Nonlinear Model Predictive Control for Spacecraft Rendezvous and Docking

a project presented to
The Faculty of the Department of Aerospace Engineering
San José State University

in partial fulfillment of the requirements for the degree
Master of Science in Aerospace Engineering

by
Ramon Medina

May 2026

approved by

Dr. Yawo Ezunkpe
Faculty Advisor



© 2026
Ramon Medina
ALL RIGHTS RESERVED

ABSTRACT

Nonlinear Model Predictive Control for Spacecraft Rendezvous and Docking

Ramon Medina

Spacecraft rendezvous and docking is a critical capability for modern space operations, enabling missions such on-orbit servicing, debris removal, and crew transfer. Traditional control strategies often rely on linearized dynamics, such as the Clohessy-Wiltshire-Hill equations, which limit accuracy and robustness during complex maneuvers or in non-circular orbits. This project presents the development and implementation of a Nonlinear model predictive control framework designed to automate the RVD maneuvers of a 3U cubesat with a cooperative target in Low Earth Orbit. A full six-degree-of-freedom nonlinear dynamic model is derived in the Local-Vertical Local-Horizontal reference frame, incorporating realistic actuator constraints for a thruster module and a pyramidal reaction wheel configuration. The control strategy is implemented in a high-fidelity MATLAB/Simulink environment, where the NMPC optimizes control inputs in real-time to minimize position errors while satisfying system constraints. Simulation results demonstrate the controller's effectiveness in guiding the chaser spacecraft from an initial hold point to a precise docking position. The system successfully converges to the target state, regulating both position and attitude despite initial nonlinearities.

Acknowledgements

I am deeply grateful to my siblings, Ruben, Silvia, and Elias, whose constant encouragement and unwavering belief in my goals provided the foundation I needed to persevere through the demanding phases of my master's studies. I could not have reached this milestone without their love and sacrifices. I am equally thankful to my father, who has always demonstrated to me that through hard work, anything is possible. I would also like to express my sincere gratitude to Dr.Yawo Ezunkpe for his guidance and unwavering support throughout the development of this project. Finally, I wish to express my profound gratitude to my late mother; it is because of her enduring influence that I am the man I am today

Table of Contents

ABSTRACT ..... **iii**
Acknowledgements ..... **iv**
List of Figures ..... **vii**
List of Symbols ..... **x**
1. Introduction **1**
1.1 Motivation 1
1.2 Literature review 2
1.2.1 Background context 2
1.2.2 Spacecraft Rendezvous and Docking - Equations of Motion 2
1.2.3 Role of Model Predictive Control 4
1.2.4 Past studies 4
1.2.5 Applications on cubesats 6
1.3 Project proposal 7
1.4 Methodology 7
1.4.1 Model dynamics 7
1.4.2 Formulation of the nonlinear MPC problem 8
1.4.3 Implement nonlinear MPC controller 8
1.4.4 Refinement of the nonlinear MPC 9
1.4.5 NMPC results and discussion 9
2. Dynamic modeling **10**
2.1 Vehicle model 10
2.2 Body Frame Orientation 14
2.3 Control Inputs 15
2.4 Equation of Motion 19
2.5 Simulink set-up 24
3. Nonlinear MPC problem **25**
3.1 Formulation of the nonlinear MPC problem 25
3.2 Simulink Set-up 26
3.3 Simulink first results 31
4. Implementation of the NMPC **43**
4.1 Force and moment availability 43
4.2 Adding atmospheric drag 46
4.3 Adding J2 gravitational perturbation 51
4.4 Modeling fuel consumption 53
4.5 Refining mission phases 56
5. Refining nonlinear MPC **59**

5.1 Refining reference trajectory..... 59
5.2 Line of sight constraint..... 59
5.3 Actuation allocation..... 64
5.4 Current results from increased model fidelity..... 68
6. Nonlinear MPC results and discussion..... 78
6.1 Improvements to simulation..... 78
6.2 Gain tuning..... 80
6.3 State noise robustness and MPC validation..... 95
6.4 Allocation Performance..... 99
6.5 Conclusion..... 104
7. Next steps..... 105
7.1 Recommended future work..... 105
8. References..... 106

List of Figures

1.1 Example of LVLH frame....3
1.2 Example of line of sight section.... 5
2.1 Thrust directions of the 3U Cubesat in the CPOD missions....10
2.2 Nanoavionics set of 4 reaction wheels....12
2.3 CAD model of 3U cubesat for plant model....13
2.4 Body frame with center of mass location....15
2.5 Top view of thruster configuration....16
2.6 Thrusters side view....16
2.7 Reaction wheel diagram....18
2.8 Earth centered inertial frame....20
2.9 Local-vertical local-horizontal frame....21
3.1 Simulation model dynamics and nonlinear model predictive controller....26
3.2 Full nonlinear simulation....30
3.3 First initial results of the nonlinear model predictive controller in the LVLH frame....31
3.4 First initial results of the nonlinear model predictive controller in the ECI frame....32
3.5 Debug results of the nonlinear mpc view 1 in the LVLH frame....33
3.6 Debug results of the nonlinear mpc view 2 in the LVLH frame....34
3.7 Debug results of the nonlinear mpc view in the ECI frame....35
3.8 Debug results of the nonlinear mpc: forces in the LVLH frame....36
3.9 Current results of the nonlinear mpc view in the LVLH frame....37
3.10 Current results of the nonlinear mpc moments in the LVLH frame....38
3.11 Current results of the nonlinear mpc forces in the LVLH frame....39
3.12 Current results of the nonlinear mpc velocity in the LVLH frame....40
3.13 Current results of the nonlinear mpc attitude in the LVLH frame....41
3.14 Current results of the nonlinear mpc position in the LVLH frame....42
4.1 Max actuation block....43
4.2 Thrust available, body Z-axis aligned with LVLH Z-axis....44
4.3 Thrust available, body Z-axis aligned with LVLH Y axis….....45
4.4 Thrust available, body Z-axis aligned with LVLH X axis....46
4.5 Results of the NMPC with atmospheric drag....48
4.6 NMPC position in the LVLH frame with atmospheric drag....49
4.7 NMPC velocity in the LVLH frame with atmospheric drag....49
4.8 NMPC cubesat attitude in the LVLH frame with atmospheric drag....50
4.9 NMPC cubesat forces in the LVLH frame with atmospheric drag....50
4.10 NMPC cubesat moments in the LVLH frame with atmospheric drag....51
4.11 Changes to equation of motion block....53
4.12 Changes to nonlinear model predictive controller block....54
4.13 Introduction of cubesat mass block....55
4.14 Overall changes to full nonlinear model....56

4.15 Example of V-bar approach....57
5.1 Cubesat orientation relative to LVLH frame....60
5.2 Example of cubesat docking [32]....61
5.3 Line of sight shaded region....61
5.4 Allocation block in simulation....66
5.5 Force comparison from allocation scheme....67
5.6 Moment comparison from allocation scheme....68
5.7 Trajectory when starting in phase 3....69
5.8 Forces from the homing phase....69
5.9 Moments from the homing phase....70
5.10 Position of chaser vehicle....71
5.11 Velocity of chaser vehicle....71
5.12 Attitude of the chaser vehicle....72
5.13 Phase change during simulation....72
5.14 Trajectory from second simulation run starting Phase 1 with line of sight constraint....73
5.15 Line Trajectory starting in Phase 1....74
5.16 Chaser position starting from phase 1....75
5.17 Chaser velocity starting from phase 1....75
5.18 Chaser Force input starting from phase 1....76
5.19 Phase change during simulation…….....76
6.1 Results when starting in phase 3 after gain tuning....82
6.2 Docking results from phase 3....83
6.3 Phase transition during phase 3 simulation....84
6.4 Phase 3 position convergence to docking position....84
6.5 Forces from phase 3 mission....85
6.6 Moments from phase 3 mission....85
6.7 Velocity from phase 3 mission....86
6.8 Chaser attitude from phase 3 mission....87
6.9 Results when starting in phase 2 after gain tuning....88
6.10 Docking results from phase 2....89
6.11 Phase transition during phase 2 simulation....90
6.12 Phase 2 position convergence to docking position....91
6.13 Forces from phase 2 mission....92
6.14 Moments from phase 2 mission....92
6.15 Velocity from phase 2 mission....93
6.16 Chaser attitude from phase 2 mission....94
6.17 Trajectory comparison between ideal and noisy state vector....96
6.18 Close up of docking with sensor noise....97
6.19 Computational solving time comparison....98
6.20 Signal routing for allocation into the loop....99

6.21 Nominal vs allocation actuation trajectory.....100
6.22 Close of up of docking with allocation in the loop.....101
6.23 Thruster commands.....102
6.24 Force produced with allocation in the loop.....102
6.25 Moment produced with allocation in the loop.....103

List of Symbols

Symbol	Definition	Units(SI)
$\mathbf{A}_{rw}$	RW torque body frame transformation	-
A_{drag}	Cross-sectional Area for drag	m^2
h_i	Normal unit vector of area for chaser cubesat	-
A	Linearized state matrix	-
B	Linearized input matrix	-
$\mathbf{B}_{act}$	Mapping matrix for actuators for control input in the LVLH frame	-
$\mathbf{P}_{J2}$	J2 effect acceleration in the ECI frame	m/s^2
$\mathbf{R}$	Satellite position vector in ECI frame	m
r	Position vector in the LVLH frame	m
a_{drag}	Acceleration due to atmospheric drag	m/s^2
a_{Pert}	Perturbation acceleration	m/s^2
a_{J2}	Differential acceleration from J2 effects on the chaser and target cubesat	m/s^2
d	Thruster moment arm length	m
F_{Thrust}	Thrust force vector in the LVLH frame	N
$M_{(xyz),B}$	Moment vector in the Body/LVLH frame	Nm
I	Moment of inertia tensor	$\mathrm{kg} \cdot m^2$
m_c	Mass of the chaser spacecraft	kg
N	Prediction steps	-
Q	MPC state weighting matrix	-

$\mathbf{Q}_{al}$	Allocation scheme error weighting matrix	-
$\mathbf{q}$	Quaternion vector	-
R	MPC control input weighting matrix	-
$\mathbf{R}_{al}$	Allocation scheme control effort weighting matrix	-
R_c	Chaser position vector in ECI frame	-
R_t	Target position vector in ECI frame	-
S	MPC terminal weighting matrix	-
u	Control input vector	[N, Nm]
v_{rel}	Relative velocity magnitude	m/s
V_{max}	Maximum velocity limit per mission phase	m/s
$\mathbf{v}_{cmd}$	Velocity command vector for the reference trajectory	m/s
J_2	Second zonal harmonic coefficient of Earth's gravitational field	m/s^2
$\mathbf{w}$	State noise vector	[m,m/s,rad,rad/s]
$\mathbf{x}$	State vector	[m,m/s,rad,rad/s]
$\mathbf{x}_{ref}$	Reference state vector	[m,m/s,rad,rad/s]
J	Total objective cost function to optimize over the finite prediction horizon window.	-
J_{al}	Total objective cost function for actuator allocation	-

H	Hessian matrix for MPC quadratic programming problem	-
H_{alloc}	Hessian matrix for allocation quadratic programming problem	-
$\mathbf{S_u}$	Forced response prediction matrix	-
$\mathbf{S_x}$	Free response prediction matrix	-
$\mathbf{c}^T$	Linear term vector for MPC quadratic programming problem	-
$\mathbf{f}^T$	Linear term vector for allocation quadratic programming problem	-
C_{los}	Constraint matrix from line of sight cone	-
C_{mod}	Linear mapping matrix used to isolate vehicle position components for line of sight constraints	-
g_0	Standard gravity at Earth's surface	m/s^2
C_D	Drag coefficient for 3U satellite	-
$\mathbf{g}$	Actuator array (thrust and reaction wheel torque commands)	[N,Nm]
$\mathbf{T}_{rw}$	Total torque from reaction wheels in body frame	Nm
$\mathbf{X}$	Predicted state vector over the prediction horizon	m

$\mathbf{U}$	Sequence of control inputs (force and moment) over the prediction horizon	[N.Nm]
$\mathbf{Q}_{big}$	Stacked MPC state weighting matrix over the prediction horizon	-
$\mathbf{R}_{big}$	Stacked MPC control input weighting matrix over the prediction horizon	-
Greek Symbols		
$\beta,\gamma,\phi,\alpha,\theta$	Geometric thruster mounting angles	deg
ζ	Reaction wheel cant pivot angle	deg
n	Target orbital rate	rad/s
θ_{los}	Line of sight constraint half angle	deg
μ	Standard gravitational parameter of Earth	m^3/s^2
ρ_{atm}	Atmospheric density	kg/m^3
τ	Torque vector	N · m
ω, ω_{body}	Angular velocity of the LVLH frame / Body Frame	rad/s
Ω	Skew-symmetric matrix of angular velocity	deg
Acronyms		
CWH	Clohessy-Wiltshire-Hil	-
ECI	Earth-Centered Inertial	-
LVLH	Local-Vertical Local-Horizonta	-
MPC	Model Predictive Control	-
NMPC	Nonlinear Model Predictive Control	-
RVD	Rendezvous and Docking	-
LOS	Line of Sight	-

1. Introduction

1.1 Motivation

Spacecraft rendezvous and docking (RVD) is a complicated sequence of maneuvers that are meant to bring two spacecraft physically together in the same orbit. In this maneuver one spacecraft is deemed “the chaser” and the other “the target”. The chaser spacecraft is the one performing actuations in order to rendezvous with the target spacecraft. This maneuver involves complex control systems in order to be successful. NASA conducted its first rendezvous and docking between two spacecraft in 1966. This mission was between Gemini VIII and the Agena spacecraft [1]. Although it had a control system onboard it needed assistance from the astronauts to successfully complete the maneuver [2]. Since 1966, advances in space technology and operations have paved the way to more robust control systems that operate much more autonomously. Currently, spacecraft rendezvous and docking represents a mission-critical phase in modern space operations. These operations include in-orbit satellite servicing, debris removal, space-station resupply, and crew transfer.

Although rendezvous and docking has become essential to modern space operations, there are still studies being conducted to optimize this maneuver. Model predictive control (MPC) has become a popular approach to fully automate and optimize space operations such as rendezvous and docking. Its popularity comes from the robustness and flexibility it offers to spacecraft missions The variables that many studies tend to optimize deal with minimizing fuel consumption and trajectory time. Other studies have been used to increase the robustness of the controllers used for this maneuver. The need for robustness comes from the need to fully automate these operations. Model predictive control would be beneficial to current projects such as CalTech’s AAReST, where cubesats would autonomously in orbit assemble their telescope [3]. Although a MPC algorithm would improve any space operation, much is still left to be developed to increase the robustness of the algorithm. Most common uses of model predictive control for rendezvous and docking have been with linear models to avoid computational cost and guarantee a convergence for its solution. This tradeoff leads to performance loss and loss in flexibility in its control strategy. One way to improve this is by focusing on development on a nonlinear model predictive control strategy. Nonlinear model predictive control would increase computational cost but with advances in hardware it is becoming more feasible to implement on spacecraft. Nonlinear Model predictive control would provide the robustness and more accuracy needed for an autonomous rendezvous and docking maneuver for any spacecraft.

1.2 Literature review

1.2.1 Background context

Autonomous rendezvous and docking (RVD) has been a key technology for spacecraft missions. It is essential for missions involving in-orbit servicing and space debris removal [4]. It is also essential to the International Space Station (ISS) when transferring crew and crew supplies. The RVD maneuver is split up into three phases [5]. The first phase is the rendezvous phase in which the chaser spacecraft transfers over to the phasing orbit in preparation for the next phase. The next phase is the close proximity operations phase, in this phase the chase vehicle closes the distance between the target and itself [5]. This is a crucial phase in a RVD maneuver as it consists of hold points for system checks and trajectory updates [5]. The final phase is the docking phase in which the chaser vehicle performs maneuvers to physically connect with the target vehicle [5]. Although there have been many control methods that could be applied to spacecraft rendezvous missions, model predictive control (MPC) has recently become of interest in many studies [6]. The application of model predictive control has the advantage of handling input and state constraint while also allowing the flexibility of updating plant parameters and objectives [6]. This flexibility comes at a cost in the form of computational time and effort [6]. Through a series of advancements in computational hardware and more efficient algorithms, implementation to spacecraft missions has become more applicable [6].

1.2.2 Spacecraft Rendezvous and Docking - Equations of Motion

In order to understand the problem being tackled by the MPC, it is crucial to understand the dynamics involved in the rendezvous and docking maneuver. Model predictive control requires a predictive model to accurately produce a good controller. One way to model the dynamics between a chaser and target is using Newton’s laws in an inertial reference frame [6]. Using Newton’s laws, the nonlinear dynamics can be retrieved between the two spacecraft [6]. Also Gauss’s equations using Keplerian orbital elements can be used to model the dynamics instead of Newton's laws [6]. Many applications of MPC for rendezvous and docking between spacecraft translate the dynamics from the inertial frame into the local-vertical, local-horizon (LVLH) frame [6].

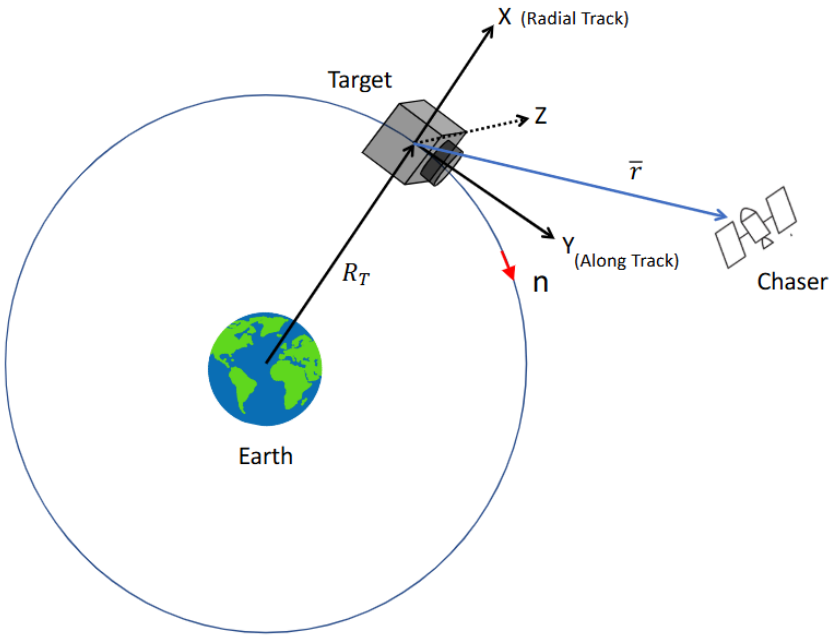


Figure 1.1 - Example of LVLH frame [7]

Figure 1.1 shows how the LVLH frame is a reference frame with an origin at a specific point on the target spacecraft [7]. The x-axis points from the origin on the Earth to the center of the spacecraft, while the y-axis points in the direction of the spacecraft's velocity [7]. In this reference frame, the equations of motion between the spacecraft can be linearized into the Clohessy-Wiltshire-Hill (CWH) equations [6]. These equations translate the linearized translation dynamics of the chaser vehicle with respect to the target vehicle [8]. Although they are linearized equations they are often still used in the implementation of linear and nonlinear model predictive control. It is important to know when the CWH equations are used as they come with many tradeoffs and assumptions that limit their effectiveness for rendezvous and docking missions. One study states that using model prediction with CWH can lead to significant errors in the prediction if the eccentricity is not perfectly zero [9]. One drawback to the linearization of these equations is that the CWH equations assumes external forces such as atmospheric drag and solar radiation pressure [9]. Another drawback in keeping CWH equations is that they become increasingly inaccurate over longer periods of time [6]. More robustness is needed to be added to MPC in order to improve the controller [6]. The reason many studies choose to linearize the equations motion for their MPC is to simplify the problem and reduce computational cost [6]. In some studies, hybrid models are used for close proximity operations when certain phases in maneuvers are linearized [10].

1.2.3 Role of Model Predictive Control

The basic concept of model predictive control is to use a dynamic mathematical model to predict the system behavior and then to optimize the prediction to derive the best control input [11]. At each time step the model predictive control solves an open-loop optimal control problem [12]. The optimal control problem is a cost function used to specify optimization of the model states and control inputs [12]. Along with this control problem, constraints can be added in on the acceptable state outputs and control inputs. This gives flexibility in the control algorithm approach unlike classical control methods. In addition to constraints, a model prediction control can also control the length of its prediction. Essentially, the cost function works to optimize over a certain number of time steps, forecasting the systems behavior [12]. This is important as it can affect the stability and the computational cost of the controller [12]. Model predictive control concepts can be used on linear and nonlinear dynamics models. As mentioned in the section before, the linearized models have their drawbacks when it comes to accuracy of the controllers prediction.

Both linear and nonlinear MPC use the same approach which requires a prediction model, constraints, and a cost function [12]. Distinguishing between the two approaches means dealing with a nonlinear prediction model or with nonlinear cost function and constraints [12]. One study compared a nonlinear predictive controller to a linear predictive controller. Although this study was not conducted on a spacecraft rendezvous and docking maneuver, it was able to compare the two approaches. This study involves underactuated unmanned surface vehicles and concludes that in this application the nonlinear MPC was the best choice [13]. It did also state initially that the nonlinear MPC had an extremely high computational time compared to the linear MPC [13]. By using a CasADi solver and with C++ as the programming language, the study concluded that nonlinear MPC was feasible to use for real-time application achieving a solution time of 10 milliseconds [13]. Choosing the correct solver for the cost function an MPC uses is highly important to the success of the controller.

1.2.4 Past studies

Past studies mention other forms of control algorithms used to optimize an autonomous rendezvous and docking mission. For example, genetic algorithms have been used in the past to optimize fuel and time of a trajectory for spacecraft rendezvous and docking [9]. It was determined that genetic algorithms were not optimal for the maneuver as they took up too much computational time for any real-time docking systems [9]. Furthermore phase-plane controllers have been implemented for maneuver only to find that this strategy is not suitable for path constraints [9]. One study conducted a comparison of classical control methods against an optimization based MPC control strategy for rendezvous [14]. The study showed that the MPC strategy showed more improvement in minimizing the propellent consumption of the rendezvous maneuver when compared to classical control methods [14]. In one simulation conducted, they found that the difference in propellant used between MPC strategy and classical control methods was roughly 35 kg [14].

Most studies reviewed explored a variation of the MPC application to the rendezvous and docking maneuver using a linearized prediction model. The reason for this is that dealing with nonlinear dynamics leads to utilizing nonlinear programming methods that do not guarantee convergence and require complex numerical solvers [4]. Many have gone the route of convexifying all elements of the nonlinear problem in order to produce real time globally optimal solutions [4]. One particular framework that has risen recently is the implementation of a linear quadratic MPC, which has been used in studies involving rendezvous and docking with obstacle avoidance [4].

Recent applications of MPC onto spacecraft rendezvous and docking problems have led to additional constraints being added to studies. One paper formulated a MPC to handle thrust constraints, Line-of-Sight (LOS) constraints, and a soft docking constraint [15]. In this paper, the thrust constraints were nonlinear inequalities, that in order to work with a linear quadratic MPC, needed to be linearized [15]. The paper mentions that if left nonlinear, then a more complex solver would be needed and that would require more time and resources that were available [15]. Line of sight constraints define the section where the chaser and target are meant to dock [15].

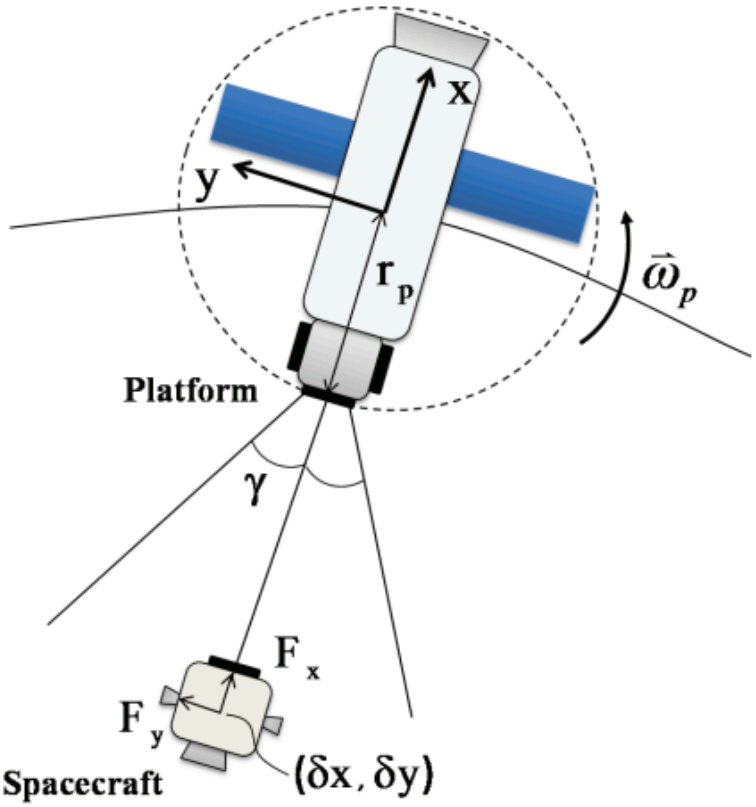


Figure 1.2 - Example of line of sight section [15]

The line of sight constraint is meant to constrain the chaser target in the vicinity of the cone intersection shown in figure 1.2. For this study the line of sight constraints are also

linearized [15]. In some MPC applications, this constraint is applied only when the chaser spacecraft is within a certain distance preparing for the docking phase. As for the soft docking constraint, this constraint has been introduced lately in MPC rendezvous and docking papers. The soft docking constraint is a constraint meant on the relative velocity of spacecraft when near the docking port [15]. This constraint has been added to many MPC applications for spacecraft proximity operations to avoid any excessive mechanical shock during the operation [15]. In this study this constraint is also linearized with the small penalty in performance. [15].

One study for rendezvous and docking of a spacecraft developed both a linear MPC controller and nonlinear MPC controller for comparison [7]. In this study both the nonlinear and linear MPC use the same constraints and cost function [7]. However, the dynamics for their predictive model is not the same. The linear MPC uses the Clohessy-Wiltshire-Hill equations, while the nonlinear MPC uses the nonlinear equations of motion [7]. The linear MPC required a standard quadratic programming algorithm to solve its optimal control problem [7]. While the nonlinear MPC required using a fmincon function with a Sequential quadratic programming algorithm to solve its optimal control problem [7]. Both approaches were solved using MATLAB/Simulink environments. The final conclusion of this study was that the nonlinear MPC performed much better than linear MPC but at the expense of computational time [7].

The computational cost of using nonlinear MPC has been outweighing the benefits it delivers when it comes to improved accuracy and flexibility with constraints. The nonlinear MPC approach requires more efficient solvers to reduce the computational time and effort. One approach presented in [16] uses a nonlinear programming solver IPOPT (interior point optimizer) to solve a nonlinear MPC problem [16]. IPOPT is an open source software package used to solve large-scale nonlinear optimization problems. Since it is open-sourced, it is readily available to be used by the public. This study also incorporates the Clohessy-Wiltshire-Hill equations but since the target spacecraft is rotating at a constant rate, the dynamics become coupled and nonlinear [16]. Also in this paper, they explored implementing constraints that switch during maneuvering phases of the chaser spacecraft [16]. This explores the flexibility of the nonlinear MPC approach. With the nonlinear MPC approach, constraints do not need to be fixed nor limited to linearity. The nonlinear MPC was tested on a MATLAB/Simulink simulator and an air-bearing test bed [16]. The conclusion of the paper was that the nonlinear MPC approach was capable of generating a solution within 0.1 seconds [16]. This does not seem very quick as the authors support further development and implementation of the nonlinear MPC approach [16].

Although the need for computation efficient nonlinear MPC approach continues, there are still many needs to be addressed. In the previous study [16], the nonlinear MPC development could extend out to handle tumbling docking targets and modeling atmospheric drag and solar pressure for improvement in robustness.

1.2.5 Applications on cubesats

The difficulty of using a model predictive controller on cubesats is the computational load does not align with the computer resources available on most cubesats [8]. This study proposes not to use model predictive control by developing optimization subproblems for fuel

optimization of an autonomous rendezvous and docking maneuver [8]. It was able to decouple the problem to two optimization subproblems for orbit energy and attitude fuel [8]. Although this was not implemented via an MPC, it shows a hill needed to be climbed for a nonlinear MPC to work effectively on cubesats. One contribution this study had on this literature review is the addition to reaction wheel control inputs not found in the other papers involving spacecraft rendezvous and docking [8]. Most studies primarily focused on the thruster inputs and optimization of fuel consumption with their implementation of spacecraft dynamics. It seems with cubesats reaction wheels are additional control inputs commonly used.

Although implementing any MPC on a cubesat is a challenge it does not mean that it has not been attempted and been successful. One article presents the challenge of implementing a MPC for a cubesat to perform rendezvous and docking with a larger spacecraft [17]. The article did many of the same strategies from previous studies such as linearizing the relative motion between the chase and target dynamics via the CWH equations [17]. Though it uses linearized dynamics, it does implement nonlinear characteristics. Some of the nonlinear characteristics include using nonlinear environmental disturbances and actuator emulation [17]. In order to apply to cubesat the study uploaded its algorithm on a micro-controller based on an ARM-9 architecture which is a common type of processor for small satellites [17]. This implementation was done to show real-time implementation is possible for cubesats [17]. The study concluded by demonstrating an execution time of 25 milliseconds and a 100% success rate within 2 centimeters of accuracy [17]. The results of that study showed that the application of MPC with nonlinearities on a cubesat is a feasible and viable approach to the rendezvous and docking problem.

1.3 Project proposal

Based on the literature review the objective of this MSAE project is to develop and implement a robust nonlinear model predictive control for cubesat rendezvous and docking with a cooperative target. The project will focus on using a nonlinear dynamic model between a chaser and target cubesat in Low Earth Orbit (LEO) as the predictive model in its model predictive control approach. This aims to build upon previous studies with the addition of modeling of the atmospheric drag and the J2 gravitational perturbation effect that is present for cubesats in LEO. The project will also aim to incorporate an allocation scheme for the designed cubesat given the thruster and reaction wheel location.

1.4 Methodology

1.4.1 Model dynamics

This section will focus on development of the nonlinear dynamics between target and chaser cubesats. This will require creating a full 6-DOF dynamic model of the relative motion between the chaser cubesat and target cubesat. To avoid the high computational load that comes from using nonlinear dynamics, the predictive model will be linearized at each step. A simple analysis will be conducted at first using only the nonlinear dynamics and then atmospheric drag and the J2 gravitational perturbation will be added in later sections. The aim of the chapter is to set up the base equations of motion between a target vehicle and chaser vehicle in circular orbit

since both cubesats will be simulated in LEO. This section will also be where the CAD model for the chaser cubesat will be designed and defined. Any thruster and reaction wheel module sourced will be defined in this section. Thrust and moment equations will also be derived in this section for future use in the nonlinear MPC development.

1.4.2 Formulation of the nonlinear MPC problem

This section will focus on formulating the nonlinear MPC optimal control problem for rendezvous and docking scenarios. This section will develop the cost function that optimizes maneuver time, and explicitly incorporates sets of state and input constraints thrust and moment limits. This section will also cover the constraints that are planned to be imposed on the cubesat such as the thrust available and maximum torque the reaction wheels can achieve.

Formulation of the constraints will be not as difficult as developing the custom cost function to be implemented with the nonlinear MPC. Many different cost functions have been used in a variety of studies for rendezvous and docking so development of the cost function will need to be thorough and adjust the weights to ensure the optimization problem focuses on solving the mission requirements.

$$J = \sum_{i=1}^{N-1} (x_i^T Q x_i + u_i^T R u_i) + x_N^T S x_N \tag{1.1}$$

Equation 1.1 demonstrates the cost function that will be used in this report. A similar cost function can be found in model predictive control publication [11]. It incorporates the use of a penalty matrix as well. The term $\bar{x}$ is the error in the state vector and the term u_i is the control input vector at the current time step. The Q matrix will control the penalty state errors and the R matrix will control the penalty on control effort. The S matrix is an additional penalty on the state error at the last step of the prediction horizon.

Along with the set-up of the cost function and selection of the solver used for NMPC, it will also display the simulink architecture used to simulate the dynamics. This section will also demonstrate the initial performance of the nonlinear MPC with the derived dynamics in the previous section.

1.4.3 Implement nonlinear MPC controller

This section will be the proper implementation of the Nonlinear MPC via MATLAB/Simulink. This will include the nonlinear dynamics of atmospheric drag and J2 gravitational perturbation effects. This section will introduce force and moment limitations based on cubesat orientation. Also at the end of the section the effect of nonlinearities being added to the dynamics will be checked on the performance of the controller.

1.4.4 Refinement of the nonlinear MPC

This section will explore refinement needed for the NMPC in terms of constraints and allocating realistic forces and moments that the thrusters and reaction wheel can physically use. The section will also refine any improvements the simulation needs for accuracy. Mission scope will be defined and mission range will be set.

1.4.5 NMPC results and discussion

This section will explore gain tuning and robustness and provide results with conclusions from the designed and developed nonlinear MPC

2. Dynamic modeling

2.1 Vehicle model

The dynamics model of this project will aim to mirror that of a 3U cubesat from NASA's CPOD (Cubesat Proximity Operations Demonstration). The 3U cubesat makes use of a thruster module with 8 valve ports for translation and attitude control. It also includes the use of a 3-axis reaction wheel configuration. Since only a few modeling parameters from that mission are available through reports, all other parameters will be estimated with the use of CAD.

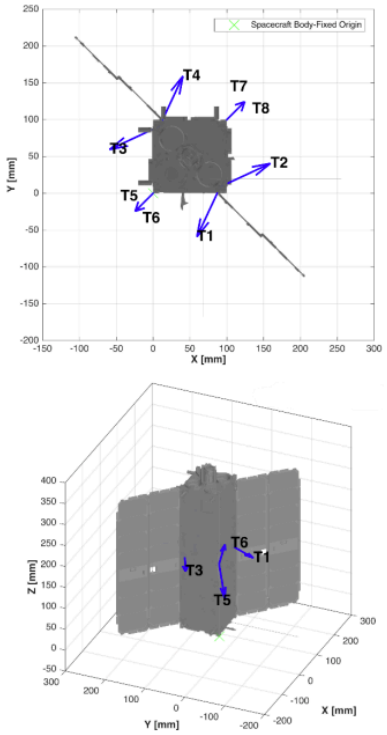


Figure 2.1 - Thrust directions of the 3U cubesat in the CPOD missions [18]

As shown in figure 2.1, the thruster module can provide thrust in 8 different directions and is mounted in the center of the cubesat. The thrust module used on the 3U cubesat is a Nammo reaction control propulsion module. The thrusters are mounted on the corners of the module and have a 15 degree offset from the cornerline and can thrust in both directions at each corner [19]. This thrust module uses cold gas thrusters using R236fa propellant [19].

Table 2.1 - Nammo reaction control propulsion module parameters [19]

Parameter	Value
Estimated Weight	1.33 kg
Specific Impulse [ISP]	40 seconds
Total Impulse	174 N-sec
Nominal Max Thrust	10 mN

One deviation from the 3U cubesat used in the CPOD mission in this project is the use of a 4 set pyramidal configuration reaction wheel. On the CPOD mission a 3 axis configuration was used, which consists of 3 reaction wheels placed in line with the cubesat's body frame axis. In this project, the cubesat will incorporate a pyramidal configuration of reaction wheels for more precision in attitude control. A pyramidal configuration set is the preferred option for the project as it provides a wider momentum envelope and higher agility needed for docking maneuvers. It also provides robustness as a pyramidal configuration can handle one failure to a reaction wheel. The project cubesat will use off the shelf reaction wheels from Nanoavionics [20]

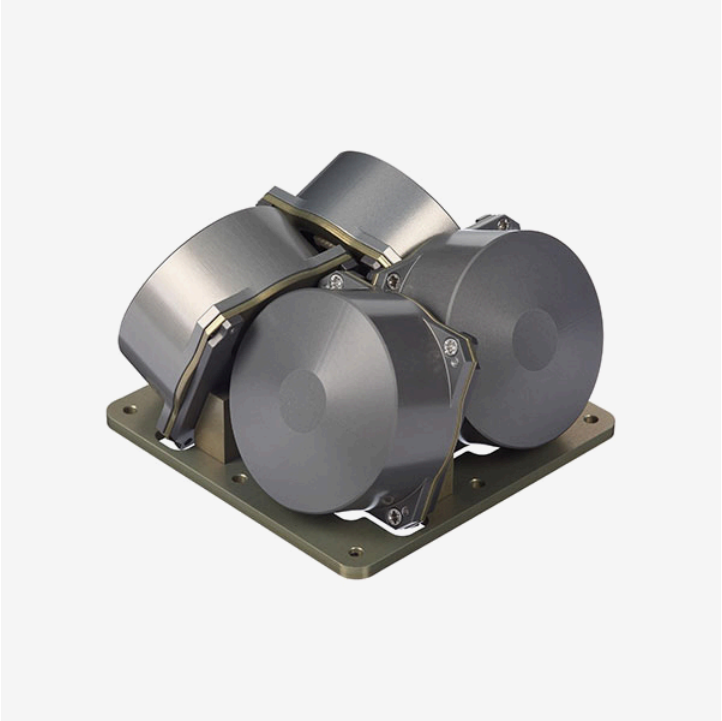


Figure 2.2 - Nanoavionics set of 4 reaction wheels [20]

Table 2.2 - Nanoavionics 4 reaction [20]

Parameter	Value
Maximum Torque (individual wheel)	3.2 mNm
Maximum Torque X-Axis (4 set configuration)	5.9 mNm
Maximum Torque Y-Axis (4 set configuration)	5.9 mNm
Maximum Torque Z-Axis (4 set configuration)	5 mNm
Estimated Weight	0.665 kg

A lot of specifics of the 3U cubesat are not available to the public so estimation is required to determine some of the needed model parameters for the dynamic model. In order to estimate the parameters, the use of CAD was essential. Using the weight for components that were listed online with CAD allowed for the estimation of the center of mass and moment of inertia. Below is the CAD model of the 3U cubesat for this project.

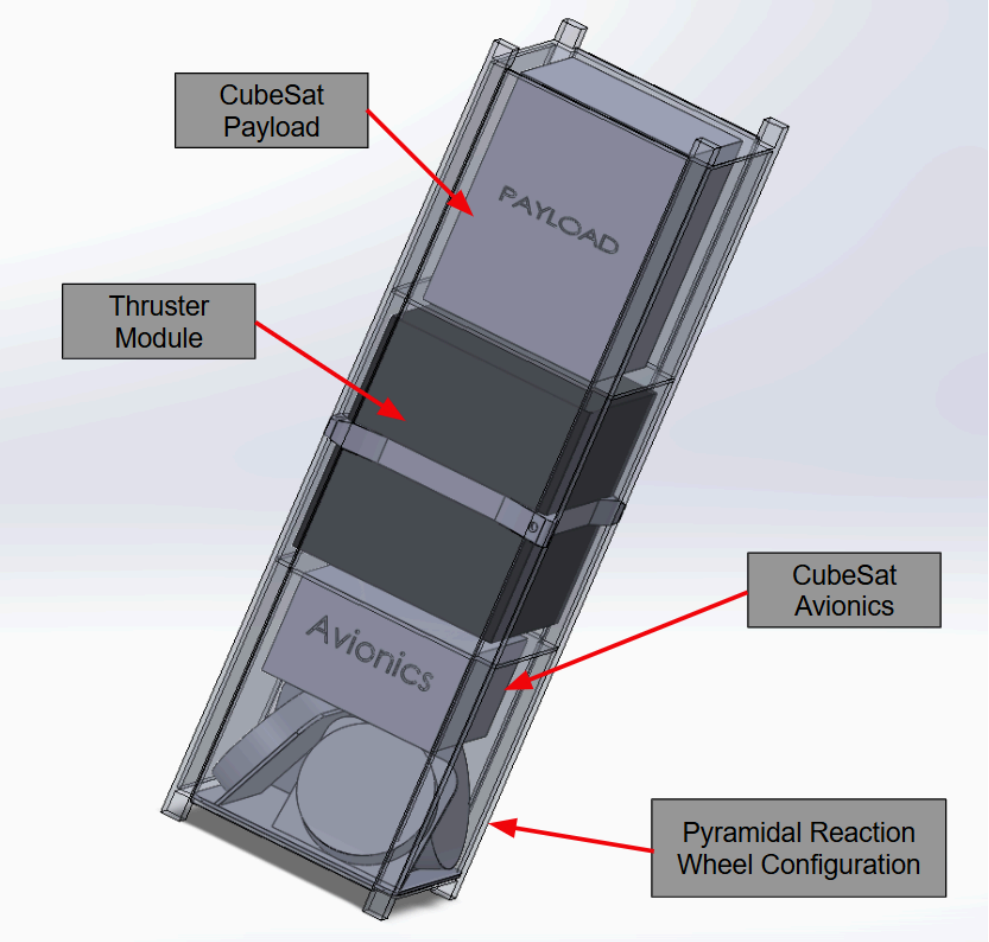


Figure 2.3 - CAD model of 3U cubesat for plant model

The weights of the payload and avionics were not available online so they were estimated by using the total nominal weight of a 3U cubesat reduced by the known weights of other components.

Table 2.3 - Cubesat model parameters

Parameter	Value
Weight	
Pyramidal Reaction Wheel [20]	0.655 kg
Thruster Module [19]	1.33 kg
Cubesat Frame [21]	0.290 kg
4 Cubesat Walls [estimated based on 6061 T6 material]	0.452 kg
Estimated Payload Weight	1.473 kg
Estimated Avionics Weight	0.8 kg
Estimated Total Weight [22]	5 kg
Model Parameters	
Moment of inertia [Ixx, Iyy, Izz]	[48467.4916, 48258.3323, 7014.0714] $kg{\cdot}mm^2$
Products of Inertia [Ixy, Iyz, Ixz]	[-7.0807,0.0296, -0.3724] $kg{\cdot}mm^2$
Drag coefficient [23]	2.2

2.2 Body Frame Orientation

The orientation of the cubesat will place the origin at the geometric center of volume. The +Z axis will be placed in the direction of the payload. Below is an image of the coordinate system with the center of mass [marked in red] also shown.

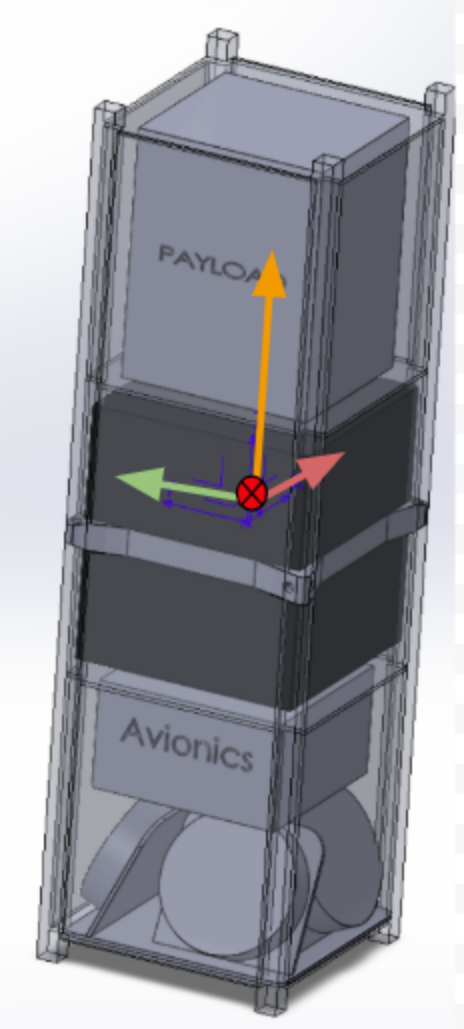


Figure 2.4: Body frame with center of mass location

The center of mass was calculated to be directly above the geometric center of volume. The center of mass was calculated to be at the XYZ position of [0, 0, 1.4054] millimeters. The orange arrow in figure 2.4 demonstrates the direction of the body z-axis, the green arrow demonstrates the direction of the body y-axis, and the red arrow demonstrates the direction of the body x-axis.

2.3 Control Inputs

As shown in figure 2.1, the thruster module for the cubesat was 8 thrust vectoring directions. With the coordinate system of the cubesat defined, the 8 thrust inputs can be broken down into 3 equations of forces in the body frame of the satellite.

