University of Colorado - Boulder

ASEN 6020
Optimal Trajectories | Spring 2025

NASCAR for CubeSats: An Application of Primer Vector Theory

Author:
Ian Faber[a]
[a]SID: 108577813

Instructor:
Daniel Scheeres

Wednesday, May 7, 2025

College of Engineering & Applied Science
UNIVERSITY OF COLORADO **BOULDER**

The concept of a "CubeSat race course" is explored, in which, given a randomized series of waypoint rings and a randomized starting position, the optimal trajectory between waypoints is computed that minimizes the time taken for a CubeSat to traverse the entire course while also minimizing the distance to the center of each ring.

I. Project Overview

Optimal control has quickly become a favorite topic of mine this semester, in particular primer vector theory. The first time we talked about it, I immediately had an idea: How can we use this to kickstart a cutting edge entertainment industry that features CubeSats threading the needle through precariously placed rings? And what if we set the rings on fire! (just kidding). While this may seem facetious, the underlying problem is fascinating, and a perfect application of primer vector theory.

For my implementation of this problem, I will be leveraging the Clohessy-Hill-Wiltshire equations for the underlying dynamics model governing the CubeSats. The state to be controlled will be the typical relative motion state, arranged as follows:

$$\mathbf{x} = [x\ y\ z\ \dot{x}\ \dot{y}\ \dot{z}\ ]^\intercal \tag{1}$$

Where the origin (classically referred to as the "chief spacecraft") is the center of the race course. The CubeSats will have thrusters to control their x, y, and z accelerations like so:

$$\mathbf{u} = \begin{bmatrix} u_x\ u_y\ u_z\ \end{bmatrix}^\intercal \tag{2}$$

Where each CubeSat has a specified maximum thrust u_{max}. Further, some CubeSats might have maximum thrusts that depend on a specific axis, i.e. each axis having a specified maximum thrust $u_{i,\,max}$. Thus, the dynamics of the system are as follows [1]:

$$\mathbf{f}(\mathbf{x}, \mathbf{u}, t) = \begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{z} \\ 2n\dot{y} + 3n^2 x + u_x \\ -2n\dot{x} + u_y \\ -n^2 z + u_z \end{bmatrix} = \begin{bmatrix} \mathbf{v} \\ \mathbf{a}_{\text{grav}} + \mathbf{u} \end{bmatrix} \tag{3}$$

Where n is the mean motion of the race course origin's orbit around Earth, x is the radial displacement, y is the along-track displacement, and z is the cross-track displacement of the CubeSat from the race course origin.

Each run of the problem will have a randomly generated race course that follows a number of physically realistic rules:

1) Race course rings will be assumed to remain fixed in the Hill Frame (radial, along-track, cross-track), i.e. the CHW equations will not apply to the rings. This is a simplification to ensure that rings don't drift over the time period of the problem.
2) Individual race course rings shall have semi-major and semi-minor axes uniformly distributed between $[1, 5]$ m.
3) Rings shall be spaced apart with uniformly distributed azimuth angles $\theta \in [-90, 90]^\circ$, elevation angles $\phi \in [-90, 90]^\circ$, and inter-ring distances $d \in [300, 350]$ m.
4) The first race course ring will be aligned along the +y axis.
5) Each race course shall consist of a uniformly distributed number of rings $l \in [15, 25]$ rings.

Each CubeSat will have a similar set of rules:

1) CubeSat initial states shall be randomized according to a covariance of diag($[250, 0, 250]$) m^2 in x, y, and z respectively around a point exactly 350 m behind the first course ring. CubeSats shall be initially at rest.
2) CubeSat final states shall be at rest after passing through the last ring.
3) CubeSats will successfully pass through the center of a ring if their velocity vector at that ring is aligned with the normal vector of that ring, their final position is within the semiminor axis of the ring, and their final position is within 0.1 meters of the plane of the ring.

The overall goal of each CubeSat is to make it through the race course in the smallest amount of time while successfully passing through each course ring as close to its center as possible. Thus, the full optimal control problem can be split into a series of Mayer optimal control problems between course rings as follows:

Initial/Final Optimal Control Problems

$$\dot{\mathbf{x}} = \mathbf{f}(\mathbf{x}, \mathbf{u}, t)$$

$$J = K(\mathbf{x}_0, t_0, \mathbf{x}_f, t_f) = \left\|\mathbf{r}_f - \mathbf{r}_{r,1}\right\| + t_f - t_0 \textbf{ or } t_f - t_0 \tag{4}$$

$$\mathbf{g}(\mathbf{x}_0, t_0, \mathbf{x}_f, t_f) = \begin{bmatrix} \mathbf{x}(t_0) - \mathbf{x}_{0,\text{init}} \\ t_0 \\ \hat{\mathbf{v}}(\mathbf{x}_f) - \hat{n}_1 \end{bmatrix} \textbf{ or } \begin{bmatrix} \mathbf{x}(t_0) - \mathbf{x}_{0,l} \\ t_0 - t_{0,l} \\ \mathbf{v}(\mathbf{x}_f) \end{bmatrix} = \mathbf{0} \tag{5}$$

$$\mathbf{h}(\mathbf{x}_f) = \begin{bmatrix} \left\|\mathbf{r}_f - \mathbf{r}_{r,1}\right\| - \min(S_1) \\ \left|(\mathbf{r}_f - \mathbf{r}_{r,1}) \cdot \hat{n}_1\right| - 0.1 \end{bmatrix} \leq \mathbf{0}, \text{ initial problem only}$$

Intermediate Optimal Control Problems

$$\dot{\mathbf{x}} = \mathbf{f}(\mathbf{x}, \mathbf{u}, t)$$

$$J = K(\mathbf{x}_0, t_0, \mathbf{x}_f, t_f) = \left\|\mathbf{r}_f - \mathbf{r}_{r,i+1}\right\| + t_f - t_0 \tag{6}$$

$$\mathbf{g}(\mathbf{x}_0, t_0, \mathbf{x}_f, t_f) = \begin{bmatrix} \mathbf{x}(t_0) - \mathbf{x}_{0,i} \\ t_0 - t_{0,i} \\ \hat{\mathbf{v}}(\mathbf{x}_f) - \hat{n}_{i+1} \end{bmatrix} = \mathbf{0} \tag{7}$$

$$\mathbf{h}(\mathbf{x}_f) = \begin{bmatrix} \left\|\mathbf{r}_f - \mathbf{r}_{r,i+1}\right\| - \min(S_{i+1}) \\ \left|(\mathbf{r}_f - \mathbf{r}_{r,i+1}) \cdot \hat{n}_{i+1}\right| - 0.1 \end{bmatrix} \leq \mathbf{0}$$

Where

- $\mathbf{r}_f$ is the position vector of the CubeSat at the end of the the course segment and $\mathbf{r}_{r,i}$ is the position vector of the center of the i'th ring. By extension, $\left\|\mathbf{r}_f - \mathbf{r}_{r,i+1}\right\|$ is the cubesat's distance to the center of the ring at the end of the course segment.
- S is a quadratic positive definite matrix encoding the shape of the next course ring in the sequence like so: $S = \begin{bmatrix} a & 0 \\ 0 & b \end{bmatrix}$, where a is the semimajor axis and b is the semiminor axis.
- $t_{0,i}$ is the time when the satellite passed through the start ring of the current course segment (t_f of the previous course segment).
- $\mathbf{x}_{0,i}$ is the initial state of the CubeSat at the start ring of the current course segment ($\mathbf{x}_f$ of the previous course segment).
- $\hat{\mathbf{v}}(\mathbf{x}_f)$ is the velocity unit vector of the final state for the current course segment.
- $\hat{n}_{i+1}$ is the normal vector of the next course ring in the sequence.

With the problem defined, let's go about solving it!

II. Necessary Conditions for CubeSat Racing

The tool I used to solve this problem was primer vector theory. While indirect optimal control methods are notoriously non-intuitive, I find primer vector theory to be the most intuitive of them. It directly ties the velocity adjoints of our extended state to the control effort expended, which was very helpful in verifying that my code works!

A. Hamiltonian Definition and Optimal Control Law

Let's start by defining the Hamiltonian for this problem:

$$H(\mathbf{x}, \mathbf{p}, \mathbf{u}, t) = L(\mathbf{x}, \mathbf{u}, t) + \mathbf{p} \cdot \mathbf{f}(\mathbf{x}, \mathbf{u}, t) \tag{8}$$

Since our problem doesn't include an integral term, the Lagrangian L in Equation 8 doesn't appear in our Hamiltonian. Applying our dynamics from Equation 3 results in the following Hamiltonian for our system:

$$\begin{aligned} H(\mathbf{x}, \mathbf{p}, \mathbf{u}, t) &= \mathbf{p} \cdot \mathbf{f}(\mathbf{x}, \mathbf{u}, t) \\ &= \mathbf{p}_r \cdot \mathbf{v} + \mathbf{p}_v \cdot \mathbf{a}_{\text{grav}} + \mathbf{p}_v \cdot \mathbf{u} \end{aligned} \tag{9}$$

Where $\mathbf{p}_r$ represents our position adjoints, $\mathbf{p}_v$ represents our velocity adjoints, and $\mathbf{a}_{\text{grav}} = \begin{bmatrix} 2n\dot{y} + 3n^2x \\ -2n\dot{x} \\ -n^2z \end{bmatrix}$ from the CHW equations, i.e. the acceleration from gravity in the Hill Frame.

Then, the Pontryagin Minimum Principle states that the optimal control to apply for any system is the one such that $\mathbf{u}^*$ minimizes the Hamiltonian of the system [2]. Mathematically, this means we want to choose $\mathbf{u}^*$ such that

$$\left.\frac{\partial H}{\partial \mathbf{u}}\right|_{\mathbf{u}^*} = \mathbf{0} \text{ if } \mathbf{u} \text{ is in the interior of U} \tag{10}$$

or

$$\mathbf{u}^*(\mathbf{x}, \mathbf{p}, t) = \arg\min_{\mathbf{u}} H(\mathbf{x}, \mathbf{p}, \mathbf{u}, t) \text{ if } \mathbf{u} \text{ is in the boundary of U} \tag{11}$$

Here, U is the set of all possible controls $\mathbf{u}$ such that $||\mathbf{u}|| \leq ||\mathbf{u}_{\max}||$. For our problem, Equation 10 would result in an expression without $\mathbf{u}$, which we would need to solve for. Thus, we must use Equation 11, implying that $\mathbf{u}$ is on the boundary of U, i.e. always at maximum magnitude.

Following the minimum principle, we must solve the following problem:

$$\begin{aligned} \mathbf{u}^*(\mathbf{x}, \mathbf{p}, t) &= \arg\min_{\mathbf{u}} H(\mathbf{x}, \mathbf{p}, \mathbf{u}, t) \\ &= \arg\min_{\mathbf{u}} \mathbf{p}_r \cdot \mathbf{v} + \mathbf{p}_v \cdot \mathbf{a}_{\text{grav}} + \mathbf{p}_v \cdot \mathbf{u} \end{aligned} \tag{12}$$

This is where primer vector theory comes in. Equation 12 shows that the only way $\mathbf{u}$ affects the Hamiltonian is by acting on $\mathbf{p}_v$, which we refer to as the primer vector. Thus, in order to minimize the Hamiltonian, we want to choose $\mathbf{u}$ such that it acts in the opposite direction as $\mathbf{p}_v$ at maximum magnitude, i.e.

$$\mathbf{u}^* = -||\mathbf{u}_{\max}||\hat{\mathbf{p}}_v \tag{13}$$

Then, substituting $\mathbf{u}^*$ into Equation 9 and recalling that $\mathbf{p}_v = ||\mathbf{p}_v||\hat{\mathbf{p}}_v$ and $\hat{\mathbf{p}}_v \cdot \hat{\mathbf{p}}_v = \hat{\mathbf{p}}_v^\intercal \hat{\mathbf{p}}_v = 1$, the optimal Hamiltonian becomes

$$
\begin{aligned}
H^*(\mathbf{x}, \mathbf{p}, t) &= H(\mathbf{x}, \mathbf{p}, \mathbf{u}^*(\mathbf{x}, \mathbf{p}, t), t) \\
&= \mathbf{p}_r \cdot \mathbf{v} + \mathbf{p}_v \cdot \mathbf{a}_{\text{grav}} + \mathbf{p}_v \cdot \mathbf{u}^* \\
&= \mathbf{p}_r \cdot \mathbf{v} + \mathbf{p}_v \cdot \mathbf{a}_{\text{grav}} + \mathbf{p}_v \cdot (-||\mathbf{u}_{\max}||\hat{\mathbf{p}}_v) \\
&= \mathbf{p}_r \cdot \mathbf{v} + \mathbf{p}_v \cdot \mathbf{a}_{\text{grav}} - ||\mathbf{p}_v|| \cdot ||\mathbf{u}_{\max}||
\end{aligned} \tag{14}
$$

We can verify that this Hamiltonian is correct by recovering our equations of motion like so, recalling that $\frac{\partial ||\mathbf{p}_v||}{\partial \mathbf{p}_v} = \hat{\mathbf{p}}_v$:

$$
\begin{aligned}
\dot{\mathbf{x}} = \frac{\partial H^*}{\partial \mathbf{p}} &= \begin{bmatrix} \frac{\partial H}{\partial \mathbf{p}_r} \\ \frac{\partial H}{\partial \mathbf{p}_v} \end{bmatrix} \\
&= \begin{bmatrix} \mathbf{v} \\ \mathbf{a}_{\text{grav}} - ||\mathbf{u}_{\max}||\hat{\mathbf{p}}_v \end{bmatrix}
\end{aligned} \tag{15}
$$

Equation 15 exactly matches Equation 3 with the substitution that $\mathbf{u} = -||\mathbf{u}_{\max}||\hat{\mathbf{p}}_v$, as expected.

We can further extract the adjoint dynamics like so:

$$
\begin{aligned}
\dot{\mathbf{p}} = -\frac{\partial H^*}{\partial \mathbf{x}} &= \begin{bmatrix} -\frac{\partial H}{\partial \mathbf{r}} \\ -\frac{\partial H}{\partial \mathbf{v}} \end{bmatrix} \\
&= \begin{bmatrix} -\left(\frac{\partial \mathbf{a}_{\text{grav}}}{\partial \mathbf{r}}\right)^{\mathsf{T}} \mathbf{p}_v \\ -\mathbf{p}_r - \left(\frac{\partial \mathbf{a}_{\text{grav}}}{\partial \mathbf{v}}\right)^{\mathsf{T}} \mathbf{p}_v \end{bmatrix}
\end{aligned} \tag{16}
$$

Then, Equation 16 can be rewritten in matrix form like so:

$$
\dot{\mathbf{p}} = \begin{bmatrix} \mathbf{0}_{3\text{x}3} & -\left(\frac{\partial \mathbf{a}_{\text{grav}}}{\partial \mathbf{r}}\right)^{\mathsf{T}} \\ -\mathbf{I}_{3\text{x}3} & -\left(\frac{\partial \mathbf{a}_{\text{grav}}}{\partial \mathbf{v}}\right)^{\mathsf{T}} \end{bmatrix} \mathbf{p} \tag{17}
$$

Equation 17 is a linear, time invariant ODE. Thus, we know its solution is $\mathbf{p}(t) = e^{A(t-t_0)}\mathbf{p}_0 = \Phi_{\mathbf{p}}(t, t_0)\mathbf{p}_0$, where A is the matrix in Equation 17. Taking the matrix exponential and recognizing the trigonometric Taylor Series patterns in the resulting matrix, we get the following for the Adjoint STM (State Transition Matrix):

$$
\Phi_{\mathbf{p}}(t, t_0) = \begin{bmatrix}
4 - 3\,\text{c}(n\Delta t) & 6(n\Delta t - \text{s}(n\Delta t)) & 0 & -3n\,\text{s}(n\Delta t) & 6n(\text{c}(n\Delta t) - 1) & 0 \\
0 & 1 & 0 & 0 & 0 & 0 \\
0 & 0 & \text{c}(n\Delta t) & 0 & 0 & n\,\text{s}(n\Delta t) \\
-\frac{1}{n}\text{s}(n\Delta t) & \frac{2}{n}(\text{c}(n\Delta t) - 1) & 0 & \text{c}(n\Delta t) & 2\,\text{s}(n\Delta t) & 0 \\
\frac{2}{n}(1 - \text{c}(n\Delta t)) & -\frac{1}{n}\text{s}(n\Delta t) & 0 & -2\,\text{s}(n\Delta t) & 4\,\text{c}(n\Delta t) - 3 & 0 \\
0 & 0 & -\frac{1}{n}\text{s}(n\Delta t) & 0 & 0 & \text{c}(n\Delta t)
\end{bmatrix} = \begin{bmatrix} \phi_{11} & \phi_{12} \\ \phi_{21} & \phi_{22} \end{bmatrix}
\tag{18}
$$

where c(·) represents cos(·), s(·) represents sin(·), Δt represents $t - t_0$, and ϕ_{ij} represents a 3x3 block of the overall STM. Thus, we know that the adjoint dynamics are smooth and differentiable along each course segment, which is important for our choice of solver later.

B. Transversality Conditions

Next, we need to put constraints on the initial and final values of our adjoints and Hamiltonian in order to create feasible trajectories between course rings. To do so, we leverage the Transversality Conditions, which are listed below:

$$\begin{aligned}
\mathbf{p}_0 &= -\frac{\partial K}{\partial \mathbf{x}_0} - \lambda \cdot \frac{\partial \mathbf{g}}{\partial \mathbf{x}_0} \\
H_0 &= \frac{\partial K}{\partial t_0} + \lambda \cdot \frac{\partial \mathbf{g}}{\partial t_0} \\
\mathbf{p}_f &= \frac{\partial K}{\partial \mathbf{x}_f} + \lambda \cdot \frac{\partial \mathbf{g}}{\partial \mathbf{x}_f} \\
H_f &= -\frac{\partial K}{\partial t_f} - \lambda \cdot \frac{\partial \mathbf{g}}{\partial t_f}
\end{aligned} \tag{19}$$

Where λ is a vector of Lagrange multipliers associated with the manifold constraints specified in Equations 5 and 7 and K is the cost function specified in Equations 4 and 6. Note, when solving the optimal control problem between rings, we treat the inequality manifold constraints $\mathbf{h}$ as additional components of the equality constraints $\mathbf{g}$. If any of the inequality constraints are evaluated to be negative, then the corresponding entry in the λ vector is set to 0, as that constraint was not needed and can be considered inactive.

1. Initial Adjoint Transversality

Solving the Transversality Conditions for $\mathbf{p}_0$ resulted in the following constraint on $\mathbf{p}_0$:

$$\mathbf{p}_0 = -\lambda_0 \tag{20}$$

Where λ_0 is a 6x1 column vector. Thus, $\mathbf{p}_0$ is essentially arbitrary, and must be guessed or solved for at the start of each course segment along with t_f.

2. Initial Hamiltonian Transversality

Solving the Transversality Conditions for H_0 resulted in the following constraint on H_0:

$$H_0 = -1 + \lambda_t \tag{21}$$

Where λ_t is a scalar. Further, we know that

$$H_0 = \mathbf{p}_0 \cdot \mathbf{f}(\mathbf{x}_0, \mathbf{u}_0, t_0) = \mathbf{p}_{r_0}^\intercal \mathbf{v}_0 + \mathbf{p}_{v_0}^\intercal \mathbf{a}_{\text{grav},0} - ||\mathbf{p}_{v_0}|| \cdot ||\mathbf{u}_{\text{max},0}||$$

Which ties together the constraint on $\mathbf{p}_0$ to the new constraint on H_0.

3. Final Adjoint Transversality

Solving the Transversality Conditions for $\mathbf{p}_f$ resulted in the following constraint on $\mathbf{p}_f$:

$$\mathbf{p}_f = \begin{bmatrix} (1+\lambda_{r_f})\frac{\mathbf{r}_f - \mathbf{r}_{r,i+1}}{||\mathbf{r}_f - \mathbf{r}_{r,i+1}||} + \lambda_n \hat{n}_{i+1} \\ \frac{1}{||\mathbf{v}_f||}(\mathbf{I}_{3\text{x}3} - \hat{\mathbf{v}}_f \hat{\mathbf{v}}_f^\intercal)\lambda_v \end{bmatrix} \tag{22}$$

Where λ_{r_f} and λ_n are scalars and λ_v is a 3x1 column vector. Further, we know from Equations 17 and 18 that

$$\mathbf{p}_f = \Phi_{\mathbf{p}}(t_f, t_0)\mathbf{p}_0$$

Which ties together the constraint on $\mathbf{p}_0$ to the new constraint on $\mathbf{p}_f$.

4. Final Hamiltonian Transversality

Solving the Transversality Conditions for H_f resulted in the following constraint on H_f:

$$H_f = -1 \tag{23}$$

Further, we know that

$$H_f = \mathbf{p}_f \cdot \mathbf{f}(\mathbf{x}_f, \mathbf{u}_f, t_f) = \mathbf{p}_{r_f}^\intercal \mathbf{v}_f + \mathbf{p}_{v_f}^\intercal \mathbf{a}_{\text{grav},f} - ||\mathbf{p}_{v_f}|| \cdot ||\mathbf{u}_{\text{max,f}}||$$

Which ties together the constraint on $\mathbf{p}_f$, and hence the constraint on $\mathbf{p}_0$, to the new constraint on H_f.

5. Other Constraints and Observations

Since our dynamics $\mathbf{f}$ in Equation 3 are time invariant and our cost function doesn't include a time varying integral term, we know that the Hamiltonian for this system is constant. Thus,

$$\begin{aligned} H_0 &= H_f \\ -1 + \lambda_t = -1 &\implies \lambda_t = 0 \end{aligned}$$

This restricts $\mathbf{p}_0$ further, which is the basis of all the other constraints.

III. Implementing in MATLAB

With all of the dynamics modeled and constraints defined, it's time to implement the project in MATLAB and get some results!

A. Solver Selection

After solving the Transversality Conditions, we were able to narrow down the variables we need to guess to just $\mathbf{p}_0$ and t_f, or 7 numbers at the start of each course sequence. Further, our dynamics and constraints are continuous and smooth, meaning a gradient descent solver should do the trick nicely. There are many options to choose to solve this problem, such as IPOPT, SNOPT, and others, but for simplicity and understanding I wanted something that I had full control over. For this reason, I chose to go with `fmincon` in MATLAB, which allows me to have full control over the entire process vs. using what is very nearly a blackbox. Further, it was the most convenient thing for me to use, as I have an active MATLAB license and don't need to download and setup any special software packages and/or configurations to get started.

