

ISTANBUL TECHNICAL UNIVERSITY ★ GRADUATE SCHOOL

**BRUTE FORCE LAUNCH VEHICLE ASCENT TRAJECTORY ASSESSMENT
WITH A NOVEL VECTORIZED SIMULATOR**



M.Sc. THESIS

Ahmet Enes Yüceyurt

Department of Aeronautical and Astronautical Engineering

Aeronautical and Astronautical Engineering Programme

FEBRUARY 2024

ISTANBUL TECHNICAL UNIVERSITY ★ GRADUATE SCHOOL

**BRUTE FORCE LAUNCH VEHICLE ASCENT TRAJECTORY ASSESSMENT
WITH A NOVEL VECTORIZED SIMULATOR**



M.Sc. THESIS

**Ahmet Enes YÜCEYURT
(511201149)**

**Department of Aeronautical and Astronautical Engineering
Aeronautical and Astronautical Engineering Programme**

Thesis Advisor: Prof. Dr. Alim Rüstem ASLAN

FEBRUARY 2024

İSTANBUL TEKNİK ÜNİVERSİTESİ ★ LİSANSÜSTÜ EĞİTİM ENSTİTÜSÜ

**VEKTÖRİZE BENZETİCİ İLE FIRLATMA ARAÇLARININ YÜKSELİŞ
YÖRÜNGESİNİ KABA KUVVET DEĞERLENDİRME**

YÜKSEK LİSANS TEZİ

**Ahmet Enes Yüceyurt
(51120149)**

Uçak ve Uzay Mühendisliği Anabilim Dalı

Uçak ve Uzay Mühendisliği Programı

Tez Danışmanı: Prof. Dr. Alim Rüstem ASLAN

ŞUBAT 2024

Ahmet Enes Yüceyurt, a M.Sc. student of ITU Graduate School student ID 511201149, successfully defended the thesis entitled “BRUTE FORCE LAUNCH VEHICLE ASCENT TRAJECTORY ASSESSMENT WITH A NOVEL VECTORIZED SIMULATOR”, which he prepared after fulfilling the requirements specified in the associated legislations, before the jury whose signatures are below.

Thesis Advisor : **Prof. Dr. Alim Rüstem ASLAN**
Istanbul Technical University

Jury Members : **Assist. Prof. Dr. Cuma YARIM**
Istanbul Technical University

.....
Dr. Ahmet SOFYALI
ROKETSAN Inc.

Date of Submission : 5 January 2024
Date of Defense : 5 February 2024





To my parents and my fiancée,

FOREWORD

As I present this thesis, I am filled with deep gratitude for the numerous individuals who have guided, supported, and inspired me throughout this journey. The completion of this work stands as a testament not only to my efforts but significantly to the contributions of those who have been part of this academic endeavor.

Firstly, I want to thank my advisor Prof. Dr. Alim Rüstem ASLAN who guided me during my work. Secondly, I want to thank Dr. Ahmet Sofyalı, without him this thesis would not come to life as his guidance was crucial and his assessments were on point.

Other than my professors, I want to thank Kubilay Özçelik who supported me with his insights during my work. I want to thank Akan Selim for helping me use Datcom software and making things easier for me.

My parents, whose belief in my journey throughout my life has been a constant source of strength and motivation, I owe a big debt of gratitude.

Lastly but certainly not least, I express my deepest gratitude to my beloved fiancée Julia for the patience, love and encouragement that she provided. The faith shown in my abilities and the comfort offered during challenging times have been instrumental in reaching this milestone.

January 2024

Ahmet Enes Yüceyurt



TABLE OF CONTENTS

	<u>Page</u>
FOREWORD	viii
TABLE OF CONTENTS	X
ABBREVIATIONS	XII
SYMBOLS	XIV
LIST OF TABLES	XVII
LIST OF FIGURES	XIX
SUMMARY	XXI
ÖZET	XXIII
1. INTRODUCTION	1
1.1 Purpose and Importance	1
1.2 Overview	4
2. PROBLEM OVERVIEW & BACKGROUND	7
2.1 Physics of Ascent Flight	7
2.1.1 Rocket equation and key metrics	7
2.1.2 Acting forces	8
2.1.3 Effect of trajectory on LV performance	8
2.2 Flight Phases	12
2.2.1 First stage	14
2.2.2 Second Stage	18
2.3 Previous Research	20
2.3.1 Trajectory performance assessment	20
2.3.2 Guidance	21
2.3.3 Vectorized simulations	22
3. FLIGHT SIMULATOR & MODELS	25
3.1 Assumptions	25
3.2 Reference Coordinate Frames	26
3.2.1 ECI and ECEF frames	26
3.2.2 Local horizon frame	28
3.2.3 Wind axis frame	28
3.3 Force and Environment Models	29
3.3.1 Thrust – mass model	29
3.3.2 Aerodynamics model	30
3.3.3 Gravity model	32
3.3.4 Atmosphere model	33
3.3.5 Guidance model	35
3.4 Dynamic Equations and ODE's	38
4. VECTORIZED SIMULATION PROGRAM AND METHODOLOGY	41
4.1 Vectorization	41
4.1.1 SIMD Units	41
4.1.2 Array programming and NumPy module	43
4.1.3 Handling memory issues	50

4.2 Core Software Concept	51
4.3 OOP Structure	53
4.3.1 Scenario	53
4.3.2 Simulator.....	56
4.3.3 Stage.....	58
4.4 User Interaction	60
4.4.1 Search space input	60
4.4.2 Launch vehicle specification input	61
4.4.3 Flight data output	62
5. BRUTE FORCE LAUNCH VEHICLE ASSESMENT METHODOLOGY..	63
5.1 Exploring Trajectories	63
5.2 Assessment Methodology.....	65
5.2.1 Flight constraints.....	65
5.2.2 Approach.....	66
6. RESULTS	69
6.1 Indirect Injection: Falcon 9	69
6.1.1 Vehicle properties	69
6.1.2 Missions	73
6.1.3 Statistics	74
6.1.4 Results.....	81
6.2 Indirect Injection: Electron.....	88
6.2.1 Vehicle properties	88
6.2.2 Missions	92
6.2.3 Statistics	93
6.2.4 Results.....	99
6.3 Success of the Indirect Injection Case Studies	104
6.4 Direct Injection – Falcon 9 Super GTO Mission	105
6.5 Direct Injection: Electron LEO Mission	108
7. CONCLUSIONS	113
REFERENCES.....	115
CURRICULUM VITAE.....	119

ABBREVIATIONS

3-DOF	: Three Degree-of-Freedom
6-DOF	: Six Degree-of-Freedom
BLAS	: Basic Linear Algebra Subprograms
CD	: Coefficient of Drag
CL	: Coefficient of Lift
CONOPS	: Concept of Operations
CPU	: Central Processing Unit
CSV	: Comma Separated Values
ECEF	: Earth Centered Earth Fixed
ECI	: Earth Centered Inertial
GLOM	: Gross Liftoff Mass
GPU	: Graphics Processing Unit
GTO	: Geostationary Transfer Orbit
ISP	: Specific Impulse
JSON	: JavaScript Object Notation
LAPACK	: Linear Algebra Package
LEO	: Low Earth Orbit
LLA	: Latitude, Longitude, Altitude
LV	: Launch Vehicle
MECO	: Main Engine Cut-off
MKL	: Math Kernel Library
OOP	: Object Oriented Programming
PMF	: Propellant Mass Fraction
POST	: Program to Optimize Simulated Trajectories
RAM	: Random Access Memory
RK4	: Runge-Kutta Method
SECO	: Second Stage Engine Cut-off
SGTO	: Super Geostationary Transfer Orbit