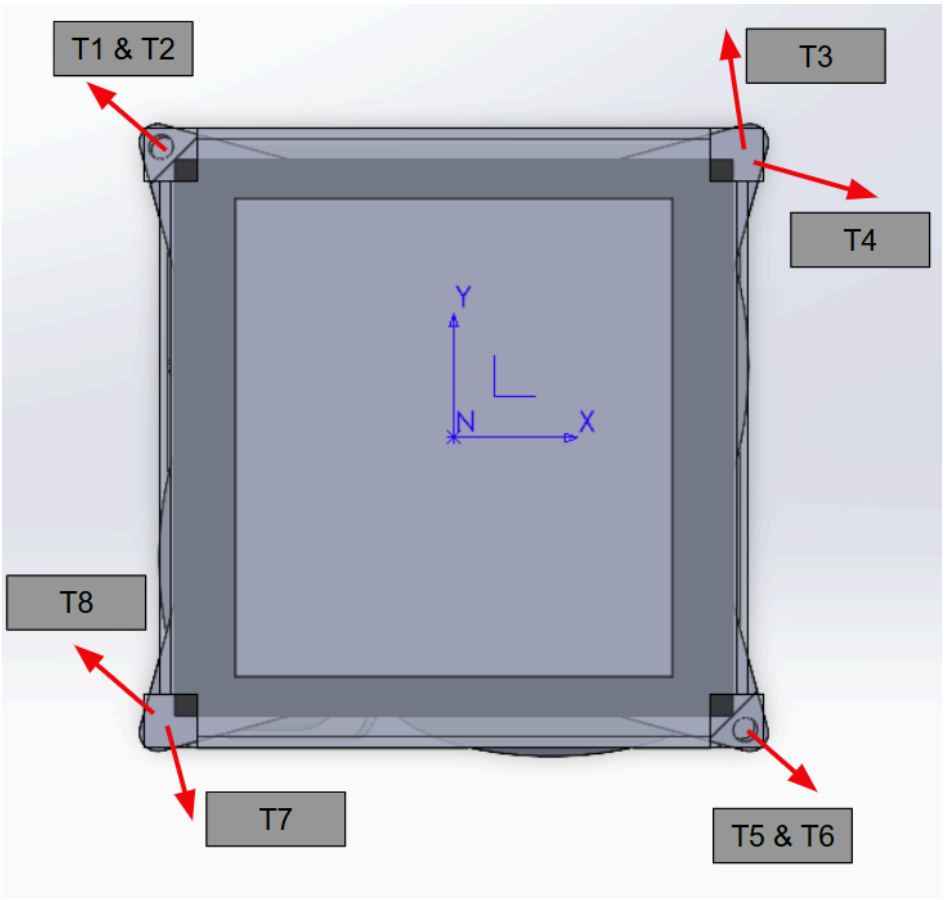


Figure 2.5 - Top view of thruster configuration

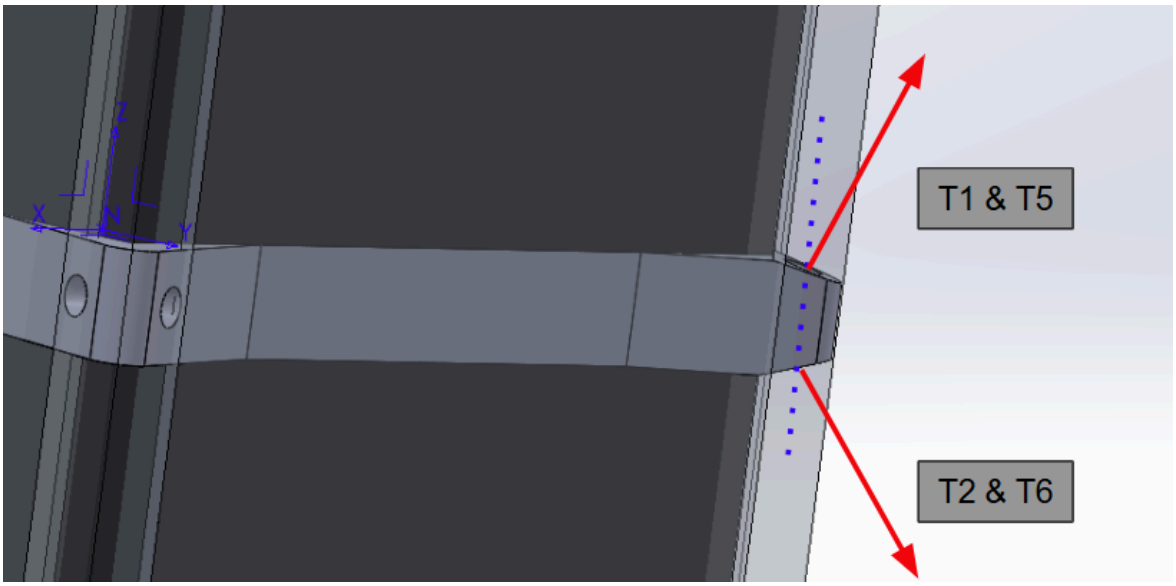


Figure 2.6 - Thrusters side view

Thrusters 1,2,5, and 6 are offset from the body z-axis by 15 degrees. Thrusters 3 and 7 are offset from the body y-axis by 30 degrees and Thrusters 4 and 8 are offset from the body x-axis by 30 degrees. To make the equations cleaner to read, table 2.4 replaces trigonometric operations with symbols to improve the readability of the equations. For moment equations, all thrusters are placed at the corners and have the same moment arm, which is calculated from CAD as 50 mm, for simplicity in the equations that moment arm will be shown as $\mathbf{r}_{moment}$.

Table 2.4 - Replace trigonometric operations with symbols

Symbol	Trigonometric Operation
β	$sin(15°)cos(45°)$
α	$cos(30°)$
ϕ	$sin(30°)$
θ	$cos(45°)$
γ	cos(15°)
d	1.4054 mm
$\mathbf{r}_{moment,x,y}$	50 mm

Below are the derived equations of the thruster input in the body frame of the cubesat.

$$F_{x,B} = ((T_6 + T_5) - (T_1 + T_2))\beta - (T_8 - T_4)\alpha - (T_3 - T_7)\phi \tag{2.1}$$

$$F_{y,B} = ((T_1 + T_2) - (T_5 + T_6))\beta - (T_4 - T_8)\alpha - (T_7 - T_3)\phi \tag{2.2}$$

$$F_{z,B} = (T_1 - T_2 - T_6 + T_5)\gamma \tag{2.3}$$

$$M_{x,B} = (T_6\gamma - T_2\gamma)\mathbf{r}_{moment} - (T_5\gamma - T_1\gamma)\mathbf{r}_{moment} + (F_{y,B})d \tag{2.4}$$

$$M_{y,B} = (T_2\gamma - T_6\gamma)\mathbf{r}_{moment} - (T_1\gamma - T_5\gamma)\mathbf{r}_{moment} + (F_{x,B})d \tag{2.5}$$

$$M_{z,B} = (F_{xThrust})r_{moment,x} + (F_{yThrust})r_{moment,y} \tag{2.6}$$

The momentum wheel is placed at the bottom of the cubesat and is set up in a pyramidal configuration. Below is a representation of how the reaction is set up from the cubesat body frame.

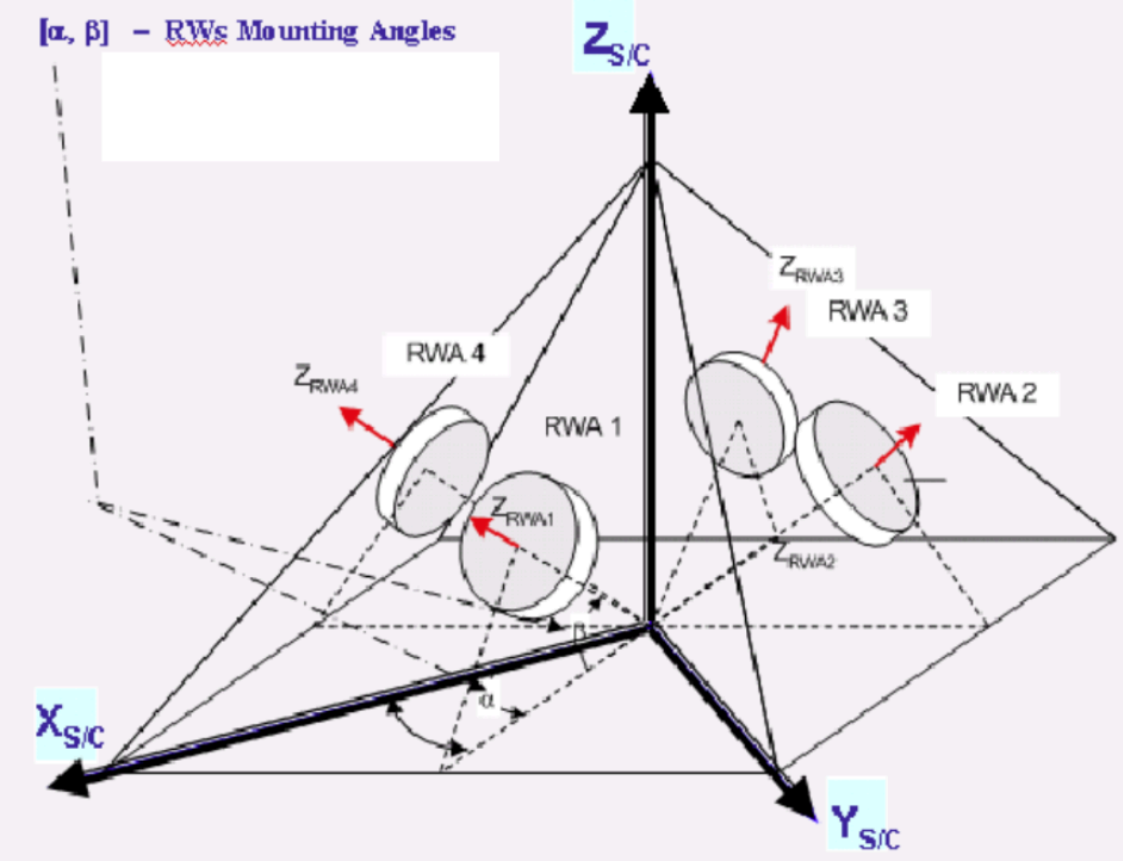


Figure 2.7 - Reaction wheel diagram [24]

The reaction wheels are set up in a pyramidal configuration and oriented as shown in figure 2.7. Each reaction wheel is set 90 degrees from each other and is rotated to be 45 degrees from the XY primary body frame axis. The reaction wheels are tilted in a way to pivot upwards from the XY plane. The pivot angle is called the cant angle and will be set to 35.26 degrees as it is the most optimal cant angle for reaction wheels in a pyramidal configuration [25]. For simplicity the cant angle will be represented in equations as the angle ζ. The reaction wheel creates 4 torque vectors. Each vector must be translated into a torque in the body frame of the cubesat. The matrix $\mathbf{A}_{rw}$ will demonstrate the transformation matrix needed to transform the four torques into reaction torques in the body frame of the cubesat.

$$T_{Rw} = [T_1, T_2, T_3, T_4] \tag{2.7}$$

$$\mathbf{A}_{rw} = \begin{bmatrix} \sin\zeta\cos 45^\circ & \sin\zeta\cos 135^\circ & \sin\zeta\cos 225^\circ & \sin\zeta\cos 315^\circ \\ \sin\zeta\sin 45^\circ & \sin\zeta\sin 135^\circ & \sin\zeta\sin 225^\circ & \sin\zeta\sin 315^\circ \\ \cos\zeta & \cos\zeta & \cos\zeta & \cos\zeta \end{bmatrix} \tag{2.8}$$

When multiplying equation 2.7 with the matrix transformation $\mathbf{A}_{rw}$, it results in a 3x1 array with the total torque contributions from the reaction wheel in the body frame of the cubesat.

2.4 Equation of Motion

To accurately model the rendezvous and docking maneuver, a high-fidelity nonlinear mathematical model is required. Unlike the linearized Clohessy-Wiltshire-Hill (CWH) equations, which assume a circular orbit and small relative separation, the nonlinear model accounts for the full gravitational nonlinearity and coupled dynamics. This robustness is critical for the Nonlinear Model Predictive Control (NMPC) formulation to handle large deviations and ensure precise docking.

The following equations of motion will deal with two reference frames. The first reference frame deals with the Earth-Centered Inertial (ECI) frame. The ECI frame has an origin placed at the center of the Earth with the z-axis aligned with the north pole. Below is an image of how the ECI reference frame is set-up.

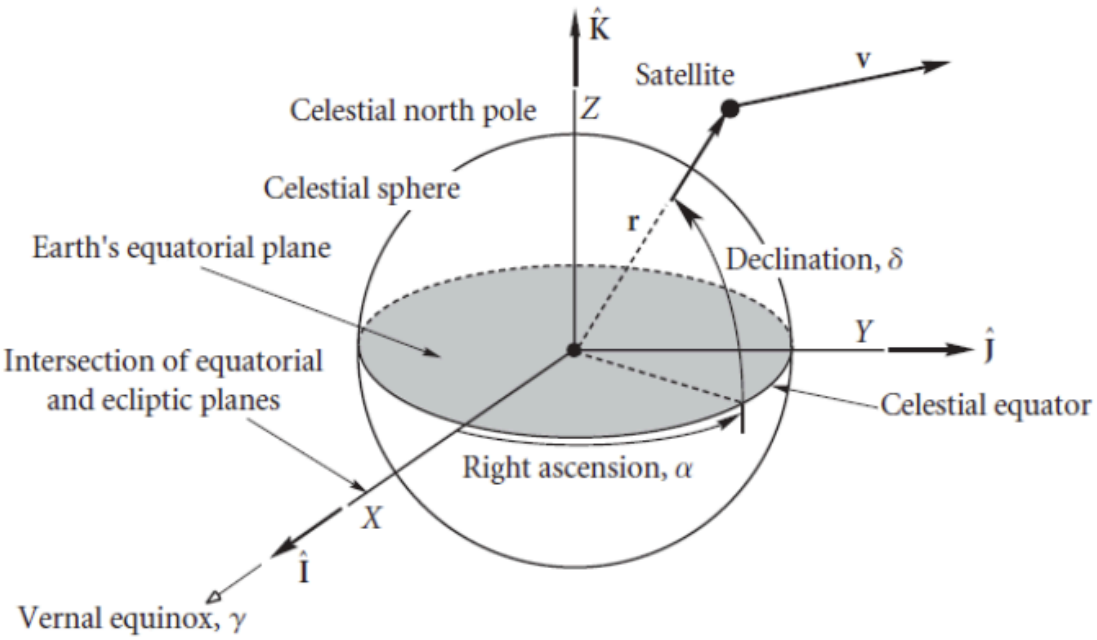


Figure 2.8 - Earth centered inertial frame [26]

The second frame is the local-vertical local-horizontal (LVLH) reference frame. The origin of this reference frame is located at the center of mass of the chaser vehicle. The x-axis of this frame is aligned with the line created between the earth's center of mass and the target vehicle's center of mass on the orbital plane of the target vehicle.

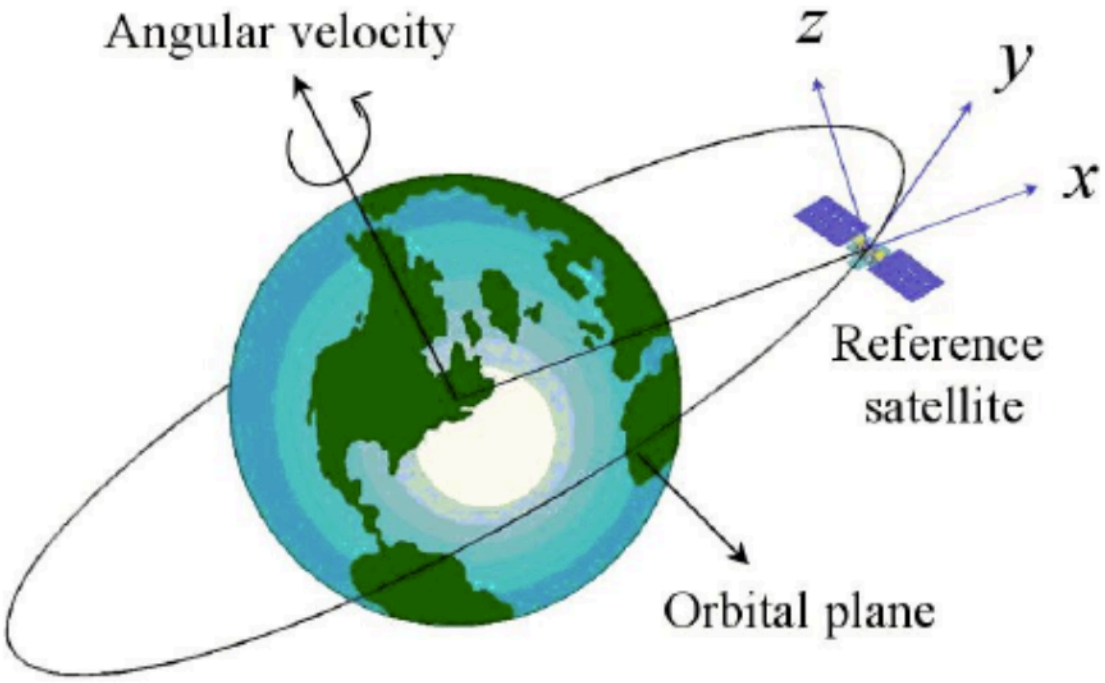


Figure 2.9 - Local-vertical local-horizontal frame [27]

The chaser cubesat equations of motion for translational dynamics will be presented in the ECI frame.

$$\ddot{\mathbf{R}} = -\frac{\mu}{\|\mathbf{R}\|^3}\mathbf{R} + \frac{\mathbf{F}_{\text{thrust}}}{m} + \mathbf{a}_{\text{pert}} \tag{2.9}$$

The vector **R** will represent the vector position of the chaser vehicle in the ECI frame. The μ is the gravitational parameter of Earth and $\boldsymbol{F}_{thrust}$ is the thrust vector produced by the thrusters described in section 2.3. The m term is the chaser vehicles total mass and the term $\mathbf{a}_{pert}$ is the disturbances from atmospheric drag and gravitational effects from the Earth's non-spherical shape.

The chaser vehicle is not the only equations of motion that need to be modeled in order to fully capture the dynamics of a rendezvous and docking mission. Moving forward the vector describing the chaser vehicle's position in the ECI frame will be $\mathbf{R}_c$. The equations of motion for the target cubesat are the same as the chaser vehicle with the exception of the $\boldsymbol{F}_{thrust}$ term since

no external forces will be applied to the target vehicle. The target vehicle's position vector will be represented with by the vector $\mathbf{R}_t$.

While in the LVLH frame the target vehicle's position can be described with a vector **p**. The vector **p** has x,y,z components in the LVLH frame. This will be the position of the target vehicle from the chaser vehicle. By combining this vector with the target vehicle's position, an equation for the chaser vehicle position can be derived.

$$\mathbf{R}_c = \mathbf{R}_t + \mathbf{p} \tag{2.10}$$

Taking the second derivative of $\mathbf{R}_c$ term in the inertial frame while taking into the account the angular velocity (ω) of the LVLH frame produces an equation that contains the acceleration of the chaser and target vehicle as well as the terms in the LVLH frame pertaining to the vector between the chaser and target vehicle.

$$\ddot{\mathbf{R}}_c = \ddot{\mathbf{R}}_t + \ddot{\rho} + 2\omega \times \dot{\rho} + \dot{\omega} \times \rho + \omega \times (\omega \times \rho) \tag{2.11}$$

Substituting equation 2.9 into $\mathbf{R}_c$ and $\mathbf{R}_t$ and equation 2.11 can be broken up into three equations for x,y,z acceleration of the position between the chaser and target vehicle.

$$\ddot{x} - 2n\dot{y} - n^2 x = -\frac{\mu(R_t + x)}{r_c^3} + \frac{\mu}{R_t^2} + \frac{F_x}{m_c} + a_{drag,x} \tag{2.12}$$

$$\ddot{y} + 2n\dot{x} - n^2 y = -\frac{\mu y}{r_c^3} + \frac{F_y}{m_c} + a_{drag,y} \tag{2.13}$$

$$\ddot{z} = -\frac{\mu z}{r_c^3} + \frac{F_z}{m_c} + a_{drag,z} \tag{2.14}$$

The equations above are the equations of motion between the two vehicles in the LVLH frame.

$$r_c = \sqrt{(R_t + x)^2 + y^2 + z^2} \tag{2.15}$$

$$n = \sqrt{\frac{\mu}{R_t^3}} \tag{2.16}$$

$\boldsymbol{F}_x$, $\boldsymbol{F}_y$, *and* $\boldsymbol{F}_z$ are the control forces acting on the chaser in the LVLH frame. The n term is the orbital rate of the target vehicle. The drag term in the equations is modeled below:

$$\mathrm{a}_{drag} = -\frac{1}{2}\rho_{atm}\frac{C_D\mathbf{A}}{m_c}|\mathbf{v}_{rel}|\mathbf{v}_{rel} \tag{2.17}$$

The term ρ_{atm} is the local atmospheric density which will be selected in the mission requirements for the density between the chaser and target vehicles orbit altitudes. The term C_D is the drag coefficient which is estimated in table 2.3. The term **A** in equation 2.17 is the cross sectional area perpendicular to the velocity vector, this term will be calculated in the simulink simulation. The term $\mathbf{v}_{rel}$ is the tangential velocity vector of the cubesat in the orbit plane.

$$\mathbf{I}\dot{\mathrm{w}}_{body} + \mathrm{w}_{body} \times (\mathbf{I}\mathrm{w}_{body}) = \tau_{ctrl} + \tau_{ext} \tag{2.18}$$

Equation 2.18 is the rotational dynamics of the chaser vehicle. The term **I** is the inertia tensor matrix as defined in table 2.3. The term w_{body} is the angular velocity of the chaser body frame relative to the inertial frame (ECI). the term τ_{ctrl} is the torque produced from the thrusters and reaction wheel as defined in section 2.3. The term τ_{ext} is the torque from external influences such as atmospheric drag and J2 gravitational effects. In simulation the use of quaternions will be used to avoid singularities with Euler angles. Once the angular velocity is given, the change in cubesat orientation will be given through a quaternion equation.

$$\dot{\mathbf{q}} = \frac{1}{2}\Omega(\omega_{body})\mathbf{q} \tag{2.19}$$

The term $\Omega(\omega_{body})$ is a skew-symmetric matrix formed by the angular velocity vector. When implementing the nonlinear model predictive controller the use of equations 2.9 to 2.19 help form the state vector **x**.

$$\mathbf{x} = [x, y, z, \dot{\mathrm{x}}, \dot{\mathrm{y}}, \dot{\mathrm{z}}, q1, q2, q3, q4, \omega_x, \omega_y, \omega_z]^T \tag{2.20}$$

$$\mathbf{u} = [\boldsymbol{F}_x, \boldsymbol{F}_y, \boldsymbol{F}_z, \tau_x, \tau_y, \tau_z]^T \tag{2.21}$$

Using the equation set-up in 2.20 and 2.21 allow for the system between the chaser and target vehicle to be set-up as general nonlinear equation $\dot{\mathrm{x}} = f(\mathbf{x}, \mathbf{u})$.

2.5 Simulink set-up

Before creating the simulation involving the nonlinear dynamics and the model predictive controller, some initialization requirements must be defined. One of the largest 3U satellites, the spire global lemur-2, is usually in orbit at altitudes of 400 km to 600 km. The ranges of the rendezvous will aim to be within these orbit altitudes. Below is a table with the summary of mission parameters to be held constant in the first set-up of the simulation.

Table 2.5: Mission constants for simulation

Mission parameter	Value
Chaser vehicle weight	5 kg
Earth's Geocentric Gravitational constant	3.986E14 m^3/s^2
Target vehicle orbit altitude	4000 m
Chaser vehicle orbit altitude	4100 m
Max thrust	10 mN
Total impulse	174 Ns
Max torque	3.2 mNm
Simulation Time	2000 seconds

These mission parameters will be initialized to be used throughout the simulation in all functions created. The first simulations will focus on creating the simulink environment and creating the nonlinear model predictive controller. As the development on the simulink model goes on, more fidelity will be added to the model, including force and moment constraints and line of sight constraints.

3. Nonlinear MPC problem

3.1 Formulation of the nonlinear MPC problem

An overview of how the nonlinear model predictive controller will be set-up for the project. This controller will implement a control strategy using a mathematical model of a system to predict its future behavior and optimize a sequence of control inputs to achieve a desired objective. Unlike other traditional controllers like PID or LQR which react to errors, the model predictive control is proactive and looks over a finite window which is called the prediction horizon.

The set-up to create this nonlinear model predictive controller involves a measurement, a prediction, optimization, application, repeat. The first step is measurement of a state vector, this involves a dynamic mathematical model, which will be presented in section 3.2. The next is the prediction, this step involves defining how many timestamps ahead does the controller look into before optimization. The method for defining this prediction will be used in the reference generation block described in section 3.2.

The formulation of the optimization step sets up the cost function to optimize control inputs while minimizing distance to the target. The general cost function structure is shown below.

$$J = \sum_{i=1}^{N-1} (x_i^T Q x_i + u_i^T R u_i) + x_N^T S x_N \tag{3.1}$$

The Q matrix in the cost function acts as weights for the state vector to assign priority to any of the elements in the **x** state vector shown in 2.20. The Q matrix will be a 13x1 matrix, each element corresponding to each state in the state vector. Increasing one of these elements in the Q matrix will place more importance on the error of that state. For example, if after running a simulation the x-position seems to overshoot the desired location, then an increase in the first element of the Q matrix will be required. The R matrix assigns weights to the control inputs dictating which force or moment input has more control authority. The R matrix is a 6x1 matrix, the first three elements correspond to the force inputs and the last 3 elements correspond to the moment inputs. Decreasing an element in the R matrix will allow for more usage of the corresponding force and moment input. The S matrix is used as a penalty to the cost function when deviations occur between the targeted state vector which will be calculated at the final prediction step.

3.2 Simulink Set-up

The simulink model was created using primarily four main MATLAB function blocks. The four blocks used were the equations of motion between the chaser and target cubesat, a calculating jacobians block, a generation of a reference trajectory block and finally the model predictive controller block.

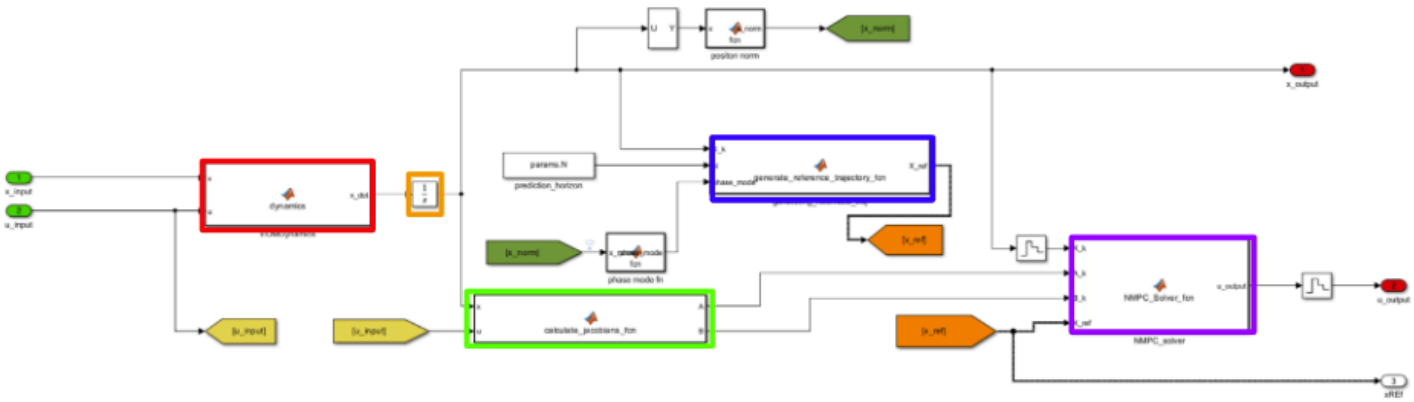


Figure 3.1: Simulation model dynamics and nonlinear model predictive controller

The first main block (red outline) is the equations of motion dynamics block, this block is coded to calculate the derivative of the dynamics of the x state vector described in equation 2.16. The dynamics MATLAB function block uses the equations derived in equations 2.12-2.14. These equations are restructured to solve for the acceleration of the state. The equations solved are listed below:

$$\ddot{x} = 2n\dot{y} + n^2 x - \frac{\mu(R_t + x)}{r_c^3} + \frac{\mu}{R_t^2} + \frac{F_x}{m_c} + a_{drag,x} \tag{3.2}$$

$$\ddot{y} = -2n\dot{x} + n^2 y - \frac{\mu y}{r_c^3} + \frac{F_y}{m_c} + a_{drag,y} \tag{3.3}$$

$$\ddot{z} = -\frac{\mu z}{r_c^3} + \frac{F_z}{m_c} + a_{drag,z} \tag{3.4}$$

The derivatives of states (3:6) are calculated using these equations. Initial conditions help produce the states needed to solve for equations in 3.2, 3.3, and 3.4. Figure 3.1 shows that the derivative of the states is outputted and then fed into an integration block that produces the x state vector that will be used for the rest of the simulation. The integration block is outlined in orange in figure 3.1. From that integration block, that state vector is fed back into the dynamics block. The accelerations found in the block at the first timestep gets integrated and becomes the

velocity vector used in the dynamics block. The velocities that are then integrated after the dynamics model become the position elements in the state vector to be used in the next timestep.

The state vector after it has been integrated from the output of the dynamics block will be inputted into the jacobian calculator along with the forces and moments control input vector. The primary purpose of the jacobians function block is to linearize the nonlinear system dynamics around the current operating point (x,u) at every timestamp. The output of this block is the linearized space state matrices **A** and **B**. **A** is the system matrix while the **B** is the control input matrix. The **A** matrix represents the system's sensitivity of the state derivative to changes to the state vector(**x**), while the **B** matrix represents the sensitivity of the state derivative to changes in the input vector (**u**).

The way the Jacobians are calculated is through the finite difference method [29]. In this method a forward difference is calculated to create an approximation of change. The formula for the finite differences method is presented below.

$$\frac{\partial f}{\partial v_j} \approx \frac{f(v+\varepsilon e_j) - f(v)}{\varepsilon} \tag{3.5}$$

This formula is applied to both state and input vectors to derive the **A** and **B** matrices needed for the nonlinear model predictive controller. The **v** vector is the vector being perturbed, the ε is the small perturbation and the e_j vector is the unit vector of the j-th position of the vector. For example, at each state it calculates the perturbation effect while zeroing out all over the terms that it is not calculating for. The term v in equation 3.5 represents the state vector x produced by the dynamics model.

The variable ε is crucial choice when developing this function for the simulink set-up. If the variable is not small enough, errors can occur in the simulation that can produce values in the A matrix that will crash. For the first simulation a value of 1e-6 will be used for ε.

The algorithm is used in the model block outlined in green presented in figure 3.1. In that model block, a for loop is created where the code goes through each element in state or input vector and calculates a perturbed derivative. At each loop, a difference quotient is calculated using equation 3.5. The output of equation 3.5 is then stored in A or B matrix depending on if the for loop is using the state vector or the control input vector

The linearized output from calculating the jacobians allows for the use of a quadratic programmer to solve the nonlinear model predictive controller cost function. This linearization allows for quick debugging and validation of simulink set-up. This also allows for the creation of

a local linear approximation of the nonlinear system. A local linear approximation allows the use of a quadratic solver to optimize the inputs of the system, this reduces the computational effort instead of relying on a nonlinear solver like IPOPT (interior Point OPTimizier).

The next block that is required for the nonlinear mpc block to work is a reference trajectory generator. This function block is outlined in blue in figure 3.1. In this matlab function a reference trajectory is determined based on the prediction horizon and the integrated x state vector, and norm position of that state vector. The norm state vector is meant to determine the distance to the target vehicle and is used to switch mission phases such as switching between rendezvous and docking phase. When the cubesat is in the rendezvous phase it uses a reference x state to calculate the new target distance at each prediction horizon. When the cubesat is within 50 meters of the target, it switches to a docking mode. In that mode it uses the target position as the reference x state so that it aggressively pursues the target position since the target is near. This block is still being developed as it has needed to be debugged often. In this section the reference trajectory is a straight line to the target position. The trajectory is determined by the position of the chaser, a determined max velocity, and a target waypoint.

The next block is the nonlinear MPC solver block. In this block outlined in purple in figure 3.1, receives four inputs and outputs the nominal force and moment control input. The four inputs received are the current state vector, the reference state vector or desired state vector, the linearized **A** matrix and the linearized **B** matrix. Other inputs such as the prediction horizon and sampling time have been hard coded. In this section of the report the prediction horizon is set to 10 and the sampling time is set 0.5 seconds.

Since the **A** and **B** matrices are linearized, the nonlinear model predictive controller can be set-up as a quadratic programming problem. The function quadprog from matlab database was used to solve for the optimal control inputs.

$$\min_{v} 0.5\mathbf{v}^T\tilde{\mathbf{H}}\mathbf{v} + \mathbf{c}^T\mathbf{v} \tag{3.6}$$

Equation 3.6 is the standard quadratic programming problem form listed in previous references [30]. To set-up equation 3.1 to be used with matlab's quadprog function, it needs to be formed into something similar to equation 3.6. By using the state reference vector x_{ref} produced by the reference generating function, equation 3.1 can be expanding into a more typical MPC optimization problem [30].

$$J = \sum_{i=1}^{N} (\mathbf{x}_{k+i} - \mathbf{x}_{ref,i})^T \mathbf{Q} (\mathbf{x}_{k+i} - \mathbf{x}_{ref,i}) + \sum_{i=0}^{N-1} \mathbf{u}_{k+i}^T \mathbf{R} \mathbf{u}_{k+i} \tag{3.7}$$

Since the S penalty matrix is only applied to state vector at the last prediction state x_N, so it does not need to be a part of the optimization problem used with the quadprog function. Using the **A** and **B** matrices from the calculated jacobian function, the predicted states can be determined. This is crucial as the optimization problem for the MPC must include the predicted states to optimize force and moment inputs. Below is the formulation of prediction matrices based on the linearized **A** and **B** matrices at time step *k*.

$$\mathbf{S_x} = [A_k, A_k^2, A_k^3, \ldots, A_k^N]^T \tag{3.8}$$

$$\mathbf{S_u} = A_k^{i-j} B_k \tag{3.9}$$

Equations 3.8 and 3.9 are the prediction matrices to be used in the MPC optimization problem shown in equation 3.7. $\mathbf{S_x}$ is the free response of the system over the prediction horizon. $\mathbf{S_u}$ is the forced response over the prediction horizon. A similar form is used in other papers when creating the prediction matrices [30]. The prediction matrices will be used to set up the predicted states and optimized outputs that will replace the x_{k+i} and u_{k+i} in equation 3.7 [30]. The size of the predicted state is determined by the prediction horizon parameter for the MPC. The number is important because it determines how far into the future the MPC controller observes to find the optimal control input. Too small a prediction horizon would mean not capturing important dynamics, too large and the computational effort increases.

$$\mathbf{X} = [\mathbf{x}_{k+1}^T, \ldots, \mathbf{x}_{k+N}^T]^T \tag{3.10}$$

$$\mathbf{U} = [\mathbf{u}_k^T, \ldots, \mathbf{u}_{k+N-1}^T]^T \tag{3.11}$$

Equations 3.10 is the predicted state and equation 3.11 is the sequence of control inputs to be optimized. In order to solve for **X** the predicted state, the prediction matrices will be used to solve for each step in the prediction horizon.

$$\mathbf{X} = \mathbf{S_x} x_k + \mathbf{S_u} \mathbf{U} \tag{3.12}$$

$$J = (\mathbf{S_x} \mathbf{x}_k + \mathbf{S_u} \mathbf{U} - \mathbf{X}_{ref})^T \mathbf{Q} (\mathbf{S_x} \mathbf{x}_k + \mathbf{S_u} \mathbf{U} - \mathbf{X}_{ref}) + \mathbf{U}^T \mathbf{R} \mathbf{U} \tag{3.13}$$

Equation 3.12 is a prediction state calculated based on free response and the forced response from a nominal control input. Equation 3.12 is substituted into equation 3.7 for variable x_{k+i} and Equation 3.11 is substituted into equation 3.7 to replace u_{k+i}. The result is equation 3.13, which will be used to set-up the quadratic programming problem. For the set-up used in this project vector v in equation 3.6 is the $\mathbf{S_u}$ is the state we want to minimize to reduce the response needed to achieve a state.

$$\mathbf{H} = 2(\mathbf{S_u}^T \mathbf{Q}_{big} \mathbf{S_u} + \mathbf{R}_{big}) \tag{3.14}$$

$$\mathbf{c}^T = 2(\mathbf{S_x} \mathbf{x}_k - \mathbf{X}_{ref})^T \mathbf{Q}_{big} \mathbf{S_u} \tag{3.15}$$

Equation 3.14 and Equation 3.15 can be retrieved from equation 3.13, they are the hessian matrix and linear term needed to be used with the quadprog function from matlab[30]. For the quadprog function to work it needs a hessian matrix, the linear term, constraint matrices, and upper and lower bounds for the optimal output. In this section the constraint matrices are left empty while adding the upper and lower bounds as the maximum force and moments shown in table 2.1 and 2.2.

In order to get the full simulation to run, the calculated state vector and control input vector must be fed back into the dynamics block. In order to accomplish this, all simulink blocks in figure 3.2 were placed into a subsystem as the output of that simulink block was fed back into the inputs of that same simulink block. This was only done by also adding unit-delay blocks to the inputs of the final subsystem block. Initial state and control vectors were added to the data dictionary and loaded before simulation runs.

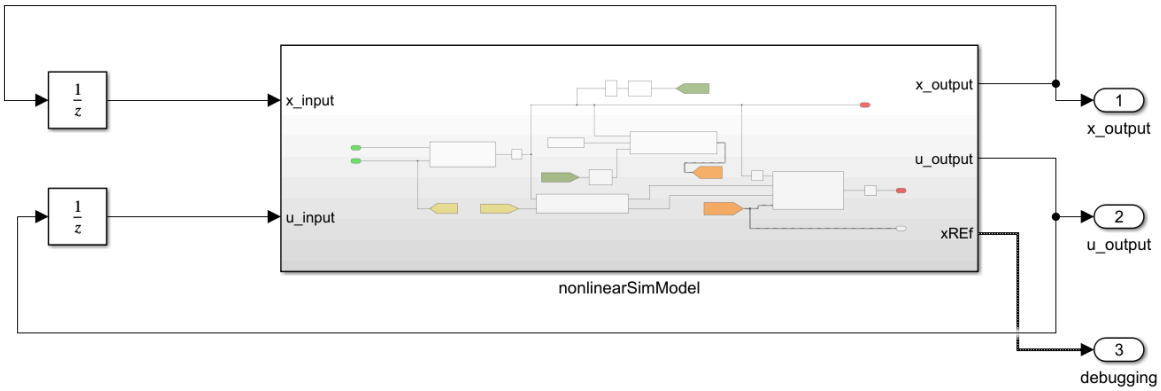


Figure 3.2: Full nonlinear simulation

3.3 Simulink first results

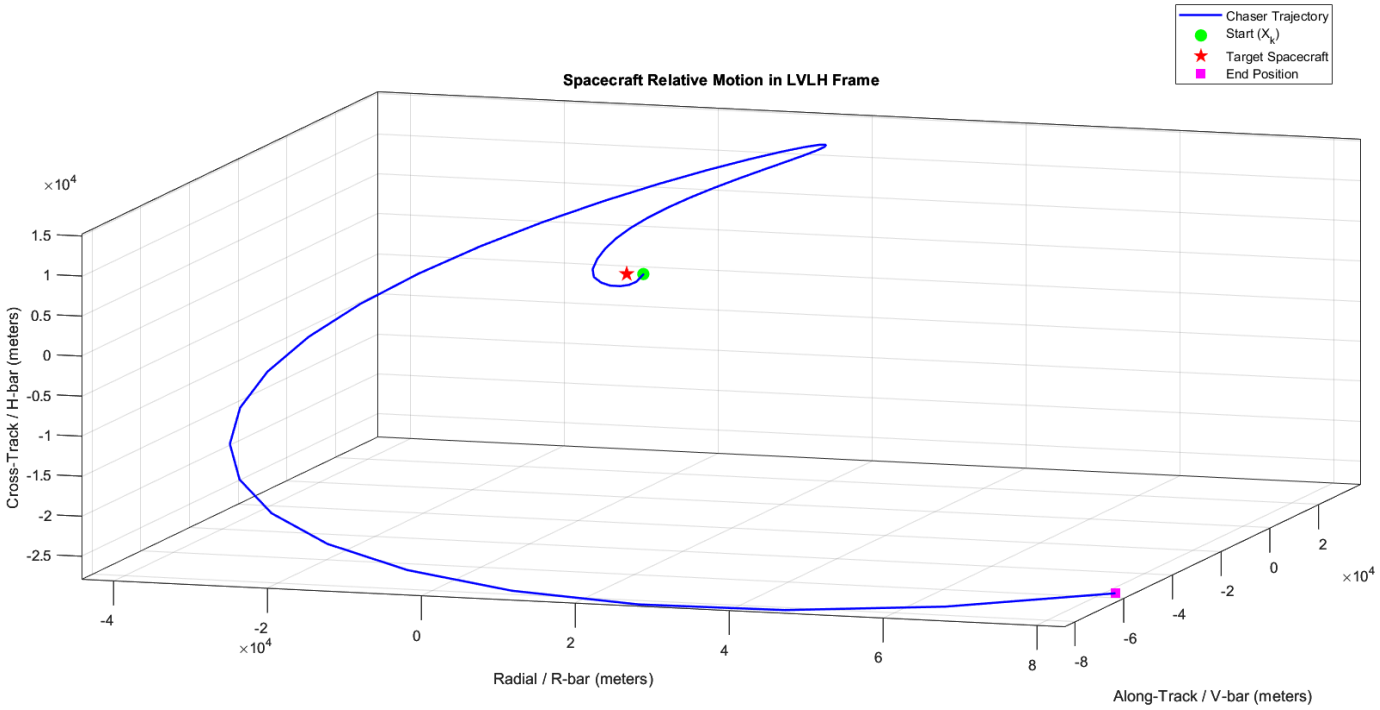


Figure 3.3 First initial results of the nonlinear model predictive controller in the LVLH frame

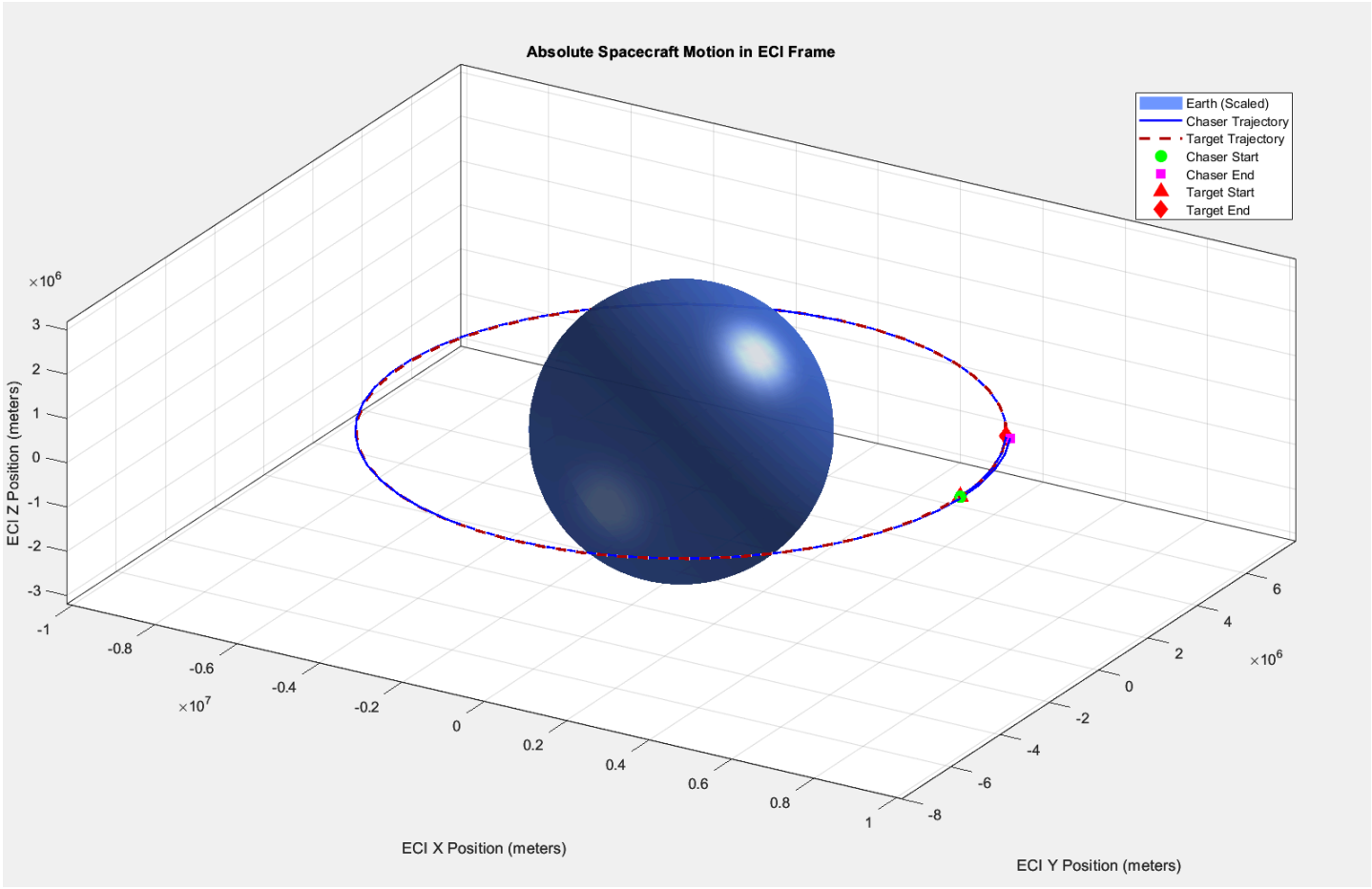


Figure 3.4: First initial results of the nonlinear model predictive controller in the ECI frame

Initial results showed that the controller did not achieve its target objective and it actually drifted away from the target vehicle. Many different attempts and changes were done to find the issue. The dynamics were checked for any inversion in which the accelerations were not aligned with the LVLH frame, the control inputs were also checked to see if the forces and moments coming out of the nonlinear model predictive controller were aligned in the LVLH frame. Debugging the issue led to altering the code so that the x reference state vector was always the target position. For the debugging the target position was set to 50 meters away from the target cubesat.