B. fmincon Implementation

`fmincon` is a pseudonym for "Function MINimizer with CONstraints," and it has a very simple MATLAB interface. To use `fmincon`, I just need to specify an objective function, a vector of parameters to achieve the minimizing, and a suite of constraint functions/settings that constrain the problem, in addition to some wrapper function to set everything up. For my problem, all of the constraints I provided were nonlinear, and my vector of parameters consisted of p_0, the initial adjoints, and t_f, the final time of the course segment. All of my code related to setting up and using `fmincon` can be found in Appendix C. The wrapper function is titled "solveTrajectory.m", the objective function is titled "costFunction.m", and the constraints function is titled "constraints.m". The main script to run the project is titled "FinalProjectMain.m". There is also a suite of utility and plotting functions, but they aren't important for the operation of `fmincon` and have not been included in the appendix. A zip file of all code has also been included in the project submission if more investigation is desired. If reading this report not in the context of a project submission, email ian.m.faber@gmail.com and I'll gladly send over the code!

IV. Results and Discussion

With the solver chosen and code written, I investigated two main scenarios to see how the concept of CubeSat racing could go and what CubeSat properties affect race course performance.

A. Scenario 1: Different Thrust Capabilities at Random Start Points with a Standard Race Course Layout

The first scenario I investigated utilizes 4 CubeSats with different thrust capabilities and random starting positions at the beginning of a course that I tailored to test different aspects of the underlying system dynamics. The four CubeSats of Scenario 1 are summarized in Table 1, and the race course is shown in Figure 1:

Table 1 Scenario 1 CubeSat Configurations

Name	Maximum Control Authority [m/s^2]	Control Type	Initial XYZ Position [m]
Eeny	0.1	Unrestricted	[-642.9, -480.5, 547.4]
Meeny	0.25	Unrestricted	[-641.4, -480.5, 534.9]
Miny	0.5	Unrestricted	[-651.1, -480.5, 532.2]
Mo	[0.1667, 0.1667, 0.1667]	Axially restricted	[-646.5, -480.5, 538.0]

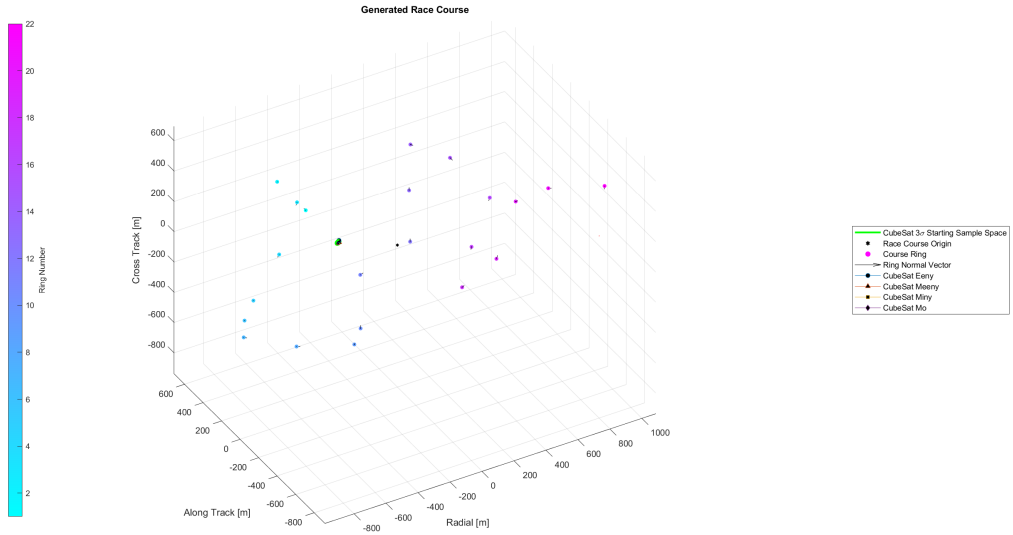

Fig. 1 Scenario 1 Race Course

Most of the CubeSats have unrestricted control authority, meaning that they can apply their full magnitude of maximum thrust along the primer vector at any time. However, I also wanted to investigate what an axial thrust restriction would look like. Thus, CubeSat Mo has a thrust restriction such that it can only thrust in the direction of the primer vector if the primer vector has a component in any of the inertial XYZ directions greater than some deadzone. I thought of this as a CubeSat that couldn't control its attitude, but could gimbal thrusters within some angular range of the CubeSat's body axes. While I didn't model thruster gimballing explicitly, I added a "deadzone" of 1 around each axis in an attempt to approximate such a gimbal restriction. As we'll see later, this results in some interesting control behavior, even if it isn't completely physically accurate. Mathematically, Mo's thrust restriction looks as follows in Equation 24:

$$
\begin{aligned}
||\mathbf{p}_v \cdot \hat{x}|| > 1 &\implies ||\mathbf{u}^* \cdot \hat{x}|| = \mathbf{u}_{\max,x}, \text{ i.e. } \mathbf{u} \text{ has a maximum x component. Otherwise, } \mathbf{u}^* \cdot \hat{x} = 0. \\
||\mathbf{p}_v \cdot \hat{y}|| > 1 &\implies ||\mathbf{u}^* \cdot \hat{y}|| = \mathbf{u}_{\max,y}, \text{ i.e. } \mathbf{u} \text{ has a maximum y component. Otherwise, } \mathbf{u}^* \cdot \hat{y} = 0. \\
||\mathbf{p}_v \cdot \hat{z}|| > 1 &\implies ||\mathbf{u}^* \cdot \hat{z}|| = \mathbf{u}_{\max,z}, \text{ i.e. } \mathbf{u} \text{ has a maximum z component. Otherwise, } \mathbf{u}^* \cdot \hat{z} = 0.
\end{aligned} \tag{24}
$$

With the CubeSats defined and the race course generated, I ran `fmincon` with each CubeSat to solve the course. The overall finished course can be seen in Figure 2, and results are summarized in Table 2.

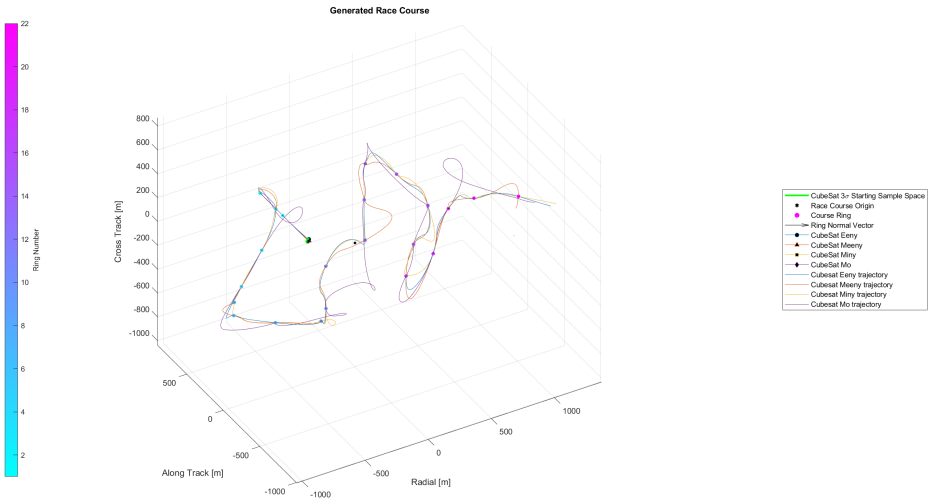

Fig. 2 Scenario 1 Completed Race Course

Table 2 Scenario 1 Results

CubeSat	Final Overall Cost	Time Taken to Complete Course [sec]	Ring Accuracy [m]	Minimum Possible Cost [sec]
Eeny	2003.944	1911.847	92.097	1052.421
Meeny	1338.647	1312.432	26.215	703.454
Miny	916.067	874.768	41.299	477.352
Mo	2128.998	1435.305	693.692	600.442

Table 2 contains both expected and surprising results. As expected, CubeSats with higher thrust capabilities completed the race course with faster times. This also corresponds to a lower minimum possible cost, which is modeled as the time taken to traverse the straight line distance between rings at full throttle given the CubeSat's initial velocity at the start ring. This follows simple kinematics, where a higher constant acceleration necessarily corresponds to traversing a given distance in less time: $t_{\min} = \max\left(\frac{-v_0 \pm \sqrt{v_0^2 - 2\mathbf{u}_{\max}\Delta d}}{\mathbf{u}_{\max}}\right)$, where $\Delta d = d_0 - d$ is the distance between rings.

However, a somewhat unexpected result is that higher thrust doesn't necessarily equate to higher ring accuracy as well. Since our overall cost is the sum of the CubeSat's time taken to complete the course and its distance to the center of each ring, we can find the CubeSat's overall ring accuracy by subtracting the time taken to complete the course from the overall cost. Doing so, we find that CubeSat Meeny, which has an intermediate thrust capability of 0.25 m/s^2, was the most accurate of all cubesats. This implies that there is some optimal thrust capability that balances the time taken to traverse a race course and ring accuracy.

Finally, we can see that Mo's ring accuracy suffered greatly as a result of its thrust restriction. Despite having a larger thrust capability on average, Mo takes longer than Meeny to complete the course, and egregiously misses some rings - ballooning its ring accuracy score. This makes sense, as there are situations where Mo needs control authority in a specific axis to pass through a ring, but with the restriction it is simply unable to make it. This implies that having attitude control, in addition to translation control, is critical to the CubeSat racing problem.

Figures 3 through 6 back up the results presented so far. In particular, we can see that all CubeSats generally do a good job at following the course - indicated by the red dots on the position axes - but generally struggle in meeting the velocity constraint - indicated by the red dots on the velocity axes. Further, we can see that the applied control varies wildly across each axis, but the primer vector always follows a smooth path over each course segment. Finally, we can see when Mo's thrust restriction kicks in over the course of the race, sometimes forcing its available thrust to 0 - rendering it helpless to affect its trajectory.

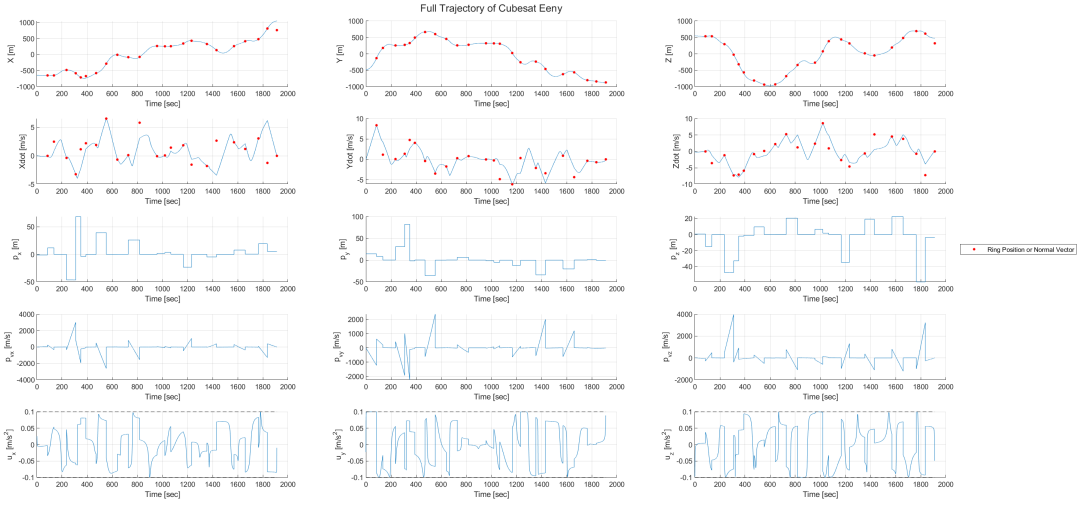

Fig. 3 CubeSat Eeny Scenario 1 Trajectory

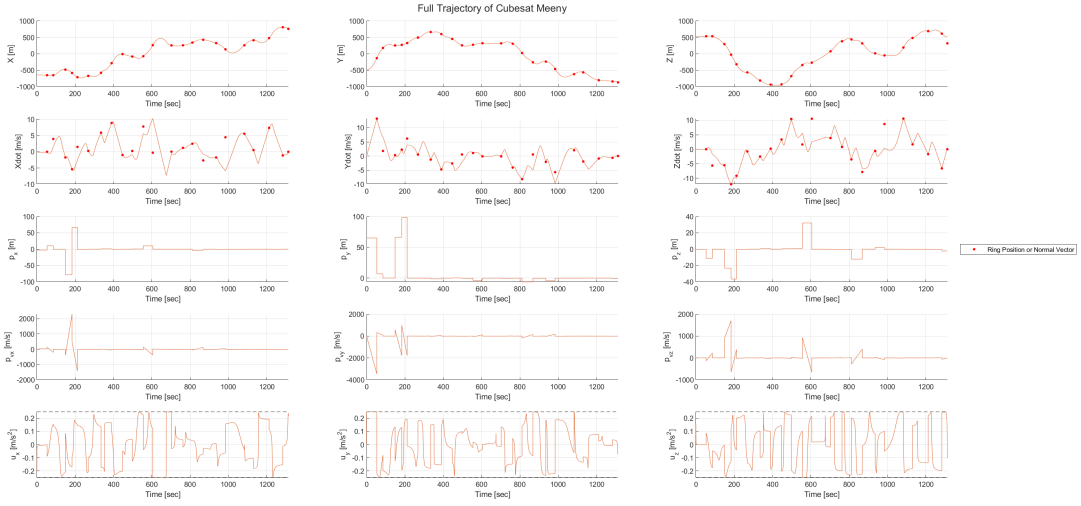

Fig. 4 CubeSat Meeny Scenario 1 Trajectory

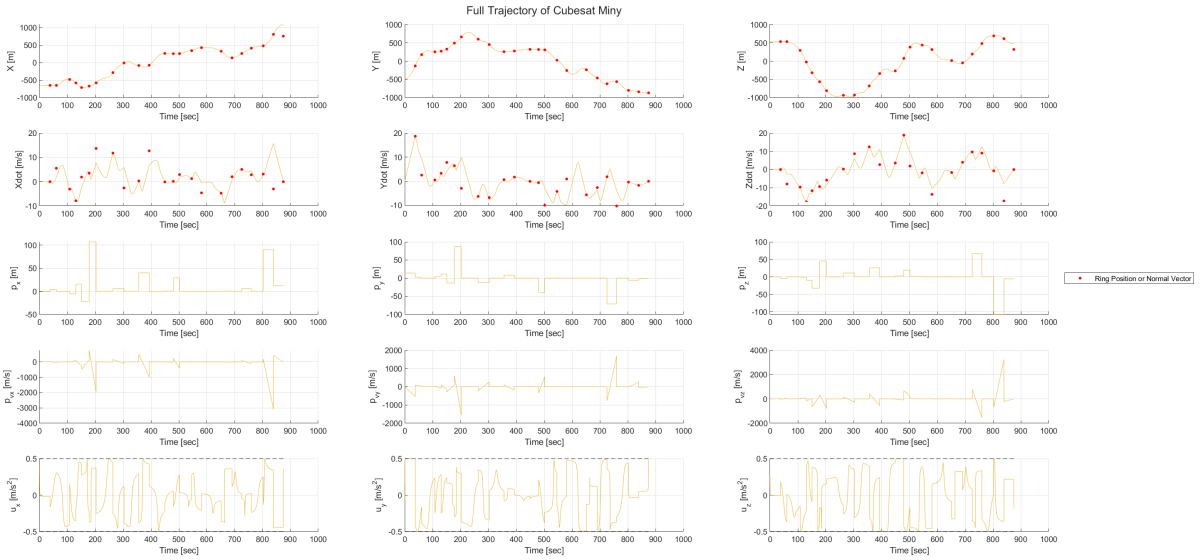

Fig. 5 CubeSat Miny Scenario 1 Trajectory

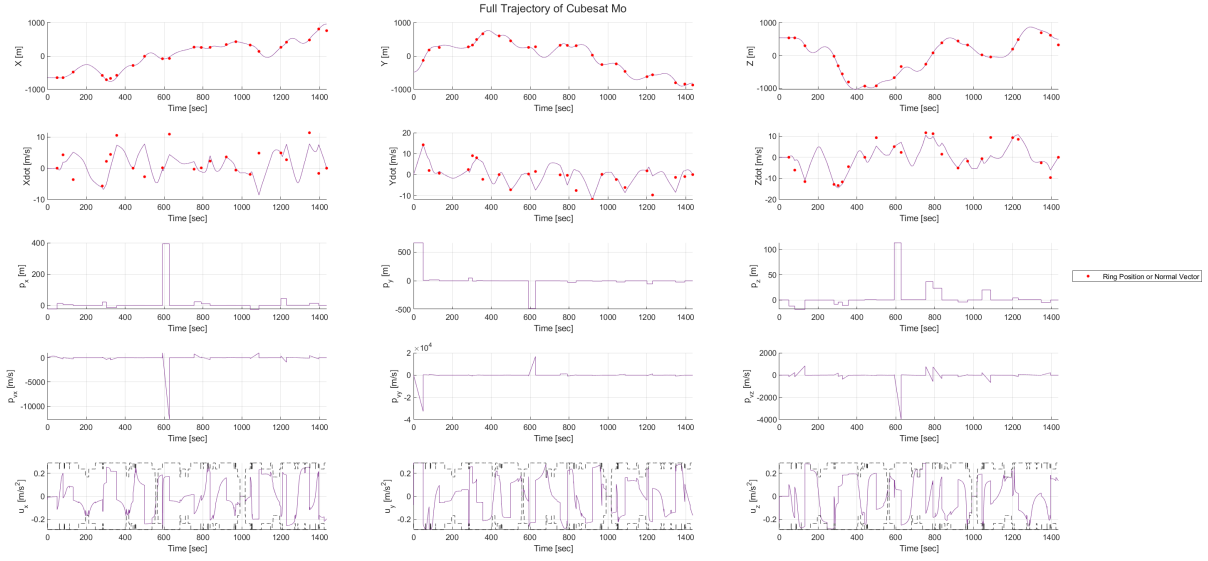

Fig. 6 CubeSat Mo Scenario 1 Trajectory

B. Scenario 2: Identical Thrust Capabilities at Random Start Points with a Randomized Race Course Layout

The next scenario I investigated utilizes 4 CubeSats with identical thrust capabilities and random starting positions at the beginning of a fully randomized race course. Scenario 1 is based on what I've dubbed the "dev" race course, i.e. the race course in which I debugged `fmincon` and got all of the underlying dynamics working. Scenario 2 is much closer to how I'd expect recreational CubeSat racing to look like, i.e. standardized CubeSats all competing on the same course with different starting positions (similar to NASCAR for modern racing!). That being said, The four CubeSats of Scenario 2 are summarized in Table 3, and the race course is shown in Figure 7:

Table 3 Scenario 2 CubeSat Configurations

Name	Maximum Control Authority [m/s^2]	Control Type	Initial XYZ Position [m]
Eeny	0.5	Unrestricted	[-298.3, -1696, 362.6]
Meeny	0.5	Unrestricted	[-299.9, -1696, 370.4]
Miny	0.5	Unrestricted	[-293.4, -1696, 360.8]
Mo	0.5	Unrestricted	[-299.4, -1696, 365.1]

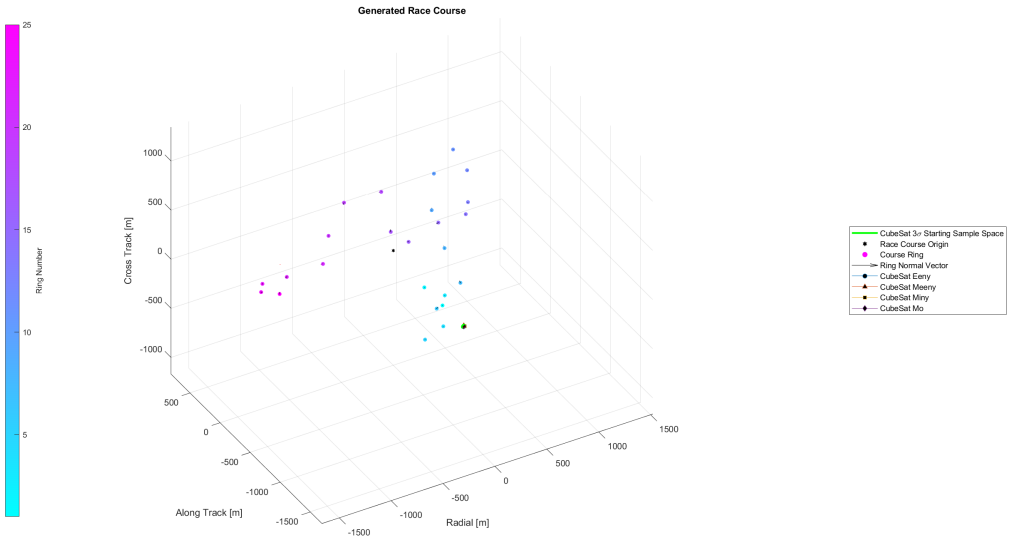

Fig. 7 Scenario 2 Race Course

This course looks quite interesting compared to the dev course. Unlike the dev course, the course for Scenario 2 meanders around and loops back on itself, compared to the dev course which more or less follows a predictable winding pattern. After defining the CubeSats and generating the course, I once again ran `fmincon` on each CubeSat to solve the course. The overall finished course can be seen in Figure 8, and the results are summarized in Table 4.

