Spacecraft Dynamics and Control Capstone Final Project Report

Ian M. Faber *
University of Colorado Boulder, Boulder, Colorado, 80303

As part of ASEN 5010, students were assigned a capstone project simulating a science mission around Mars. Specifically, the project focuses on the pointing scenarios relayed in the "Attitude Dynamics and Control of a Nano-Satellite Orbiting Mars" PDF [1]. The project allows the student to demonstrate mastery of the content in ASEN 5010 as well as get a look at simulating a simple space mission.

I. Nomenclature

$\mathcal{B}$	=	Nano-satellite Body Frame
$[BN]$	=	DCM for mapping coordinates in the Inertial Frame ($\mathcal{N}$) to coordinates in the Body Frame ($\mathcal{B}$)
$\mathcal{H}$	=	Hill Frame
h	=	Spacecraft orbit altitude
$[I]$	=	Rigid body inertia tensor
${}^{\mathcal{B}}[I]$	=	Rigid body inertia tensor, relative to body frame
$[M_n(\theta)]$	=	Pure rotation about the n axis by angle θ
$\mathcal{N}$	=	Inertial Frame
$\mathcal{O}$	=	Orbit Frame
$\mathbf{r}$	=	Nano-satellite inertial position vector
${}^{\mathcal{N}}\mathbf{r}$	=	Nano-satellite inertial position vector, expressed in inertial frame coordinates
r_{LMO}	=	Nano-satellite orbit radius
r_{GMO}	=	Mothercraft orbit radius (areosynchronous)
$R_{♂}$	=	Radius of Mars
$\dot{\mathbf{r}}$	=	Nano-satellite inertial velocity vector
${}^{\mathcal{N}}\dot{\mathbf{r}}$	=	Nano-satellite inertial velocity vector, expressed in inertial frame coordinates
$\mathcal{R}$	=	Reference Frame
$\mathcal{R}_c$	=	Communication-Pointing Frame
$\mathcal{R}_n$	=	Mars Nadir-Pointing Frame
$\mathcal{R}_s$	=	Sun-Pointing Frame
t	=	Time
T	=	Kinetic Energy
$\mathbf{u}$	=	Control Torque Vector
$\sigma_{B/N}$	=	Attitude of spacecraft relative to inertial frame, expressed as Modified Rodriguez Parameters
$\omega_{B/N}$	=	Angular velocity of spacecraft relative to inertial frame
${}^{\mathcal{B}}\omega_{B/N}$	=	Angular velocity of spacecraft relative to inertial frame, expressed in body frame coordinates
Ω	=	Orbit right ascension of the ascending node
i	=	Orbit inclination
θ	=	Orbit true latitude angle
$\dot{\theta}_{\text{LMO}}$	=	Orbit rate of the nano-satellite
$\dot{\theta}_{\text{GMO}}$	=	Orbit rate of the mothercraft
μ	=	Mars gravity constant
${}^{\mathcal{N}}\begin{bmatrix} a \\ b \\ c \end{bmatrix}$	=	Generic vector written in inertial coordinates

*Undergraduate Student, Ann & H.J. Smead Aerospace Engineering Sciences, 3775 Discovery Dr, Boulder, CO 80303, AIAA student (1602781)

II. Introduction

The ASEN 5010 Capstone Project combines everything that has been taught in the course over the Spring 2024 semester. The project report showcases the results of completing the project tasks, verified by Coursera's auto-grader where applicable.

The scenario we are simulating involves two spacecraft traveling along circular orbits. The first spacecraft is a nano-satellite that collects data on the night side of Mars and is powered with solar panels. The second spacecraft is the "mothercraft" situated in an areosynchronous orbit. The goal is for the nano-satellite to collect data when it is on the night side of Mars, transmit this data to the mothercraft when it is within the nano-satellite's line of sight, and point towards the sun to charge its batteries in every other scenario. An illustration of the mission can be seen in Figure 1 and an illustration of the spacecraft can be seen in Figure 2.

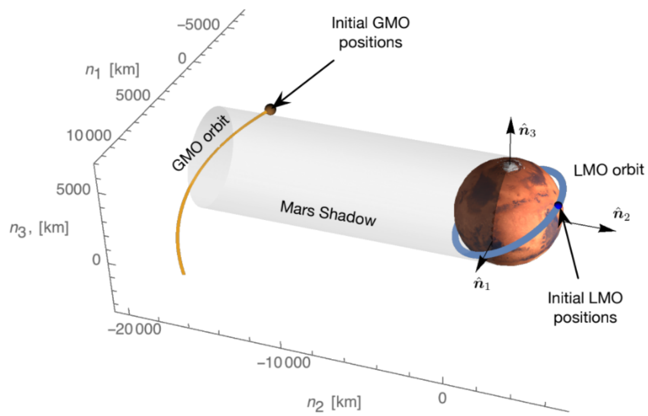

Fig. 1 Illustration of the nano-satellite science mission

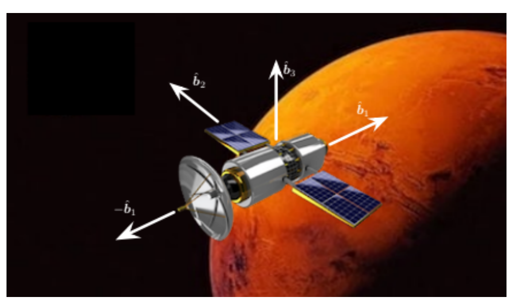

Fig. 2 Illustration of the nano-satellite body frame

A summary of the science mission scenarios can be seen in table 1. Reproduced from [1].

Table 1 Nano-satellite Pointing Scenario Summary

Orbital Situation	Primary Pointing Scenario Goals
SC on sunlit Mars side *	Point solar panel axis $\hat{b}_3$ at the Sun
SC not on sunlit Mars side & GMO visible †	Point antenna axis $-\hat{b}_1$ at the GMO
SC not on sunlit Mars side & GMO not visible ‡	Point sensor axis $\hat{b}_1$ along the Mars nadir direction

*sunlit Mars side: spacecraft inertial position has a positive $\hat{n}_2$ coordinate
†visible: LMO and GMO position vectors are separated by $\leq 35°$
‡not visible: LMO and GMO position vectors are separated by $\geq 35°$

The initial conditions for this scenario are as follows [1]:

$$\sigma_{B/N}(t_0) = \begin{bmatrix} 0.3 \\ -0.4 \\ 0.5 \end{bmatrix} \tag{1}$$

$$^{\mathcal{B}}\omega_{B/N}(t_0) = {}^{\mathcal{B}}\begin{bmatrix} 1.00 \\ 1.75 \\ -2.20 \end{bmatrix} \text{deg/s} \tag{2}$$

The inertia matrix of the nano-satellite is as follows [1]:

$$^{\mathcal{B}}[I] = \begin{bmatrix} 10 & 0 & 0 \\ 0 & 5 & 0 \\ 0 & 0 & 7.5 \end{bmatrix} \text{kg m}^2 \tag{3}$$

Finally, the orbit information for both spacecraft can be seen in table 2. Reproduced from [1]:

Table 2 Spacecraft Orbit Information

Spacecraft	Ω	i	$\theta(t_0)$	$\dot{\theta}$	$R_{♂}$	h
LMO	20°	30°	60°	0.000884797 rad/s	3396.19 km	400 km
GMO	0°	0°	250°	0.0000709003 rad/s	3396.19 km	17028.01 km

III. Task 1: Orbit Simulation

The inertial frame $\mathcal{N}$ is defined as $\mathcal{N} : \{\hat{n}_1, \hat{n}_2, \hat{n}_3\}$ and the orbit frame $\mathcal{O}$ is defined as $\mathcal{O} : \{\hat{\imath}_r, \hat{\imath}_\theta, \hat{\imath}_h\}$, as shown in Figure 3. The orbit can be described in terms of (3-1-3) Euler angles as follows: (Ω, i, θ). The orbit is circular with constant radius r, so $\dot{\theta} = \sqrt{\frac{\mu}{r^3}}$ can be assumed constant. All of the code for this section can be found in Appendix A.B.

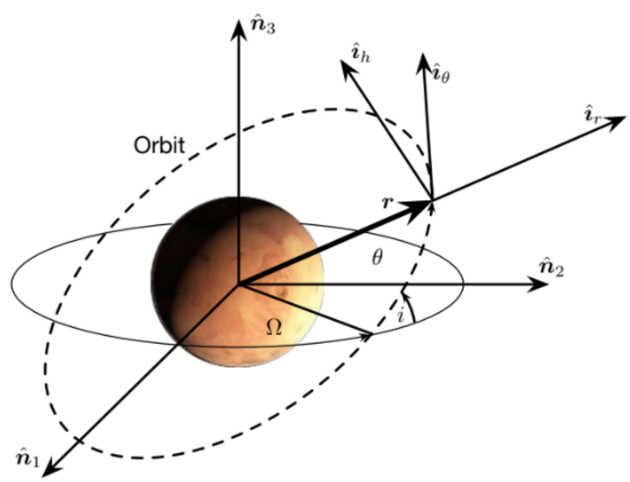

Fig. 3 Illustration of the inertial frame $\mathcal{N}$ and orbit frame $\mathcal{O}$

A. Task 1 Part 1: $\dot{\mathbf{r}}$ derivation

Using transport theorem, we know that

$$\dot{\mathbf{r}} = \frac{{}^{\mathcal{N}}\mathrm{d}}{\mathrm{d}t}(\mathbf{r}) = \frac{{}^{\mathcal{O}}\mathrm{d}}{\mathrm{d}t}(\mathbf{r}) + {}^{\mathcal{O}}\omega_{\mathcal{O}/\mathcal{N}} \times \mathbf{r} \tag{4}$$

Since we are assuming the orbit is circular with constant radius r, we can say that $\frac{{}^{\mathcal{O}}\mathrm{d}}{\mathrm{d}t}(\mathbf{r}) = 0$. If the orbit was elliptical, this assumption would be false. Here, the circular orbit assumption means that

$$\dot{\mathbf{r}} = {}^{\mathcal{O}}\omega_{\mathcal{O}/\mathcal{N}} \times \mathbf{r} \tag{5}$$

For a circular orbit, we are given $\mathbf{r} = r\hat{\imath}_r$ and $\omega_{\mathcal{O}/\mathcal{N}} = \dot{\theta}\hat{\imath}_h$. Since both vectors are already written in the same frame, the cross product is valid. Thus,

$$\boxed{\dot{\mathbf{r}} = r\dot{\theta}\hat{\imath}_\theta} \tag{6}$$

B. Task 1 Part 2: Orbit ${}^{\mathcal{N}}\mathbf{r}$ and ${}^{\mathcal{N}}\dot{\mathbf{r}}$ calculation program

For plotting and error checking, it is useful to compute $\mathbf{r}$ and $\dot{\mathbf{r}}$ in inertial coordinates. We have both in orbit frame coordinates, thus we need a DCM from the $\mathcal{O}$ frame to the $\mathcal{N}$ frame. This DCM is denoted as $[NO]$. For the (3-1-3) Euler angles (Ω, i, θ),

$$[ON] = [M_3(\theta)][M_1(i)][M_3(\Omega)] = \begin{bmatrix} \cos\theta & \sin\theta & 0 \\ -\sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos i & \sin i \\ 0 & -\sin i & \cos i \end{bmatrix} \begin{bmatrix} \cos\Omega & \sin\Omega & 0 \\ -\sin\Omega & \cos\Omega & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{7}$$

After carrying out the matrix multiplication, we get the following DCM [2]:

$$[ON] = \begin{bmatrix} \cos\theta\cos\Omega - \sin\theta\cos i\sin\Omega & \cos\theta\sin\Omega + \sin\theta\cos i\cos\Omega & \sin\theta\sin i \\ -(\sin\theta\cos\Omega + \cos\theta\cos i\sin\Omega) & -\sin\theta\sin\Omega + \cos\theta\cos i\cos\Omega & \cos\theta\sin i \\ \sin i\sin\Omega & -\sin i\cos\Omega & \cos i \end{bmatrix} \tag{8}$$

We know that $[NO] = [ON]^{-1} = [ON]^{\intercal}$, which is a straightforward action to code.

With this DCM, we then know that

$${}^{\mathcal{N}}\mathbf{r} = [NO]{}^{\mathcal{O}}\mathbf{r} = {}^{\mathcal{N}}\begin{bmatrix} r(\cos\theta\cos\Omega - \sin\theta\cos i\sin\Omega) \\ r(\cos\theta\sin\Omega + \sin\theta\cos i\cos\Omega) \\ r(\sin\theta\sin i) \end{bmatrix} \tag{9}$$

and

$${}^{\mathcal{N}}\dot{\mathbf{r}} = [NO]{}^{\mathcal{O}}\dot{\mathbf{r}} = {}^{\mathcal{N}}\begin{bmatrix} -r\dot{\theta}(\sin\theta\cos\Omega + \cos\theta\cos i\sin\Omega) \\ r\dot{\theta}(-\sin\theta\sin\Omega + \cos\theta\cos i\cos\Omega) \\ r\dot{\theta}(\cos\theta\sin i) \end{bmatrix} \tag{10}$$

However, this doesn't tell us how the Euler angles are changing, which is useful information to know. Luckily, we know that the rate of change of the Euler angles is related to ω with the following kinematic differential equation [2]:

$$\begin{bmatrix} \dot{\Omega} \\ \dot{i} \\ \dot{\theta} \end{bmatrix} = \frac{1}{\sin i}\begin{bmatrix} \sin\theta & \cos\theta & 0 \\ \cos\theta\sin i & -\sin\theta\sin i & 0 \\ -\sin\theta\cos i & -\cos\theta\cos i & \sin i \end{bmatrix} {}^{\mathcal{O}}\omega_{\mathcal{O}/\mathcal{N}} \tag{11}$$

We know that $\dot{\theta}$ is constant for each circular orbit and the satellite isn't doing any maneuvers to change Ω or i, meaning that $\theta(t) = \dot{\theta}t$ and i and Ω remain constant. This can be verified with equation 11 above since ${}^{\mathcal{O}}\omega_{\mathcal{O}/\mathcal{N}} = \begin{bmatrix} 0 \\ 0 \\ \dot{\theta} \end{bmatrix}$ at all times.

Finally, we can create a program that propagates ${}^{\mathcal{N}}\mathbf{r}$, ${}^{\mathcal{N}}\dot{\mathbf{r}}$, and the (3-1-3) Euler angles based on the scenario's initial conditions using the matrix math and algebraic expressions described above.

C. Task 1 Part 3: ${}^{\mathcal{N}}\mathbf{r}$ and ${}^{\mathcal{N}}\dot{\mathbf{r}}$ program verification at $t = 450$s (LMO) and $t = 1150$s (GMO)

At $t = 450s$, I found that $\boxed{{}^{\mathcal{N}}\mathbf{r}_{\text{LMO}}(450s) = {}^{\mathcal{N}}\begin{bmatrix}-669.3\\3227.5\\1883.2\end{bmatrix}\text{ km}}$ and $\boxed{{}^{\mathcal{N}}\dot{\mathbf{r}}_{\text{LMO}}(450s) = {}^{\mathcal{N}}\begin{bmatrix}-3.256\\-0.798\\0.210\end{bmatrix}\text{ km/s}}$.

At $t = 1150s$, I found that $\boxed{{}^{\mathcal{N}}\mathbf{r}_{\text{GMO}}(1150s) = {}^{\mathcal{N}}\begin{bmatrix}-5399.1\\-19697.6\\0.0\end{bmatrix}\text{ km}}$ and $\boxed{{}^{\mathcal{N}}\dot{\mathbf{r}}_{\text{GMO}}(1150s) = {}^{\mathcal{N}}\begin{bmatrix}1.397\\-0.383\\0.000\end{bmatrix}\text{ km/s}}$.

Both of these results were verified with Coursera as correct!

IV. Task 2: Orbit Frame Orientation

When simulating orbits, it is useful to define the orbit reference frame in terms of simply the current inertial position and velocity vectors. This frame is called the Hill Frame, and is defined as $\mathcal{H} : \{\hat{\imath}_r, \hat{\imath}_\theta, \hat{\imath}_h\}$, similar to the orbit frame. However, the unit vectors are defined as follows:

$$\hat{\imath}_r = \frac{\mathbf{r}}{|\mathbf{r}|}, \qquad \hat{\imath}_\theta = \hat{\imath}_h \times \hat{\imath}_r, \qquad \hat{\imath}_h = \frac{\mathbf{r} \times \dot{\mathbf{r}}}{|\mathbf{r} \times \dot{\mathbf{r}}|} \tag{12}$$

All of the code for this section can be found in Appendix A.C.

A. Task 2 Part 1: Analytical expression for $[HN]$

We know that the DCM between $\mathcal{H}$ and $\mathcal{N}$ can be written as

$$[HN] = [{}^{\mathcal{H}}\hat{n}_1, {}^{\mathcal{H}}\hat{n}_2, {}^{\mathcal{H}}\hat{n}_3] = \begin{bmatrix}{}^{\mathcal{N}}\hat{\imath}_r^\intercal\\{}^{\mathcal{N}}\hat{\imath}_\theta^\intercal\\{}^{\mathcal{N}}\hat{\imath}_h^\intercal\end{bmatrix} \tag{13}$$

Since we have expressions for $\hat{\imath}_r, \hat{\imath}_\theta,$ and $\hat{\imath}_h$, we can derive an analytical expression for $[HN]$ solely in terms of ${}^{\mathcal{N}}\mathbf{r}$ and ${}^{\mathcal{N}}\dot{\mathbf{r}}$. Then, using our previous program to obtain ${}^{\mathcal{N}}\mathbf{r}$ and ${}^{\mathcal{N}}\dot{\mathbf{r}}$, we can compute $[HN]$ at any point in time.

Firstly, we know that $|\mathbf{r}| = r$, so $\boxed{{}^{\mathcal{N}}\hat{\imath}_r = \frac{{}^{\mathcal{N}}\mathbf{r}}{r}}$

Secondly, we know that $\mathbf{r} \times \dot{\mathbf{r}} = r\hat{\imath}_r \times r\dot{\theta}\hat{\imath}_\theta = r^2\dot{\theta}\hat{\imath}_h$, hence $|\mathbf{r} \times \dot{\mathbf{r}}| = r^2\dot{\theta}$ and $\boxed{{}^{\mathcal{N}}\hat{\imath}_h = \frac{{}^{\mathcal{N}}\mathbf{r} \times {}^{\mathcal{N}}\dot{\mathbf{r}}}{r^2\dot{\theta}}}$.

Finally, we can simplify $\hat{\imath}_\theta = \hat{\imath}_h \times \hat{\imath}_r$ using vector math and the assumption that the nano-satellite's orbit is circular. Based on the Hill Frame definition, we know that

$$\hat{\imath}_\theta = \hat{\imath}_h \times \hat{\imath}_r = \frac{\mathbf{r} \times \dot{\mathbf{r}}}{|\mathbf{r} \times \dot{\mathbf{r}}|} \times \frac{\mathbf{r}}{|\mathbf{r}|} = \frac{\mathbf{r} \times \dot{\mathbf{r}}}{r^2\dot{\theta}} \times \frac{\mathbf{r}}{r} \tag{14}$$

By the triple cross product property, we know that $(\mathbf{A} \times \mathbf{B}) \times \mathbf{C} = \mathbf{B}(\mathbf{A} \cdot \mathbf{C}) - \mathbf{C}(\mathbf{A} \cdot \mathbf{B})$. Applying to equation 14,

$$\hat{\imath}_\theta = \frac{\mathbf{r} \times \dot{\mathbf{r}}}{r^2\dot{\theta}} \times \frac{\mathbf{r}}{r} = \frac{1}{r^2\dot{\theta}}\frac{1}{r}[\dot{\mathbf{r}}(\mathbf{r} \cdot \mathbf{r}) - \mathbf{r}(\mathbf{r} \cdot \dot{\mathbf{r}})] \tag{15}$$

Since the nano-satellite's orbit is circular, we know that $\mathbf{r} \cdot \dot{\mathbf{r}} = 0$ as the inertial position and velocity vectors are always orthogonal in a circular orbit. We also know that $\mathbf{r} \cdot \mathbf{r} = r^2$. Therefore,

$$\hat{\imath}_\theta = \frac{1}{r^2\dot{\theta}}\frac{1}{r}[\dot{\mathbf{r}}(\mathbf{r} \cdot \mathbf{r}) - \mathbf{r}(\mathbf{r} \cdot \dot{\mathbf{r}})] = \frac{1}{r^2\dot{\theta}}\frac{1}{r}[\dot{\mathbf{r}}(r^2)] = \frac{\dot{\mathbf{r}}}{r\dot{\theta}} \tag{16}$$

Therefore, $\boxed{{}^{\mathcal{N}}\hat{\imath}_\theta = \frac{{}^{\mathcal{N}}\dot{\mathbf{r}}}{r\dot{\theta}}}$

Putting everything together, we have

$$\boxed{[HN] = \begin{bmatrix} \frac{{}^{\mathcal{N}}\mathbf{r}^{\mathsf{T}}}{r} \\ \frac{{}^{\mathcal{N}}\dot{\mathbf{r}}^{\mathsf{T}}}{r\dot{\theta}} \\ \frac{({}^{\mathcal{N}}\mathbf{r}\times{}^{\mathcal{N}}\dot{\mathbf{r}})^{\mathsf{T}}}{r^2\dot{\theta}} \end{bmatrix}} \tag{17}$$

See Equations 9 and 10 for explicit definitions of ${}^{\mathcal{N}}\mathbf{r}$ and ${}^{\mathcal{N}}\dot{\mathbf{r}}$, for brevity's sake they are not repeated here as it would only result in the same DCM as in equation 8. In essence, the Hill Frame becomes the Orbit Frame, except instead of being defined by Euler angles it is now defined by the inertial position and velocity vectors!

B. Task 2 Part 2: $[HN]$ calculation program

All of the operations in part 1 are quite easy for a program like MATLAB to perform, and the orbit calculation function from the previous section allows us to find ${}^{\mathcal{N}}\mathbf{r}$ and ${}^{\mathcal{N}}\dot{\mathbf{r}}$ at any given point in time. Thus, this program is as simple as calling the previous function, carrying out the individual unit vector math, and finally concatenating everything into the final $[HN]$ matrix.

C. Task 2 Part 3: $[HN]$ program verification at $t = 300s$

At $t = 300s$, I found that $\boxed{[HN](300s) = \begin{bmatrix} -0.0465 & 0.8741 & 0.4834 \\ -0.9842 & -0.1229 & 0.1277 \\ 0.1710 & -0.4698 & 0.8660 \end{bmatrix}}$

Coursera verified this answer as correct!

V. Task 3: Sun-Pointing Reference Frame Orientation

One of the pointing scenarios of the nano-satellite is to point its solar panels at the sun, which is purely in the $\hat{n}_2$ direction since we are not modeling Mars' motion around the sun. To facilitate this mode, we need a reference DCM for the satellite to point to, namely, $[R_sN]$. The reference frame should have $\hat{r}_3$ pointing at the sun in the $\hat{n}_2$ direction, and $\hat{r}_1$ pointing in the $-\hat{n}_1$ direction. All of the code for this section can be found in Appendix A.D.