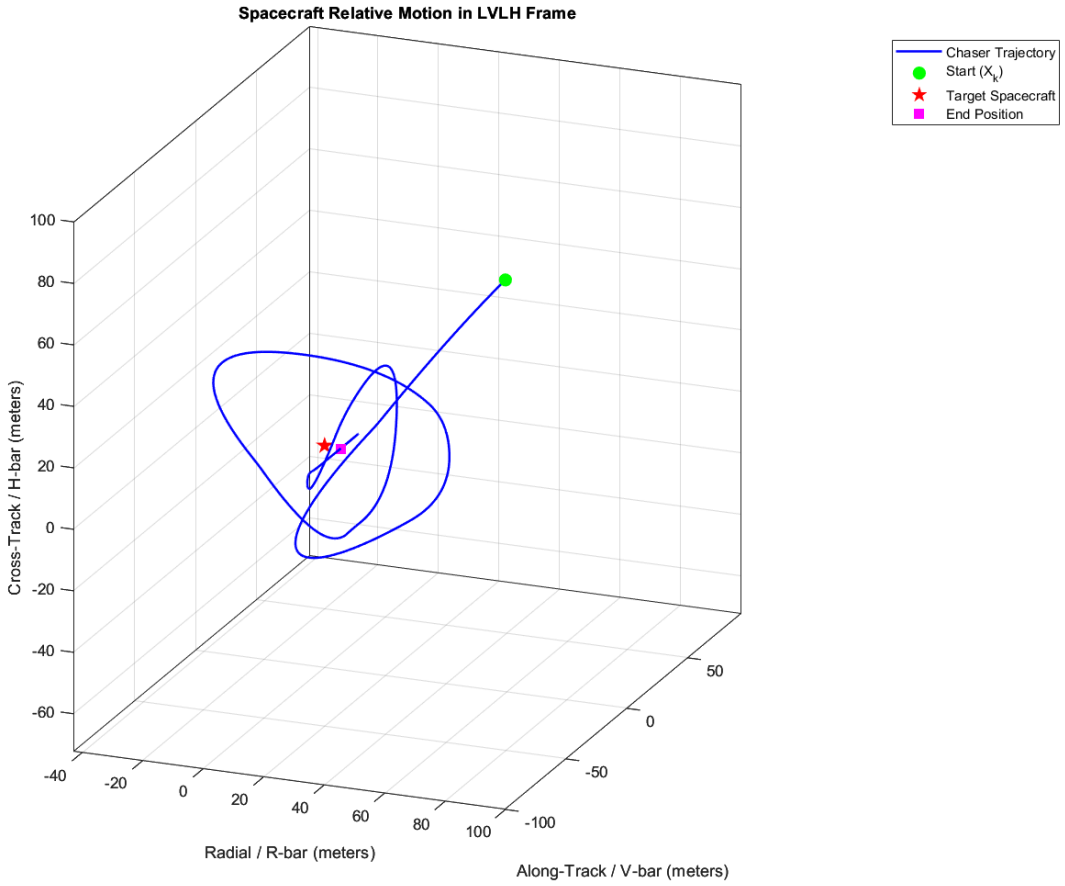


Figure 3.5: Debug results of the nonlinear mpc view 1 in the LVLH frame

The results showed promise as the ending position of the chaser vehicle was much closer to the target vehicle. The initial look shows that the chaser vehicle overshoots the target vehicle and returns back to the target position. Seems when the x reference state vector is constantly asking for high error so the nonlinear model predictive controller is commanding high forces and moments to restore the ship back to the target position.

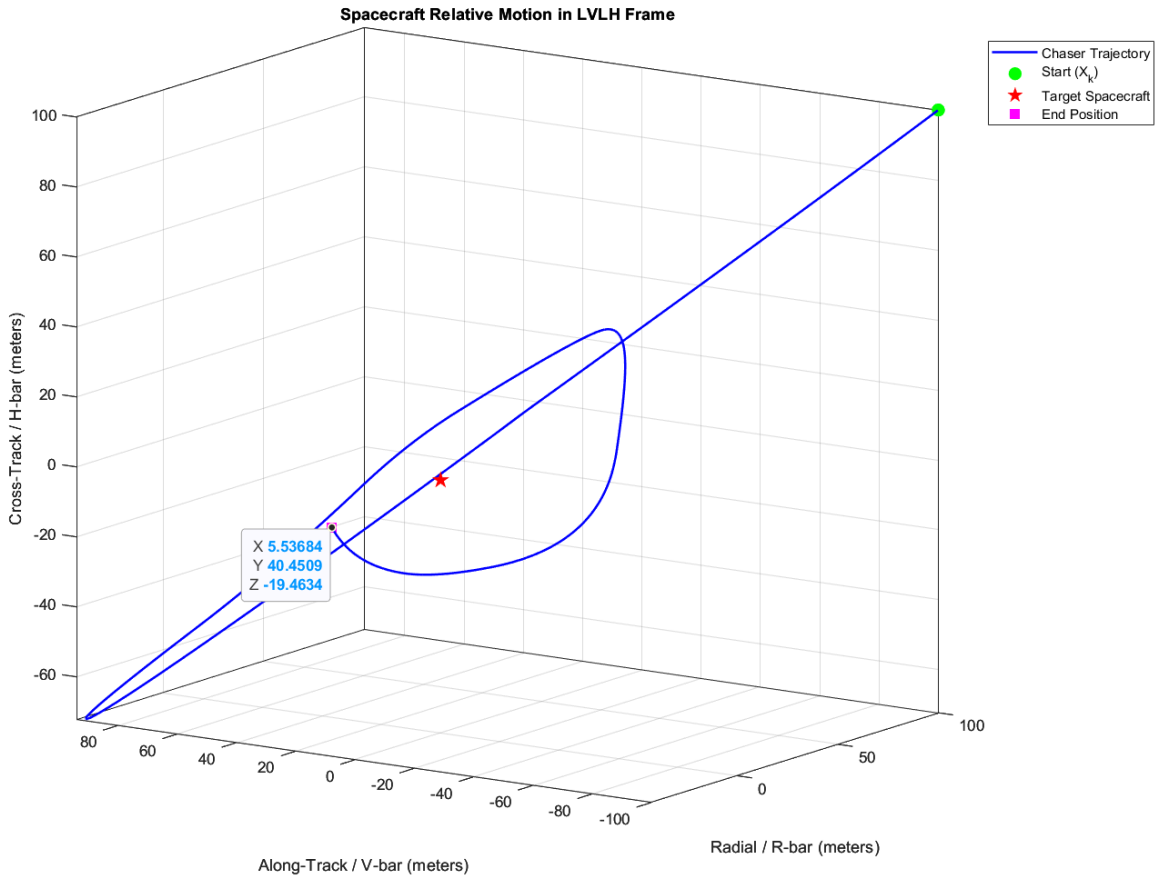


Figure 3.6: Debug results of the nonlinear mpc view 2 in the LVLH frame

Upon a closer look it does look as the chaser vehicle is reaching the target location. In the rendezvous target it was set to 50 meters, so the chaser is close to that distance from the target. This shows that the nonlinear model predictive controller is working well.

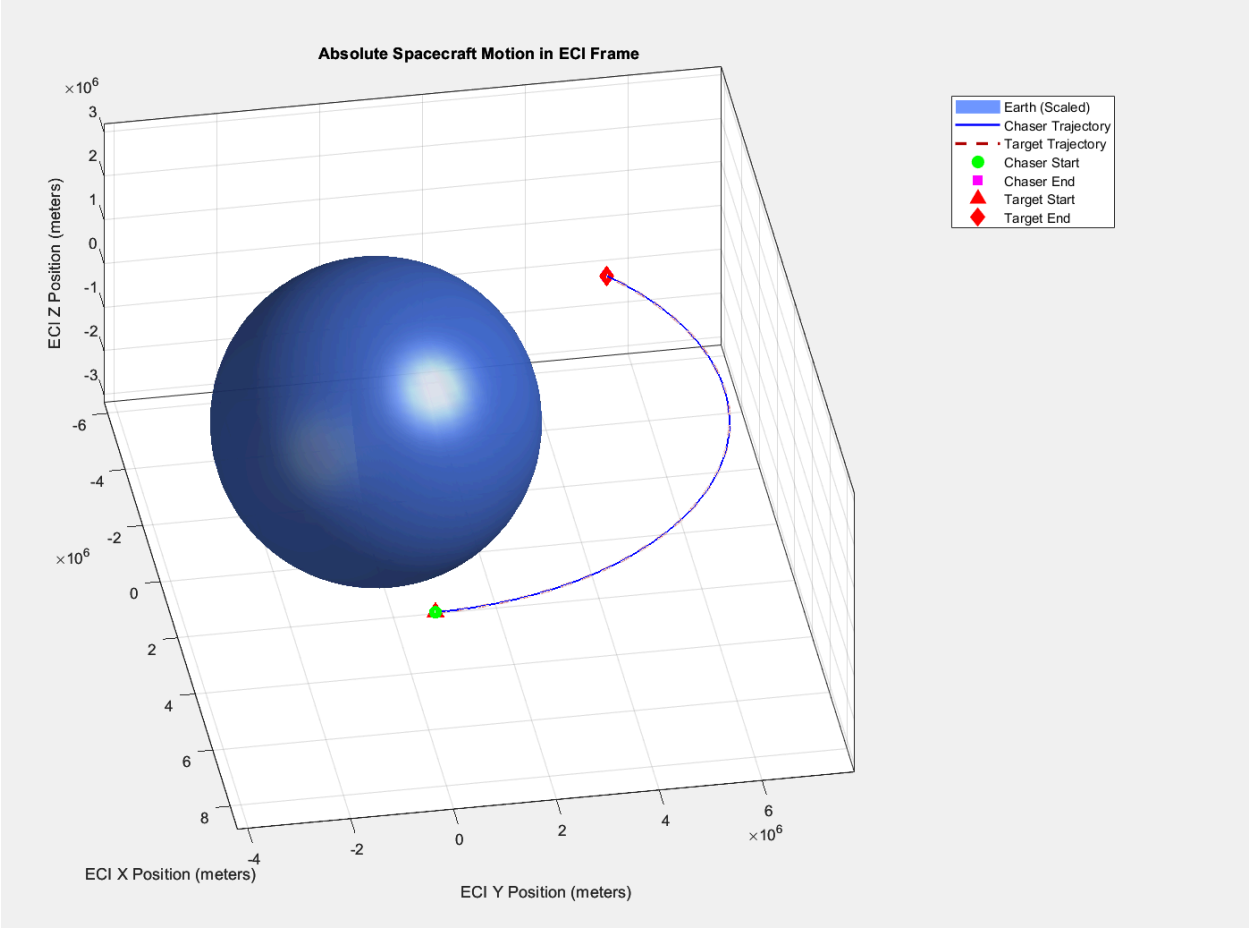


Figure 3.7: Debug results of the nonlinear mpc view in the ECI frame

The ECI frame plotting doesn't show much and will not appear in the rest of the report as it does not give useful information. Perhaps in future iterations that are positioned farther away it will be used.

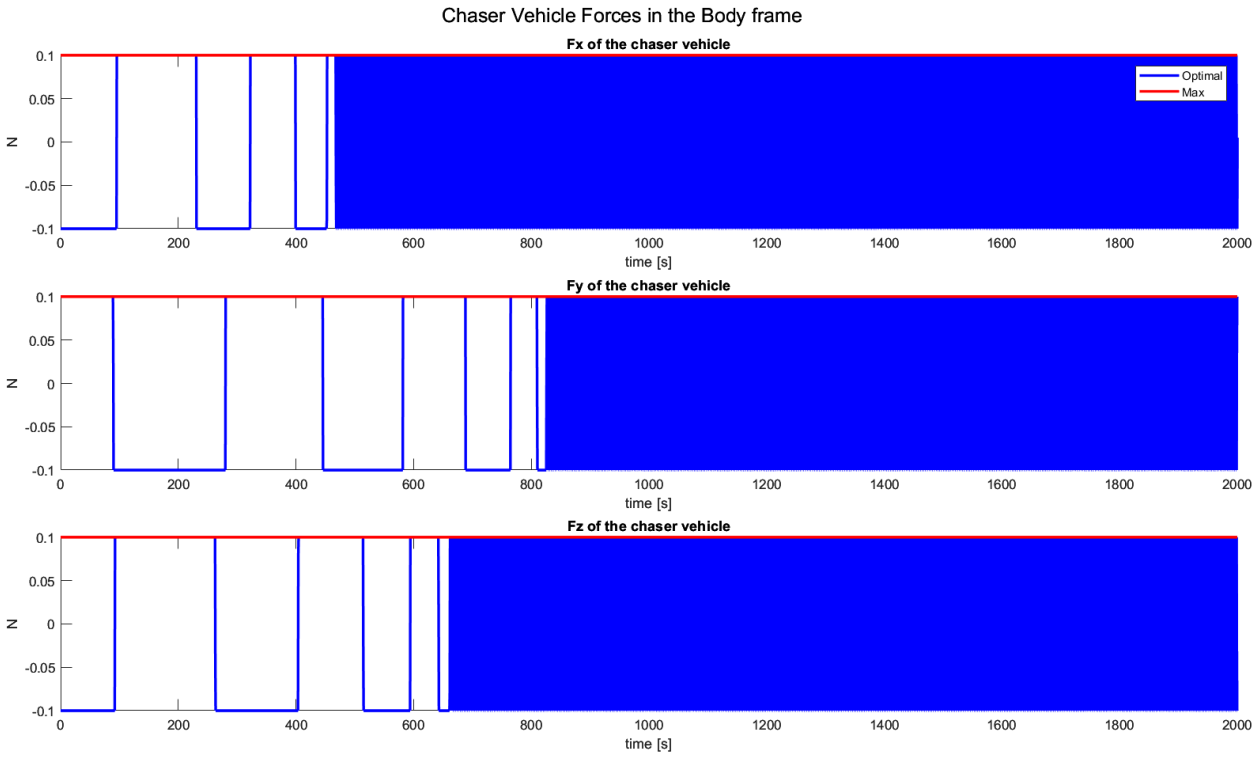


Figure 3.8: Debug results of the nonlinear mpc: forces in the LVLH frame

Figure 3.8 shows a lot of binary forces and moments going from plus or minus max force. After nearly 700 seconds it seems that the control inputs on force become rapidly binary that it looks like a solid block of inputs. This can be interpreted as the cubesat has arrived at its target location and is holding its position.

Analysis of the simulation led to finding an error in how the x state vector was being fed into the reference trajectory generator block of code. An extra zero-order hold block was placed between the state vector and the input to the generated reference trajectory block, this caused an error in the reference state being fed into the nonlinear mpc solver. The simulation was run again with the corrected issue and to further test the robustness the initial conditions were changed.

Table 3.1: Initial conditions and parameters for current simulation

Parameter name	Value
Initial position of the chaser cubesat	[-100;-100;100] meters
Prediction window (N)	10
Max thrust in any axis	0.1 N
Max torque in any axis	0.006 Nm
Initial euler angles	[45;0;0] degrees

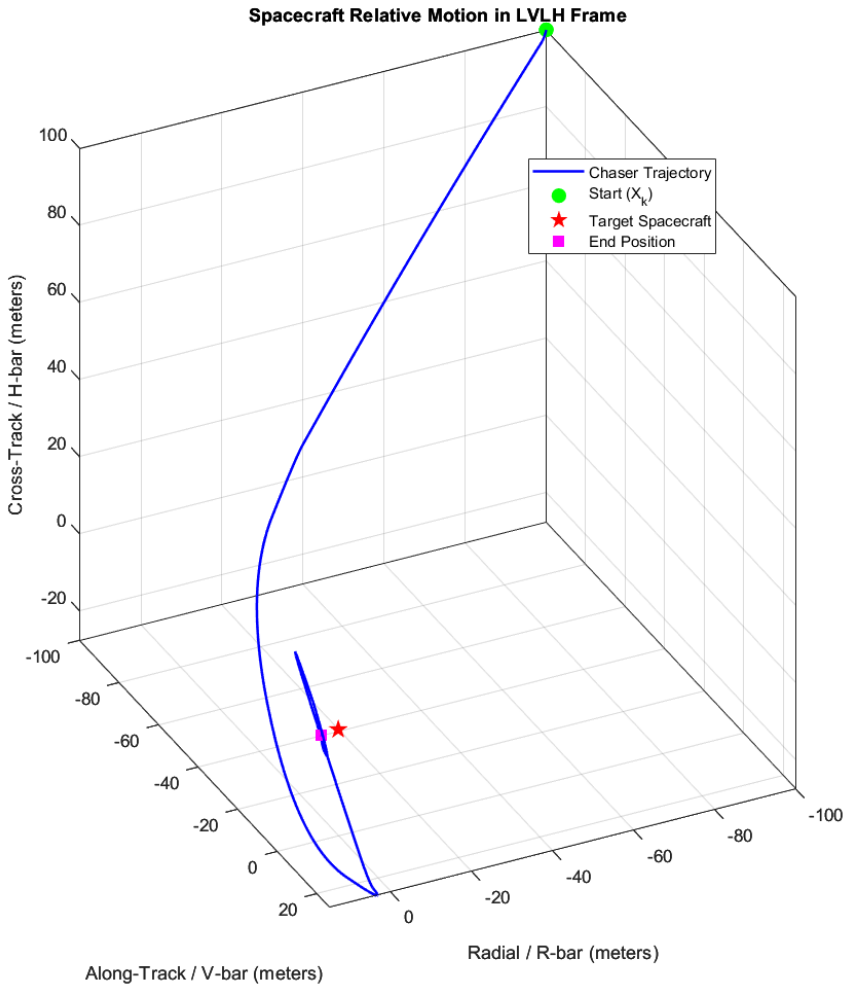


Figure 3.9: Current results of the nonlinear mpc view in the LVLH frame

The current simulations show that even with starting further away than other simulations and starting at a different attitude than the target cubesat, it still converges. Figure 3.9 shows an overshoot but not by much as it only goes 20 meters passed and then returns. Simulation may have run longer than needed as the chaser is going back and forth on trying to maintain the docking position which was set to 5 meters from the target cubesat.

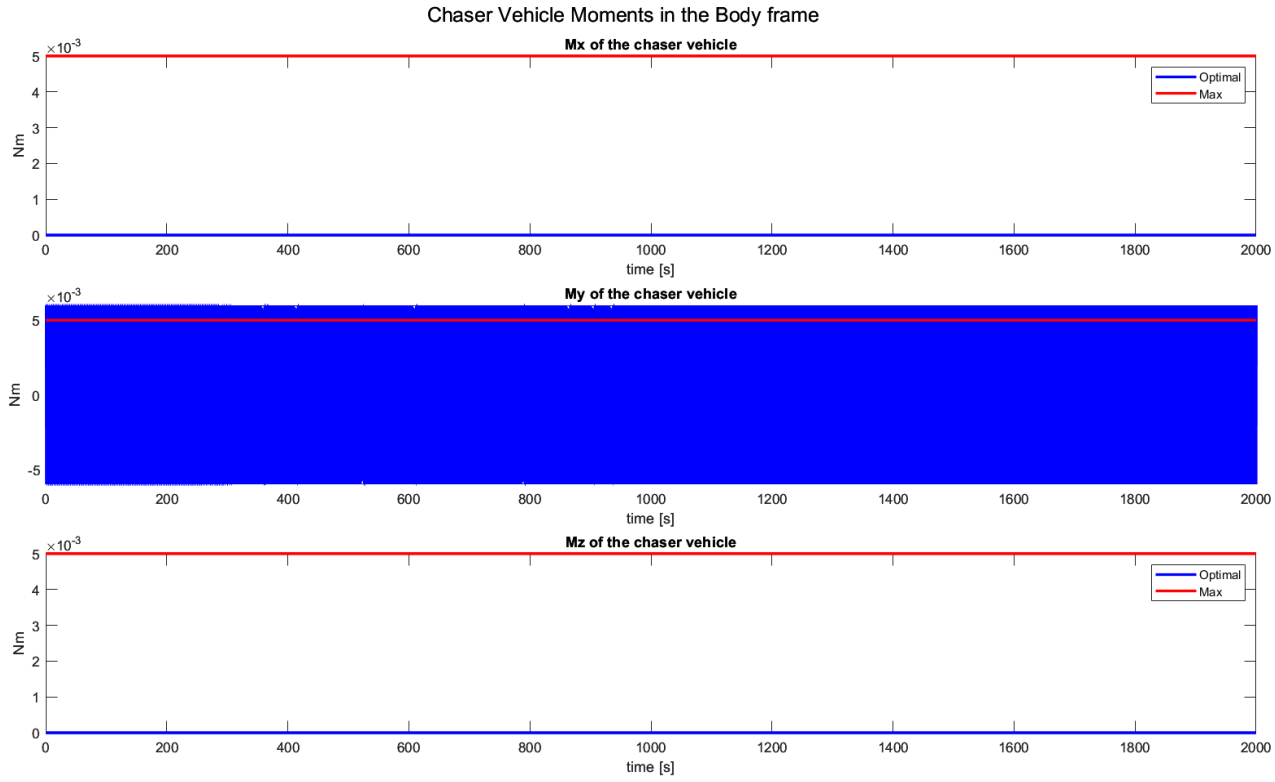


Figure 3.10: Current results of the mpc moments in the LVLH frame

Figure 3.10 shows a lot of min and max response on torque to achieve and maintain a constant pitch angle needed for docking. This might suggest the cubesat might be under actuated in attitude control. Though it should be noted that currently the max torque is only from the reaction wheels and does not have any assistance from the thrusters as that has not been modelled in yet.

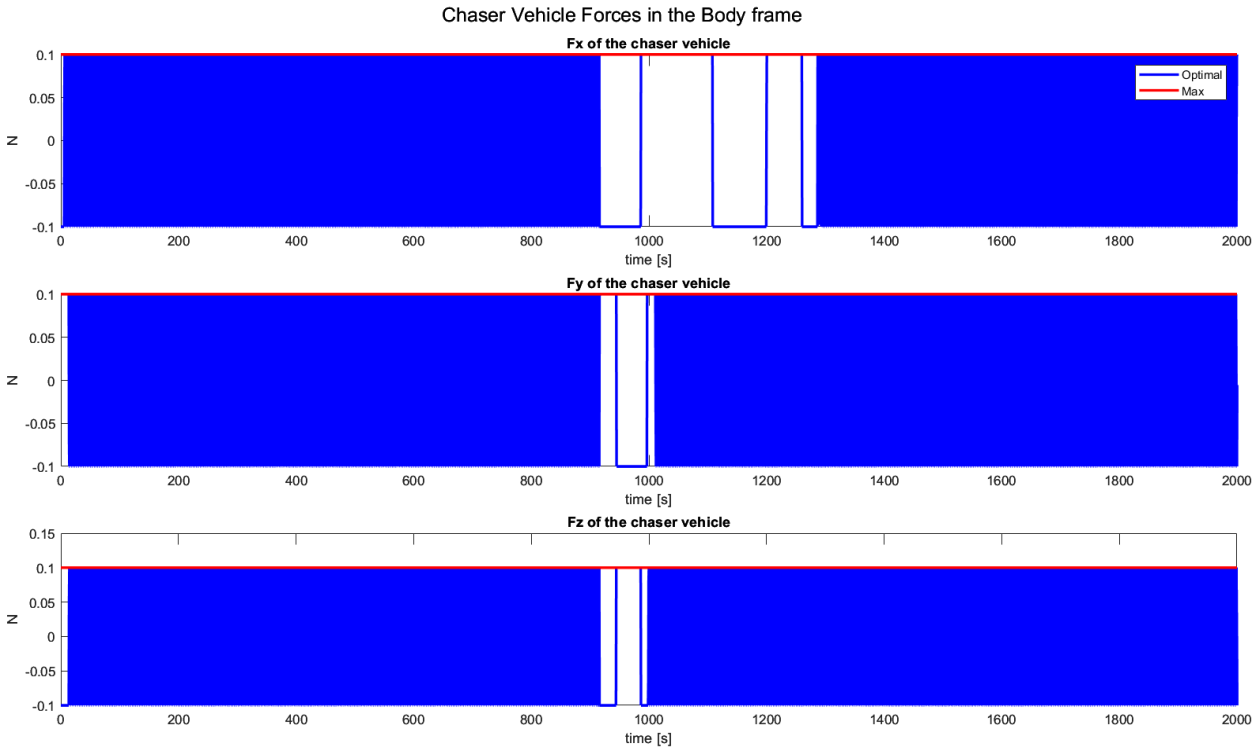


Figure 3.11: Current results of the nonlinear mpc forces in the LVLH frame

The forces also seem to saturate to achieve the necessary movements to reach the target cubesat. This may lead to a review of the weights used on the Q and R matrices, potentially loosening up on how strict the thrusters respond.

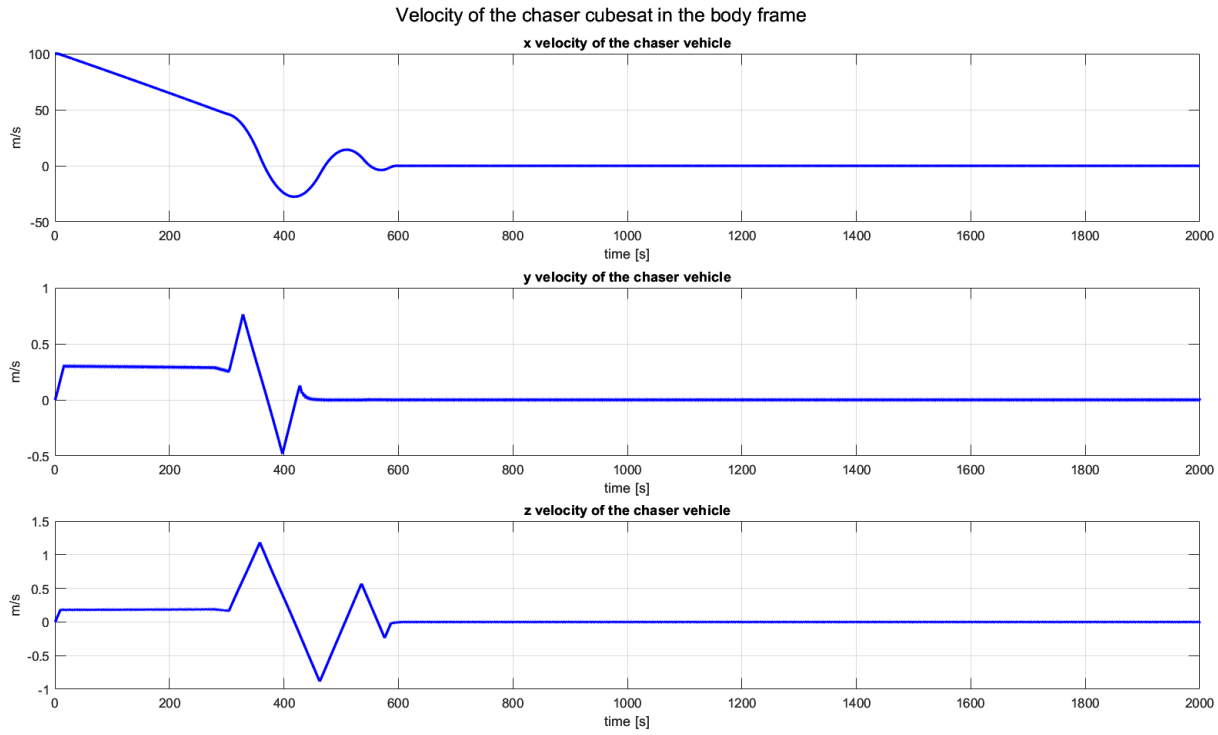


Figure 3.12: Current results of the nonlinear mpc velocity in the LVLH frame

The velocity seems to hit a maximum of 100 m/s in the x axis of the LVLH frame. This suggests a strong response to increase orbit altitude to match the target vehicle position. Y and Z velocity shows constant velocity than a couple of spikes, suggesting corrections needed to keep to the target position set from the reference trajectory.

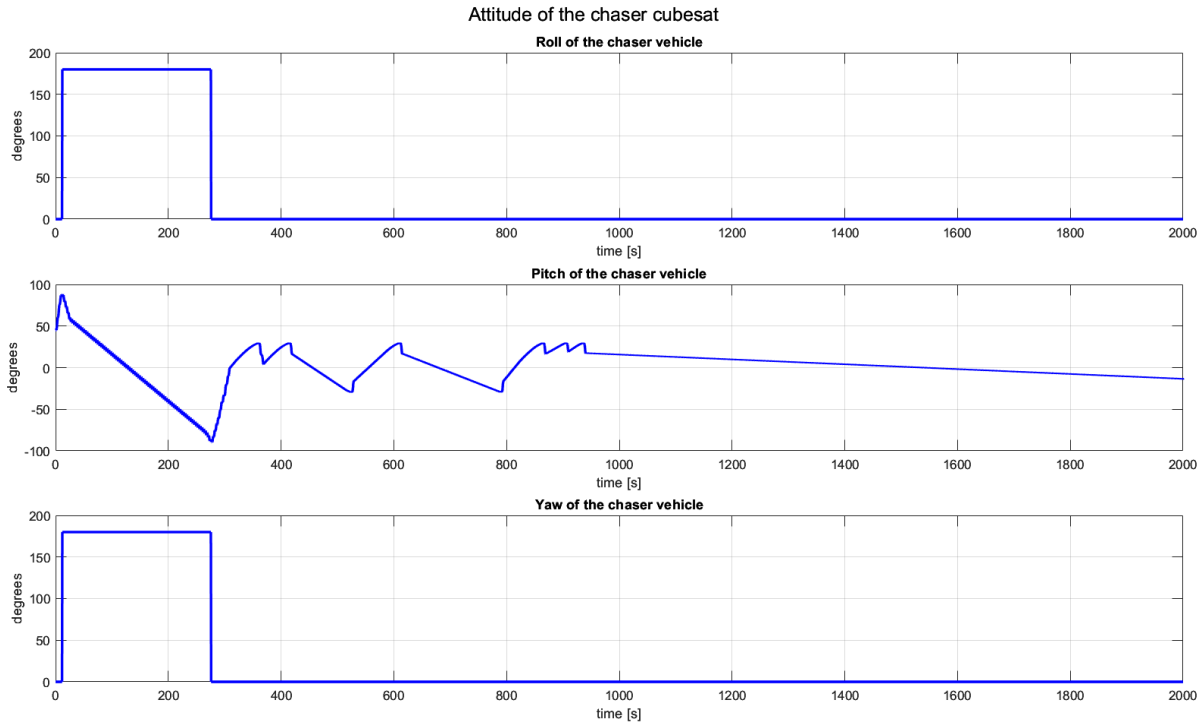


Figure 3.13: Current results of the nonlinear mpc attitude in the LVLH frame

Figure 3.13 shows the corrections done to the Euler angles of the cubesat to align with the target's reference alignment. The target cubesat reference alignment was kept to not deviate from the LVLH frame. There seems to be a response from in roll and yaw when the chaser cubesat is correctly for pitch, this will need to be resolved in future iterations. A lot of oscillation in pitch with the nonlinear model predictive controller attempting to correct for pitch angle, this suggests revisiting the **Q** matrix and elements relating to the quaternions.

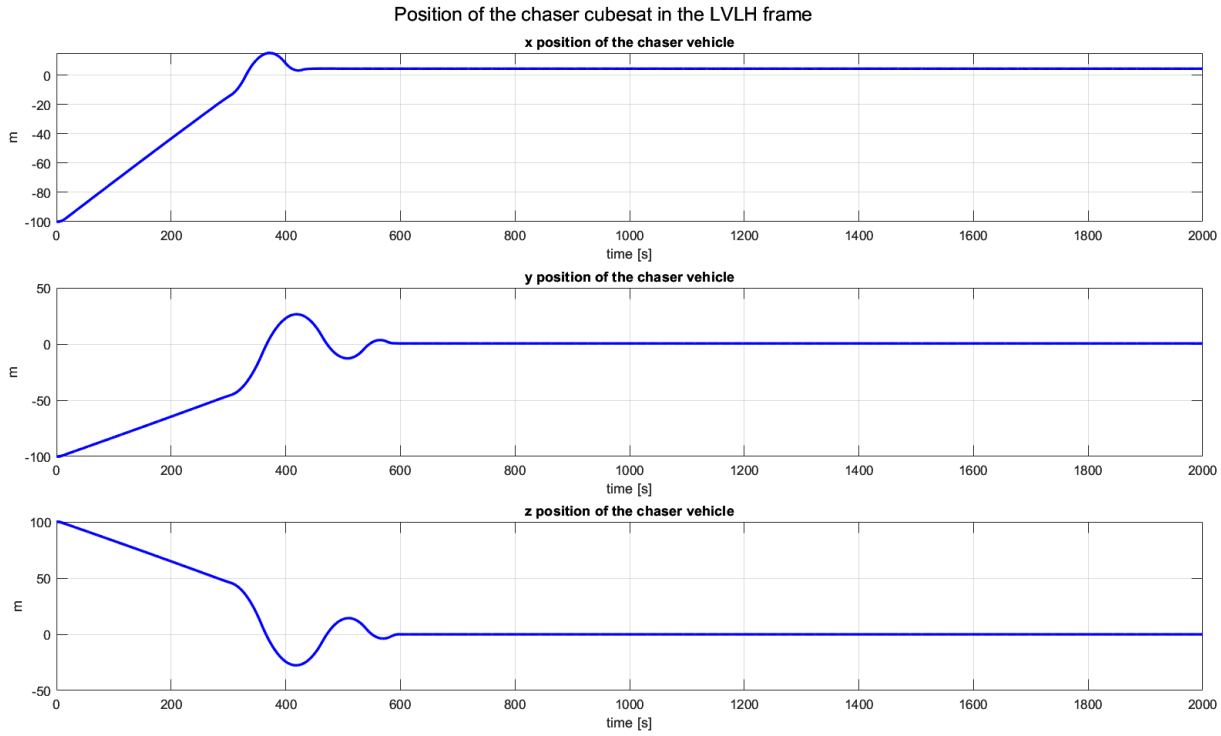


Figure 3.14: Current results of the nonlinear mpc position in the LVLH frame

Figure 3.14 shows the position change of the chaser cubesat relative to the target cubesat. The chaser vehicle does better at zeroing out the x position between the target vehicle than the other axis. This may be most optimal as it seems to prioritize changes in orbit altitude over fine adjustments in y and z in the LVLH frame. Oscillations seen on the position seem to occur after the 50 m mark, this suggests that the docking phase in the code has been activated and the new target position for the chaser cubesat is 5 meters from the target cubesat. The x position seems to show that it is slightly off zero which indicates an attempt to hold that 5 meter distance from the cubesat, this suggests the code is working well as the docking distance was set to 5 meters away from the target cubesat.

4. Implementation of the NMPC

4.1 Force and moment availability

Section 3 demonstrated architecture of the nonlinear model predictive controller in a simulink environment. It used a constant constraint on the maximum force and moments achievable shown in table 3.1. In order to increase the fidelity of the model, the orientation of the cubesat must be taken into account to correctly obtain the maximum force and moment available

Based on the thrusters orientation described in section 2.3, the maximum thrust and moment produced by the thrusters can be calculated in the body frame at the beginning of the simulation. A new matlab function block will need to be added before the nonlinear model predictive controller in order to determine the maximum force and moment available in the LVLH frame. The availability in the forces and moments are determined by the attitude of the cubesat at any given moment in the simulation.

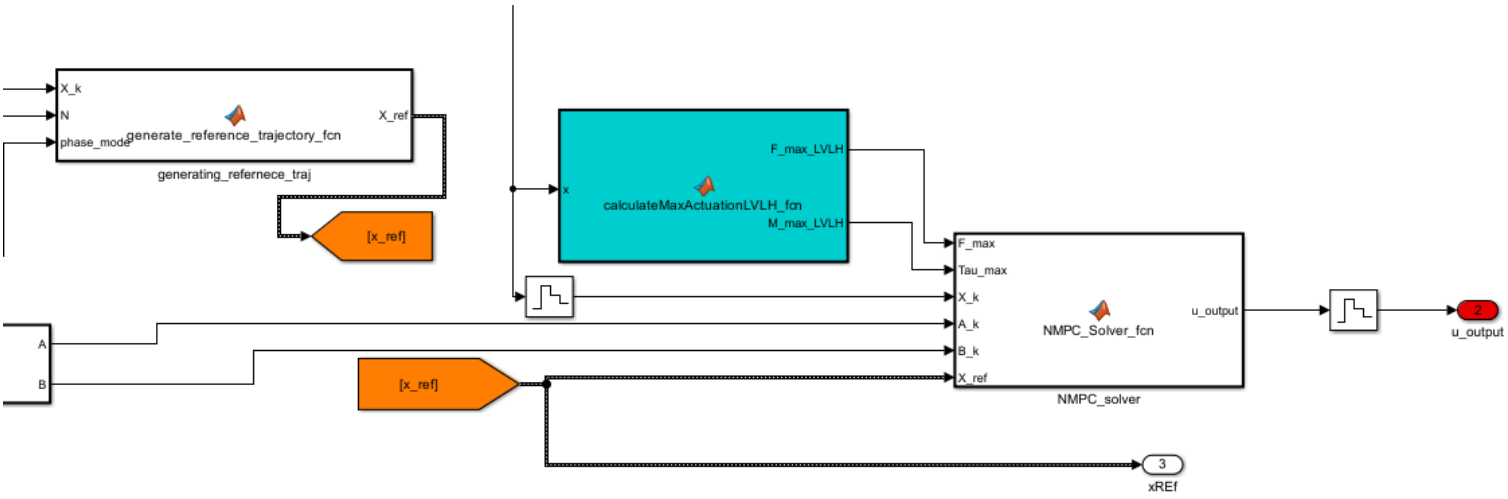


Figure 4.1: Max actuation block (cyan)

The max actuation block in cyan coloring is the new block added to the simulation presented in figure 3.1. Also modified is the Nonlinear MPC solver block, the changes include new inputs which will be the outputs from the max actuator block. Not only does the block calculate the maximum force and moment available from the thrusters, it also calculates the maximum moment that can be provided by the reaction wheels in the LVLH frame. This ensures

that the MPC solver is not requesting more than what is available in terms of control authority adding the fidelity of the simulation.

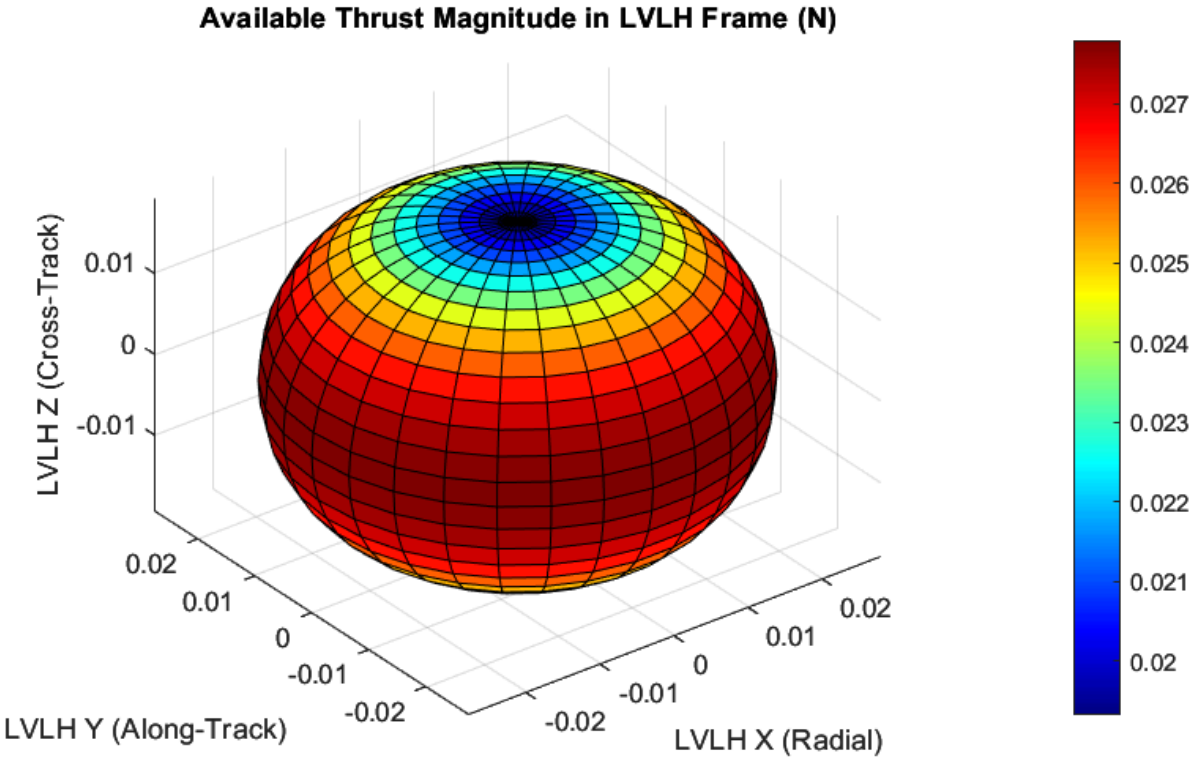


Figure 4.2: Thrust available, body Z-axis aligned with LVLH Z-axis

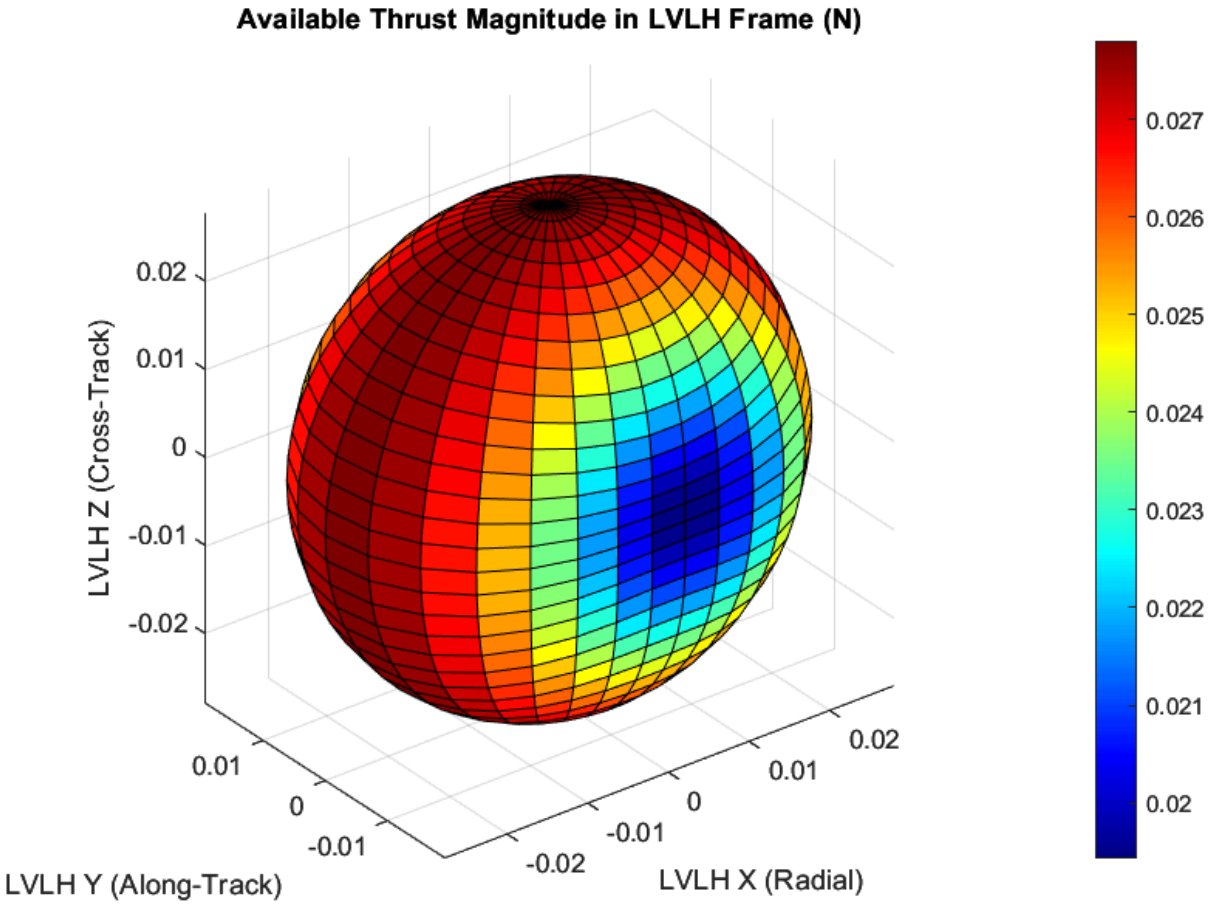


Figure 4.3: Thrust available, body Z-axis aligned with LVLH Y-axis

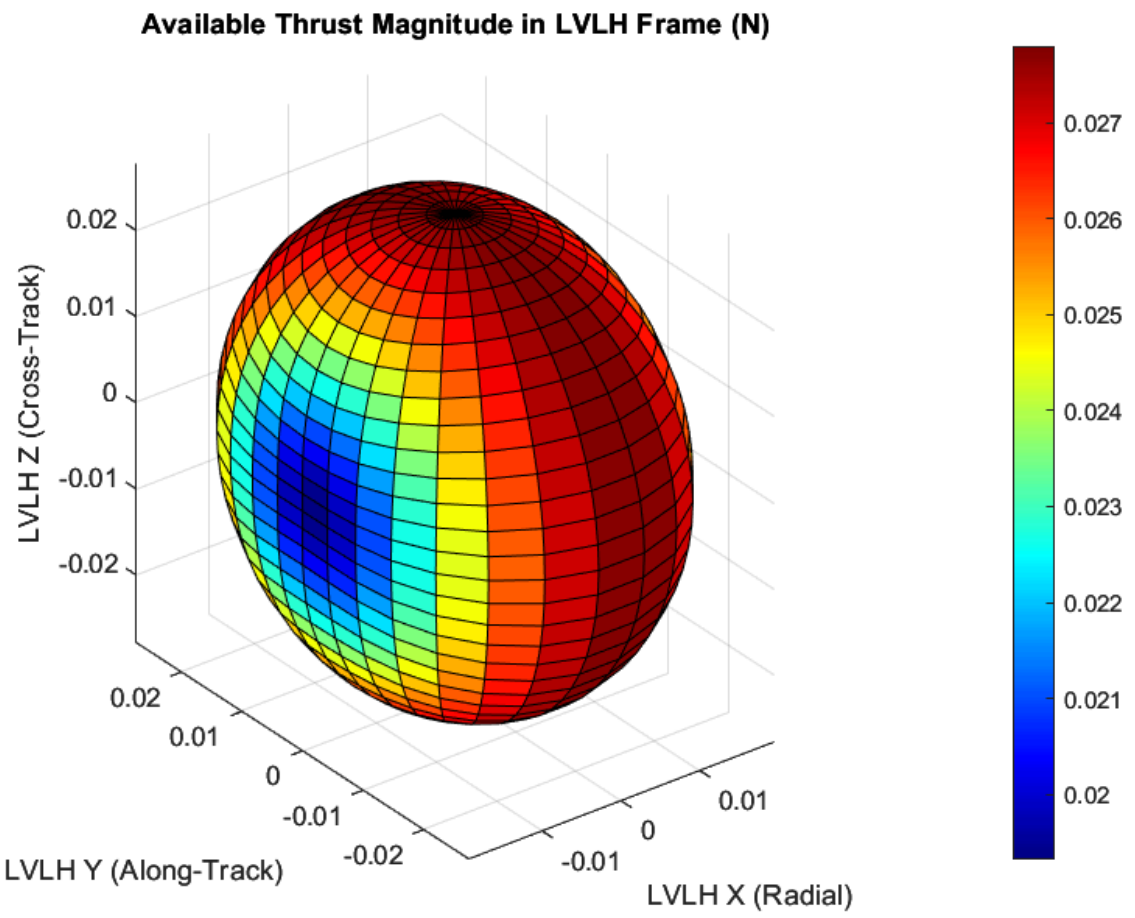


Figure 4.4: Thrust available, body Z-axis aligned with LVLH X-axis

Figure 4.2 to figure 4.4 demonstrate the maximum available thrust given the thruster configuration shown in figure 2.4 and 2.5. Each plot demonstrates which direction is the cubesat most strongest and weakest when aligned with one of the major axes in the LVLH frame. This is useful information as it depicts which orientation the cubesat should be in if high thrust is needed. That information is translated into which orientation the reference trajectory should request to the NMPC solver to maximize performance.