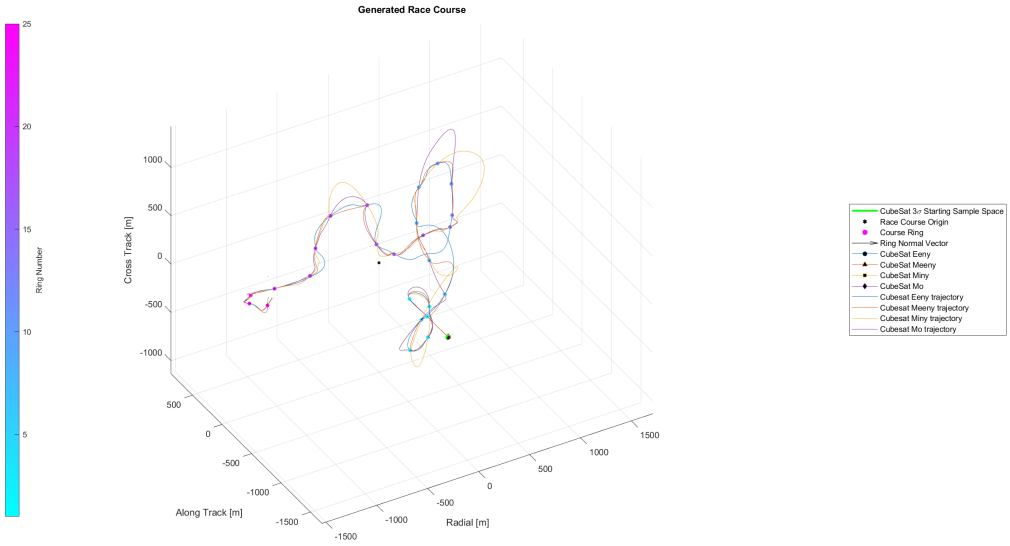

Fig. 8 Scenario 2 Completed Race Course

Table 4 Scenario 2 Results

CubeSat	Final Overall Cost	Time Taken to Complete Course [sec]	Ring Accuracy [m]	Minimum Possible Cost [sec]
Eeny	1574.212	965.223	608.989	526.207
Meeny	1133.655	1085.772	47.883	558.876
Miny	1761.798	1096.043	665.755	555.450
Mo	1161.586	1018.303	143.283	516.700

The results from Scenario 2 are much more varied than I thought they would be initially. Despite the CubeSats all having the same available thrust capabilities and completing the course in roughly the same order of magnitude of time, their ring accuracies vary drastically. Thus, despite CubeSat Eeny finishing the race course first, its ring accuracy caused it to lose out overall to Meeny, who finished the course second to last but had significantly better ring accuracy. This suggests that the initial random positions of the CubeSats have a massive impact on ring accuracy performance, so much so that it could be the difference between winning a race and coming in dead last. Further, we see that Mo had the potential to score best, as it has the lowest minimum possible cost. This implies that it was generally moving the fastest overall, but unfortunately couldn't compete with Meeny's superior ring accuracy. Scenario 2 provides an argument for the entertainment value of CubeSat racing at the very least: You never know who's going to win until all the scores are completely tallied up!

Figures 9 through 12 show the individual trajetories of each CubeSat over this course, and we see the same trends as in Scenario 1. Namely, that they generally do a good job following the course, but struggle to meet the velocity constraint when exiting each ring.

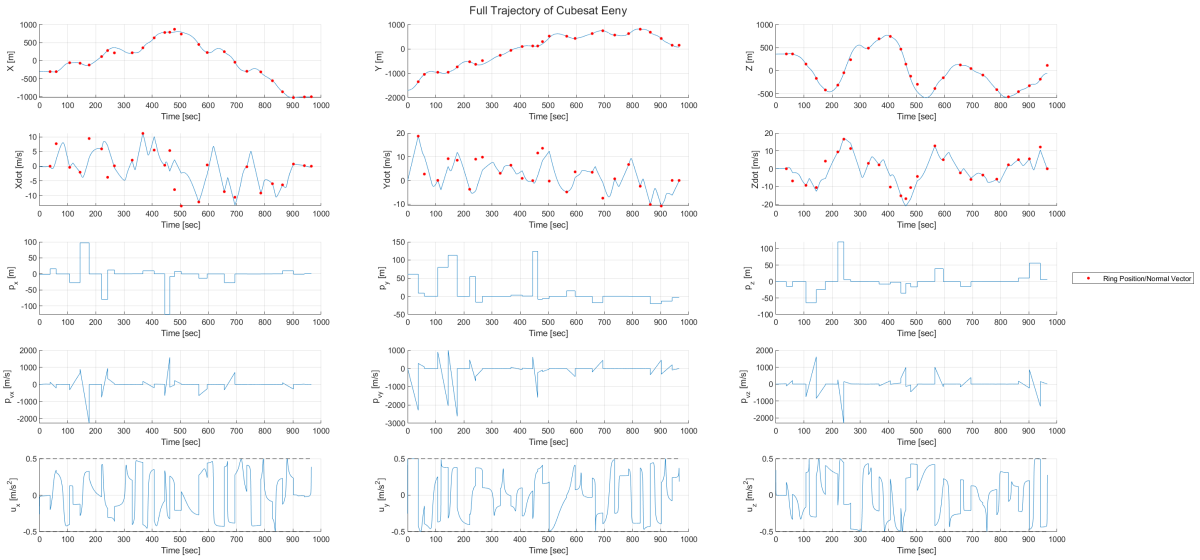

Fig. 9 CubeSat Eeny Scenario 2 Trajectory

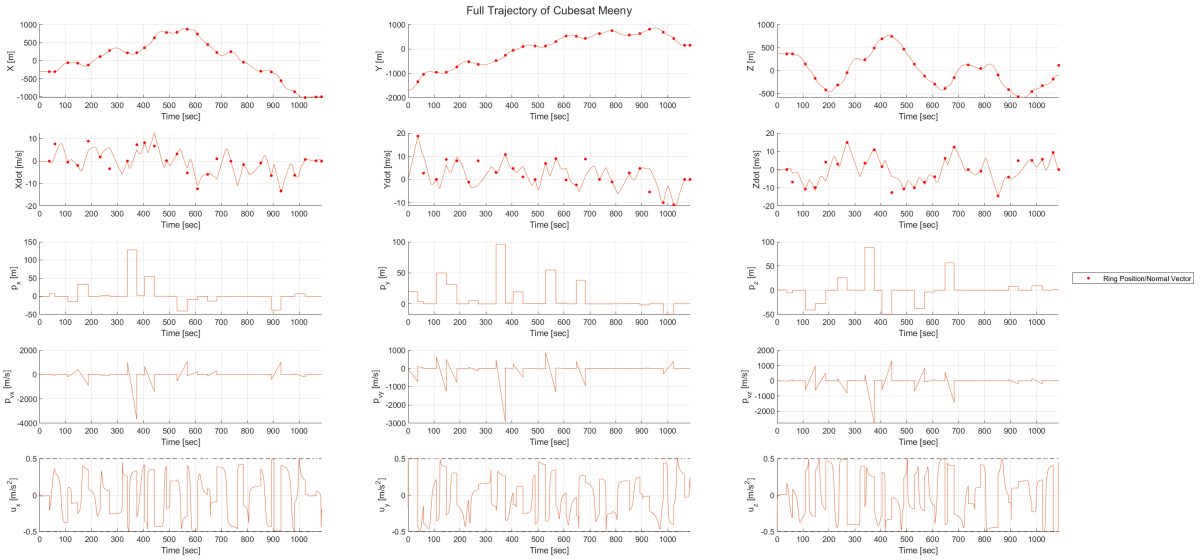

Fig. 10 CubeSat Meeny Scenario 2 Trajectory

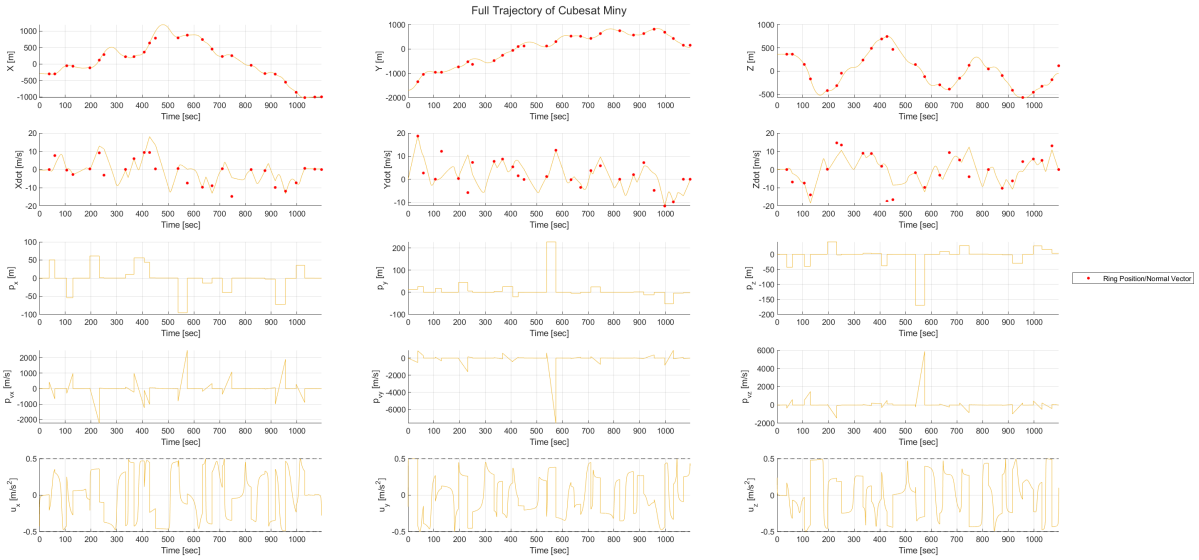

Fig. 11 CubeSat Miny Scenario 2 Trajectory

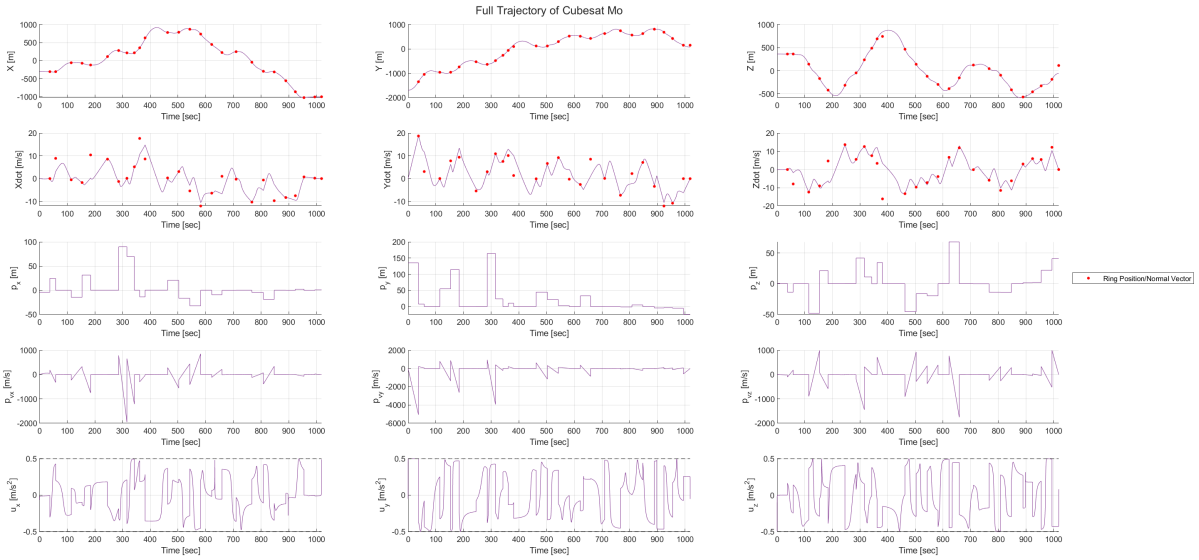

Fig. 12 CubeSat Mo Scenario 2 Trajectory

V. Conclusions and Future Work

After completing this project, I've been convinced that CubeSat racing is actually physically viable, given that CubeSat technology is advanced enough. The biggest liberty I took in this project is drastically inflating the thrust capabilities of modern CubeSats. According to [3], a CubeSat designer can buy thrusters that provide 10-500 μN of thrust for the CubeSat. Applied to a standard 1U 1 kg CubeSat and using our friend $F = ma$, this is only 0.01-0.5 mm/s^2 of thrust, or on the order of 1000x less than the 0.1-0.5 m/s^2 values I've used in this project. However, given the evolution of the CubeSat industry in recent years, it is very possible that advances in CubeSat thrust could be made to reach my numbers. Plus, if a society is at the point where they are recreationally racing CubeSats, then the CubeSat industry is likely in a good spot and R&D has probably come a long way.

As for my optimal control approach, I think there is still quite a bit of tuning left to be done. As we saw in both scenarios, the CubeSats generally struggle to meet the velocity pointing constraint. I believe this has to do with how I initialize the guess on $\mathbf{p}_0$ that I pass into `fmincon`. As discussed in Section IV.A, I have a relatively good initial guess for t_f, namely the straight line minimum time between rings. However, there is no easy, analytical way that I'm aware of to initialize the adjoints. My best attempt so far is to initialize them as the final adjoints of the previous ring segment, but as we see in all of the trajactory plots, the adjoints are generally discontinuous from ring segment to ring segment. Thus, while they are continuous during the segment and obey the STM we found in Equation 18, simply initializing them as the previous adjoints could be orders of magnitude off of where they need to start, not to mention potentially the wrong sign. This could also explain why my code takes so long to run, as it's possible that the initial adjoint guess is so bad that `fmincon` can't converge to a solution that satisfies all the constraints and it tries to do the best it can.

This leads to a related potential avenue of future work, which is expanding what makes up a ring segment. Right now, my code goes from ring segment to ring segment, each of which includes a single starting ring and a single end ring. This means that as far as `fmincon` is concerned, the race course is only made up of that ring segment and it won't respect the idea that ring segments need to flow together - hence the velocity pointing constraint. If there was a way to include multiple rings in each segment, then the velocity pointing constraint issue could be remedied by more closely mirroring the true nature of the race course in `fmincon` - that being a sequence of rings.

This project was incredibly fun to work on, and I enjoyed applying such a technical concept to a fun and lighthearted problem! I have ideas of making this into a full-fledged video game, but I have to graduate with my Master's degree first... One step at a time!

VI. Acknowledgements

I'd like to thank the following people for all of their help during this project:

1) Professor Scheeres for his infinite patience in answering all of my constraint and feasibility questions, as well as for approving this project in the first place!
2) Emily Matteson for our debugging sessions, complete with woes and successes!
3) Anirudh Etagi for encouraging me to stay strong in completing this project despite ASEN 6080 breathing down our necks at the same time!
4) Niko Natsoulas for bouncing ideas back and forth and for helping me proofread this report!

References

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

[2] Lawden, D. F., *Analytical Methods of Optimization*, 1st ed., Dover Publications Inc., 2006.

[3] CubeSatShop, "Cluster of IFM Nano Thruster for Smallsats," , 2025. URL `https://www.cubesatshop.com/product/cluster-ifm-nano-thruster-smallsats/`, last accessed 6 May 2025.

A. Appendix index

Use the appendix index below to navigate to the different appendices!

B. Scenario Animations

Back to appendix index: A.A

To access animations for each Scenario in this report, click the associated Google Drive link!

- Scenario 1: `https://drive.google.com/file/d/1YwVPg9fsCD9heEVu9C1HQ1WoUc_bBFpA/view?usp=sharing`
- Scenario 2: `https://drive.google.com/file/d/12x3zqbFPNTMp4vRGmrBfVbysQisEkVIL/view?usp=sharing`

Enjoy!

C. MATLAB Code Relevant to fmincon Implementation

Back to appendix index: A.A

Use the code index below to navigate to the relevant functions for my fmincon implementation!