A. Task 3 Part 1: Analytical expression for $[R_sN]$

Similarly to $[HN]$, we know that

$$[R_sN] = [{}^{\mathcal{R}}\hat{n}_1, {}^{\mathcal{R}}\hat{n}_2, {}^{\mathcal{R}}\hat{n}_3] = \begin{bmatrix} {}^{\mathcal{N}}\hat{r}_1^{\mathsf{T}} \\ {}^{\mathcal{N}}\hat{r}_2^{\mathsf{T}} \\ {}^{\mathcal{N}}\hat{r}_3^{\mathsf{T}} \end{bmatrix} \tag{18}$$

We are told that $\hat{r}_3 = \hat{n}_2$ and $\hat{r}_1 = -\hat{n}_1$. To make the frame right handed, we need $\hat{r}_2 = \hat{r}_3 \times \hat{r}_1 = \hat{n}_2 \times -\hat{n}_1 = \hat{n}_3$.

Therefore, $\boxed{{}^{\mathcal{N}}\hat{r}_1 = {}^{\mathcal{N}}\begin{bmatrix} -1 \\ 0 \\ 0 \end{bmatrix}, {}^{\mathcal{N}}\hat{r}_2 = {}^{\mathcal{N}}\begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}, \text{and} {}^{\mathcal{N}}\hat{r}_3 = {}^{\mathcal{N}}\begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix}}$

Plugging this into the $[R_sN]$ definition, we get

$$\boxed{[R_sN] = \begin{bmatrix} -1 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \end{bmatrix}} \tag{19}$$

B. Task 3 Part 2: $[R_sN]$ calculation program

Since $[R_sN]$ doesn't change with time for this project, we can simply hard code it into a function that solely returns the DCM. If we were modeling the orbital motion of Mars, we would need to incorporate how the Mars frame rotates with respect to the Sun, but that is out of this project's scope.

C. Task 3 Part 3: $[R_sN]$ program verification at $t = 0s$

Plugging in $[R_sN]$ exactly as above was verified as correct on Coursera!

D. Task 3 Part 4: ${}^{\mathcal{N}}\omega_{R_s/N}$ derivation

Since this frame is defined directly as the inertial unit vectors and they don't rotate with time, ${}^{\mathcal{N}}\omega_{R_s/N}$ must be the zero vector. Verifying with Coursera, this is correct!

VI. Task 4: Nadir-Pointing Reference Frame Orientation

Another of the pointing scenarios of the nano-satellite is to point its science instruments at Mars' night side. The instruments are pointed along the $\hat{b}_1$ axis, and to be most effective they should point towards Mars' center along the nadir direction, $-\hat{\imath}_r$. This means we need another reference frame $[R_nN]$, whose $\hat{r}_1$ axis points at the planet in the $-\hat{\imath}_r$ direction and $\hat{r}_2$ axis points in the velocity direction $\hat{\imath}_\theta$. All of the code for this section can be found in Appendix A.E.

A. Task 4 Part 1: Analytical expression for $[R_nN]$

Again, we know that

$$[R_nN] = [{}^{\mathcal{R}}\hat{n}_1, {}^{\mathcal{R}}\hat{n}_2, {}^{\mathcal{R}}\hat{n}_3] = \begin{bmatrix} {}^{\mathcal{N}}\hat{r}_1^\intercal \\ {}^{\mathcal{N}}\hat{r}_2^\intercal \\ {}^{\mathcal{N}}\hat{r}_3^\intercal \end{bmatrix} \tag{20}$$

We are told that $\hat{r}_1 = -\hat{\imath}_r$ and $\hat{r}_2 = \hat{\imath}_\theta$. To make the frame right-handed, we need $\hat{r}_3 = \hat{r}_1 \times \hat{r}_2 = -\hat{\imath}_h$. These unit vectors are written in Hill/Orbit Frame coordinates, meaning we need to transform them with $[NH]$ or $[NO]$ to get them into inertial coordinates. Doing so results in the following:

$${}^{\mathcal{N}}\hat{r}_1 = \underbrace{[NH](-\hat{\imath}_r)}_{\text{using Hill Frame}} = \underbrace{{}^{\mathcal{N}}\begin{bmatrix} -(\cos\theta\cos\Omega - \sin\theta\cos i\sin\Omega) \\ -(\cos\theta\sin\Omega + \sin\theta\cos i\cos\Omega) \\ -(\sin\theta\sin i) \end{bmatrix}}_{\text{using Orbit Frame}} \tag{21}$$

$${}^{\mathcal{N}}\hat{r}_2 = \underbrace{[NH](\hat{\imath}_\theta)}_{\text{using Hill Frame}} = \underbrace{{}^{\mathcal{N}}\begin{bmatrix} -(\sin\theta\cos\Omega + \cos\theta\cos i\sin\Omega) \\ -\sin\theta\sin\Omega + \cos\theta\cos i\cos\Omega \\ \cos\theta\sin i \end{bmatrix}}_{\text{using Orbit Frame}} \tag{22}$$

$${}^{\mathcal{N}}\hat{r}_3 = \underbrace{[NH](-\hat{\imath}_h)}_{\text{using Hill Frame}} = \underbrace{{}^{\mathcal{N}}\begin{bmatrix} -\sin i\sin\Omega \\ \sin i\cos\Omega \\ -\cos i \end{bmatrix}}_{\text{using Orbit Frame}} \tag{23}$$

Therefore, if using the Hill Frame to convert,

$$\boxed{[R_nN] = \begin{bmatrix} (-[NH](\hat{\imath}_r))^\intercal \\ ([NH](\hat{\imath}_\theta))^\intercal \\ (-[NH](\hat{\imath}_h))^\intercal \end{bmatrix}} \tag{24}$$

and if using the Orbit Frame to convert,

$$\boxed{[R_nN] = \begin{bmatrix} -(\cos\theta\cos\Omega - \sin\theta\cos i\sin\Omega) & -(\cos\theta\sin\Omega + \sin\theta\cos i\cos\Omega) & -(\sin\theta\sin i) \\ -(\sin\theta\cos\Omega + \cos\theta\cos i\sin\Omega) & -\sin\theta\sin\Omega + \cos\theta\cos i\cos\Omega & \cos\theta\sin i \\ -\sin i\sin\Omega & \sin i\cos\Omega & -\cos i \end{bmatrix}} \tag{25}$$

The astute reader will notice that this is simply $[ON]$ from equation 8 with the first and third rows negative!

B. Task 4 Part 2: $[R_nN]$ calculation program

The program for calculating $[R_nN]$ looks largely like the program for calculating $[HN]$ from section IV.B, this time we will be calling the $[HN]$ program at a given point in time and using the $[R_nN]$ definition from equation 24.

C. Task 4 Part 3: ${}^{\mathcal{N}}\omega_{R_n/N}$ calculation program

Since $[R_nN]$ includes $\hat{\imath}_r$ and $\hat{\imath}_\theta$ in its definition, both of which are rotating about $\hat{\imath}_h$ at a constant rate $\dot{\theta}$, we know that ${}^{\mathcal{N}}\omega_{R_n/N}$ is non-zero. Because it contains both of those vectors, we can also say that ${}^{\mathcal{N}}\omega_{R_n/N}$ should rotate about the same axis, namely parallel to $\hat{\imath}_h$. Using vector math, we know that $\omega_{R_n/N} = \omega_{R_n/O} + \omega_{O/N}$. Since $[R_nN]$ rotates with $[ON]$, $\omega_{R_n/O} = \mathbf{0}$. Introducing coordinates, we ultimately get ${}^{\mathcal{N}}\omega_{R_n/N} = {}^{\mathcal{N}}\omega_{O/N}$. We know ${}^{\mathcal{O}}\omega_{O/N} = \dot{\theta}\hat{\imath}_h$, so ${}^{\mathcal{N}}\omega_{O/N} = [NO]{}^{\mathcal{O}}\omega_{O/N}$. Plugging in the DCM, we get

$${}^{\mathcal{N}}\omega_{Rn/N} = {}^{\mathcal{N}}\omega_{O/N} = \begin{bmatrix} \cos\theta\cos\Omega - \sin\theta\cos i\sin\Omega & -(\sin\theta\cos\Omega + \cos\theta\cos i\sin\Omega) & \sin i\sin\Omega \\ \cos\theta\sin\Omega + \sin\theta\cos i\cos\Omega & -\sin\theta\sin\Omega + \cos\theta\cos i\cos\Omega & -\sin i\cos\Omega \\ \sin\theta\sin i & \cos\theta\sin i & \cos i \end{bmatrix} {}^{\mathcal{O}}\begin{bmatrix} 0 \\ 0 \\ \dot{\theta} \end{bmatrix} \tag{26}$$

Finally,

$$\boxed{{}^{\mathcal{N}}\omega_{R_n/N} = {}^{\mathcal{N}}\begin{bmatrix} \dot{\theta}\sin i\sin\Omega \\ -\dot{\theta}\sin i\cos\Omega \\ \dot{\theta}\cos i \end{bmatrix}} \tag{27}$$

D. Task 4 Part 4: $[R_nN]$ program verification at $t = 330s$

At $t = 330s$, I got $\boxed{[R_nN] = \begin{bmatrix} 0.0726 & -0.8706 & -0.4866 \\ -0.9826 & -0.1461 & 0.1148 \\ -0.1710 & -0.4698 & -0.8660 \end{bmatrix}}$

Coursera verified this as correct!

E. Task 4 Part 5: ${}^{\mathcal{N}}\omega_{R_n/N}$ program verification at $t = 330s$

At $t = 330s$, I got $\boxed{{}^{\mathcal{N}}\omega_{R_n/N} = {}^{\mathcal{N}}\begin{bmatrix} 0.1513\text{e-}3 \\ -0.4157\text{e-}3 \\ 0.7663\text{e-}3 \end{bmatrix}}$

Coursera verified this as correct!

VII. Task 5: GMO-Pointing Reference Frame Orientation

The final pointing scenario of the nano-satellite is to point its communication antenna towards the mothercraft. The antenna is pointed along the $-\hat{b}_1$ axis, and the nano-satellite should only point at the mothercraft when it is visible and on Mars' night side. To facilitate this scenario, we need a reference frame $[R_c N]$, whose $-\hat{r}_1$ axis points towards the mothercraft. Since we are pointing at the mothercraft, which has an inertial position vector of its own, we need a difference vector $\Delta\mathbf{r} = \mathbf{r}_{\text{GMO}} - \mathbf{r}_{\text{LMO}}$. Then,

$$\hat{r}_1 = \frac{-\Delta\mathbf{r}}{|\Delta\mathbf{r}|}, \qquad \hat{r}_2 = \frac{\Delta\mathbf{r} \times \hat{n}_3}{|\Delta\mathbf{r} \times \hat{n}_3|}, \qquad \hat{r}_3 = \hat{r}_1 \times \hat{r}_2. \tag{28}$$

All of the code for this section can be found in Appendix A.F.

A. Task 5 Part 1: Analytical expression for $[R_c N]$

Once again, we know that

$$[R_c N] = [{}^{\mathcal{R}}\hat{n}_1, {}^{\mathcal{R}}\hat{n}_2, {}^{\mathcal{R}}\hat{n}_3] = \begin{bmatrix} {}^{\mathcal{N}}\hat{r}_1^{\intercal} \\ {}^{\mathcal{N}}\hat{r}_2^{\intercal} \\ {}^{\mathcal{N}}\hat{r}_3^{\intercal} \end{bmatrix} \tag{29}$$

If we calculate $\mathbf{r}_{\text{GMO}}$ and $\mathbf{r}_{\text{LMO}}$ in inertial coordinates and denote ${}^{\mathcal{N}}\Delta\mathbf{r} = {}^{\mathcal{N}}\begin{bmatrix} a \\ b \\ c \end{bmatrix}$, then $\boxed{{}^{\mathcal{N}}\hat{r}_1 = \frac{{}^{\mathcal{N}}\mathbf{r}_{\text{LMO}} - {}^{\mathcal{N}}\mathbf{r}_{\text{GMO}}}{|{}^{\mathcal{N}}\mathbf{r}_{\text{GMO}} - {}^{\mathcal{N}}\mathbf{r}_{\text{LMO}}|} = \frac{{}^{\mathcal{N}}\begin{bmatrix} -a \\ -b \\ -c \end{bmatrix}}{a^2 + b^2 + c^2}}$

Next, we know that $\hat{r}_2 = \frac{\Delta\mathbf{r} \times \hat{n}_3}{|\Delta\mathbf{r} \times \hat{n}_3|}$. When we take the cross product of $\Delta\mathbf{r}$ with $\hat{n}_3$, the resulting vector will have a 0 component in the $\hat{n}_3$ direction. Therefore, $\boxed{{}^{\mathcal{N}}\hat{r}_2 = \frac{\begin{bmatrix} a \\ b \\ c \end{bmatrix} \times \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}}{\left|\begin{bmatrix} a \\ b \\ c \end{bmatrix} \times \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}\right|} = \frac{{}^{\mathcal{N}}\begin{bmatrix} b \\ -a \\ 0 \end{bmatrix}}{a^2 + b^2}}$

Finally, we know that $\hat{r}_3 = \hat{r}_1 \times \hat{r}_2 = \frac{{}^{\mathcal{N}}\begin{bmatrix} -a \\ -b \\ -c \end{bmatrix}}{a^2+b^2+c^2} \times \frac{{}^{\mathcal{N}}\begin{bmatrix} b \\ -a \\ 0 \end{bmatrix}}{a^2+b^2}$. Doing the math, we get

$$\boxed{{}^{\mathcal{N}}\hat{r}_3 = \frac{{}^{\mathcal{N}}\begin{bmatrix} -ac \\ -bc \\ a^2 + b^2 \end{bmatrix}}{(a^2 + b^2)(a^2 + b^2 + c^2)}} \tag{30}$$

Thus,

$$\boxed{[R_c N] = \begin{bmatrix} \frac{-a}{a^2+b^2+c^2} & \frac{-b}{a^2+b^2+c^2} & \frac{-c}{a^2+b^2+c^2} \\ \frac{b}{a^2+b^2} & \frac{-a}{a^2+b^2} & 0 \\ \frac{-ac}{(a^2+b^2)(a^2+b^2+c^2)} & \frac{-bc}{(a^2+b^2)(a^2+b^2+c^2)} & \frac{a^2+b^2}{(a^2+b^2)(a^2+b^2+c^2)} \end{bmatrix}} \tag{31}$$

Where, again, ${}^{\mathcal{N}}\Delta\mathbf{r} = {}^{\mathcal{N}}\begin{bmatrix} a \\ b \\ c \end{bmatrix}$.

B. Task 5 Part 2: $[R_cN]$ calculation program

Writing a function to return $[R_cN]$ as a function is fairly straightforward. We simply have to call the program from section IV.B twice, once for the nano-satellite and once for the mothercraft. We can then compute ${}^{\mathcal{N}}\Delta\mathbf{r}$ at that instant in time, and use the $[R_cN]$ definition from equation 31 to get $[R_cN]$ at that point!

C. Task 5 Part 3: ${}^{\mathcal{N}}\omega_{R_c/N}$ calculation program

We know that $\frac{{}^{\mathcal{N}}\mathrm{d}}{\mathrm{d}t}[R_cN] = -[{}^{\mathcal{R}}\tilde{\omega}_{R_c/N}][R_cN]$, which means that if we can solve for $[{}^{\mathcal{R}}\tilde{\omega}_{R_c/N}]$ we can back out ${}^{\mathcal{N}}\omega_{R_c/N}$. We can get $[R_cN]$ from equation 31, but an analytical expression for $\frac{{}^{\mathcal{N}}\mathrm{d}}{\mathrm{d}t}[R_cN]$ is very difficult to get. Since both the mothercraft and nano-satellite are moving along their own respective orbits, the coefficients a, b, and c from ${}^{\mathcal{N}}\Delta\mathbf{r}$ are changing in a complicated way at any given point in time. Luckily, we have access to computers and numerical methods, meaning we can use a finite difference quotient. A finite difference quotient allows us to approximate the derivative of some function f over a discrete timestep dt like so:

$$\frac{\mathrm{d}f}{\mathrm{d}t} \approx \frac{f(t_0 + dt) - f(t_0)}{dt} \tag{32}$$

In fact, this is how derivatives are defined! Simply take the limit of dt to 0, and the derivative will match exactly.

In our program, we can now solve for $\frac{{}^{\mathcal{N}}\mathrm{d}}{\mathrm{d}t}[R_cN]$ using the finite difference quotient at a given point in time t_0, using some dt that matches the output of the program from section III.B, and solve for $[{}^{\mathcal{R}}\tilde{\omega}_{R_c/N}]$ like so:

$$[{}^{\mathcal{R}}\tilde{\omega}_{R_c/N}] = -\frac{{}^{\mathcal{N}}\mathrm{d}}{\mathrm{d}t}[R_cN][R_cN]^{-1} = -\frac{{}^{\mathcal{N}}\mathrm{d}}{\mathrm{d}t}[R_cN][NR_c] \tag{33}$$

Then, since we know that $[{}^{\mathcal{R}}\tilde{\omega}_{R_c/N}] = {}^{\mathcal{R}}\begin{bmatrix} 0 & -\omega_3 & \omega_2 \\ \omega_3 & 0 & -\omega_1 \\ -\omega_2 & \omega_1 & 0 \end{bmatrix}$, we can construct ${}^{\mathcal{R}}\omega_{R_c/N} = {}^{\mathcal{R}}\begin{bmatrix} \omega_1 \\ \omega_2 \\ \omega_3 \end{bmatrix}$.

Finally, we need to convert ${}^{\mathcal{R}}\omega_{R_c/N}$ into inertial coordinates using $[R_cN]$ like so:

$$\boxed{{}^{\mathcal{N}}\omega_{R_c/N} = [R_cN]^{-1}\,{}^{\mathcal{R}}\omega_{R_c/N} = [NR_c]\,{}^{\mathcal{R}}\omega_{R_c/N}} \tag{34}$$

D. Task 5 Part 4: $[R_cN]$ program verification at $t = 330s$

At $t = 330$s, I got $\boxed{[R_cN] = \begin{bmatrix} 0.2655 & 0.9609 & 0.0784 \\ -0.9639 & 0.2663 & 0 \\ -0.0209 & -0.0755 & 0.9969 \end{bmatrix}}$.

Coursera verified this as correct!

E. Task 5 Part 5: ${}^{\mathcal{N}}\omega_{R_c/N}$ program verification at $t = 330s$

At $t = 330$s, I got $\boxed{{}^{\mathcal{N}}\omega_{R_c/N} = {}^{\mathcal{N}}\begin{bmatrix} 0.0198\text{e-}3 \\ -0.0055\text{e-}3 \\ 0.1913\text{e-}3 \end{bmatrix}}$

Coursera verified this as correct!

VIII. Task 6: Attitude Error Evaluation

For pointing missions, it is important to be able to calculate the attitude and angular velocity errors of some frame $\mathcal{B}$ relative to another frame $\mathcal{R}$. In this mission, attitude tracking error is denoted as $\sigma_{B/R}$ and angular velocity tracking error is denoted as ${}^{\mathcal{B}}\omega_{B/R}$. All of the code for this section can be found in Appendix A.G.

A. Task 6 Part 1: Tracking error calculation program

We know that $[BR] = [BN][NR]$ and ${}^{\mathcal{B}}\omega_{B/R} = {}^{\mathcal{B}}\omega_{B/N} - {}^{\mathcal{B}}\omega_{R/N} = {}^{\mathcal{B}}\omega_{B/N} - [BN]^{\mathcal{N}}\omega_{R/N}$. Therefore, this program is as simple as computing the MRP set from the DCM created in the first matrix multiplication, then carrying out the angular velocity vector math using the nano-satellite's current body angular velocity and the inertial angular velocity found from programs V.D, VI.C, and VII.C.

B. Task 6 Part 2: Tracking error program verification at t_0

At $t = t_0$, I got the following results in table 3, which Coursera verified as correct:

Table 3 Tracking errors at $t = t_0$

Reference frame	$\sigma_{B/R}$	${}^{\mathcal{B}}\omega_{B/R}$
Sun-pointing	$\begin{bmatrix} -0.7754 \\ -0.4739 \\ 0.0431 \end{bmatrix}$	$\begin{bmatrix} 0.0175 \\ 0.0305 \\ -0.0384 \end{bmatrix}$ rad/s
Nadir-pointing	$\begin{bmatrix} 0.2623 \\ 0.5547 \\ 0.0394 \end{bmatrix}$	$\begin{bmatrix} 0.0168 \\ 0.0309 \\ -0.0389 \end{bmatrix}$ rad/s
GMO-pointing	$\begin{bmatrix} 0.0170 \\ -0.3828 \\ 0.2076 \end{bmatrix}$	$\begin{bmatrix} 0.0173 \\ 0.0307 \\ -0.0384 \end{bmatrix}$ rad/s

IX. Task 7: Numerical Attitude Estimator

Since the point of this project is to simulate a pointing mission, we need some way to numerically integrate the attitude of the nano-satellite under some control and/or disturbance torque. All of the code for this section can be found in Appendix A.H.

A. Task 7 Part 1: RK4 integrator

To integrate attitude over this simulation, I created a 4th-order Runga-Kutta integrator. Details on the RK4 algorithm can be found in [1], all I had to do was define $\dot{\mathbf{X}} = f(\mathbf{X})$ with a custom function. In this case, $\mathbf{X} = \begin{bmatrix} \sigma_{B/N} \\ {}^{\mathcal{B}}\omega_{B/N} \end{bmatrix}$. Then, we know from [2] that $\boxed{\dot{\mathbf{X}} = f(\mathbf{X}) = \begin{bmatrix} \dot{\sigma}_{B/N} \\ {}^{\mathcal{B}}\dot{\omega}_{B/N} \end{bmatrix} = \begin{bmatrix} \frac{1}{4}[(1-\sigma^2)[I_{3x3}] + 2\tilde{\sigma} + 2\sigma\sigma^\intercal]{}^{\mathcal{B}}\omega_{B/N} \\ {}^{\mathcal{B}}[I]^{-1}(-[{}^{\mathcal{B}}\tilde{\omega}_{B/N}]{}^{\mathcal{B}}[I]{}^{\mathcal{B}}\omega_{B/N} + \mathbf{u}) \end{bmatrix}}$.

With $f(\mathbf{X})$ defined, we can use it in our RK4 integrator under a series of different controller scenarios to simulate the pointing mission. For the rest of the tasks, I will be using an RK4 integrator timestep of 1 second, as requested by [1].