4.2 Adding atmospheric drag

This section will discuss adding nonlinearity to our model by incorporating atmospheric drag into our simulation. Equation 2.17 displays how atmospheric drag is calculated with most variables relatively easy to calculate. Some variables are not readily available, for example, the cross-sectional area **A** area term will need to be calculated by determining the direction of the velocity vector. The term $\frac{C_D\mathbf{A}}{m_c}$ itself is called the ballistic coefficient and for a 3U satellite is

roughly between 70-100 kg/m^2[17]. This will serve as a baseline to determine if our drag modeling matches other 3U cubesats.

As for the calculation of the atmospheric drag, the only variable to be constantly changing will be the cross-sectional area as the cubesat maneuvers to rendezvous and dock with its target vehicle.

Table 4.1: Recap of cubesat drag parameters

Parameter name	Value
Cubesat mass	5 kg
Drag Coefficient [17]	2.2
Cubesat Dimensions (LxWxH)	0.10 x 0.10 x 0.30 m
Minimum Area (small face)	0.01 m^2
Maximum Area (large face)	0.03 m^2

Table 4.1 recaps the constant variables used to determine the atmospheric drag. The minimum and maximum area is included as it is used in the code to ensure that the area is physically correct and can not be miscalculated to an unrealistic value. A combination of the cubesat dimensions and the velocity vector in the LVLH frame will be used to determine the effective area for drag calculations. All changes will be made to the equations of motion dynamics block where the input to the block is the x state vector shown in equation 2.20. The method of calculating the effective area begins with using the quaternions that are in the state vector to calculate the rotation matrix to convert values from LVLH frame to the body frame of the cubesat. With the rotation matrix calculated at each time step, the unit velocity vector in the body frame can be computed.

$$A_{drag} = \sum_{i=1}^{6} A_{drag,i} \max(0, \mathbf{h}_i \cdot \mathbf{v}_n) \tag{4.1}$$

Equation 4.1 calculates the effective area through summation of the projected area onto the velocity vector. The summation is through 6 iterations, each having to represent one face on a cubesat. The term $A_{drag,i}$ is the calculated area at each face on the cubesat and the term $\mathbf{h}_i$ represents the normal unit vector of that area. The term $\mathbf{v}_n$ is unit velocity vector that is calculated from the state vectors velocity vector and quaternions at each step. The effective area is then used in equation 2.17 to calculate the effective atmospheric drag.

After implementation of the drag model and implementation of a realistic force and moment availability, we will conduct the same simulation and compare to the results shown in section 3 of this report.

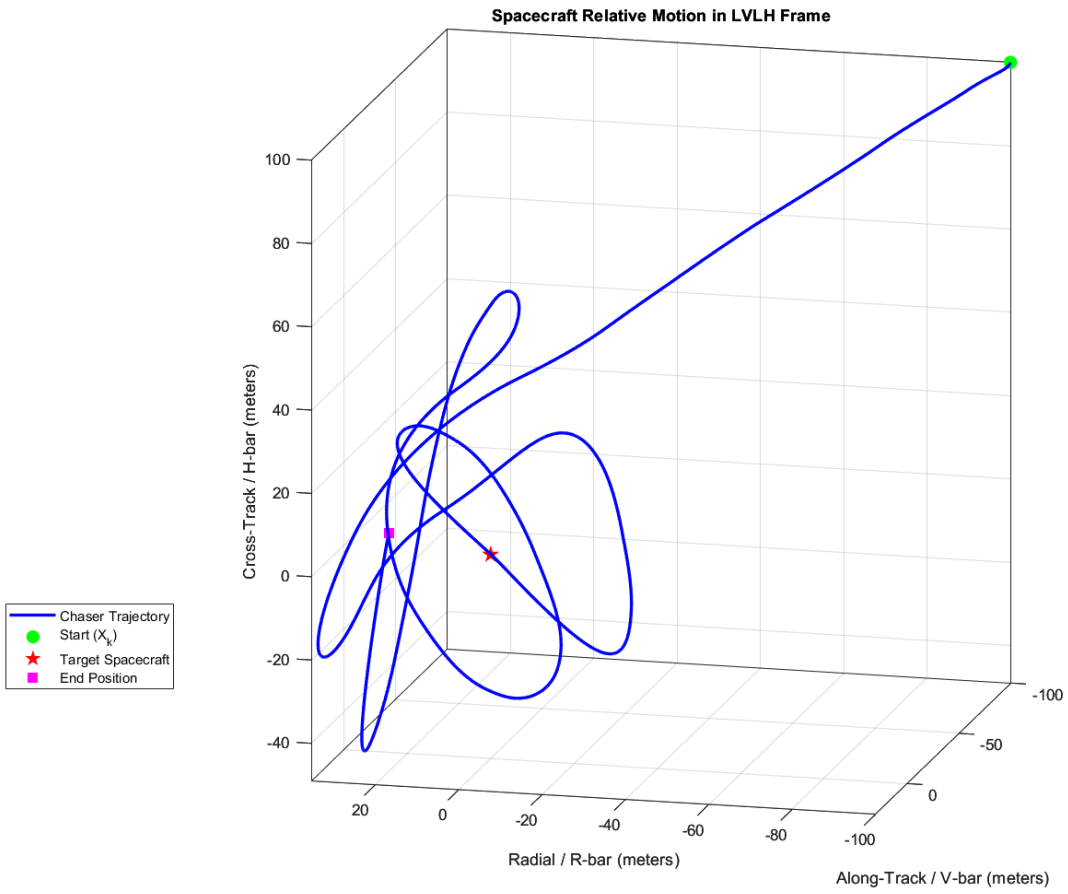


Figure 4.5: Results of the NMPC with atmospheric drag

Comparing results to figure 3.9, it seems that the nonlinear mpc is having more difficulties with station keeping for docking than before the new changes. Though it appears that the rendezvous part of the controller is working similarly as the cubesat has little difficulty reaching the target cubesat.

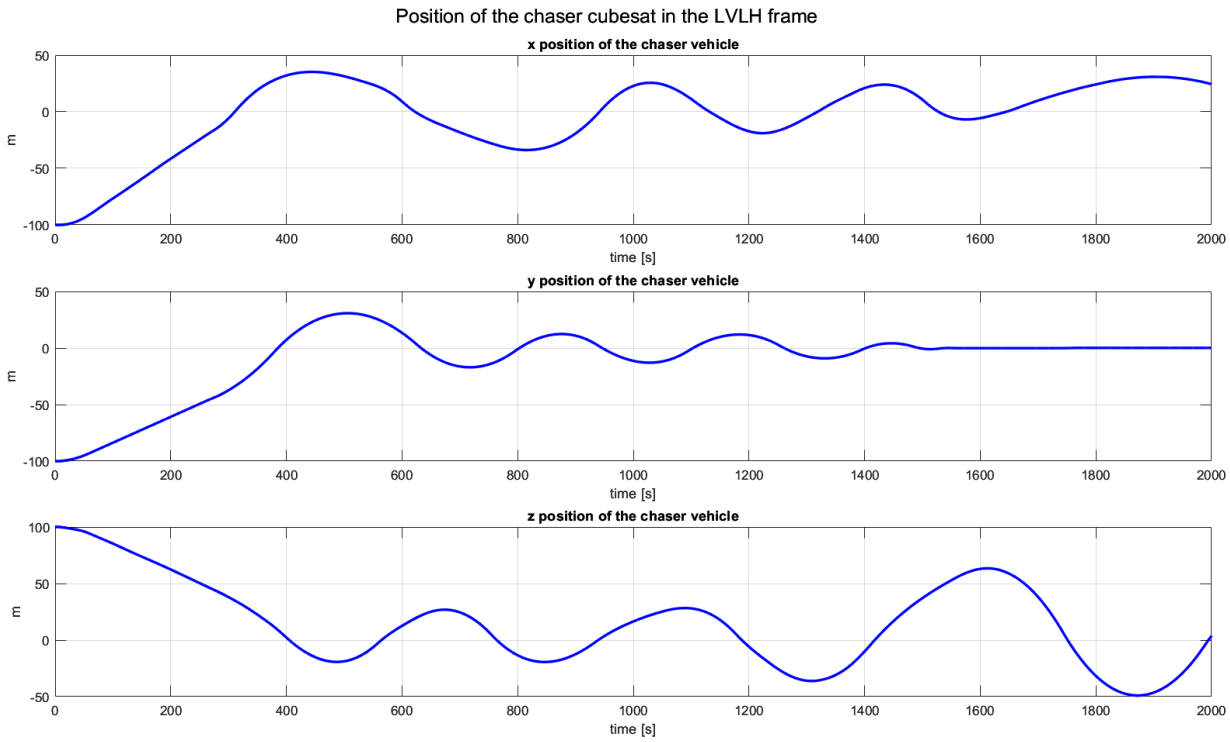


Figure 4.6: NMPC position in the LVLH frame with atmospheric drag

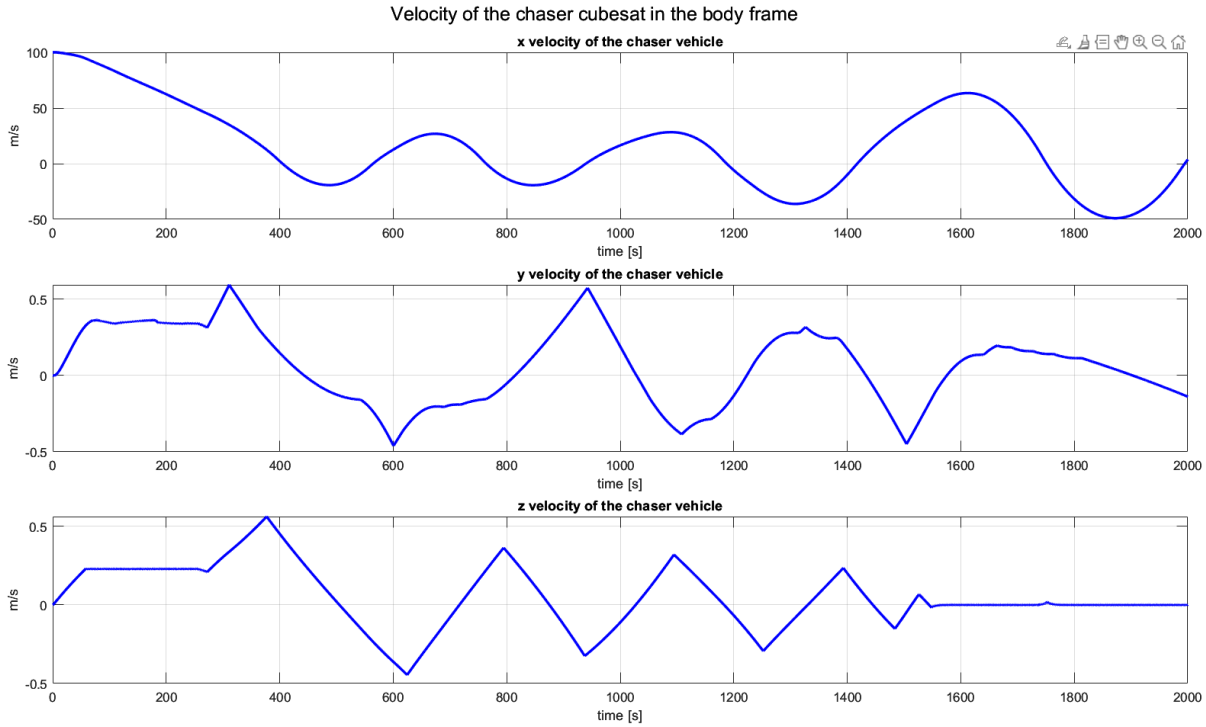


Figure 4.7: NMPC velocity in the LVLH frame with atmospheric drag

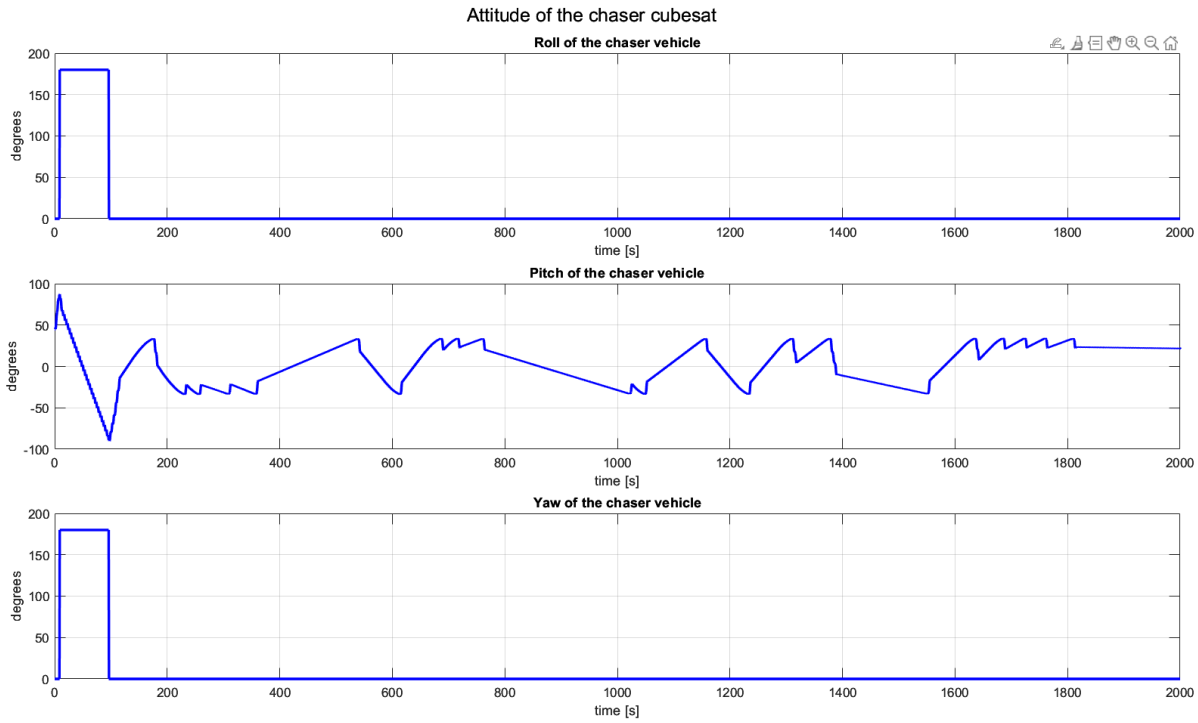


Figure 4.8: NMPC cubesat attitude in the LVLH frame with atmospheric drag

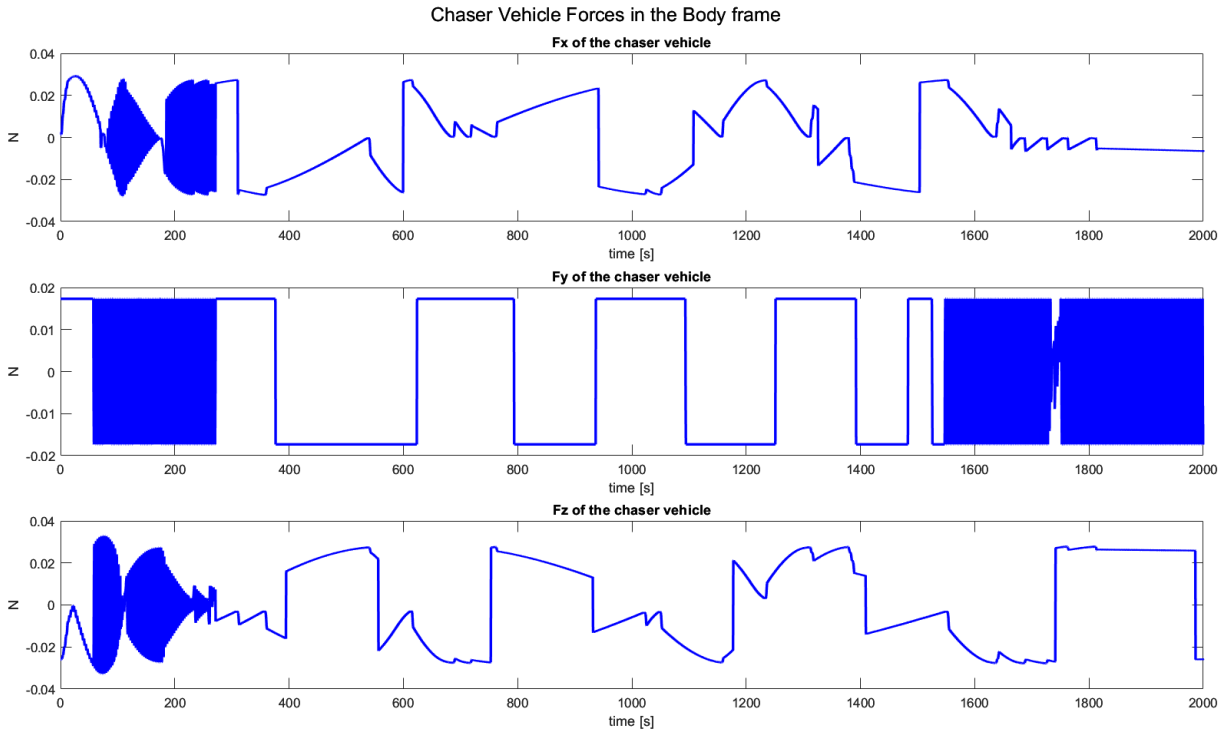


Figure 4.9: NMPC cubesat forces in the LVLH frame with atmospheric drag

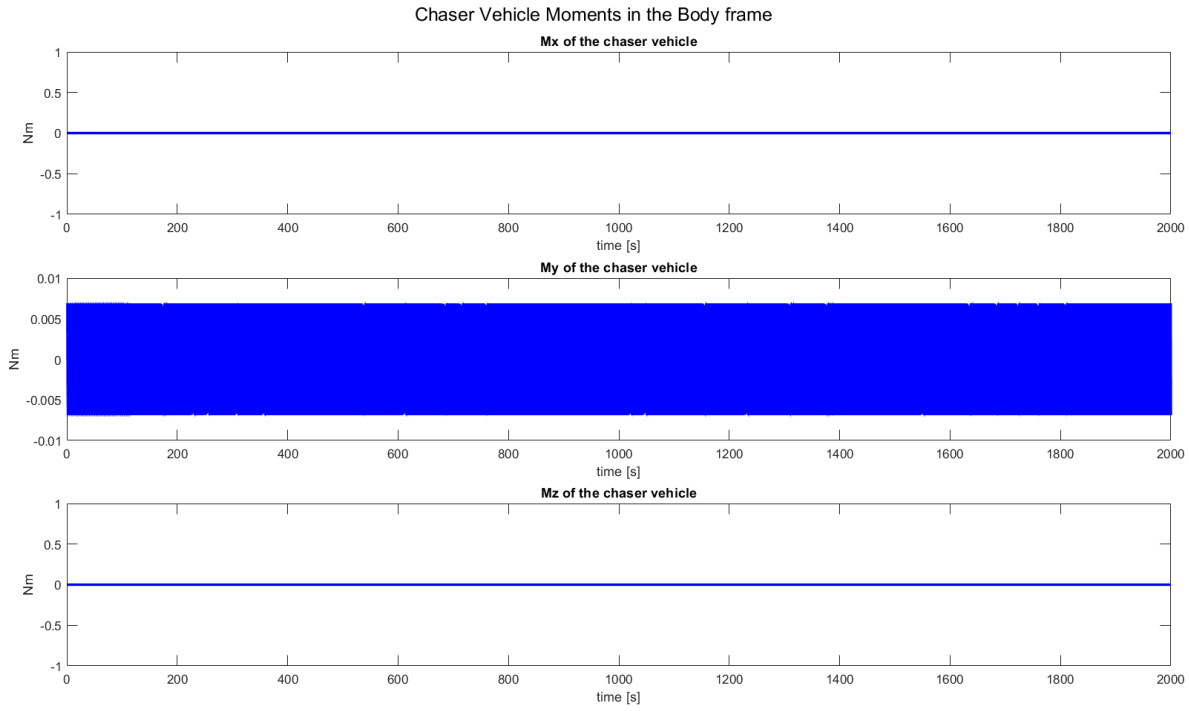


Figure 4.10: NMPC cubesat moments in the LVLH frame with atmospheric drag

One key difference is the improvement in realistic forces being commanded by the nonlinear mpc. Comparing figure 4.9 to figure 3.11, the forces are no longer commanding between maximum and minimum force. Seems the improvements applied in section 4.1 have made it for the controller to not saturate on forces commanded. Though it is noted that saturation still occurs on the y-moment control for cubesat. This indicates too high weights on the attitude precision in the R matrix.

4.3 Adding J2 gravitational perturbation

The J2 effect is one of the primary non-spherical perturbations of Earth's gravity field, which is the harmonic coefficient due to the Earth's oblateness [31]. This effect is the most dominant among the other coefficients for orbits near earth [31]. Since the target cubesat and chaser cubesat are in LEO, the Earth's oblateness will impact them with significant gravitational disturbances. To further increase the fidelity of the project's simulation and to capture the nonlinear dynamics between the cubesats the j2 effect must be modelled and added to the equations of motion derived in equations 3.2 to 3.4.

$$\mathbf{P}_{J2} = \frac{3J_2\mu R^2}{2r^4}\left[\frac{x}{r}\left(5\frac{z^2}{r^2}-1\right)\hat{\mathbf{i}} + \frac{y}{r}\left(5\frac{z^2}{r^2}-1\right)\hat{\mathbf{j}} + \frac{z}{r}\left(5\frac{z^2}{r^2}-3\right)\hat{\mathbf{k}}\right] \tag{4.2}$$

Equation 4.2 represents the j2 acceleration effect on the xyz axis in the Earth Centered inertial frame [31]. This effect is seen on both the target and chaser cubesat, since the dynamics calculated are in the LVLH frame, a differential between the two is calculated and added to equations 3.2-3.4.

$$\Delta\mathbf{a}_{J2} = \mathbf{a}_{J2,\text{chaser}} - \mathbf{a}_{J2,\text{target}} \tag{4.3}$$

Equation 4.3 represents the differential acceleration between the chaser and target vehicle. The differential equation is used since both vehicles are experiencing the J2 effect. To simplify the differential equation, the target cubesat will be in an equatorial orbit, giving the target vehicle only a J2 effect in the x_axis. For the first simulations only a small offset will be used for the cross-track thus simplifying the equations further for the chaser cubesat.

$$a_{J2,t,x} = -\frac{3}{2}J_2\left(\frac{\mu R_e^2}{R_t^4}\right) \tag{4.4}$$

$$a_{J2,c,x} = -\frac{3}{2}J_2\left(\frac{\mu R_e^2}{r_c^4}\right)\frac{R_t + x}{r_c} \tag{4.5}$$

$$a_{J2,c,y} = -\frac{3}{2}J_2\left(\frac{\mu R_e^2}{r_c^4}\right)\frac{y}{r_c} \tag{4.6}$$

Since simulation will only run orbits with negligible cross-track offset from the cubesat, the z term in equation 4.2 is zero and equations 4.4, 4.5, 4.6 are the accelerations affecting the chaser and the target cubesat. The only differential to calculate is the x-terms between the chaser and target vehicle. The differential between equation 4.4 and 4.5 is added to the right hand side of equation 3.2 and equation 4.5 will be added to the right hand side of equation 3.3.

4.4 Modeling fuel consumption

Section 3 of this report demonstrated the performance of a nominal model predictive controller with mass held constant. This section will discuss how modeling fuel consumption will improve the fidelity of our model. In order to incorporate the fuel consumption into the model presented in section 3, a few of the matlab blocks will need to be altered.

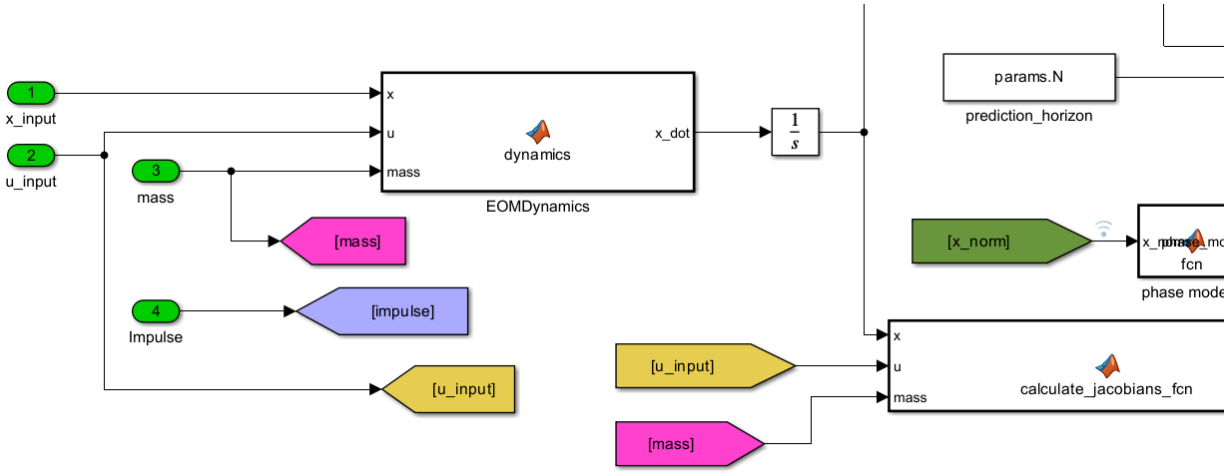


Figure 4.11: Changes to equation of motion block

The first block to be altered is the equations of motion block. In this block the mass of the chaser cubesat is now an input to the equations of motion. Since mass will be an ever changing variable with fuel consumption being modeled it will have to be an input to the equation of motion block. Also in the calculation of jacobians block, mass will have to be an input as well since in that block it also uses the dynamics matlab function.

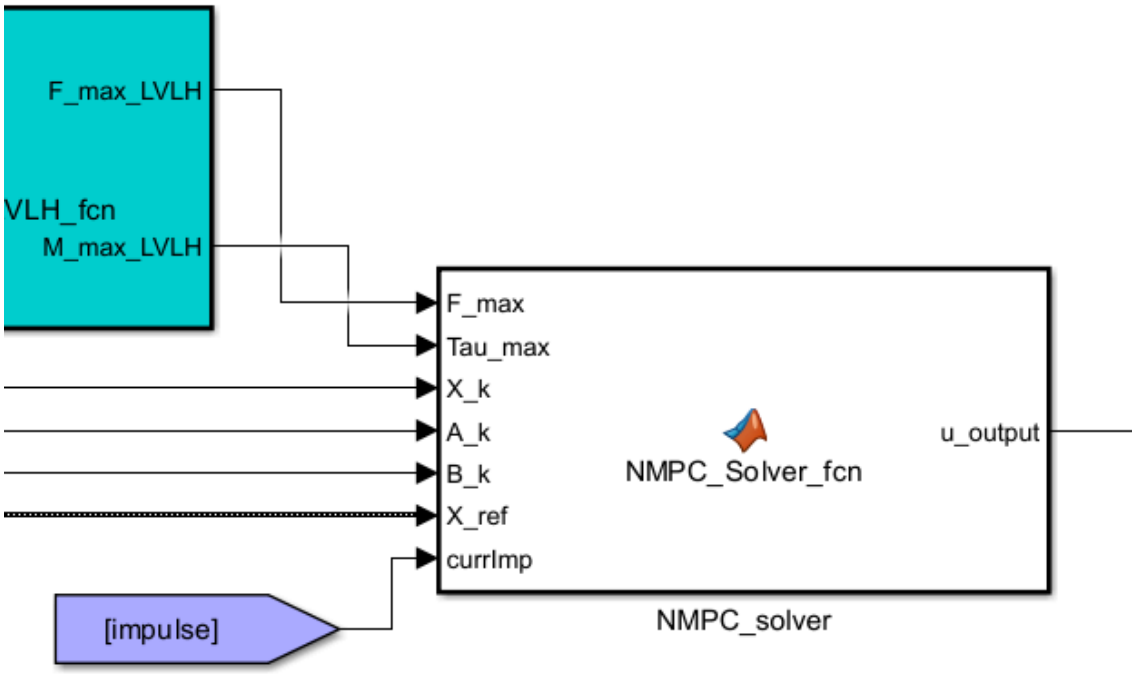


Figure 4.12: Changes to nonlinear model predictive controller block

Figure 4.12 shows that the current impulse is needed to be added as an input to the nonlinear model predictive controller solver. This change isn't needed for fuel consumption modeling but will be needed for future gain tuning. The solver will need to know the amount of impulse left as fuel is consumed in order for future gain tuning. The solver will need to adjust the cost function gains as fuel is being consumed, this will be discussed in the next chapter.

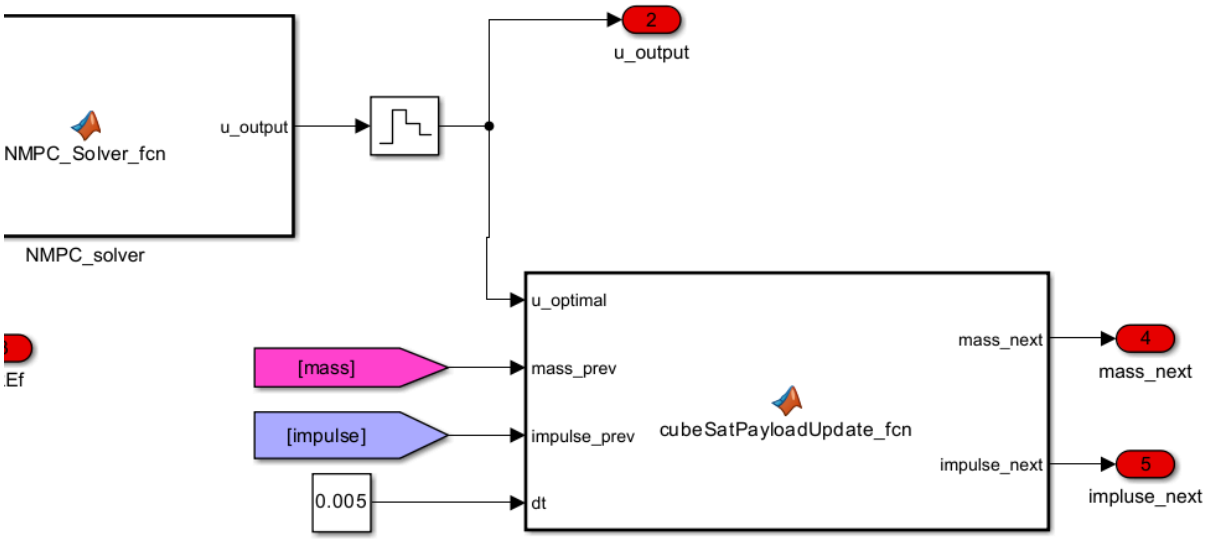


Figure 4.13: Introduction of cubesat mass block

Figure 4.13 shows the new matlab function block that was added into the simulation. In this block, the fuel consumed is calculated and outputs the new mass based on the fuel consumed by the optimal force output from the nonlinear model predictive controller solver. This function takes in the mass and impulse used for the dynamics and takes the nominal control output to calculate the reduction in mass and available impulse of the cubesat for next timestep.

$$\dot{m} = \frac{\mathbf{F}_{\text{thrust}}}{g_0 I_{sp}} \tag{4.7}$$

Equation 4.7 shows the calculation of mass flow rate via the division of the magnitude of thrust by the multiplication of gravity and specific impulse. The mass flow rate is then multiplied by the sample time of simulation to retrieve the mass reduction of fuel per each timestep. The calculation of reduction in available impulse is calculated by multiplying the sample time step by the magnitude of the thrust vector. The mass and impulse reductions are tracked to add a protection to the simulation, in the event mass or available impulse reaches zero, all thrusting is set to zero.

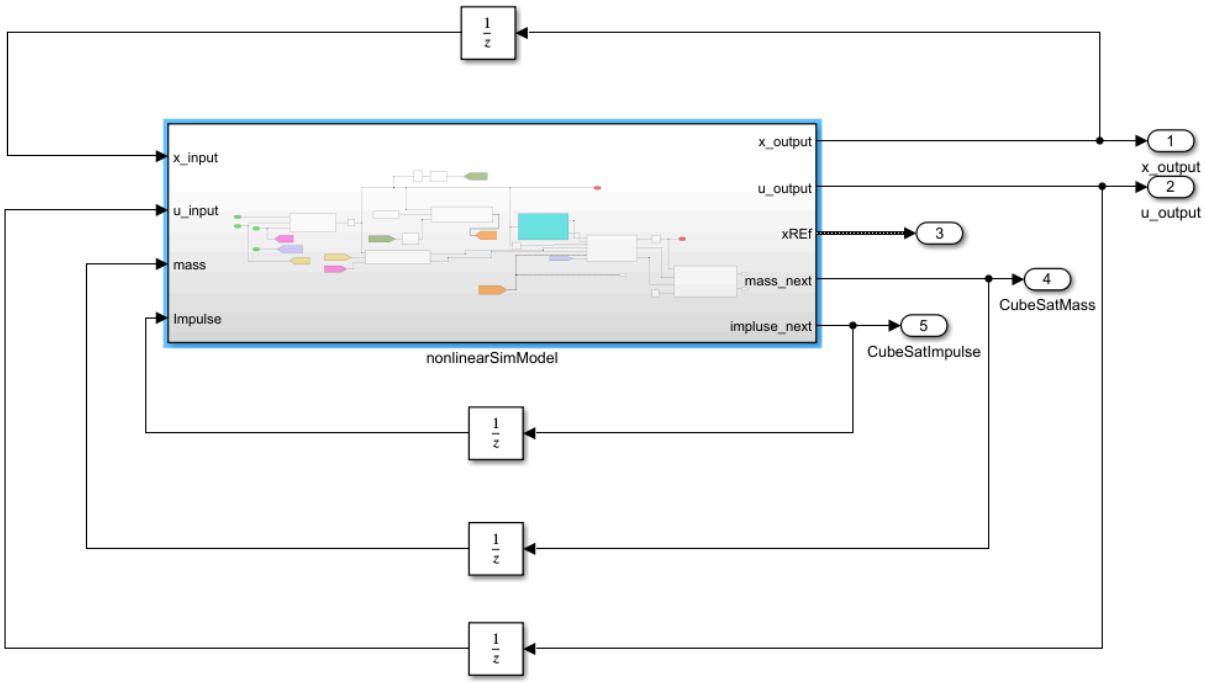


Figure 4.14: Overall changes to full nonlinear model

Figure 4.14 demonstrates the external changes needed to the full nonlinear model to add fuel consumption to the simulation. Comparing figure 4.14 to figure 3.2, a key difference is in the two new inputs and outputs to the model. The variables are the mass and impulse. As shown in figure 4.13, the updated mass and remaining impulse are calculated at the end of the simulation, this in turn is fed back into the model as the new inputs to the simulation. Delay blocks are added with initial conditions hard coded in them. At the first timestep the mass and impulse feedback into the model is merely the starting conditions of the cubesat.

4.5 Refining mission phases

By looking at the results shown in section 4.2, it is clear that one set of gains is not sufficient to create a sufficient nonlinear model predictive controller. Figure 4.7 shows constant fluctuations on pitch attitude of the spacecraft. While the attitude of the spacecraft is important during the docking phase, it is not as crucial during the rendezvous phase. The gains on the R matrix can be relaxed during the rendezvous phase and increased during the docking phase. Fuel consumption also added another layer of optimization needed for the cost function. As fuel is depleted a penalty can be imposed on the gains in the R matrix related to thrust to minimize fuel consumption. Currently the simulation has been utilizing two phases of the mission with one set of gains. Those two phases are the rendezvous and docking phase. Each phase is dictated by the normal distance between the target and chaser cubesat. This section will review the phases of the mission and how the solver can be optimized at different phases of flight.

Many rendezvous missions incorporate many different phases in the rendezvous mission before a straight line approach for the docking phase [31]. For most of this report the rendezvous phase has been simulated to be close to the target before the MPC simulation is run. Each attempt to simulate beyond 1 kilometer has resulted in low performance of the nonlinear MPC solver. New strategies are needed to further increase the capabilities of the nonlinear MPC for a long rendezvous and docking mission.

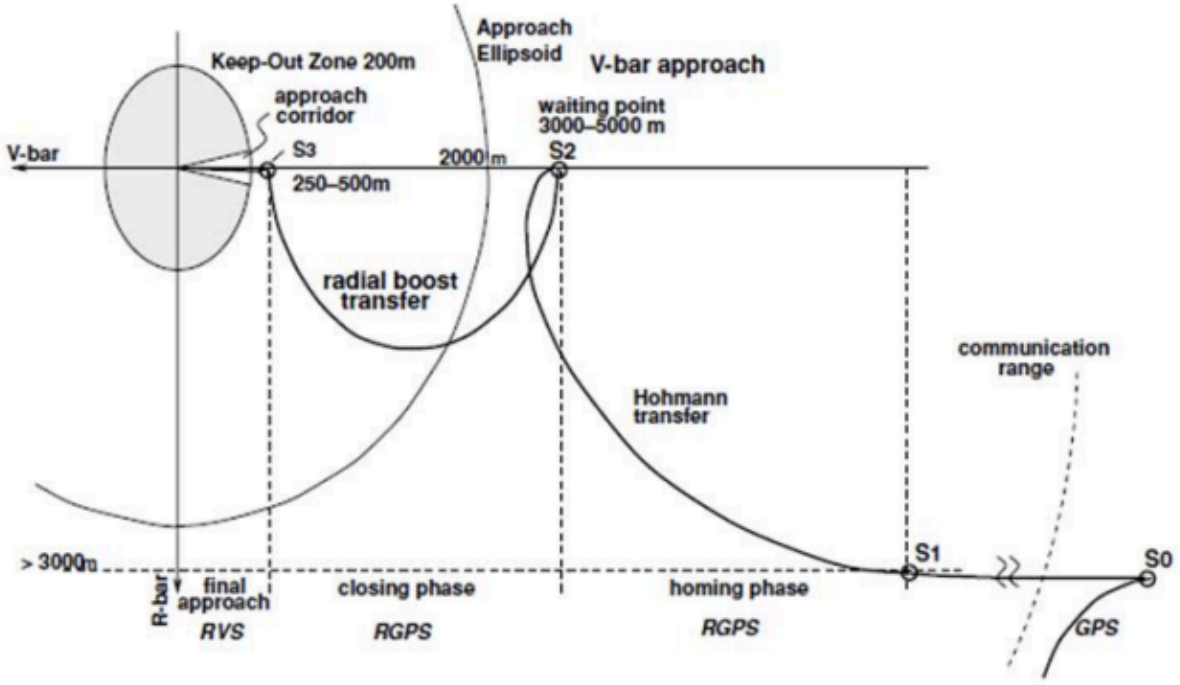


Figure 4.15: Example of V-bar approach [31]

Figure 4.11 demonstrates an example of the V-bar approach strategy for the European ATV project to ISS [31]. This approach breaks down the rendezvous and docking mission into 5 phases [31]. The first phase places the chaser vehicle within communication range which is 3-5 kilometers away from the target vehicle [31]. Phase 2 requires the vehicle to conduct a Hohmann transfer so that it is 2 kilometers from the target vehicle [31]. Phase 3 requires the chaser conduct a radial boost transfer to bring the chaser vehicle within 250-500 meters of the target vehicle [31]. Phase 4 is the final approach where the chaser vehicle will attempt to dock with the target vehicle. The nonlinear MPC developed in section 3 and 4 have been with the focus on phase 3 and 4. Incorporating new phases will also require a new set of gains for each phase. Each new phase will also need to have its reference trajectory determined so that the target hold position can be adequately evaluated within the MPC solver.

Table 4.2: Mission constants for simulation

Phase #	Mission Phase	Mission Region
Phase 3	Homing Phase	5000 m < Target > 2000 m
Phase 2	Closing Phase	2000 m < Target > 1000 m
Phase 1	Final Approach Phase	1000 m < Target > 50 m
Phase 0	Docking Phase	50 m < Target > 1 m

Table 4.2 lists the phases that will be implemented on the project instead of the two phases of rendezvous and docking. Each phase will introduce a new set of gains for optimizing the force and moment commands. Another area that will be impacted with this introduction will be the reference trajectory as that can be altered to fit the nominal x state needed for the MPC solver to solve.

5. Refining nonlinear MPC

5.1 Refining reference trajectory

In section 3.2 of this report, the simulink set-up was discussed and in that set-up was the way the reference trajectory was being used for the nonlinear model predictive controller. That set-up was a simple line trajectory tracking the position of the chaser vehicle to the target vehicle. This set-up was used to focus on close targets for shorter simulation time. Since the reference trajectory was set-up as a straight line the MPC solver demands high forces and moments since the trajectory is in constant error. While a straight line nominal reference trajectory may be sufficient when operating in the final approach phase or closer, it does not work well for mission phases farther out.

The role the reference trajectory plays with the MPC solver is providing a reference state for the solver to track. This means each phase has a nominal path and state must be determined for the solver to achieve. Currently the reference trajectory uses a reference position to target and the MPC solves until that error between the target position and the vehicle position is zero. Other considerations are the other states, such as the velocity and quaternions. Currently the velocity in the reference trajectory has velocity limits as well defining how fast the vehicle is allowed to move in a given mission phase.

Table 5.1: Reference trajectory speed limits

Mission Phase	Speed limits
Homing Phase	10 m/s
Closing Phase	2 m/s
Final Approach Phase	0.5 m/s
Docking Phase	0.3 m/s

5.2 Line of sight constraint

The nonlinear MPC solver until now has been using an arbitrary point in space to target as it rendezvous and docks. This is sufficient for development but when it comes to more specific details of docking, more constraints are needed to be applied.

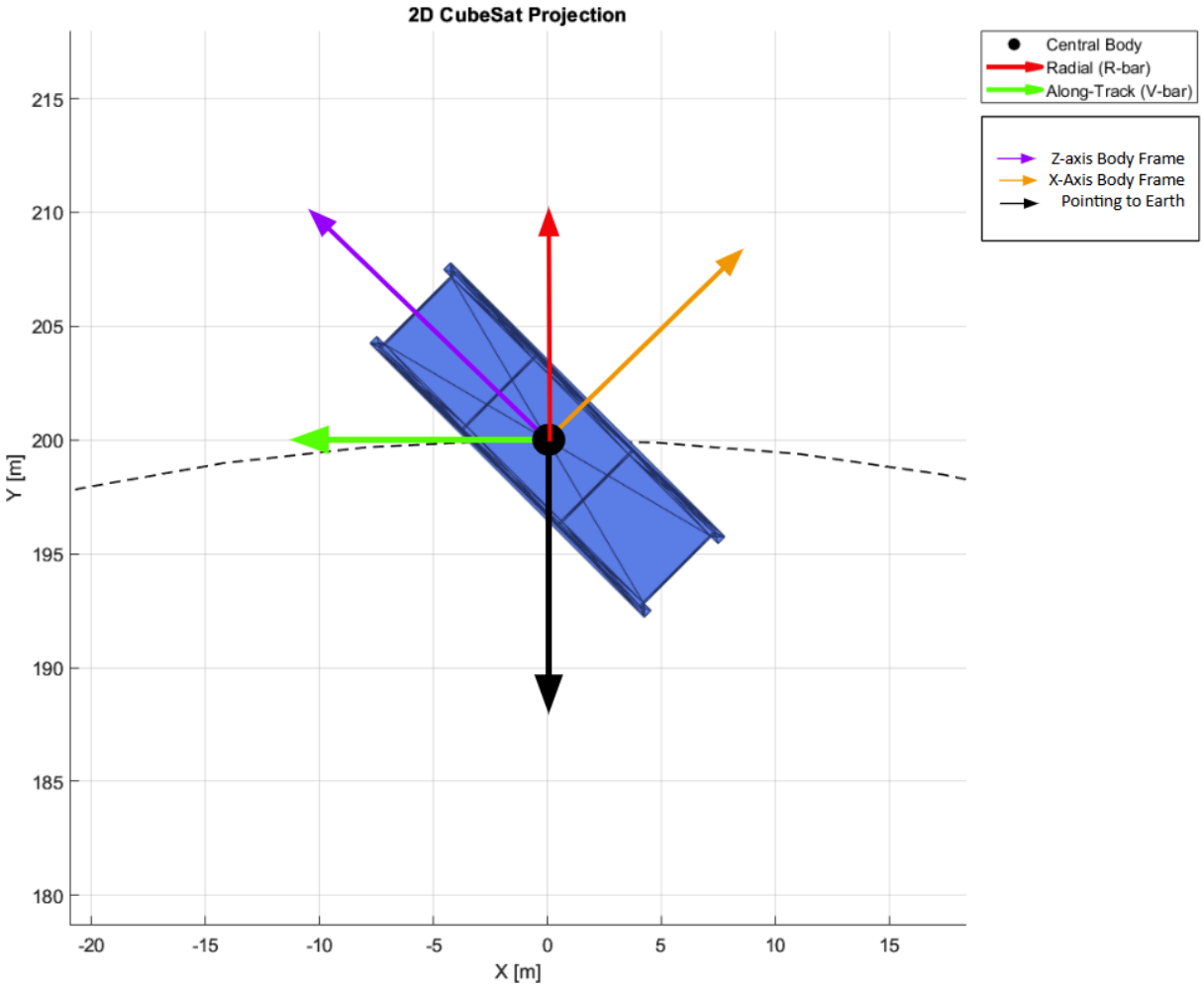


Figure 5.1: Cubesat orientation relative to LVLH frame

Figure 5.1 displays the orientation of the cubSat body frame relative to the LVLH frame. In figure 5.1, The cubesat body frame is a single rotation about its y-axis which is coincident with the z-axis of the LVLH frame. This only represents the ideal frame translation when in the rendezvous phase and docking phase as it gets closer to mating with the target cubesat. Based on figure 4.2, having the body frame z-axis aligned with the LVLH frame z-axis allows the chaser vehicle to produce max thrust in the x and y axis in the LVLH frame. Also to avoid adding another matrix transformation, allocation and constraints were solved with the body frame and LVLH frame aligned in every axis. The orientation of the cubesat is of importance to the MPC solver as it will have to incorporate the final resting position for docking. In order for the MPC solver to begin to incorporate this, a final docking position must be selected. For simplicity of this report, the specifics of the mating mechanism for the docking will not be explored, it will be assumed to be a magnetic contact surface.