B. Main Script

Back to index: C.A

```
%% ASEN 6020 Final Project Main Script
% By: Ian Faber

%% Housekeeping
clc; clear; close all;

%% Logistics
addpath(genpath(".\CubeSats"));
addpath(genpath(".\RaceCourse"));
addpath(genpath(".\Plotting"));

% %% Enable parallel computing
% parpool;

%% Ask user for input
menuMsg = sprintf("\n\t\tWelcome to the CubeSat racer! What would you like to
    do?\n\n" + ...
```

```
                    "\t1. Generate and run a new race course (time intensive, ~30 min to 1 hour!)\n" + ...
                    "\t2. Analyze an old run\n" + ...
                    "\t3. Animate an old run\n" + ...
                    "\t4. Exit\n\n" + ...
                    "\tChoice (enter number of option to do): ");
choice = input(menuMsg);

%% Decide what to do
switch choice
    case 1
        %% Setup Problem Pameters
        menuMsg = sprintf("\n\tRun dev course or generate new course?\n" + ...
                          "\t1. Run dev course (different thrust configs)\n" + ...
                          "\t2. Generate new course (same thrust configs)\n" + ...
                          "\tChoice (enter number of option to do): ");
        choice = input(menuMsg);

        if choice == 1
            randomCourse = false;
            fprintf("Running dev race course, this will take some time!\n")
        else
            randomCourse = true;
            fprintf("Generating and solving new race course, this will take some time!\n")
        end

            % Define ring semi-major and minor axis ranges
        semiMaj = [1, 5]; % m
        semiMin = [1, 5]; % m

            % Define inter-ring parameter ranges
        interRingDist = [300, 350]; % m
        azimuthAngle = deg2rad([-90, 90]); % deg -> rad
        elevationAngle = deg2rad([-90, 90]); % deg -> rad

            % Define number of rings range
        numRings = [15, 25];

            % Define course origin orbit
        T = 180*60; % 180 minute circular orbit
        n = 2*pi/T;

            % Define course parameters structure
        courseParams = struct("semiMaj", semiMaj, "semiMin", semiMin, "dist", interRingDist, ...
                              "azAng", azimuthAngle, "elAng", elevationAngle, "numRings", numRings, ...
                              "n", n);

            % ODE45 options
        opt = odeset('AbsTol', 1e-12, 'RelTol', 1e-12);
```

```
65
66          %% Generate race course
67              % Set rng seed for race course generation
68          if ~randomCourse
69              seed = 2*69420; % cool options: 0, 2, 3, 4
70              rng(seed);
71          else
72              rng("shuffle");
73          end
74
75              % Make course
76          rings = generateRaceCourse(courseParams);
77
78              % Define CubeSat starting ring as the maximum distance behind the
                    start
79              % ring at a 3 sigma covariance of 5x the largest semi-major/minor
                    axis
80          startRing = generateRing(50*max(semiMaj), 50*max(semiMin), 0, 0, -max(
                interRingDist), rings(1));
81          rings(1).params.lastRing = startRing;
82
83              % Define CubeSat ending ring as the minimum distance after the
                    last
84              % ring at a covariance corresponding to the smallest possible ring
85          endRing = generateRing(min(semiMaj),min(semiMin),0,0,min(interRingDist
                ),rings(end));
86          rings = [rings; endRing];
87
88              % Move race course so origin is the mean of all ring centers
89          centers = [];
90          for k = 1:length(rings)
91              centers = [centers, rings.center];
92          end
93
94          newOrigin = mean(centers, 2);
95
96          startRing.center = startRing.center - newOrigin;
97          startRing.params.lastRing.center = startRing.params.lastRing.center -
                newOrigin;
98          for k = 1:length(rings)
99              rings(k).center = rings(k).center - newOrigin;
100             rings(k).params.lastRing.center = rings(k).params.lastRing.center
                    - newOrigin;
101         end
102         endRing.center = endRing.center - newOrigin;
103         endRing.params.lastRing.center = endRing.params.lastRing.center -
                newOrigin;
104
105         %% Generate CubeSats
106             % Max thrust based on
107             % https://www.cubesatshop.com/product/cluster-ifm-nano-thruster-
                    smallsats/
108             % - 10-500 uN. Boosted by 1e3 for this problem to make things
109             % more interesting!
```

```
numSats = 4;
names = ["Eeny", "Meeny", "Miny", "Mo"]; % Names :P Mo has attitude control system problems, and can only use x-y-z thrusters!
markers = ["o", "^", "square", "diamond"]; % Markers for plotting
colors = ["#0072BD", "#D95319", "#EDB120", "#7E2F8E"]; % Colors for plotting
if ~randomCourse
    thrusts = [100, 250, 500, 500]*(10^-3); % Maximum thrust values
    thrustConfigs = [false, false, false, true]; % Whether thrust is split amongst x-y-z thrusters or a general direction
else
    thrusts = 500*10^-3*ones(1,4); % Make cubesats with max thrust
    thrustConfigs = false(1,4); % No thrust configuration shenanigans
end

    % Make CubeSats
cubesats = [];
for k = 1:numSats
    cubesats = [cubesats; generateCubesat(thrusts(k), thrustConfigs(k), startRing, names(k), markers(k), colors(k))];
end

%% Show off course and cubesat starting positions
figNum = 420;
titleText = sprintf("Generated Race Course");
xLabel = sprintf("Radial [m]"); yLabel = sprintf("Along Track [m]"); zLabel = sprintf("Cross Track [m]");
courseFig = plotCourse(startRing, rings, endRing, cubesats, figNum, titleText, xLabel, yLabel, zLabel);

for k = 0:180
    view(-30 + 2*k, 35);
    drawnow
end

%% Run the race course!
debug = [false; false; false]; % iteration plotting+display; trajectory plotting; disable progress sequence
for k = 1:length(cubesats)
    fprintf("\nCubesat %s is starting the race course!\n", cubesats(k).name)
    for kk = 1:length(rings) % Loop through each course segment
        % Determine which cost function and constraints to
        % apply
        if kk == length(rings) % Final problem, come to a stop in minimum time
            isFinal = true;
        else % Initial/intermediate problem, aim for rings
            isFinal = false;
        end

        % Solve trajectory according to problem
        [t,X,u,optParams,initGuess,cost] = solveTrajectory(cubesats(k), rings(kk), courseParams, opt, isFinal, debug);
```

```
154                 cubesats(k).X = [cubesats(k).X; X];
155                 cubesats(k).t = [cubesats(k).t; t];
156                 cubesats(k).u = [cubesats(k).u; u];
157                 cubesats(k).optParams = [cubesats(k).optParams, optParams]; %
                        Parameters solved by fmincon
158                 cubesats(k).initGuess = [cubesats(k).initGuess, initGuess]; %
                        Initial guess for parameters
159                 cubesats(k).ringSeg = [cubesats(k).ringSeg, [kk-1; kk]]; %
                        Ring segment of this trajectory (from kk-1 to kk)
160                 cubesats(k).tSeg = [cubesats(k).tSeg, [t(1); t(end)]]; % Time
                        this ring segment started and ended
161                 cubesats(k).cost = [cubesats(k).cost, cost]; % Cost for this
                        ring segment
162
163                     % Update t0 and X0
164                 cubesats(k).X0 = X(end,1:6)';
165                 cubesats(k).t0 = t(end);
166
167                     % Plot segment
168                 titleText = sprintf("Cubesat %s trajectory segment: Ring %.0f
                        to %.0f", cubesats(k).name, kk-1, kk);
169                 xLabel = "Radial [m]"; yLabel = "Along Track [m]"; zLabel = "
                        Cross Track [m]";
170                 if ~debug(1)
171                     figNum = kk + (k-1)*length(rings);
172                 else
173                     figNum = kk + (k-1)*length(rings) + 1;
174                 end
175                 plotSegment(cubesats(k), rings(kk), t, X, u, figNum, titleText
                        , xLabel, yLabel, zLabel);
176
177                     % Report progress
178                 fprintf("\tCubesat %s passed through ring %.0f in %.3f sec!\n
                        ", cubesats(k).name, kk, t(end) - t(1));
179             end
180             fprintf("\n\tCubesat %s finished the course in %.3f sec!\n",
                    cubesats(k).name, cubesats(k).t(end)-cubesats(k).t(1));
181
182                 % Add trajectory to race plot
183             figure(420)
184             plot3(cubesats(k).X(:,1), cubesats(k).X(:,2), cubesats(k).X(:,3),
                    '-', 'Color', cubesats(k).color, 'DisplayName', sprintf("
                    Cubesat %s trajectory", cubesats(k).name));
185
186             figure; tl = tiledlayout(1,2);
187             title(tl, sprintf("Cubesat %s Adjoints", cubesats(k).name))
188             nexttile
189                 title(sprintf("Cubesat %s Position Adjoints", cubesats(k).name
                        ))
190                 hold on; grid on; axis equal;
191                 plot3(cubesats(k).X(:,7), cubesats(k).X(:,8), cubesats(k).X
                        (:,9), 'k.');
192                 view([30 35])
193             nexttile
```

```
                title(sprintf("Cubesat %s Velocity Adjoints", cubesats(k).name
                    ))
                hold on; grid on; axis equal;
                plot3(cubesats(k).X(:,10), cubesats(k).X(:,11), cubesats(k).X
                    (:,12), 'k.');
                view([30 35])
            end

                % Save run!
            time = datetime('now');
            runName = sprintf("Run_%s", time);
            runName = replace(runName, " ", "_");
            runName = replace(runName, ":", "");
            runName = sprintf(".\\Runs\\%s.mat", runName);
            save(runName, "cubesats", "rings", "startRing", "endRing", "
                courseParams", '-mat');

        case 2
            %% Choose run to analyze
            fprintf("\n\tChoose a run to analyze!\n");
            cd .\Runs\
            [file, path, ~] = uigetfile('.mat', "Pick a MAT file to analyze");
            cd ..\

            load([path,file])

            fileParts = split(file, "_");
            scenario = extractBefore(fileParts{end}, ".mat");

            menuMsg = sprintf("\n\tWould you like to save new figures?\n" + ...
                              "\t1. Yes\n" + ...
                              "\t2. No\n" + ...
                              "\tChoice (enter number of option to do): ");
            choice = input(menuMsg);

            if choice == 1
                makeFigures = true;
            else
                makeFigures = false;
            end

            %% Plot race course and example ring
                % Race course
            courseNum = 420;
            titleText = sprintf("Generated Race Course"); atEnd = false;
            xLabel = sprintf("Radial [m]"); yLabel = sprintf("Along Track [m]");
                zLabel = sprintf("Cross Track [m]");
            courseFig = plotCourse(startRing, rings, endRing, cubesats, courseNum,
                 titleText, xLabel, yLabel, zLabel, atEnd);

            if makeFigures
                angles = [0, 90, 180, 270];
                for k = angles
                    view([-30 + k, 35]);
```

```
            fileName = sprintf(".\\Figures\\%s\\Course\\InitCourse_%.0fdegRot.png", scenario, k);
            saveas(courseFig, fileName);
        end
    end
    courseFig.WindowState = "minimized";

        % Example ring
    ringNum = 421;
    titleText = sprintf("Example Race Course Ring");
    xLabel = sprintf("Radial [m]"); yLabel = sprintf("Along Track [m]"); zLabel = sprintf("Cross Track [m]");
    ringFig = plotExampleRing(rings, ringNum, titleText, xLabel, yLabel, zLabel);

    if makeFigures
        for k = angles
            view([30 + k, 35]);
            fileName = sprintf(".\\Figures\\%s\\Ring\\ExampleRing_%.0fdegRot.png", scenario, k);
            saveas(courseFig, fileName);
        end
    end
    ringFig.WindowState = "minimized";

    %% Plot trajectory segments
    menuMsg = sprintf("\n\tWould you like detailed segment plots or just an overview?\n" + ...
                      "\t1. Detailed plots (one plot per segment per cubesat, including time history plots. Generates LOTS of figures)\n" + ...
                      "\t2. Overview (one plot per segment with all cubesat trajectories, no time history plots)\n" + ...
                      "\tChoice (enter number of option to do): ");
    choice = input(menuMsg);

    if choice == 1
        for k = 1:length(cubesats)
            fprintf("\nPlotting Cubesat %s's Trajectory!\n", cubesats(k).name)
            for kk = 1:length(rings)
                    % Find indices
                kStart = find(cubesats(k).t == cubesats(k).tSeg(1,kk), 1, 'last');
                kEnd = find(cubesats(k).t == cubesats(k).tSeg(2,kk), 1, 'first');

                    % Extract proper segment
                t = cubesats(k).t(kStart:kEnd);
                X = cubesats(k).X(kStart:kEnd, :);
                u = cubesats(k).u(kStart:kEnd, :);

                    % Plot segment
```

```
                    figNum = kk + (k-1)*length(rings);
                    if kk == length(rings)
                        titleText = sprintf("Cubesat %s trajectory segment:
                            Ring %.0f to end", cubesats(k).name, kk-1);
                    else
                        titleText = sprintf("Cubesat %s trajectory segment:
                            Ring %.0f to %.0f", cubesats(k).name, kk-1, kk);
                    end
                    xLabel = "Radial [m]"; yLabel = "Along Track [m]"; zLabel
                        = "Cross Track [m]";
                    figDetailed = plotSegment(cubesats(k), rings(kk), t, X, u,
                        figNum, titleText, xLabel, yLabel, zLabel);

                    if makeFigures
                        for idx = angles
                            nexttile(1);
                            view([-30 + idx, 35]);
                            fileName = sprintf(".\\Figures\\%s\\
                                SegmentsDetailed\\DetailedSegment_%s_Ring%.0
                                fTo%.0f_%.0fdegRot.png", scenario, cubesats(k)
                                .name, kk-1, kk, idx);
                            saveas(figDetailed, fileName);
                        end
                    end
                    figDetailed.WindowState = "minimized";

                        % Report progress
                    fprintf("\tCubesat %s passed through ring %.0f in %.3f sec
                        !\n", cubesats(k).name, kk, cubesats(k).t(kEnd) -
                        cubesats(k).t(kStart));
                end
                fprintf("\n\tCubesat %s finished the course in %.3f sec!\n",
                    cubesats(k).name, cubesats(k).t(end)-cubesats(k).t(1));


            end
        else
            for k = 1:length(rings)
                    % Plot segment
                figNum = k;
                if k == length(rings)
                    titleText = sprintf("Cubesat trajectory segment: Ring %.0f
                        to end", k-1);
                else
                    titleText = sprintf("Cubesat trajectory segment: Ring %.0f
                        to %.0f", k-1, k);
                end
                xLabel = "Radial [m]"; yLabel = "Along Track [m]"; zLabel = "
                    Cross Track [m]";
                figOverview = plotSegment_all(cubesats, rings(k), k, figNum,
                    titleText, xLabel, yLabel, zLabel);

                if makeFigures
                    for idx = angles
```

```
                    nexttile(1);
                    view([-30 + idx, 35]);
                    fileName = sprintf(".\\Figures\\%s\\SegmentsOverview\\
                        OverviewSegment_Ring%.0fTo%.0f_%.0fdegRot.png",
                        scenario, k-1, k, idx);
                    saveas(figOverview, fileName);
                end
            end
            figOverview.WindowState = "minimized";
        end
    end

        % Plot completed course
    courseFig = figure(courseNum);
    courseFig.WindowState = "maximized";
    view(-30, 35)
    for k = 1:length(cubesats)
        plot3(cubesats(k).X(:,1), cubesats(k).X(:,2), cubesats(k).X(:,3),
            '-', 'Color', cubesats(k).color, 'DisplayName', sprintf("
            Cubesat %s trajectory", cubesats(k).name));
    end

    if makeFigures
        for idx = angles
            view([-30 + idx, 35]);
            fileName = sprintf(".\\Figures\\%s\\Course\\FinishedCourse_%.0
                fdegRot.png", scenario, idx);
            saveas(courseFig, fileName);
        end
    end
    courseFig.WindowState = "minimized";

    %% Plot full trajectory
    for k = 1:length(cubesats)
        trajNum = 421 + k;
        titleText = sprintf("Full Trajectory of Cubesat %s", cubesats(k).
            name);
        trajFig = plotFullTrajectory(cubesats(k), rings, trajNum,
            titleText);
        if makeFigures
            fileName = sprintf(".\\Figures\\%s\\Trajectories\\Trajectory_%
                s.png", scenario, cubesats(k).name);
            saveas(trajFig, fileName);
        end
        trajFig.WindowState = "minimized";
    end


    %% Analyze race course results
    fprintf("\n\t---Analyzing Race Outcome---\n")

        % Compile costs and course times
    costs = [];
    minCosts = [];
```

```
        times = [];
        for k = 1:length(cubesats)
            cost = sum(cubesats(k).cost(1,:));
            minCost = sum(cubesats(k).cost(2,:));
            time = cubesats(k).t(end) - cubesats(k).t(1);
            costs = [costs; cost];
            minCosts = [minCosts; minCost];
            times = [times; time];
        end
        accuracies = costs - times; % cost = time cost + accuracy cost

            % Find winner (smallest cost)
        [minCost, winner] = min(costs);

            % Report results
        for k = 1:length(cubesats)
            fprintf("\nCubesat %s results: overall cost = %.3f, time = %.3f,
               accuracy = %.3f. \n\tMinimum possible cost: %.3f\n", cubesats(
               k).name, costs(k), times(k), accuracies(k), minCosts(k));
        end

        fprintf("\n\n\t~~~Cubesat %s won the race!~~~\n\n", cubesats(winner).
           name);

    case 3
        %% Choose run to animate
        fprintf("\n\tChoose a run to animate!\n");
        cd .\Runs\
        [file, path, ~] = uigetfile('.mat', "Pick a MAT file to analyze");
        cd ..\

        load([path,file])

        fileParts = split(file, "_");
        scenario = extractBefore(fileParts{end}, ".mat");

        %% Animate!
        saveMovie = true;
        movieTitle = sprintf(".\\Videos\\ASEN6020_%s_animated", scenario);
        figAnim = animateRun(cubesats, rings, saveMovie, movieTitle);

    case 4
        %% Say goodbye
        fprintf("\n\tHave a great day!\n")
        return;
end
```

C. fmincon Wrapper

Back to index: C.A

```
function [tOpt, XOpt, uOpt, xSolved, x0, cost] = solveTrajectory(cubesat, ring
    , courseParams, opt, isFinal, debug)
% Function that applies the optimal control law to navigate from one ring
% of the race course to another for the current ring problem
%   Inputs:
%       - cubesat: CubeSat currently running the course
%       - ring: Ring the CubeSat is attempting to navigate to
%       - courseParams: Course parameters structure that includes mean
%                       motion of the race course origin
%       - opt: ODE45 options via odeset
%       - isFinal: Boolean indicating whether this ring corresponds to the
%                  final problem or not
%       - debug: Boolean vector that enables or disables plotting and debug
%                features, like so:
%                - debug(1): Whether fmincon outputs a message at each
%                            iteration and plots the current guessed point
%                            true: output message and plot point
%                - debug(2): Whether the current course segment guess is
%                            plotted at each call of the cost function
%                            true: Plot current course segment guess
%                - debug(3): Whether the command window print sequence is
%                            disabled or not.
%                            true: Disable print sequence
%   Outputs:
%       - tOpt: Optimal time solved for this section of the course
%       - XOpt: Optimal CubeSat trajectory solved for this section of the
%               course
%       - uOpt: Optimal control applied over this optimal trajectory
%       - xSolved: The resulting guess on p0 and t_f, organized as
%                  [p0; t_f]
%       - x0: The initial guess on p0 and t_f, organized as [p0; t_f]
%       - cost: The actual and minimum possible cost for this course
%               segment, organized as [actualCost; minCost]
%
%   By: Ian Faber, 04/19/2025
%

    % Set up problem: Trying to solve for p0 and t_f such that K is minimized.
        % Set initial guesses

            % p0
r0 = cubesat.X0(1:3);
v0 = cubesat.X0(4:6);
vHat0 = v0/norm(v0);
dVec = ring.center - r0;
uMax = cubesat.uMax;
if norm(v0) == 0 % Initial ring - cubesat at rest
    p0 = [(r0 - ring.params.lastRing.center)/norm(r0 - ring.params.lastRing.
        center);
          -dVec/norm(dVec)];
else
```

```
    p0 = [(r0 - ring.params.lastRing.center)/norm(r0 - ring.params.lastRing.center);
          -(1/norm(v0))*dVec];
end

            % tf
        % Set lower bound on tf to force it to be positive and physically
        % possible given the maximum possible acceleration for this
        % cubesat. "minimum time" modeled with kinematics off of the
        % straight line distance between the cubesat and next ring if
        % everything lined up perfectly (it won't!). If it does, the
        % minimum time is slightly overconfident (reduced) to ensure the
        % optimal solution isn't skipped.
d = 0 - norm(dVec); % d = d0 - df, where d0 = 0
tMin = max((1/norm(uMax))*[-norm(v0) + sqrt(norm(v0)^2 - 2*norm(uMax)*d); -norm(v0) - sqrt(norm(v0)^2 - 2*norm(uMax)*d)]);

tf = cubesat.t0 + 10*tMin;

            % Combine initial guesses
x0 = [p0; tf];

        % Set bounds
lb = [-1e3*ones(6,1); cubesat.t0 + tMin];
ub = [1e3*ones(6,1); Inf];

    % Solve problem
if debug(1)
    options = optimoptions("fmincon", "Algorithm", "sqp", "EnableFeasibilityMode", true, "MaxFunctionEvaluations", 3000, ...
                           "SubproblemAlgorithm", "cg", 'Display', 'iter-detailed', 'PlotFcn', 'optimplotx');
else
    options = optimoptions("fmincon", "Algorithm", "sqp", "EnableFeasibilityMode", true, "MaxFunctionEvaluations", 3000, ...
                           "SubproblemAlgorithm", "cg", 'Display', 'none');
end

fprintf("\n") % Buffer newline for command window print sequence

[xSolved, cost] = fmincon(@(x)costFunction(x,ring,cubesat,courseParams,opt,isFinal,debug),x0,[],[],[],[],lb,ub,@(x)constraints(x,ring,cubesat,courseParams,opt,isFinal), options);

fprintf("\b") % Delete newline or last character of print sequence

    % Report constraints
[ineq, eq] = constraints(xSolved, ring, cubesat, courseParams, opt, isFinal);

if isFinal % Final problem constraints
    eqLabels = ["v_f at rest constraint", "H equivalence constraint", "H_f constraint", "X_0 constraint", "t_0 constraint"];

    [~, maxEqIdx] = max(abs(eq));
```

```
96
97      fprintf("\t\tNo inequality constraints in final problem!\n")
98
99      if any(maxEqIdx == 1:3)
100         const = eqLabels(1);
101         offset = 1;
102     elseif maxEqIdx == 4
103         const = eqLabels(2);
104         offset = 4;
105     elseif maxEqIdx == 5
106         const = eqLabels(3);
107         offset = 5;
108     elseif any(maxEqIdx == 6:11)
109         const = eqLabels(4);
110         offset = 6;
111     elseif maxEqIdx == 12
112         const = eqLabels(5);
113         offset = 12;
114     else
115         const = "Unknown constraint";
116         offset = maxEqIdx;
117     end
118     fprintf("\t\tMax equality constraint magnitude: %.3e at position %.0f, %s\
            n", eq(maxEqIdx), maxEqIdx-offset, const);
119 else % Initial/intermediate problem constraints
120     eqLabels = ["H equivalence constraint", "H_f constraint", "X_0 constraint
            ", "t_0 constraint", "vHat_f pointing constraint"];
121     ineqLabels = ["r_f ring distance constraint", "r_f ring plane constraint
            "];
122
123     [~, maxIneqIdx] = max(abs(ineq));
124     [~, maxEqIdx] = max(abs(eq));
125
126     fprintf("\t\tMax inequality constraint magnitude: %.3e, %s\n", ineq(
            maxIneqIdx), ineqLabels(maxIneqIdx));
127
128     if maxEqIdx == 1
129         const = eqLabels(1);
130         offset = 1;
131     elseif maxEqIdx == 2
132         const = eqLabels(2);
133         offset = 2;
134     elseif any(maxEqIdx == 3:8)
135         const = eqLabels(3);
136         offset = 3;
137     elseif maxEqIdx == 9
138         const = eqLabels(4);
139         offset = 9;
140     elseif any(maxEqIdx == 10:12)
141         const = eqLabels(5);
142         offset = 10;
143     else
144         const = "Unknown constraint";
145         offset = maxEqIdx;
```

```
    end
    fprintf("\t\tMax equality constraint magnitude: %.3e at position %.0f, %s\
       n", eq(maxEqIdx), maxEqIdx-offset, const);

end

    % Propagate optimal trajectory
p0 = xSolved(1:6);
tf = xSolved(7);

X0 = [cubesat.X0; p0];

dt = 0.1;
tspan = cubesat.t0:dt:tf;
if tspan(end) ~= tf
    tspan = [tspan, tf];
end

[tOpt, XOpt] = ode45(@(t,X)CHWEOM(t,X,cubesat,courseParams), tspan, X0, opt);

    % Append absolute minimum cost to the cost vector for comparison.
    % Absolute minimum cost corresponds to the straight line distance time
    % taken between ring centers, assuming all velocities line up and the
    % Cubesat hits the center of the ring exactly.
cost = [cost; tMin];

    % Back out optimal control according to control law
uOpt = [];
for k = 1:length(tOpt)
    pvx = XOpt(k,10);
    pvy = XOpt(k,11);
    pvz = XOpt(k,12);

    uMax = 0;
    if length(cubesat.uMax) > 1 % Axial thrusting is at play!
        uComp = cubesat.uMax;
        if abs(pvx) > 1 % Cubesat has max x acceleration at time t
            uMax = uMax + [uComp(1); 0; 0];
        end
        if abs(pvy) > 1 % Cubesat has max y acceleration at time t
            uMax = uMax + [0; uComp(2); 0];
        end
        if abs(pvz) > 1 % Cubesat has max z acceleration at time t
            uMax = uMax + [0; 0; uComp(3)];
        end
    else
        uMax = cubesat.uMax;
    end

    pvHat = [pvx; pvy; pvz]/norm([pvx; pvy; pvz]);
    u = -norm(uMax)*pvHat';
    uOpt = [uOpt; u];
end

```

```
199 end
```

D. Cost Function

Back to index: C.A

```
function K = costFunction(x, ring, cubesat, courseParams, opt, isFinal, debug)
% Implements the cost function for the CubeSat racing current ring problem
%   Inputs:
%       - x: Vector of parameters to minimize the cost function with. Here,
%            that's [p0; t_f]
%       - ring: Ring structure for the ring the CubeSat is trying to get
%               to, as defined in generateRing.m
%       - cubesat: Cubesat structure for the CubeSat in question, as
%                  defined by generateCubesat.m
%       - courseParams: Course parameters structure that includes the mean
%                       motion of the race course origin's orbit
%       - opt: ODE45 options via odeset
%       - isFinal: Boolean indicating whether this ring corresponds to the
%                  final problem or not
%       - debug: Boolean vector that enables or disables plotting and debug
%                features, like so:
%                - debug(1): Whether fmincon outputs a message at each
%                            iteration and plots the current guessed point
%                            true: output message and plot point
%                - debug(2): Whether the current course segment guess is
%                            plotted at each call of the cost function
%                            true: Plot current course segment guess
%                - debug(3): Whether the command window print sequence is
%                            disabled or not.
%                            true: Disable print sequence
%   Outputs:
%       - K: Cost function value for the current ring problem
%
%   By: Ian Faber, 04/19/2025
%

    % Command window print sequence
persistent idx;
if isempty(idx)
    idx = 1;
end
sequence = ['/','/','/','-','-','-','\','\','\','|','|','|'];
if all(~debug) % Only print sequence if no other debugging is happening
    fprintf("\b%s",sequence(idx));
end
idx = idx + 1;
if idx > length(sequence)
    idx = 1;
end

    % Pull out current p0 and tf guesses
p0 = x(1:6);
tf = x(7);

    % Propagate [X0; p0] through CHW equations according to the guess
X0 = [cubesat.X0; p0];
```

```
[t, X] = ode45(@(t,X)CHWEOM(t,X,cubesat,courseParams), [cubesat.t0, tf], X0,
    opt);

    % Calculate optimal control if segment guess plotting is enabled
if debug(2)
        % Back out optimal control
    uOpt = [];
    for k = 1:length(t)
        pvx = X(k,10);
        pvy = X(k,11);
        pvz = X(k,12);

        uMax = 0;
        if length(cubesat.uMax) > 1 % Axial thrusting is at play!
            uComp = cubesat.uMax;
            if abs(pvx) > 1 % Cubesat has max x acceleration at time t
                uMax = uMax + [uComp(1); 0; 0];
            end
            if abs(pvy) > 1 % Cubesat has max y acceleration at time t
                uMax = uMax + [0; uComp(2); 0];
            end
            if abs(pvz) > 1 % Cubesat has max z acceleration at time t
                uMax = uMax + [0; 0; uComp(3)];
            end
        else
            uMax = cubesat.uMax;
        end

        pvHat = [pvx; pvy; pvz]/norm([pvx; pvy; pvz]);
        u = -norm(uMax)*pvHat';
        uOpt = [uOpt; u];
    end
end

    % Pull out final position
rf = X(end,1:3)';

    % Assign cost depending on the problem being solved
if isFinal % Final problem - just minimize time to come to a stop
    K = tf - cubesat.t0;
else % Initial/intermediate problem - aim for rings and minimize time
    dVec = rf - ring.center;
    K = norm(dVec) + (tf - cubesat.t0);
end

    % Plot iteration
if debug(2)
    % figure(69420); clf;
    %     hold on; grid on; axis equal
    %     title("Iterated Trajectory");
    %     plotRing(ring.params.lastRing, 'g-');
    %     plot3(X(:,1), X(:,2), X(:,3), 'k-');
    %     plotRing(ring, 'r-');
    %     xlabel("Radial [km]"); ylabel("Along Track [km]"); zlabel("Cross
```

```
        Track [km]");
    %     view([30 35]); drawnow;
    titleText = sprintf("Iterated Trajectory");
    xLabel = "Radial [m]"; yLabel = "Along Track [m]"; zLabel = "Cross Track [
        m]";
    plotSegment(cubesat, ring, t, X, uOpt, 69420, titleText, xLabel, yLabel,
        zLabel);
end

end
```