SIMD	: Single Instruction Multiple Data
SISD	: Single Instruction Single Data
SSO	: Sun Synchronous Orbit
T	: Thrust
T/W or TW	: Thrust to Weight Ratio
UEN	: Up East North



SYMBOLS

A_{nozzle}	: Nozzle Area
C_D	: Coefficient of Drag
D_{nozzle}	: Nozzle Diameter
F_D	: Drag Force
F_{ECI}	: Force in ECI Coordinate Frame
F_G	: Gravity Force
F_T	: Thrust Force
F_{net}	: Coefficient of Drag
I_{sp}	: Specific Impulse
P_{atm}	: Atmospheric Pressure
T_{wind}	: Thrust in Wind Axes
V_{rel}	: Relative Velocity
X_{JD}	: State variables in Geodetic Coordinates
$\vec{g}$	: Gravity Vector
g_0	: Gravity at Sea level
j_2	: Geopotential Coefficient
k_1	: Runge Kutta Parameter 1
k_2	: Runge Kutta Parameter 2
k_3	: Runge Kutta Parameter 3
k_4	: Runge Kutta Parameter 4
m_0	: Initial Mass
m_p	: Propellant Mass
t_f	: Final time
λ_E	: East Longitude
Δt	: Timestep
$n_{simulation}$	: Number of simulations
$n_{timesteps}$	: Number of timesteps

v_e	: Exhaust Velocity
X_{ECI}	: ECI State Variables
DCM	: Direction Cosine Matrix
M	: Mach Number
T	: Thrust
V	: Velocity
a	: Acceleration
m	: Mass
q	: Dynamic Pressure
ΔV	: Delta v
α	: Angle of Attack
γ	: Flight Path Angle
θ	: Sidereal Time Angle
μ	: Standard Gravitational Parameter of Earth
ρ	: Density
ω	: Radial Velocity of Earth



LIST OF TABLES

	<u>Page</u>
Table 3.1 : Assumptions of the simulator.	25
Table 3.2 : An example search space for vectorized simulator.....	37
Table 5.1 : Maneuver search space.	64
Table 5.2 : Mission constraints.	65
Table 6.1 : Falcon 9 assumed properties.	70
Table 6.2 : Falcon 9 mission scenarios.	74
Table 6.3 : Falcon 9 LEO mission design space statistics.	75
Table 6.4 : Falcon 9 SGTO mission design space statistics.....	76
Table 6.5 : Falcon 9 successful LEO missions - top 20 results.....	82
Table 6.6 : Event sequence for Falcon 9 LEO mission.....	85
Table 6.7 : Falcon 9 successful SGTO missions - top 20 results.....	86
Table 6.8 : Event sequence for Falcon 9 Super GTO mission.	88
Table 6.9 : Rocket Lab Electron – assumed properties.....	89
Table 6.10 : Rocket Lab Electron missions.	92
Table 6.11 : Design space reduction Electron SSO mission.	93
Table 6.12 : Design space reduction Electron LEO mission.....	94
Table 6.13 : Electron SSO mission event sequence.	101
Table 6.14 : Electron LEO mission event sequence.....	101
Table 6.15 : Electron SSO mission top 20 results.....	102
Table 6.16 : Electron LEO mission top 20 results.	103
Table 6.17 : Payload and mission success summaries.	104
Table 6.18 : SGTO direct injection mission.....	105
Table 6.19 : Falcon 9 SGTO direct injection mission success.....	108
Table 6.20 : Falcon 9 SGTO direct injection orbital elements.....	108
Table 6.21 : Rocket Lab Electron LEO direct injection mission.	109
Table 6.22 : Electron direct injection mission success.	112
Table 6.23 : Electron direct injection orbital elements.	112



LIST OF FIGURES

	<u>Page</u>
Figure 1.1: Increase in investment for space companies (Edberg & Costa, 2022).	1
Figure 1.2 : Description of Program to Optimize Simulated Trajectories (POST) (Brauer, Cornick, & Stevenson, 1977).	2
Figure 1.3 : Importance of trajectory assessment on preliminary design cycle of launch vehicles (Edberg & Costa, 2022).	3
Figure 2.1 : Gravity loss thrust vector.	10
Figure 2.2 : Steering loss visualization.	11
Figure 2.3 : An example CONOPS, Flight phases of Korea Space Launch Vehicle-I (Sun, Park, & Rae Roh, 2010).	12
Figure 2.4 : Event sequence of Saturn V Apollo 12 mission (NASA, 1969).	13
Figure 2.5 : Space Shuttle performing roll and pitch maneuver after lift-off (Shuttlesource, 2023).	16
Figure 2.6 : Gravity turn vector visualization.	17
Figure 2.7 : Demonstration of a traditional fairing jettison (ESA, 2019).	19
Figure 3.1 - ECI and ECEF frames (Chobotov, 2002).	27
Figure 3.2 : Local horizon frame (Tewari, 2007).	28
Figure 3.3 : An example CD graph vs Mach (Missile DATCOM).	31
Figure 3.4 : Spherical harmonics of an axisymmetric planet (Tewari, 2007).	33
Figure 3.5 : Atmosphere parameters by altitude.	34
Figure 3.6 : Sinusoidal profiles pitch-over maneuvers.	36
Figure 3.7 : RK4 method.	40
Figure 4.1 : Working mechanism of and SIMD unit.	42
Figure 4.2 : SISD vs SIMD.	43
Figure 4.3 : Usage of Numpy in black hole imagery (NumPy, 2019).	44
Figure 4.4 : Basic vectorized operations in NumPy (Harris, et al., 2020).	46
Figure 4.5 : Speed comparison of if vs Boolean masks.	47
Figure 4.6 : Demonstration of 10x speed improvement for the dot product.	48
Figure 4.7 : Demonstration of 450x speed improvement for algebraic expressions.	49
Figure 4.8 : Propagation of state variables, written in their native Python array notation.	53
Figure 4.9 : Variable checking flags for sanity check.	54
Figure 4.10 : Scenario compilation function.	55
Figure 4.11 : Code snippet from guidance scenario compiler.	56
Figure 4.12 : Code snippet from Simulator object.	57
Figure 4.13 : Data arranging code snippet from Simulator object.	58
Figure 4.14 : Code snippet from Stage object initialization.	59
Figure 4.15 : Code snippet depicting state flags.	60
Figure 4.16 : Helper spreadsheet for specifying the search space.	61
Figure 4.17 : Snippet from a Scenario JSON file.	62

Figure 5.1 : Optimum thrust direction in Saturn V vacuum flight (Haeussermann, 1970).	64
Figure 5.2 : Flow Diagram of the vectorized simulator.	68
Figure 6.1 : Simple Falcon 9 geometry for aerodynamics.	71
Figure 6.2 : Falcon 9 reference geometry model comparison.	72
Figure 6.3 : Calculated drag coefficient of Falcon 9.	73
Figure 6.4 : Falcon 9 LEO mission first phase design space.	78
Figure 6.5 : Falcon 9 LEO mission final phase design space.	79
Figure 6.6 : Falcon 9 SGTO mission first phase design space.	80
Figure 6.7 : Falcon 9 SGTO mission last phase design space.	81
Figure 6.8 : Plots of the most successful flight.	83
Figure 6.9 : Plots of the most successful flight.	87
Figure 6.10 : Geometric comparison of reference model.	91
Figure 6.11 : CD of Electron obtained via Missile DATCOM.	92
Figure 6.12 : Electron SSO mission first phase design space.	95
Figure 6.13 : Electron SSO mission final phase design space.	96
Figure 6.14 : Electron LEO mission first phase design space.	97
Figure 6.15 : Electron LEO mission final phase design space.	98
Figure 6.16 : Electron SSO mission plots of the best flight.	99
Figure 6.17 : Electron LEO mission plots of the best flight.	100
Figure 6.18 : Falcon 9 initial trajectory from Cape Canaveral.	106
Figure 6.19 : Falcon 9 boost trajectory to orbit.	106
Figure 6.20 : Falcon 9 SGTO ground track.	107
Figure 6.21 : Electron trajectory after lift-off.	109
Figure 6.22 : Electron boost trajectory to orbit.	110
Figure 6.23 : Rocket Lab Electron LEO ground track.	111

BRUTE FORCE LAUNCH VEHICLE ASCENT TRAJECTORY ASSESSMENT WITH A NOVEL VECTORIZED SIMULATOR

SUMMARY

This thesis is developed to contribute a preliminary design tool for launch vehicles, in the context of trajectory performance assessment. The launch vehicle market is witnessing substantial growth, driven by the increasing demand for satellite deployment and space exploration. In the current era of the space race, the strategic importance for a nation to build their own launch vehicle is being increasingly acknowledged. This trend is not merely about strengthening national autonomy in space; it represents an opportunity for these countries to leap forward in terms of technological and scientific progress.