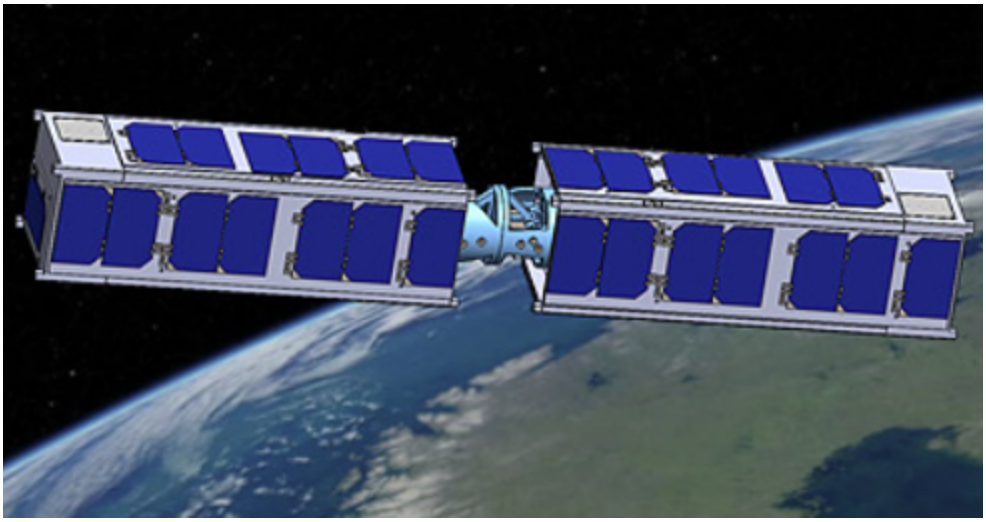


Figure 5.2: Example of cubesat docking [32]

Figure 5.2 demonstrates a magnetic contact between two cubesats. This will be the strategy for the simulation to have the chaser vehicle connect via the z-axis to the target vehicle. The connection point will be the bottom of the target vehicle to the top of the chaser vehicle. In order to improve the MPC solver for this docking, a line of sight constraint must be added.

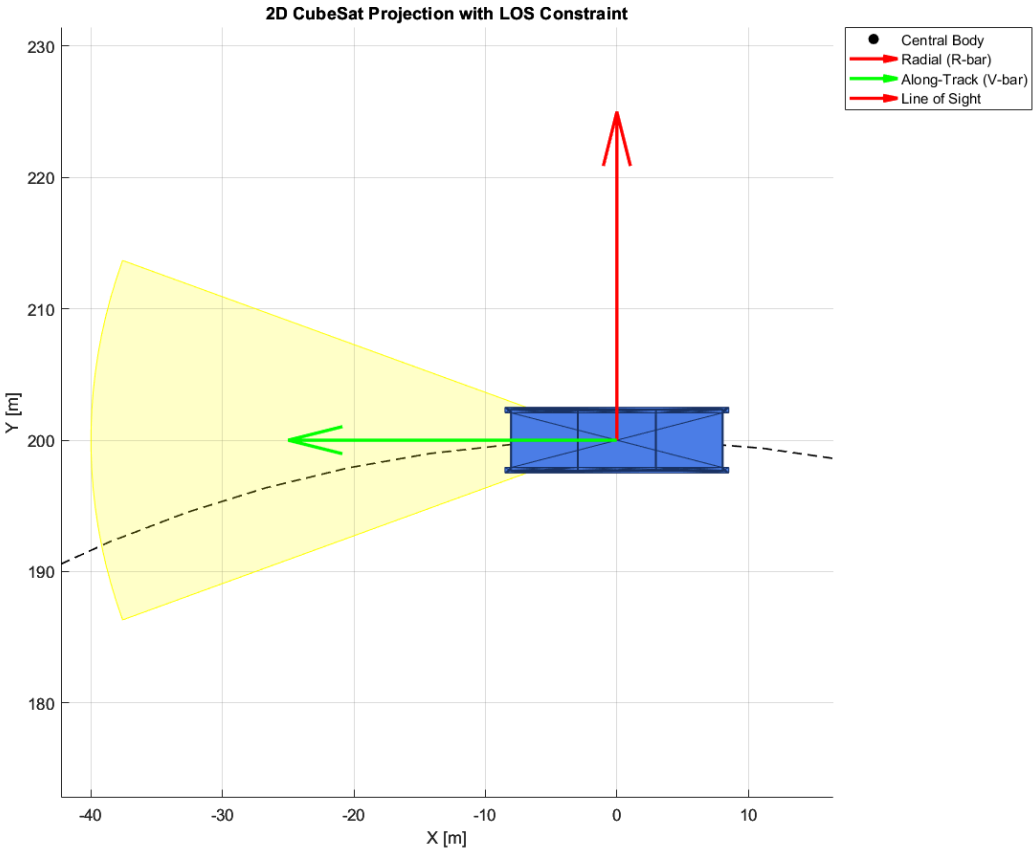


Figure 5.3: Line of sight shaded region

A line of sight (LOS) constraint is typically defined as a cone with a half-angle θ_{los}[15]. The half angle is defined as the angle from the origin, therefore the whole shaded region covers an area with the angle $2\theta_{los}$.

$$\sqrt{x^2 + y^2} \leq z \tan(\theta_{los}) \tag{5.1}$$

Equation 5.1 lists the constraint that will be added to the MPC solver when in the docking phase of the simulation. This constraint will be added to the simulink nonlinear MPC solver to ensure that when the chase vehicle is in the docking phase that it also prioritizes staying within the shaded region of the target vehicle. As to not affect other constraints, this will only be active when in the docking phase.

The solver used in the MPC is a quadprog solver, it requires linear constraints[33]. In order to do so, equation 5.1 is linearized into a set of planar constraints. Instead of enforcing a smooth round cone as a constraint as shown in equation 5.1, the constraint is altered into 4 distinct bounding planes that represent a square pyramid. The four planes are constructed from equation 5.1 by decoupling the y and z axes.

$$|y| \leq x \cdot \tan(\theta_{los}) \tag{5.2}$$

$$|z| \leq x \cdot \tan(\theta_{los}) \tag{5.3}$$

Linearizing the geometric constraints preserves the compatibility with the quadprog matlab function. The theta used in the constraints originally was the max angle used to create the constraint cone, but this fixed constraint crashed the simulations a few times, therefore it was created to be a dynamic cone angle depending on the position of the target vehicle when approaching. The simulation was set-up to calculate the current angle between the position vector of the chaser cubesat and the centerline approach axis (the x-axis). If the calculated angle was greater than the original hard coded max angle constraint, it would switch the constraint theta angle to where the chaser vehicle was, this provided a feasible solution for the quadprog to solve without crashing the simulation. If the chaser vehicle was in the max hard coded theta angle it would continue to use that angle as the constraint. Following the structural logic of the C matrix of a previous reported line of sight constraint implementation, a state space output matrix can be constructed for this project [15]

$$C_{los} = \begin{bmatrix} -\tan(\theta_{los}) & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ -\tan(\theta_{los}) & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ -\tan(\theta_{los}) & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ -\tan(\theta_{los}) & 0 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \tag{5.4}$$

While the original method introduced in previous reports embedded complex, time-varying coordinate functions due to a rotating target vehicle, this project uses a more simplified approach since the target vehicle is cooperative and non-rotating. The only elements in the C matrix represent the line of sight constraints. To enforce the constraints into the quadprog matlab function, it must be restructured to match the implementation in section 3.2 of this report.

$$A \cdot \ x \leq b \tag{5.5}$$

Equation 5.5 is the linear constraint listed in the description of the quadprog matlab function on their website [33]. This constraint must work with the implementation described in section 3.2. Since the state we want to constrain is the A matrix in equation 5.5, we can use the C_{los} matrix in equation 5.4. As for the x term, that was determined to be $\mathbf{S_u}$ in section 3.2 and be used in equation 5.5. The combination of those two terms determine how the control inputs at each time step affect the future boundary violations. For this project the b term in equation 5.5 can be described as the amount of room available before the spacecraft hits a constraint boundary. This term is calculated based on the difference between the hard limit at the edge of the line of sight cone and the location of the chaser vehicle relative to the line of sight boundary.

$$C_{\text{mod}} = C_{los} \cdot \mathbf{S_x} \cdot \mathbf{x} \tag{5.6}$$

$$b = b_{\text{max}} - C_{\text{mod}} \tag{5.7}$$

Equation 5.6 demonstrates creating a matrix that represents where the spacecraft would be relative to the line of sight boundary if provided zero control input over the prediction horizon. The term $\mathbf{S_x}$ is the free response of the chaser vehicle over the prediction horizon, and when multiplied by **x** state vector, it provides the predicted states of the chaser vehicle over the full prediction. The product of those two terms are multiplied by the C_{los} matrix to isolate the position vectors and to map the x position to the tangent of theta used for the line of sight constraint. The b_{max} term is similar C_{mod} but only maps the x axis position. This term is a scalar term that acts as a free-response baseline for max corridor boundary for the inequality constraint in equation 5.7. Subtracting C_{mod} from b_{max} produces the remaining constraint space to be used in equation 5.5.

Implementing this constraint into the solver caused an error in the simulation. The constraint was too rigid. Therefore to ensure numerical stability and solver feasibility, the line of sight constraint was implemented as a soft constraint. Two items were done to enforce the soft constraint. One was adding 0.2 value to the b_{max} this ensures that there will always be a buffer in the constraint space, when b was 0, there were errors in the simulation. The second was the introduction of a slack variable. The variable will be listed at the last column of the product of *A* and x in equation 5.5 is set to -1. For this project that means the last column of the product between C_{los} and $\mathbf{S_u}$. This allows the quadprog solver to break the constraints at high cost if no feasible path exists, thus preventing the simulation from crashing.

5.3 Actuation allocation

The cubesat designed for this report has 12 actuators. The 12 actuators are broken up into 8 unidirectional thrusters and 4 reaction wheels. Each thruster can provide almost instantaneous force while each reaction wheel provides a torque that adds a cumulative moment in the body frame of the cubesat. Section 4.1 of this report covers how the system has been handling maximum force and moment availability. This block only calculated the maximum force and moments available depending on the orientation of the vehicle. Since the thrusters were mounted in fixed positions and are unidirectional they only had a specific maximum thrust in 3 axes of LVLH frame for a given orientation. The thrusters can also provide moments that can be complimentary to the moments provided by the reaction wheels, The reaction wheels have a maximum and minimum torque that can be provided in the body frame based on their datasheet [20]. Since the MPC solver only solves the nominal force and moment output, it falls short of providing how the cubesat achieves the force and moments from its actuators. Aside from the MPC solver, an allocation scheme is required to determine the system inputs for the thrust for each thruster and torque for each reaction wheel to achieve the nominal force and moment command.

The advantage of having an allocation scheme alongside the MPC solver, is that while the MPC focuses on what is needed from the system to achieve its desired location, the allocation scheme focuses on how to allocate actuator resources to meet force and moment demands.

The next approach was to tackle the allocation scheme as a quadratic programming optimization problem. Similar to the current MPC solver, the matlab function quadprog [33] can be set-up to solve for the allocation of actuator inputs for a given nominal force and moments input.

$$\mathbf{g} = [T_1, T_2, T_3, T_4, T_5, T_6, T_7, T_8, \tau_1, \tau_2, \tau_3, \tau_4] \tag{5.8}$$

$$J_{al} = (B_{act}\mathbf{g} - \mathbf{u}_{\text{body}})^T Q_{al}(B_{act}\mathbf{g} - \mathbf{u}_{\text{body}}) + \mathbf{g}^T R_{al}\mathbf{g} \tag{5.9}$$

Equation 5.8 is the actuator array consisting of thrust commands for each thruster and additionally the torque commands for each reaction wheel listed in section 2.3. Equation 5.9 is the objective function used for allocation to minimize error between actuator response and requested, while also minimizing the control effort. The matrix B_{act} maps thruster outputs and torques outputs from the reaction wheels to a 6x1 vector representing the LVLH frame forces and moment. The matrix was derived from equations 2.1-2.6. By subtracting that 6x1 vector from the nominal u_{body} produced from the nonlinear MPC, a state vector can be created to be using their difference. The Q_{al} matrix can then choose which element in that force and moment vector to prioritize in reducing the difference. The R_{al} matrix is then left with the actuator state vector to penalize usage

$$\min_{\mathbf{g}} \quad \frac{1}{2}\mathbf{g}^T H_{alloc}\mathbf{g} + \mathbf{f}_{alloc}^T\mathbf{g} \tag{5.10}$$

$$J_{al} = \mathbf{g}^T B_{act}^T Q_{al} B_{act}\mathbf{g} - 2(B_{act}^T Q_{al} u_{body})^T\mathbf{g} + u_{body}^T Q_{al} u_{body} + \mathbf{g}^T R_{al}\mathbf{g} \tag{5.11}$$

$$H_{alloc} = 2(B_{act}^T Q_{al} B_{act} + R_{al}) \tag{5.12}$$

$$\mathbf{f}_{alloc} = -2(B_{act}^T Q_{al}\mathbf{u}_{\mathrm{body}}) \tag{5.13}$$

The objective function shown in equation 5.9 must be placed in a quadratic programming problem as shown in equation 5.10 [33]. This is the same equation used in equation 3.6 for solving the nonlinear MPC cost function. Equation 5.9 was expanded into 5.11 to isolate the terms to equate to equation 5.10 [34]. This expansion allows for equations 5.10 to 5.11 to be equated and find equations for H and f. The result is equations 5.12 and 5.13, these are the inputs to the quadratic programing solver function MATLAB provides [33].

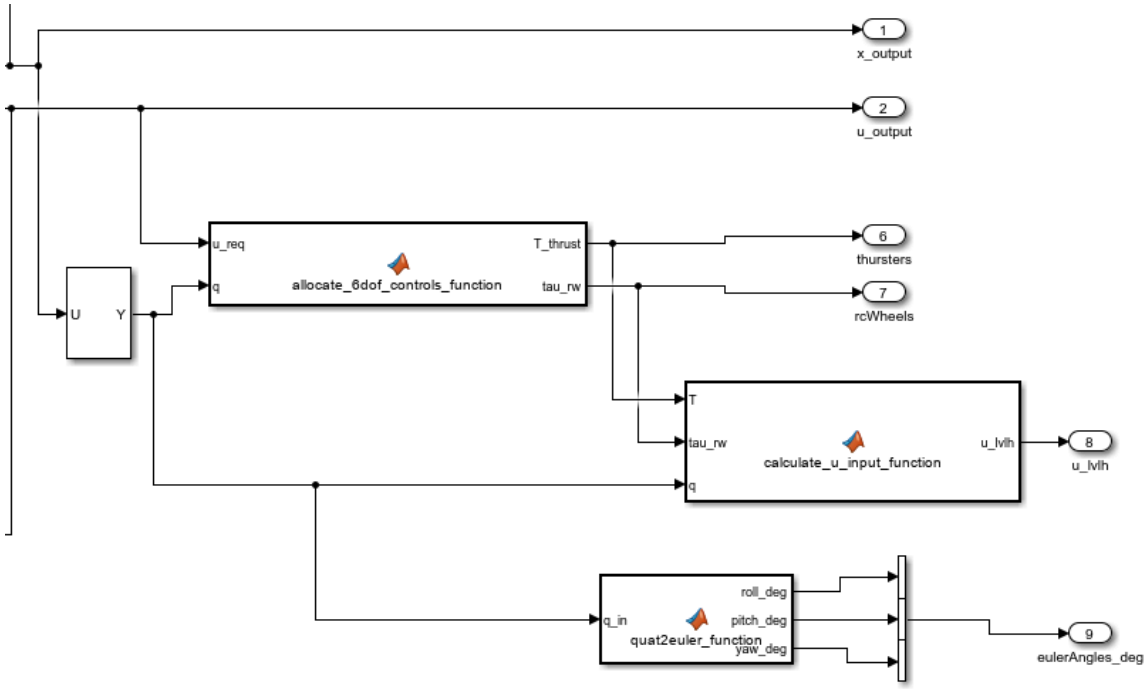


Figure 5.4: Allocation block in simulation

Figure 5.4 shows the simulation set-up for debugging when implementing the allocation scheme. Before including its effect into simulation, a comparison was made to view the impact of allocation on the force and moment output. The simulation calculates the allocation commands for every control input provided by the MPC and then uses the actuator inputs to output the force and moment produced. Those outputs are saved but not feedback into the simulation for debugging purpose

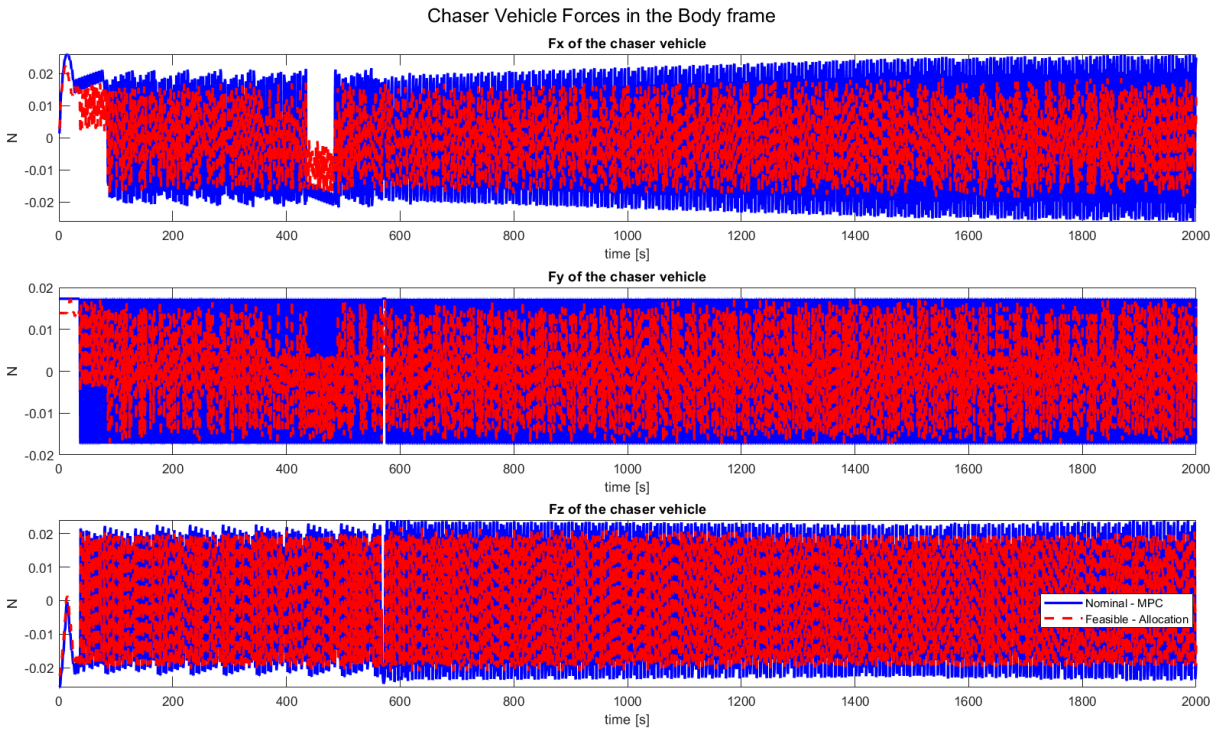


Figure 5.5: Force comparison from allocation scheme

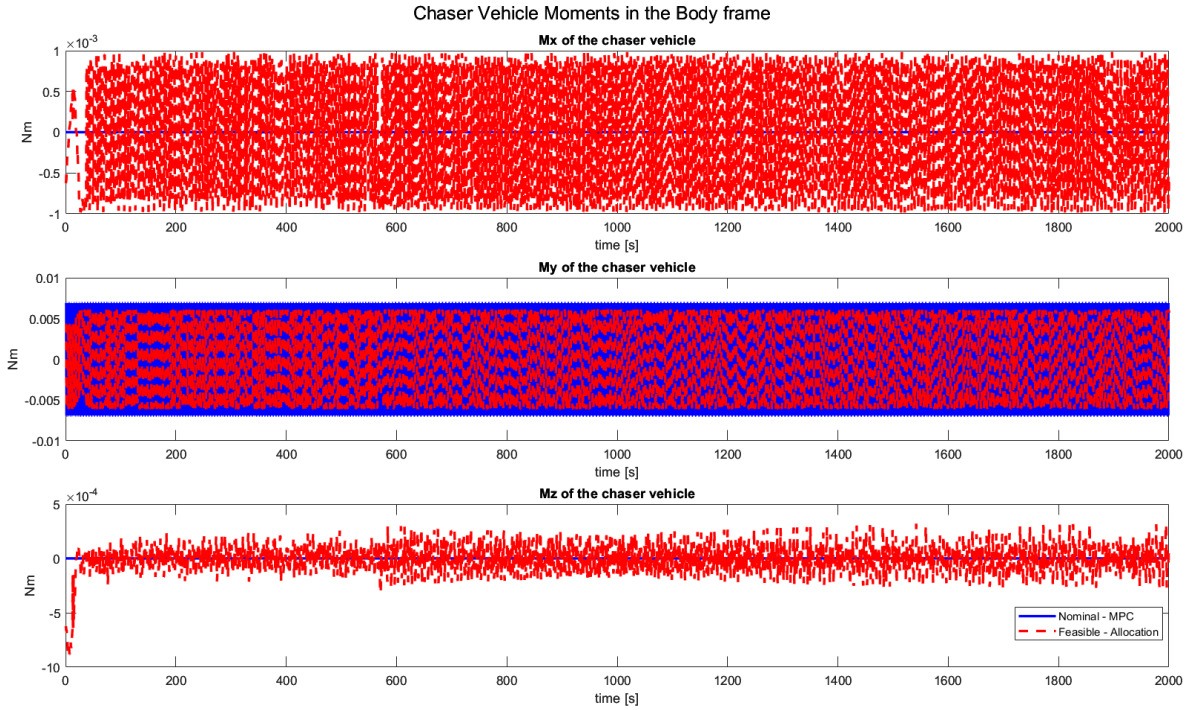


Figure 5.6: Moment comparison from allocation scheme

Figure 5.5 and 5.6 display the comparison of the force requested from the MPC and the force produced by the allocation scheme. Although the MPC is not tuned and is heavily aggressive in reaching its target and depending nearly on maximum forces and moments, the allocation is able to nearly keep up. The only areas of concern would be the Mx moment but this will be addressed in section 6 after gain tuning has occurred.

5.4 Current results from increased model fidelity

As model fidelity has increased, it is best to check on how the simulation runs for all phases of the mission. The first simulation starts the target position of 500 meters away in the R-bar axis (x axis in the LVLH frame), 4 km away in the V-bar axis (y-axis in the LVLH frame), and 500 meters away in the cross track axis (z axis in the LVLH frame). Based on Table 4.1 this will start the simulation in phase 3.

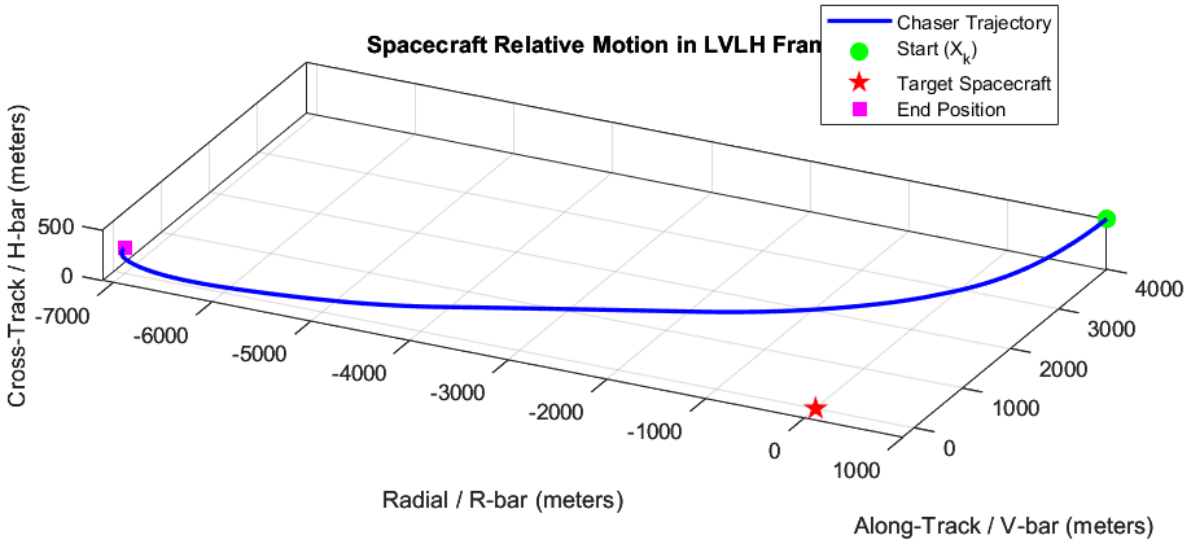


Figure 5.7: Trajectory when starting in phase 3

From figure 5.7 it is clear the simulation diverges and does not reach the target when starting in a phase 3 location.

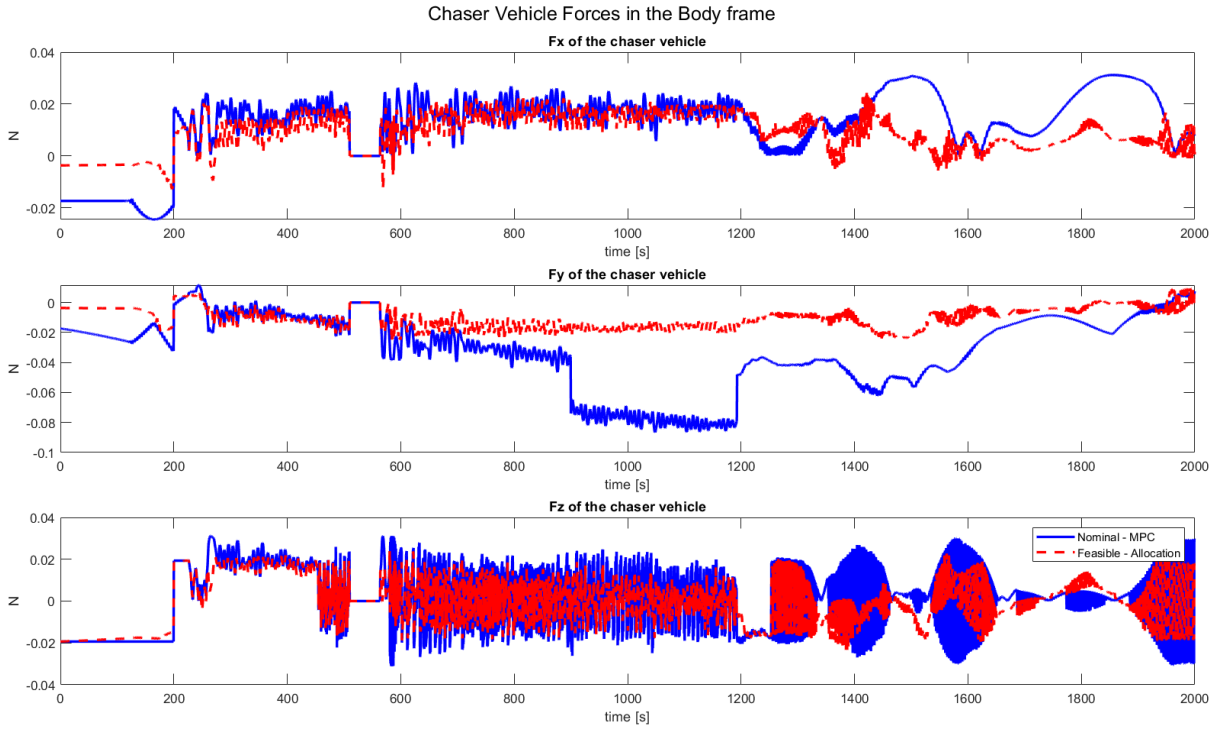


Figure 5.8: Forces from the homing phase

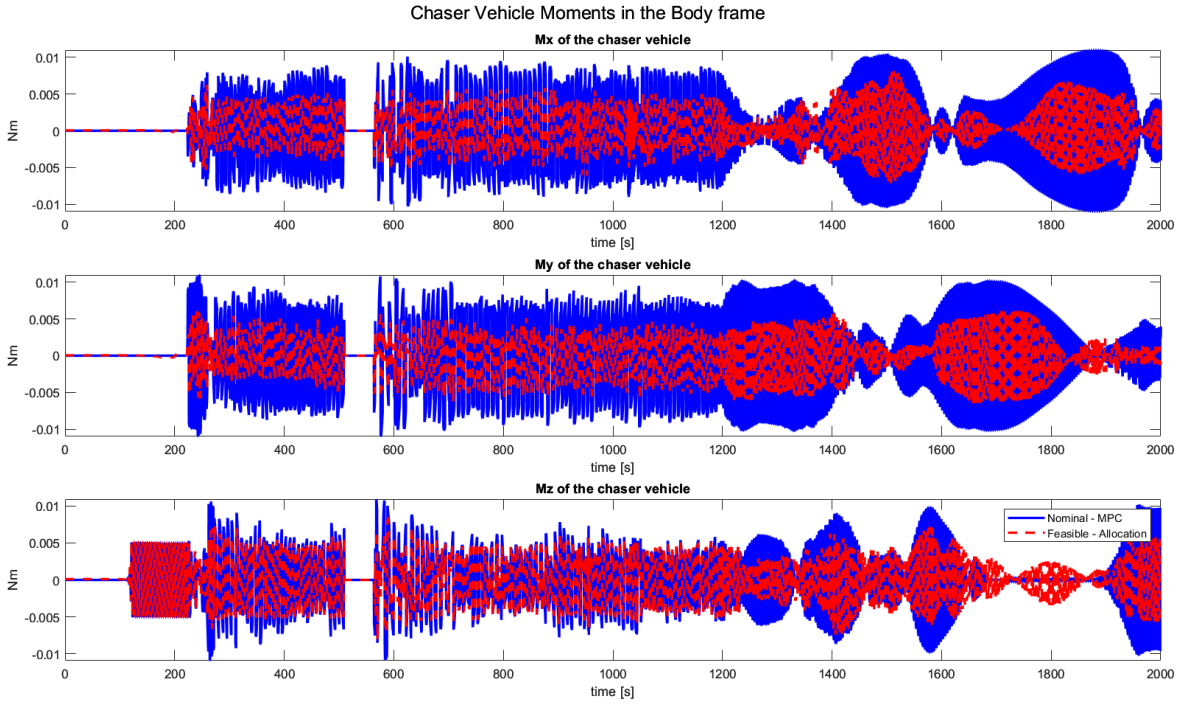


Figure 5.9: Moments from the homing phase

Figure 5.8 and 5.9 display the force and moments on the cubesat when trying to reach the target from a phase 3 location. It is clear that the NMPC is requesting forces that the thrusters can not produce. From figure 5.10 we see that the y position and x position has huge errors as they go past the 0 point on the graph, this suggests a higher penalty on position and possibly the S gain.

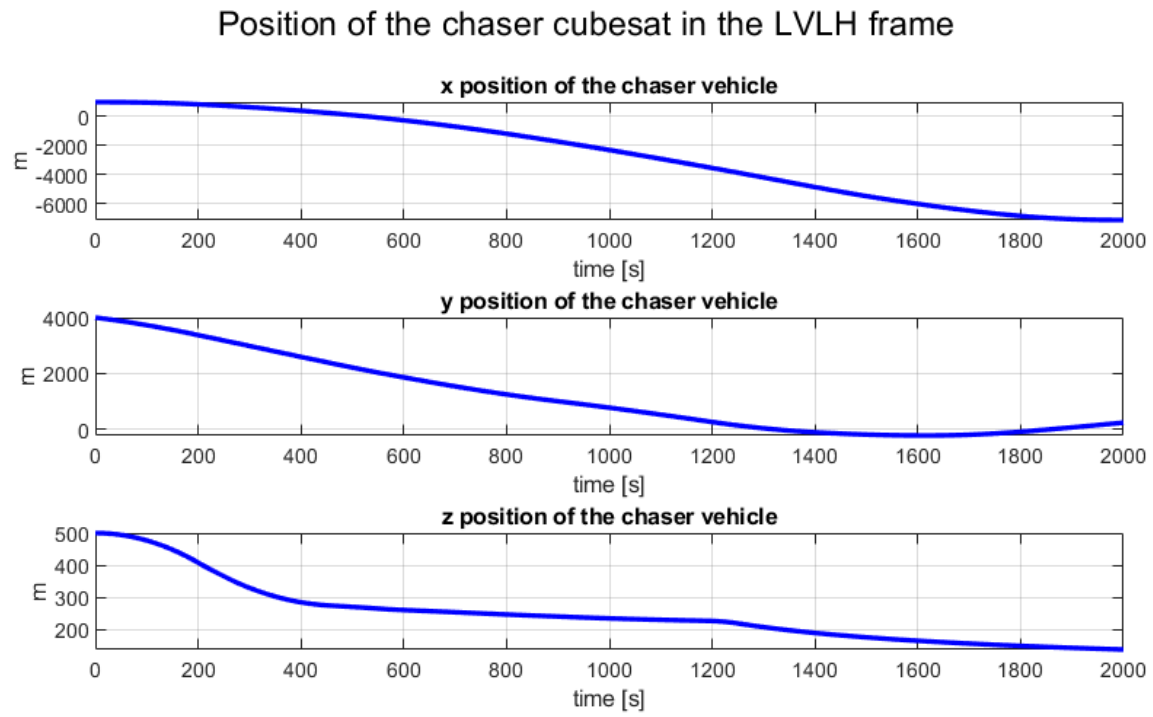


Figure 5.10: Position of chaser vehicle

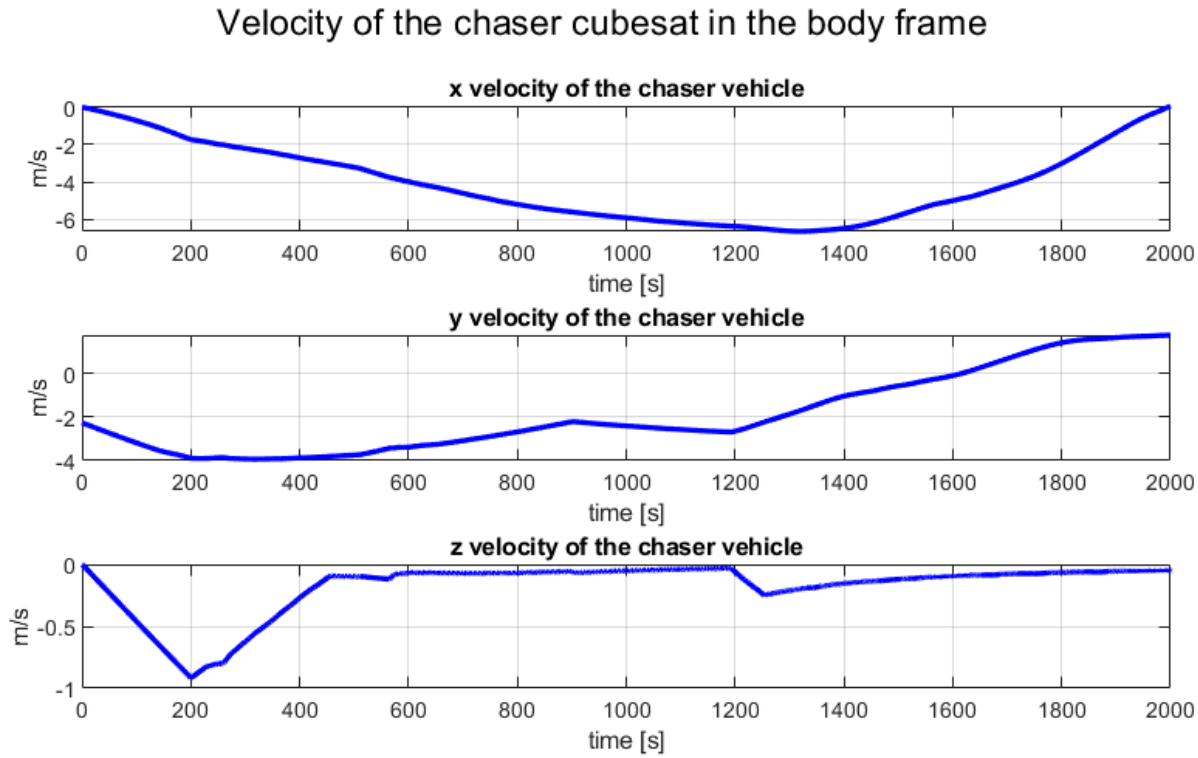


Figure 5.11: Velocity of chaser vehicle

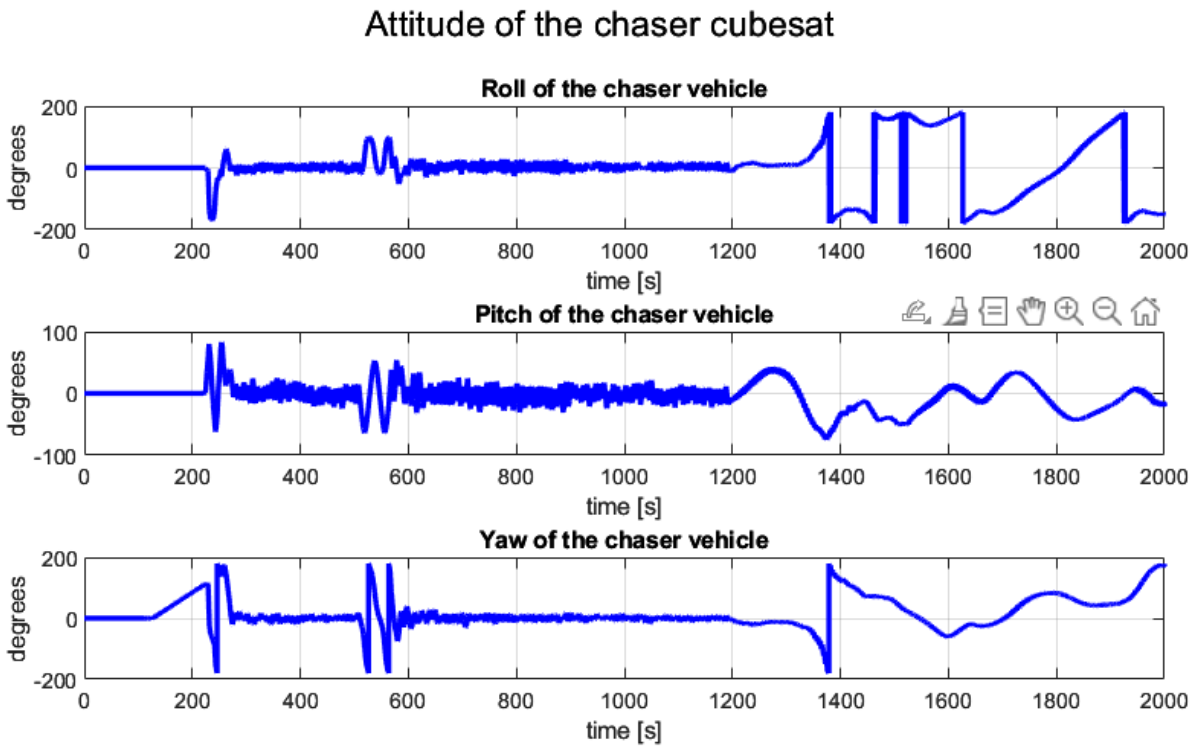


Figure 5.12: Attitude of the chaser vehicle

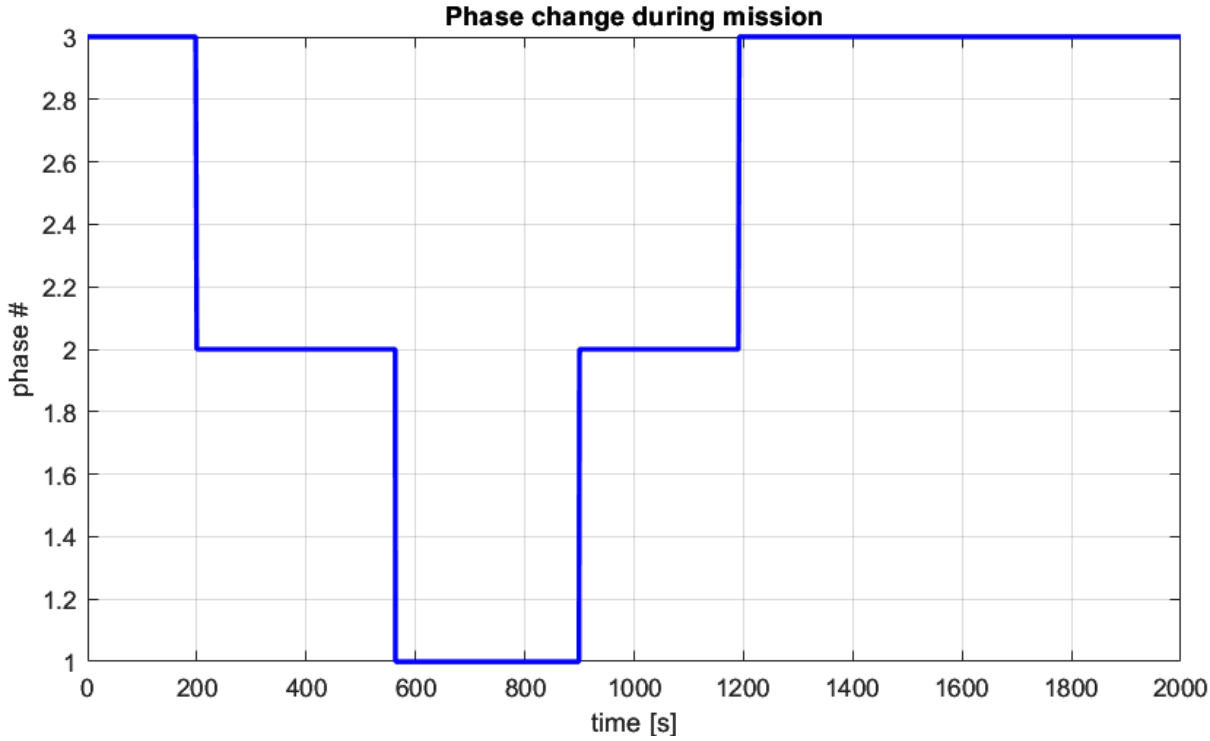


Figure 5.13: Phase change during simulation

Figure 5.11 shows a massive increase in x speed, the gains will have to be tuned to avoid high errors from the max velocity. Also might suggest the reference trajectory might need to incorporate a speed brake to avoid high velocities when reaching a target hold point as the gains will change when entering one phase and exiting another. Figure 5.13 displays phase changes, so at one point the chaser does get close enough to enter final approach but then swings out and flies away, this definitely suggests the chaser cubesat is coming in too fast.