E. Constraints

Back to index: C.A

```
function [c, ceq] = constraints(x, ring, cubesat, courseParams, opt, isFinal)
% Nonlinear constraint function that is passed into fmincon, places
% constraints on the trajectory and other aspects of the current ring
% problem
%   Inputs:
%       - x: Vector of parameters to minimize the cost function with. Here,
%            that's [p0; t_f]
%       - ring: Ring structure for the ring the CubeSat is trying to get
%               to, as defined in generateRing.m
%       - cubesat: Cubesat structure for the CubeSat in question, as
%                  defined by generateCubesat.m
%       - courseParams: Course parameters structure that includes the mean
%                       motion of the race course origin's orbit
%       - opt: ODE45 options via odeset
%       - isFinal: Boolean indicating whether this ring corresponds to the
%                  final problem or not
%
%   Outputs:
%       - c: Nonlinear inequality vector, such that c(x) <= 0 for all
%            entries of c. If no inequalities, c = [].
%       - ceq: Nonlinear equality vector, such that ceq(x) == 0 for all
%              entries of c
%
%   By: Ian Faber, 04/19/2025
%

    % Pull out parameters
p0 = x(1:6);
tf = x(7);

    % Pull out ring parameters
n = ring.normal;
nHat = n/norm(n);

    % Find initial and final states for current p0 and t_f guess
X0 = [cubesat.X0; p0];
[t, X] = ode45(@(t,X)CHWEOM(t,X,cubesat,courseParams), [cubesat.t0 tf], X0,
    opt);

r0 = X(1,1:3)';
v0 = X(1,4:6)';

rf = X(end,1:3)';
vf = X(end,4:6)';

vfHat = vf/norm(vf);

pf = X(end,7:end)';

    % Pull out cubesat max acceleration at t0 and tf
uMax = cubesat.uMax;
```

```
uMax0 = zeros(3,1);
uMaxf = zeros(3,1);
if length(uMax) > 1  % Axial thrusting is at play!
        % t0
    if abs(dot(p0(4:6),[1;0;0])) > 0 % CubeSat has max x acceleration at t0
        uMax0 = uMax0 + [uMax(1); 0; 0];
    end
    if abs(dot(p0(4:6),[0;1;0])) > 0 % CubeSat has max y acceleration at t0
        uMax0 = uMax0 + [0; uMax(2); 0];
    end
    if abs(dot(p0(4:6),[0;0;1])) > 0 % CubeSat has max z acceleration at t0
        uMax0 = uMax0 + [0; 0; uMax(3)];
    end
        % tf
    if abs(dot(pf(4:6),[1;0;0])) > 0 % CubeSat has max x acceleration at tf
        uMaxf = uMaxf + [uMax(1); 0; 0];
    end
    if abs(dot(pf(4:6),[0;1;0])) > 0 % CubeSat has max y acceleration at tf
        uMaxf = uMaxf + [0; uMax(2); 0];
    end
    if abs(dot(pf(4:6),[0;0;1])) > 0 % CubeSat has max z acceleration at tf
        uMaxf = uMaxf + [0; 0; uMax(3)];
    end
else
    uMax0 = uMax;
    uMaxf = uMax;
end

    % Calculate initial and final gravity accelerations
g_0 = [2*courseParams.n*v0(2) + 3*courseParams.n^2*r0(1); -2*courseParams.n*v0
    (1); -courseParams.n^2*r0(3)];
g_f = [2*courseParams.n*vf(2) + 3*courseParams.n^2*rf(1); -2*courseParams.n*vf
    (1); -courseParams.n^2*rf(3)];

    % Calculate initial and final Hamiltonians
H0 = p0(1:3)'*v0 + p0(4:6)'*g_0 - norm(p0(4:6))*norm(uMax0);
Hf = pf(1:3)'*vf + pf(4:6)'*g_f - norm(pf(4:6))*norm(uMaxf);

    % Assign final constraints according to problem type
if isFinal % Final problem, want vFinal = 0 but also dynamically consistent!
    No rings to hit
        % Final velocity constraint
    ceq_v = vf; % vf = 0

        % Hamiltonian constraints
    ceq_Hf = Hf - (-1); % Hf = -1
    ceq_H = Hf - H0; % Hf = H0

        % Manifold constraints
    ceq_X0 = [r0; v0] - cubesat.X0;
    ceq_t0 = t(1) - cubesat.t0;

        % Assign final constraints for fmincon
    ceq = [ceq_v; ceq_H; ceq_Hf; ceq_X0; ceq_t0];
```

```
    c = [];
else % Initial and intermediate problems, want to ensure we pass through rings and keep things dynamically consistent
        % Ring inequality constraints
    c_rf_dist = norm(rf - ring.center) - min(ring.S, [], 'all'); % Want in-plane component of final intersection to be close to the ring center, i.e. |r_f - r_r,i| <= smallest size of S
    c_rf_plane = abs(dot((rf - ring.center),nHat)) - 0.1; %min(ring.S, [], 'all'); % Want normal component of final intersection to be close to the plane of the ring, i.e. dot((r_f = r_r,i), nHat) <= smallest size of S

        % Hamiltonian constraints
    ceq_Hf = Hf - (-1); % H_f = -1
    ceq_H = Hf - H0; % H_f = H_0

        % Manifold constraints
    ceq_X0 = [r0; v0] - cubesat.X0;
    ceq_t0 = t(1) - cubesat.t0;
    ceq_vHatf = vfHat - nHat;

        % Assign final constraints for fmincon
    ceq = [ceq_H; ceq_Hf; ceq_X0; ceq_t0; ceq_vHatf];
    c = [c_rf_dist; c_rf_plane];
end

end
```

F. Dynamics

Back to index: C.A

```
function dX = CHWEOM(t, X, cubeSatParams, courseParams)
% Function that implements the Clohessy-Hill-Wiltshire equations, including
% the optimal control law for the CubeSat racing problem
%   Inputs:
%       - t: Current integration time
%       - X: Current CubeSat state, including adjoints:
%            [x; y; z; vx; vy; vz; px; py; pz; pvx; pvy; pvz]
%       - cubeSatParams: Parameters structure for the CubeSat of interest,
%                        as defined by generateCubesat.m
%       - courseParams: Parameters structure for the current race course
%   Outputs:
%       - dX: Rate of change of CubeSat state including adjoints
%
%   By: Ian Faber, 03/23/2025
%
%% Parse state
x = X(1);
y = X(2);
z = X(3);
vx = X(4);
vy = X(5);
vz = X(6);
px = X(7);
py = X(8);
pz = X(9);
pvx = X(10);
pvy = X(11);
pvz = X(12);

%% Get constants
n = courseParams.n;

uMax = 0;
if length(cubeSatParams.uMax) > 1 % Axial thrusting is at play!
    u = cubeSatParams.uMax;
    if abs(pvx) > 1 % Cubesat has max x acceleration at time t
        uMax = uMax + [u(1); 0; 0];
    end
    if abs(pvy) > 1 % Cubesat has max y acceleration at time t
        uMax = uMax + [0; u(2); 0];
    end
    if abs(pvz) > 1 % Cubesat has max z acceleration at time t
        uMax = uMax + [0; 0; u(3)];
    end
else
    uMax = cubeSatParams.uMax;
end

%% Physical dynamics
rDot = [vx; vy; vz];

```

```
pvHat = [pvx; pvy; pvz]/norm([pvx; pvy; pvz]);
g = [2*n*vy + 3*n^2*x; -2*n*vx; -n^2*z];
vDot = g - norm(uMax)*pvHat;

%% Adjoint dynamics
pr = [px; py; pz];
pv = [pvx; pvy; pvz];

prDot = -[3*n^2 0 0; 0 0 0; 0 0 -n^2]'*pv;
pvDot = -pr - [0 2*n 0; -2*n 0 0; 0 0 0]'*pv;

%% Assign output
dX = [rDot; vDot; prDot; pvDot];

end
```

D. Scenario 1 Individual Ring Segments

Back to appendix index: A.A

Here is a deluge of plots showing off each ring segment of Scenario 1. Enjoy!