B. Task 7 Part 2: RK4 verification at $t = 500$s, no control torque

In this scenario, no control torque is applied so $\mathbf{u} = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}$ Nm. The goal is to show that the RK4 integrator properly propagates the attitude with no torques first, then progress to more complicated scenarios. I ran this scenario, and got the following outputs at $t = 500$ seconds:

$$\boxed{{}^{\mathcal{B}}H(500\text{s}) = {}^{\mathcal{B}}I^{\mathcal{B}}\omega_{B/N} = \begin{bmatrix} 0.138 \\ 0.133 \\ -0.316 \end{bmatrix} \text{kgm}^2/\text{s}}$$

$$\boxed{T(500\text{s}) = {}^{\mathcal{B}}\omega_{B/N}^{\intercal}\, {}^{\mathcal{B}}I^{\mathcal{B}}\omega_{B/N} = 0.00938 \text{ J}}$$

$\boxed{{}^{\mathcal{N}}H(500\text{s}) = [NB]^{\mathcal{B}}I^{\mathcal{B}}\omega_{B/N} = \begin{bmatrix} -0.264 \\ 0.253 \\ 0.055 \end{bmatrix} \text{kgm}^2/\text{s}}$, where $[NB] = [BN]^\intercal = \left[I_{3x3} + \frac{8[\tilde{\sigma_{B/N}}]^2 - 4[\tilde{\sigma_{B/N}}]}{(1+\sigma_{B/N}^2)^2}\right]^\intercal$. Coursera verified all these answers as correct!

As for the integrator, it seems to be doing its job. Figure 4 shows the results of integrating this scenario for 1000 seconds on the state vector $\mathbf{X}$, as well as the control input $\mathbf{u}$:

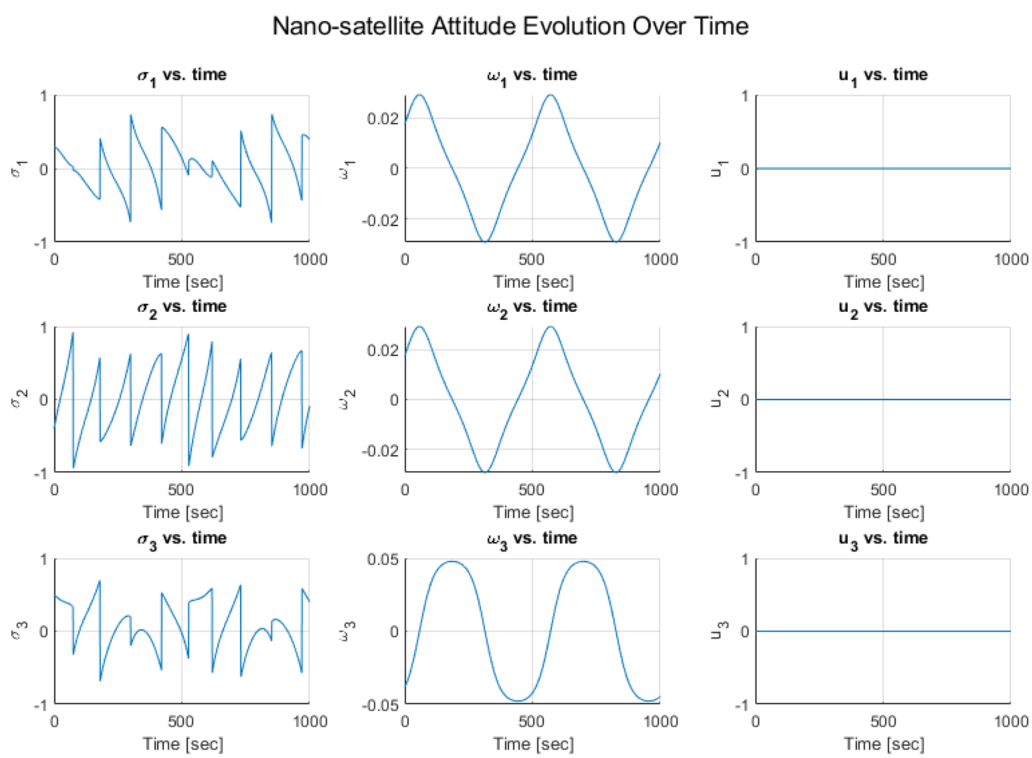

Fig. 4 State vector response to no control torque

C. Task 7 Part 2: RK4 verification at $t = 100$s, fixed control torque

In this scenario, a fixed control torque of $\mathbf{u} = {}^{\mathcal{B}}\begin{bmatrix} 0.01 \\ -0.01 \\ 0.02 \end{bmatrix}$ Nm is applied to the nano-satellite. The goal is to show that the RK4 integrator properly integrates the attitude with a net torque before moving onto non-fixed controller scenarios. After running this scenario, I got the following attitude output at $t = 100$s:

$\boxed{\sigma_{B/N}(100\text{s}) = \begin{bmatrix} -0.227 \\ -0.641 \\ 0.243 \end{bmatrix}}$. Coursera verified this as correct!

As for the integrator, it seems to handle a constant torque well. Figure 5 shows the results of integrating this scenario for 1000 seconds on the state vector **X**, as well as the control input **u**:

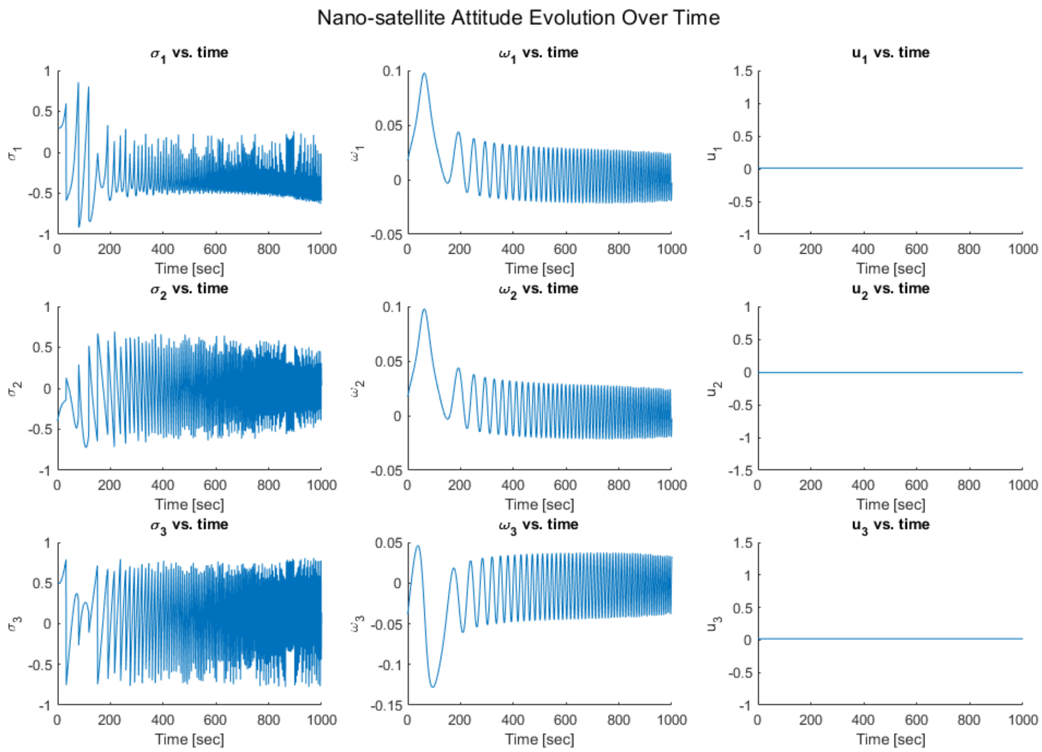

Fig. 5 State vector response to a fixed control torque

We can see that the angular velocities oscillate faster and faster around 0, which would be expected for a constant torque acting on the spacecraft. Similarly, we can see the attitude oscillations accelerate as the torque is applied, resulting in a chaotic plot. There is likely some numerical artifacting happening due to the chosen time step of 1 second, as at some point the attitude and angular velocities will be oscillating much faster than can be modeled over 1 second.

X. Task 8: Sun Pointing Control

One of the pointing scenarios of this mission is to point the nano-satellite's solar panels at the sun when the it is on the sunlit side of Mars. In order to achieve this, we need to develop an appropriate controller that can point at a stationary, non-moving frame. Since the sun is considered stationary over the simulation time of this project, we can use the $[R_sN]$ frame that we calculated in Section V.A to tune our controller for the rest of the project. All of the code for this section can be found in Appendix A.I.

A. Task 8 Part 1: PD control implementation

For this mission, we are implementing the simple PD control law ${}^{\mathcal{B}}\mathbf{u} = -K\sigma_{B/R} - P\,{}^{\mathcal{B}}\omega_{B/R}$. We can get $\sigma_{B/R}$ and ${}^{\mathcal{B}}\omega_{B/R}$ by passing the desired reference frame into the error tracking program in section VIII.A, then simply implement the control law for **u** inside the RK4 integrator in section IX.A. The gains K and P will be determined in the next section.

B. Task 8 Part 2: K and P gain selection

To determine K and P, we can use the linearized closed loop dynamics of a regulator problem, specifically the reduced equations for performance characteristics by inertia axis i:

$$T_i = \frac{2I_i}{P_i}, \text{ time decay constant} \tag{35}$$

$$\xi_i = \frac{P_i}{\sqrt{KI_i}}, \text{ damping ratio} \tag{36}$$

[1] specifies a maximum T_i of 120 seconds and a maximum ξ_i of 1, or critically damped. We can see from equation 35 that the largest time decay will occur on the largest inertia axis. In this scenario, the largest inertia is $I_1 = 10$ kgm^2. Thus, to achieve the longest time decay of 120 seconds, $P = \frac{2I_1}{T_{max}} = \frac{20}{120} = \frac{1}{6}$. From equation 36, we know that the largest damping ratio will occur on the smallest inertia axis. In this scenario, the smallest inertia is $I_2 = 5$ kgm^2. Thus, to achieve the largest damping ratio of 1, $K = \frac{P^2}{\xi_{max}^2 I_2} = \frac{\frac{1}{36}}{5} = \frac{1}{180}$. Therefore, the gains that will be used for the rest of the project are $\boxed{K = \frac{1}{6} = 0.1\bar{6}7 \text{ and } P = \frac{1}{180} = 0.0056}$

C. Task 8 Part 3: Sun Pointing Control mode verification at t = 15s, 100s, 200s, and 400s

After implementing the PD control law with the above gains, I propagated the simulation for 1000 seconds solely with sun pointing control, or the $[R_sN]$ frame from the program in section V.B. The results can be seen in Figure 6:

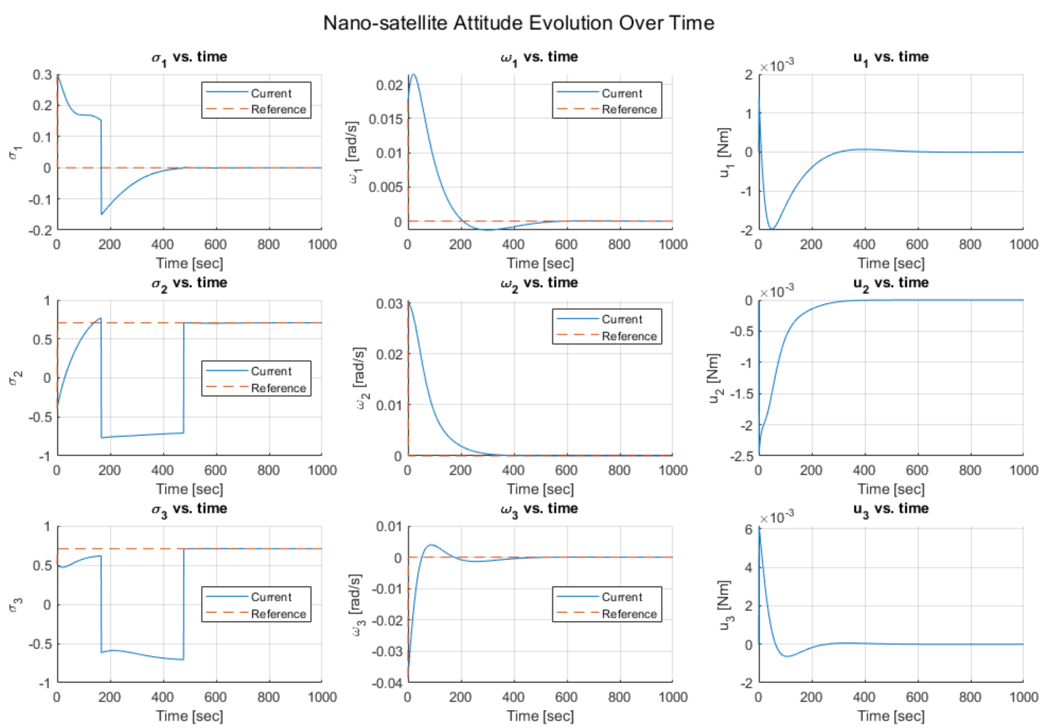

Fig. 6 Sun pointing simulation response

As can be seen, the attitude of the nano-satellite converges to the $[R_sN]$ frame, or $\sigma_{B/N} = \begin{bmatrix} 0 \\ \frac{\sqrt{2}}{2} \\ \frac{\sqrt{2}}{2} \end{bmatrix}$. At t =15, 100, 200, and 400 seconds, the body attitude is as follows in table 4:

Table 4 Body attitude at various simulation times - sun pointing

t **[sec]**	15	100	200	400
$\sigma_{B/N}$	$\begin{bmatrix} 0.266 \\ -0.160 \\ 0.473 \end{bmatrix}$	$\begin{bmatrix} 0.169 \\ 0.548 \\ 0.579 \end{bmatrix}$	$\begin{bmatrix} -0.118 \\ -0.758 \\ -0.591 \end{bmatrix}$	$\begin{bmatrix} -0.01 \\ -0.719 \\ -0.686 \end{bmatrix}$

Coursera verified these attitudes as correct!

XI. Task 9: Nadir Pointing Control

Now that we have a controller that can reliably point at a stationary frame, we can move onto tracking moving frames. The next scenario in the pointing mission is to point at Mars, namely in the Nadir direction. We calculated this frame, the $[R_nN]$ frame, in section VI.A. All of the code for this section can be found in Appendix A.J.

A. Task 9 Part 1: Nadir Pointing Control mode implementation

To implement Nadir pointing control, all we need to do is pass in $[R_nN]$ into the error tracking program from section VIII.A. We can calculate $[R_nN]$ with the program from section VI.B, and the control law is the same as in section X.A.

B. Task 9 Part 2: Nadir Pointing Control mode verification

After changing from the sun pointing frame to the nadir pointing frame, I once again propagated the simulation for 1000 seconds solely pointing at $[R_nN]$. The results can be seen in Figure 7:

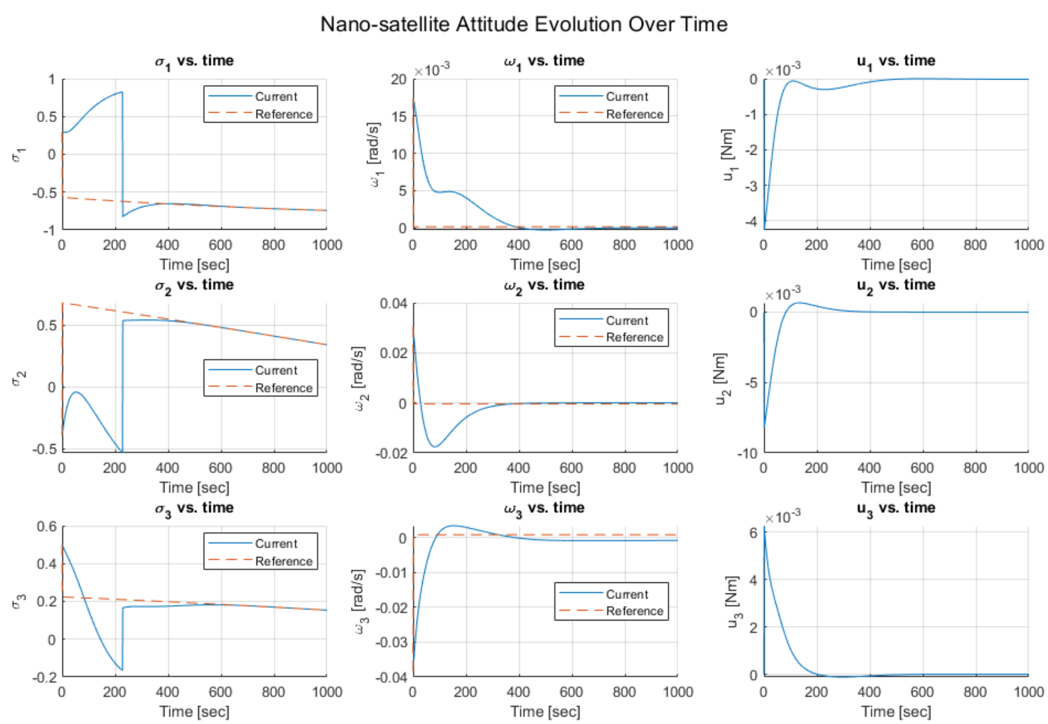

Fig. 7 Nadir pointing simulation response

The attitude looks to converge to the nadir pointing frame, as specified in section VI.A. At t =15, 100, 200, and 400 seconds, the body attitude is as follows in table 5:

Table 5 Body attitude at various simulation times - nadir pointing

t [sec]	15	100	200	400
$\sigma_{B/N}$	$\begin{bmatrix} 0.291 \\ -0.191 \\ 0.454 \end{bmatrix}$	$\begin{bmatrix} 0.566 \\ -0.137 \\ 0.152 \end{bmatrix}$	$\begin{bmatrix} 0.796 \\ -0.460 \\ -0.127 \end{bmatrix}$	$\begin{bmatrix} -0.653 \\ 0.535 \\ 0.175 \end{bmatrix}$

Coursera verified these attitudes as correct!

XII. Task 10: GMO Pointing Control

Next, the satellite needs to point at the mothercraft to relay science data and communicate with mission control. The necessary frame is $[R_cN]$, calculated in section VII.A. All of the code for this section can be found in Appendix A.K.

A. Task 10 Part 1: GMO Pointing Control mode implementation

To implement GMO pointing control, we need to pass $[R_cN]$ into the error tracking program from section VIII.A. We can calculate $[R_cN]$ with the program from section VII.B, and the control law is the same as in section X.A.

B. Task 10 Part 2: GMO Pointing Control mode verification

After changing from the nadir pointing frame to the GMO pointing frame, I again propagated the simulation for 1000 seconds solely pointing at $[R_cN]$. The results can be seen in Figure 8:

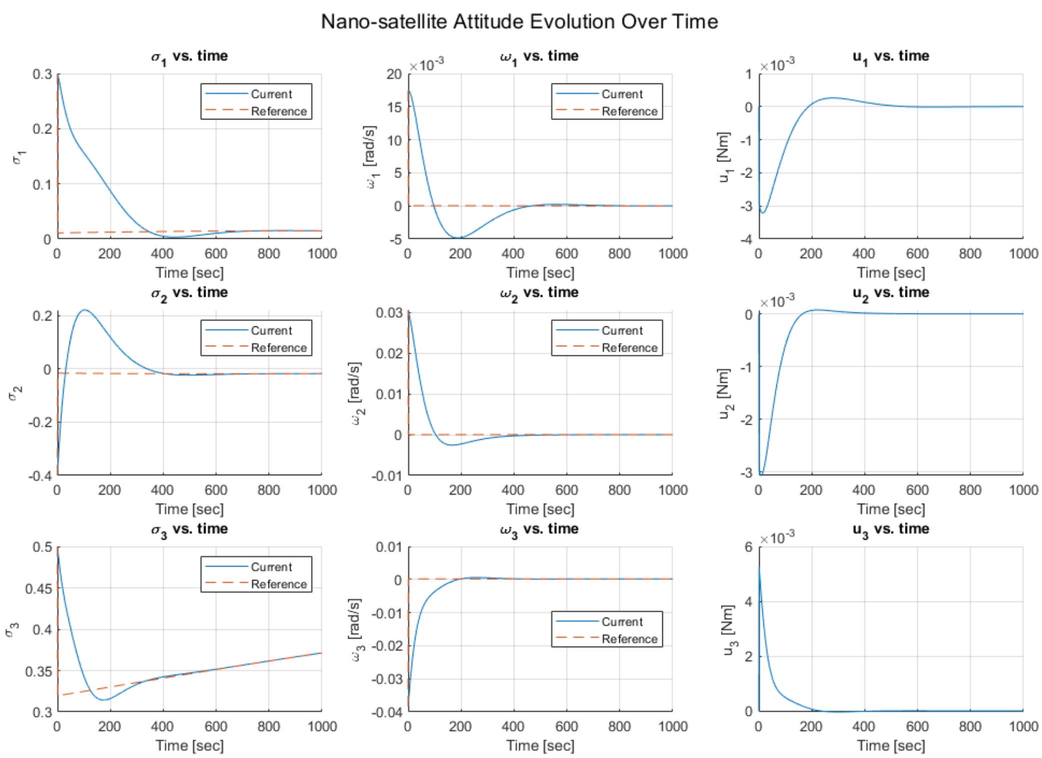

Fig. 8 GMO pointing simulation response

The attitude looks to converge to the GMO pointing frame, as specified in section VII.A. At t =15, 100, 200, and 400 seconds, the body attitude is as follows:

Table 6 Body attitude at various simulation times - GMO pointing

t **[sec]**	15	100	200	400
$\sigma_{B/N}$	$\begin{bmatrix} 0.265 \\ -0.169 \\ 0.459 \end{bmatrix}$	$\begin{bmatrix} 0.156 \\ 0.222 \\ 0.343 \end{bmatrix}$	$\begin{bmatrix} 0.087 \\ 0.119 \\ 0.316 \end{bmatrix}$	$\begin{bmatrix} 0.005 \\ -0.016 \\ 0.342 \end{bmatrix}$

Coursera verified these attitudes as correct!

XIII. Task 11: Mission Scenario Simulation

Finally, with all pointing modes programmed we can simulate the entire mission. All of the code for this section can be found in Appendix A.L.