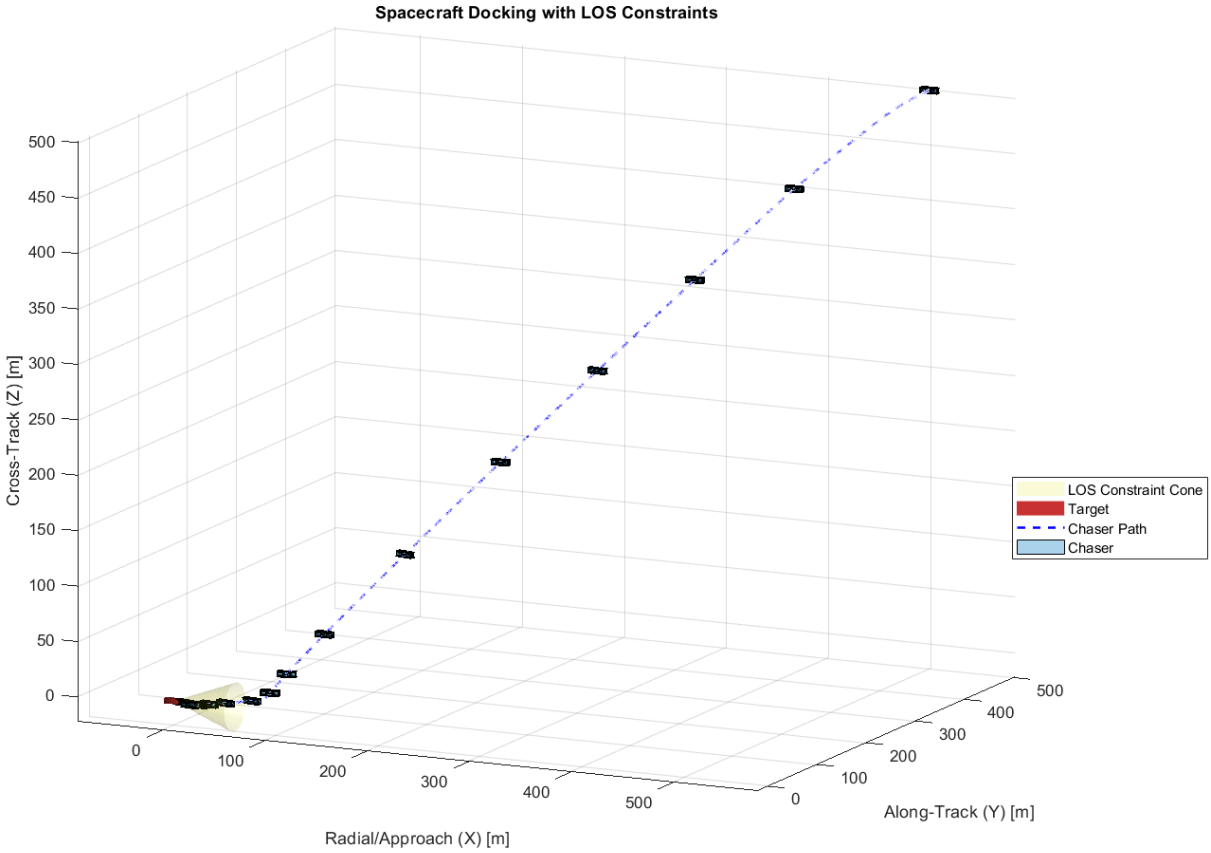


Figure 5.14: Trajectory from second simulation run starting phase 1 with line of sight constraint

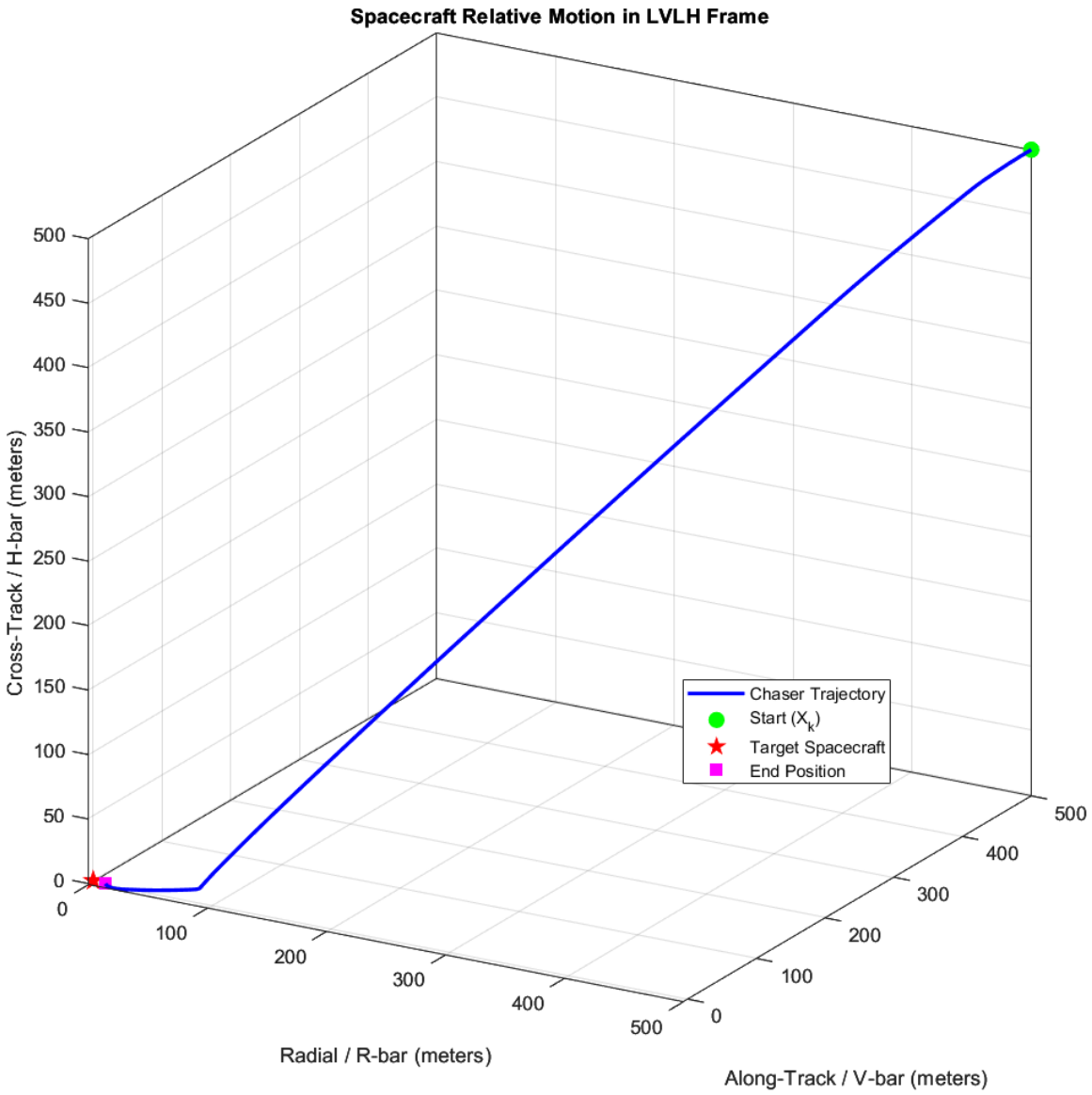


Figure 5.15: Line trajectory starting in phase 1

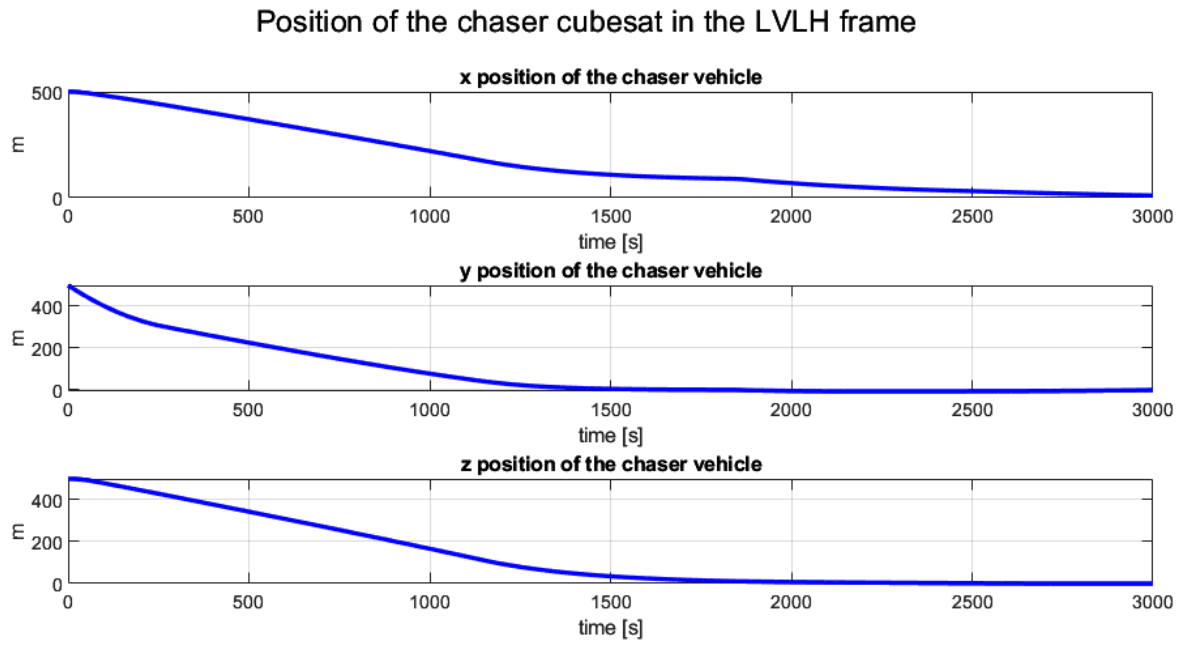


Figure 5.16: Chaser position starting from phase 1

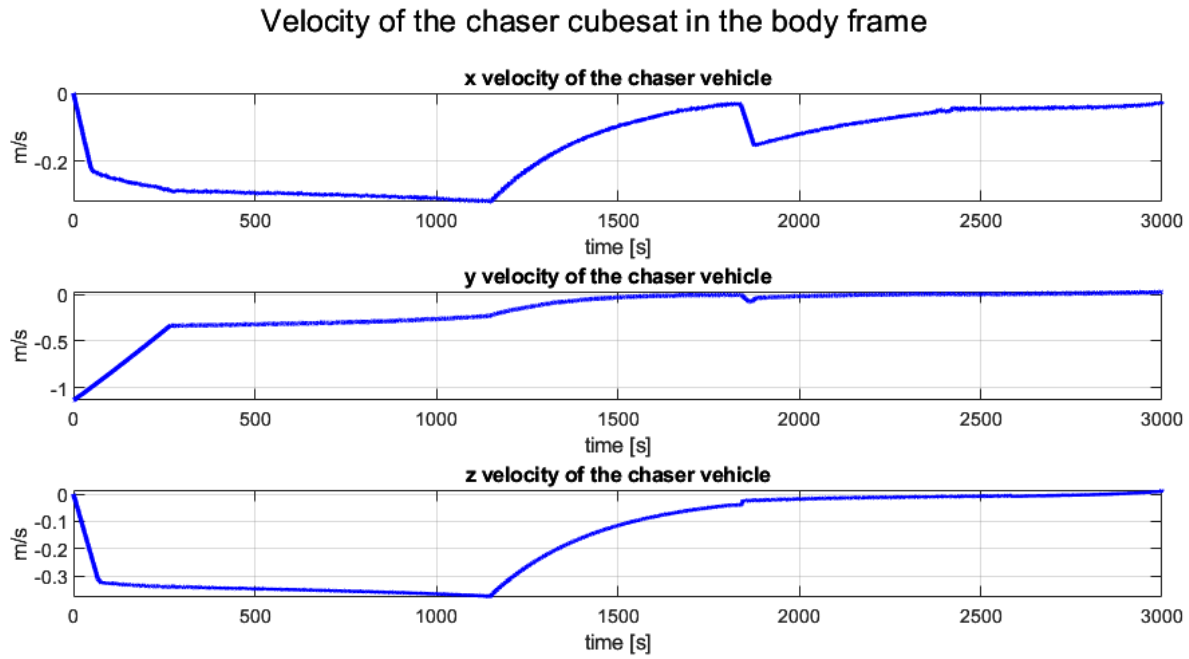


Figure 5.17: Chaser velocity starting from phase 1

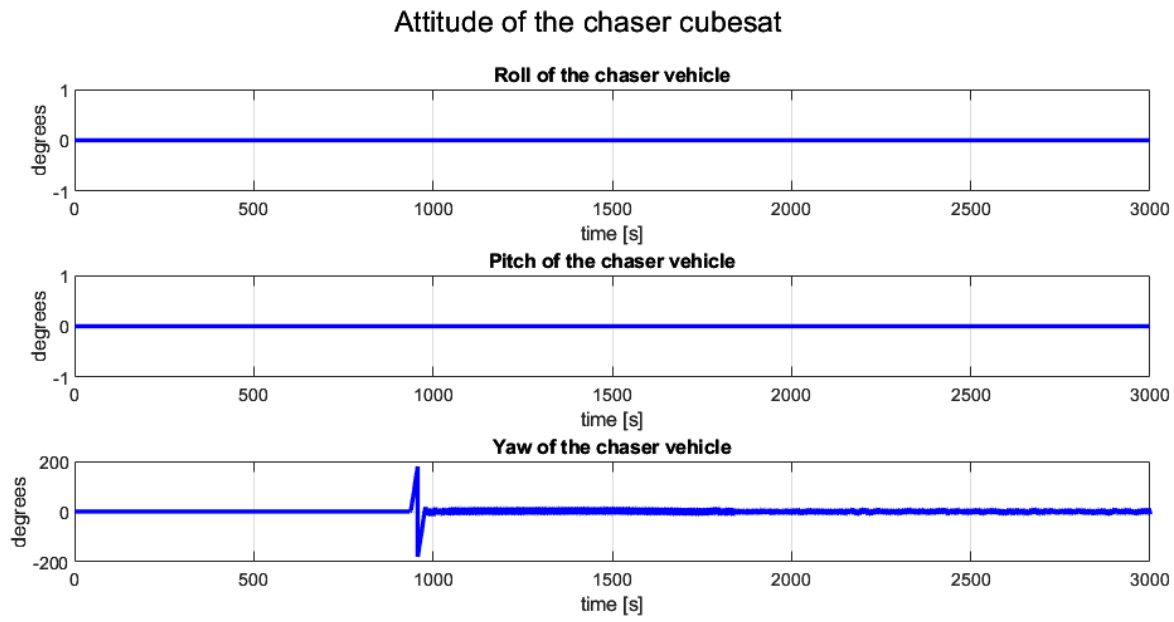


Figure 5.18: Chaser force input starting from phase 1

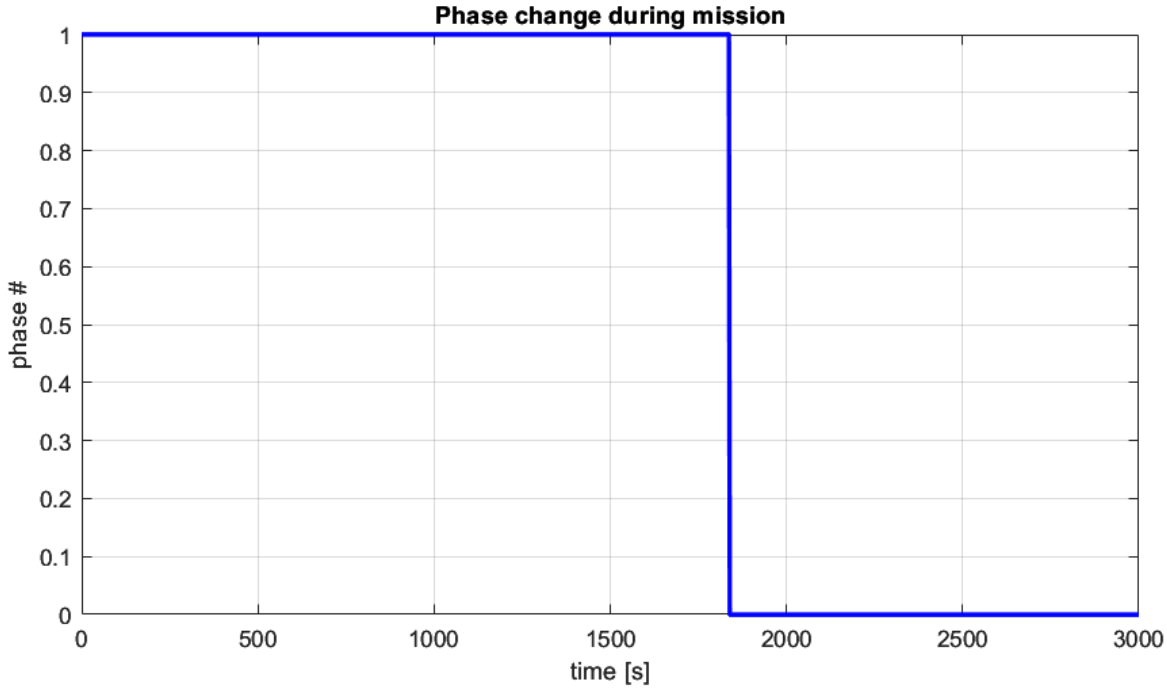


Figure 5.19: Phase change during simulation

Figures 5.14 to 5.19 display the results for nonlinear MPC when starting in phase 1. This simulation starts the chaser at 500 meters away in all three axes. From figure 5.14 and figure 5.15, the nonlinear MPC is successful in reaching the target cubesat and bonus it stays within the

line of sight cone implemented in this section. The reaction of the cubesat is very clean with oscillations in attitude or velocities. A similar test was conducted for a simulation start in phase 2, but similar results were concluded from the simulation in phase 3.

6. Nonlinear MPC results and discussion

6.1 Improvements to simulation

The results shown in chapter 5 demonstrated that the controller worked for missions between final approach phase and the docking phase which are listed in table 4.1. Though gain tuning is required to achieve more realistic demands of the thrusters and reaction wheel, the current controller gains still achieve the objective between those two phases. The issues were present between the homing phase and the closing phase. Based on the discussion in chapter 5, the reference trajectory script was altered to switch between the straight line approach to a reference trajectory that considers the curvature of the chaser cubesat.

$$\mathbf{x}_{\text{ref}}(1:3) = \mathbf{r}_{\text{curr}} + \mathbf{v}_{\text{cmd}}\Delta t \tag{6.1}$$

Equation 6.1 demonstrates the straight line approximation of the state estimating the position using the current velocity command. This approach has worked for the final approach phase and the docking phase but has failed the phases when the chaser vehicle is far away. Since the MPC solver is using linearized A and B matrices at every point, the reference trajectory can use linearized equations to create a reference state that will be used by the MPC solver. Creating a reference trajectory more along the lines of an orbital path rather than a straight line path.

$$\ddot{x} - 3n^2x - 2n\dot{y} = 0 \tag{6.2}$$

$$\ddot{y} + 2n\dot{x} = 0 \tag{6.3}$$

$$\ddot{z} + n^2z = 0 \tag{6.4}$$

The above equations are the clohessy-wiltshire equations [35]. The matrices are defined as Φ_{rr}, Φ_{rv}, Φ_{vr}, and Φ_{vv}, they correspond to coefficients used in defining the relative position and velocity between a chaser and target using the clohessy-wiltshire equations [35].

$$x_{\text{ref}} = x_{\text{curr}}\cos(n\Delta t) + \frac{\dot{x}_{\text{cmd}}}{n}\sin(n\Delta t) + \frac{2\dot{y}_{\text{cmd}}}{n}(1 - \cos(n\Delta t)) \tag{6.5}$$

$$y_{\text{ref}} = y_{\text{curr}} - 3x_{\text{curr}} n\Delta t + \frac{2\dot{x}_{\text{cmd}}}{n}(\cos(n\Delta t) - 1) + \frac{4\dot{y}_{\text{cmd}}}{n}\sin(n\Delta t) \tag{6.6}$$

$$z_{\text{ref}} = z_{\text{curr}}\cos(n\Delta t) + \frac{\dot{z}_{\text{cmd}}}{n}\sin(n\Delta t) \tag{6.6}$$

The equations above demonstrate how the state reference position is calculated when the chaser is in the closing phase and the homing phase. Equation 6.1 is still used during the final approach and docking phase, where equations 6.5-6.7 are used for phase 2 and 3. The use of these functions can be seen in the appendix-C.

Another change to the simulation made was to the calculation of the jacobains. Originally an ϵ of 1e-6 was used to create the perturbation needed to calculate the discretized A and B matrices. This proved to not work for longer range missions. The ϵ was increased to 1e-9 which yielded better results as phase 2 eventually converged when originally it drifted away from the target similarly figure 5.7 with starting the simulation from phase 3.

Another implementation occurred on the reference trajectory function. As noted in section 5, the velocity of the cubesat was too high, so a braking parameter was added to the velocity state reference coming from the reference trajectory function. At each phase there is a maximum velocity command detail in table 5.1, this velocity maximum must be altered when getting close to the phase target hold spot.

$$\mathbf{v}_{cmd} = V_{max} \times \left(\frac{\text{distance}}{\text{braking_distance}}\right) \tag{6.7}$$

Equation 6.7 is the equation used to reduce the maximum velocity coming out of the reference trajectory function block shown in figure 3.1. The distance in the equation is the norm of the chaser's state position minus the phase's target hold point. Only when the norm of that state position is less than the calculated distance in equation 6.7 will the velocity reference command be reduced. Each phase will have its own braking distance as some phases have higher velocities to deal with and need longer distances to slow down.

Table 6.1: Phase breaking distance

Mission parameter	Breaking distance
Phase 3	600 meters
Phase 2	300 meters
Phase 1	150 meters
Phase 0	50 meters

6.2 Gain tuning

Since the NMPC has many parameters to tune, manual tuning will be attempted with keeping certain variables fixed. Future projects may introduce some optimization scheme to optimize all parameter gains but that is outside the scope of this project. The table below will showcase the parameters used before manual gain tuning, along with new updates from the previous chapter. The prediction horizon and prediction sample time will remain fixed as they have been shown to be sufficient in previous chapters.

Table 6.2: Fixed parameters

Mission parameter	Value
Phase 3 target position	[500;2000;100] meters
Phase 2 target position	[200;500;100] meters
Phase 1 target position	[80;2.0;0.5] meters
Phase 0 target position	[1;0;0] meters
Prediction horizon (N)	20
Prediction sample time	0.2

Table 6.2 displays the values that will remain constant throughout the gain tuning process.

Table 6.3: Simulation parameters prior to tuning

Mission parameter	Value
Phase 3 Q gains	[20;20;20;1000;1000;1000;1;1;1;1;1;100; 100;100]
Phase 3 R gains	[0.01;0.01;0.01;5;10;5]
Phase 3 S penalty	1000
Phase 2 Q gains	[10;20;10;50;10;10;1;1;1;1;1;10;10;10]
Phase 2 R gains	[0.1;0.1;0.1;5;10;5]
Phase 2 S penalty	500
Phase 1 Q gains	[20;20;20;10;10;10;1;1;1;1;1;10;10;10]
Phase 1 R gains	[10, 10, 10, 5, 10, 5];
Phase 1 S penalty	500
Phase 0 Q gains	[20;20;20;10;10;10;50;50;50;50;50;10;10;10]
Phase 0 R gains	[1, 1, 1, 0.1, 0.1, 0.1];
Phase 0 S penalty	500

Table 6.3 demonstrates the gains used for phases in section 5. Since phase 1 and 0 worked extremely well, gain tuning will occur on phases 2 and 3 for this section. The objective is to run a simulation starting in phase 3 and reaching the docking port of the target cubesat.

Table 6.4: Simulation parameters after manual tuning

Mission parameter	Value
Phase 3 Q gains	[100;200;50;2000;1000;10;1;1;1;1;1;10; 10;10]
Phase 3 R gains	[0.4;0.3;10;5;10;5]
Phase 3 S penalty	1000
Phase 2 Q gains	[200;199;50;20879;1000;10;1;1;1;1;1;10;10;10]
Phase 2 R gains	[0.3932;0.2919;10;5;10;5]
Phase 2 S penalty	100
Target position phase 3	[1000;4000;500]
Target position phase 2	[500;2500;500]

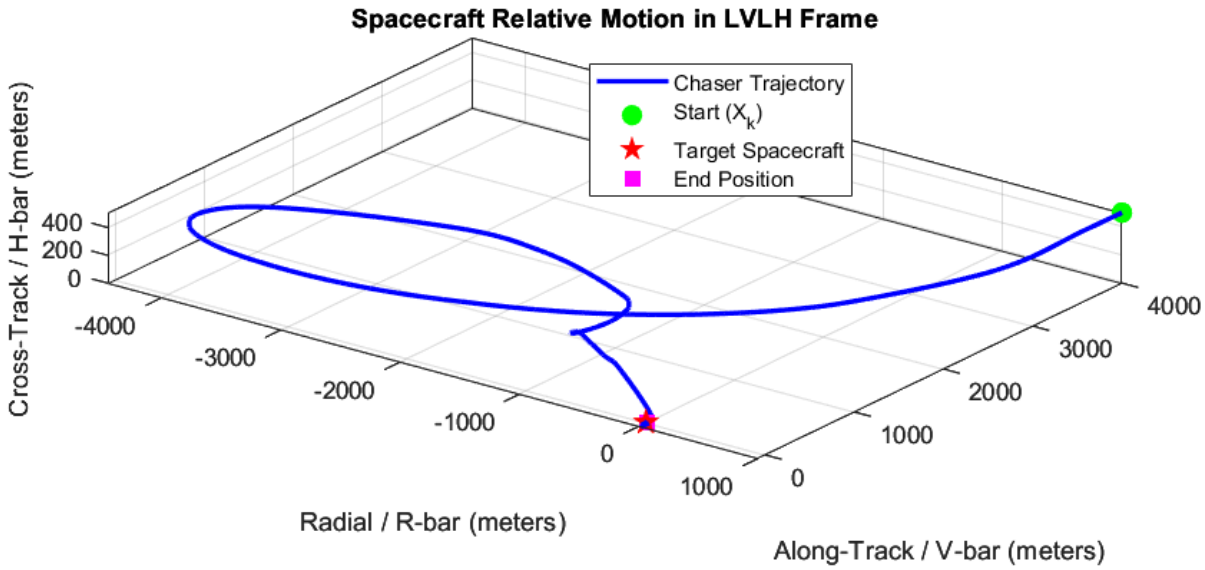


Figure 6.1: Results when starting in phase 3 after gain tuning

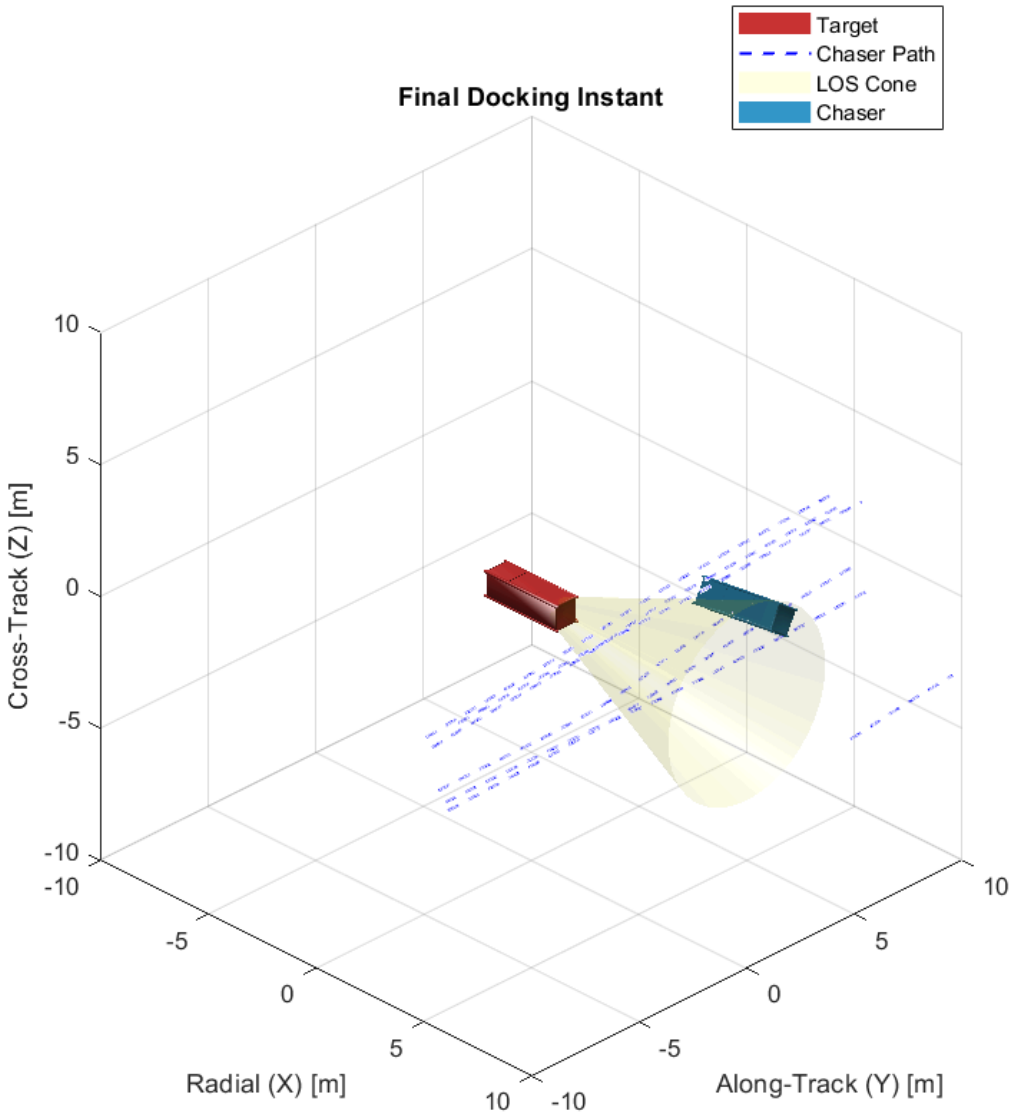


Figure 6.2: Docking results from phase 3

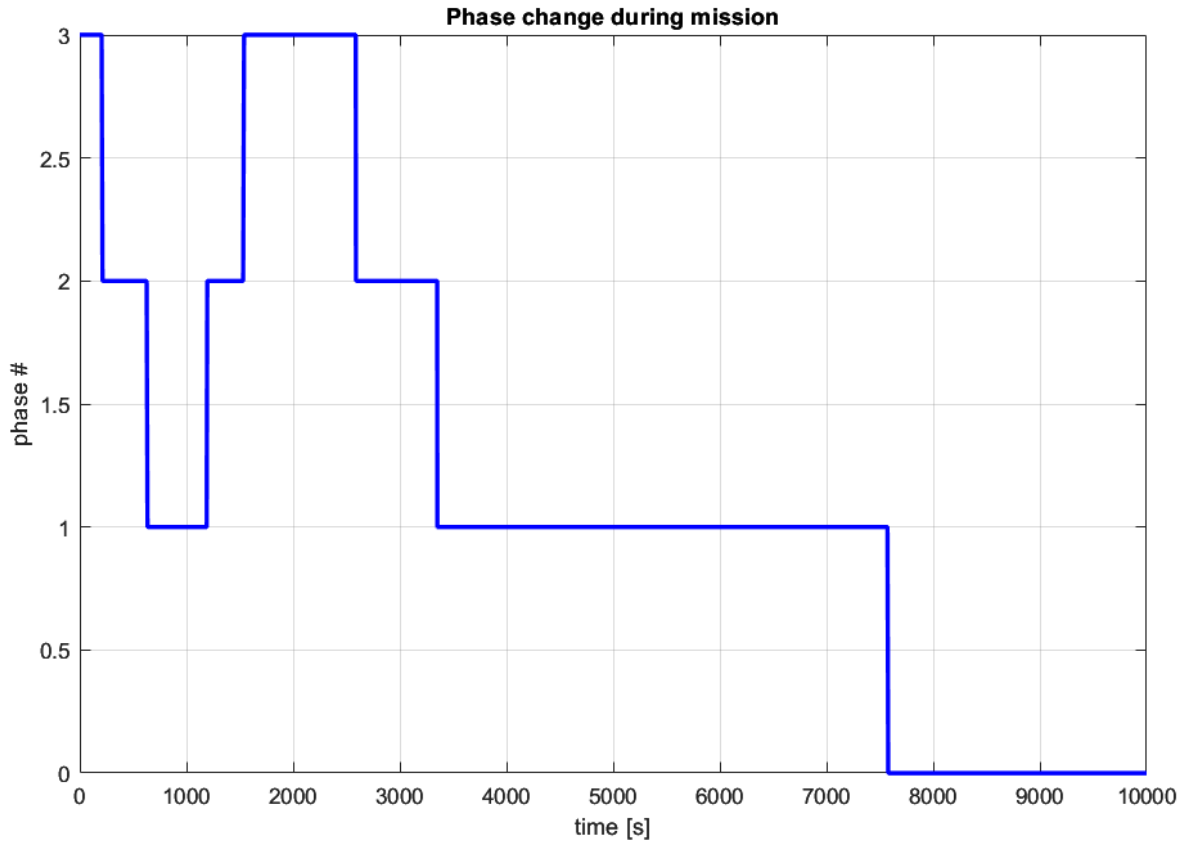


Figure 6.3: Phase transition during phase 3 simulation

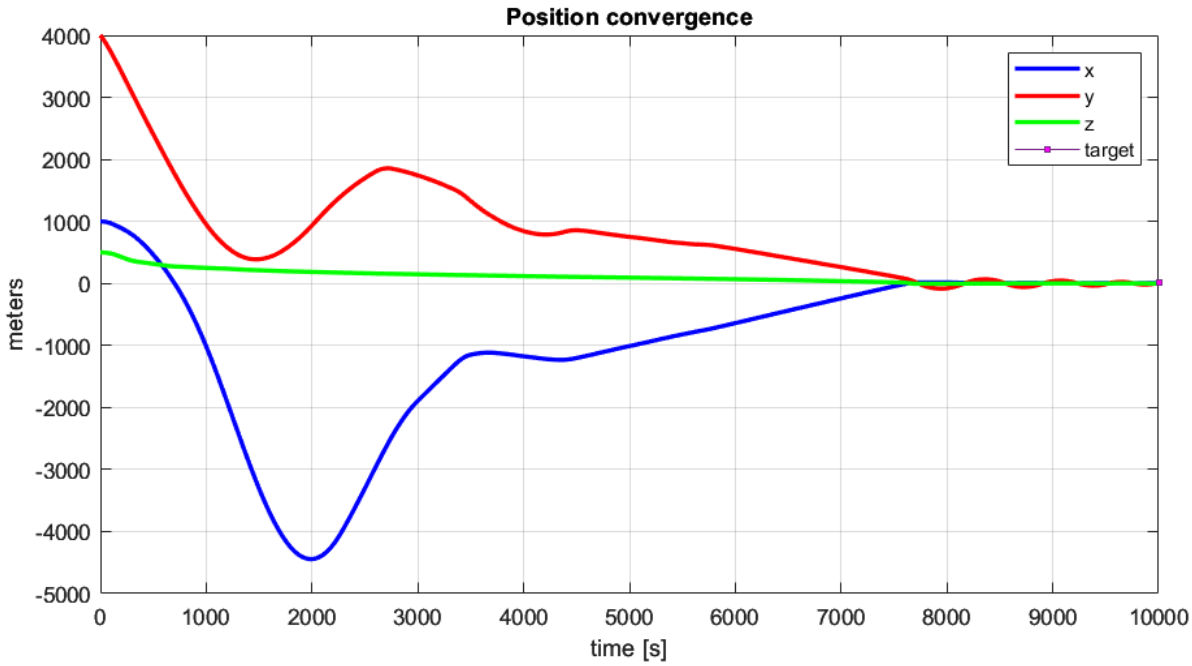


Figure 6.4: Phase 3 position convergence to docking position

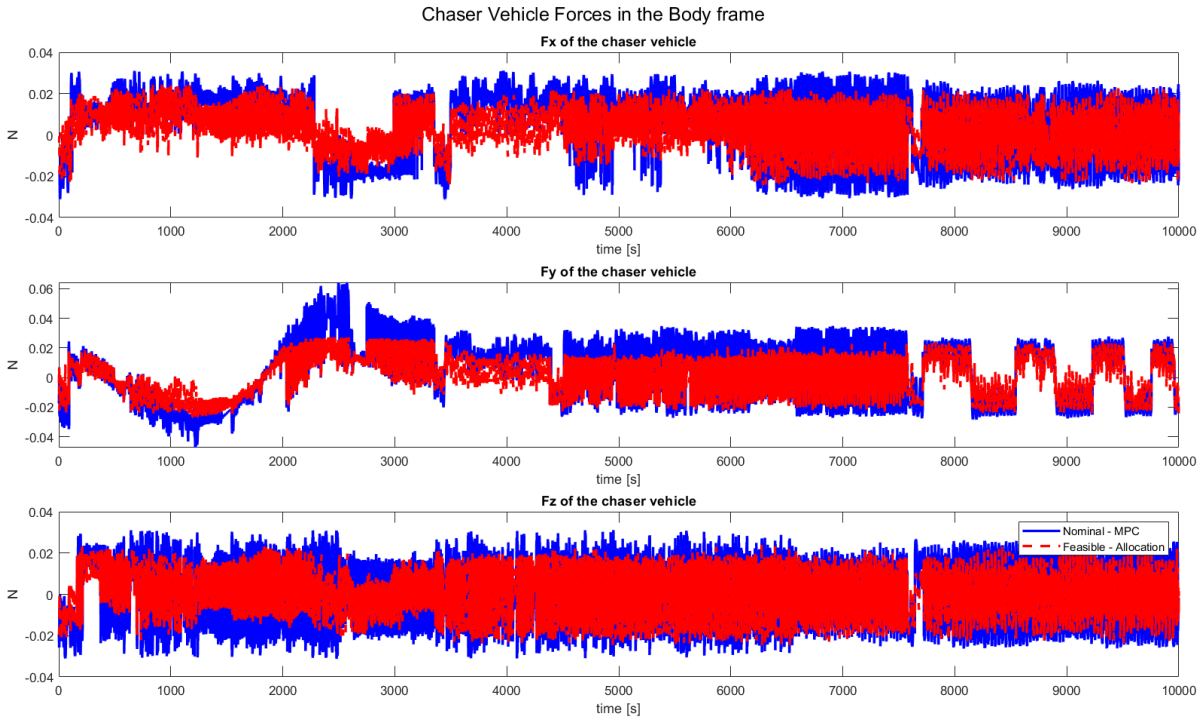


Figure 6.5: Forces from phase 3 mission

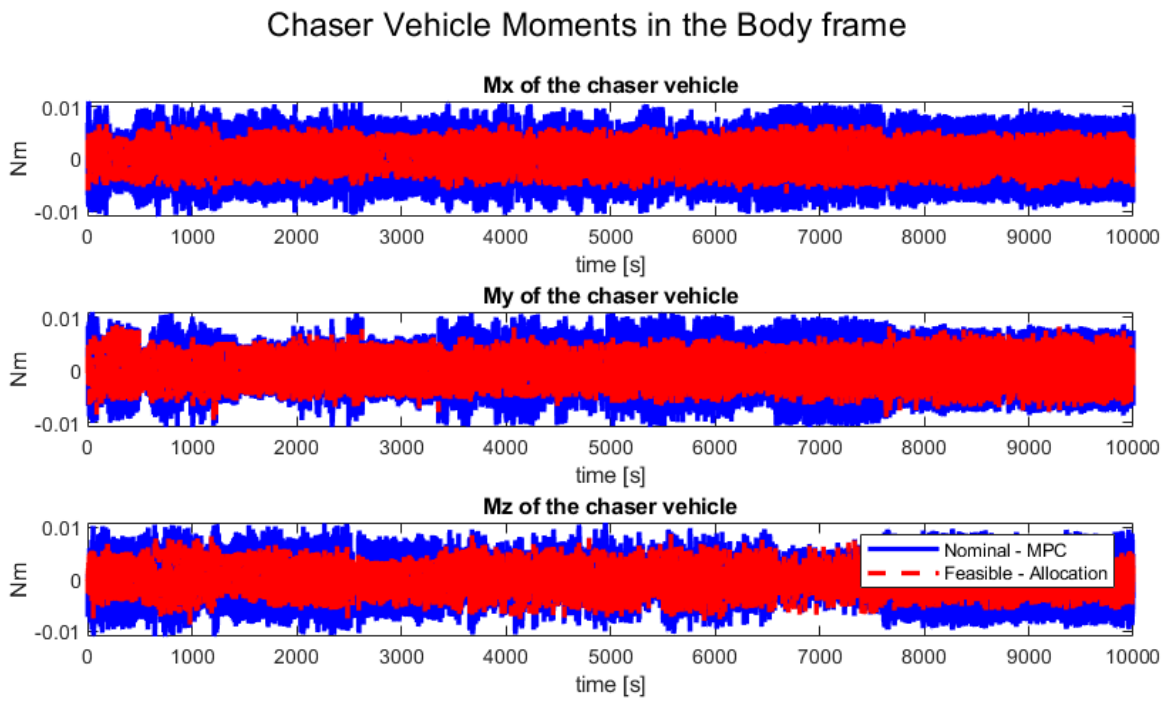


Figure 6.6: Moments from phase 3 mission

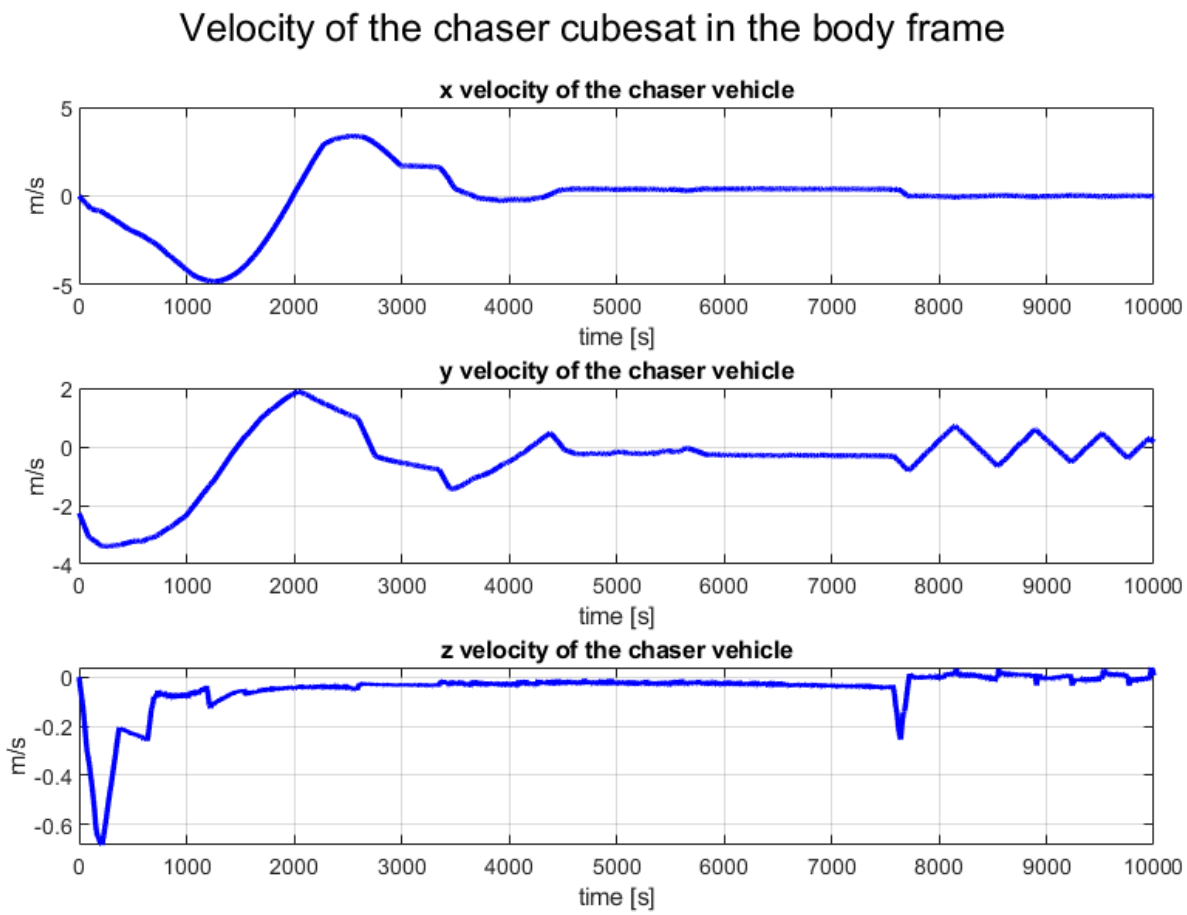


Figure 6.7: Velocity from phase 3 mission

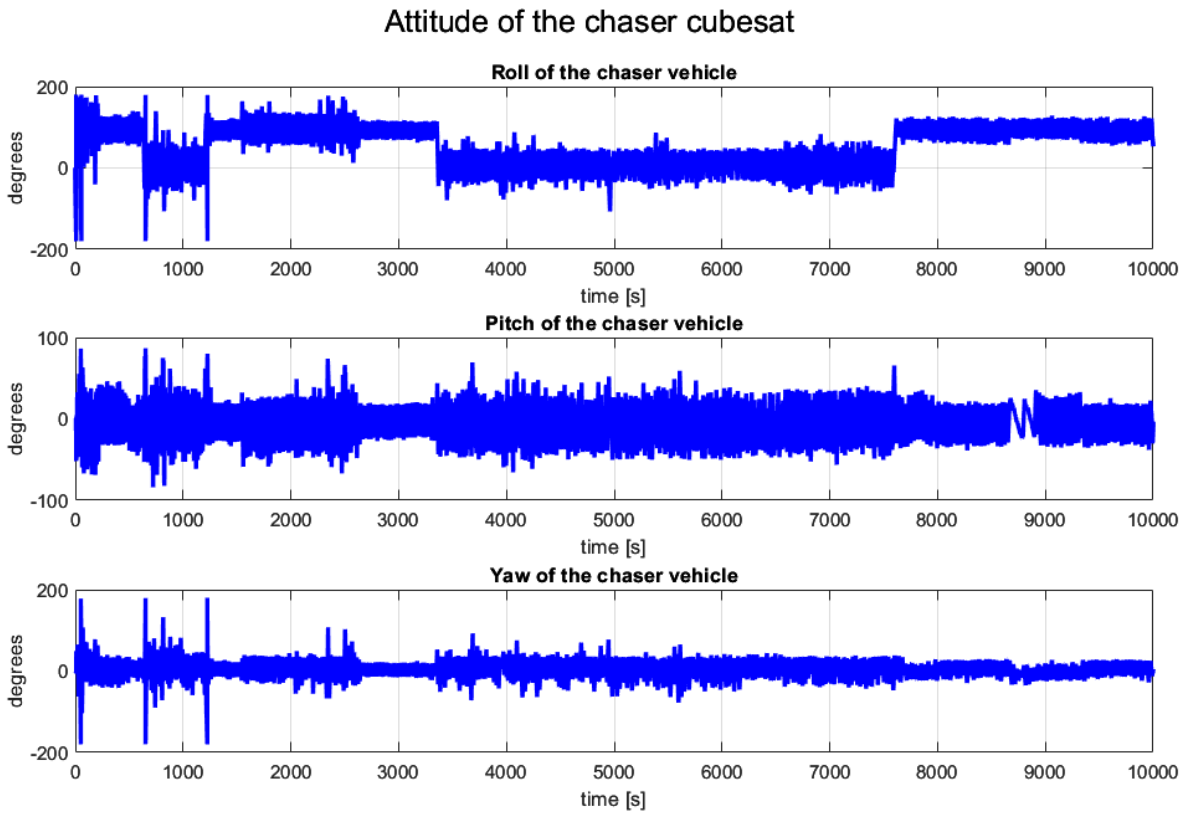


Figure 6.8: Chaser attitude from phase 3 mission

Figure 6.1-6.8 demonstrates the performance of the NMPC when starting 4000 meters away and is in phase 3. Even though the trajectory took a drastic turn away from the target, it recovered and was able to locate the target cubesat. The docking needed work as it was very oscillating around the line of sight. The y velocity seems to oscillate as well when close to docking, it would be recommended to increase the y-velocity gain in the Q-matrix if not for figure 6.8. Figure 6.8 shows the attitude of the chaser vehicle which seems to spike in some attitudes and never really settles. This suggests that perhaps an increase in the angular velocity or attitude states in the Q matrix is required in both the docking phase and phase 3. Giant tuning for phase 3 is difficult since running simulations takes time when the chaser cubesat is far away, the current simulation to go from phase 3 to dock is roughly ten thousand seconds.