The need for tools like the vectorized simulator presented in this thesis becomes clear when considering the unique challenges faced by smaller countries in space technology. Developing launch vehicles is a complex, resource-intensive endeavor, often constrained by budget and technological limitations. Therefore, efficient and sophisticated design tools are essential. The vectorized simulator offers a faster, more efficient way to explore a wide range of design options and trajectory paths, enabling the companies to optimize their resources and accelerate the development process. By facilitating rapid iteration and refinement of vehicle designs, this tool plays a critical role in making preliminary design phase of launch vehicles faster.

One crucial component for the development of launch vehicles is the concept of trajectory optimization. This process is crucial for ensuring efficient, cost-effective, and successful missions. Optimal trajectories reduce fuel consumption, maximize payload capacity, and ensure stringent safety and regulatory standards. As smaller nations get more involved into space, the ability to optimize launch trajectories becomes a key factor in their space program's feasibility and success.

Recognizing the need for advanced tools in this area, thesis introduces an innovative vectorized simulation program designed to have capabilities of evaluating a vast number of trajectories for finding an optimal trajectory with the mission constraints just like the traditional Program to Optimize Simulated Trajectories (POST). While POST is an instrumental tool, it is only open to use for US Defence related entities. While POST is much more advanced and complex, since it has been written and developed for over 50 years, this new tool also address the evolving complexity and computational demands of modern launch vehicle designs. The scope of the thesis encompasses not only the development of this tool but also its application in rapidly assessing a wide range of launch vehicle performance parameters.

Vectorization, as a computational strategy, involves processing multiple data points simultaneously rather than executing individual operations sequentially. This technique has its roots in early supercomputers, where it was used to accelerate large-scale scientific calculations. Its rise in popularity is closely linked to the development

of modern CPUs and GPUs that feature vector processing capabilities. Equipped with SIMD (Single Instruction, Multiple Data) architectures, these processors can perform identical operations on several data points at once, dramatically enhancing speed and efficiency. This approach is particularly beneficial in domains with heavy computational demands, such as data analysis, machine learning, and scientific simulations, and has become increasingly vital in the age of big data and complex algorithmic challenges.

The vectorized simulation program central to this thesis uses this advancement in computational technology. By harnessing the latest in computational techniques and the capabilities of modern hardware, this program significantly outpaces traditional simulation methods in speed. This accelerated processing capability is critical for exploring a wide number of trajectories rapidly, therefore allowing vehicle designers to iterate and refine their designs with greater efficiency. In an industry where the pace of innovation and the ability to quickly adapt to evolving mission requirements are key competitive factors, the speed offered by fast trajectory simulations becomes invaluable. This program not only exemplifies the power of vectorization in computational tasks but also highlights its practical application in the dynamic field of vehicle design and trajectory optimization.

In this thesis, the initial focus will be on defining the problem at hand and delving into the mathematical principles under it. We will then proceed to dissect the various flight phases and the constraints associated with them. Following this, our attention will shift to the foundational physical models and reference frames that form the basis of our study. A significant portion of the thesis is dedicated to the primary innovation: the vectorized simulator. This discussion will cover the concept of vectorization, its advantages, and how it compares in terms of speed with traditional methods. We'll explore the methods of vectorized array programming and then transition into discussing the structure of the program's objects and models.

After the section discussing the software foundations of the study, the post-processing algorithm that works using the outputs of the software will be discussed. The post-processing algorithm is a vectorized software like a simulator and operates separately from the simulator. After obtaining multiple flight data, this algorithm examines the outputs of thousands of simulations at once, identifies the desired orbits after the necessary flight restrictions, and makes the final assessment for the user. Following this, it creates a spreadsheet summarizing the performance of all simulations for the user.

The concluding section of the thesis encompasses a detailed examination of six distinct case studies, all of which utilize the capabilities of the vectorized simulator alongside the post-processing algorithm. Specifically, the analysis will cover two missions involving transfer orbit injections and one mission focusing on direct orbit injection, all based on the Falcon 9 vehicle. Similarly, for the Rocket Lab Electron vehicle, two missions related to transfer orbit injections and one mission concerning direct orbit injection will be scrutinized. This portion of the thesis will delve into the application of the two-stage approach integrated within the program, assessing its effectiveness and outcomes when applied to these mentioned vehicles, thereby providing a comprehensive overview of its practical implications.

VEKTÖRİZE BENZETİCİ İLE FIRLATMA ARAÇLARININ YÜKSELİŞ YÖRÜNGESİNİ KABA KUVVET DEĞERLENDİRME

ÖZET

Fırlatma hizmetlerine olan talep, uydu teknolojilerinin kullanım alanlarının artması ile birlikte son zamanlarda çok hızlı bir şekilde artmaktadır. Artan bu talebi fark eden özel sektör firmaları da ekonomik değer üretmek için bu alanda artan bir şekilde yer almaya başlamıştır. SpaceX, RocketLab ve Firefly Aerospace gibi özel firmaların fırlatma sektöründe elde ettikleri başarı, diğer girişimcilerin de bu alanda faaliyet gösterme isteğini arttırmıştır. Mevcut durumda fırlatma hizmeti sağlayan firmaların, uydu fırlatmak için oluşan talebi karşılayamaması sebebiyle oluşan arz talep dengesizliği piyasaya yeni çıkacak fırlatma araçları için hazır müşteri portföyü oluşturmaktadır. Sürekli büyüyen uzay ekonomisi birçok ülke için fırlatma aracı geliştirme projelerine yönelik olarak katalizör etkisi yaratmıştır. Hem devletler hem de yüksek teknoloji alanında yatırım yapmaya hevesli olan özel girişim firmaları, artan bu uzay ekonomisi pastasından payını almak için çeşitli fırlatma aracı geliştirme projeleri başlatmaktadır. Yeni teknolojik trendlerin yakalanabilmesi açısından da artan uzay çalışmalarında yer alabilmek birçok ülke, firma ve girişimciler için de önem kazanmıştır.

