09.wrl fire

update
earth2.wrl setRotation 0 0 1 $EarthTheta
earth2.wrl touch 1
earth2.wrl fire

#viewer 0 redraw
#viewer 0 setCameraFocalDistance 20
#viewer 0 setCameraPosition $OrbitX $OrbitY $OrbitZ
#viewer 0 setCameraOrientation 1 1 1 $CameraTheta
#viewer 0 setCameraType orthographic
#viewer 0 setCameraHeight 20

viewer 0 setCameraOrientation 0.437319 0.731464 0.523175 1.99951
viewer 0 setCameraPosition $OrbitX $OrbitY $OrbitZ
viewer 0 setCameraFocalDistance 20
viewer 0 setCameraNearDistance -10.2322
viewer 0 setCameraFarDistance 7055.82
viewer 0 setCameraType orthographic
viewer 0 setCameraHeight 21.6883
viewer 0 setAutoRedraw 1
viewer 0 redraw


# echo "$sat(positions)"
echo " "
echo "---------Step [expr $counter - $timeInt(min)]----------------------"
echo "HCW x = $HCWRelX"
echo "HCW y = $HCWRelY"
echo "HCW z = $HCWRelZ"
echo "Orbiting x = $OrbitX"
echo "Orbiting y = $OrbitY"
```

```
                        echo "Orbiting z = $OrbitZ"
                        echo "HCW Orbit x = $HCWInertiaX"
                        echo "HCW Orbit y = $HCWInertiaY"
                        echo "HCW Orbit z = $HCWInertiaZ"
                        echo "HCW xAxis = [lindex $sat(positions) 6]"
                        echo "HCW yAxis = [lindex $sat(positions) 7]"
                        echo "HCW zAxis = [lindex $sat(positions) 8]"
                        echo "Orbiting xAxis = [lindex $sat(positions) 9]"
                        echo "Orbiting yAxis = [lindex $sat(positions) 10]"
                        echo "Orbiting zAxis = [lindex $sat(positions) 11]"
                        echo "HCW Theta = [lindex $sat(positions) 12]"
                        echo "Orbiting Theta = [lindex $sat(positions) 13]"

                }

        set counter [expr $counter + 1]
        set timeInt(max) [expr $counter - $timeInt(min)]
        }

}



############################################################
#-------------destructor-------------------------------#
# destructor is called when scro is destroyed #
############################################################
$this proc destructor {} {}

#########################################################################
#--------------------compute--------------------------------------#
# the "compute" method is called whenever a port has changed #
#########################################################################
#$this proc compute {} {
#
#       # when the time slider is touched, translate object
#       if [$this time isNew] {
#               set tminmax [$this time getMinMax]
#               $this translateObject [$this targName getValue] [$this time getValue] [lindex $tminmax 0] [lindex
$tminmax 1]
#       }
#}

#############################################################################
#----------------translateObject-----------------------------------------#
# translate attached object $obj between two positions, corresponding to #
# time step $t within $t_minmax. #
#############################################################################
$this proc translateObject {objSatellite t tmin tmax} {

        #repeat for Satellite
        if {$objSatellite == ""} {
                echo "$this: no data Satellite object connected."
                $this data disconnect
                return
        }

        if {![$objSatellite hasInterface HxSpatialData]} {
        $echo "$this: $objSatellite is not a spatial data object."
```

```
		$this data disconnect
		return
		}

		# Let procedure know to look in global namespace
		global xPositionT1
		global yPositionT1
		global zPositionT1
		global xPositionT2
		global yPositionT2
		global zPositionT2

		# Drop down to the lat integer of timestep
		set floorTime [expr floor([$this time getValue])]
		set intTime [expr int($floorTime)]

		# Set the translation/rotation values
		set xSatellite1 [lindex $xPositionT1 $intTime]
		set ySatellite1 [lindex $yPositionT1 $intTime]
		set zSatellite1 [lindex $zPositionT1 $intTime]
		set xSatellite2 [lindex $xPositionT2 $intTime]
		set ySatellite2 [lindex $yPositionT2 $intTime]
		set zSatellite2 [lindex $zPositionT2 $intTime]