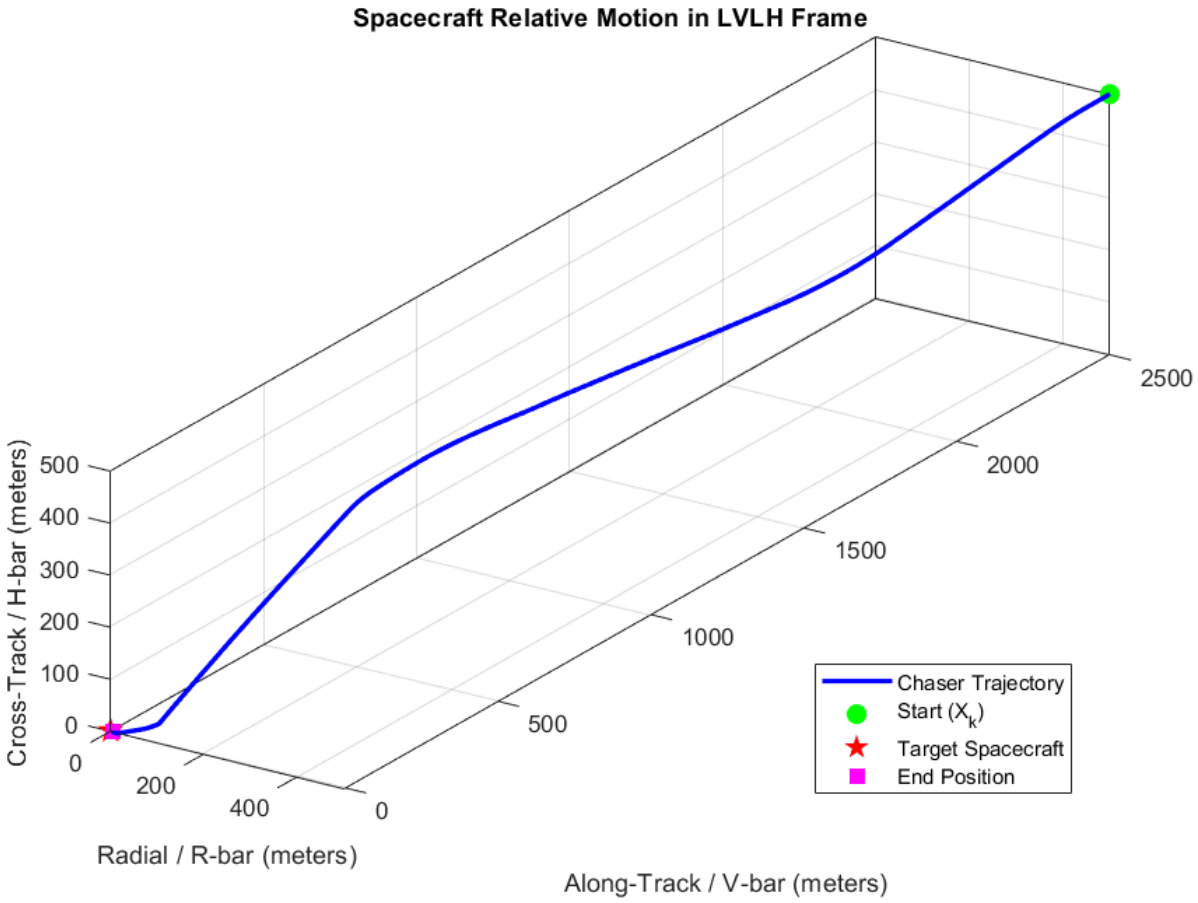


Figure 6.9: Results when starting in phase 2 after gain tuning

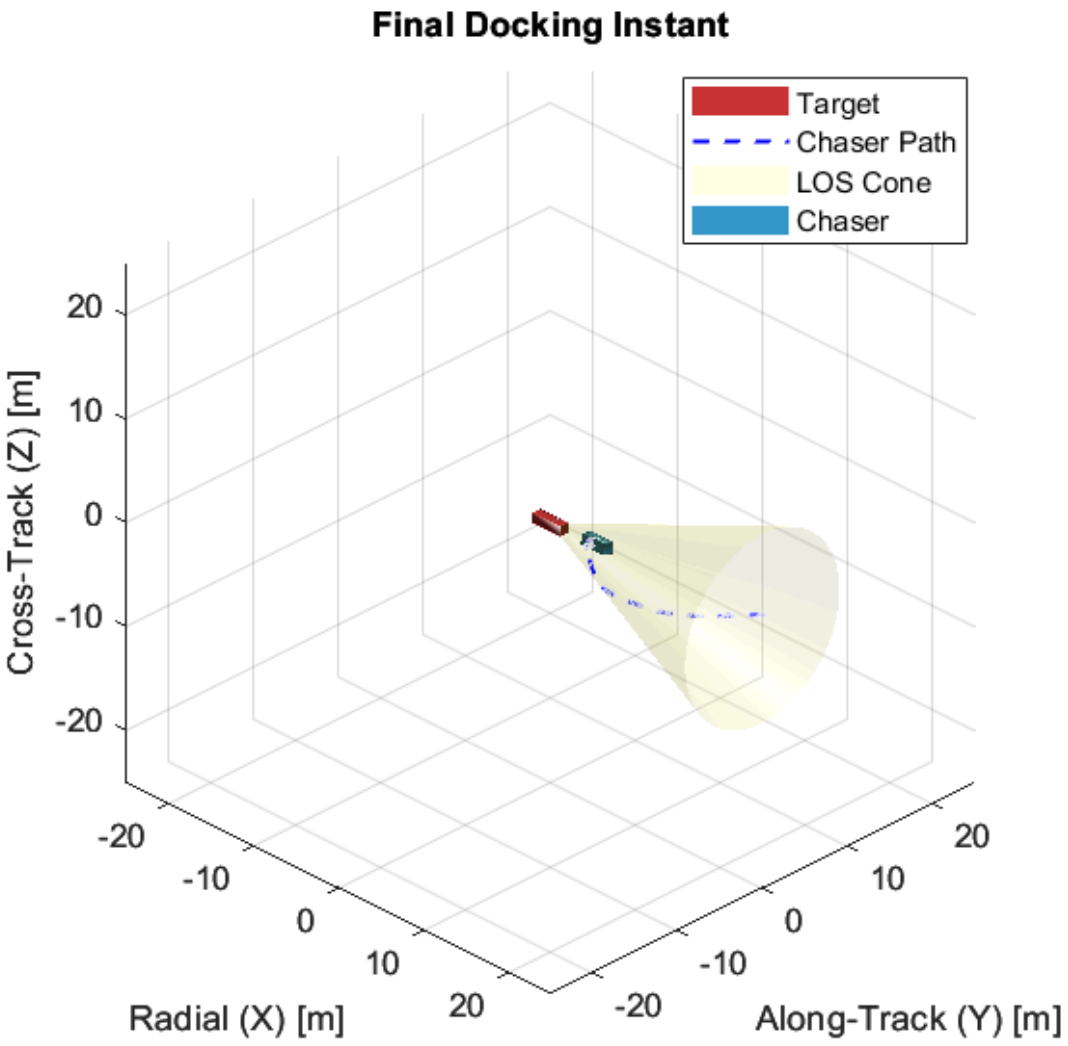


Figure 6.10: Docking results from phase 2

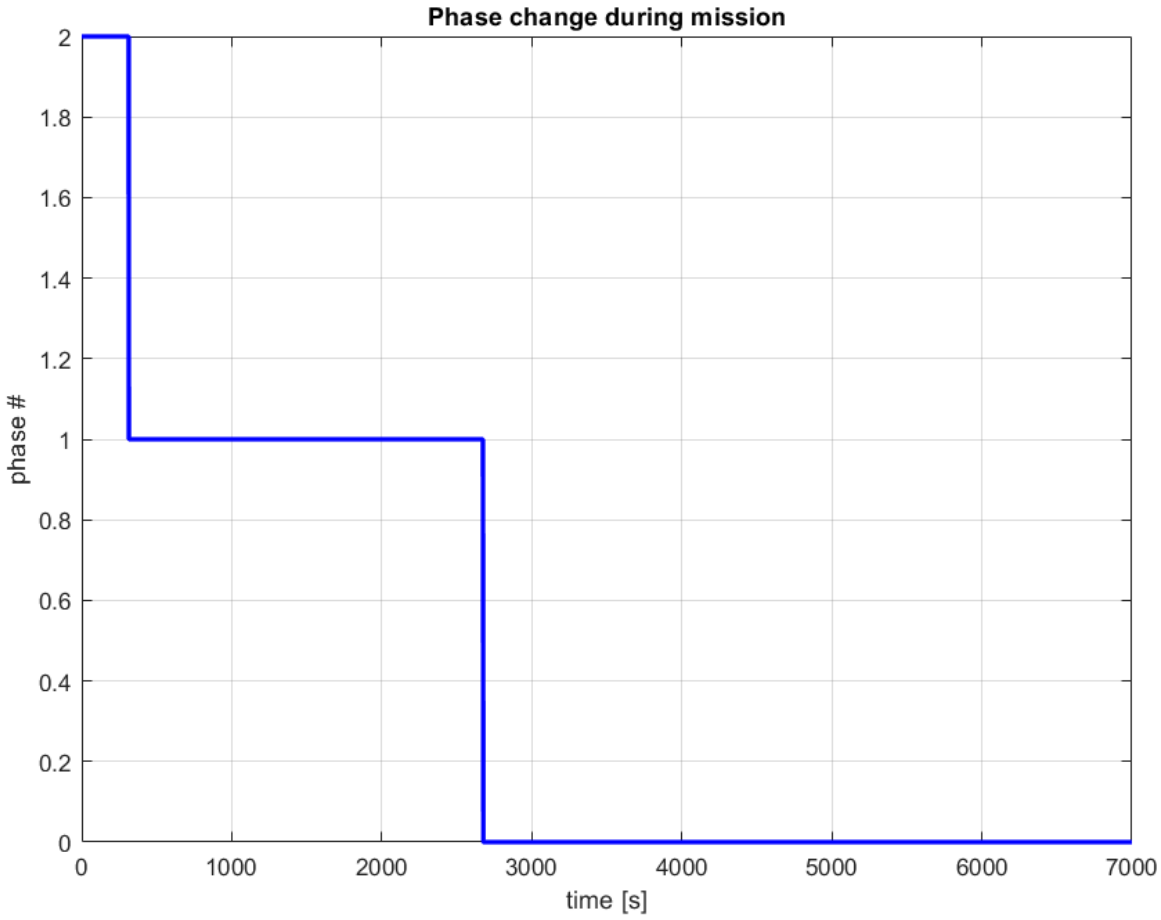


Figure 6.11: Phase transition during phase 2 simulation

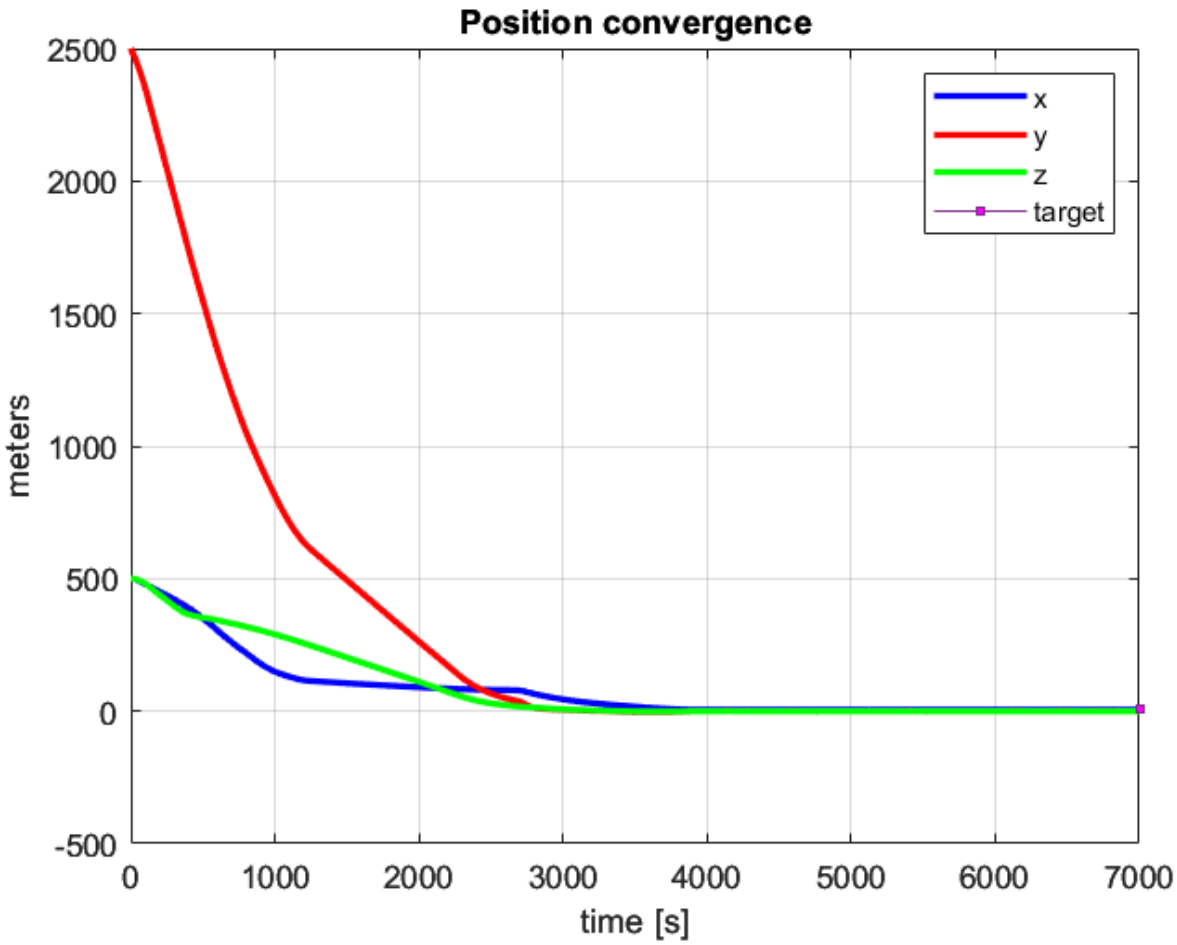


Figure 6.12: Phase 2 position convergence to docking position

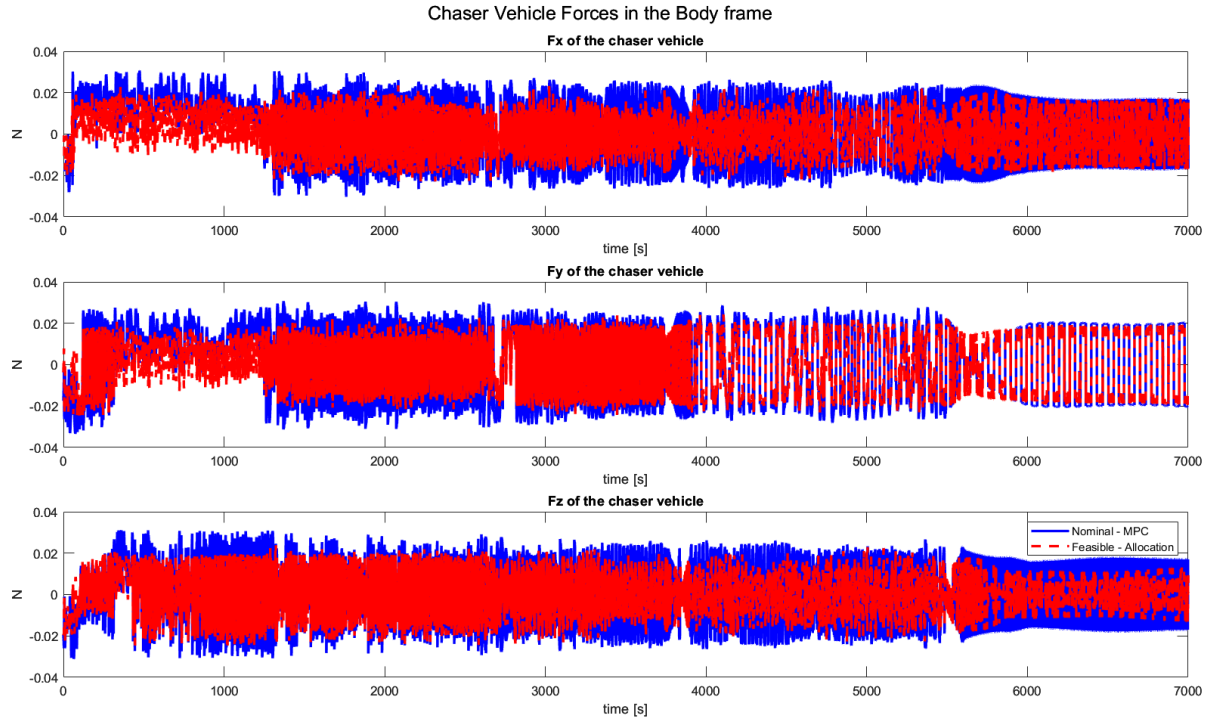


Figure 6.13: Forces from phase 2 mission

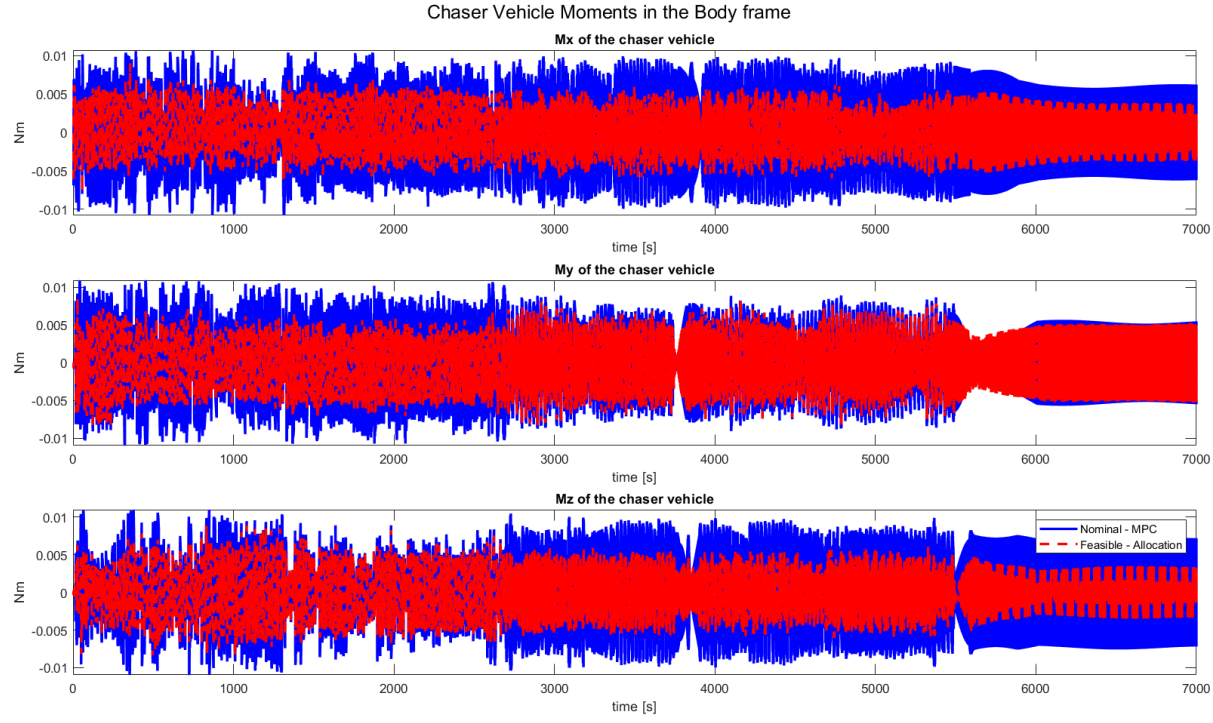


Figure 6.14: Moments from phase 2 mission

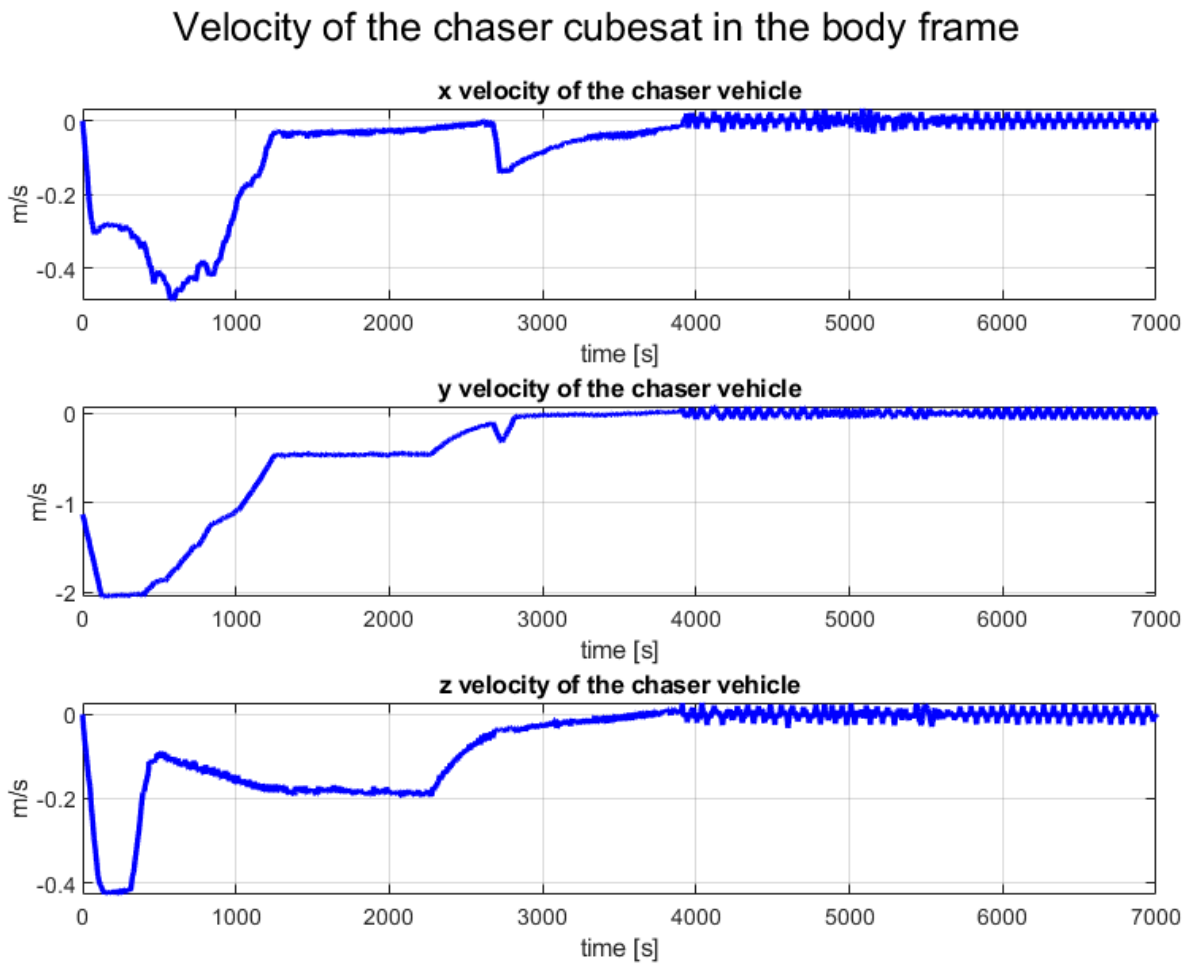


Figure 6.15: Velocity from phase 2 mission

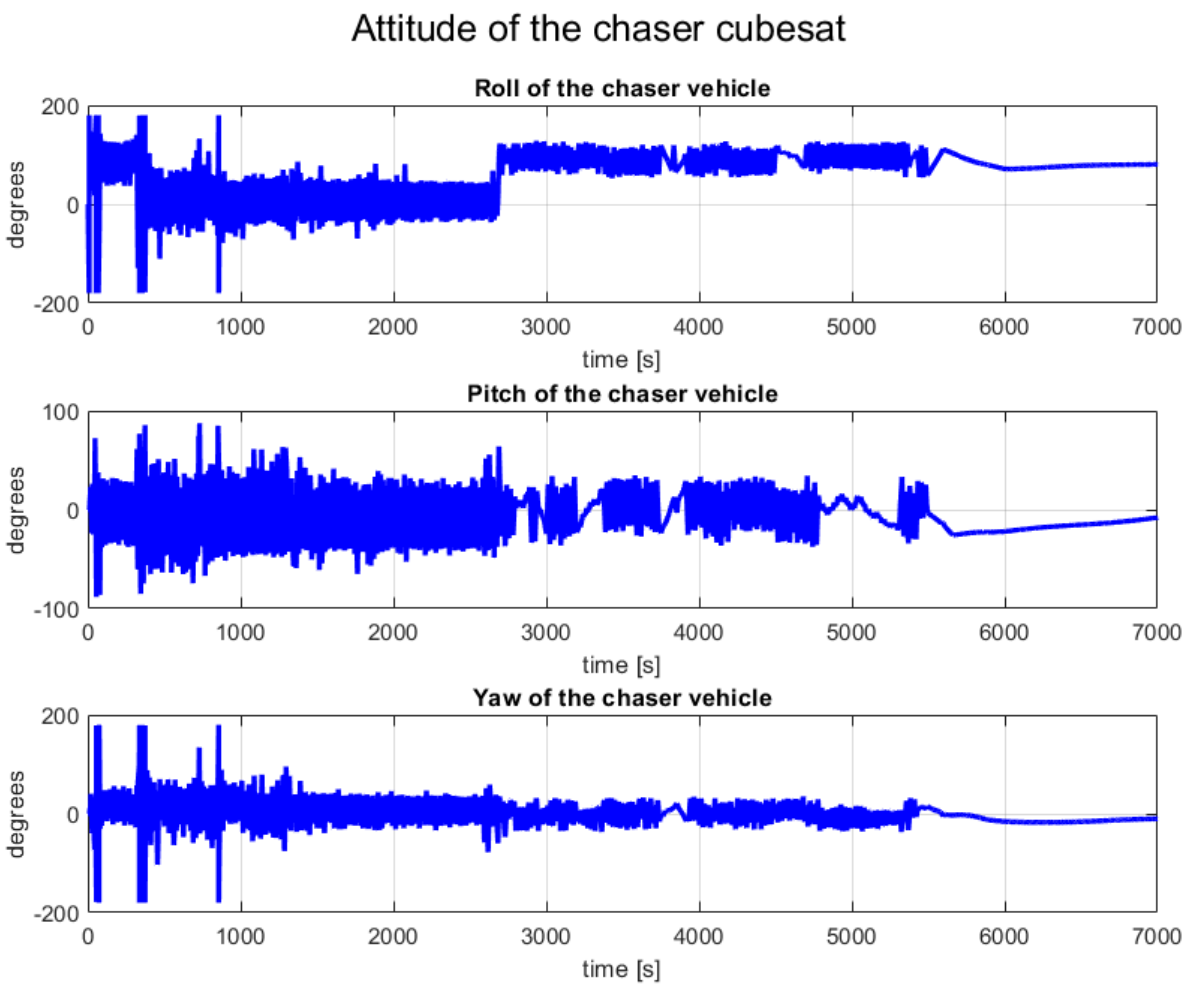


Figure 6.16: Chaser attitude from phase 2 mission

Figure 6.9-6.16 demonstrates the performance of the NMPC when starting in phase 2. The simulation started in phase 2 but quickly entered phase 1. Future tests may require starting at a distance further away to fully test phase 2. Seems new gains might be too strong as they enforce a lot of attitude changes early on and cause even more oscillations in moment input than phase 3. The docking worked excellent as figure 6.10 shows a trajectory entering the line of sight cone. One observation is that the simulation ran for 7000 seconds and even after getting close to docking around 4000 seconds, it wasn't until after 5500 seconds did the oscillations in the chaser cubesat attitude eventually dampen out. Even though starting a mission in phase 2 resulted in a great trajectory, there is still improvement needed to be made on the controller to improve robustness.

6.3 State noise robustness and MPC validation

The results presented earlier assumed a full state feedback for the nonlinear MPC. In a realistic environment the MPC would only receive state feedback from the sensors onboard the cubesat. Ideally the sensors onboard would be able to provide measurements to calculate or estimate each state in equation 2.20. Since the sensor model was not created for this project and saved for future development, a simple case of adding noise to the state vector will be introduced to test the robustness of the nonlinear MPC.

$$\mathbf{x}_{\text{noisy}} = \mathbf{x} + \mathbf{w} \tag{6.8}$$

The term w in the equation 6.8 represents a random noise being added to the state vector described in equation 2.20. Accuracy of position and velocity were found to be roughly +/- 1 m and between +/- 0.01 to 0.05 m/s accurate on satellite GPS receivers [36]. The noise on the quaternions and angular velocity were kept to relatively low noise levels.

Table 6.5: Noise levels on state vector

Mission parameter	Value
Position	+/- 1 m
Velocity	+/- 0.05 m/s
Quaterions	1e-3 rad
Angular velocity	1e-3 rad/sec

The same simulation ran in section 6.2 for a 2500 meter away mission was reran with a noisy state vector being inputted into the jacobians for linearization and for the nonlinear MPC solver block.

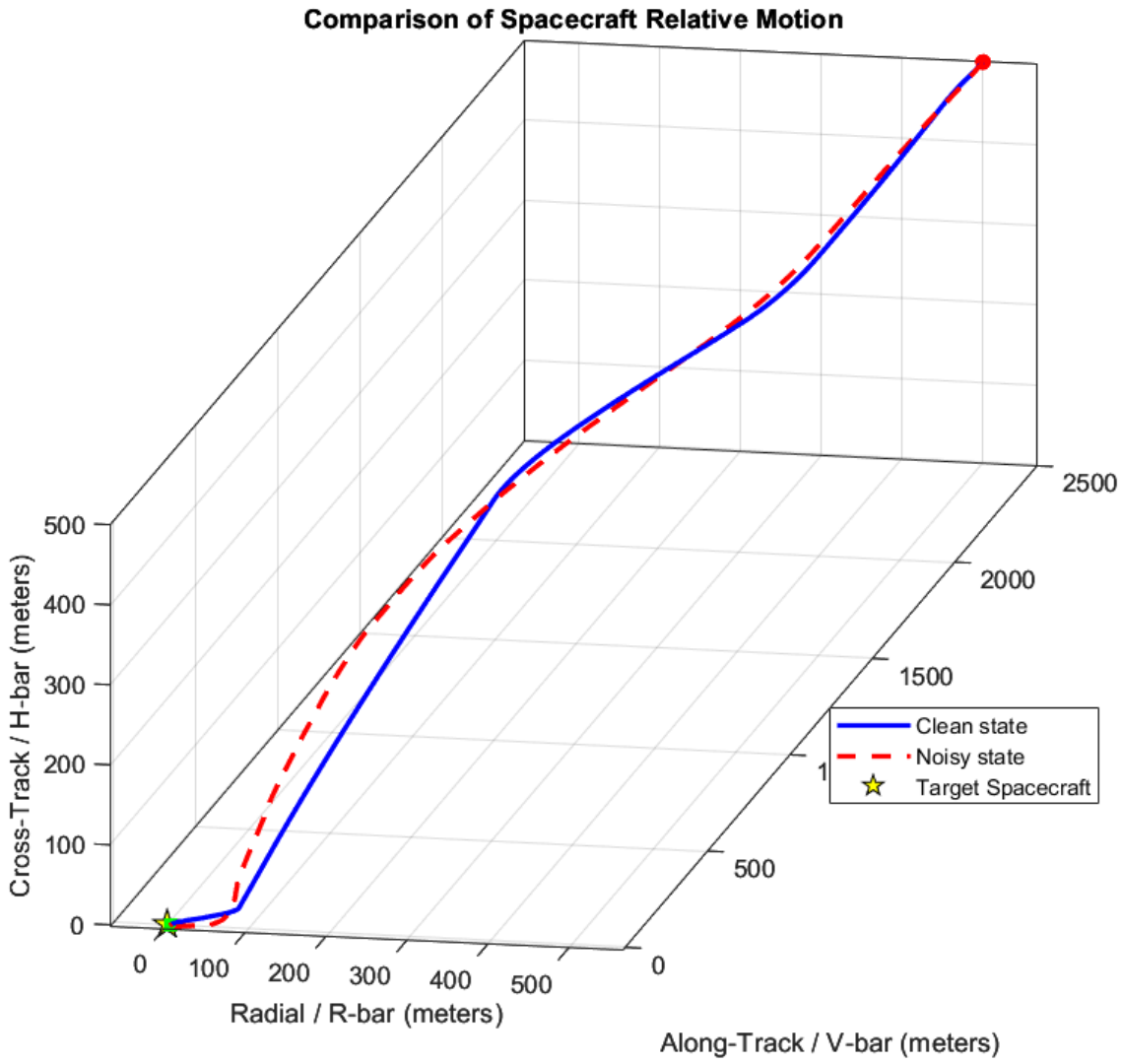


Figure 6.17: Trajectory comparison between ideal and noisy state vector

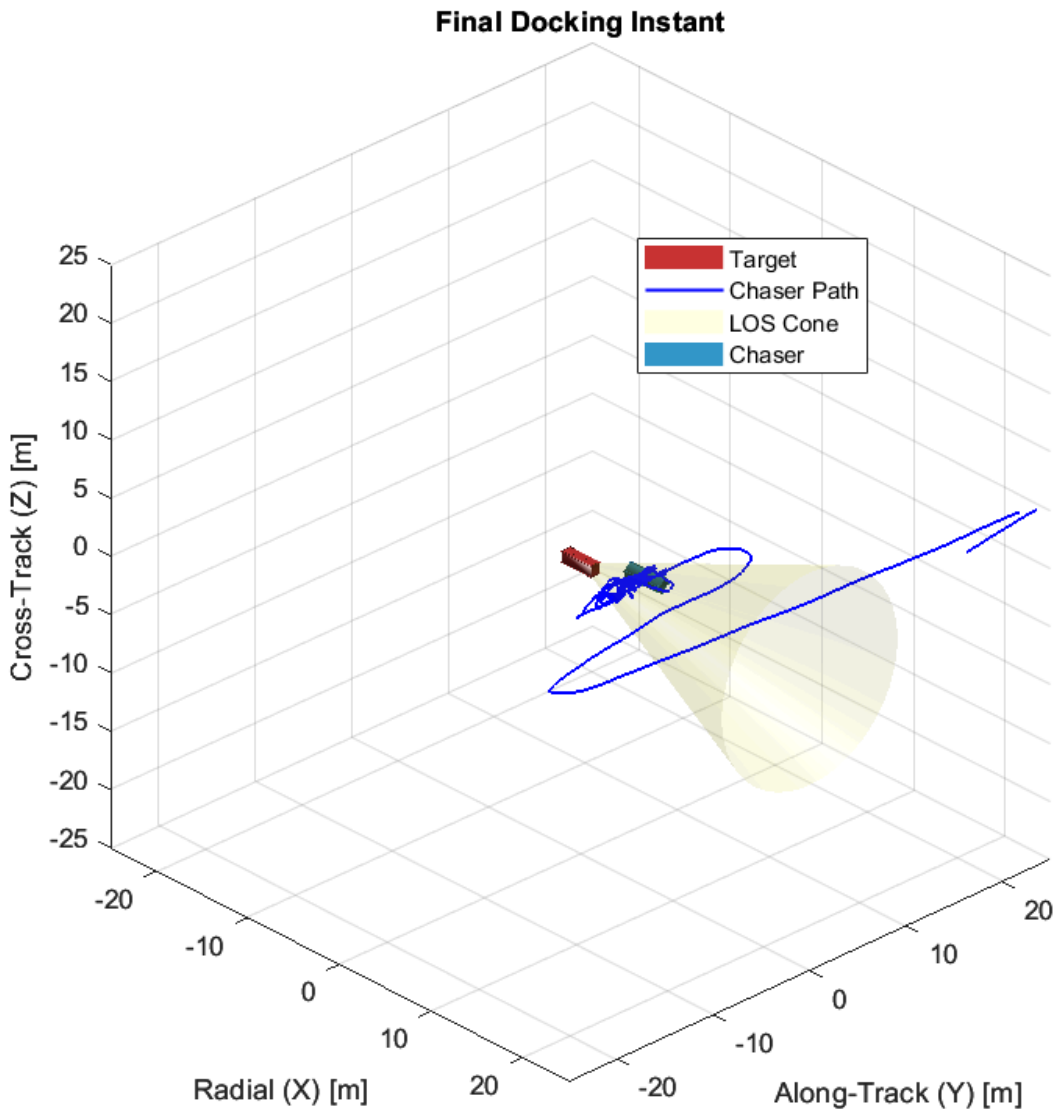


Figure 6.18: Close up of docking with sensor noise

Figure 6.18 and 6.17 showcase the performance of the nonlinear MPC when noise has been added to the state vector. Figure 6.17 compares the nominal trajectory previously shown in figure 6.9 with the new trajectory when the state has amplitude of noise shown in table 6.5. The trajectory is slightly different, with the cubesat unable to make a clear straight path to the reference points listed in table 6.2. Although even with a noise state vector, the nonlinear MPC was able to complete the rendezvous. The docking phase did not yield the same result as shown in figure 6.18. With noisy states the nonlinear MPC was able to completely stay within the line of sight constraint and handle a hard time station keeping at the final docking position.

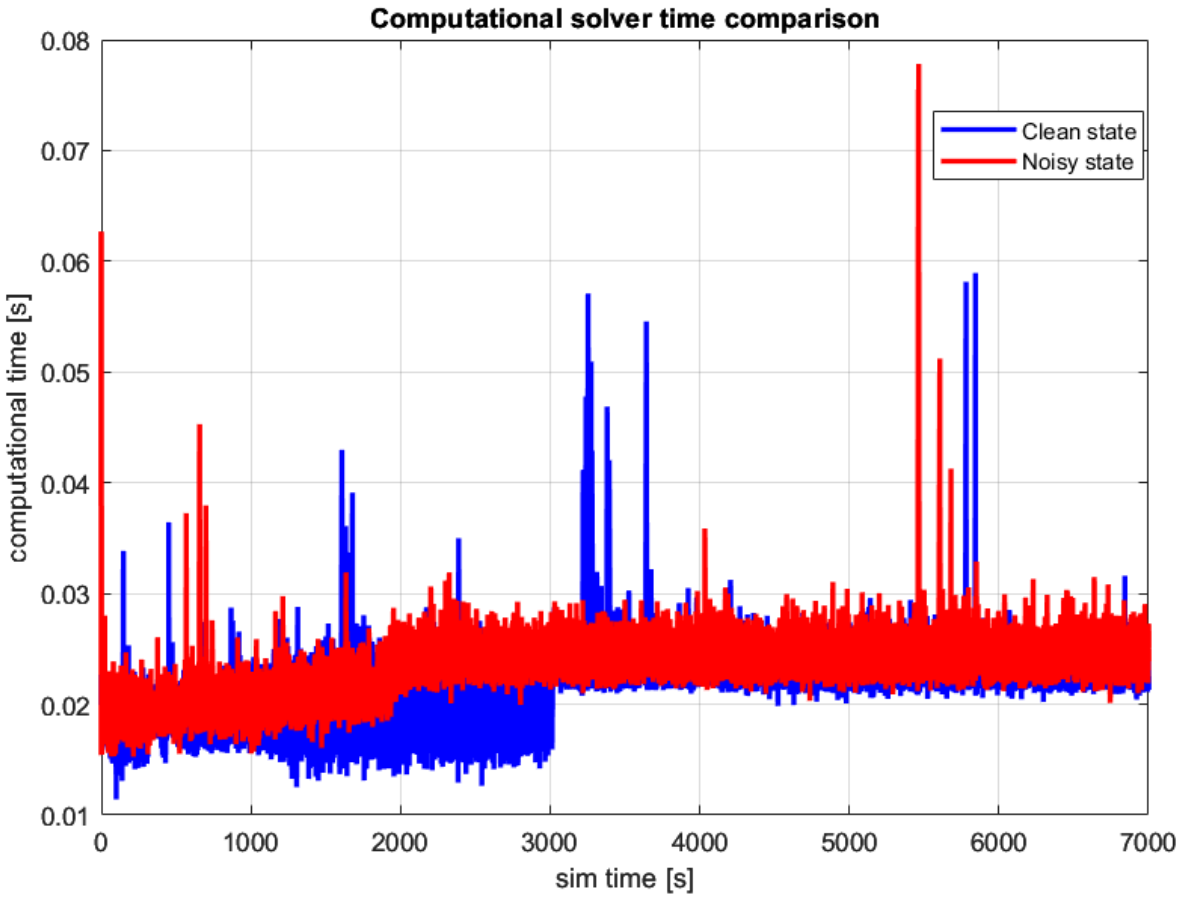


Figure 6.19: Computational solving time comparison

Figure 6.19 demonstrates the nonlinear MPC computational time throughout the simulation. It compares the computational load when receiving clean state data and noise state data. The simulation as whole runs at 200 Hz, with the nonlinear MPC solver running at 5Hz. Each iteration to compute a solution from the nonlinear MPC is roughly between 33 Hz and 50 Hz. This showcases the nonlinear MPC's ability for real time feasibility as it is able to perform below the sample rate of control loop frequency.

6.4 Allocation Performance

The results in section 6.2 and 6.3 were conducted with nominal forces and moments being applied to the dynamics of the chaser cubesat. In section 5.3 the allocation method was discussed by setting up an objective function to minimize force and moment response from the actuators and the force and moment input produced by the nonlinear MPC. Some simulink rewiring is needed to overwrite the nonlinear MPC solver's demanded force and moment vector with the force and moment vector produced from allocation.

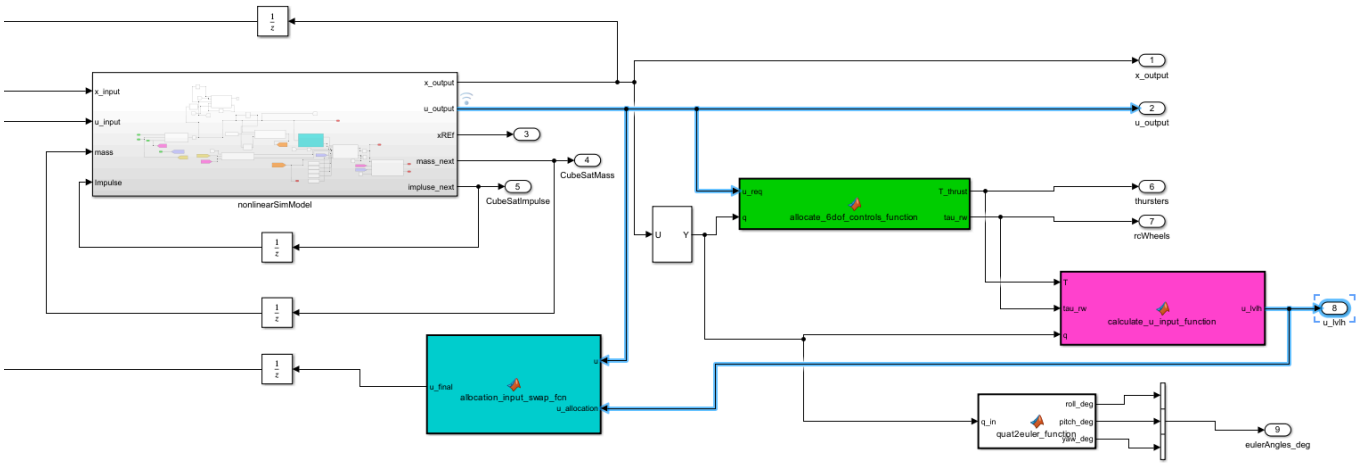


Figure 6.20: Signal routing for allocation into the loop

In figure 6.20 the green colored box is the allocation function box, the magenta box is a matlab function block that takes in the allocation output along with the state quaternions to determine what feasible force and moment vector the actuators can produce from the nonlinear MPC demands. The cyan function box is a custom matlab function to overwrite the nonlinear MPC demands with the output of the magenta box, this uses the actuator achievable force and moments from the allocation scheme in the control loop rather than the demands of nonlinear MPC. Once the simulation was rewired to take in the allocation force and moment input, a simulation was run at 1200 meters away to compare performance between nominal nonlinear MPC and a nonlinear MPC with allocation in the loop.