Ülkeler için yörüngeye erişim kabiliyeti, politik ve ekonomik olarak da büyük avantajlar sağlamaktadır. Çok az ülkede yörüngeye ulaşma kabiliyeti bulunduğu için kabiliyeti bulunmayan diğer ülkelerin uydularını yörüngeye yerleştirmesi için çoğu durumda bu ülkelerle siyasi ve ekonomik olarak güçlü ilişkileri bulunması gerekmektedir. Birçok durumda fırlatma kabiliyeti bulunan ülkeler diğer ülkelerin istihbarat kapasitesini arttıracak uydularını yörüngeye yerleştirmeyi tercih etmemektedir. Bu durum fırlatma aracına sahip olmayan ulusların uzaydaki bağımsızlığını tehdit ettiği için birçok ulus kendi fırlatma aracı programını başlatmaktadır. Fırlatma aracı olan ülkeler kendi uydularını diğer ülkelerin kısıtlaması olmadan özgürce yörüngeye yerleştirerek uzay tabanlı teknolojilerden daha verimli olarak yararlanabilmektedir. Kendi fırlatma aracı olan ülkeler, uzay tabanlı teknolojiler açısından, ekonomik, askeri ve bilimsel ilerleme olarak önemli bir avantaj elde etmektedir. Özellikle uydu teknolojilerinin ve kabiliyetlerin her geçen gün arttığı günümüz Dünya'sında uzay konusunda söz sahibi olabilmek güçlü ülkeler için oldukça kritik olmaktadır. Ayrıca son zamanlarda tekrardan Ay'a erişim için başlayan uzay yarışının da fırlatma sektörü üzerinde, fırlatma talebini arttırıcı yönde olumlu etkilerinin olması beklenmektedir.

Bu tezin sunduğu vektörize benzetici gibi araçların gerekliliği, yörüngeye erişim yeteneği olmayan ülkelerin uzay teknolojisi geliştirme yolunda karşılaştıkları benzersiz zorluklar göz önünde bulundurulduğunda, açıkça ortaya çıkmaktadır. Fırlatma araçlarını geliştirmek, kompleks ve kaynak tüketici yoğun bir çaba gerektirip, genellikle bütçesel ve teknolojik anlamda sınırlamalarla kısıtlanmaktadır. Bu nedenle fırlatma araçlarının kavramsal tasarım süreçlerinde etkin ve sofistike araçlar kullanmak hayati öneme sahiptir. Kavramsal tasarım aşamasında maliyet etkin

çözümler üretebilmek ve mevcut altyapılarla üretimin ve erişilebilirliğin sağlanabileceği şekilde optimize edilmiş araçların çıkartılması proje bütçesi üzerinde önemli etkilerde bulunmaktadır. Fırlatma araçlarının alt sistemlerinin etkin bir şekilde optimize edilebilmesi için uçuş performansı ile birlikte yörüngeye çıkartılabileceği faydalı yük miktarının hızlı bir şekilde belirenebilmesi oldukça önemlidir. Böylelikle tasarımların erken aşamalarında aracın faydasını arttıracak kritik tasarım kararlarının alınmasını kolaylaştıracaktır. Vektörize benzetici, fırlatma aracı yükseliş uçuşunda manevra uzayını taramak için hızlı ve etkin bir yol sunmakta, böylece şirketlerin kaynaklarını optimize etmelerine ve geliştirme sürecini hızlandırmalarına olanak tanıyabilmektedir. Bahsekonu program, fırlatma aracı tasarım alternatiflerinin hızlı bir şekilde performanslarının değerlendirilmesini ve rafine edilmesini kolaylaştırarak tasarım iterasyonunun otonomlaştırılmasında önemli bir yere sahiptir.

Fırlatma araçlarının tasarım aşamasının kritik yapıtaşlardan biri yörünge optimizasyonudur. Bu süreç, etkili, maliyet-etkin ve başarılı görevlerin gerçekleşmesi için büyük öneme sahiptir. Optimal yörüngeler, yakıt tüketimini azaltıp, görev yükü kapasitesini maksimize eder ve aynı zamanda uçuş boyunca güvenlik standartlarını sağlamada rol oynar. Fırlatma aracının alt sistemleri ile birlikte optimizasyon iterasyonuna girildiğinde çok yüksek sayıda yörünge simülasyonu ve optimizasyonunun da birlikte yapılması gerekmektedir. Bu teze bahsekonu olan yazılım, bu alanda ileri düzey araçlara olan ihtiyaç gözetilerek geliştirilmiştir. NASA tarafından geliştirilmiş olan “Simüle Edilmiş Yörüngeleri Optimize Etme Programı”na (POST) benzer şekilde, bir görevin kısıtları dahilinde optimal bir yörünge bulmak için çok sayıda yörüngeyi değerlendirme kapasitesine sahip yenilikçi bir yaklaşım uygulanmıştır. POST, ABD içerisindeki havacılık uzay şirketleri arasında yaklaşık 50 senedir yükseliş uçuşu yörünge optimizasyonu için kullanılmakta olup, yalnızca ABD içerisindeki havacılık uzay kurumları için kullanıma açık olan kapalı kaynak bir yazılımdır. 50 yıla yakın bir süredir yazılıp geliştirilmesi nedeniyle çok daha ileri düzey ve karmaşık olsa da, bu yeni araç da modern fırlatma aracı tasarımlarının gelişen karmaşıklığına ve hesaplama taleplerine hitap etmektedir. Ancak tabii ki de tam donanımlı bir yörünge optimizatörü olması için yazılımın çok daha fazla geliştirmeye ihtiyacının olduğu söylenebilir. Tezin kapsamı, bu aracın geliştirilmesini ve fırlatma aracı yörünge performans parametrelerinin geniş bir yelpazede hızlı bir şekilde değerlendirilmesini içermektedir.

Teze konu olan benzetim yazılımının arkasındaki temel programlama metodu olan vektörizasyon, hesaplamalar için döngüler kullanmak yerine, tüm dizileri veya büyük veri kümelerini aynı anda işlemeyi sağlayan bir programlama yöntemidir. Bu teknik, büyük ölçekli bilimsel hesaplamaları hızlandırmak için kullanıldığı erken dönem süper bilgisayarlarda kök salmıştır. Popülerliğinin artışı, vektör işleme yeteneklerine sahip modern CPU'lar ve GPU'ların geliştirilmesiyle yakından bağlantılıdır. SIMD (Tek Talimat, Çoklu Veri, İng. Single Instruction Multiple Data) mimarileriyle donatılmış bu işlemciler, birçok veri noktasında aynı işlemleri aynı anda gerçekleştirerek hızı ve verimliliği dramatik bir şekilde artırabilir. Bu yaklaşım, özellikle veri analizi, hesaplamalı akışkanlar dinamiği, makine öğrenimi ve bilimsel simülasyonlar gibi yoğun hesaplama gerektiren alanlarda olmazsa olmaz bir kavram olarak önümüze çıkmaktadır. Benzer şekilde yörüngeye ulaşmakla ilgili yapılan uçuş benzetimleri de ciddi bilgisayar gücü kullanmaktadır. Vektörizasyonla birlikte bilgisayarların verimli olan hesaplama kabiliyeti yörünge benzetim hızını arttırarak uçuşu optimize etmeyi kolaylaştırmaktadır. Tez kapsamında alçak dünya yörüngelerine erişim konusunda çalışmalar gerçekleştirilse de bu yöntemin Ay'a erişmek ve diğer gezegenlere erişmek

için gerekli yörünge benzetimlerini koşturmak için de uygulanabileceği düşünülmektedir.

Bu tezin merkezinde yer alan vektörize benzetim programı, bu hesaplama teknolojisindeki ilerlemeyi lehine kullanmaktadır. Sürekli geliştirilmekte olan yeni işlemci ve ekran kartlarının hedefinde olan, vektörleri daha hızlı ve verimli bir şekilde çözmesinden faydalanmaktadır. Dolayısıyla işlemci ve ekran kartlarının vektörleri daha hızlı çözebilmesi bu tezde kullanılan vektörize benzetimin de daha hızlanmasını sağlayacaktır. Vektörize benzetim programı modern donanımın azami kapasitelerini zorlayarak ve en son hesaplama tekniklerini kullanarak, geleneksel simülasyon yöntemlerini hız açısından büyük ölçüde geride bırakmaktadır. Bu hızlandırılmış işleme yeteneği, çok sayıda yörüngeyi hızla inceleyebilmek için kritik öneme sahiptir, bu da araç tasarımcılarının tasarımlarını daha verimli bir şekilde yinelemelerine ve geliştirmelerine olanak tanır. İnovasyon hızı ve evrilen görev gereksinimlerine hızla uyum sağlama yeteneğinin ana rekabet faktörleri olduğu bir endüstride, hızlı yörünge benzetimi yapabilme yeteneği oldukça önemlidir. Ayrıca hem fırlatma aracı hem de yörünge benzetimleri üzerinde yapılacak olan belirsizlik ve hassasiyet analizi çalışmaları için de bu yöntemin hız avantajı sağlayacağı düşünülmektedir.