		# preform the translation/rotation
		$objSatellite1 setTranslation $xSatellite1 $ySatellite1 $zSatellite1
		$objSatellite1 touch 1;
		$objSatellite1 fire
		$objSatellite2 setTranslation $xSatellite2 $ySatellite2 $zSatellite2
		$objSatellite2 touch 1;
		$objSatellite2 fire

}

##############################################################################
# USER: DO NOT EDIT!!!! #
##############################################################################
proc simcom_connect { host port } {

		global simcom_error
		global tcl_platform

		set ret [catch { set sock [socket $host $port] } dog]
		if { $ret != 0 } { return -1 }
		fconfigure $sock -buffering none
		fconfigure $sock -translation binary

		# Send over endianess
		if { $tcl_platform(byteOrder) == "littleEndian" } {
			puts -nonewline $sock \x01\x00
		} else {
			puts -nonewline $sock \x00\x00
		}

		# set client id ( only used in error messages )
		set client_id [binary format I 8675309]

		# set client tag ( buffered to 80 bytes )
		set client_tag [binary format a80 "var_client"]
```

```
	puts -nonewline $sock $client_id$client_tag

	# Read off the server's endianess read $sock 2
	set simcom_error 0
	return $sock;

}

proc simcom_send_cmd { sock cmd } {

	global simcom_error

	# Send command to variable server
	regsub {\n} $cmd { } cmd
	set cmd_len [binary format I [string length $cmd] ]
	if { [catch {puts -nonewline $sock $cmd_len}] } {
		set simcom_error 1
	}
	if { [catch {puts -nonewline $sock $cmd}] } {
		set simcom_error 1
	}
}

# EDITED TRICK SIMCOM PACKAGE END
# VIZLAB
##############################################################################
# USER: INSERT APPROPRIATE VARIABLES FROM TRICK
##############################################################################
```

Avizo .HX File

satellite_sim/AvizoBlue.hx

```
# Avizo Project
# Avizo
remove -all
remove "physics.icol" "region1sc_Spring2.wrl" "region1sc_Spring09.wrl" "earth2.wrl" "serverconnect.scro" "Display
Open Inventor Scene 2" "Display Open Inventor Scene" "Display Open Inventor Scene 4"

# Create viewers
viewer setVertical 0

viewer 0 setBackgroundMode 3
viewer 0 setBackgroundColor 0.145098 0.152941 0.176471
viewer 0 setBackgroundColor2 0.435294 0.45098 0.498039
viewer 0 setTransparencyType 5
viewer 0 setBackgroundImage N:/MySVNProjects/satellite_sim/Starfield0.jpg
viewer 0 setAutoRedraw 0
viewer 0 show
mainWindow show

set hideNewModules 1
[ load ${AMIRA_ROOT}/data/colormaps/physics.icol ] setLabel "physics.icol"
"physics.icol" setIconPosition 0 0
"physics.icol" setNoRemoveAll 1
"physics.icol" fire
"physics.icol" setMinMax 0 1
"physics.icol" flags setValue 1
"physics.icol" shift setMinMax -1 1
"physics.icol" shift setButtons 0
"physics.icol" shift setEditButton 1
"physics.icol" shift setIncrement 0.133333
"physics.icol" shift setValue 0
"physics.icol" shift setSubMinMax -1 1
"physics.icol" scale setMinMax 0 1
"physics.icol" scale setButtons 0
"physics.icol" scale setEditButton 1
"physics.icol" scale setIncrement 0.1
"physics.icol" scale setValue 1
"physics.icol" scale setSubMinMax 0 1
"physics.icol" fire
"physics.icol" setViewerMask 65535