A. Task 11 Part 1: Pointing logic implementation

In order to simulate the mission, we need to program a way to switch between reference frames automatically based on where the nano-satellite and mothercraft are in their respective orbits. The mission scenarios are laid out in table 1, so the pointing logic can be implemented as a state machine that matches the table. The state machine can be seen in figure 9:

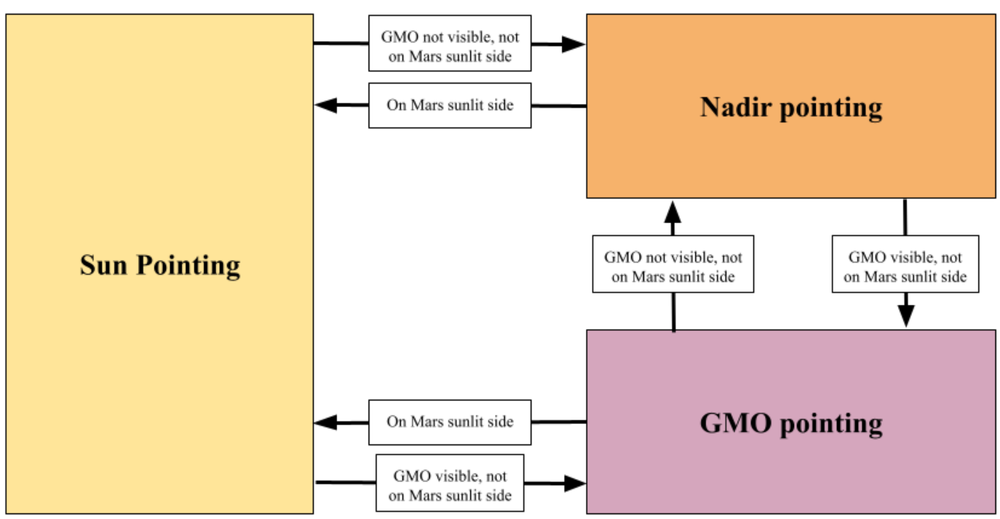

Fig. 9 Pointing mission state machine

Programmatically, the state transitions occur according to table 7:

Table 7 Nano-satellite Pointing Transitions

Pointing mode	Condition
Sun-pointing	${}^{N}\mathbf{r}_{LMO} \cdot \hat{n}_2 > 0$
GMO-pointing	$\cos^{-1}\left(\frac{{}^{N}\mathbf{r}_{LMO} \cdot {}^{N}\mathbf{r}_{GMO}}{\lvert {}^{N}\mathbf{r}_{LMO}\rvert \lvert {}^{N}\mathbf{r}_{GMO}\rvert}\right) \leq 35°$, ${}^{N}\mathbf{r}_{LMO} \cdot \hat{n}_2 \leq 0$
Nadir-pointing	$\cos^{-1}\left(\frac{{}^{N}\mathbf{r}_{LMO} \cdot {}^{N}\mathbf{r}_{GMO}}{\lvert {}^{N}\mathbf{r}_{LMO}\rvert \lvert {}^{N}\mathbf{r}_{GMO}\rvert}\right) \geq 35°$, ${}^{N}\mathbf{r}_{LMO} \cdot \hat{n}_2 \leq 0$

We can then implement the logic in Table 7 into our RK4 algorithm from section IX.A to automatically switch the reference pointing frame at different points along the orbit!

B. Task 11 Part 2: Mission simulation verification

After implementing the pointing logic from table 7, I propagated the simulation for 6500 seconds. The results are shown in Figure 10:

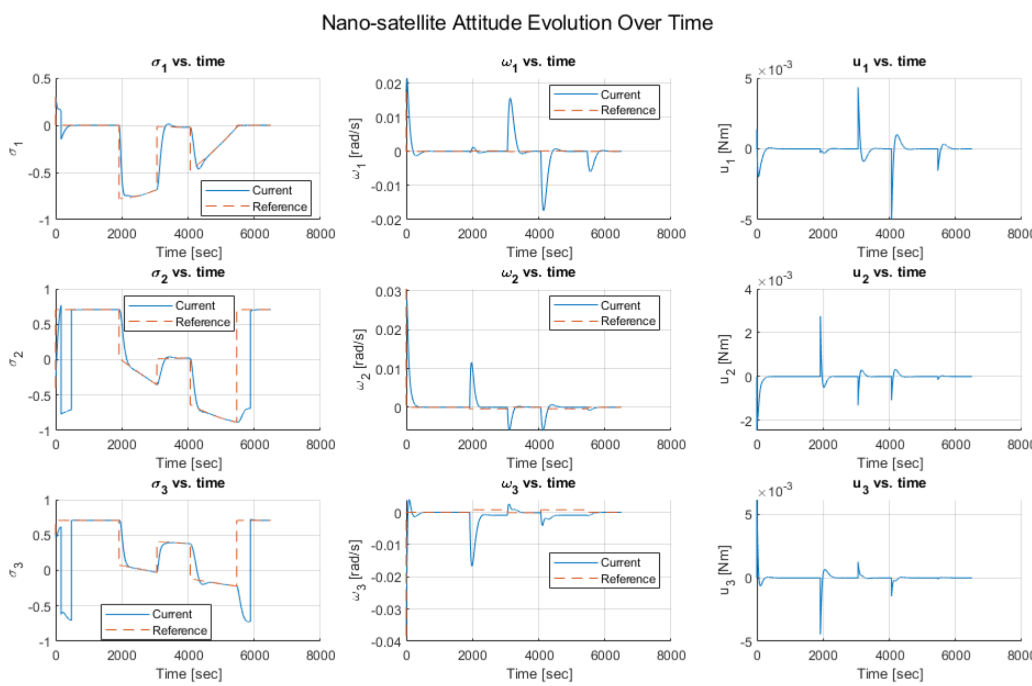

Fig. 10 6500 second mission simulation response

The attitude looks to converge to each reference frame as needed, within performance requirements! At t =300, 2100, 3400, 4400, and 5600 seconds, the body attitude is as follows in table 8:

Table 8 Body attitude at various simulation times - full mission simulation

t [sec]	300	2100	3400	4400	5600
$\sigma_{B/N}$	$\begin{bmatrix}-0.044\\-0.739\\-0.631\end{bmatrix}$	$\begin{bmatrix}-0.746\\0.114\\0.158\end{bmatrix}$	$\begin{bmatrix}0.013\\0.040\\0.391\end{bmatrix}$	$\begin{bmatrix}-0.443\\-0.732\\-0.188\end{bmatrix}$	$\begin{bmatrix}-0.001\\-0.826\\-0.504\end{bmatrix}$

Coursera verified these attitudes as correct!

At each of these times, the corresponding orbital situation and body attitude can be seen in figures 11, 12, 13, 14, and 15:

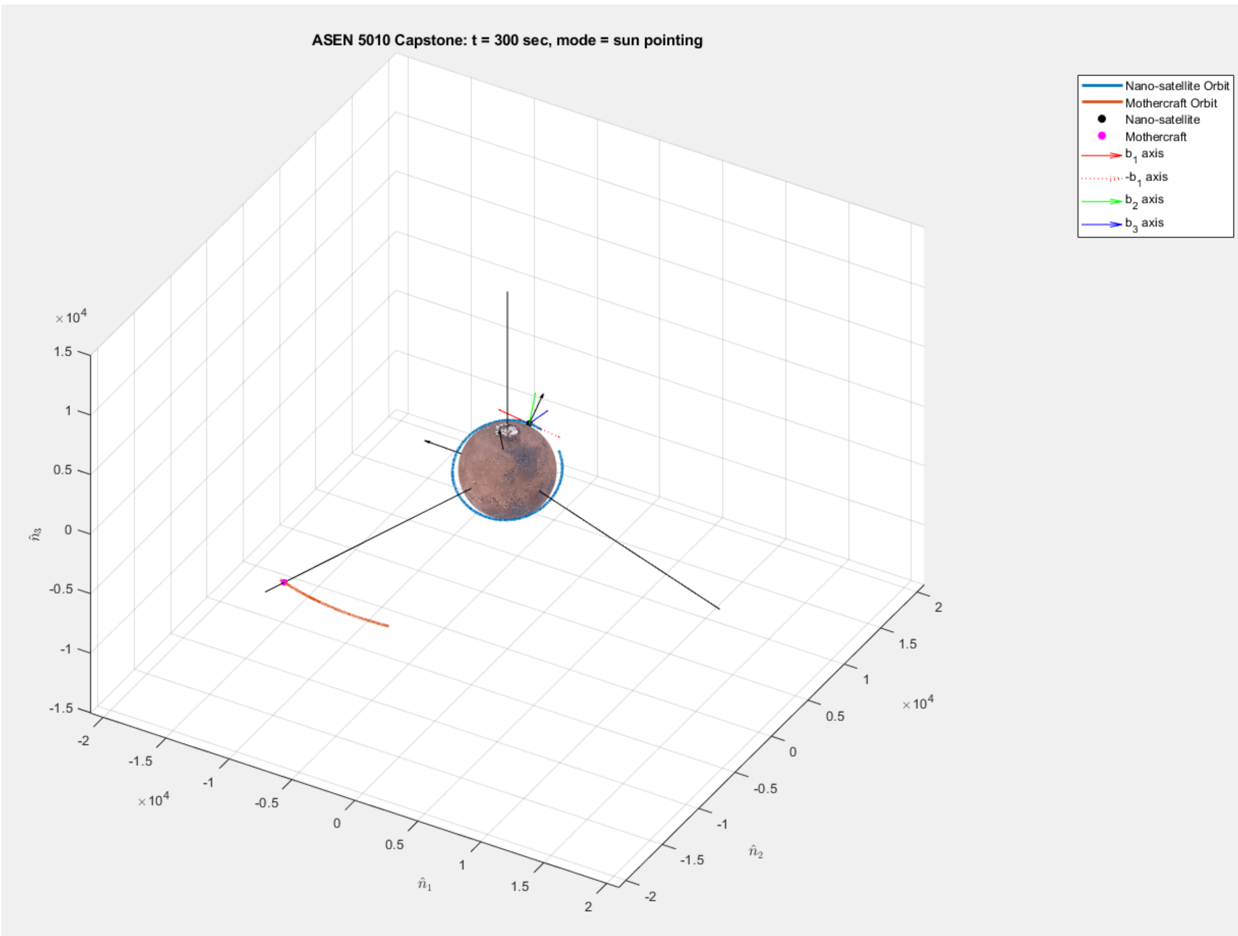

Fig. 11 Nano-satellite pointing state at t = 300 sec

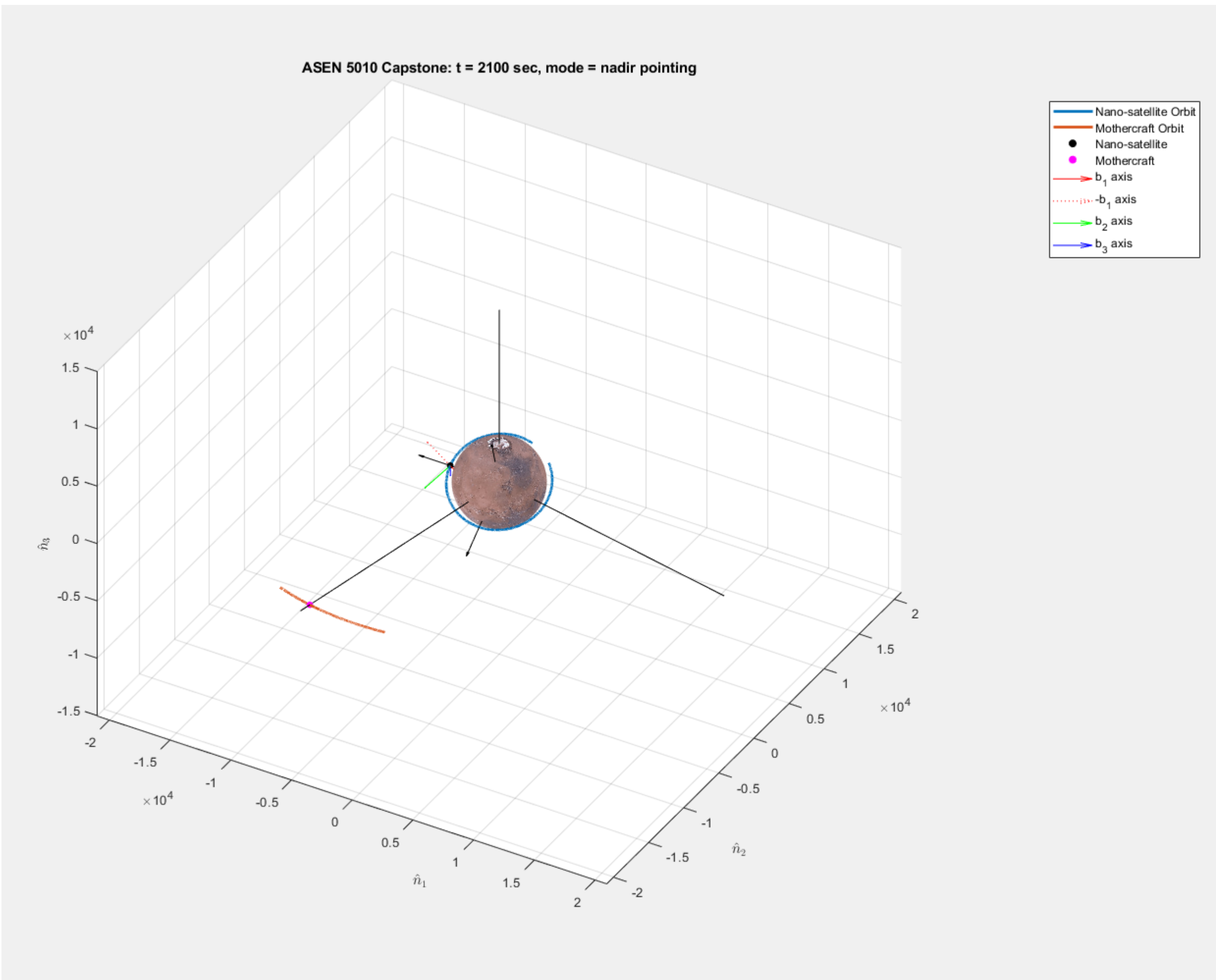

Fig. 12 Nano-satellite pointing state at t = 2100 sec

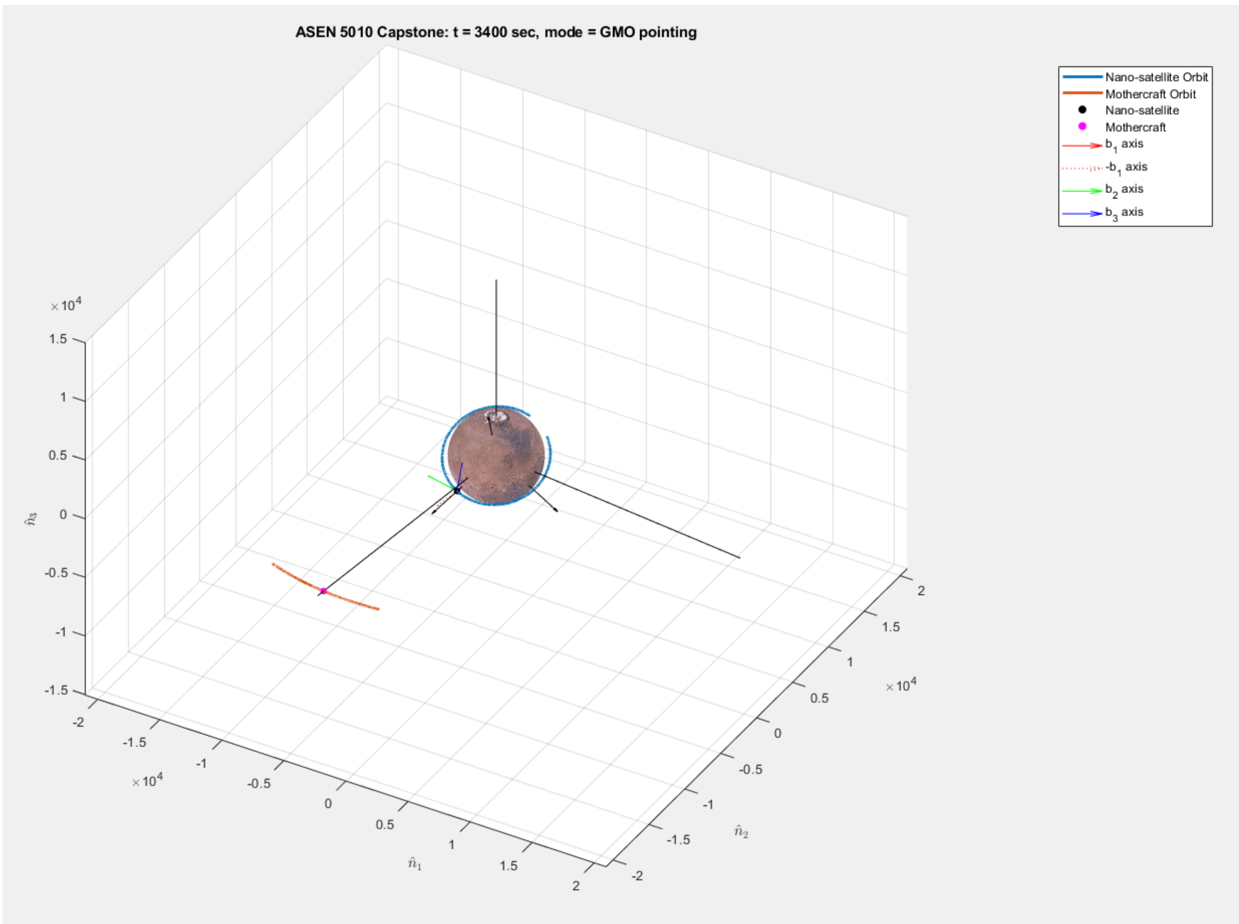

Fig. 13 Nano-satellite pointing state at t = 3400 sec

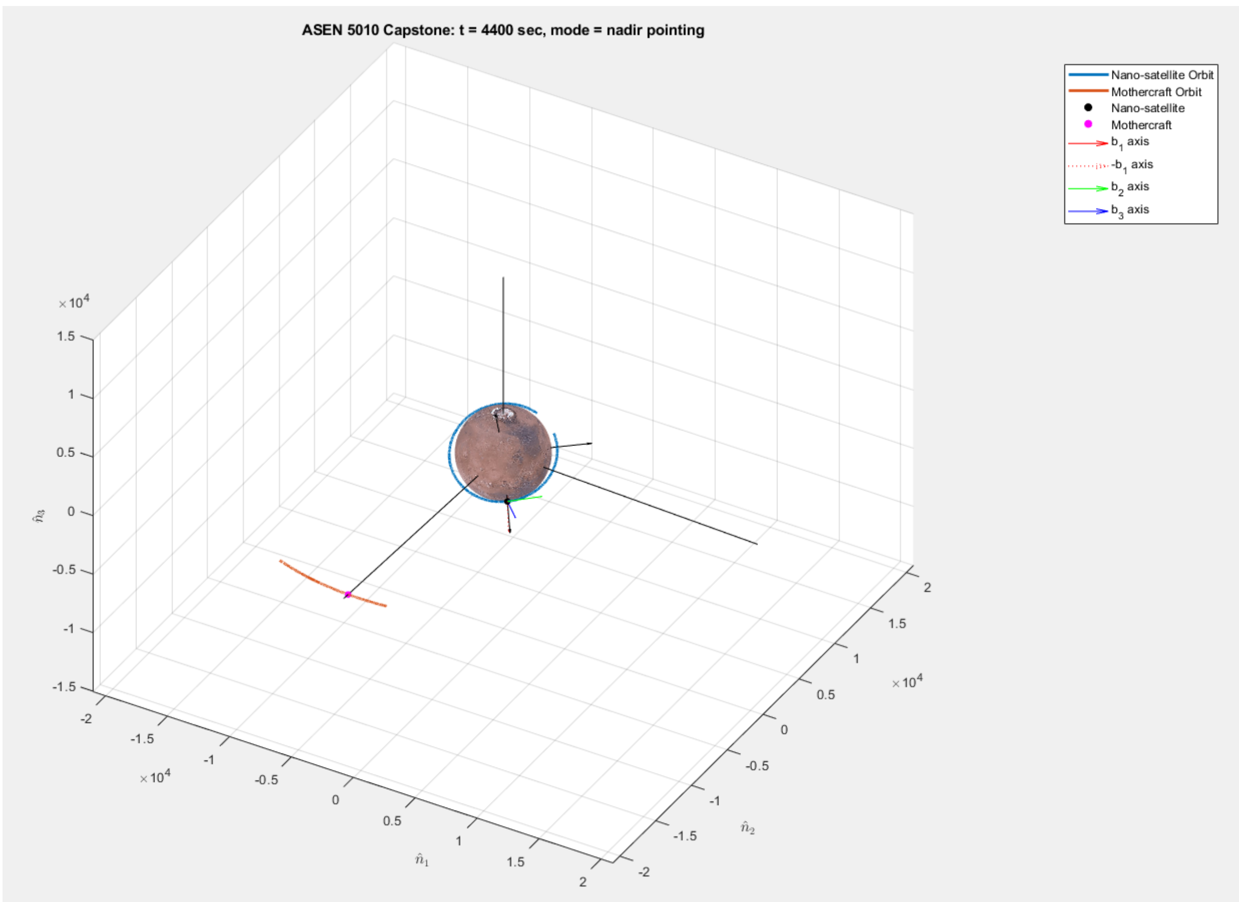

Fig. 14 Nano-satellite pointing state at t = 4400 sec

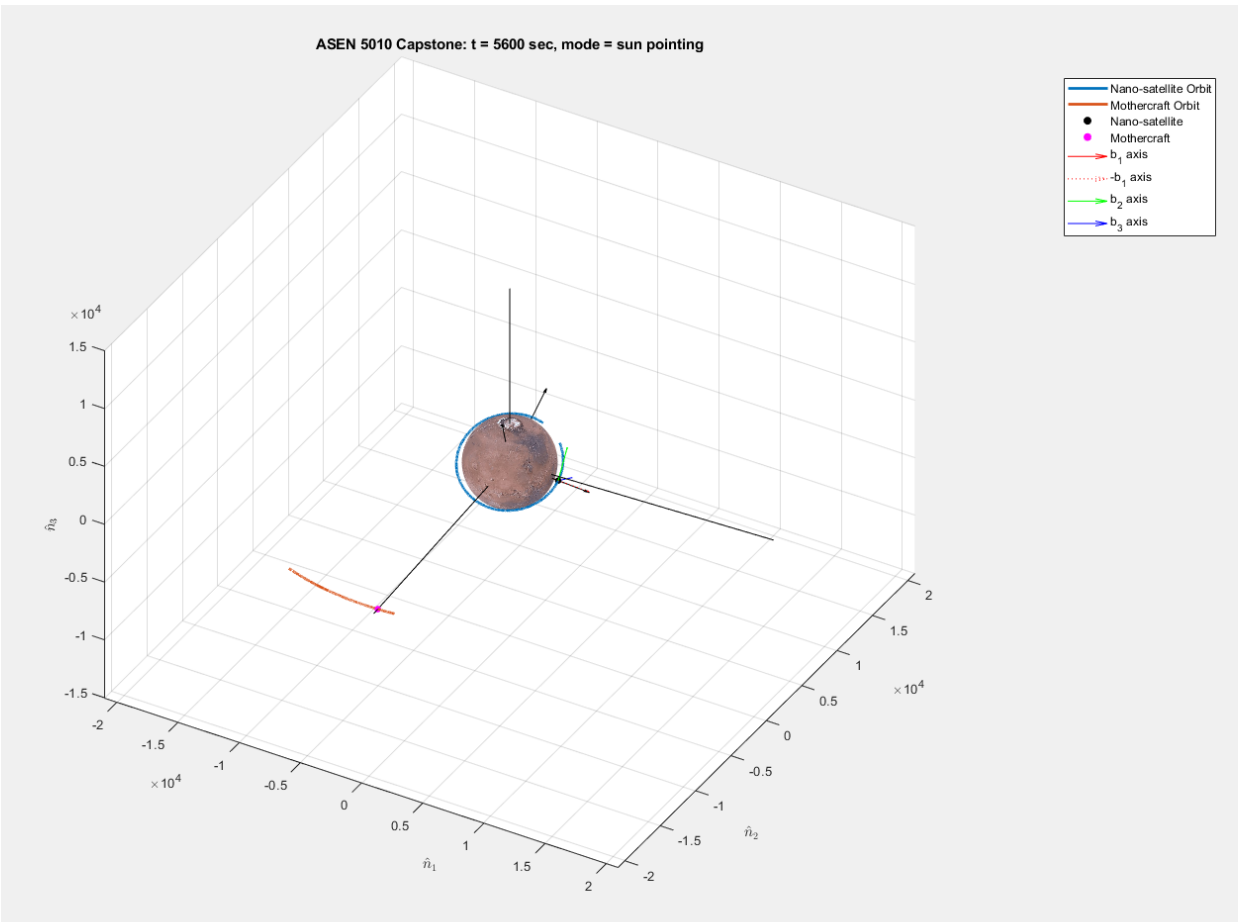

Fig. 15 Nano-satellite pointing state at t = 5600 sec

For visualization and debugging purposes, I made an animation of the pointing scenario relayed in [1], i.e. up to 6500 seconds. For fun, I also made a longer simulation that lasts for a full Mars day, or 88620 seconds! I can't embed a video in the report (for obvious reasons...) but the day-long animation can be found at the following link: `https://drive.google.com/file/d/1UgJ4yo-EHOU5XMEML6TSnC-hcen1oi1M/view?usp=sharing`

XIV. Conclusion

This project was incredibly fun to work on, and it reinforced my understanding of the class material immensely! I have a full appreciation for attitude dynamics and controls, and I feel like I understand the fundamentals of what a pointing mission like this needs from a theoretical standpoint. I might come back in my free time to try some more advanced control laws, just to see how things compare!