Çalışmada, ilk olarak yükselme uçuşu problemi ele alınacak ve arkasında yatan matematiksel prensiplere değinilecek, daha sonrasında ise çeşitli uçuş aşamalarının ve bunlarla ilişkili kısıtların detaylı bir şekilde incelemesi üzerine devam edecektir. Bunu takiben çalışmada kullanılan itki, aerodinamik, yer çekimi, atmosfer gibi fiziksel modeller ve güdüm gibi tasarım modellerinden bahsedilecektir. Yörünge ilerletmesinde ve kuvvetlerin hesaplanmasında kullanılan referans koordinat sistem takımları açıklanacaktır. Tezin önemli bir bölümü, tezdeki ana inovasyon faktörü olan vektörize benzetici üzerine odaklanmaktadır. Vektörizasyon kavramı, avantajları, dezavantajları ve geleneksel yöntemlerden nasıl ayrıştığından bahsedilecek ve hız kazancı birkaç örnek ile belgelenecektir. Bunu takiben vektörize dizi programlama ve yöntemlerinden bahsedilecek ve ardından bahse konu yazılımın bu metotları nasıl kullandığının ve verilerin nesne odaklı programlama ile nasıl daha verimli halde benzetici içerisinde aktarılabilmesinin üzerinde durulacaktır. Benzetici yarı nesne odaklı bir yaklaşımla yazıldığı için bir kısmı sınıflardan (İng. class), bir kısmı fonksiyonlardan oluşmaktadır. Bu sınıfların işlevlerinden ve çalışma prensiplerinden bahsedilecektir.

Çalışmanın yazılımsal temellerinden bahsedilen bölümün sonrasında ise yazılımın çıktılarını kullanarak çalışan son işleme algoritmasından bahsedilecektir. Son işleme algoritması, benzetici gibi vektörize çalışan bir yazılım olup benzeticiden ayrı olarak çalışmaktadır. Çoklu uçuş verileri elde edildikten sonra bu algoritma binlerce benzetimin çıktılarını tek seferde inceleyerek gerekli uçuş kısıtlarından sonra istenen yörüngeleri tespit eder ve kullanıcı için son değerlendirmeyi yapar. Bunun üzerine kullanıcıya bütün benzetimlerinin performanslarının özetlendiği bir excel döküm dosyası oluşturur.

Tezin en son kısmındaysa, vektörize benzetici ve son işleme algoritması kullanılarak yapılan 6 adet vaka çalışması yer almaktadır. Falcon 9 aracı baz alınarak yapılan 2 adet transfer yörüngeli yörüngeye yerleşme ve 1 adet doğrudan yörüngeye yerleşme, Rocket Lab Electron aracı baz alınarak yapılan 2 adet transfer yörüngeli yerleşme ve 1 adet doğrudan yerleşme görevleri incelenecektir. Bu kısımda programda mevcut olan 2 aşamalı yaklaşımın nasıl uygulandığı ve bu araçlar üzerindeki sonuçları incelenecektir.



1. INTRODUCTION

1.1 Purpose and Importance

The launch vehicle market is currently experiencing a rise in demand, driven by the trend towards deploying mini and microsatellites and reduced costs of reaching space and advancement of technology. **Figure 1.1** shows how drastically the total investments into space related companies increase year by year since 2012.

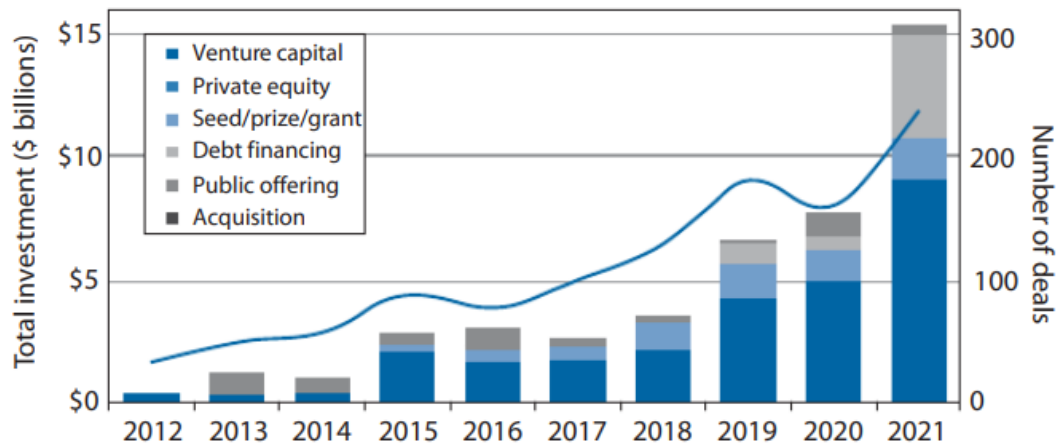


Figure 1.1: Increase in investment for space companies (Edberg & Costa, 2022).

This trend is also significant for smaller countries seeking to establish a presence in space. Building national launch capability allows these nations to have direct control over their space assets, which is crucial for national security, economic, and scientific purposes. It empowers them with independence in satellite deployment, avoiding reliance on international launch service providers. Consequently, there is a growing emphasis on the development of efficient launch vehicles that can serve to the unique demands of launching satellites, where the balance between cost-effectiveness and reliability is very important.

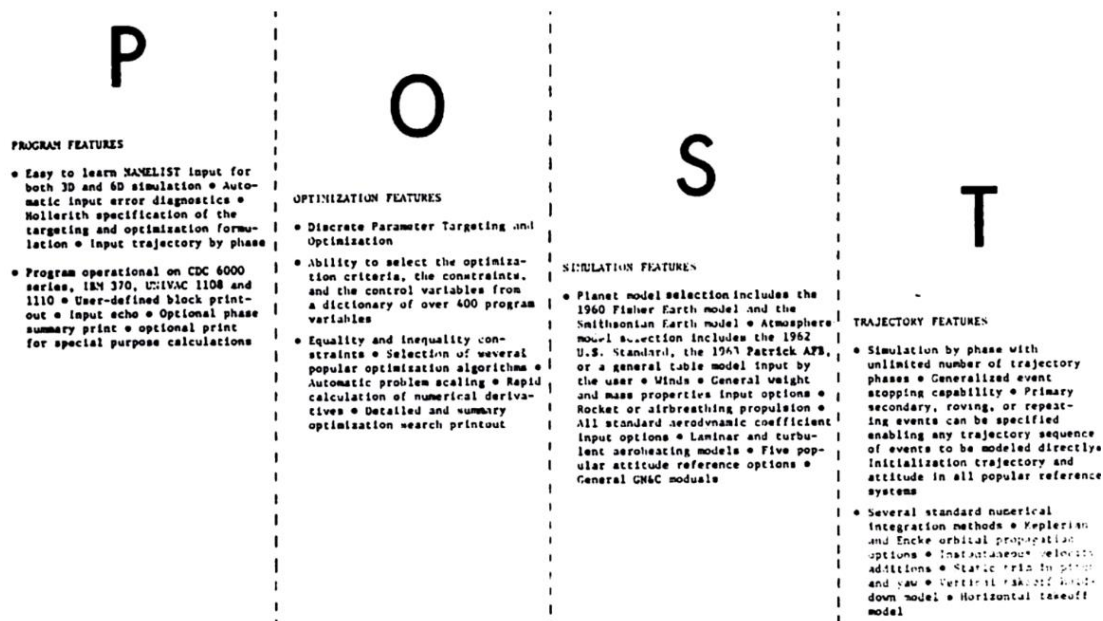


Figure 1.2 : Description of Program to Optimize Simulated Trajectories (POST)
(Brauer, Cornick, & Stevenson, 1977).