set hideNewModules 0
[ load ${SCRIPTDIR}/region1sc_Spring2.wrl ] setLabel "region1sc_Spring2.wrl"
"region1sc_Spring2.wrl" setIconPosition 22 82
"region1sc_Spring2.wrl" setTransform -1.3113e-006 -2.01166e-007 -1 0 2.98023e-007 1 1.86265e-007 0 1 2.68221e-007 -
1.2964e-006 0 6163.18 2159.86 0 1
"region1sc_Spring2.wrl" fire
"region1sc_Spring2.wrl" setViewerMask 65535

set hideNewModules 0
[ load ${SCRIPTDIR}/region1sc_Spring09.wrl ] setLabel "region1sc_Spring09.wrl"
"region1sc_Spring09.wrl" setIconPosition 23 135
"region1sc_Spring09.wrl" setTransform -1.2815e-006 2.38419e-007 -0.999999 0 -2.98023e-007 1 -1.63913e-007 0
0.999999 -3.8743e-007 -1.49012e-006 0 6160.35 2158.87 0 1
"region1sc_Spring09.wrl" fire
"region1sc_Spring09.wrl" setViewerMask 65535
```

```
set hideNewModules 0
[ load ${SCRIPTDIR}/earth2.wrl ] setLabel "earth2.wrl"
"earth2.wrl" setIconPosition 25 159
"earth2.wrl" setTransform -0.316964 0 0.679731 0 0 0.75 0 0 -0.679731 0 -0.316964 0 0 0 0 1
"earth2.wrl" fire
"earth2.wrl" setViewerMask 65535
"earth2.wrl" select

set hideNewModules 0
create HxScriptObject "serverconnect.scro"
"serverconnect.scro" script setValue ${SCRIPTDIR}/serverconnect.scro
"serverconnect.scro" setIconPosition 158 11
"serverconnect.scro" setVar "scroTypeTranslateObject" {1}
"serverconnect.scro" setVar "scroTypeTranslateObject" {1}
"serverconnect.scro" fire
"serverconnect.scro" time setMinMax 1 628
"serverconnect.scro" time setSubMinMax 1 628
"serverconnect.scro" time setValue 1
"serverconnect.scro" time setDiscrete 0
"serverconnect.scro" time setIncrement 1
"serverconnect.scro" time animationMode -once
"serverconnect.scro" ipAddress setState 128.46.118.212
"serverconnect.scro" portNumber setState 7002
"serverconnect.scro" targName1 setState region1sc_Spring2.wrl
"serverconnect.scro" targName2 setState region1sc_Spring09.wrl
"serverconnect.scro" targName3 setState earth2.wrl
"serverconnect.scro" cycle_rate setState 0.1
"serverconnect.scro" numberSteps setState 628
"serverconnect.scro" applyTransformToResult 1
"serverconnect.scro" fire
"serverconnect.scro" setViewerMask 65535
"serverconnect.scro" setPickable 1

set hideNewModules 0
create HxIvDisplay "Display Open Inventor Scene 2"
"Display Open Inventor Scene 2" setIconPosition 198 82
"Display Open Inventor Scene 2" data connect "region1sc_Spring2.wrl"
"Display Open Inventor Scene 2" fire
"Display Open Inventor Scene 2" drawStyle setIndex 0 0
"Display Open Inventor Scene 2" fire
"Display Open Inventor Scene 2" setViewerMask 65535
"Display Open Inventor Scene 2" setShadowStyle 0
"Display Open Inventor Scene 2" setPickable 1

set hideNewModules 0
create HxIvDisplay "Display Open Inventor Scene"
"Display Open Inventor Scene" setIconPosition 207 131
"Display Open Inventor Scene" data connect "region1sc_Spring09.wrl"
"Display Open Inventor Scene" fire
"Display Open Inventor Scene" drawStyle setIndex 0 0
"Display Open Inventor Scene" fire
"Display Open Inventor Scene" setViewerMask 65535
"Display Open Inventor Scene" setShadowStyle 0
"Display Open Inventor Scene" setPickable 1

set hideNewModules 0
create HxIvDisplay "Display Open Inventor Scene 4"
"Display Open Inventor Scene 4" setIconPosition 198 159
"Display Open Inventor Scene 4" data connect "earth2.wrl"
```