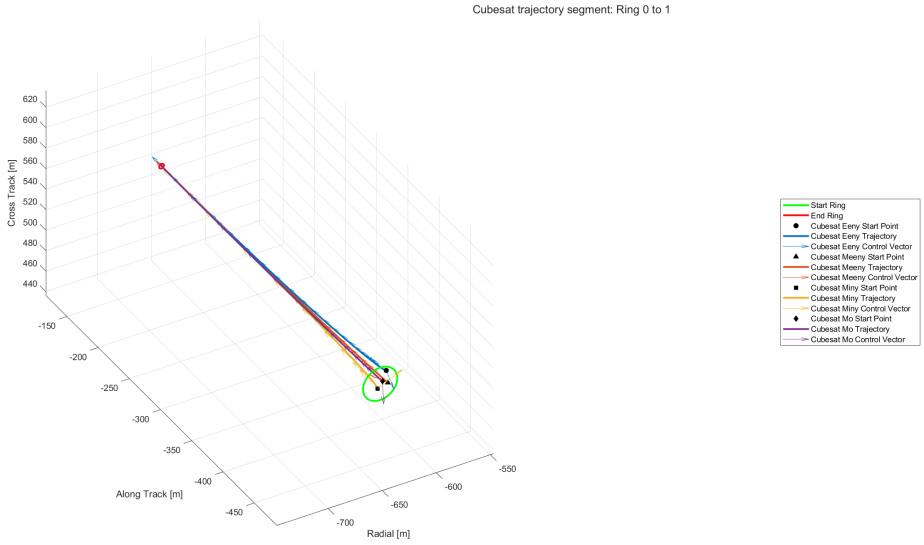

Fig. 13 Scenario 1 Segment 0 to 1

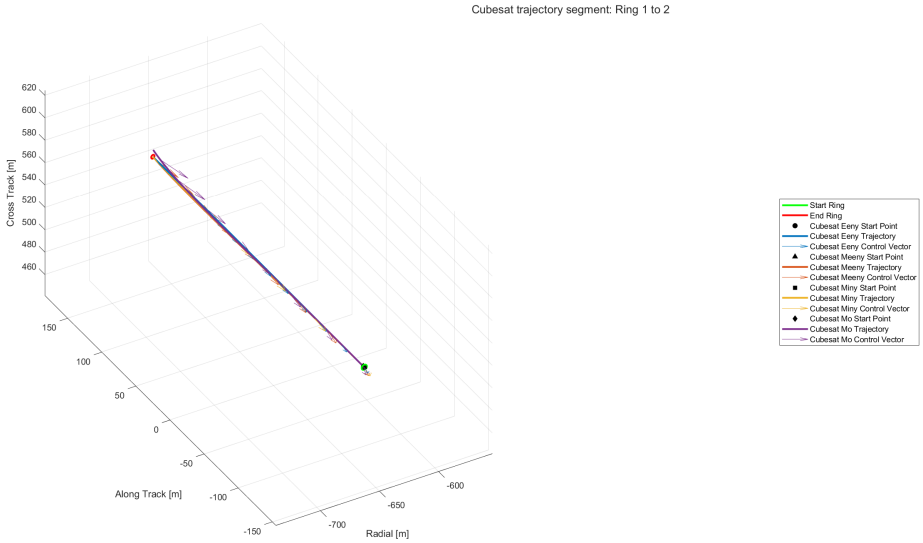

Fig. 14 Scenario 1 Segment 1 to 2

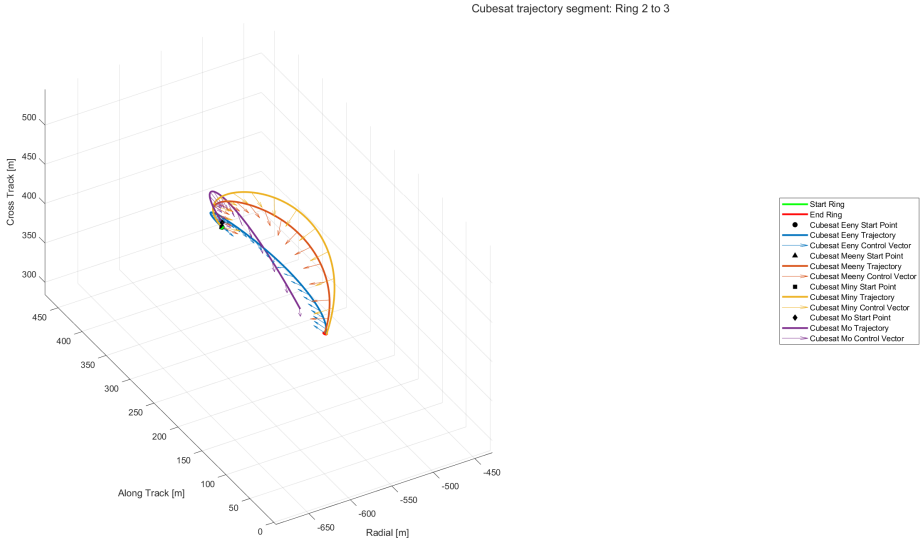

Fig. 15 Scenario 1 Segment 2 to 3

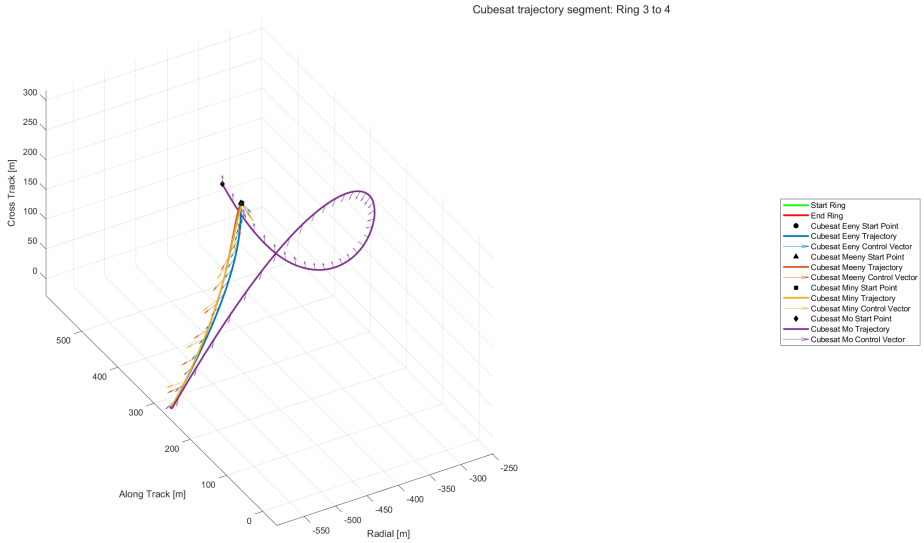

Fig. 16 Scenario 1 Segment 3 to 4

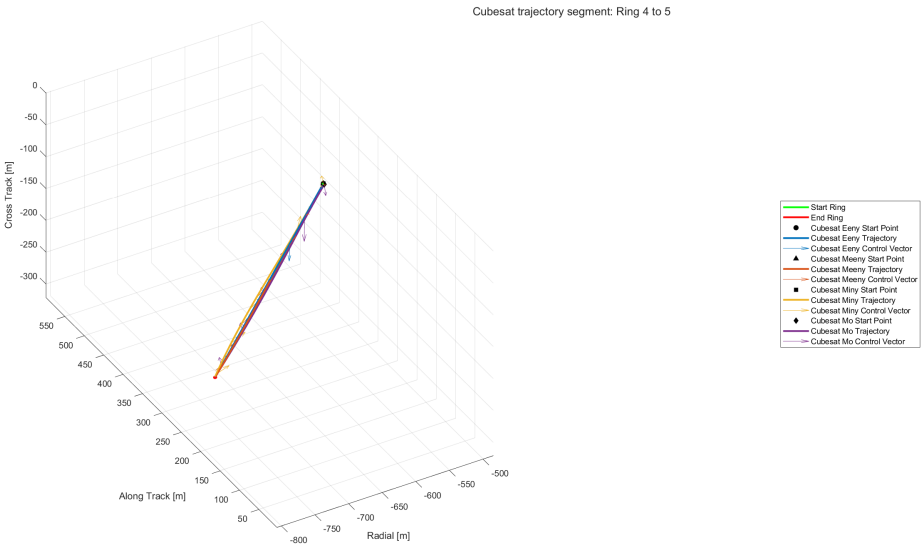

Fig. 17 Scenario 1 Segment 4 to 5

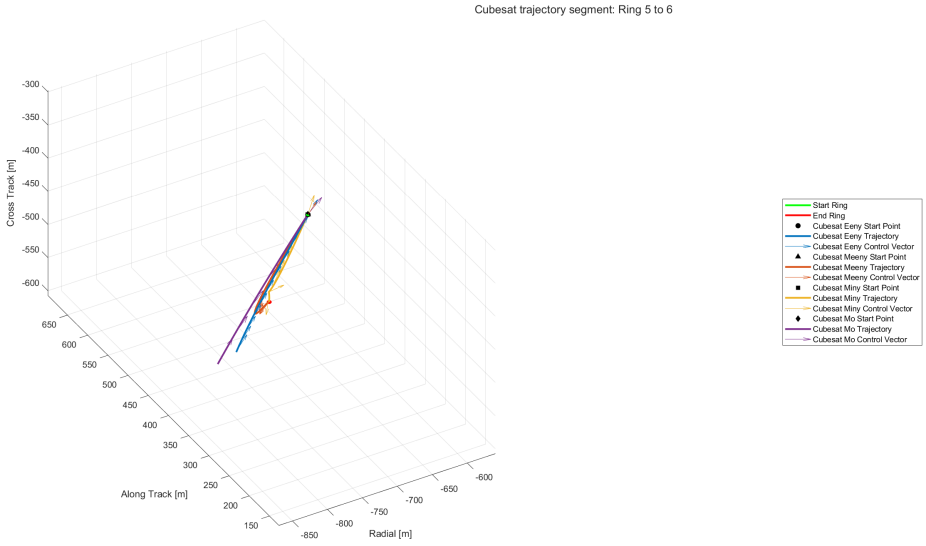

Fig. 18 Scenario 1 Segment 5 to 6

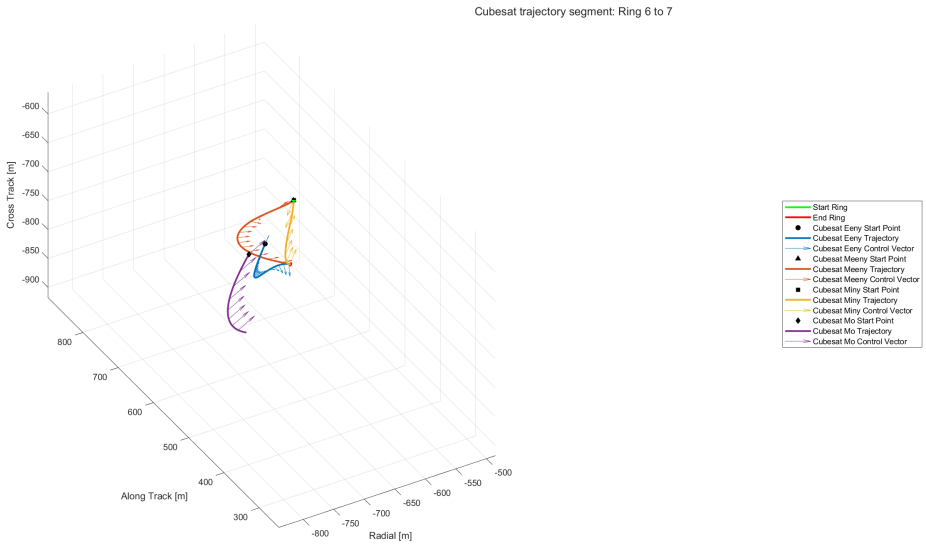

Fig. 19 Scenario 1 Segment 6 to 7

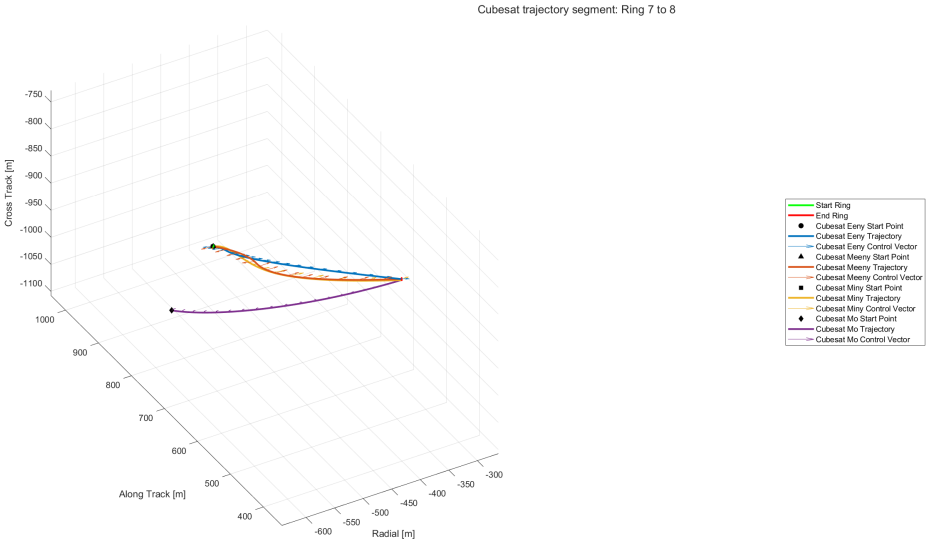

Fig. 20 Scenario 1 Segment 7 to 8

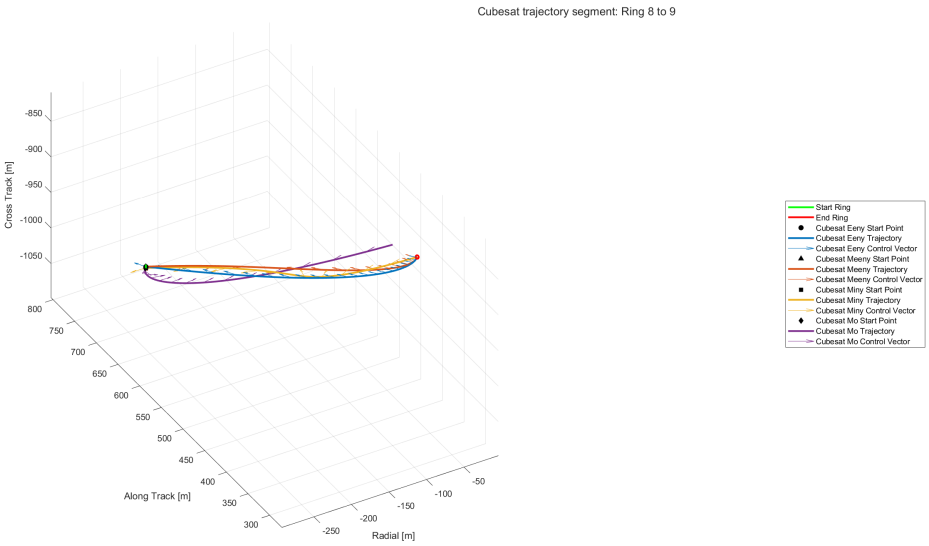

Fig. 21 Scenario 1 Segment 8 to 9

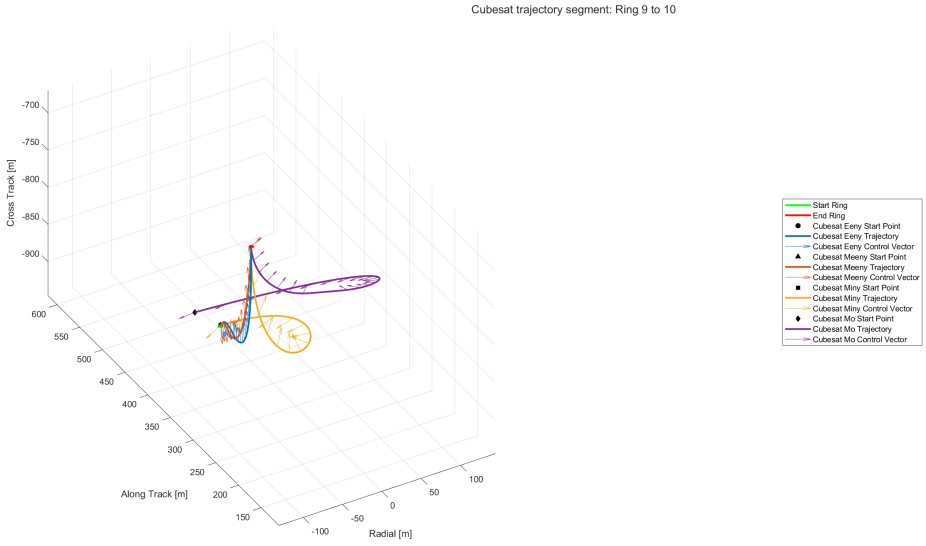

Fig. 22 Scenario 1 Segment 9 to 10

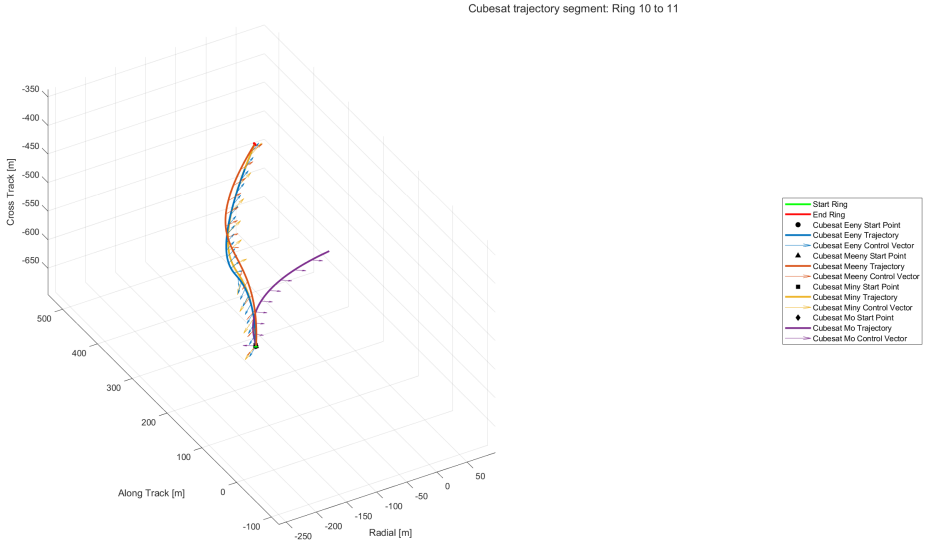

Fig. 23 Scenario 1 Segment 10 to 11

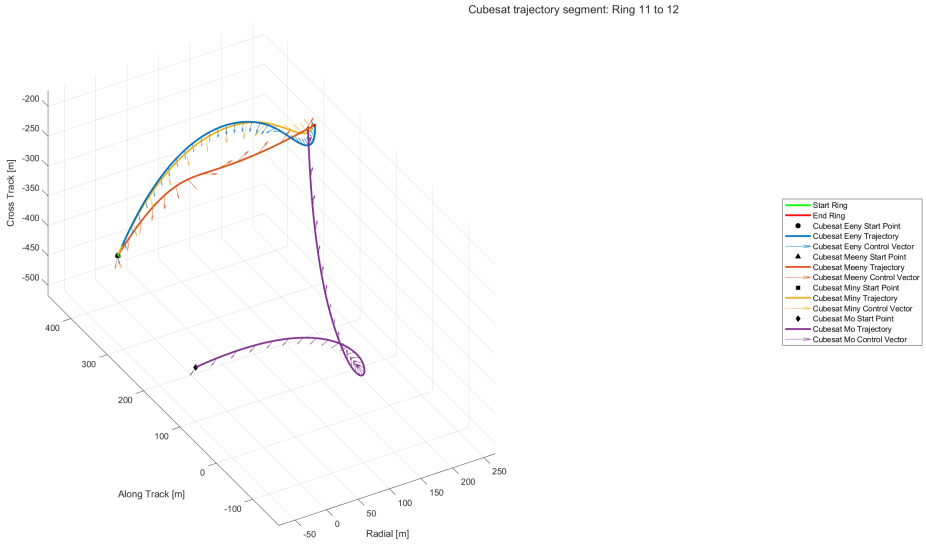

Fig. 24 Scenario 1 Segment 11 to 12

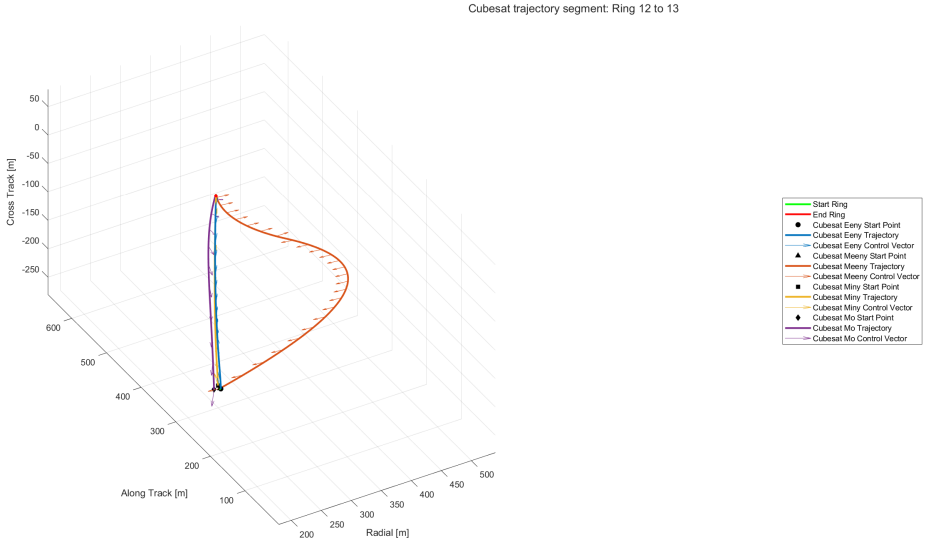

Fig. 25 Scenario 1 Segment 12 to 13

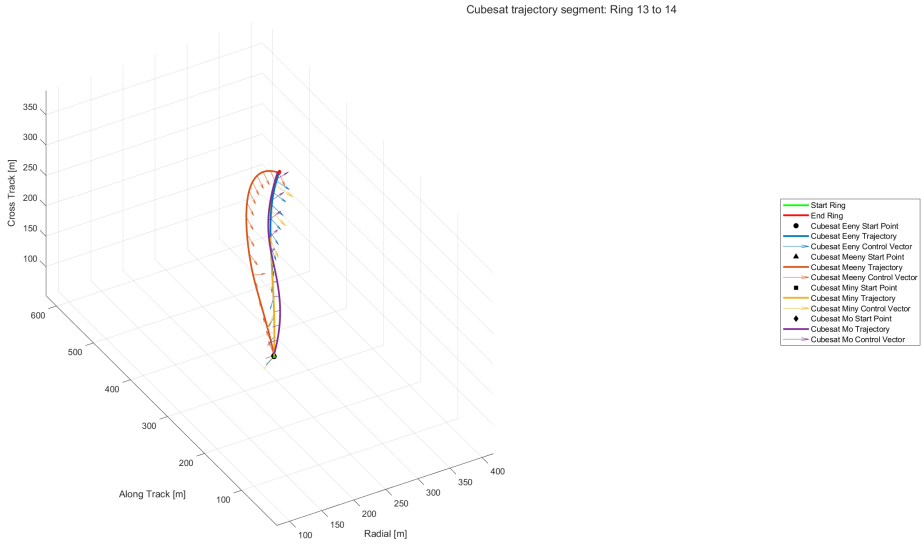

Fig. 26 Scenario 1 Segment 13 to 14

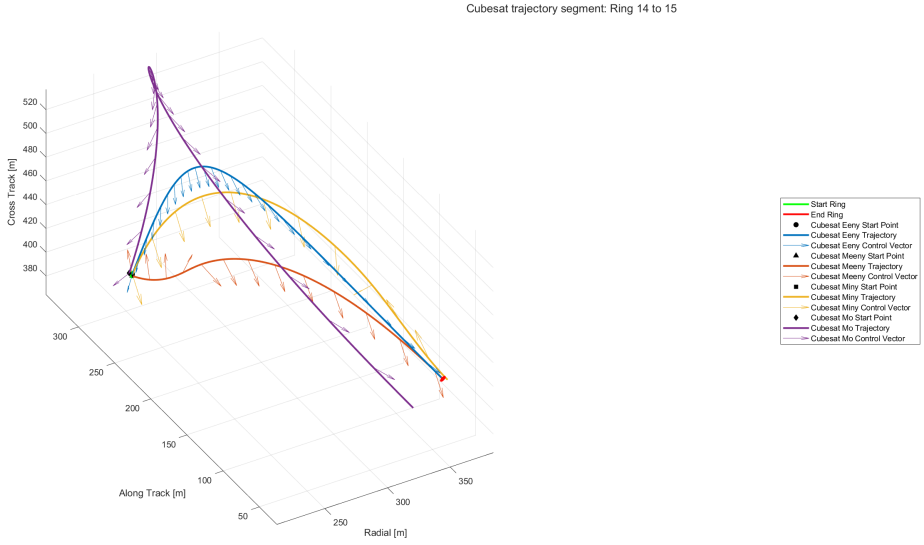

Fig. 27 Scenario 1 Segment 14 to 15

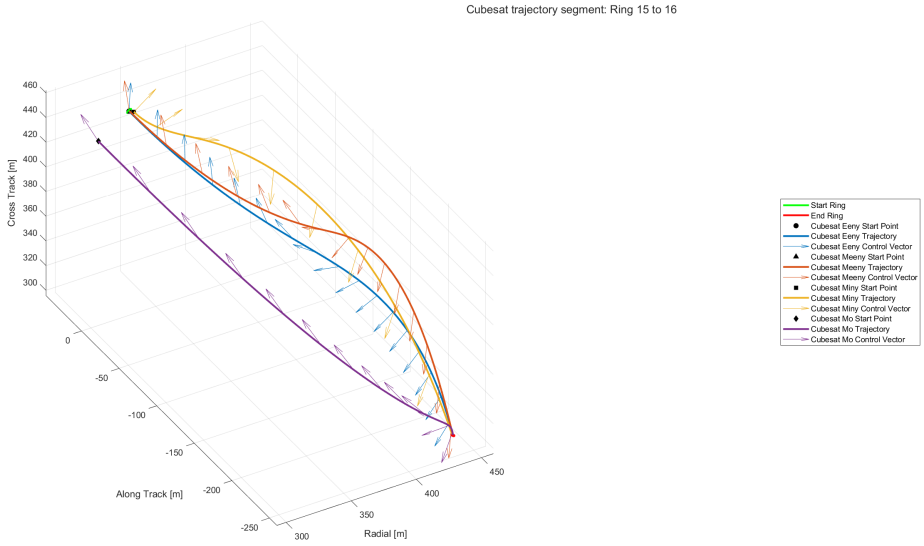

Fig. 28 Scenario 1 Segment 15 to 16

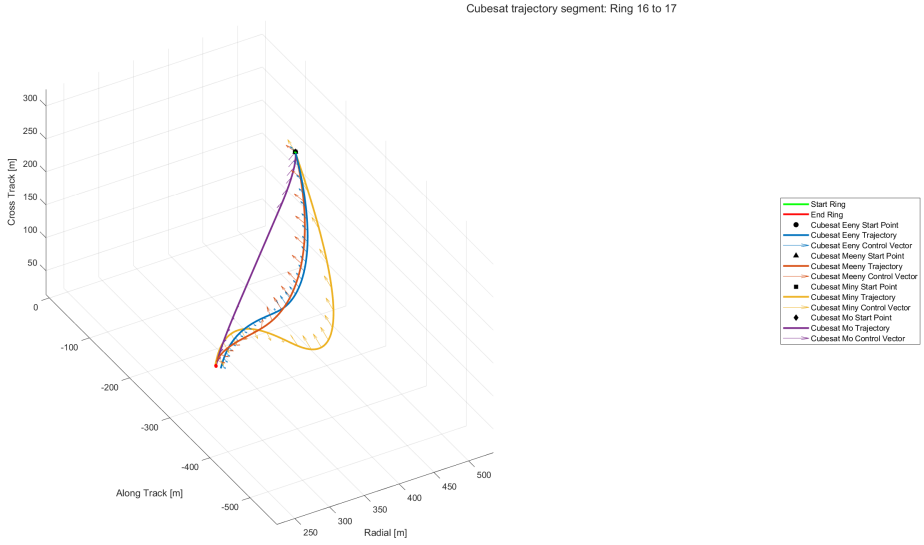

Fig. 29 Scenario 1 Segment 16 to 17

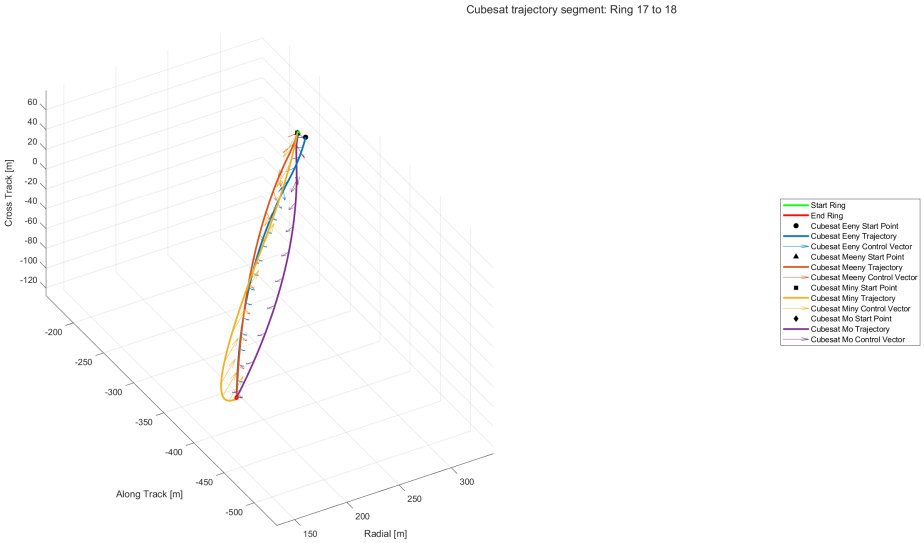

Fig. 30 Scenario 1 Segment 17 to 18

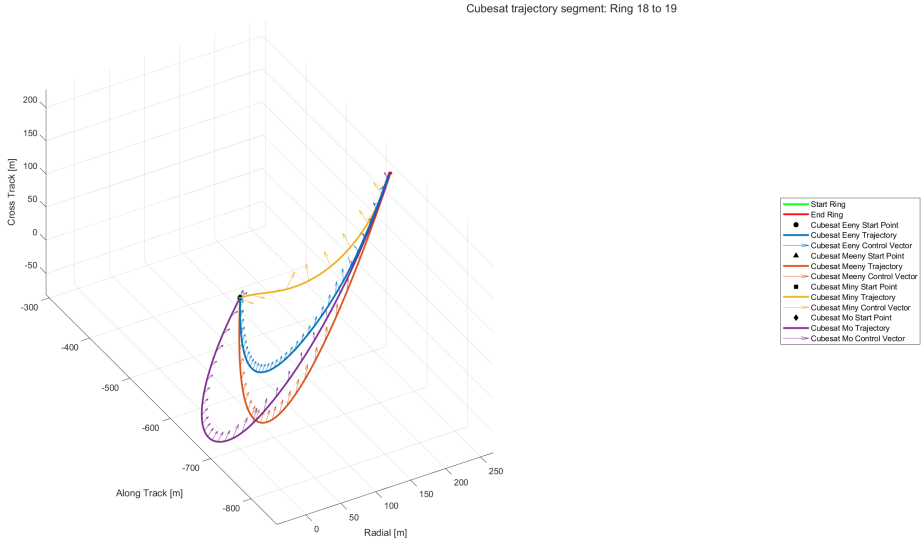

Fig. 31 Scenario 1 Segment 18 to 19

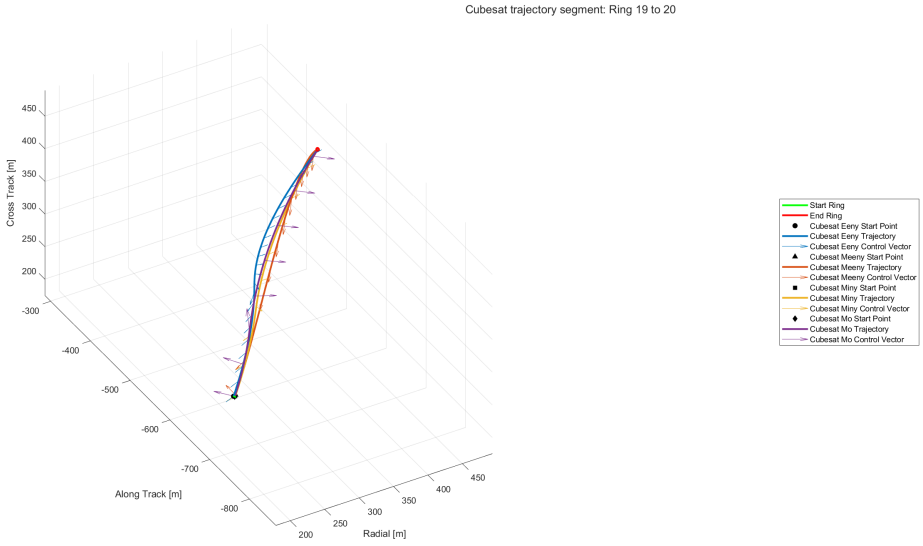

Fig. 32 Scenario 1 Segment 19 to 20

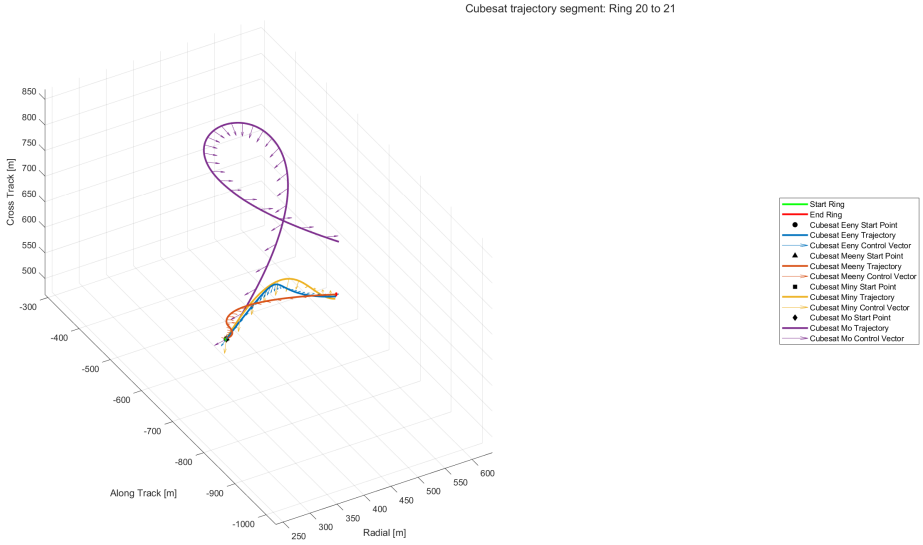

Fig. 33 Scenario 1 Segment 20 to 21

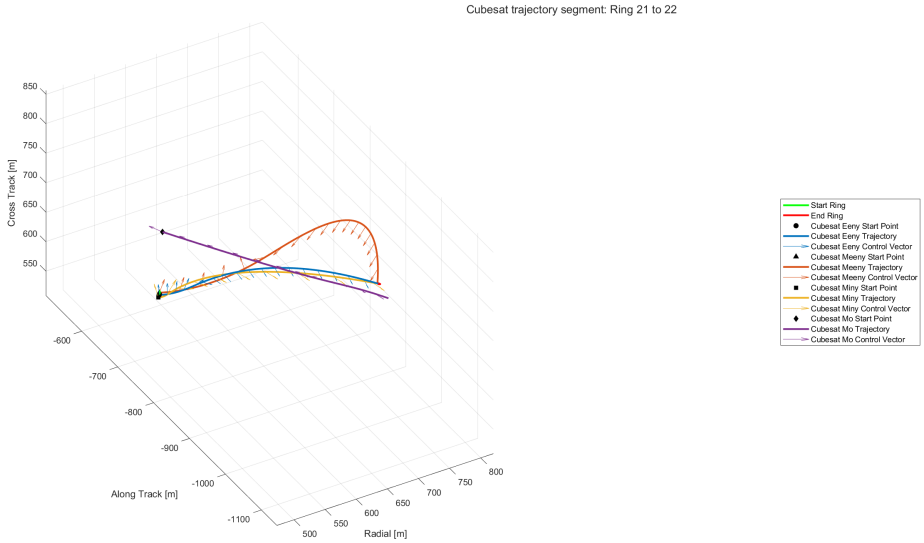

Fig. 34 Scenario 1 Segment 21 to 22

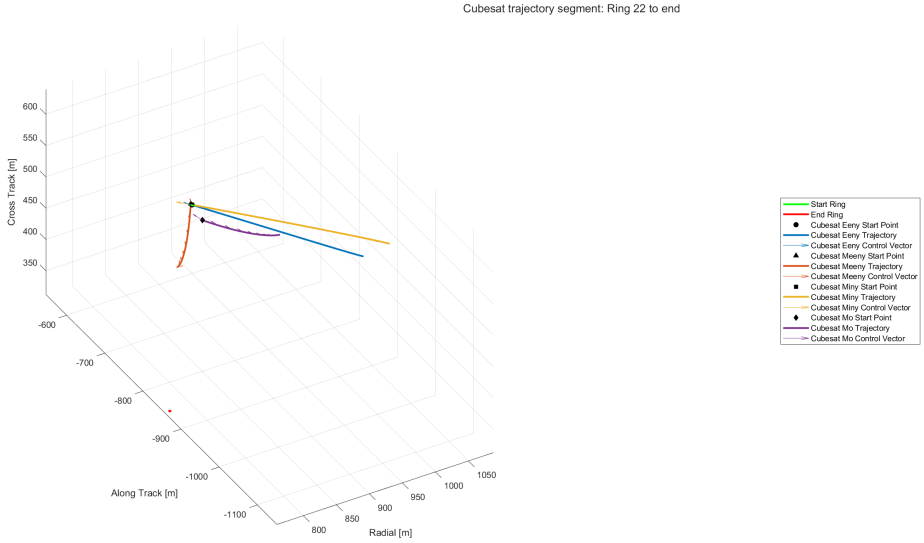

Fig. 35 Scenario 1 Segment 22 to end

E. Scenario 2 Individual Ring Segments

Back to appendix index: A.A

Here is a deluge of plots showing off each ring segment of Scenario 2. Enjoy!