In the context of designing an efficient launch vehicle, the importance of precise trajectory assessment cannot be overstated. The initial stages of launch vehicle design are inextricably linked to its eventual trajectory and mission profile. NASA's Program for Optimizing Simulated Trajectories (POST) (Brauer, Cornick, & Stevenson, 1977) seen in **Figure 1.2** which was developed in the 70's fulfills this task of preliminary trajectory optimization for launch vehicles for fast assessment of a design configuration. With this tool, the teams can iterate on different configurations and design choices to find a suitable configuration that meets the specific requirements from the contractor. POST, however, is only in use for the US government related entities. Therefore, it is crucial for the other countries to build their own specific tools for this purpose. As such, sophisticated trajectory analysis tools, like the vectorized simulator discussed in this thesis, serves a great purpose and a small step in the way to designing a launch vehicle. It is designed as a means to rapidly evaluate wide range of trajectory options with flight and mission constraints to assess the performance of a preliminary design concept. This can include considerations like fuel efficiency, payload mass optimization etc.

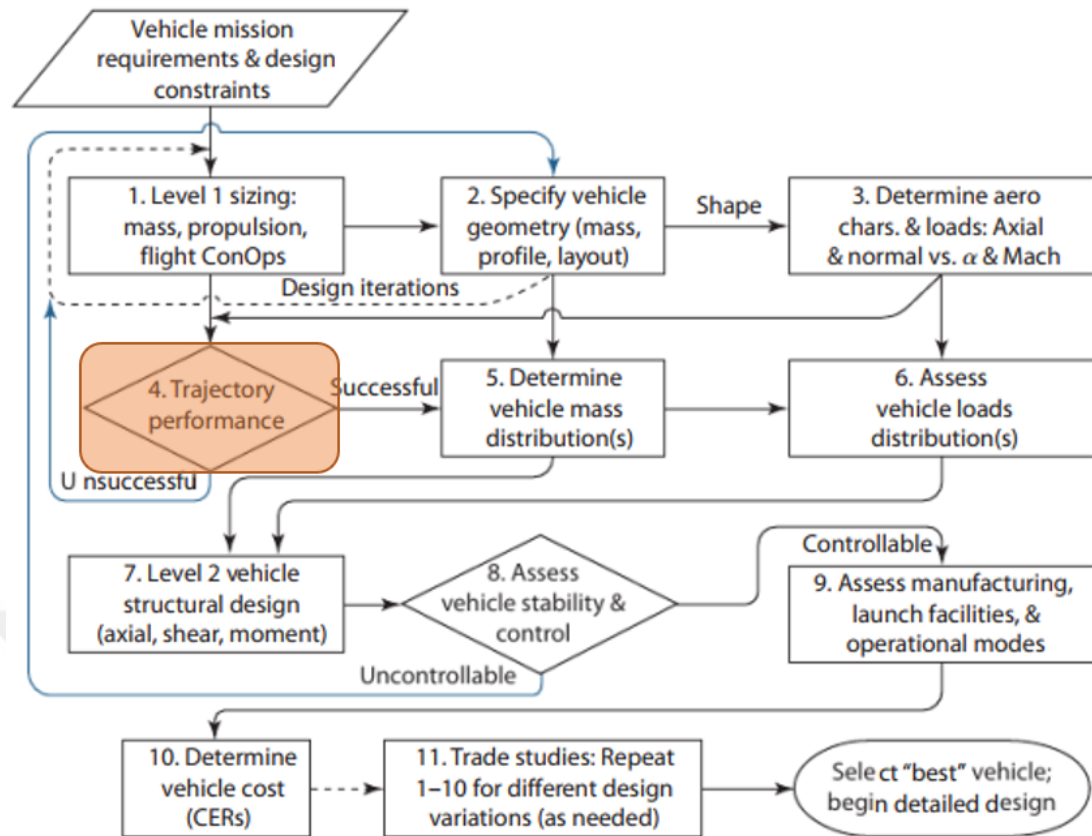


Figure 1.3 : Importance of trajectory assessment on preliminary design cycle of launch vehicles (Edberg & Costa, 2022).

Figure 1.3 emphasizes the iterative process of preliminary launch vehicle design, emphasizing the pivot role of trajectory performance assessment. This step is situated at a crucial juncture in the design cycle, following the initial sizing and specification of the vehicle's geometry. The assessment of trajectory performance acts as a gatekeeper to the subsequent design iterations; if the trajectory is deemed successful, the process advances to refine the vehicle's mass distribution and aerodynamic loads. If not, the design reverts to level 1, involving structural design adjustments.

Trajectory performance assessment is critical because it directly impacts the vehicle's ability to fulfill its mission requirements within the constraints of its design. It is the trajectory that determines the vehicle's environmental conditions throughout the flight, from atmospheric pressures and temperatures to dynamic loads that affect structural integrity and stability. An efficient trajectory can reduce fuel consumption, increase payload capacity, and optimize the use of onboard resources, thereby enhancing the overall mission success and economic viability.

In the preliminary design phase, trajectory assessment provides early validation of the vehicle's performance capabilities. It informs designers whether the vehicle, as conceived, can achieve the intended orbit and sustain the loads encountered during ascent. This step makes sure that the design does not proceed on a flawed trajectory, which could lead to costly redesigns or mission failure. By integrating trajectory assessment early and iteratively, potential issues can be identified, thereby streamlining the development process and paving the way for more advanced studies like stability, control, and cost analysis.

In the context of the thesis, trajectory performance assessment using the vectorized simulation approach is not just a step in the design cycle; it is a continuous thread that influences every aspect of the vehicle's design. The ability to rapidly simulate and optimize trajectories allows for a more agile and informed design process, reflecting the modern demands of the launch vehicle industry where efficiency, adaptability, and precision are important.

1.2 Overview

This thesis introduces a vectorized simulation program, conceived as a response to the growing need for fast tools in launch vehicle trajectory optimization. Program is adept at evaluating a multitude of trajectories to pinpoint the near-optimal path within given mission constraints, a task traditionally undertaken by the Program to Optimize Simulated Trajectories (POST). Although POST is a sophisticated and established tool, its accessibility is limited to U.S. Defense entities. In contrast, the newly developed tool addresses both the rising complexity and the increasing computational demands of contemporary launch vehicle designs. The thesis encompasses the development and application of this tool, focusing on assessing a broad spectrum of launch vehicle trajectory performance parameters.

Vectorization, the core computational strategy of this program, allows for the simultaneous processing of multiple data points, contrasting with the sequential execution typical in traditional computing. This method has evolved alongside advancements in CPU and GPU technologies, especially those equipped with SIMD (Single Instruction, Multiple Data) architectures. Such architectures enable the execution of the same operation on multiple data points concurrently, leading to

significant improvements in speed and efficiency. Vectorization is particularly advantageous in fields that require extensive computational resources, like data analysis, machine learning, and scientific simulations, and has become increasingly crucial in the modern era of complex computational challenges.