References

[1] Shaub, H., "Attitude Dynamics and Control of a Nano-Satellite Orbiting Mars," *ASEN 5010*, 2024.

[2] Shaub, H., and Junkins, J. L., *Analytical Mechanics of Space Systems*, 4^{th} ed., AIAA Education Series, 2018. https://doi.org/10.2514/4.105210.

A. Appendix: Project code, organized by task

A. Code Index

B. Code for Task 1

Back to index: A.A

```
%% ASEN 5010 Task 1 main script
% By: Ian Faber

%% Housekeeping
clc; clear; close all;

%% Setup
addpath('..\..\Utilities\')

R_Mars = 3396.19; % km

[marsX, marsY, marsZ] = sphere(100);
marsX = R_Mars*marsX;
marsY = R_Mars*marsY;
marsZ = R_Mars*marsZ;

w_LMO = [0; 0; 0.000884797]; % rad/s, O frame coords
EA_LMO = deg2rad([20; 30; 60]); % Omega, i, theta
h_LMO = 400; % km
radius_LMO = R_Mars + h_LMO; % km
x0_LMO = [radius_LMO; EA_LMO; w_LMO];

w_GMO = [0; 0; 0.0000709003]; % rad/s, O frame coords
EA_GMO = deg2rad([0; 0.0000000; 250]); % Omega, i, theta
h_GMO = 17028.01; % km
radius_GMO = R_Mars + h_GMO; % km
x0_GMO = [radius_GMO; EA_GMO; w_GMO];

t0 = 0; % sec
dt = 1; % sec
tf = 6500; % sec

%% Propagate orbits
out_LMO = RK4_Orbit(x0_LMO, t0, dt, tf);
out_GMO = RK4_Orbit(x0_GMO, t0, dt, tf);
```

```matlab

%% Extract answers to text files
t_LMO = 450;
t_GMO = 1150;


LMO_ans = out_LMO(out_LMO(:,1) == t_LMO,:);
r_LMO = LMO_ans(2:4);
v_LMO = LMO_ans(5:7);

f1 = fopen("LMO_1.txt", "w");
ans_LMO_1 = fprintf(f1, "%.1f %.1f %.1f", r_LMO(1), r_LMO(2), r_LMO(3));
fclose(f1);

f2 = fopen("LMO_2.txt", "w");
ans_LMO_2 = fprintf(f2, "%.3f %.3f %.3f", v_LMO(1), v_LMO(2), v_LMO(3));
fclose(f2);


GMO_ans = out_GMO(out_GMO(:,1) == t_GMO,:);
r_GMO = GMO_ans(2:4);
v_GMO = GMO_ans(5:7);

f3 = fopen("GMO_1.txt", "w");
ans_GMO_1 = fprintf(f3, "%.1f %.1f %.1f %.1f", r_GMO(1), r_GMO(2), r_GMO(3));
fclose(f3);

f4 = fopen("GMO_2.txt", "w");
ans_GMO_2 = fprintf(f4, "%.3f %.3f %.3f", v_GMO(1), v_GMO(2), v_GMO(3));
fclose(f4);


%% Plot for error checking
fig = figure;

% ax2 = axes();
% I2 = imread("marsStars.jpg");
% imshow(I2, 'parent', ax2)

ax1 = axes();
title("Orbit Simulation", 'Color', 'w')
hold on
grid on
axis equal
plot3(out_LMO(:,2), out_LMO(:,3), out_LMO(:,4), 'LineWidth', 3)
plot3(out_GMO(:,2), out_GMO(:,3), out_GMO(:,4), 'LineWidth', 3)

mars = surf(marsX, marsY, -marsZ); % Need to flip the sphere for image to map properly
I = imread("marsSurface.jpg");
set(mars,'FaceColor','texturemap','cdata',I,'edgecolor','none');

% set(gca, 'Color', 'none', 'XColor', 'w', 'YColor', 'w', 'ZColor', 'w', 'GridColor', 'w')
```

```

view([30 35])
```

```
function [dX, r, rDot] = calculateOrbit(X, radius)
% Calculates the inertial position and velocity vectors expressed in
% inertial components for a circular orbit
%
%   Inputs:
%       - X: State vector at a given point in time
%               [Omega; inclination; theta; w_1; w_2; w_3]
%       - radius: Radius of circular orbit
%   Outputs:
%       - dX: Rate of change vector based on the current state
%               [EADot; wDot]
%       - r: Current position vector of spacecraft
%       - rDot: Current velocity vector of spacecraft
%

EA = X(1:3);
w = X(4:6);

rVec = radius*[1; 0; 0];
rDotVec = radius*w(3)*[0; 1; 0];

ON = EA2DCM(EA, [3,1,3]);
NO = ON';

r = NO*rVec;
rDot = NO*rDotVec;

inc = EA(2);
theta = EA(3);

if inc ~= 0
    EADot = (1/sin(inc))*[
                            sin(theta),             cos(theta),             0;
                            cos(theta)*sin(inc),    -sin(theta)*sin(inc),   0;
                            -sin(theta)*cos(inc),   -cos(theta)*cos(inc),
                               sin(inc);
                         ]*w;
else
    EADot = [0; 0; w(3)];
end
wDot = zeros(3,1);

dX = [EADot; wDot];


end
```

```
function out = RK4_Orbit(x0, t0, dt, tf)
% Function that implements the Runga-Kutta 4 algorithm to integrate
% circular orbital motion based on a set of initial conditions
%   Inputs:
```

```
%       - x0: Initial state vector, with w written in orbit coordinates
%               [radius; EA_0; w_0]
%       - t0: Time that integration will start, in seconds
%       - dt: Time step for integration, in seconds
%       - tf: Time that integration will stop, in seconds
%
%   Outputs:
%       - out: Integration output matrix, each column is a vector with the
%              same number of elements n as there were timesteps
%               [t (nx1), r (nx3), rDot (nx3), EA (nx3), w (nx3)]
%
    radius = x0(1);
    EA_0 = x0(2:4);
    w_0 = x0(5:7);

    X = [EA_0; w_0];
    t = t0;

    [~, r, rDot] = calculateOrbit(X, radius);

    out = zeros(length(t0:dt:tf)-1, 13);
    out(1,:) = [t0, r', rDot', X']; % t, r(1:3), rDot(1:3), EA(1:3), w(1:3)
    k = 1;

    while t < tf
        k1 = dt*calculateOrbit(X,radius);
        k2 = dt*calculateOrbit(X+k1/2,radius);
        k3 = dt*calculateOrbit(X+k2/2,radius);
        k4 = dt*calculateOrbit(X+k3,radius);

        X = X + (1/6)*(k1 + 2*k2 + 2*k3 + k4);

        t = t + dt;
        k = k + 1;

        [~, r, rDot] = calculateOrbit(X, radius);

        out(k, :) = [t, r', rDot', X'];

    end

end
```

C. Code for Task 2

Back to index: A.A

```
%% ASEN 5010 Project Task 2 Main Script
% By: Ian Faber

%% Housekeeping
clc; clear; close all;

%% Setup
addpath('..\..\Utilities\')
addpath('..\Task1\')

R_Mars = 3396.19; % km
h = 400; % km
radius = R_Mars + h; % km

w_0 = [0; 0; 0.000884797]; % rad/s, O frame coords
EA_0 = deg2rad([20; 30; 60]); % Omega, i, theta

x_0 = [radius; EA_0; w_0];

t0 = 0;
dt = 0.5;
tf = 6500;

%% Propagate orbit and find HN
out = RK4_Orbit(x_0, t0, dt, tf);

t = 300;
HN = calcHN(t, out)

%% Save answer to text file
f1 = fopen("HN_ans.txt",'w');
ans_HN = fprintf(f1, "%.3f %.3f %.3f %.3f %.3f %.3f %.3f %.3f %.3f", HN(1,1),
    HN(1,2), HN(1,3), HN(2,1), HN(2,2), HN(2,3), HN(3,1), HN(3,2), HN(3,3));
fclose(f1);
```

```
function HN = calcHN(t, RK4_out)
% Calculates the Hill Frame DCM [HN] at a given point in time along an
% orbit
%   Inputs:
%       - t: time to calculate HN at, in seconds
%       - RK4_out: Output of RK4 integration for the orbit
%                  [t (nx1), r (nx3), rDot (nx3), EA (nx3), w (nx3)]
%   Outputs:
%       - HN: Hill Frame DCM
%

data = RK4_out(RK4_out(:,1) == t, :);

r = data(2:4)';
rDot = data(5:7)';

```

```
i_r = r/norm(r);
i_h = (cross(r, rDot)/norm(cross(r,rDot)));
i_theta = cross(i_h, i_r);

HN = [i_r'; i_theta'; i_h'];

end
```

D. Code for Task 3

Back to index: A.A

```
%% ASEN 5010 Project Task 3 Main Script
% By: Ian Faber

%% Housekeeping
clc; clear; close all;

%% Get [R_sN] and save answer

RsN = calcRsN();

f1 = fopen("RsN_ans.txt", 'w');
ans_RsN = fprintf(f1, "%.3f %.3f %.3f %.3f %.3f %.3f %.3f %.3f %.3f", RsN(1,1)
    , RsN(1,2), RsN(1,3), RsN(2,1), RsN(2,2), RsN(2,3), RsN(3,1), RsN(3,2),
    RsN(3,3));
fclose(f1);

%% Save answer for omega_{Rs/N}
f2 = fopen("w_ans.txt", 'w');
ans_w = fprintf(f2, "%.7f %.7f %.7f", 0, 0, 0);
fclose(f2)
```

```
function RsN = calcRsN()
% Returns the sun-pointing reference frame DCM [R_sN]
%   Inputs:
%       - None
%   Outputs:
%       - RsN: Sun-pointing reference frame DCM

RsN = [-1 0 0; 0 0 1; 0 1 0];

end
```

E. Code for Task 4

Back to index: A.A

```
%% ASEN 5010 Task 4 Main Script
% By: Ian Faber

%% Housekeeping
clc; clear; close all;

%% Setup
addpath('..\..\Utilities')
addpath('..\Task1')
addpath('..\Task2')

R_Mars = 3396.19; % km
h = 400; % km
radius = R_Mars + h; % km

w_0 = [0; 0; 0.000884797]; % rad/s, O frame coords
EA_0 = deg2rad([20; 30; 60]); % Omega, i, theta

x_0 = [radius; EA_0; w_0];

t0 = 0;
dt = 0.5;
tf = 6500;

%% Propagate orbit and find R_nN
out = RK4_Orbit(x_0, t0, dt, tf);

t = 330;
RnN = calcRnN(t, out)
w_RnN = calcW_RnN(t, out)

%% Save answer to text file
f1 = fopen("RnN_ans.txt",'w');
ans_RnN = fprintf(f1, "%.3f %.3f %.3f %.3f %.3f %.3f %.3f %.3f %.3f", RnN(1,1)
    , RnN(1,2), RnN(1,3), RnN(2,1), RnN(2,2), RnN(2,3), RnN(3,1), RnN(3,2),
    RnN(3,3));
fclose(f1);

f2 = fopen("w_RnN_ans.txt",'w');
ans_w_RnN = fprintf(f2, "%.8f %.8f %.8f", w_RnN(1), w_RnN(2), w_RnN(3));
fclose(f2);
```

```
function RnN = calcRnN(t, RK4_out)
% Calculates the Nadir-pointing frame DCM [RnN] at a given point in time
%   Inputs:
%       - t: Time to evaluate RnN at, in seconds
%       - RK4_out: Output of RK4 integration for the orbit
%                  [t (nx1), r (nx3), rDot (nx3), EA (nx3), w (nx3)]
%   Outputs:
%       - RnN: Nadir-pointing frame DCM [RnN]
%
```

```

HN = calcHN(t, RK4_out);
NH = HN';

RnN = [
        (-NH*[1; 0; 0])'; % -i_r
        (NH*[0; 1; 0])'; % i_theta
        (-NH*[0; 0; 1])' % -i_h
      ];

end
```

```
function w_RnN = calcW_RnN(t, RK4_out)
% Calculates the angular velocity vector between the nadir-pointing frame
% and inertial frame at a given point in time
%   Inputs:
%       - t: Time to evaluate the vector at
%       - RK4_out: Output of RK4 integration for the orbit
%                  [t (nx1), r (nx3), rDot (nx3), EA (nx3), w (nx3)]
%   Outputs:
%       - w_RnN: Anglar velocity vector between Rn and N in inertial
%                coordinates
%

HN = calcHN(t, RK4_out);
NH = HN';

w = RK4_out((RK4_out(:,1) == t), 11:13)';

w_RnN = NH*w;

end
```

F. Code for Task 5

Back to index: A.A

```
%% ASEN 5010 Task 5 Main Script
% By: Ian Faber

%% Housekeeping
clc; clear; close all;

%% Setup
addpath('..\..\Utilities')
addpath('..\Task1')
addpath('..\Task2')
addpath('..\Task4')

R_Mars = 3396.19; % km
h_LMO = 400; % km
h_GMO = 17028.01; % km
radius_LMO = R_Mars + h_LMO; % km
radius_GMO = R_Mars + h_GMO; % km

w_0_LMO = [0; 0; 0.000884797]; % rad/s, O frame coords
EA_0_LMO = deg2rad([20; 30; 60]); % Omega, i, theta

w_0_GMO = [0; 0; 0.0000709003]; % rad/s, O frame coords
EA_0_GMO = deg2rad([0; 0; 250]); % Omega, i, theta

x_0_LMO = [radius_LMO; EA_0_LMO; w_0_LMO];
x_0_GMO = [radius_GMO; EA_0_GMO; w_0_GMO];

t0 = 0;
dt = 0.5;
tf = 6500;

%% Propagate orbits and find R_cN
out_LMO = RK4_Orbit(x_0_LMO, t0, dt, tf);
out_GMO = RK4_Orbit(x_0_GMO, t0, dt, tf);

t = 330;
RcN = calcRcN(t, out_GMO, out_LMO)
w_RcN = calcW_RcN(t, dt, out_GMO, out_LMO)

%% Save answer as text file
f1 = fopen("RcN_ans.txt",'w');
ans_RcN = fprintf(f1, "%.3f %.3f %.3f %.3f %.3f %.3f %.3f %.3f %.3f", RcN(1,1), RcN(1,2), RcN(1,3), RcN(2,1), RcN(2,2), RcN(2,3), RcN(3,1), RcN(3,2), RcN(3,3));
fclose(f1);

f2 = fopen("w_RcN_ans.txt",'w');
ans_w_RnN = fprintf(f2, "%.8f %.8f %.8f", w_RcN(1), w_RcN(2), w_RcN(3));
fclose(f2);
```

```
function RcN = calcRcN(t, RK4_out_GMO, RK4_out_LMO)
```

```
% Calculates the GMO-pointing frame DCM [RcN] at a given point in time
%   Inputs:
%       - t: Time to evaluate [RcN] at, in seconds
%       - RK4_out_GMO: Output of RK4 integration for the mothercraft orbit
%                       [t (nx1), r_GMO (nx3), rDot_GMO (nx3),
%                        EA_GMO (nx3), w_GMO (nx3)]
%       - RK4_out_LMO: Output of RK4 integration for the nano-satellite
%                      orbit
%                       [t (nx1), r_LMO (nx3), rDot_LMO (nx3),
%                        EA_LMO (nx3), w_LMO (nx3)]
%   Outputs:
%       - RcN: GMO-pointing frame DCM [RcN]
%

r_GMO = RK4_out_GMO(RK4_out_GMO(:,1) == t, 2:4)';
r_LMO = RK4_out_LMO(RK4_out_LMO(:,1) == t, 2:4)';

delR = r_GMO - r_LMO;

r1 = -delR/norm(-delR);
r2 = cross(delR, [0; 0; 1])/norm(cross(delR, [0; 0; 1]));
r3 = cross(r1, r2);

RcN = [r1'; r2'; r3'];


end
```

```
function w_RcN = calcW_RcN(t, dt, RK4_out_GMO, RK4_out_LMO)
% Calculates the angular velocity vector between the GMO-pointing frame
% and inertial frame at a given point in time
%   Inputs:
%       - t: Time to evaluate the vector at
%       - dt: Discrete time for numerical derivative
%       - RK4_out_GMO: Output of RK4 integration for the mothercraft orbit
%                       [t (nx1), r_GMO (nx3), rDot_GMO (nx3),
%                        EA_GMO (nx3), w_GMO (nx3)]
%       - RK4_out_LMO: Output of RK4 integration for the nano-satellite
%                      orbit
%                       [t (nx1), r_LMO (nx3), rDot_LMO (nx3),
%                        EA_LMO (nx3), w_LMO (nx3)]
%   Outputs:
%       - w_RcN: Anglar velocity vector between Rc and N in inertial
%                coordinates
%

RcN_t0 = calcRcN(t, RK4_out_GMO, RK4_out_LMO);
RcN_t1 = calcRcN(t+dt, RK4_out_GMO, RK4_out_LMO);

f = {t, RcN_t0; t+dt, RcN_t1};
dfdt = finiteDifMat(t, dt, f);

wTildeMat = -dfdt*RcN_t0'; % This is in Rc coords!

```

```
w_RcN = RcN_t0'*[-wTildeMat(2,3); wTildeMat(1,3); -wTildeMat(1,2)]; % convert
    to inertial

end
```

G. Code for Task 6

Back to index: A.A

```
%% ASEN 5010 Task 6 Main Script
% By: Ian Faber

%% Housekeeping
clc; clear; close all;

%% Setup
addpath('..\..\Utilities')
addpath('..\Task1')
addpath('..\Task2')
addpath('..\Task3')
addpath('..\Task4')
addpath('..\Task5')

R_Mars = 3396.19; % km
h_LMO = 400; % km
h_GMO = 17028.01; % km
radius_LMO = R_Mars + h_LMO; % km
radius_GMO = R_Mars + h_GMO; % km

w_0_LMO = [0; 0; 0.000884797]; % rad/s, O frame coords
EA_0_LMO = deg2rad([20; 30; 60]); % Omega, i, theta

w_0_GMO = [0; 0; 0.0000709003]; % rad/s, O frame coords
EA_0_GMO = deg2rad([0; 0; 250]); % Omega, i, theta

x_0_LMO = [radius_LMO; EA_0_LMO; w_0_LMO];
x_0_GMO = [radius_GMO; EA_0_GMO; w_0_GMO];

sigBN_0 = [0.3; -0.4; 0.5];
omegBN_0 = deg2rad([1.00; 1.75; -2.20]);

t0 = 0;
dt = 0.5;
tf = 6500;

%% Propagate orbits and find R_cN
out_LMO = RK4_Orbit(x_0_LMO, t0, dt, tf);
out_GMO = RK4_Orbit(x_0_GMO, t0, dt, tf);

t = t0;

RsN = calcRsN();
w_RsN = [0; 0; 0];

RnN = calcRnN(t, out_LMO);
w_RnN = calcW_RnN(t, out_LMO);

RcN = calcRcN(t, out_GMO, out_LMO);
w_RcN = calcW_RcN(t, dt, out_GMO, out_LMO);

```

```
[sigBR_sun, omegBR_sun] = calcError(sigBN_0, omegBN_0, RsN, w_RsN) % Sun-pointing error

[sigBR_nadir, omegBR_nadir] = calcError(sigBN_0, omegBN_0, RnN, w_RnN) % Sun-pointing error

[sigBR_GMO, omegBR_GMO] = calcError(sigBN_0, omegBN_0, RcN, w_RcN) % Sun-pointing error

%% Formulate text files for submission
f1 = fopen("sigBR_sun_ans.txt",'w');
ans_sigBR_sun = fprintf(f1, "%.8f %.8f %.8f", sigBR_sun(1), sigBR_sun(2), sigBR_sun(3));
fclose(f1);

f2 = fopen("omegBR_sun_ans.txt",'w');
ans_omegBR_sun = fprintf(f2, "%.8f %.8f %.8f", omegBR_sun(1), omegBR_sun(2), omegBR_sun(3));
fclose(f2);


f3 = fopen("sigBR_nadir_ans.txt",'w');
ans_sigBR_nadir = fprintf(f3, "%.8f %.8f %.8f", sigBR_nadir(1), sigBR_nadir(2), sigBR_nadir(3));
fclose(f3);

f4 = fopen("omegBR_nadir_ans.txt",'w');
ans_omegBR_nadir = fprintf(f4, "%.8f %.8f %.8f", omegBR_nadir(1), omegBR_nadir(2), omegBR_nadir(3));
fclose(f4);


f5 = fopen("sigBR_GMO_ans.txt",'w');
ans_sigBR_GMO = fprintf(f5, "%.8f %.8f %.8f", sigBR_GMO(1), sigBR_GMO(2), sigBR_GMO(3));
fclose(f5);

f6 = fopen("omegBR_GMO_ans.txt",'w');
ans_omegBR_GMO = fprintf(f6, "%.8f %.8f %.8f", omegBR_GMO(1), omegBR_GMO(2), omegBR_GMO(3));
fclose(f6);
```

```
function [sigBR, omegBR] = calcError(sigBN, omegBN, RN, omegRN)
% Calculates the attitude and angular velocity tracking errors between two
% frames B and R.
%
%   Inputs:
%       - t: Time to compute error at
%       - sigBN: Current MRP set describing the nano-satellite's
%                orientation at time t
%       - omegBN: Current angular velocity vector of the nano-satellite at
%                 time t, in body coordinates
%       - RN: Current reference frame orientation DCM
%       - omegRN: Current angular velocity vector of the reference frame at
```

```
%                  time t, in inertial coordinates
%   Outputs:
%       - sigBR: MRP set describing the attitude tracking error of B
%                 relative to R
%       - omegBR: angular velocity vector describing the angular velocity
%                  tracking error of B relative to R, in B coordinates

BN = MRP2DCM(sigBN);
sigBR = DCM2MRP(BN*RN', 1);

omegBR = BN*(BN'*omegBN - omegRN);


end
```