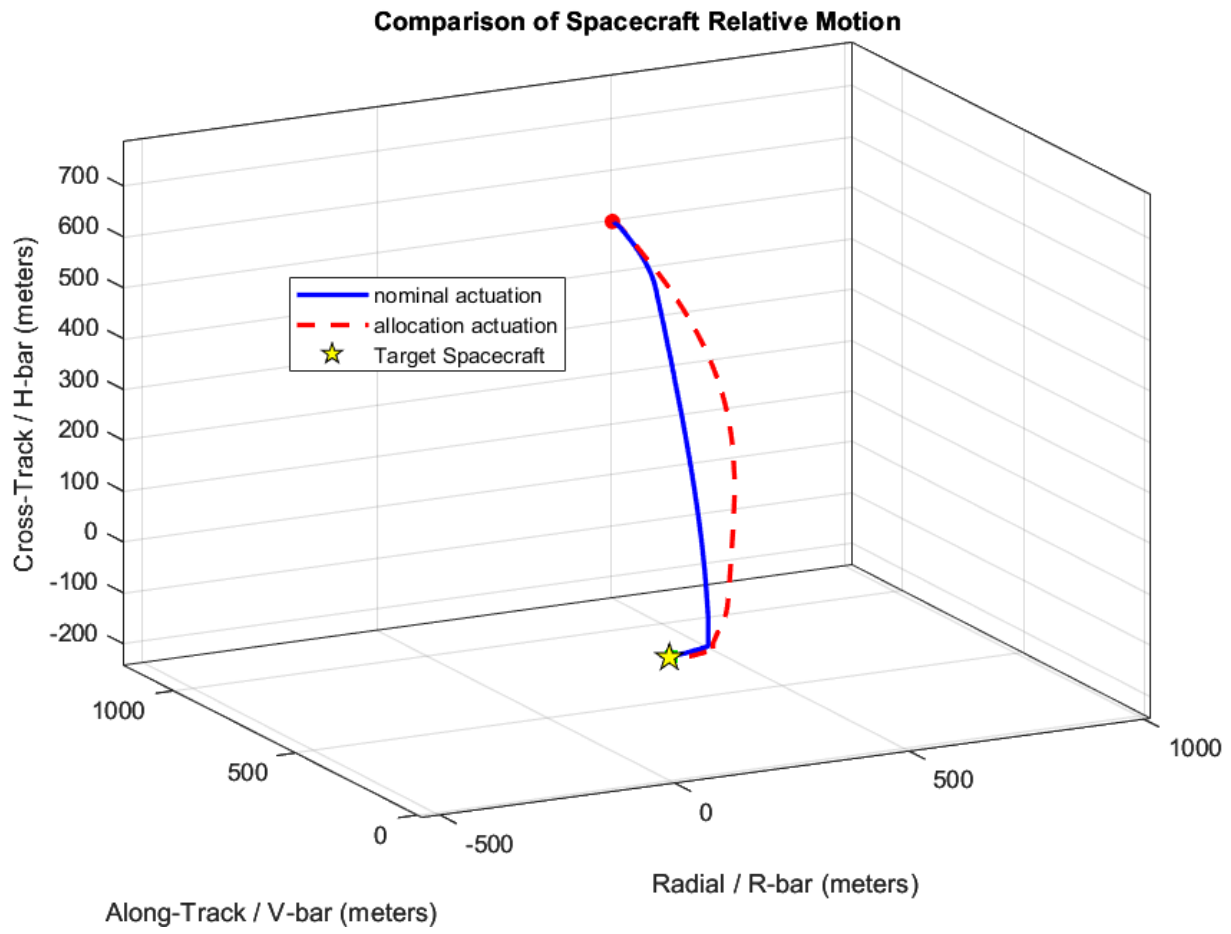


Figure 6.21: Nominal vs allocation actuation trajectory

Figure 6.21 demonstrates the trajectory comparison of a 1200 meter away mission with and without allocation. Allocation provides realistic force and moment inputs based on the thrusters and reaction wheel discussed in chapter 2. The comparison shows that with realistic force and moments the nonlinear MPC has a tougher time reaching for its rendezvous set point as listed in table 6.2. Though the solver does not take the same trajectory as when using the nonlinear MPC demanded force and moments, it still achieves its target of rendezvous and docking.

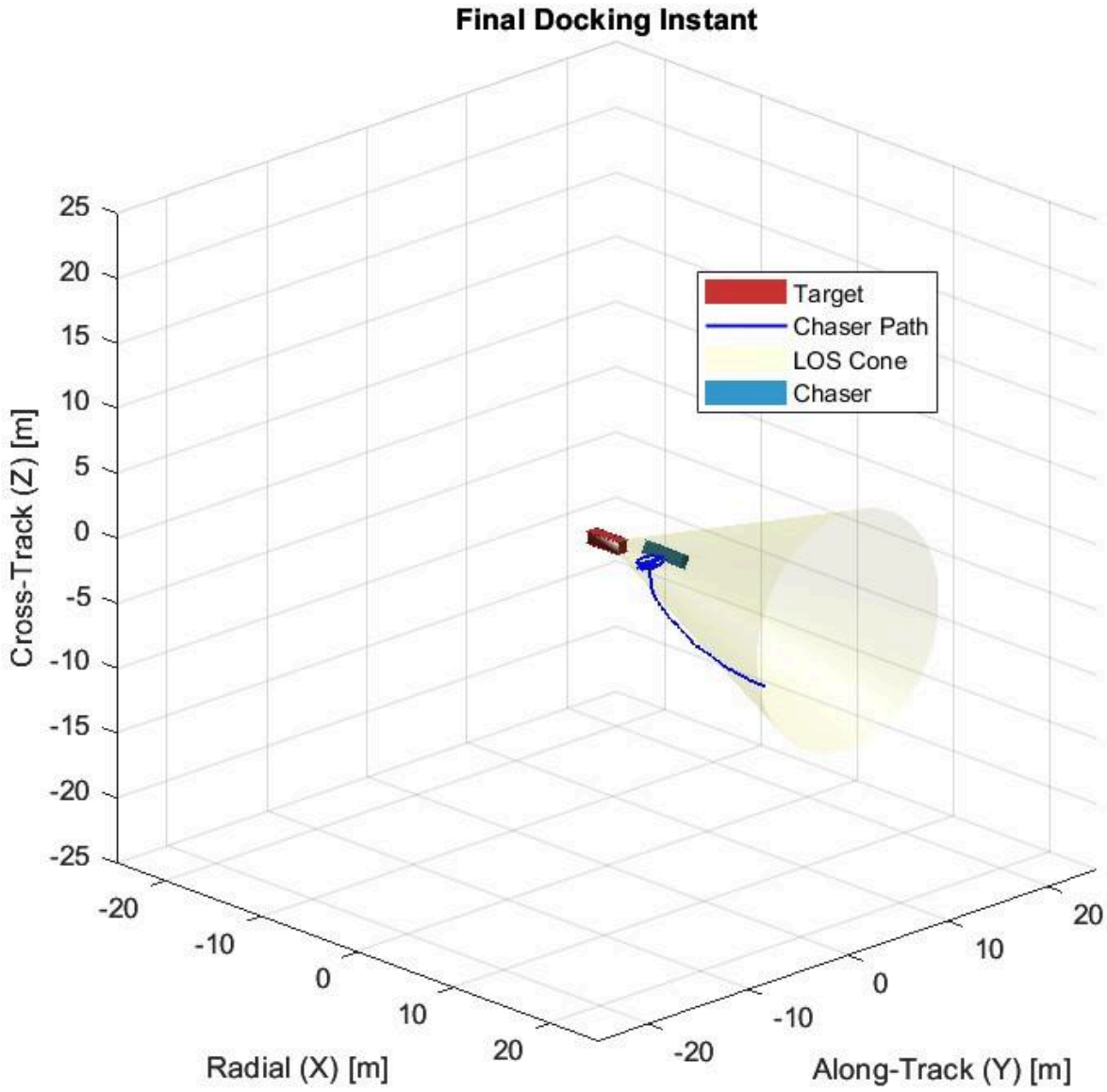


Figure 6.22: Close of up of docking with allocation in the loop

Figure 6.22 shows a close up of the trajectory during docking when allocation is in the control loop. The trajectory is on the border of the line of sight constraint and struggles with station keeping, this suggests that tuning of either allocation or the nonlinear MPC gains are required when using realistic actuator forces and moments.

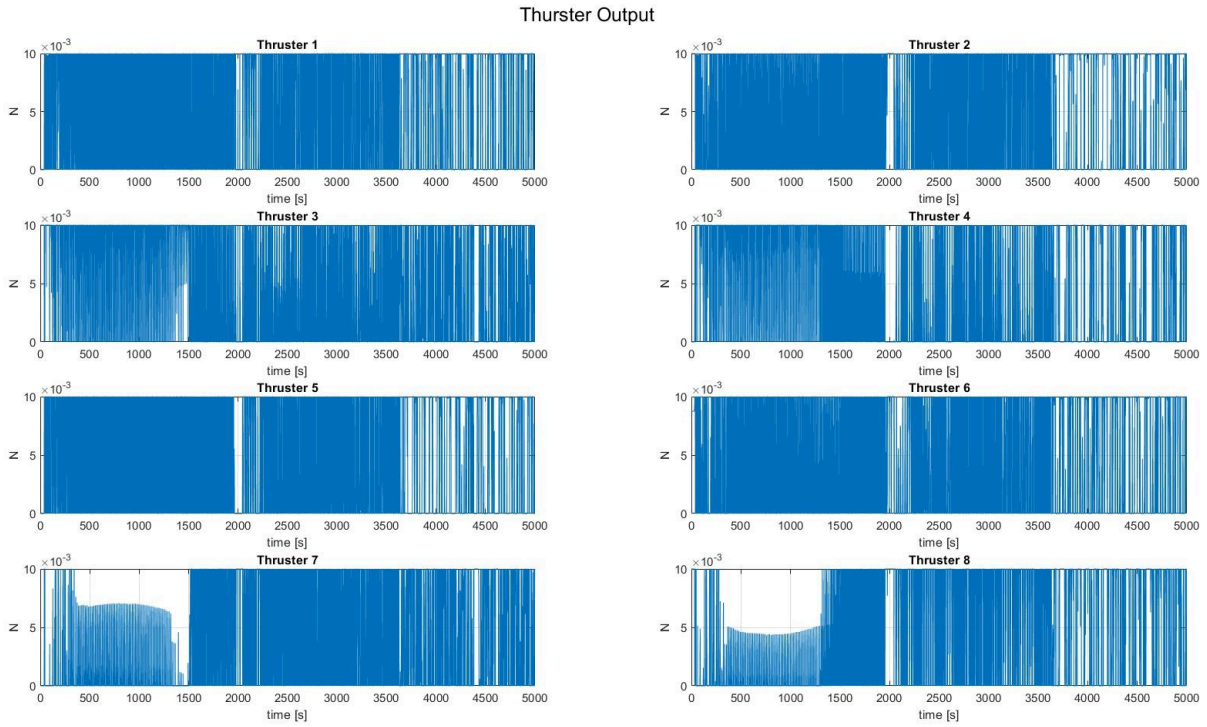


Figure 6.23: Thruster commands

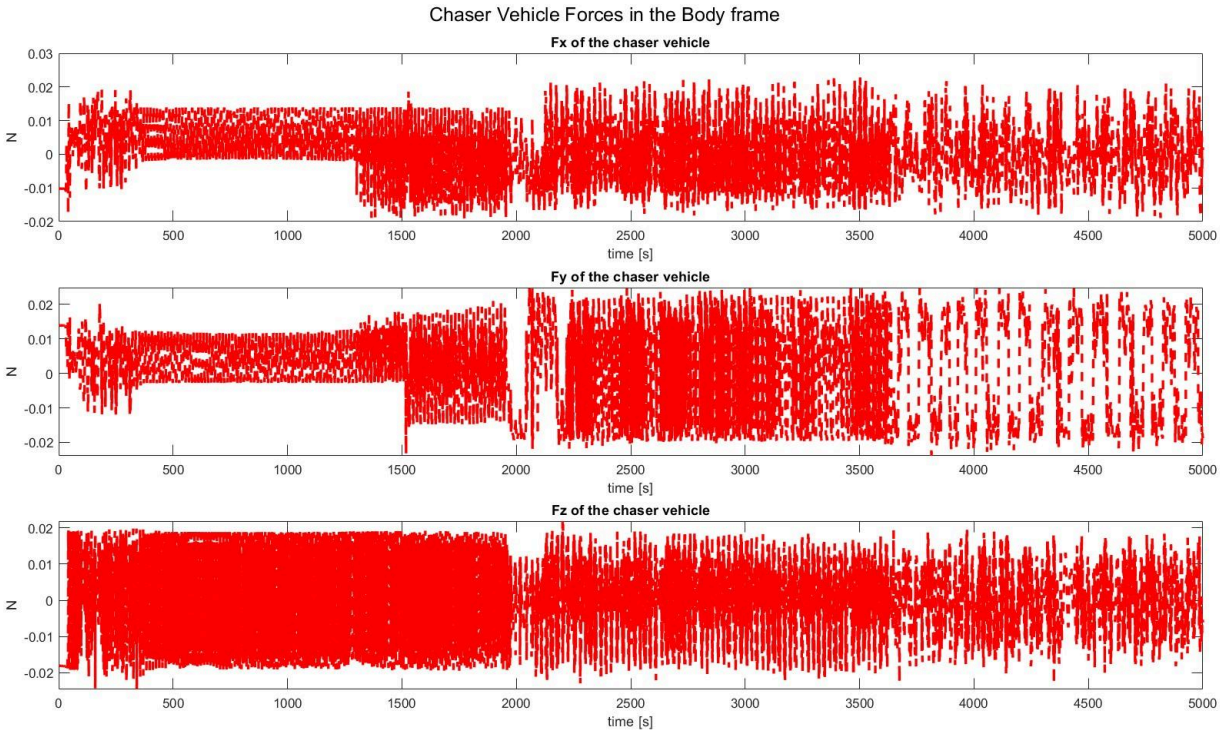


Figure 6.24: Force produced with allocation in the loop

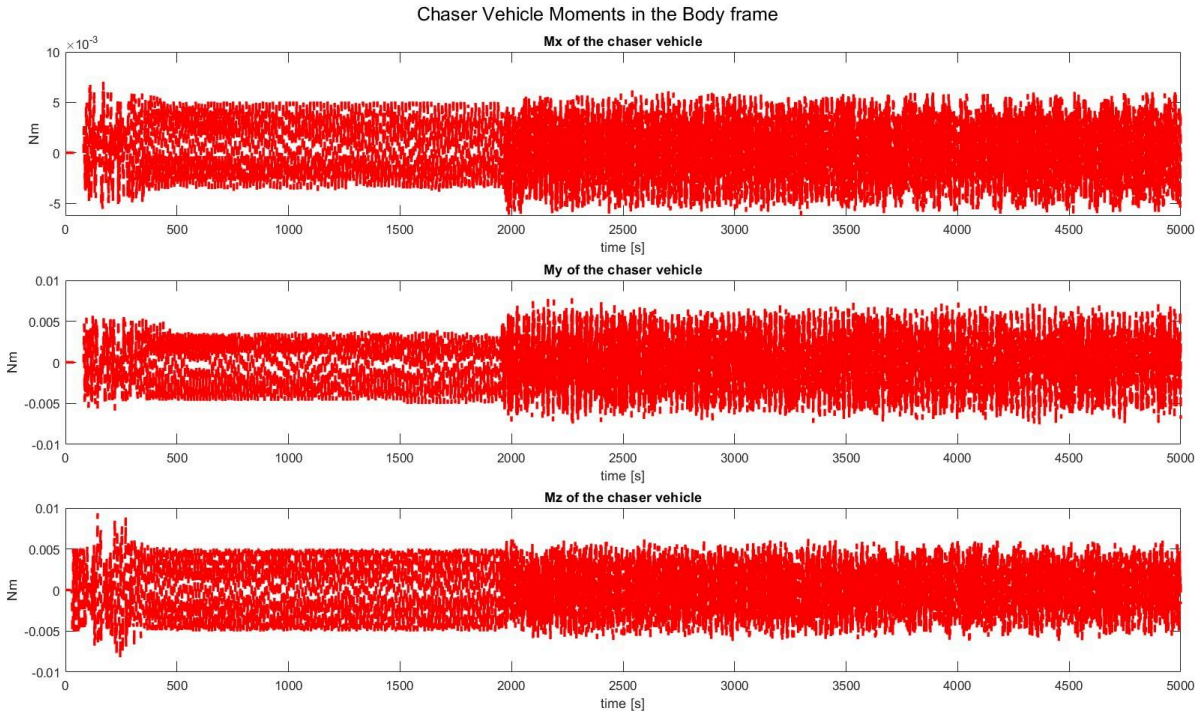


Figure 6.25: Moment produced with allocation in the loop

Figure 6.23 displays the thruster commands for all 8 thrusters described in chapter 2. The thrusters are firing aggressively as it tries to meet the force and moment demands of the nonlinear MPC. Although allocation has provided realistic forces and moments, more modeling is still required as thrusters may not be instantaneous. As for frequency of the thrusters firing, this can be reduced by tuning the R matrix in either the nonlinear MPC or in the R matrix in the allocation scheme. The force and moments produced in figures 6.24 and 6.25 showed similar oscillations when running the simulation without allocation, the only difference being the magnitude of the forces were less than without allocation in the control loop.

6.5 Conclusion

While the nonlinear MPC worked for a mission starting phase 3, it was not optimal as it flew past the target on the initial pass but with tuned gains the nonlinear MPC was able to reach the target position. Phase 2 worked very similarly to phase 1, it took some time but it also was able to reach its final docking port. Phase 1 and 0 worked extremely well as they had little to no problem reaching their target positions as shown in section 5. In both phase 1 and 0 the error between the reference state and the actual state was minimal compared to phases 2 and phases 3.

Seeing struggles with phases 2 and 3 suggests that perhaps a better reference trajectory is needed so that the path is more feasible for the NMPC to solve for control inputs. Other concerns could be that from long distances the cubesat is underpowered and does not have the thrust available to meet the demands of NMPC to meet the predicted states from the solver. This suggests that either stronger thrusters are required or a more optimal reference trajectory is needed so that NMPC does not command extreme control inputs.

Based on the results in sections 5 and 6, the nonlinear MPC does an adequate job in phases 2 and 3 and an excellent job in the rendezvous and docking phases 1 and 0. Though the controller is not optimal it still manages to achieve its objective. When adding realistic sensor noise to the state vector, the nonlinear MPC was still able to achieve its target goal, though the trajectory did differ from the nominal test simulation. Also based on the solver computational time in figure 6.18, the nonlinear MPC demonstrated its real time feasibility as it was well within the control loop sampling time. As for the allocation, it demonstrated the ability to provide the thruster inputs and reaction wheel torques needed to complete a rendezvous and docking mission from at least 1200 meters away. Further out missions required gain tuning as the mission was still achievable but did not take an optimal trajectory to the target set points.

7. Next steps

7.1 Recommended future work

Oscillations in the force and moments demanded from the nonlinear MPC suggest more gain tuning is required in either the allocation scheme or the nonlinear MPC cost function. The allocation scheme in this project was set-up to provide a reference of how the thrusters were being allocated and when held to physical constraints (thrusters being pushers only) could they achieve the control input demands of the nonlinear MPC. The gains for both the allocation scheme and nonlinear MPC have not been optimized, leaving room for improvement.

Another area of future work would be to create an optimization scheme to optimize the gains of this cubesat. Perhaps looking into the plentiful MATLAB global optimization toolbox and work with genetic algorithms or particle swarm optimization. Hopefully these optimization algorithms can help smooth out the thrust and moment inputs from the NMPC.

There are a few items that could go a long way to increasing the fidelity of the dynamics used for the NMPC. In this project the J2 effect assumed that there is no inclination between target and chaser cubesat, this simplified the equations and assisted with simulation time. To further increase the fidelity of the dynamics, J2 effects will need to be considered when inclination is present.

Lastly the reference trajectory can be updated to handle phase 3 missions much better by adding a hohmann transfer trajectory in that phase as suggested in one of the references [30]. This will yield more accurate results in determining reference states for the NMPC to work with.

8. References

[1] Uri, J., “55 Years Ago: Gemini VIII, the First Docking in Space”, *NASA*, 16 March 2021. Retrieved 28 September 2025 from https://www.nasa.gov/history/55-years-ago-gemini-viii-the-first-docking-in-space/.

[2] Neufeld, M., “Spinning Out of Control: Gemini VIII's Near-Disaster,” *National Air and Space Museum*, 16 March 2021. Retrieved 28 September 2025 from https://airandspace.si.edu/stories/editorial/spinning-out-control-gemini-viii.

[3] Pellegrino, S., and California Institute of Technology, “Autonomous Assembly of a Reconfigurable Space Telescope (AAReST),” *Caltech*. Retrieved 29 September 2025 from https://www.pellegrino.caltech.edu/aarest-project.

[4] Bashnick, C. and Ulrich, S., “Fast model predictive control for spacecraft rendezvous and docking with obstacle avoidance,” *Journal of Guidance, Control, and Dynamics*, Vol. 46, 2023, pp. 998–1007.

[5] MathWorks, "Spacecraft Rendezvous and Docking," *MathWorks Help Center*. Retrieved 29 September, 2025 from https://www.mathworks.com/help/aeroblks/spacecraft-rendezvous-and-docking.html.

[6] Hartley, E. N., "A tutorial on model predictive control for spacecraft rendezvous," *2015 European Control Conference (ECC)*, Linz, Austria, July 2015, pp. 1355-1361. DOI: 10.1109/ECC.2015.7330727.

[7] Patel, J., and Kumar, S. R., "Spacecraft Rendezvous and Docking with Obstacle Avoidance using Model Predictive Control," *AIAA SciTech Forum*, 2023, Paper No. 2023-2501.

[8] Mehamed, A. O., and Kim, D., "Real-Time Fuel Optimization and Guidance for Spacecraft Rendezvous and Docking," *Aerospace*, Vol. 9, No. 5, Article 276, 2022.

[9] Burak, I. O., "Model predictive control for spacecraft rendezvous and docking with uncooperative targets," Thesis, Nanyang Technological University (NTU), Singapore, 2020. doi:10.32657/10356/144018.

[10] Basu, H., Fernandez, M. C., Sanfelice, R. G., and Kolmanovsky, I., “Hybrid Model Predictive Control Approach for Spacecraft Proximity Maneuvering and Docking Accounting for Collisions,” *2025 American Control Conference (ACC)*, IEEE, 2025, pp. 4954–4959. doi: 10.23919/acc63710.2025.11107626.

[11] Rawlings, J. and Mayne, D., *Model Predictive Control: Theory and Design*, Nob Hill Pub., 2009.

[12] Botelho, A., Parreira, B., Rosa, P. N., and Lemos, J. M., *Predictive Control for Spacecraft Rendezvous*, Springer, Cham, Switzerland, 2021.

[13] Li, W., Zhang, X., Wang, Y., and Xie, S., "Comparison of Linear and Nonlinear Model Predictive Control in Path Following of Underactuated Unmanned Surface Vehicles," *Journal of Marine Science and Engineering*, Vol. 12, No. 4, 2024, Art. No. 575. Retrieved 29 September 2025 from https://www.mdpi.com/2077-1312/12/4/575.

[14] Basnayake, I., Park, H., Kohler, J., Hudson, J., and Romano, M., "In-Space Demonstration of Model Predictive Control Approaches for Space Towing of Uncertain Loads," AIAA 2024-1069, *AIAA SciTech Forum*, 2024. doi: 10.2514/6.2024-1069.

[15] Di Cairano, S., Park, H., and Kolmanovsky, I., "Model Predictive Control Approach for Guidance of Spacecraft Rendezvous and Proximity Maneuvering," TR2012-099, Mitsubishi Electric Research Laboratories, Cambridge, MA, 2012.

[16] Park, H., Zappulla II, R., Zagaris, C., Virgili-Llop, J., and Romano, M., "Nonlinear Model Predictive Control for Spacecraft Rendezvous and Docking with a Rotating Target," *27th AAS/AIAA Spaceflight Mechanics Meeting*, San Antonio, TX, AAS Paper 17-496, Feb. 2017.

[17] Stesina, F., "Tracking Model Predictive Control for Docking Maneuvers of a CubeSat with a Big Spacecraft," *Aerospace*, Vol. 8, No. 8, Article 197 (2021).

[18] Spiegel, I. A., Zhou, B., Goodloe, R. J., Fox, B., and DiMatteo, J., "CubeSat Proximity Operations Demonstration (CPOD) Mission Results," Presented at the 37th Annual Small Satellite Conference, Logan, UT, Aug. 2023.

[19] Nammo, "Reaction Control Module (RCM) Technical Data Sheet," Report No. X13003000-01, Nammo, Raufoss, Norway, 2016. Retrieved 10 October 2025 from https://cubesat-propulsion.com/wp-content/uploads/2022/04/X13003000-01_RCM_2016update.pdf.

[20] NanoAvionics, "CubeSat Reaction Wheels Control System SatBus 4RW," NanoAvionics, Vilnius, Lithuania. Retrieved 10 October 2025 from https://nanoavionics.com/cubesat-components/cubesat-reaction-wheels-control-system-satbus-4rw/.

[21] "2ndSpace VERSE-03 3U Structure." SatSearch .Retrieved 31 October 2025 from https://satsearch.co/products/2ndspace-verse-03-3u-structure

[22] NASA, "CubeSat Proximity Operations Demonstration (CPOD)," *Small Spacecraft Technology Program*, NASA, Washington, DC, 25 Jul. 2023. Retrieved 31 October 2025 from https://www.nasa.gov/smallspacecraft/cpod-project/.

[23] Martínez Calzado, A., "An analysis of the thermospheric drag upon 3U CubeSats," Master's Thesis, Dept. of Aerospace Engineering, Technical University of Catalonia (UPC), Barcelona,

Spain, 2023. Retrieved 31 October 2025, from https://upcommons.upc.edu/server/api/core/bitstreams/86404399-c379-422d-9a7f-737f3af55b39/content.

[24] Lam, Q., Anderson, B., and Xin, M., "Preserving Spacecraft Attitude Control Accuracy Using Theta-D Controller Subject to Reaction Wheel Failures," AIAA Infotech@Aerospace 2010, Atlanta, GA, April 2010, Paper 2010-3522.

[25] Yoon, H., Seo, H. H., and Choi, H.-T., "Optimal uses of reaction wheels in the pyramid configuration using a new minimum infinity-norm solution," *Aerospace Science and Technology*, Vol. 39, No. 1, 2014, pp. 109–119. doi: 10.1016/j.ast.2014.09.002.

[26] smit009, "2. The Different Frames and the Keplerian Elements," *ADCS For Beginners*, 12 July 2015. Retrieved 10 November 2025 from https://adcsforbeginners.wordpress.com/tag/earth-centred-inertial-frame/.

[27] Lim, H., Bang, H., and Lee, S., "Adaptive Backstepping Control for Satellite Formation Flying with Mass Uncertainty," *Journal of Astronomy and Space Sciences*, Vol. 23, No. 4, 2006, pp. 405–414. https://doi.org/10.5140/JASS.2006.23.4.405

[28] "Spire Global Nanosatellite Constellation", *eoPortal*, 29 July 2025. Retrieved 1 December 2025 from https://www.eoportal.org/satellite-missions/spire-global.

[29] Wang, L., *Model Predictive Control System Design and Implementation Using MATLAB®*, Springer-Verlag, London, 2009

[30] Hovd, M., "A Brief Introduction to Model Predictive Control," Course Notes TTK4135, Dept. of Engineering Cybernetics, Norwegian Univ. of Science and Technology, Trondheim, Norway, March 2004.

[31] Fehse, W., *Automated Rendezvous and Docking of Spacecraft*, Cambridge University Press, Cambridge, UK, 2003.

[32] Krebs, G. D., "PAN (Pathfinder for Autonomous Navigation)," *Gunter's Space Page*. Retrieved 26 March 2026 from https://space.skyrocket.de/doc_sdat/pan_cubesat.htm

[33] The MathWorks, Inc., “quadprog,” *Optimization Toolbox Documentation*. Retrieved 5 April 2026 from https://www.mathworks.com/help/optim/ug/quadprog.html.

[34] Bodson, M., "Evaluation of Optimization Methods for Control Allocation," *Journal of Guidance, Control, and Dynamics*, Vol. 25, No. 4, 2002, pp. 703–711. https://doi.org/10.2514/2.4937

[35] Curtis, H. D., *Orbital Mechanics for Engineering Students*, 3rd ed., Elsevier, Oxford, UK, 2014.

[36] Kim, E., Han, S., and Al Sayegh, A. M., "Sensitivity of the Gravity Model and Orbital Frame for On-board Real-Time Orbit Determination: Operational Results of GPS-12 GPS Receiver," *Remote Sensing*, Vol. 11, No. 13, 2019, 1542. https://doi.org/10.3390/rs11131542

Appendices

Appendix A — MATLAB Code: LVLH frame dynamics function

```
function [x_dot] = dynamics_lvlh(x, u, mass, params)
% DYNAMICS_lvlh Implements the continuous-time nonlinear state dynamics
% between the target and chaser cubeSat
eps = 1e-12;
% Translational States
r = x(1:3);
r_dot = x(4:6);
r_z_offset = params.xo(1);
% Rotational States
q = x(7:10);
q = q / (norm(q) + 1e-12); % eps added to avoid division by zero
omega_b = x(11:13);
q_mod = [q(4), q(1), q(2), q(3)];
% division by 0 protection
if norm(r_dot) < eps
  v_unit_inertial = [1;0;0];
else
  v_unit_inertial = r_dot / norm(r_dot);
end
% Convert Quaternion to Rotation Matrix and Rotate Velocity
R_I2B = myQuat2Rotm(q_mod);
v_body = R_I2B * v_unit_inertial;
% Nan protection
if any(isnan(v_body))
  v_body = [0;0;0];
end

areas = [0.01, 0.01, 0.03, 0.03, 0.03, 0.03];
normals = [1,0,0; -1,0,0; 0,1,0; 0,-1,0; 0,0,1; 0,0,-1];

A_eff = 0;
for i = 1:6
```

```
    projection = dot(normals(i,:), v_body);
    if projection > 0
        A_eff = A_eff + (areas(i) * projection);
    end
end
% Control Inputs
F_ctrl = u(1:3);
tau_ctrl = u(4:6);
% Physical Parameters
mu = 3.986004418e14;
R_t = 6378000 + r_z_offset;
n = sqrt(mu / R_t^3);
J = diag([0.05, 0.05, 0.05]);
z_alt = R_t + r(3);
% Atmospheric Drag Calculation
rho = 2.418e-11; % constant for now
CD = 2.2;
v_rel = norm(r_dot);
if v_rel > eps
    a_drag = -0.5 * rho * (CD * A_eff / mass) * v_rel * r_dot;
else
    a_drag = zeros(3,1);
end
% J2 Gravitational Perturbation Calculation
J2 = 1.08263e-3;
Re = 6378137;
mu = 3.986004418e14;
% Absolute position of Target
r_t_mag = R_t;
a_j2_target_x = -(3/2) * J2 * (mu * Re^2 / r_t_mag^4);
% Absolute position of Chaser
r_c_mag = sqrt((R_t + r(1))^2 + r(2)^2 + r(3)^2);
r_c_mag = max(r_c_mag, Re);
a_j2_chaser_x = -(3/2) * J2 * (mu * Re^2 / r_c_mag^4) * ((R_t + r(1))/r_c_mag);
```

```
a_j2_chaser_y = -(3/2) * J2 * (mu * Re^2 / r_c_mag^4) * (r(2)/r_c_mag);
% Differential Acceleration
a_j2_lvlh_x = a_j2_chaser_x - a_j2_target_x;
a_j2_lvlh_y = a_j2_chaser_y;
a_j2_lvlh_z = 0;
% Total Acceleration update
a_j2_x = a_j2_lvlh_x;
a_j2_y = a_j2_lvlh_y;
a_j2_z= a_j2_lvlh_z;
% Nonlinear Gravity Gradient
G_NL_x = - (mu * (R_t + r(1))) / r_c_mag^3 + mu / R_t^2;
G_NL_y = - (mu * r(2)) / r_c_mag^3;
G_NL_z = - (mu * r(3)) / r_c_mag^3;
% Rotational/Coriolis Effects
a_rotational = [2*n*r_dot(2) + (n^2)*r_dot(1);
            -2*n*r_dot(1) + (n^2)*r_dot(2);
             0];
% Total Acceleration
x_dot = zeros(13, 1);
x_dot(1:3) = r_dot;
x_dot(4) = a_rotational(1) + G_NL_x + F_ctrl(1)/mass + a_drag(1) + a_j2_x;
x_dot(5) = a_rotational(2) + G_NL_y + F_ctrl(2)/mass + a_drag(2) + a_j2_y;
x_dot(6) = a_rotational(3) + G_NL_z + F_ctrl(3)/mass + a_drag(3) + a_j2_z;
% Quaternion Kinematics
Omega = [ 0,         -omega_b(1), -omega_b(2), -omega_b(3);
        omega_b(1),  0,          omega_b(3), -omega_b(2);
        omega_b(2), -omega_b(3),  0,          omega_b(1);
        omega_b(3),  omega_b(2), -omega_b(1),  0          ];
x_dot(7:10) = 0.5 * Omega * q;
% Rotational Dynamics
cross_term = cross(omega_b, J * omega_b);
omega_dot = J \ (tau_ctrl - cross_term);
x_dot(11:13) = omega_dot;
End
```

Appendix B — MATLAB Code: NMPC Solver Function

```
function U_optimal = NMPC_Solver(F_max, Tau_max, X_k, A_k, B_k, X_ref,
currImp)
  %NMPC_solver takes in maximum force and moments, state vector from the
  %nonlinear dynamics function, linearized A and B matrices from the
  %jacobian calculator function, reference state from the reference
  %trajectory function and uses quadprog function to output nominal force
  %and moment vector u.
  % state set-up
  N = 20;
  NX = 13;
  NU = 6;
  n_max = (NU * N) + 1;

  % Coder Setup for quadprog function
  coder.varsize('H_qp', [n_max, n_max], [1, 1]);
  coder.varsize('f_qp', [n_max, 1], [1, 0]);
  coder.varsize('A_inst', [5*N, n_max], [1, 1]);
  coder.varsize('b_inst', [5*N, 1], [1, 0]);
  x_pos = X_k(1);
  dist = norm(X_k(1:3));
  dist_y = abs(X_k(2));
  if dist > 8500
     phase = 3;
  elseif dist > 2000
     phase = 2;
  elseif dist > 90
     phase = 1;
  else
     phase = 0;
  end
  if phase == 3
     Q_diag = [200, 199, 10, 2000, 1000, 10, 1, 1, 1, 1, 10, 10, 10];
     R_diag = [0.7, 0.5, 0.5, 5, 10, 5];
```

```
    S_penalty = 1000;
    use_los = false;
elseif phase == 2
    Q_diag = [200, 199, 50, 20879, 1000, 10, 1, 1, 1, 1, 10, 10, 10];
    R_diag = [0.3932, 0.2919, 10, 5, 10, 5];
    S_penalty = 100;
    use_los = false;
elseif phase == 1
    Q_diag = [20, 20, 10, 10, 10, 10, 1, 1, 1, 1, 10, 10, 10];
    R_diag = [1, 3, 1, 2, 2, 2];
    S_penalty = 500;
    use_los = false;
else
    Q_diag = [20, 20, 20, 10, 10, 10, 100, 100, 100, 100, 10, 10, 10];
    R_diag = [1, 1, 1, 0.1, 0.01, 0.1];
    S_penalty = 500;
    use_los = true;
end
Ad = A_k;
Bd = B_k;
% Numerical Safety: Cap Ad to prevent Inf/NaN at final N
Ad(abs(Ad) > 100) = sign(Ad(abs(Ad) > 100)) * 100;
if any(~isfinite(Ad(:)))
    Ad = eye(NX);
end
Phi = zeros(NX*N, NX);
Gamma = zeros(NX*N, NU*N);
A_p = eye(NX);
for i = 1:N
    A_p = A_p * Ad;
    Phi((i-1)*NX+1:i*NX,:) = A_p;
    B_acc = eye(NX);
    for j = 1:i
        Gamma((i-1)*NX+1:i*NX, (j-1)*NU+1:j*NU) = B_acc * Bd;
```

```
        B_acc = Ad * B_acc;
    end
end
Q_step = diag(Q_diag);
Q_big = kron(eye(N), Q_step);
Q_big(end-NX+1:end, end-NX+1:end) = S_penalty * Q_step;

R_big = kron(eye(N), diag(R_diag));
H_base = Gamma' * Q_big * Gamma + R_big;
f_base = Gamma' * Q_big * (Phi * X_k - reshape(X_ref(:,2:N+1), NX*N, 1));
% Los constraint
max_drop = -100000;
C_safe = zeros(1, 13);
C_safe(1) = -1;

if use_los
    target_th = deg2rad(20);
    curr_th = atan2(norm(X_k(2:3)), abs(X_k(1)) + 1e-6);
    tan_th = tan(max(target_th, curr_th + deg2rad(2)));
    C_los = [-tan_th 1 0 zeros(1,10); -tan_th -1 0 zeros(1,10); ...
            -tan_th 0 1 zeros(1,10); -tan_th 0 -1 zeros(1,10)];

    H_qp = blkdiag(H_base, 1e4);
    f_qp = [f_base; 0];
    A_inst = zeros(5*N, (NU*N)+1);
    b_inst = zeros(5*N, 1);

    for i = 1:N
        G_i = Gamma((i-1)*NX+1:i*NX, :);
        P_i = Phi((i-1)*NX+1:i*NX, :);
        b_val = (abs(P_i(1,:)*X_k)*tan_th + 0.2);
        A_inst((i-1)*4+1:i*4, 1:NU*N) = C_los * G_i;
        A_inst((i-1)*4+1:i*4, end) = -1;
        b_inst((i-1)*4+1:i*4) = b_val - C_los*P_i*X_k;
```

```
            A_inst(4*N + i, 1:NU*N) = C_safe * G_i;
            b_inst(4*N + i) = abs(max_drop) - C_safe*P_i*X_k;
        end
    else
        H_qp = H_base;
        f_qp = f_base;
        A_inst = zeros(N, NU*N);
        b_inst = zeros(N, 1);
        for i = 1:N
            A_inst(i,:) = C_safe * Gamma((i-1)*NX+1:i*NX, :);
            b_inst(i) = abs(max_drop) - C_safe * Phi((i-1)*NX+1:i*NX, :) * X_k;
        end
    end
    F_max_c = F_max;
    % check to see if fuel is not out.
    if currImp <= 1e-4
        F_max_c = [0;0;0];
    end
    n_vars = length(f_qp);
    lb = [repmat([-F_max_c; -Tau_max], N, 1)];
    ub = [repmat([F_max_c; Tau_max], N, 1)];
    if use_los
        lb = [lb; 0];
        ub = [ub; 1e3];
    end

    [U_sol, ~, exitflag] = quadprog((H_qp+H_qp')/2, f_qp, A_inst, b_inst, [], [], lb,
ub, zeros(n_vars,1), ...

optimoptions('quadprog','Display','off','Algorithm','active-set'));
    U_optimal = U_sol;
end
```

Appendix C — MATLAB Code: Reference Trajectory Generator

```
function X_ref = generate_reference_trajectory(X_k, N, phase_mode)
  Ts = 0.2;
  theta_deg = 20;
  X_ref = zeros(13, N + 1);
  X_ref(:, 1) = X_k;
  mu = 3.986004418e14;
  Rt = 6378000 + 500; % Altitude of target orbit
  n = sqrt(mu / Rt^3);
  % Holding points for each phase
  r_target_Hom    = [500; 2000; 100];
  r_target_prox   = [200; 500; 100];
  r_target_hold   = [80.0; 2.0; 0.5];
  r_target_dock   = [10.0; 0.0; 0];

  w_stop = [0.0; 0.0; 0.0];
  % reference switch for each phase
  if phase_mode == 3
     r_goal = r_target_Hom;
     V_max = 10;
     q_aligned = [0.7071; 0.0; 0.7071; 0.0]; %align z-axis with v-bar
  elseif phase_mode == 2
     r_goal = r_target_prox;
     V_max = 2.0;
     q_aligned = [0.7071; 0.7071; 0.0; 0.0];%align z-axis with v-bar
  elseif phase_mode == 1
     r_goal = r_target_hold;
     V_max = 0.5;
     q_aligned = [1; 0.0; 0.0; 0.0];%align z-axis with cross-track, more drag :)
  else
     r_goal = r_target_dock;
     V_max = 0.3;
     q_aligned = [0.7071; 0.7071; 0.0; 0.0];%align z-axis with v-bar for docking.
  end
```

```
% Guidance Logic
r0 = X_k(1:3);
vec_to_goal = r_goal - r0;
dist = norm(vec_to_goal);
u_vec = [0;0;0];
if dist > 1e-3
    u_vec = vec_to_goal / dist;
end
% Velocity Scaling
% Braking funnel
if phase_mode == 0 || phase_mode == 1
    braking_distance = 150; % Start slowing down at 150m
elseif phase_mode == 2
    braking_distance = 300;
else
    braking_distance = 1000;
end

% Use a linear ramp for V_cmd to make deceleration more predictable
if dist > braking_distance
    V_cmd = V_max;
else
    V_cmd = V_max * (dist / braking_distance);
end

% Ensure a minimum approach speed so the chaser doesn't stall
V_cmd = max(V_cmd, 0.05);

% make v_cmd_vec a 3x1 vector pointing toward the goal
v_cmd_vec = u_vec * V_cmd;
for i = 1:N
    dt = i * Ts;

    if phase_mode == 0 || phase_mode == 1
```

```
            % STRAIGHT LINE: Predicted position moves toward goal at V_cmd
            X_ref(1:3, i+1) = r0 + (v_cmd_vec * dt);

        else
            % CW CURVE: Prediction anchored to the COMMANDED velocity, not
actual speed
            % This forces the NMPC to brake if the actual speed exceeds v_cmd_vec.
            X_ref(1, i+1) = r0(1)*cos(n*dt) + (v_cmd_vec(1)/n)*sin(n*dt) +
(2/n)*v_cmd_vec(2)*(1-cos(n*dt));
            X_ref(2, i+1) = r0(2) + (2/n)*v_cmd_vec(1)*(cos(n*dt)-1) +
(v_cmd_vec(2)/n)*(4*sin(n*dt)-3*n*dt) - 3*r0(1)*n*dt;
            X_ref(3, i+1) = r0(3)*cos(n*dt) + (v_cmd_vec(3)/n)*sin(n*dt);
            X_ref(4, i+1) = -r0(1)*n*sin(n*dt) + v_cmd_vec(1)*cos(n*dt) +
2*v_cmd_vec(2)*sin(n*dt);
            X_ref(5, i+1) = 2*v_cmd_vec(1)*-sin(n*dt) +
v_cmd_vec(2)*(4*cos(n*dt)-3) - 3*r0(1)*n;
            X_ref(6, i+1) = -r0(3)*n*sin(n*dt) + v_cmd_vec(3)*cos(n*dt);
        end
        % Reference velocity is the speed limit, not the current speed
        if phase_mode == 0 || phase_mode == 1 || phase_mode == 2
            X_ref(4:6, i+1) = v_cmd_vec;
        end
    end
    % Assemble Orientation and Damping
    if phase_mode == 0
        tan_lim = tan(deg2rad(theta_deg - 2));
        for i = 1:N
            limit = max(0.1, abs(X_ref(1, i+1)) * tan_lim);
            X_ref(2, i+1) = max(min(X_ref(2, i+1), limit), -limit);
            X_ref(3, i+1) = max(min(X_ref(3, i+1), limit), -limit);
        end
    end
    X_ref(7:10, 2:N+1) = repmat(q_aligned, 1, N);
    X_ref(11:13, 2:N+1) = repmat(w_stop, 1, N);
```

```
end
```

Appendix D — MATLAB Code: Calculate Jacobians

```
function [Ad, Bd] = calculate_jacobians(x, u, mass, params)
% mpc states and perturbation parameter
NX = 13;
NU = 6;
epsilon = 1e-9;

Ac = zeros(NX, NX);
Bc = zeros(NX, NU);
f_nom = dynamics_lvlh(x, u, mass, params);
% Perturb for Ac
for j = 1:NX
  x_p = x; x_p(j) = x_p(j) + epsilon;
  Ac(:, j) = (dynamics_lvlh(x_p, u, mass, params) - f_nom) / epsilon;
end
% Perturb for Bc
for j = 1:NU
  u_p = u; u_p(j) = u_p(j) + epsilon;
  Bc(:, j) = (dynamics_lvlh(x, u_p, mass, params) - f_nom) / epsilon;
end
% Manual Discretization
M = expm([Ac, Bc; zeros(NU, NX+NU)] * Ts);
Ad = M(1:NX, 1:NX);
Bd = M(1:NX, NX+1:end);
end
```

Appendix E — MATLAB Code: Calculating max available force and moment

```
function [F_max_LVLH, M_max_LVLH] = calculateMaxActuationLVLH(x)
%quarterions from state vector
q = x(7:10);
q_norm = norm(q);
% ensure quarterions are unit vectors and do not build up.
if q_norm > 1e-6
  q = q / q_norm;
else
  q = [1; 0; 0; 0];
end
q1 = q(1); q2 = q(2); q3 = q(3); q4 = q(4);
% rotation matrix lvlh to the body frame
R_LVLH2Body = [1 - 2*q3^2 - 2*q4^2,   2*(q2*q3 - q4*q1),   2*(q2*q4 + q3*q1);
          2*(q2*q3 + q4*q1),   1 - 2*q2^2 - 2*q4^2,   2*(q3*q4 - q2*q1);
          2*(q2*q4 - q3*q1),   2*(q3*q4 + q2*q1),   1 - 2*q2^2 - 2*q3^2];
% rotation matrix body frame to the lvlh frame
R_Body2LVLH = R_LVLH2Body';
% Geometric & Physical Constants
T_max = 0.01;
gamma = cosd(15);
beta = sind(15)*cosd(45);
alpha = cosd(30);
phi = sind(30);
% moment Arms
d = 1.4054 / 1000;
r_bar = 50 / 1000;
% Calculate Max Body-Frame Forces
Fx_max_B = (2*T_max*beta) + (T_max*alpha) + (T_max*phi);
Fy_max_B = (2*T_max*beta) + (T_max*alpha) + (T_max*phi);
Fz_max_B = (2*T_max*gamma);
if abs(Fx_max_B) > 0.4
  Fx_max_B = sign(Fx_max_B)*0.4;
end
if abs(Fy_max_B) > 0.4
```

```
  Fy_max_B = sign(Fy_max_B)*0.4;
end
if abs(Fz_max_B) > 0.4
  Fz_max_B = sign(Fz_max_B)*0.4;
end
% Assemble max body force vector
F_body = [Fx_max_B; Fy_max_B; Fz_max_B];
% Calculate Max Body-Frame Moments
Mx_max_B = (Fz_max_B * r_bar) + (Fy_max_B * d);
My_max_B = (Fz_max_B * r_bar) + (Fx_max_B * d);
Mz_max_B = (Fx_max_B + Fy_max_B) * r_bar;
M_body = [Mx_max_B+0.0059; My_max_B+0.0059; Mz_max_B+0.0050];
% Rotate the moment capability vector into the orbital frame
F_max_LVLH = abs(R_Body2LVLH * F_body);
M_max_LVLH = abs(R_Body2LVLH * M_body);
end
```

Appendix –-E: Link to plotting scripts and simulink file

Google drive link: AE295B: Ramon Medina Code Folder