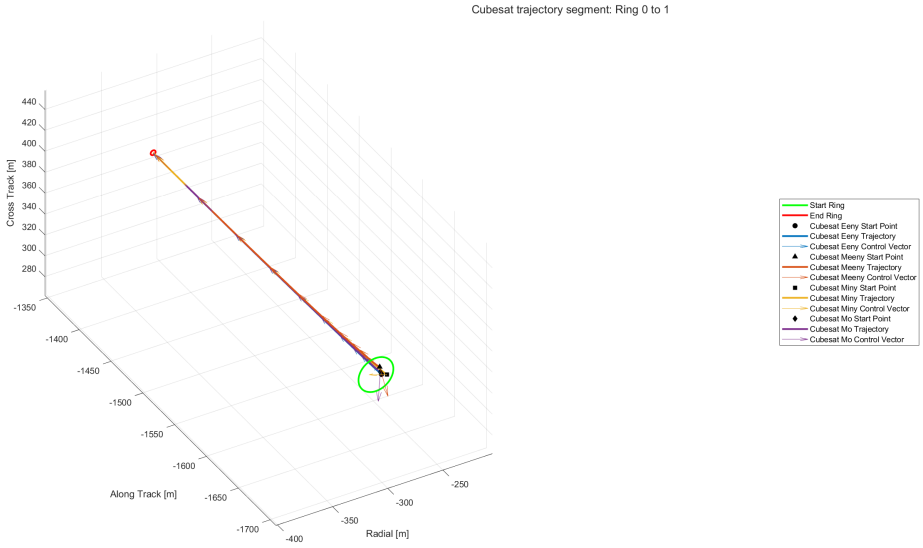

Fig. 36 Scenario 2 Segment 0 to 1

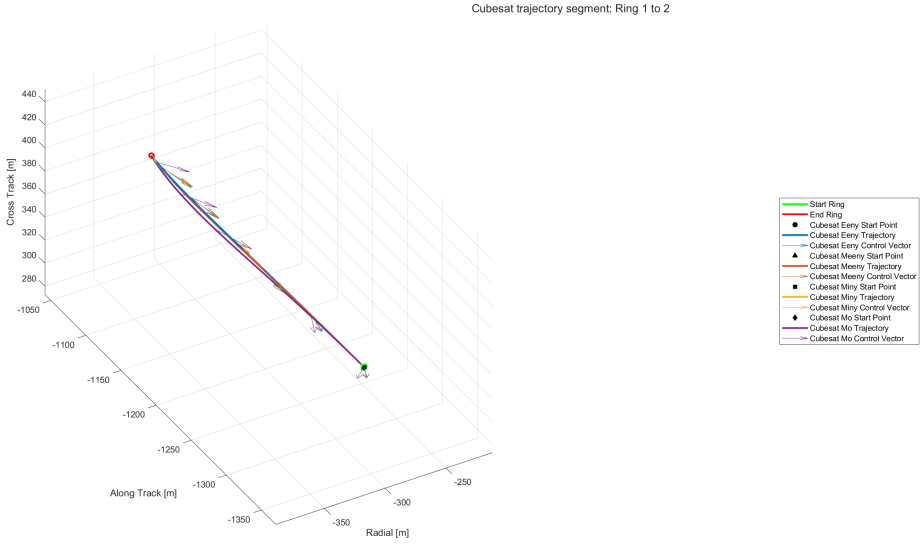

Fig. 37 Scenario 2 Segment 1 to 2

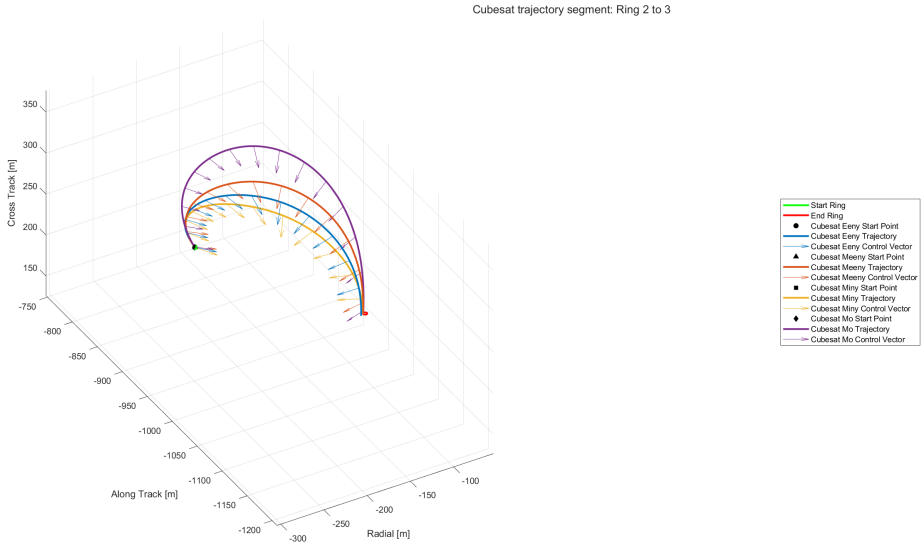

Fig. 38 Scenario 2 Segment 2 to 3

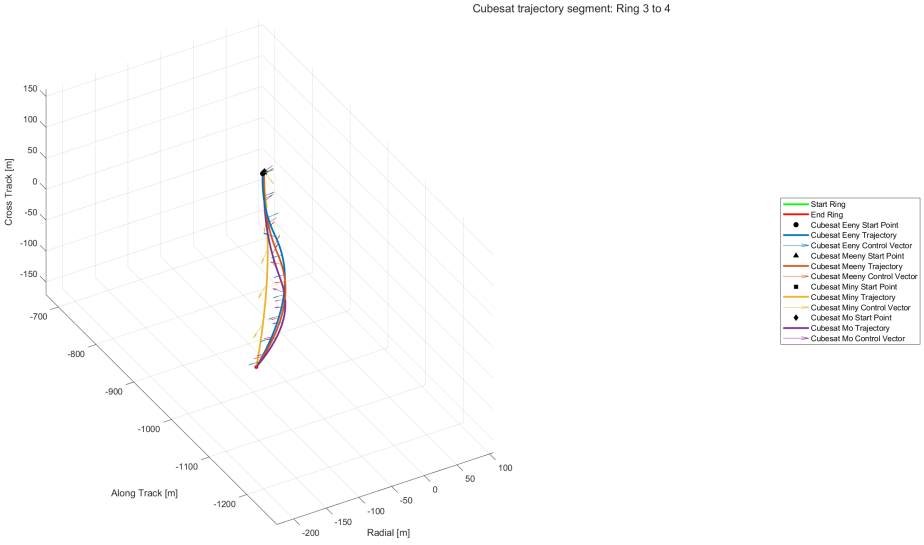

Fig. 39 Scenario 2 Segment 3 to 4

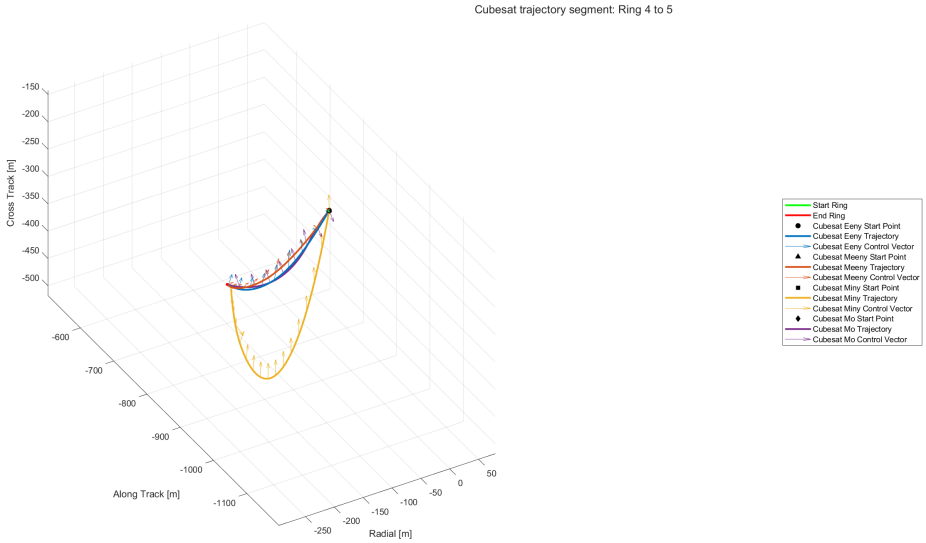

Fig. 40 Scenario 2 Segment 4 to 5

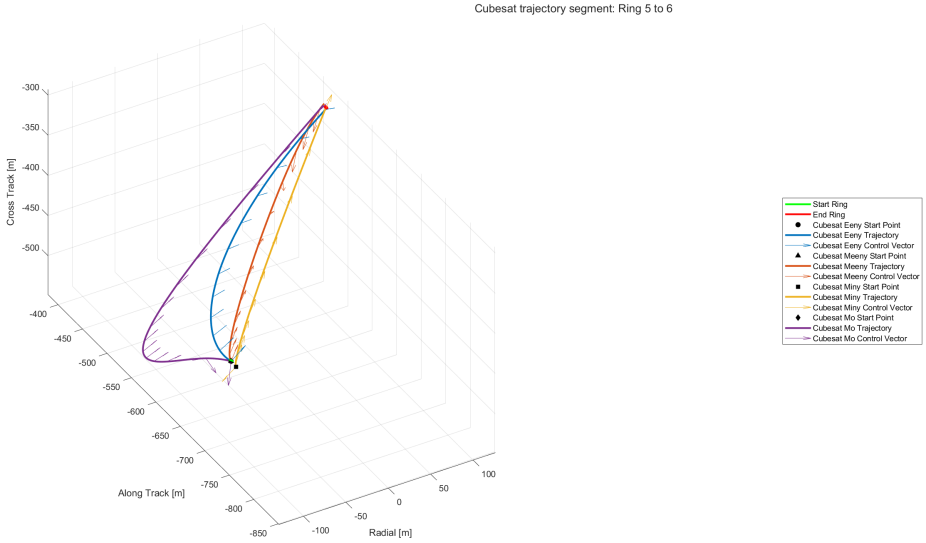

Fig. 41 Scenario 2 Segment 5 to 6

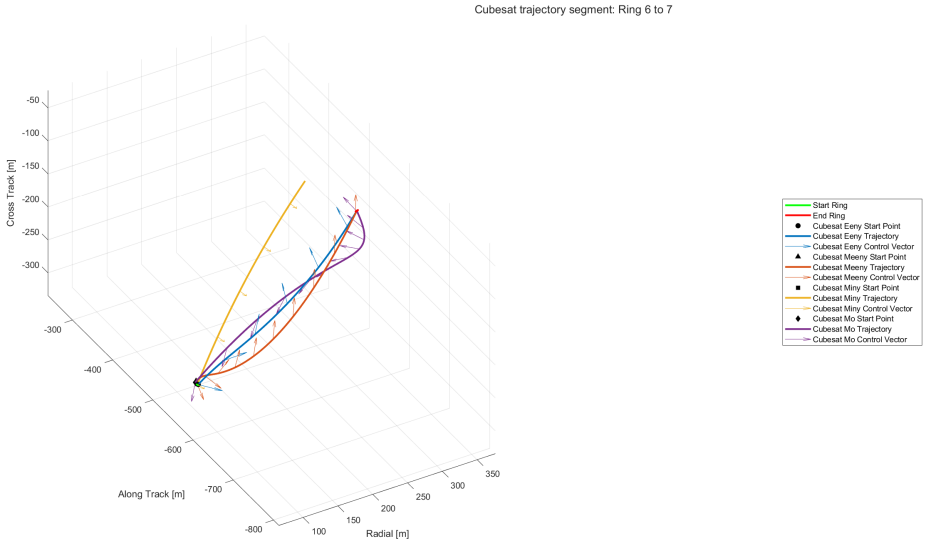

Fig. 42 Scenario 2 Segment 6 to 7

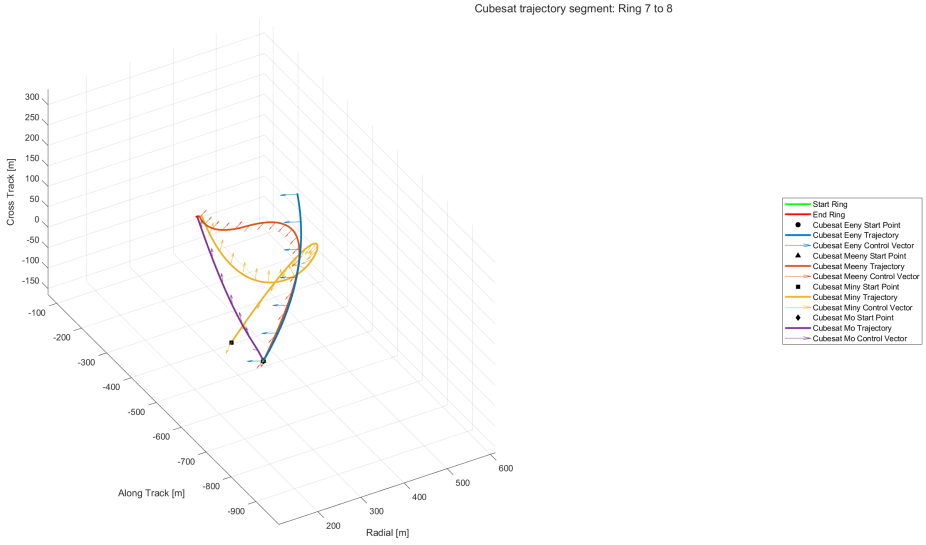

Fig. 43 Scenario 2 Segment 7 to 8

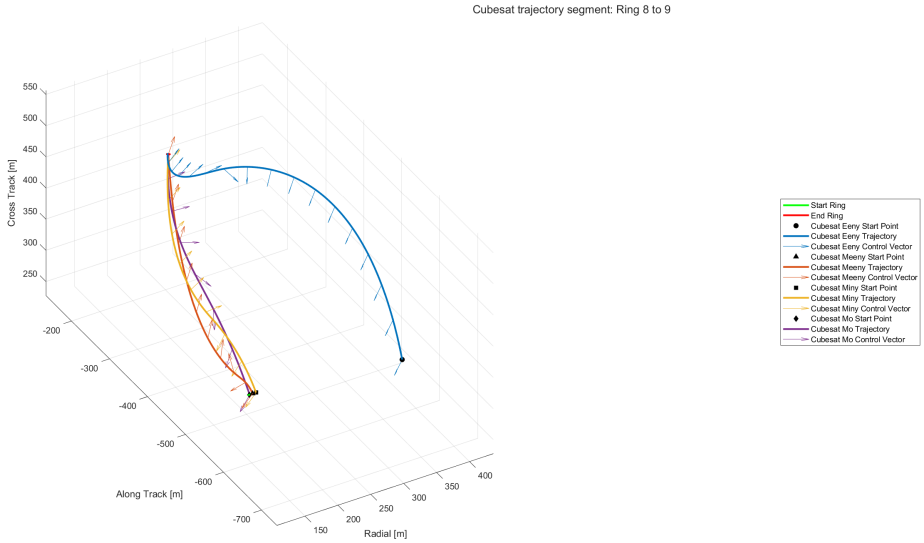

Fig. 44 Scenario 2 Segment 8 to 9

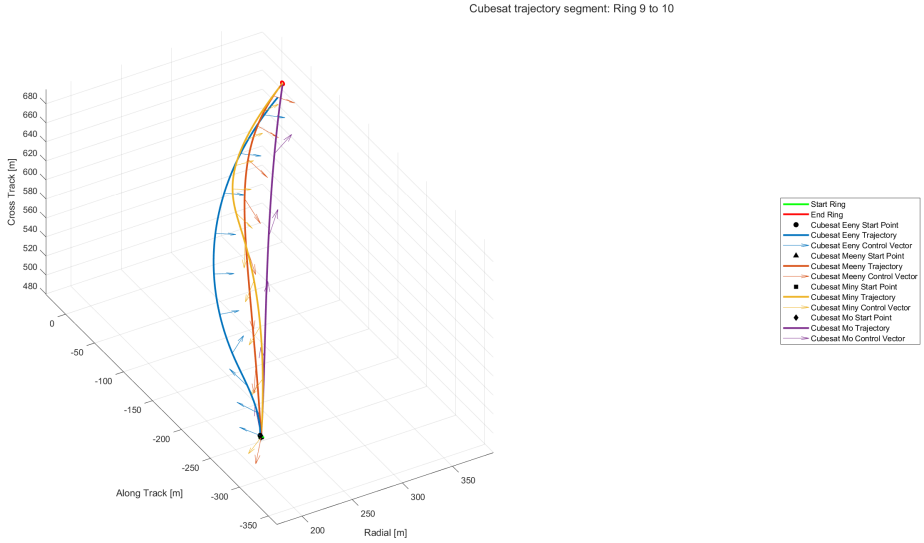

Fig. 45 Scenario 2 Segment 9 to 10

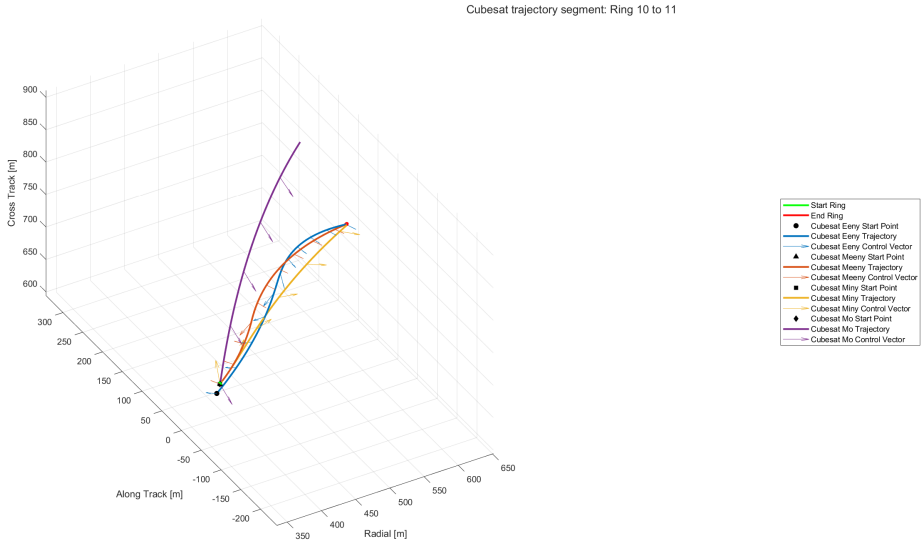

Fig. 46 Scenario 2 Segment 10 to 11

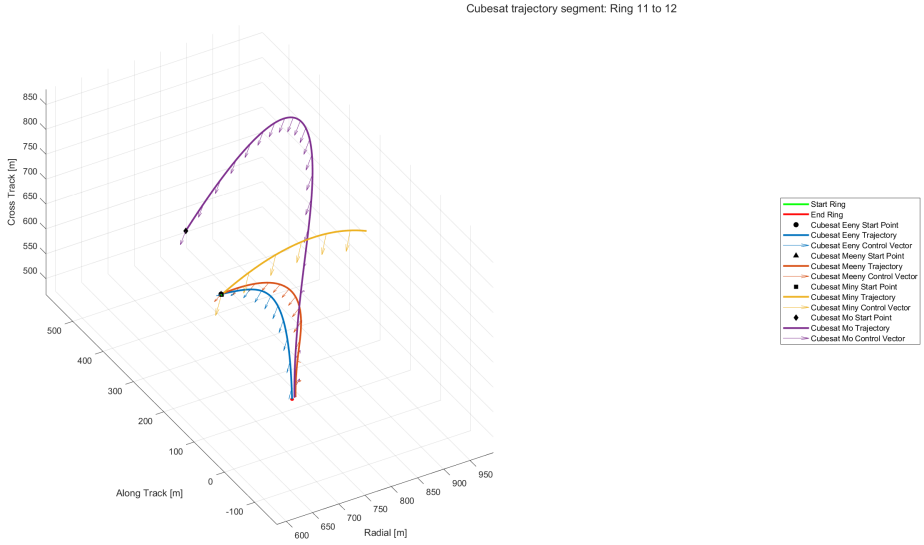

Fig. 47 Scenario 2 Segment 11 to 12

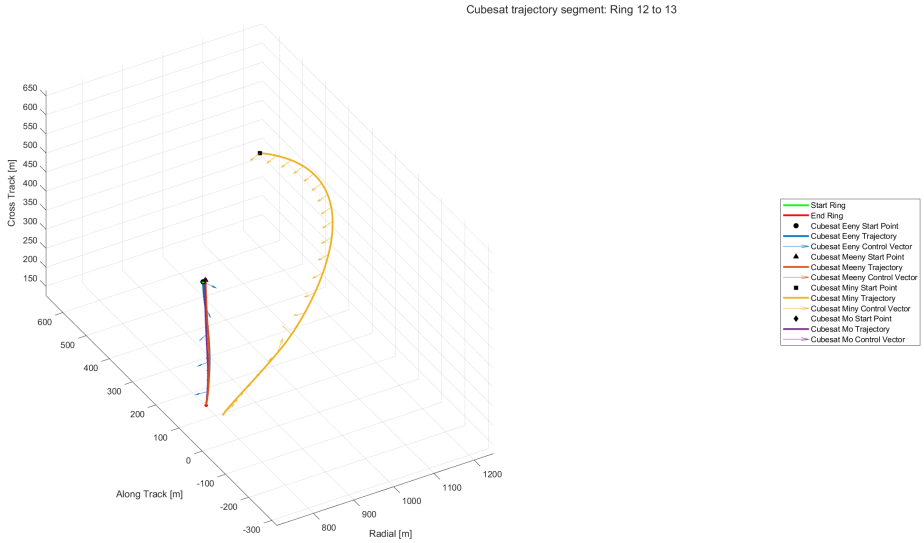

Fig. 48 Scenario 2 Segment 12 to 13