H. Code for Task 7

Back to index: A.A

```
%% ASEN 5010 Task 7 Main Script
% By: Ian Faber

%% Housekeeping
clc; clear; close all

%% Setup
% Include all of our lovely task functions
addpath('..\..\Utilities')
addpath('..\Task1')
addpath('..\Task2')
addpath('..\Task3')
addpath('..\Task4')
addpath('..\Task5')
addpath('..\Task6')

% Make Mars
R_Mars = 3396.19; % km
[marsX, marsY, marsZ] = sphere(100);
marsX = R_Mars*marsX;
marsY = R_Mars*marsY;
marsZ = R_Mars*marsZ;

% Initial orbit parameters
h_LMO = 400; % km
h_GMO = 17028.01; % km
radius_LMO = R_Mars + h_LMO; % km
radius_GMO = R_Mars + h_GMO; % km

w_0_LMO = [0; 0; 0.000884797]; % rad/s, O frame coords
EA_0_LMO = deg2rad([20; 30; 60]); % Omega, i, theta

w_0_GMO = [0; 0; 0.0000709003]; % rad/s, O frame coords
EA_0_GMO = deg2rad([0; 0; 250]); % Omega, i, theta

x_0_LMO = [radius_LMO; EA_0_LMO; w_0_LMO];
x_0_GMO = [radius_GMO; EA_0_GMO; w_0_GMO];

% Initial attitude parameters
sigBN_0 = [0.3; -0.4; 0.5]; % unitless
omegBN_0 = deg2rad([1.00; 1.75; -2.20]); % Converted to rad/s from deg/s
u_0 = zeros(3,1); % Nm
I = diag([10, 5, 7.5]); % kgm^2, body coords

x_0_att = {I; sigBN_0; omegBN_0; u_0};

% RK4 params
t0 = 0;
dt = 1;
tf = 1000;

```

```matlab
%% Propagate orbits and attitude with 0 control
out_LMO = RK4_Orbit(x_0_LMO, t0, dt, tf);
out_GMO = RK4_Orbit(x_0_GMO, t0, dt, tf);

out_Attitude = RK4_Attitude(x_0_att, t0, dt, tf, 'no torque');

%% Plot simulation outputs
t = out_Attitude(:,1);
sig = out_Attitude(:,2:4);
w = out_Attitude(:,5:7);
u = out_Attitude(:,8:10);

% Orbit
figOrbit = figure;

ax2 = axes();
marsStars = imread("marsStars.jpg");
imshow(marsStars, 'parent', ax2)

ax1 = axes();
title("Orbit Simulation", 'Color', 'w')
hold on
grid on
axis equal
plot3(out_LMO(:,2), out_LMO(:,3), out_LMO(:,4), 'LineWidth', 3)
plot3(out_GMO(:,2), out_GMO(:,3), out_GMO(:,4), 'LineWidth', 3)

mars = surf(marsX, marsY, -marsZ); % Need to flip the sphere for image to map properly
marsSurface = imread("marsSurface.jpg");
set(mars,'FaceColor','texturemap','cdata',marsSurface,'edgecolor','none');

set(gca, 'Color', 'none', 'XColor', 'w', 'YColor', 'w', 'ZColor', 'w', 'GridColor', 'w')

xlabel("$\hat{n}_1$", "Interpreter","latex")
ylabel("$\hat{n}_2$", "Interpreter","latex")
zlabel("$\hat{n}_3$", "Interpreter","latex")

view([30 35])

% Attitude
figAttitude = figure;

sgtitle("Nano-satellite Attitude Evolution Over Time")

subplot(3,3,1)
hold on
grid on
title("\sigma_1 vs. time")
plot(t, sig(:,1))
xlabel("Time [sec]")
ylabel("\sigma_1")

```

```
subplot(3,3,2)
hold on
grid on
title("\omega_1 vs. time")
plot(t, w(:,1))
xlabel("Time [sec]")
ylabel("\omega_1")

subplot(3,3,3)
hold on
grid on
title("u_1 vs. time")
plot(t, u(:,1))
xlabel("Time [sec]")
ylabel("u_1")

subplot(3,3,4)
hold on
grid on
title("\sigma_2 vs. time")
plot(t, sig(:,2))
xlabel("Time [sec]")
ylabel("\sigma_2")

subplot(3,3,5)
hold on
grid on
title("\omega_2 vs. time")
plot(t, w(:,1))
xlabel("Time [sec]")
ylabel("\omega_2")

subplot(3,3,6)
hold on
grid on
title("u_2 vs. time")
plot(t, u(:,2))
xlabel("Time [sec]")
ylabel("u_2")

subplot(3,3,7)
hold on
grid on
title("\sigma_3 vs. time")
plot(t, sig(:,3))
xlabel("Time [sec]")
ylabel("\sigma_3")

subplot(3,3,8)
hold on
grid on
title("\omega_3 vs. time")
plot(t, w(:,3))
xlabel("Time [sec]")
```

```
ylabel("\omega_3")

subplot(3,3,9)
hold on
grid on
title("u_3 vs. time")
plot(t, u(:,3))
xlabel("Time [sec]")
ylabel("u_3")

%% Extract answers to text files
t = 500;

sig_ans = sig(out_Attitude(:,1) == t, :)';
w_ans = w(out_Attitude(:,1) == t, :)';

H_ans = I*w_ans; % In body coords
T_ans = 0.5*w_ans'*I*w_ans;

BN = MRP2DCM(sig_ans);
NB = BN';

H_ans_2 = NB*H_ans; % In inertial coords

f1 = fopen("H_ans.txt", "w");
ans_H = fprintf(f1, "%.3f %.3f %.3f", H_ans(1), H_ans(2), H_ans(3));
fclose(f1);

f2 = fopen("T_ans.txt", "w");
ans_T = fprintf(f2, "%.5f", T_ans);
fclose(f2);

f3 = fopen("sig_ans.txt", "w");
ans_sig = fprintf(f3, "%.3f %.3f %.3f", sig_ans(1), sig_ans(2), sig_ans(3));
fclose(f3);

f4 = fopen("H_ans_2.txt", "w");
ans_H_2 = fprintf(f4, "%.3f %.3f %.3f", H_ans_2(1), H_ans_2(2), H_ans_2(3));
fclose(f4);

%% Propagate orbits and attitude with fixed [0.01; -0.01; 0.02] Nm control
out_LMO = RK4_Orbit(x_0_LMO, t0, dt, tf);
out_GMO = RK4_Orbit(x_0_GMO, t0, dt, tf);

u_0 = [0.01; -0.01; 0.02]; % Nm, body coords
x_0_att = {I; sigBN_0; omegBN_0; u_0};

out_Attitude = RK4_Attitude(x_0_att, t0, dt, tf, 'torque');

%% Plot simulation outputs
t = out_Attitude(:,1);
sig = out_Attitude(:,2:4);
w = out_Attitude(:,5:7);
u = out_Attitude(:,8:10);
```

```

% Orbit
figOrbit = figure;

ax2 = axes();
marsStars = imread("marsStars.jpg");
imshow(marsStars, 'parent', ax2)

ax1 = axes();
title("Orbit Simulation", 'Color', 'w')
hold on
grid on
axis equal
plot3(out_LMO(:,2), out_LMO(:,3), out_LMO(:,4), 'LineWidth', 3)
plot3(out_GMO(:,2), out_GMO(:,3), out_GMO(:,4), 'LineWidth', 3)

mars = surf(marsX, marsY, -marsZ); % Need to flip the sphere for image to map properly
marsSurface = imread("marsSurface.jpg");
set(mars,'FaceColor','texturemap','cdata',marsSurface,'edgecolor','none');

set(gca, 'Color', 'none', 'XColor', 'w', 'YColor', 'w', 'ZColor', 'w', 'GridColor', 'w')

view([30 35])

% Attitude
figAttitude = figure;

sgtitle("Nano-satellite Attitude Evolution Over Time")

subplot(3,3,1)
hold on
title("\sigma_1 vs. time")
plot(t, sig(:,1))
xlabel("Time [sec]")
ylabel("\sigma_1")

subplot(3,3,2)
hold on
title("\omega_1 vs. time")
plot(t, w(:,1))
xlabel("Time [sec]")
ylabel("\omega_1")

subplot(3,3,3)
hold on
title("u_1 vs. time")
plot(t, u(:,1))
xlabel("Time [sec]")
ylabel("u_1")

subplot(3,3,4)
hold on
```

```
title("\sigma_2 vs. time")
plot(t, sig(:,2))
xlabel("Time [sec]")
ylabel("\sigma_2")

subplot(3,3,5)
hold on
title("\omega_2 vs. time")
plot(t, w(:,1))
xlabel("Time [sec]")
ylabel("\omega_2")

subplot(3,3,6)
hold on
title("u_2 vs. time")
plot(t, u(:,2))
xlabel("Time [sec]")
ylabel("u_2")

subplot(3,3,7)
hold on
title("\sigma_3 vs. time")
plot(t, sig(:,3))
xlabel("Time [sec]")
ylabel("\sigma_3")

subplot(3,3,8)
hold on
title("\omega_3 vs. time")
plot(t, w(:,3))
xlabel("Time [sec]")
ylabel("\omega_3")

subplot(3,3,9)
hold on
title("u_3 vs. time")
plot(t, u(:,3))
xlabel("Time [sec]")
ylabel("u_3")

%% Extract answers to text files
t = 100;

sig_ans_2 = sig(out_Attitude(:,1) == t, :)';

f5 = fopen("sig_ans_2.txt", "w");
ans_sig_2 = fprintf(f5, "%.3f %.3f %.3f", sig_ans_2(1), sig_ans_2(2),
    sig_ans_2(3));
fclose(f5);
```

```
function dX = calculateAttitude(X, I, u)
% Calculates the nano-satellite body attitude relative to inertial space
%
%   Inputs:
```

```
%       - X: State vector at a given point in time
%               [sig_1; sig_2; sig_3; w_1; w_2; w_3]
%       - I: Body fixed inertia matrix
%               [diag(I_11, I_22, I_33)]
%       - u: Control input vector
%               [ u_1; u_2; u_3]
%   Outputs:
%       - dX: Rate of change vector based on the current state
%               [sigDot; wDot]
%

sig = X(1:3);
w = X(4:6);

sigSqr = dot(sig,sig);
sigDot = 0.25*((1-sigSqr)*eye(3) + 2*tilde(sig) + 2*(sig*sig'))*w;

wDot = (I^-1)*(-tilde(w)*I*w + u);

dX = [sigDot; wDot];

end
```

```
function out = RK4_Attitude(x0, t0, dt, tf, sit)
% Function that implements the Runga-Kutta 4 algorithm to integrate
% the attitude of a rigid body subject to a set of initial conditions
%   Inputs:
%       - x0: Initial state vector, with w written in body coordinates
%               {I; sig_0; w_0; u_0}
%       - t0: Time that integration will start, in seconds
%       - dt: Time step for integration, in seconds
%       - tf: Time that integration will stop, in seconds
%       - sit: Situation to simulate ('no torque', 'torque')
%
%   Outputs:
%       - out: Integration output matrix, each column is a vector with the
%              same number of elements n as there were timesteps
%               [t (nx1), sig (nx3), w (nx3), u(nx3)]
%
    I = x0{1};
    sig_0 = x0{2};
    w_0 = x0{3};
    u_0 = x0{4};

    X = [sig_0; w_0];
    t = t0;

    out = zeros(length(t0:dt:tf)-1, 10);
    out(1,:) = [t0, X', u_0']; % t, sig(1:3), w(1:3), u(1:3)
    k = 1;

    while t < tf
        switch sit
            case 'no torque'
```

```
                u = zeros(3,1);
            case 'torque'
                u = u_0;
        end

        k1 = dt*calculateAttitude(X,I,u);
        k2 = dt*calculateAttitude(X+k1/2,I,u);
        k3 = dt*calculateAttitude(X+k2/2,I,u);
        k4 = dt*calculateAttitude(X+k3,I,u);

        X = X + (1/6)*(k1 + 2*k2 + 2*k3 + k4);

        sigNorm = norm(X(1:3));
        if sigNorm > 1
            X(1:3) = -X(1:3)/(sigNorm^2);
        end

        t = t + dt;
        k = k + 1;

        out(k, :) = [t, X', u'];

    end

end
```

I. Code for Task 8

Back to index: A.A

```
%% ASEN 5010 Task 8 Main Script
% By: Ian Faber

%% Housekeeping
clc; clear; close all

%% Setup
% Include all of our lovely task functions
addpath('..\..\Utilities')
addpath('..\Task1')
addpath('..\Task2')
addpath('..\Task3')
addpath('..\Task4')
addpath('..\Task5')
addpath('..\Task6')

% Make Mars
R_Mars = 3396.19; % km
[marsX, marsY, marsZ] = sphere(100);
marsX = R_Mars*marsX;
marsY = R_Mars*marsY;
marsZ = R_Mars*marsZ;

% Initial orbit parameters
h_LMO = 400; % km
h_GMO = 17028.01; % km
radius_LMO = R_Mars + h_LMO; % km
radius_GMO = R_Mars + h_GMO; % km

w_0_LMO = [0; 0; 0.000884797]; % rad/s, O frame coords
EA_0_LMO = deg2rad([20; 30; 60]); % Omega, i, theta

w_0_GMO = [0; 0; 0.0000709003]; % rad/s, O frame coords
EA_0_GMO = deg2rad([0; 0; 250]); % Omega, i, theta

x_0_LMO = [radius_LMO; EA_0_LMO; w_0_LMO];
x_0_GMO = [radius_GMO; EA_0_GMO; w_0_GMO];

% Initial attitude parameters
sigBN_0 = [0.3; -0.4; 0.5]; % unitless
omegBN_0 = deg2rad([1.00; 1.75; -2.20]); % Converted to rad/s from deg/s
u_0 = zeros(3,1); % Nm
I = diag([10, 5, 7.5]); % kgm^2, body coords

x_0_att = {I; sigBN_0; omegBN_0; u_0};

% Controller parameters
K = 1/180; % 0.0056, from zeta requirement
P = 1/6; % 0.1667, from time decay requirement
controlParams = [K; P];

```

```
% RK4 params
t0 = 0;
dt = 1;
tf = 1000;

%% Propagate orbits and attitude with sun-pointing control
out_LMO = RK4_Orbit(x_0_LMO, t0, dt, tf);
out_GMO = RK4_Orbit(x_0_GMO, t0, dt, tf);

out_Attitude = RK4_Attitude(x_0_att, t0, dt, tf, 'sun pointing', out_LMO,
    out_GMO, controlParams);

%% Plot simulation outputs
t = out_Attitude(:,1);
sig = out_Attitude(:,2:4);
w = out_Attitude(:,5:7);
u = out_Attitude(:,8:10);
sigRef = out_Attitude(:,11:13);
wRef = out_Attitude(:,14:16);

% Orbit
figOrbit = figure;

ax2 = axes();
marsStars = imread("marsStars.jpg");
imshow(marsStars, 'parent', ax2)

ax1 = axes();
title("Orbit Simulation", 'Color', 'w')
hold on
grid on
axis equal
plot3(out_LMO(:,2), out_LMO(:,3), out_LMO(:,4), 'LineWidth', 3)
plot3(out_GMO(:,2), out_GMO(:,3), out_GMO(:,4), 'LineWidth', 3)

mars = surf(marsX, marsY, -marsZ); % Need to flip the sphere for image to map
    properly
marsSurface = imread("marsSurface.jpg");
set(mars,'FaceColor','texturemap','cdata',marsSurface,'edgecolor','none');

set(gca, 'Color', 'none', 'XColor', 'w', 'YColor', 'w', 'ZColor', 'w', '
    GridColor', 'w')

xlabel("$\hat{n}_1$", "Interpreter","latex")
ylabel("$\hat{n}_2$", "Interpreter","latex")
zlabel("$\hat{n}_3$", "Interpreter","latex")

view([30 35])

% Attitude
figAttitude = figure;

sgtitle("Nano-satellite Attitude Evolution Over Time")

```

```
subplot(3,3,1)
hold on
grid on
title("\sigma_1 vs. time")
plot(t, sig(:,1))
plot(t, sigRef(:,1), '--')
xlabel("Time [sec]")
ylabel("\sigma_1")
legend("Current", "Reference", 'location', 'best')

subplot(3,3,2)
hold on
grid on
title("\omega_1 vs. time")
plot(t, w(:,1))
plot(t, wRef(:,1), '--')
xlabel("Time [sec]")
ylabel("\omega_1 [rad/s]")
legend("Current", "Reference", 'location', 'best')

subplot(3,3,3)
hold on
grid on
title("u_1 vs. time")
plot(t, u(:,1))
xlabel("Time [sec]")
ylabel("u_1 [Nm]")

subplot(3,3,4)
hold on
grid on
title("\sigma_2 vs. time")
plot(t, sig(:,2))
plot(t, sigRef(:,2), '--')
xlabel("Time [sec]")
ylabel("\sigma_2")
legend("Current", "Reference", 'location', 'best')

subplot(3,3,5)
hold on
grid on
title("\omega_2 vs. time")
plot(t, w(:,2))
plot(t, wRef(:,2), '--')
xlabel("Time [sec]")
ylabel("\omega_2 [rad/s]")
legend("Current", "Reference", 'location', 'best')

subplot(3,3,6)
hold on
grid on
title("u_2 vs. time")
plot(t, u(:,2))
xlabel("Time [sec]")
```

```
ylabel("u_2 [Nm]")

subplot(3,3,7)
hold on
grid on
title("\sigma_3 vs. time")
plot(t, sig(:,3))
plot(t, sigRef(:,3), '--')
xlabel("Time [sec]")
ylabel("\sigma_3")
legend("Current", "Reference", 'location', 'best')

subplot(3,3,8)
hold on
grid on
title("\omega_3 vs. time")
plot(t, w(:,3))
plot(t, wRef(:,3), '--')
xlabel("Time [sec]")
ylabel("\omega_3 [rad/s]")
legend("Current", "Reference", 'location', 'best')

subplot(3,3,9)
hold on
grid on
title("u_3 vs. time")
plot(t, u(:,3))
xlabel("Time [sec]")
ylabel("u_3 [Nm]")

%% Extract answers to text files
time = t;

t = 15;
sig15 = sig(time == t, :);

t = 100;
sig100 = sig(time == t, :);

t = 200;
sig200 = sig(time == t, :);

t = 400;
sig400 = sig(time == t, :);

f1 = fopen("gains.txt", "w");
ans_gains = fprintf(f1, "%.4f %.4f", P, K);
fclose(f1);

f2 = fopen("sig15_ans.txt", "w");
ans_sig15 = fprintf(f2, "%.3f %.3f %.3f", sig15(1), sig15(2), sig15(3));
fclose(f2);

f3 = fopen("sig100_ans.txt", "w");
```

```
ans_sig100 = fprintf(f3, "%.3f %.3f %.3f", sig100(1), sig100(2), sig100(3));
fclose(f3);

f4 = fopen("sig200_ans.txt", "w");
ans_sig200 = fprintf(f4, "%.3f %.3f %.3f", sig200(1), sig200(2), sig200(3));
fclose(f4);

f5 = fopen("sig400_ans.txt", "w");
ans_sig400 = fprintf(f5, "%.3f %.3f %.3f", sig400(1), sig400(2), sig400(3));
fclose(f5);
```

```
function dX = calculateAttitude(X, I, u)
% Calculates the nano-satellite body attitude relative to inertial space
%
%   Inputs:
%       - X: State vector at a given point in time
%               [sig_1; sig_2; sig_3; w_1; w_2; w_3]
%       - I: Body fixed inertia matrix
%               [diag(I_11, I_22, I_33)]
%       - u: Control input vector
%               [ u_1; u_2; u_3]
%   Outputs:
%       - dX: Rate of change vector based on the current state
%               [sigDot; wDot]
%

sig = X(1:3);
w = X(4:6);

sigSqr = dot(sig,sig);
sigDot = 0.25*((1-sigSqr)*eye(3) + 2*tilde(sig) + 2*(sig*sig'))*w;

wDot = (I^-1)*(-tilde(w)*I*w + u);

dX = [sigDot; wDot];

end
```

```
function out = RK4_Attitude(x0, t0, dt, tf, sit, orbit_LMO, orbit_GMO,
    controlParams)
% Function that implements the Runga-Kutta 4 algorithm to integrate
% the attitude of a rigid body subject to a set of initial conditions
%   Inputs:
%       - x0: Initial state vector, with w written in body coordinates
%               {I; sig_0; w_0; u_0}
%       - t0: Time that integration will start, in seconds
%       - dt: Time step for integration, in seconds
%       - tf: Time that integration will stop, in seconds
%       - sit: Situation to simulate ('sun pointing', 'nadir pointing', 'GMO
    pointing')
%       - orbit: RK4 output for the orbit of interest
%               [t (nx1), r (nx3), rDot (nx3), EA (nx3), w (nx3)]
%
%   Outputs:
```

```
%       - out: Integration output matrix, each column is a vector with the
%              same number of elements n as there were timesteps
%               [t (nx1), sig (nx3), w (nx3), u (nx3), sigRef (nx3), wRef (nx3)]
%
    I = x0{1};
    sig_0 = x0{2};
    w_0 = x0{3};
    u_0 = x0{4};

    K = controlParams(1);
    P = controlParams(2);

    X = [sig_0; w_0];
    t = t0;

    out = zeros(length(t0:dt:tf)-1, 16);
    out(1,:) = [t0, X', u_0', X']; % t, sig(1:3), w(1:3), u(1:3), sigRef_0(1:3), wRef_0(1:3)
    k = 1;

    while t < tf
        switch sit
            case "sun pointing"
                R = calcRsN(); % Reference frame is RsN
                wR = zeros(3,1); % RsN doesn't rotate inertially
            case "nadir pointing"
                R = calcRnN(t, orbit_LMO); % Reference frame is RnN
                wR = calcW_RnN(t, orbit_LMO);
            case "GMO pointing"
                R = calcRcN(t, orbit_GMO, orbit_LMO); % Reference frame is RcN
                wR = calcW_RcN(t, dt, orbit_GMO, orbit_LMO);
        end

        sigRef = DCM2MRP(R, 1); % Get around singularity problem with RsN
        wRef = wR;

        [sigBR, omegBR] = calcError(X(1:3), X(4:6), R, wR);

        u = -K*sigBR - P*omegBR;

        k1 = dt*calculateAttitude(X,I,u);
        k2 = dt*calculateAttitude(X+k1/2,I,u);
        k3 = dt*calculateAttitude(X+k2/2,I,u);
        k4 = dt*calculateAttitude(X+k3,I,u);

        X = X + (1/6)*(k1 + 2*k2 + 2*k3 + k4);

        sigNorm = norm(X(1:3));
        if sigNorm > 1.0001
            X(1:3) = -X(1:3)/(sigNorm^2);
        end

        t = t + dt;
```

```
        k = k + 1;

        out(k, :) = [t, X', u', sigRef', wRef'];

    end

end
```