The central vectorized simulation program of this thesis leverages these advancements in computing technology. By utilizing state-of-the-art computational techniques and using the power of modern hardware, the program achieves speeds far surpassing those of conventional simulation methods. This rapid processing is essential for quickly exploring a vast array of trajectories, thus enabling designers to refine their vehicle designs more effectively. In the fast-paced and innovation-driven aerospace industry, the ability to rapidly adapt to changing mission requirements is a crucial competitive edge, making the speed offered by fast trajectory simulations invaluable. This program not only demonstrates the effectiveness of vectorization in computational tasks but also its practical application in the dynamic world of vehicle design and trajectory optimization.

In this thesis, the initial chapters will define the problem and explore the mathematical foundations underpinning it. The subsequent sections will dissect various flight phases and associated constraints. The thesis will then delve into the foundational physical models and reference frames essential to the study. A significant portion is dedicated to discussing the vectorized simulator, breaking down the concept of vectorization, its advantages, and a comparison of its speed against traditional methods. The thesis will also explore the methods of vectorized array programming and detail the structure of the program's classes and models.

The methodology developed for trajectory assessment will be elaborated, detailing the simulator's capability to simulate vast numbers of flight corridors, select optimal trajectories, and integrate various approaches. The thesis will conclude with an analysis of six case studies using two staged assessment approaches, where the findings will be presented and discussed.



2. PROBLEM OVERVIEW & BACKGROUND

2.1 Physics of Ascent Flight

The ascent flight of a launch vehicle is a complex interplay of forces, energy trade-offs, and maneuvering, all governed by the fundamental laws of physics. Understanding these dynamics is crucial for determining the final orbit and the successful deployment of payloads into space.

2.1.1 Rocket equation and key metrics

The primary performance indicator for launch vehicles or spacecraft is their "change in velocity," or ΔV , a measure of the potential velocity increase capacity achievable by the vehicle. The Tsiolkovsky rocket equation, which factors in the mass ratio and the exhaust velocity of the propellants, offers a simplified way to describe this parameter. The mass ratio is defined as the initial mass of the launch vehicle divided by its final mass, with higher ratios indicating a greater proportion of the rocket's mass is active and not just dead weight. This formula is relevant for both multi-stage and single-stage launch vehicles. It is important to note that if the vehicle is multistage, ΔV capacity of the vehicle should be calculated stage by stage and added together.

$$\Delta V = v_e \ln \frac{m_0}{m_f} \quad (2.1)$$

Here, v_e represents the propellant's exhaust velocity. Typically, however, the specific impulse (I_{sp}), which is a measure of the efficiency of an engine in producing thrust per unit mass of propellant, is used as a more practical metric than exhaust velocity.

$$\Delta V = I_{sp} g_0 \ln \frac{m_0}{m_f} \quad (2.2)$$

Another commonly used term is the propellant mass fraction (PMF), calculated as:

$$\text{PMF} = \zeta = \frac{m_p}{m_0} \quad (2.3)$$

In this equation, m_p is the mass of the propellant used to generate thrust, and m_0 is the initial mass of the vehicle. Essentially, PMF indicates what percentage of the vehicle's mass is dedicated to producing thrust. Both propellant mass fraction and specific impulse are crucial factors in designing launch vehicles.

2.1.2 Acting forces

The flight of a launch vehicle from the ground into space is influenced by several key forces. Primarily, thrust, generated by the vehicle's engines, propels it upwards. This force is essential for overcoming the pull of gravity, which acts downwards, continuously opposing the ascent. As the vehicle moves through Earth's atmosphere, it also encounters aerodynamic drag – resistance from air molecules, which creates an opposing force that must be overcome for successful ascent. Simplified force balance can be seen in the below equation.

$$\mathbf{F}_{net} = \mathbf{F}_T - \mathbf{F}_G - \mathbf{F}_D = m\mathbf{a} \quad (2.4)$$

Detailed examination of all these forces can be seen below in the force and environment models' chapter.

2.1.3 Effect of trajectory on LV performance

The concept of a delta-v budget is crucial to understanding and planning the ascent phase of a launch vehicle. As mentioned before, Delta-v, denoted as ΔV , represents the change in velocity required for a spacecraft to accomplish its mission objectives, including reaching the desired orbit. Delta-v budget is the breakdown of different delta-v requirements for the mission. This includes the velocity needed for target orbit, gravity loss, drag loss, steering loss and propulsion losses. Designing an efficient mission is related to minimize the losses and maximize the gains. Delta-v budget for ascent flight can be seen below.

$$\Delta V_{vehicle} = \Delta V_{orbit} + \Delta V_{gravity\ loss} + \Delta V_{drag\ loss} + \Delta V_{steering\ loss} + \Delta V_{propulsion\ loss} + \Delta V_{launch\ site} \quad (2.5)$$

Gravity loss refers to the velocity expended to counteract Earth's gravitational pull during ascent, effectively the work done against gravity. Drag loss, on the other hand, is the energy used to overcome atmospheric resistance. Steering loss happens when the thrust direction is not aligned with the flight path during maneuvering. Propulsion loss occurs when the rocket engine's exhaust doesn't expand optimally due to atmospheric pressure while flying through the atmosphere, which causes thrust to decrease when the ambient pressure increases. Additionally, the Earth's rotation plays a role as well. Launching from a location with a latitude similar to the target orbital inclination yields a velocity gain due to Earth's eastward rotation. On the other hand, launching into retrograde or polar orbits can result in a velocity loss, as the vehicle will have to counteract the Earth's eastward rotation.

Estimating the various losses involved in a launch vehicle's trajectory analytically is quite difficult, primarily because these losses are interconnected and affect each other. While there are some correlations available, they are not in the scope of this thesis.

A numerical calculation of gravity loss is given below. It's obvious from the equation and **Figure 2.1** that the flight path angle θ is a key factor, as it directly influences the magnitude of gravity loss. The smaller the flight path angle, the less the gravity loss, suggesting that the most efficient path to orbit for a launch vehicle would theoretically be a horizontal flight relative to Earth's surface. However, this is impractical due to atmospheric effects. In conclusion, it shows that any thrust applied in the opposite direction of the gravity will result in a loss for gaining horizontal velocity.

$$\Delta V_{gravity\ loss} = \int_{t_0}^{t_f} G(t) \sin(\gamma) dt \quad (2.6)$$

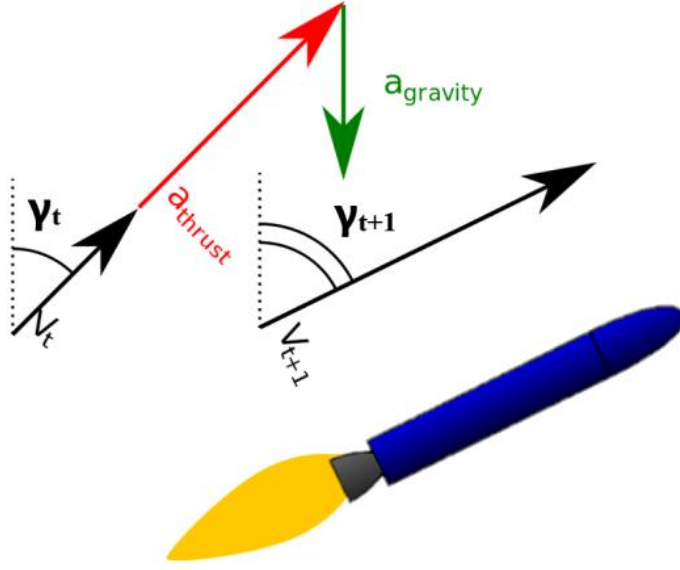


Figure 2.1 : Gravity loss thrust vector.