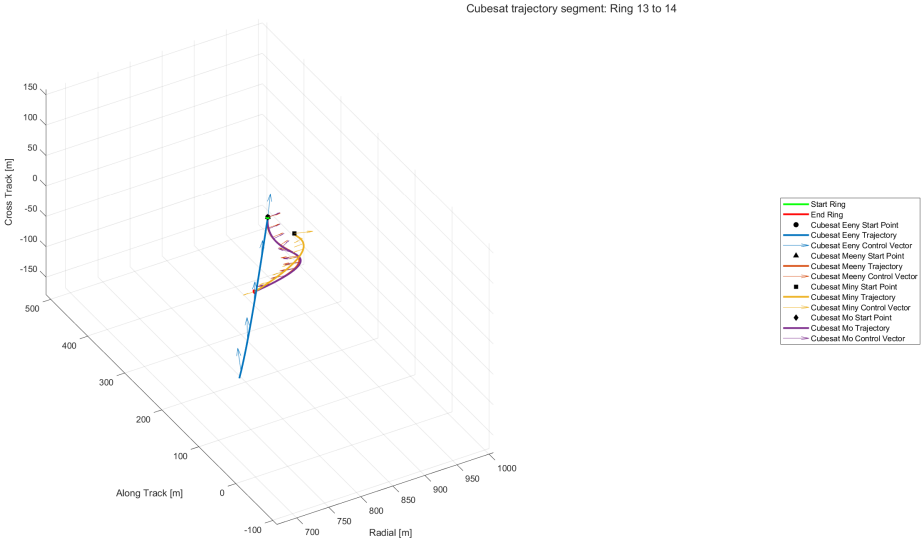

Fig. 49 Scenario 2 Segment 13 to 14

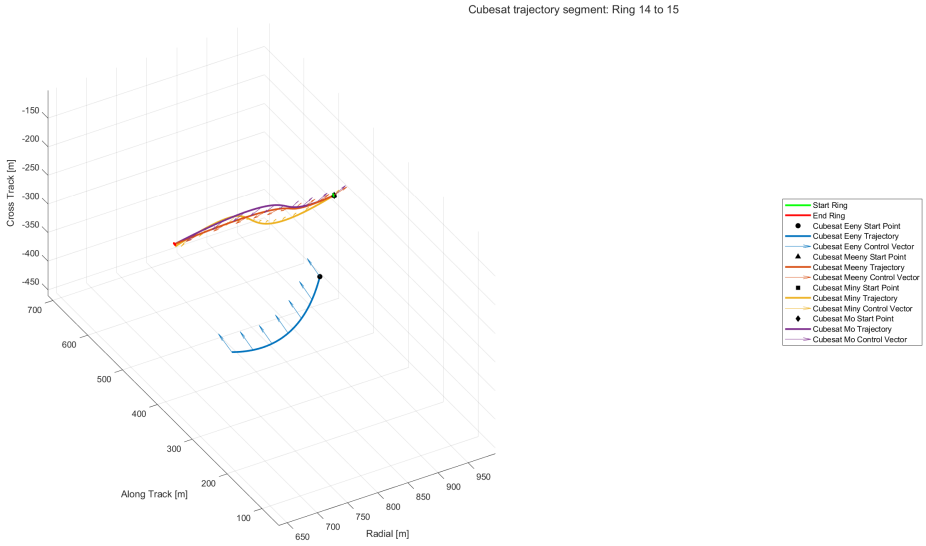

Fig. 50 Scenario 2 Segment 14 to 15

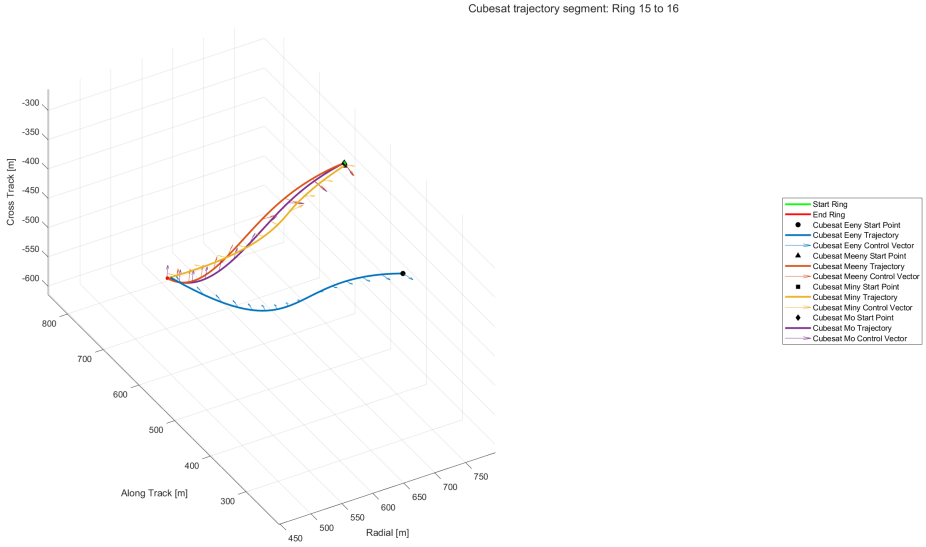

Fig. 51 Scenario 2 Segment 15 to 16

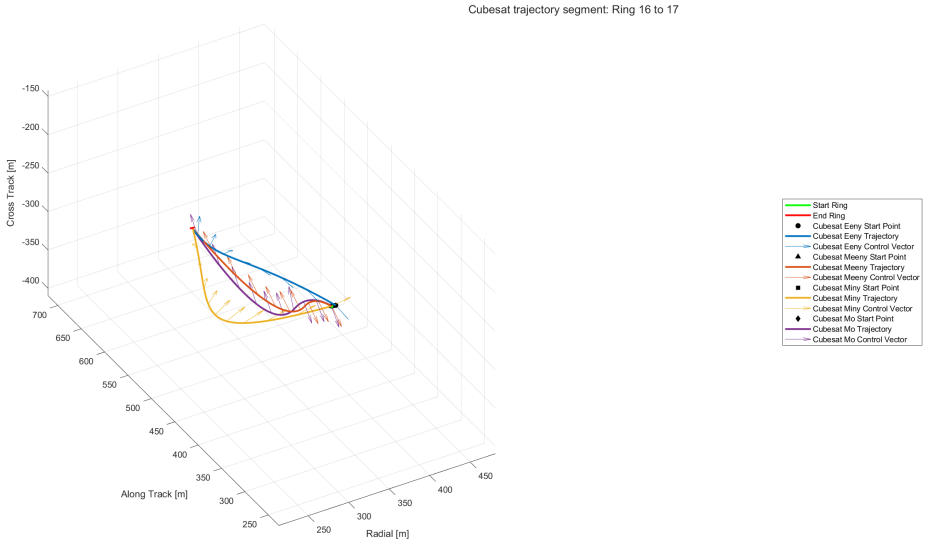

Fig. 52 Scenario 2 Segment 16 to 17

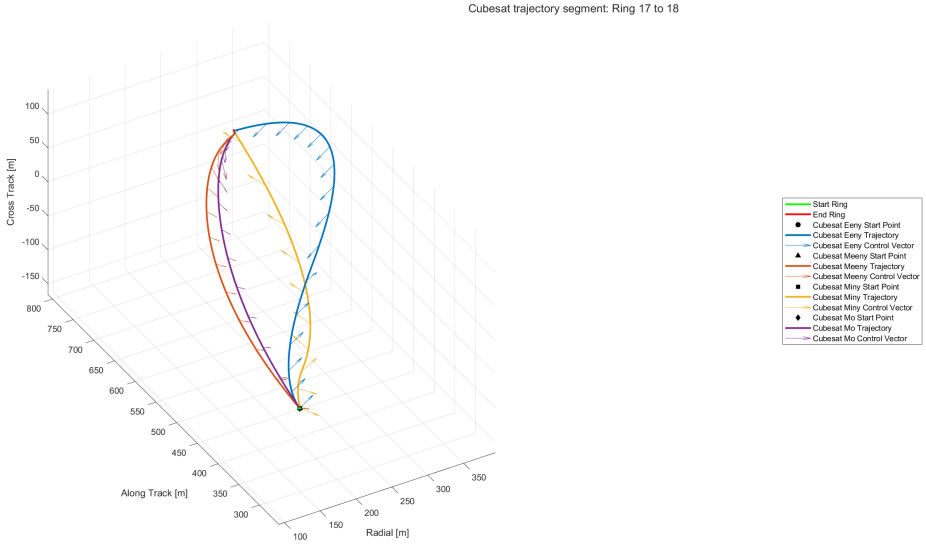

Fig. 53 Scenario 2 Segment 17 to 18

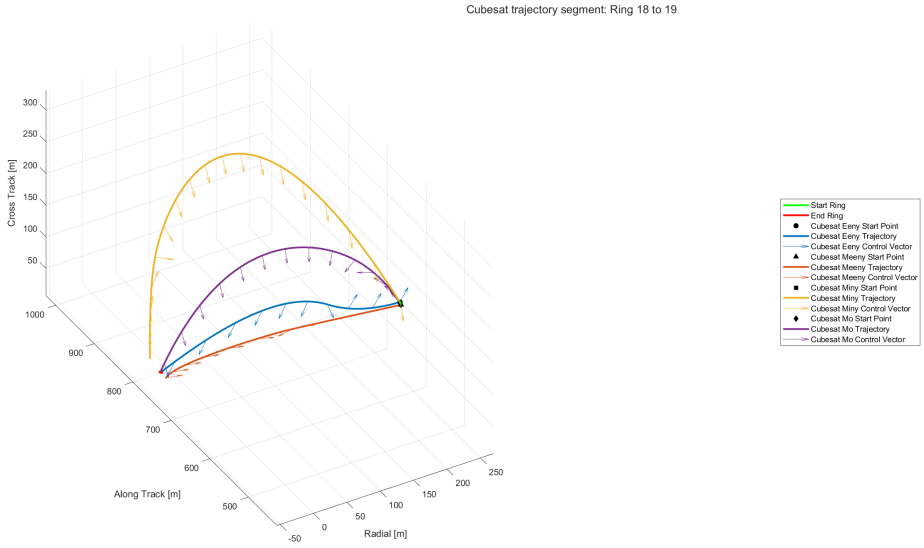

Fig. 54 Scenario 2 Segment 18 to 19

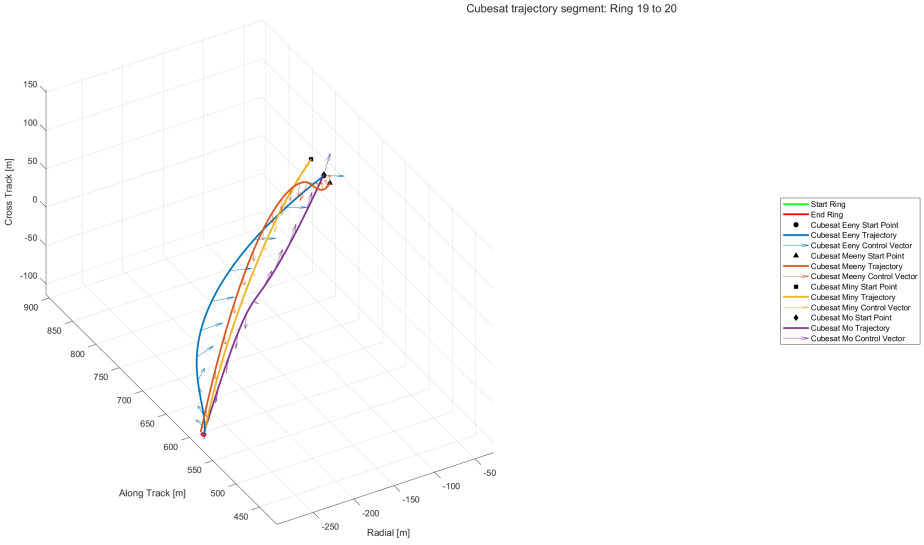

Fig. 55 Scenario 2 Segment 19 to 20

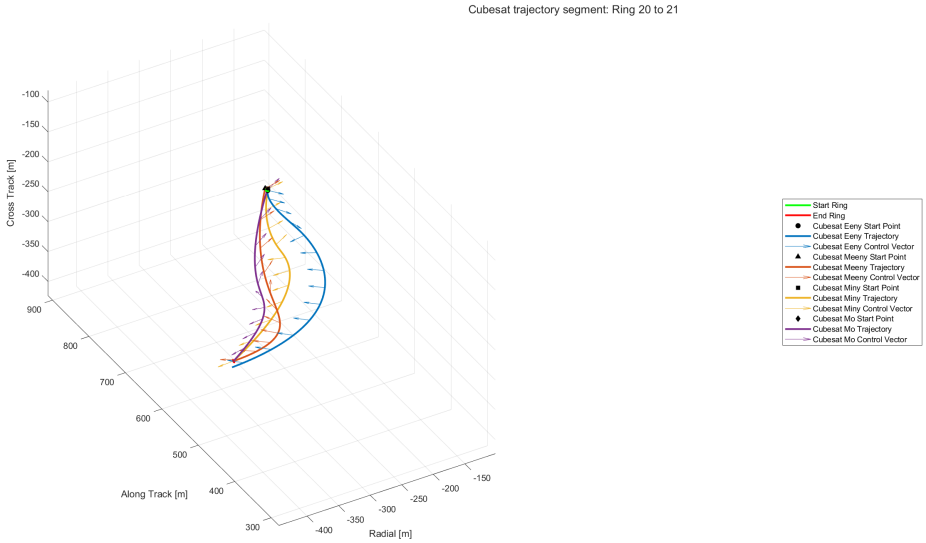

Fig. 56 Scenario 2 Segment 20 to 21

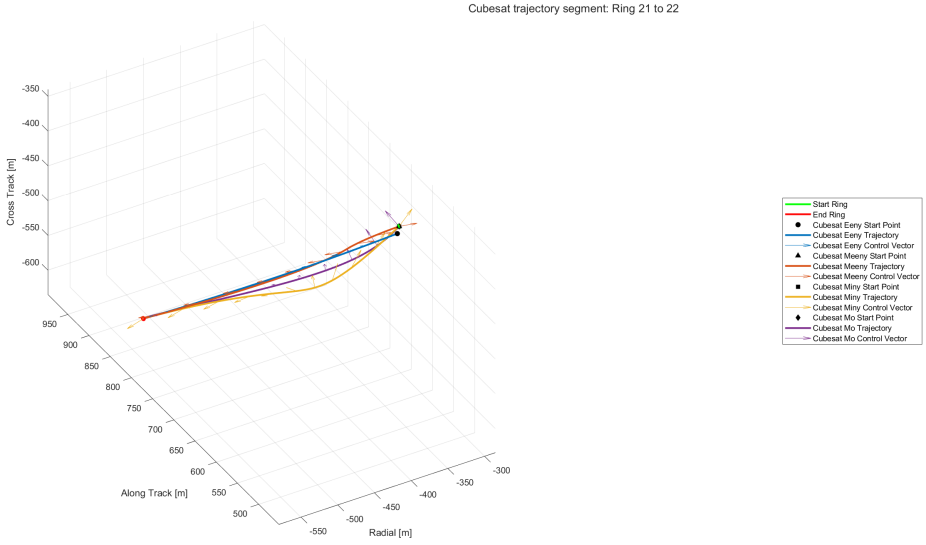

Fig. 57 Scenario 2 Segment 21 to 22

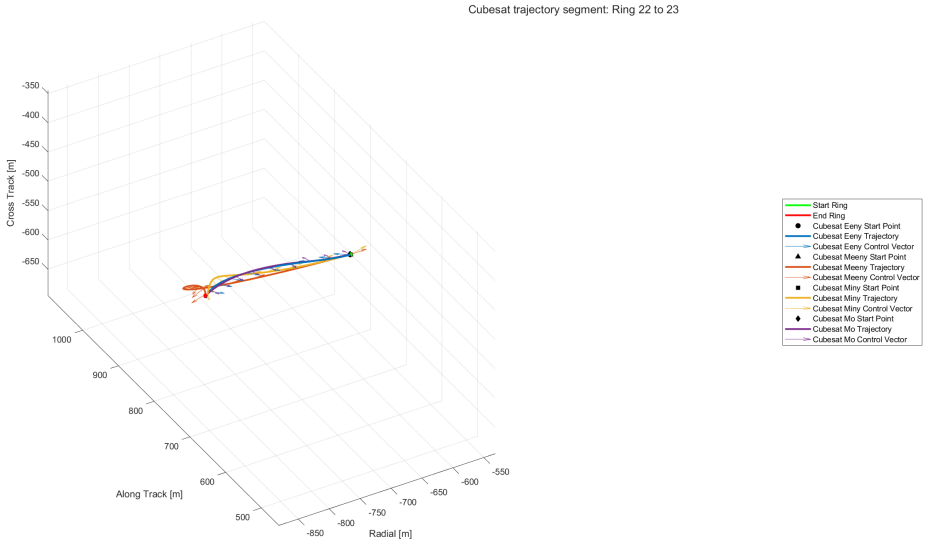

Fig. 58 Scenario 2 Segment 22 to 23

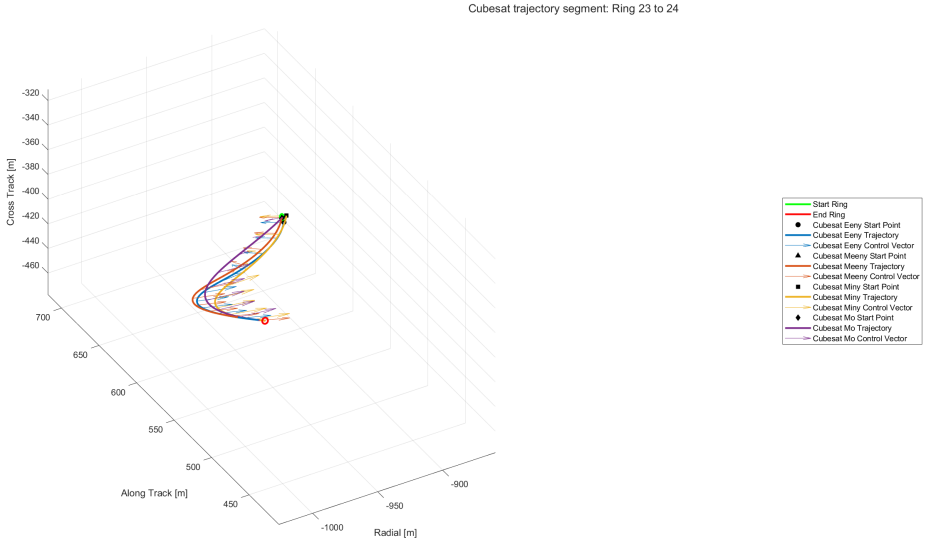

Fig. 59 Scenario 2 Segment 23 to 24

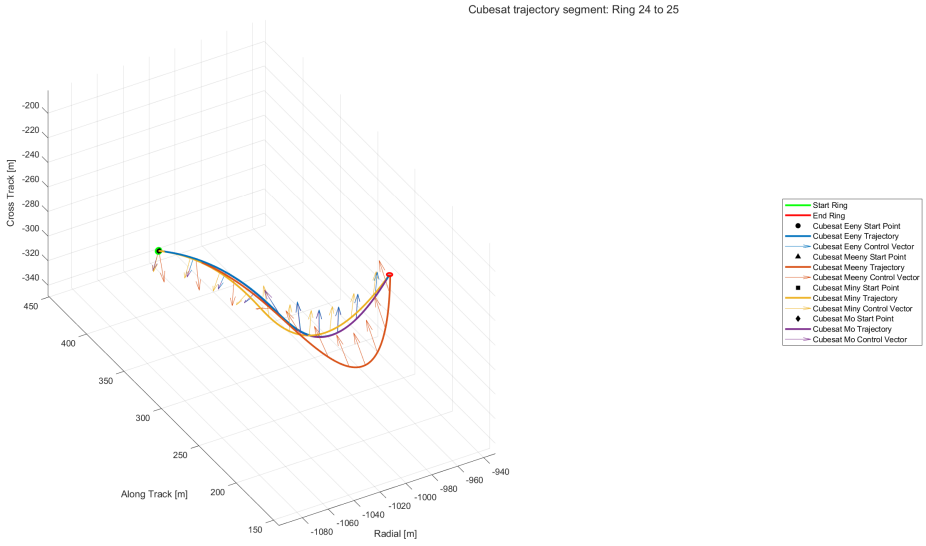

Fig. 60 Scenario 2 Segment 24 to 25

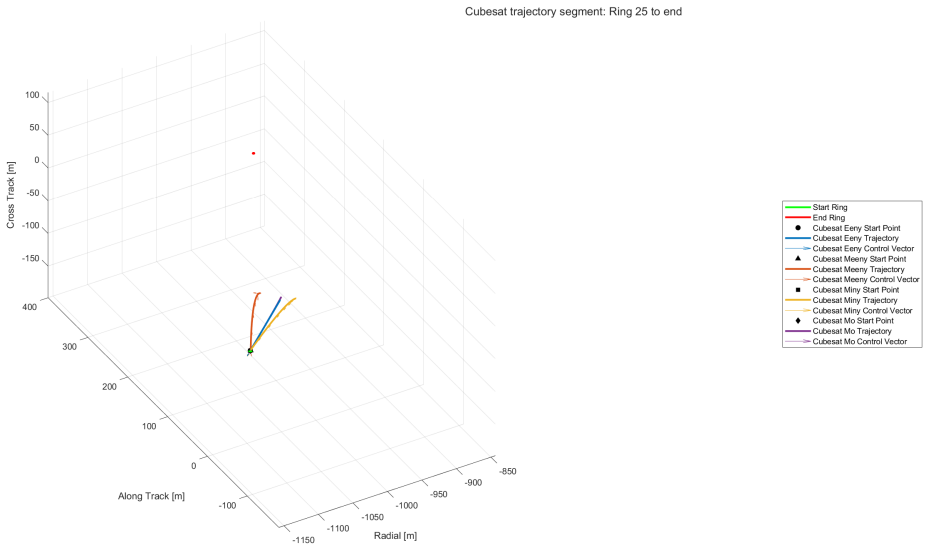

Fig. 61 Scenario 2 Segment 25 to end