J. Code for Task 9

Back to index: A.A

```
%% ASEN 5010 Task 9 Main Script
% By: Ian Faber

%% Housekeeping
clc; clear; close all

%% Setup
% Include all of our lovely task functions
addpath('..\..\Utilities')
addpath('..\Task1')
addpath('..\Task2')
addpath('..\Task3')
addpath('..\Task4')
addpath('..\Task5')
addpath('..\Task6')

% Make Mars
R_Mars = 3396.19; % km
[marsX, marsY, marsZ] = sphere(100);
marsX = R_Mars*marsX;
marsY = R_Mars*marsY;
marsZ = R_Mars*marsZ;

% Initial orbit parameters
h_LMO = 400; % km
h_GMO = 17028.01; % km
radius_LMO = R_Mars + h_LMO; % km
radius_GMO = R_Mars + h_GMO; % km

w_0_LMO = [0; 0; 0.000884797]; % rad/s, O frame coords
EA_0_LMO = deg2rad([20; 30; 60]); % Omega, i, theta

w_0_GMO = [0; 0; 0.0000709003]; % rad/s, O frame coords
EA_0_GMO = deg2rad([0; 0; 250]); % Omega, i, theta

x_0_LMO = [radius_LMO; EA_0_LMO; w_0_LMO];
x_0_GMO = [radius_GMO; EA_0_GMO; w_0_GMO];

% Initial attitude parameters
sigBN_0 = [0.3; -0.4; 0.5]; % unitless
omegBN_0 = deg2rad([1.00; 1.75; -2.20]); % Converted to rad/s from deg/s
u_0 = zeros(3,1); % Nm
I = diag([10, 5, 7.5]); % kgm^2, body coords

x_0_att = {I; sigBN_0; omegBN_0; u_0};

% Controller parameters
K = 1/180; % 0.0056, from zeta requirement
P = 1/6; % 0.1667, from time decay requirement
controlParams = [K; P];

```

```
% RK4 params
t0 = 0;
dt = 1;
tf = 1000;

%% Propagate orbits and attitude with nadir-pointing control
out_LMO = RK4_Orbit(x_0_LMO, t0, dt, tf);
out_GMO = RK4_Orbit(x_0_GMO, t0, dt, tf);

out_Attitude = RK4_Attitude(x_0_att, t0, dt, tf, 'nadir pointing', out_LMO,
    out_GMO, controlParams);

%% Plot simulation outputs
t = out_Attitude(:,1);
sig = out_Attitude(:,2:4);
w = out_Attitude(:,5:7);
u = out_Attitude(:,8:10);
sigRef = out_Attitude(:,11:13);
wRef = out_Attitude(:,14:16);

% Orbit
figOrbit = figure;

ax2 = axes();
marsStars = imread("marsStars.jpg");
imshow(marsStars, 'parent', ax2)

ax1 = axes();
title("Orbit Simulation", 'Color', 'w')
hold on
grid on
axis equal
plot3(out_LMO(:,2), out_LMO(:,3), out_LMO(:,4), 'LineWidth', 3)
plot3(out_GMO(:,2), out_GMO(:,3), out_GMO(:,4), 'LineWidth', 3)

mars = surf(marsX, marsY, -marsZ); % Need to flip the sphere for image to map
    properly
marsSurface = imread("marsSurface.jpg");
set(mars,'FaceColor','texturemap','cdata',marsSurface,'edgecolor','none');

set(gca, 'Color', 'none', 'XColor', 'w', 'YColor', 'w', 'ZColor', 'w', '
    GridColor', 'w')

xlabel("$\hat{n}_1$", "Interpreter","latex")
ylabel("$\hat{n}_2$", "Interpreter","latex")
zlabel("$\hat{n}_3$", "Interpreter","latex")

view([30 35])

% Attitude
figAttitude = figure;

sgtitle("Nano-satellite Attitude Evolution Over Time")

```

```
subplot(3,3,1)
hold on
grid on
title("\sigma_1 vs. time")
plot(t, sig(:,1))
plot(t, sigRef(:,1), '--')
xlabel("Time [sec]")
ylabel("\sigma_1")
legend("Current", "Reference", 'location', 'best')

subplot(3,3,2)
hold on
grid on
title("\omega_1 vs. time")
plot(t, w(:,1))
plot(t, wRef(:,1), '--')
xlabel("Time [sec]")
ylabel("\omega_1 [rad/s]")
legend("Current", "Reference", 'location', 'best')

subplot(3,3,3)
hold on
grid on
title("u_1 vs. time")
plot(t, u(:,1))
xlabel("Time [sec]")
ylabel("u_1 [Nm]")

subplot(3,3,4)
hold on
grid on
title("\sigma_2 vs. time")
plot(t, sig(:,2))
plot(t, sigRef(:,2), '--')
xlabel("Time [sec]")
ylabel("\sigma_2")
legend("Current", "Reference", 'location', 'best')

subplot(3,3,5)
hold on
grid on
title("\omega_2 vs. time")
plot(t, w(:,2))
plot(t, wRef(:,2), '--')
xlabel("Time [sec]")
ylabel("\omega_2 [rad/s]")
legend("Current", "Reference", 'location', 'best')

subplot(3,3,6)
hold on
grid on
title("u_2 vs. time")
plot(t, u(:,2))
xlabel("Time [sec]")
```

```
ylabel("u_2 [Nm]")

subplot(3,3,7)
hold on
grid on
title("\sigma_3 vs. time")
plot(t, sig(:,3))
plot(t, sigRef(:,3), '--')
xlabel("Time [sec]")
ylabel("\sigma_3")
legend("Current", "Reference", 'location', 'best')

subplot(3,3,8)
hold on
grid on
title("\omega_3 vs. time")
plot(t, w(:,3))
plot(t, wRef(:,3), '--')
xlabel("Time [sec]")
ylabel("\omega_3 [rad/s]")
legend("Current", "Reference", 'location', 'best')

subplot(3,3,9)
hold on
grid on
title("u_3 vs. time")
plot(t, u(:,3))
xlabel("Time [sec]")
ylabel("u_3 [Nm]")

%% Extract answers to text files
time = t;

t = 15;
sig15 = sig(time == t, :);

t = 100;
sig100 = sig(time == t, :);

t = 200;
sig200 = sig(time == t, :);

t = 400;
sig400 = sig(time == t, :);

f1 = fopen("sig15_ans.txt", "w");
ans_sig15 = fprintf(f1, "%.3f %.3f %.3f", sig15(1), sig15(2), sig15(3));
fclose(f1);

f2 = fopen("sig100_ans.txt", "w");
ans_sig100 = fprintf(f2, "%.3f %.3f %.3f", sig100(1), sig100(2), sig100(3));
fclose(f2);

f3 = fopen("sig200_ans.txt", "w");
```

```
ans_sig200 = fprintf(f3, "%.3f %.3f %.3f", sig200(1), sig200(2), sig200(3));
fclose(f3);

f4 = fopen("sig400_ans.txt", "w");
ans_sig400 = fprintf(f4, "%.3f %.3f %.3f", sig400(1), sig400(2), sig400(3));
fclose(f4);
```

```
function dX = calculateAttitude(X, I, u)
% Calculates the nano-satellite body attitude relative to inertial space
%
%   Inputs:
%       - X: State vector at a given point in time
%               [sig_1; sig_2; sig_3; w_1; w_2; w_3]
%       - I: Body fixed inertia matrix
%               [diag(I_11, I_22, I_33)]
%       - u: Control input vector
%               [ u_1; u_2; u_3]
%   Outputs:
%       - dX: Rate of change vector based on the current state
%               [sigDot; wDot]
%

sig = X(1:3);
w = X(4:6);

sigSqr = dot(sig,sig);
sigDot = 0.25*((1-sigSqr)*eye(3) + 2*tilde(sig) + 2*(sig*sig'))*w;

wDot = (I^-1)*(-tilde(w)*I*w + u);

dX = [sigDot; wDot];

end
```

```
function out = RK4_Attitude(x0, t0, dt, tf, sit, orbit_LMO, orbit_GMO, controlParams)
% Function that implements the Runga-Kutta 4 algorithm to integrate
% the attitude of a rigid body subject to a set of initial conditions
%   Inputs:
%       - x0: Initial state vector, with w written in body coordinates
%               {I; sig_0; w_0; u_0}
%       - t0: Time that integration will start, in seconds
%       - dt: Time step for integration, in seconds
%       - tf: Time that integration will stop, in seconds
%       - sit: Situation to simulate ('sun pointing', 'nadir pointing', 'GMO pointing')
%       - orbit: RK4 output for the orbit of interest
%               [t (nx1), r (nx3), rDot (nx3), EA (nx3), w (nx3)]
%
%   Outputs:
%       - out: Integration output matrix, each column is a vector with the
%              same number of elements n as there were timesteps
%               [t (nx1), sig (nx3), w (nx3), u (nx3), sigRef (nx3), wRef (nx3)]
```

```
%
    I = x0{1};
    sig_0 = x0{2};
    w_0 = x0{3};
    u_0 = x0{4};

    K = controlParams(1);
    P = controlParams(2);

    X = [sig_0; w_0];
    t = t0;

    out = zeros(length(t0:dt:tf)-1, 16);
    out(1,:) = [t0, X', u_0', X']; % t, sig(1:3), w(1:3), u(1:3), sigRef_0
        (1:3), wRef_0(1:3)
    k = 1;

    while t < tf
        switch sit
            case "sun pointing"
                R = calcRsN(); % Reference frame is RsN
                wR = zeros(3,1); % RsN doesn't rotate inertially
            case "nadir pointing"
                R = calcRnN(t, orbit_LMO); % Reference frame is RnN
                wR = calcW_RnN(t, orbit_LMO);
            case "GMO pointing"
                R = calcRcN(t, orbit_GMO, orbit_LMO); % Reference frame is RcN
                wR = calcW_RcN(t, dt, orbit_GMO, orbit_LMO);
        end

        sigRef = DCM2MRP(R, 1); % Get around singularity problem with RsN
        wRef = wR;

        [sigBR, omegBR] = calcError(X(1:3), X(4:6), R, wR);

        u = -K*sigBR - P*omegBR;

        k1 = dt*calculateAttitude(X,I,u);
        k2 = dt*calculateAttitude(X+k1/2,I,u);
        k3 = dt*calculateAttitude(X+k2/2,I,u);
        k4 = dt*calculateAttitude(X+k3,I,u);

        X = X + (1/6)*(k1 + 2*k2 + 2*k3 + k4);

        sigNorm = norm(X(1:3));
        if sigNorm > 1.0005
            X(1:3) = -X(1:3)/(sigNorm^2);
        end

        t = t + dt;
        k = k + 1;

        out(k, :) = [t, X', u', sigRef', wRef'];

```

```
    end

end
```

K. Code for Task 10

Back to index: A.A

```
%% ASEN 5010 Task 10 Main Script
% By: Ian Faber

%% Housekeeping
clc; clear; close all

%% Setup
% Include all of our lovely task functions
addpath('..\..\Utilities')
addpath('..\Task1')
addpath('..\Task2')
addpath('..\Task3')
addpath('..\Task4')
addpath('..\Task5')
addpath('..\Task6')

% Make Mars
R_Mars = 3396.19; % km
[marsX, marsY, marsZ] = sphere(100);
marsX = R_Mars*marsX;
marsY = R_Mars*marsY;
marsZ = R_Mars*marsZ;

% Initial orbit parameters
h_LMO = 400; % km
h_GMO = 17028.01; % km
radius_LMO = R_Mars + h_LMO; % km
radius_GMO = R_Mars + h_GMO; % km

w_0_LMO = [0; 0; 0.000884797]; % rad/s, O frame coords
EA_0_LMO = deg2rad([20; 30; 60]); % Omega, i, theta

w_0_GMO = [0; 0; 0.0000709003]; % rad/s, O frame coords
EA_0_GMO = deg2rad([0; 0; 250]); % Omega, i, theta

x_0_LMO = [radius_LMO; EA_0_LMO; w_0_LMO];
x_0_GMO = [radius_GMO; EA_0_GMO; w_0_GMO];

% Initial attitude parameters
sigBN_0 = [0.3; -0.4; 0.5]; % unitless
omegBN_0 = deg2rad([1.00; 1.75; -2.20]); % Converted to rad/s from deg/s
u_0 = zeros(3,1); % Nm
I = diag([10, 5, 7.5]); % kgm^2, body coords

x_0_att = {I; sigBN_0; omegBN_0; u_0};

% Controller parameters
K = 1/180; % 0.0056, from zeta requirement
P = 1/6; % 0.1667, from time decay requirement
controlParams = [K; P];

```

```
% RK4 params
t0 = 0;
dt = 1;
tf = 1000;

%% Propagate orbits and attitude with GMO-pointing control
out_LMO = RK4_Orbit(x_0_LMO, t0, dt, tf);
out_GMO = RK4_Orbit(x_0_GMO, t0, dt, tf);

out_Attitude = RK4_Attitude(x_0_att, t0, dt, tf, 'GMO pointing', out_LMO,
    out_GMO, controlParams);

%% Plot simulation outputs
t = out_Attitude(:,1);
sig = out_Attitude(:,2:4);
w = out_Attitude(:,5:7);
u = out_Attitude(:,8:10);
sigRef = out_Attitude(:,11:13);
wRef = out_Attitude(:,14:16);

% Orbit
figOrbit = figure;

ax2 = axes();
marsStars = imread("marsStars.jpg");
imshow(marsStars, 'parent', ax2)

ax1 = axes();
title("Orbit Simulation", 'Color', 'w')
hold on
grid on
axis equal
plot3(out_LMO(:,2), out_LMO(:,3), out_LMO(:,4), 'LineWidth', 3)
plot3(out_GMO(:,2), out_GMO(:,3), out_GMO(:,4), 'LineWidth', 3)

mars = surf(marsX, marsY, -marsZ); % Need to flip the sphere for image to map
    properly
marsSurface = imread("marsSurface.jpg");
set(mars,'FaceColor','texturemap','cdata',marsSurface,'edgecolor','none');

set(gca, 'Color', 'none', 'XColor', 'w', 'YColor', 'w', 'ZColor', 'w', '
    GridColor', 'w')

xlabel("$\hat{n}_1$", "Interpreter","latex")
ylabel("$\hat{n}_2$", "Interpreter","latex")
zlabel("$\hat{n}_3$", "Interpreter","latex")

view([30 35])

% Attitude
figAttitude = figure;

sgtitle("Nano-satellite Attitude Evolution Over Time")

```

```
subplot(3,3,1)
hold on
grid on
title("\sigma_1 vs. time")
plot(t, sig(:,1))
plot(t, sigRef(:,1), '--')
xlabel("Time [sec]")
ylabel("\sigma_1")
legend("Current", "Reference", 'location', 'best')

subplot(3,3,2)
hold on
grid on
title("\omega_1 vs. time")
plot(t, w(:,1))
plot(t, wRef(:,1), '--')
xlabel("Time [sec]")
ylabel("\omega_1 [rad/s]")
legend("Current", "Reference", 'location', 'best')

subplot(3,3,3)
hold on
grid on
title("u_1 vs. time")
plot(t, u(:,1))
xlabel("Time [sec]")
ylabel("u_1 [Nm]")

subplot(3,3,4)
hold on
grid on
title("\sigma_2 vs. time")
plot(t, sig(:,2))
plot(t, sigRef(:,2), '--')
xlabel("Time [sec]")
ylabel("\sigma_2")
legend("Current", "Reference", 'location', 'best')

subplot(3,3,5)
hold on
grid on
title("\omega_2 vs. time")
plot(t, w(:,2))
plot(t, wRef(:,2), '--')
xlabel("Time [sec]")
ylabel("\omega_2 [rad/s]")
legend("Current", "Reference", 'location', 'best')

subplot(3,3,6)
hold on
grid on
title("u_2 vs. time")
plot(t, u(:,2))
xlabel("Time [sec]")
```

```
ylabel("u_2 [Nm]")

subplot(3,3,7)
hold on
grid on
title("\sigma_3 vs. time")
plot(t, sig(:,3))
plot(t, sigRef(:,3), '--')
xlabel("Time [sec]")
ylabel("\sigma_3")
legend("Current", "Reference", 'location', 'best')

subplot(3,3,8)
hold on
grid on
title("\omega_3 vs. time")
plot(t, w(:,3))
plot(t, wRef(:,3), '--')
xlabel("Time [sec]")
ylabel("\omega_3 [rad/s]")
legend("Current", "Reference", 'location', 'best')

subplot(3,3,9)
hold on
grid on
title("u_3 vs. time")
plot(t, u(:,3))
xlabel("Time [sec]")
ylabel("u_3 [Nm]")

%% Extract answers to text files
time = t;

t = 15;
sig15 = sig(time == t, :);

t = 100;
sig100 = sig(time == t, :);

t = 200;
sig200 = sig(time == t, :);

t = 400;
sig400 = sig(time == t, :);

f1 = fopen("sig15_ans.txt", "w");
ans_sig15 = fprintf(f1, "%.3f %.3f %.3f", sig15(1), sig15(2), sig15(3));
fclose(f1);

f2 = fopen("sig100_ans.txt", "w");
ans_sig100 = fprintf(f2, "%.3f %.3f %.3f", sig100(1), sig100(2), sig100(3));
fclose(f2);

f3 = fopen("sig200_ans.txt", "w");
```

```
ans_sig200 = fprintf(f3, "%.3f %.3f %.3f", sig200(1), sig200(2), sig200(3));
fclose(f3);

f4 = fopen("sig400_ans.txt", "w");
ans_sig400 = fprintf(f4, "%.3f %.3f %.3f", sig400(1), sig400(2), sig400(3));
fclose(f4);
```

```
function dX = calculateAttitude(X, I, u)
% Calculates the nano-satellite body attitude relative to inertial space
%
%   Inputs:
%       - X: State vector at a given point in time
%               [sig_1; sig_2; sig_3; w_1; w_2; w_3]
%       - I: Body fixed inertia matrix
%               [diag(I_11, I_22, I_33)]
%       - u: Control input vector
%               [ u_1; u_2; u_3]
%   Outputs:
%       - dX: Rate of change vector based on the current state
%               [sigDot; wDot]
%

sig = X(1:3);
w = X(4:6);

sigSqr = dot(sig,sig);
sigDot = 0.25*((1-sigSqr)*eye(3) + 2*tilde(sig) + 2*(sig*sig'))*w;

wDot = (I^-1)*(-tilde(w)*I*w + u);

dX = [sigDot; wDot];

end
```

```
function out = RK4_Attitude(x0, t0, dt, tf, sit, orbit_LMO, orbit_GMO,
    controlParams)
% Function that implements the Runga-Kutta 4 algorithm to integrate
% the attitude of a rigid body subject to a set of initial conditions
%   Inputs:
%       - x0: Initial state vector, with w written in body coordinates
%               {I; sig_0; w_0; u_0}
%       - t0: Time that integration will start, in seconds
%       - dt: Time step for integration, in seconds
%       - tf: Time that integration will stop, in seconds
%       - sit: Situation to simulate ('sun pointing', 'nadir pointing', 'GMO
    pointing')
%       - orbit: RK4 output for the orbit of interest
%               [t (nx1), r (nx3), rDot (nx3), EA (nx3), w (nx3)]
%
%   Outputs:
%       - out: Integration output matrix, each column is a vector with the
%              same number of elements n as there were timesteps
%               [t (nx1), sig (nx3), w (nx3), u (nx3), sigRef (nx3), wRef (nx3
    )]
```

```
%
    I = x0{1};
    sig_0 = x0{2};
    w_0 = x0{3};
    u_0 = x0{4};

    K = controlParams(1);
    P = controlParams(2);

    X = [sig_0; w_0];
    t = t0;

    out = zeros(length(t0:dt:tf)-1, 16);
    out(1,:) = [t0, X', u_0', X']; % t, sig(1:3), w(1:3), u(1:3), sigRef_0
        (1:3), wRef_0(1:3)
    k = 1;

    while t < tf
        switch sit
            case "sun pointing"
                R = calcRsN(); % Reference frame is RsN
                wR = zeros(3,1); % RsN doesn't rotate inertially
            case "nadir pointing"
                R = calcRnN(t, orbit_LMO); % Reference frame is RnN
                wR = calcW_RnN(t, orbit_LMO);
            case "GMO pointing"
                R = calcRcN(t, orbit_GMO, orbit_LMO); % Reference frame is RcN
                wR = calcW_RcN(t, dt, orbit_GMO, orbit_LMO);
        end

        sigRef = DCM2MRP(R, 1); % Get around singularity problem with RsN
        wRef = wR;

        [sigBR, omegBR] = calcError(X(1:3), X(4:6), R, wR);

        u = -K*sigBR - P*omegBR;

        k1 = dt*calculateAttitude(X,I,u);
        k2 = dt*calculateAttitude(X+k1/2,I,u);
        k3 = dt*calculateAttitude(X+k2/2,I,u);
        k4 = dt*calculateAttitude(X+k3,I,u);

        X = X + (1/6)*(k1 + 2*k2 + 2*k3 + k4);

        sigNorm = norm(X(1:3));
        if sigNorm > 1.0005
            X(1:3) = -X(1:3)/(sigNorm^2);
        end

        t = t + dt;
        k = k + 1;

        out(k, :) = [t, X', u', sigRef', wRef'];