```
"Display Open Inventor Scene 4" fire
"Display Open Inventor Scene 4" drawStyle setIndex 0 0
"Display Open Inventor Scene 4" fire
"Display Open Inventor Scene 4" setViewerMask 65535
"Display Open Inventor Scene 4" setShadowStyle 0
"Display Open Inventor Scene 4" setPickable 1

set hideNewModules 0


viewer 0 setCameraOrientation 0.283766 0.924142 0.255811 1.53604
viewer 0 setCameraPosition 6160.51 2160.02 -0.588237
viewer 0 setCameraFocalDistance 20
viewer 0 setCameraNearDistance -1108.89
viewer 0 setCameraFarDistance 12353.6
viewer 0 setCameraType orthographic
viewer 0 setCameraHeight 5881.32
viewer 0 setAutoRedraw 1
viewer 0 redraw

theObjectPool setSelectionOrder "earth2.wrl"
```

Staged Controller GUI File
satellite_sim/SIM_satellite/StagedGUI.tcl

```
# #! /bin/sh
# \
exec wish "$0" "$@"

# Grab Trick's SimCom package
global auto_path
set auto_path [linsert $auto_path 0 $env(TRICK_HOME)/bin/tcl]
package require Simcom
namespace import Simcom::*

proc createGUI { } {
	global data
	button .bS1 -text "Approach to 5m vicinity" -command "Approach $data(socket)"
	button .bS2 -text "Orient Satellites" -command "Orient $data(socket)"
			button .bS3 -text "Rendezvous" -command "Rendezvous $data(socket)"

			pack .bS1 .bS2 .bS3

	}



proc Approach { sock } {
	Simcom::send_cmd $sock "dyn.rmot1.S1flag = 1 ;"
	}

proc Orient { sock } {
	Simcom::send_cmd $sock "dyn2.rot1.S2flag = 1 ;"
			Simcom::send_cmd $sock "dyn2.rot2.S2flag = 1 ;"
	}

proc Rendezvous { sock } {
	Simcom::send_cmd $sock "dyn.rmot1.S3flag = 1 ;"
	}



proc get_sim_data { } {
	global data

	Simcom::send_cmd $data(socket) "var_cycle = 0.01 ;"
	Simcom::send_cmd $data(socket) "var_add dyn.rmot1.S1flag ;"
			Simcom::send_cmd $data(socket) "var_add dyn2.rot1.S2flag ;"
			Simcom::send_cmd $data(socket) "var_add dyn2.rot2.S2flag ;"
			Simcom::send_cmd $data(socket) "var_add dyn.rmot1.S3flag ;"

	while { [gets $data(socket) sim_data] != -1 } {
		set data(flags) $sim_data; #[format "%.2f" $sim_data]
		update
		}
	}



proc main { } {
	global data
```

```
	global argv

	set data(port) [lindex $argv 0]
	set data(socket) [Simcom::connect "localhost" $data(port)]

	createGUI

	get_sim_data
}

main
```

References

[1] Ploen, S.R., Scharf, D.P., Hadaegh, F.Y., and Acikmese, A.B., "Dynamics of Earth Orbiting Formations," Jet Propulsion Lab, California Institute of Technology.

[2] Frazho, A. *State space control for the two degrees of freedom helicopter* [PDF document]. Retrieved from https://engineering.purdue.edu/AAE/Academics/Courses/aae364L.

[3] Frazho, A. *PID control for the two degree of freedom helicopter* [PDF document]. Retrieved from https://engineering.purdue.edu/AAE/Academics/Courses/aae364L.

[4] Vetter, Keith; Chen, Hong. "Trick Simulation Environment User Training Materials Trick 2007.0 Release". NASA JSC Automation, Robotics, and Simulation Division. 2007.

[5] "Trick User's Guide". NASA JSC Automation, Robotics, and Simulation Division. May 2008.

[6] Brown, Todd; Olikara, Zubin; Patterson, Chris; Schlei, Wayne; Short, Cody. "Avizo: The Supreme Users Manual". August 2011.