```

```
    end

end
```

L. Code for Task 11

Back to index: A.A

```
%% ASEN 5010 Task 11 Main Script
% By: Ian Faber

%% Housekeeping
clc; clear; close all

%% Setup
% Include all of our lovely task functions
addpath('..\..\Utilities')
addpath('..\Task1')
addpath('..\Task2')
addpath('..\Task3')
addpath('..\Task4')
addpath('..\Task5')
addpath('..\Task6')

% Make Mars
R_Mars = 3396.19; % km
[marsX, marsY, marsZ] = sphere(100);
marsX = R_Mars*marsX;
marsY = R_Mars*marsY;
marsZ = R_Mars*marsZ;
marsAngle = 0; % rad

% Initial orbit parameters
h_LMO = 400; % km
h_GMO = 17028.01; % km
radius_LMO = R_Mars + h_LMO; % km
radius_GMO = R_Mars + h_GMO; % km

w_0_LMO = [0; 0; 0.000884797]; % rad/s, O frame coords
EA_0_LMO = deg2rad([20; 30; 60]); % Omega, i, theta

w_0_GMO = [0; 0; 0.0000709003]; % rad/s, O frame coords
EA_0_GMO = deg2rad([0; 0; 250]); % Omega, i, theta

x_0_LMO = [radius_LMO; EA_0_LMO; w_0_LMO];
x_0_GMO = [radius_GMO; EA_0_GMO; w_0_GMO];

% Initial attitude parameters
sigBN_0 = [0.3; -0.4; 0.5]; % unitless
omegBN_0 = deg2rad([1.00; 1.75; -2.20]); % Converted to rad/s from deg/s
u_0 = zeros(3,1); % Nm
I = diag([10, 5, 7.5]); % kgm^2, body coords

x_0_att = {I; sigBN_0; omegBN_0; u_0};

% Controller parameters
K = 1/180; % 0.0056, from zeta requirement
P = 1/6; % 0.1667, from time decay requirement
controlParams = [K; P];
```

```matlab

% RK4 params
t0 = 0;
dt = 1;
tf = 6500;
% tf = 90000;

%% Propagate orbits and attitude with full pointing control
out_LMO = RK4_Orbit(x_0_LMO, t0, dt, tf);
out_GMO = RK4_Orbit(x_0_GMO, t0, dt, tf);

[out_Attitude, states_attitude] = RK4_Attitude(x_0_att, t0, dt, tf, out_LMO,
    out_GMO, controlParams);

%% Extract and process simulation outputs
t = out_Attitude(:,1);
sig = out_Attitude(:,2:4);
w = out_Attitude(:,5:7);
u = out_Attitude(:,8:10);
sigRef = out_Attitude(:,11:13);
wRef = out_Attitude(:,14:16);
states = states_attitude;

idxSun = states == "sun pointing";
idxNadir = states == "nadir pointing";
idxGMO = states == "GMO pointing";

tInt = [300; 2100; 3400; 4400; 5600];

%% Attitude and control plots
figAttitude = figure;

sgtitle("Nano-satellite Attitude Evolution Over Time")

subplot(3,3,1)
hold on
grid on
title("\sigma_1 vs. time")
plot(t, sig(:,1))
plot(t, sigRef(:,1), '--')
xlabel("Time [sec]")
ylabel("\sigma_1")
legend("Current", "Reference", 'location', 'best')

subplot(3,3,2)
hold on
grid on
title("\omega_1 vs. time")
plot(t, w(:,1))
plot(t, wRef(:,1), '--')
xlabel("Time [sec]")
ylabel("\omega_1 [rad/s]")
legend("Current", "Reference", 'location', 'best')

```

```
subplot(3,3,3)
hold on
grid on
title("u_1 vs. time")
plot(t, u(:,1))
xlabel("Time [sec]")
ylabel("u_1 [Nm]")

subplot(3,3,4)
hold on
grid on
title("\sigma_2 vs. time")
plot(t, sig(:,2))
plot(t, sigRef(:,2), '--')
xlabel("Time [sec]")
ylabel("\sigma_2")
legend("Current", "Reference", 'location', 'best')

subplot(3,3,5)
hold on
grid on
title("\omega_2 vs. time")
plot(t, w(:,2))
plot(t, wRef(:,2), '--')
xlabel("Time [sec]")
ylabel("\omega_2 [rad/s]")
legend("Current", "Reference", 'location', 'best')

subplot(3,3,6)
hold on
grid on
title("u_2 vs. time")
plot(t, u(:,2))
xlabel("Time [sec]")
ylabel("u_2 [Nm]")

subplot(3,3,7)
hold on
grid on
title("\sigma_3 vs. time")
plot(t, sig(:,3))
plot(t, sigRef(:,3), '--')
xlabel("Time [sec]")
ylabel("\sigma_3")
legend("Current", "Reference", 'location', 'best')

subplot(3,3,8)
hold on
grid on
title("\omega_3 vs. time")
plot(t, w(:,3))
plot(t, wRef(:,3), '--')
xlabel("Time [sec]")
ylabel("\omega_3 [rad/s]")
```

```matlab
legend("Current", "Reference", 'location', 'best')

subplot(3,3,9)
hold on
grid on
title("u_3 vs. time")
plot(t, u(:,3))
xlabel("Time [sec]")
ylabel("u_3 [Nm]")

%% Orbit animation
figOrbit = figure('Position',[0 0 1920 1080]);

movieVector = [];
frames = [];
dTime = 50;

for k = 1:dTime:length(t)
    clf
    hold on
    grid on
    axis equal

    % subplot(1,2,1)
    % ax2 = axes();
    % marsStars = imread("marsStars.jpg");
    % imshow(marsStars, 'parent', ax2)

    % ax1 = axes();
    titleText = sprintf("ASEN 5010 Capstone: t = %.0f sec, mode = %s", t(k),
        states(k));
    title(titleText, 'Color', 'k')
    % axis equal

    % Orbits
    nanoOrbit = plot3(out_LMO(:,2), out_LMO(:,3), out_LMO(:,4), 'LineWidth',
        2);
    motherOrbit = plot3(out_GMO(:,2), out_GMO(:,3), out_GMO(:,4), 'LineWidth',
         2);

    % Spacecraft
        % Nano-sat
    nanoNH = calcHN(t(k), out_LMO)';
    nanoDCM = MRP2DCM(out_Attitude(k,2:4))';
    nanoSat = scatter3(out_LMO(k,2), out_LMO(k,3), out_LMO(k,4), 25, 'black',
        'filled');
    quiver3(0, 0, 0, nanoNH(1,1), nanoNH(2,1), nanoNH(3,1), 6250, 'k') % o_1
        axis
    quiver3(0, 0, 0, nanoNH(1,2), nanoNH(2,2), nanoNH(3,2), 6250, 'k') % o_2
        axis
    quiver3(0, 0, 0, nanoNH(1,3), nanoNH(2,3), nanoNH(3,3), 6250, 'k') % o_3
        axis
    b_1 = quiver3(out_LMO(k,2), out_LMO(k,3), out_LMO(k,4), nanoDCM(1,1),
        nanoDCM(2,1), nanoDCM(3,1), 2500, 'r'); % b_1 axis
```

```
    negB_1 = quiver3(out_LMO(k,2), out_LMO(k,3), out_LMO(k,4), -nanoDCM(1,1), -nanoDCM(2,1), -nanoDCM(3,1), 2500, 'r:'); % -b_1 axis
    b_2 = quiver3(out_LMO(k,2), out_LMO(k,3), out_LMO(k,4), nanoDCM(1,2), nanoDCM(2,2), nanoDCM(3,2), 2500, 'g'); % b_2 axis
    b_3 = quiver3(out_LMO(k,2), out_LMO(k,3), out_LMO(k,4), nanoDCM(1,3), nanoDCM(2,3), nanoDCM(3,3), 2500, 'b'); % b_3 axis
        % Mothercraft
    GMOSat = scatter3(out_GMO(k,2), out_GMO(k,3), out_GMO(k,4), 25, 'magenta', 'filled');
    GMONH = calcHN(t(k), out_GMO)';
    quiver3(0, 0, 0, GMONH(1,1), GMONH(2,1), GMONH(3,1), 25000, 'k') % o_1 axis
    quiver3(0, 0, 0, GMONH(1,2), GMONH(2,2), GMONH(3,2), 25000, 'k') % o_2 axis
    quiver3(0, 0, 0, GMONH(1,3), GMONH(2,3), GMONH(3,3), 25000, 'k') % o_3 axis

    % Other frames
        % Communication frame
    % RcN = calcRcN(t(k), out_GMO, out_LMO)';
    % quiver3(out_LMO(k,2), out_LMO(k,3), out_LMO(k,4), RcN(1,1), RcN(2,1), RcN(3,1), 10000, 'r') % r_1 axis
    % quiver3(out_LMO(k,2), out_LMO(k,3), out_LMO(k,4), -RcN(1,1), -RcN(2,1), -RcN(3,1), 10000, 'r:') % -r_1 axis
    % quiver3(out_LMO(k,2), out_LMO(k,3), out_LMO(k,4), RcN(1,2), RcN(2,2), RcN(3,2), 10000, 'g') % r_2 axis
    % quiver3(out_LMO(k,2), out_LMO(k,3), out_LMO(k,4), RcN(1,3), RcN(2,3), RcN(3,3), 10000, 'b') % r_3 axis

    % Mars
    marsAngle = marsAngle + dTime*(w_0_GMO(3)); % How much has Mars rotated over the timestep?
    marsXrot = marsX*cos(marsAngle) - marsY*sin(marsAngle);
    marsYrot = marsX*sin(marsAngle) + marsY*cos(marsAngle);
    marsZrot = -marsZ; % Need to flip the sphere for image to map properly
    mars = surf(marsXrot, marsYrot, marsZrot);
    marsSurface = imread("marsSurface.jpg");
    set(mars,'FaceColor','texturemap','cdata',marsSurface,'edgecolor','none');

    % set(gca, 'Color', 'none', 'XColor', 'k', 'YColor', 'k', 'ZColor', 'k', 'GridColor', 'k')
    xlim([-21000 21000])
    ylim([-21000 21000])
    zlim([-15000 15000])
    xlabel("$\hat{n}_1$", "Interpreter","latex")
    ylabel("$\hat{n}_2$", "Interpreter","latex")
    zlabel("$\hat{n}_3$", "Interpreter","latex")

    view([30 35])

    legend([nanoOrbit, motherOrbit, nanoSat, GMOSat, b_1, negB_1, b_2, b_3], "Nano-satellite Orbit", "Mothercraft Orbit", "Nano-satellite", "Mothercraft", "b_1 axis", "-b_1 axis", "b_2 axis", "b_3 axis", 'Location', 'best')
```

```
    % if any(t(k) == tInt)
    %     frames = [frames; getframe(figOrbit)];
    % end

    % subplot(1,2,2)
    % % Spacecraft
    % scatter3(out_LMO(k,2), out_LMO(k,3), out_LMO(k,4), 25, 'black', 'filled')
    % scatter3(out_GMO(k,2), out_GMO(k,3), out_GMO(k,4), 25, 'black', 'filled')

    drawnow
    % movieVector = [movieVector; getframe(figOrbit)];
end

% for k = 1:length(frames)
%     switch k
%         case 1
%             imwrite(frames(k).cdata, "pointingAt300s.png")
%         case 2
%             imwrite(frames(k).cdata, "pointingAt2100s.png")
%         case 3
%             imwrite(frames(k).cdata, "pointingAt3400s.png")
%         case 4
%             imwrite(frames(k).cdata, "pointingAt4400s.png")
%         case 5
%             imwrite(frames(k).cdata, "pointingAt5600s.png")
%     end
% end

% movie = VideoWriter('FinalProjectMovie','MPEG-4');
% movie.FrameRate = 30;
%
% %Open the VideoWriter object, write the movie, and close the file
% open(movie);
% writeVideo(movie, movieVector);
% close(movie);

%% Extract answers to text files
time = t;

t = 300;
sig300 = sig(time == t, :);

t = 2100;
sig2100 = sig(time == t, :);

t = 3400;
sig3400 = sig(time == t, :);

t = 4400;
sig4400 = sig(time == t, :);

```

```
t = 5600;
sig5600 = sig(time == t, :);

f1 = fopen("sig300_ans.txt", "w");
ans_sig300 = fprintf(f1, "%.3f %.3f %.3f", sig300(1), sig300(2), sig300(3));
fclose(f1);

f2 = fopen("sig2100_ans.txt", "w");
ans_sig2100 = fprintf(f2, "%.3f %.3f %.3f", sig2100(1), sig2100(2), sig2100(3)
    );
fclose(f2);

f3 = fopen("sig3400_ans.txt", "w");
ans_sig3400 = fprintf(f3, "%.3f %.3f %.3f", sig3400(1), sig3400(2), sig3400(3)
    );
fclose(f3);

f4 = fopen("sig4400_ans.txt", "w");
ans_sig4400 = fprintf(f4, "%.3f %.3f %.3f", sig4400(1), sig4400(2), sig4400(3)
    );
fclose(f4);

f5 = fopen("sig5600_ans.txt", "w");
ans_sig5600 = fprintf(f5, "%.3f %.3f %.3f", sig5600(1), sig5600(2), sig5600(3)
    );
fclose(f5);
```

```
function dX = calculateAttitude(X, I, u)
% Calculates the nano-satellite body attitude relative to inertial space
%
%   Inputs:
%       - X: State vector at a given point in time
%               [sig_1; sig_2; sig_3; w_1; w_2; w_3]
%       - I: Body fixed inertia matrix
%               [diag(I_11, I_22, I_33)]
%       - u: Control input vector
%               [ u_1; u_2; u_3]
%   Outputs:
%       - dX: Rate of change vector based on the current state
%               [sigDot; wDot]
%

sig = X(1:3);
w = X(4:6);

sigSqr = dot(sig,sig);
sigDot = 0.25*((1-sigSqr)*eye(3) + 2*tilde(sig) + 2*(sig*sig'))*w;

wDot = (I^-1)*(-tilde(w)*I*w + u);

dX = [sigDot; wDot];

end
```

```
function [out, states] = RK4_Attitude(x0, t0, dt, tf, orbit_LMO, orbit_GMO, controlParams)
% Function that implements the Runga-Kutta 4 algorithm to integrate
% the attitude of a rigid body subject to a set of initial conditions
%   Inputs:
%       - x0: Initial state vector, with w written in body coordinates
%               {I; sig_0; w_0; u_0}
%       - t0: Time that integration will start, in seconds
%       - dt: Time step for integration, in seconds
%       - tf: Time that integration will stop, in seconds
%       - orbit: RK4 output for the orbit of interest
%               [t (nx1), r (nx3), rDot (nx3), EA (nx3), w (nx3)]
%
%   Outputs:
%       - out: Integration output matrix, each column is a vector with the
%              same number of elements n as there were timesteps
%               [t (nx1), sig (nx3), w (nx3), u (nx3), sigRef (nx3),
%                wRef (nx3)]
%       - states: Vector of pointing states at the current time (nx1)
%
    I = x0{1};
    sig_0 = x0{2};
    w_0 = x0{3};
    u_0 = x0{4};

    K = controlParams(1);
    P = controlParams(2);

    X = [sig_0; w_0];
    t = t0;

    out = zeros(length(t0:dt:tf)-1, 16);
    states = strings(length(t0:dt:tf)-1,1);
    out(1,:) = [t0, X', u_0', X']; % t, sig(1:3), w(1:3), u(1:3), sigRef_0(1:3), wRef_0(1:3)
    states(1) = "initial";
    k = 1;

    while t < tf
        r_LMO = orbit_LMO(orbit_LMO(:,1) == t, 2:4);
        r_GMO = orbit_GMO(orbit_GMO(:,1) == t, 2:4);

        angle = acosd(dot(r_LMO, r_GMO)/(norm(r_LMO)*norm(r_GMO))); % degrees

        % Determine pointing reference frame state
        if r_LMO(2) > 0 % Spacecraft is on the positive n_2 side of Mars
            sit = "sun pointing";
        elseif abs(angle) <= 35 % Spacecraft and mothercraft are separated by less than or equal to 35 degrees
            sit = "GMO pointing";
        else % Spacecraft isn't in the sun and can't communicate, do science!
            sit = "nadir pointing";
        end
```

```matlab

        if t== 300 || t == 2100 || t == 3400 || t == 4400 || t == 5600
            fprintf("t = %.1f s, angle = %.3f deg, sit = %s \n", t, angle, sit
                )
        end

        % Reference frame state machine
        switch sit
            case "sun pointing"
                R = calcRsN(); % Reference frame is RsN
                wR = zeros(3,1); % RsN doesn't rotate inertially
            case "nadir pointing"
                R = calcRnN(t, orbit_LMO); % Reference frame is RnN
                wR = calcW_RnN(t, orbit_LMO);
            case "GMO pointing"
                R = calcRcN(t, orbit_GMO, orbit_LMO); % Reference frame is RcN
                wR = calcW_RcN(t, dt, orbit_GMO, orbit_LMO);
        end

        sigRef = DCM2MRP(R, 1); % Get around singularity problem with RsN
        wRef = wR;

        [sigBR, omegBR] = calcError(X(1:3), X(4:6), R, wR);

        u = -K*sigBR - P*omegBR;

        k1 = dt*calculateAttitude(X,I,u);
        k2 = dt*calculateAttitude(X+k1/2,I,u);
        k3 = dt*calculateAttitude(X+k2/2,I,u);
        k4 = dt*calculateAttitude(X+k3,I,u);

        X = X + (1/6)*(k1 + 2*k2 + 2*k3 + k4);

        sigNorm = norm(X(1:3));
        if sigNorm > 1.00005 % Buffer for sigma at 1 exactly
            X(1:3) = -X(1:3)/(sigNorm^2);
        end

        t = t + dt;
        k = k + 1;

        out(k, :) = [t, X', u', sigRef', wRef'];
        states(k) = sit;

    end

end
```

M. Utility code

Back to index: A.A

```
function EP = DCM2EP(C)
    % Sheppard's method
    q0 = sqrt(0.25*(1 + trace(C)));
    q1 = sqrt(0.25*(1 - trace(C) + 2*C(1,1)));
    q2 = sqrt(0.25*(1 - trace(C) + 2*C(2,2)));
    q3 = sqrt(0.25*(1 - trace(C) + 2*C(3,3)));

    q = [q0, q1, q2, q3];

    [~, idx] = max(q);

    if idx == 1 % q0 was largest, can divide by it safely
        q1 = (C(2,3)-C(3,2))/(4*q0);
        q2 = (C(3,1)-C(1,3))/(4*q0);
        q3 = (C(1,2)-C(2,1))/(4*q0);
        % q0 = (C(2,3)-C(3,2))/(4*q1); % Check sign on q0
    elseif idx == 2 % q1 was largest, can divide by it safely
        q0 = (C(2,3)-C(3,2))/(4*q1);
        q2 = (C(1,2)+C(2,1))/(4*q1);
        q3 = (C(3,1)+C(1,3))/(4*q1);
        % q1 = (C(2,3)-C(3,2))/(4*q0); % Check sign on q1
    elseif idx == 3 % q2 was largest, can divide by it safely
        q0 = (C(3,1)-C(1,3))/(4*q2);
        q1 = (C(1,2)+C(2,1))/(4*q2);
        q3 = (C(2,3)+C(3,2))/(4*q2);
        % q2 = (C(3,1)-C(1,3))/(4*q0); % Check sign on q2
    else
        q0 = (C(1,2)-C(2,1))/(4*q3);
        q1 = (C(3,1)+C(1,3))/(4*q3);
        q2 = (C(2,3)+C(3,2))/(4*q3);
        % q3 = (C(1,2)-C(2,1))/(4*q0); % Check sign on q3
    end

    EP = [q0, q1, q2, q3]';

end
```

```
function sigma = DCM2MRP(mat, short)

% q = DCM2CRP(mat)
%
% sig = q/(1+sqrt(1+q'*q));
%
% sigMag = norm(sig)^2
%
% if short
%     if sigMag <= 1
%         sigma = sig;
%     else
%         sigma = -sig/(sigMag^2);
%     end
```

```
% else
%     if sigMag <= 1
%         sigma = -sig/(sigMag^2);
%     else
%         sigma = sig;
%     end
% end

zeta = sqrt(trace(mat) + 1);

if zeta == 0
    EP = DCM2EP(mat);
    sig = [EP(2)/(1+EP(1)); EP(3)/(1+EP(1)); EP(4)/(1+EP(1))];
else
    sig = (1/(zeta*(zeta + 2)))*[mat(2,3) - mat(3,2); mat(3,1) - mat(1,3); mat
       (1,2) - mat(2,1)];
end

sigMag = norm(sig)^2;

if short
    if sigMag <= 1
        sigma = sig;
    else
        sigma = -sig/(sigMag^2);
    end
else
    if sigMag <= 1
        sigma = -sig/(sigMag^2);
    else
        sigma = sig;
    end
end

end
```

```
function mat = MRP2DCM(sigma)

mat = eye(3) + (1/(1+norm(sigma)^2)^2)*(8*tilde(sigma)*tilde(sigma) - 4*(1-
   norm(sigma)^2)*tilde(sigma));

end
```

```
function mat = EA2DCM(angles, type)
% Function that converts Euler angles to a DCM for 3D rotations
%   Inputs:
%       - angles: Vector of Euler angles corresponding to the axes in
%                 "type" in RADIANS
%       - type: Vector of axis rotations that the angles in "angles" will
%               be applied to
%   Outputs:
%       - mat: DCM for the (type) Euler Angle rotation through (angles)
%
%   Example:
```

```matlab
%       mat = EA2DCM([15, 34, 86], [3,2,1])
%
%             This will result in a (3-2-1) Euler angle rotation through
%             the angles 15, 34, and 86 degrees. In this case, mat will
%             be the DCM resulting from a (15) degree rotation about the
%             (3) axis, then a (34) degree rotation about the (2) axis,
%             then an (86) degree rotation about the (1) axis
%
%   Author: Ian Faber

M1 = @(theta) [
                1,          0,          0;
                0,   cos(theta), sin(theta);
                0,   -sin(theta), cos(theta)
              ];

M2 = @(theta) [
                cos(theta), 0,   -sin(theta);
                0,          1,            0;
                sin(theta), 0,   cos(theta)
              ];


M3 = @(theta) [
                cos(theta), sin(theta), 0;
                -sin(theta), cos(theta), 0;
                0,          0,          1
              ];

rotMats = {M1, M2, M3};

mat = (rotMats{type(3)}(angles(3)))*(rotMats{type(2)}(angles(2)))*(rotMats{
    type(1)}(angles(1)));

end
```

```matlab
function dfdt = finiteDifMat(t0, dt, f)
% Caculates a numerical difference quotient for a 3x3 matrix

f1 = cell2mat(f(cell2mat(f(:,1)) == t0+dt, 2));
f2 = cell2mat(f(cell2mat(f(:,1)) == t0, 2));
dfdt = (f1 - f2)./dt;

end
```

```matlab
function mat = tilde(v)
% Outputs the tilde (cross product) matrix for a given vector v
%
%   - Inputs: 3x1 vector v
%   - Outputs: 3x3 matrix mat

mat = [
        0,      -v(3),   v(2);
        v(3),   0,      -v(1);
```

```
        -v(2),  v(1),   0
      ];


end
```