The numerical calculation of drag loss, which involves the time integration of the drag force, is given below. Further information regarding the calculation of drag force can be found in the chapters on force and environmental models.

$$\Delta V_{drag\ loss} = \int_{t_0}^{t_f} \frac{D(t)}{m(t)} dt \quad (2.7)$$

Steering loss arises when there's a misalignment between the thrust vector and the vehicle's velocity vector.

$$\Delta V_{steering\ loss} = \int_{t_0}^{t_f} \frac{T(t)}{m(t)} (1 - \cos(\delta + \alpha)) dt \quad (2.8)$$

The effects and dynamics of steering loss are further shown in **Figure 2.2**.

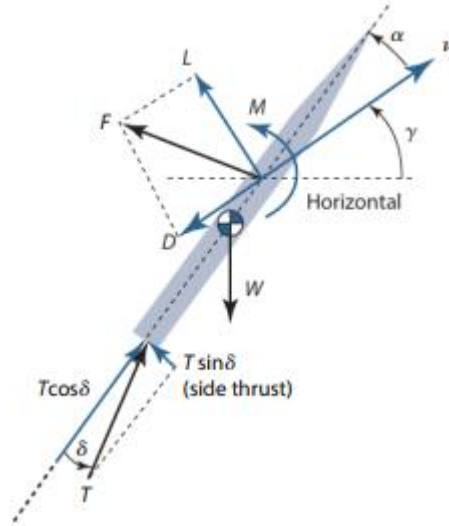


Figure 2.2 : Steering loss visualization.

Estimating pressure loss is a challenging task, one that is highly dependent on engine parameters and flight profile as well. It is worth noting that engine designers can approximate this loss, and it is often represented as a parameter known as effective specific impulse. While the simulation account for the decrease in thrust due to ambient pressure, the detailed calculation of pressure loss will not be covered within the scope of this thesis.

In summary, the challenge of achieving an "Earth to Orbit" trajectory essentially boils down to a fundamental trade-off between acquiring kinetic and potential energy. To attain orbit, a spacecraft must achieve orbital velocity, which is entirely horizontal for circular orbits and nearly horizontal for elliptical ones. This requires a gain in horizontal velocity, ideally achieved by applying horizontal thrust, as this approach also minimizes gravity loss, most dominant loss component of the delta-v budget. However, horizontal ascent within the atmosphere is impractical due to drag and other atmospheric factors that could compromise the structural integrity of the launch vehicle or cause excessive heating. Consequently, the spacecraft must also ascend to exit the atmosphere, essentially gaining potential energy. The root of the problem lies in simultaneously gaining altitude to leave the atmosphere while gradually increasing horizontal velocity until orbital speed is achieved upon atmospheric exit. The implementation of an effective trajectory is crucial in optimizing this balance, particularly within the constraints of the specific mission.

2.2 Flight Phases

The ascent flight of a launch vehicle is a sequence comprising several distinct phases, each with its challenges and constraints. Understanding these phases and associated constraints is crucial for successful mission planning and execution.

In this chapter, the discussion will focus on various flight phases that are common to all launch vehicle ascent flights. The planning and design of these phases are meticulously detailed for multiple reasons. These include objectives like maximizing the payload mass injected into orbit, reducing the risk of structural and payload damage, ensuring range safety, maintaining effective communication (for instance, enabling the Space Shuttle to direct its antennas) (NASA, 1973), and aligning the heading to match the required azimuth for the desired orbital inclination, among others. Additionally, detailed planning is essential for mission success, as it helps manage factors such as fuel efficiency and trajectory precision. Correctly planning these phases and their associated constraints is therefore vital for the overall success of the mission. An example flight CONOPS can be seen in **Figure 2.3** for Korea's Space Launch Vehicle 1.

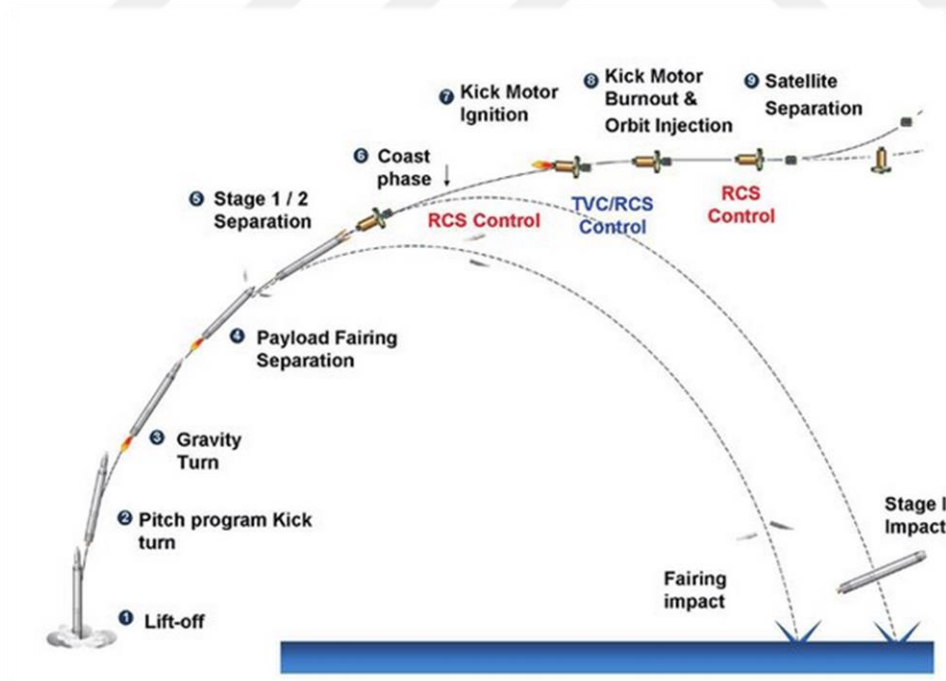


Figure 2.3 : An example CONOPS, Flight phases of Korea Space Launch Vehicle-I (Sun, Park, & Rae Roh, 2010).

A very detailed event sequence table of an actual Saturn V flight can be seen in **Figure 2.4**.

TYPICAL CRITICAL EVENT SEQUENCE, FIRST OPPORTUNITY TLI (EVENT TIMES ARE BASED ON AS-507 SIMULATIONS FOR MISSION H-1 NOVEMBER 14, 1969 WINDOW, 72.029° FLIGHT AZIMUTH)					
TIME FROM FIRST MOTION (HR:MIN:SEC)	TIME FROM REFERENCE (HR:MIN:SEC)	EVENT	TIME FROM FIRST MOTION (HR:MIN:SEC)	TIME FROM REFERENCE (HR:MIN:SEC)	EVENT
-0:00:17.3 0:00:00.0 0:00:00.4 0:00:01.4	T ₁ -0:00:17.7 T ₁ -0:00:00.4 T ₁ +0:00:00.0 T ₁ +0:00:01.0	Guidance Reference Release First Motion Liftoff Begin Tower Clearance Yaw Maneuver	0:12:28.8 0:12:56.9 0:13:18.3	T ₅ +0:00:59.0 T ₅ +0:01:27.1 T ₅ +0:01:48.5	LH ₂ Continuous Vent Open S-IVB APS Ullage Cutoff Begin Orbital Navigation Calculations
0:00:09.4 0:00:12.3 0:01:06.9 0:01:23.5	T ₁ +0:00:09.0 T ₁ +0:00:11.9 T ₁ +0:01:06.5 T ₁ +0:01:23.1	End Yaw Maneuver Pitch and Roll Initiation Mach 1 Maximum Dynamic Pressure	2:37:41.8 2:38:23.8	T ₆ +0:00:00.0 T ₆ +0:00:42.0	Begin S-IVB Restart Preparations O ₂ H ₂ Burner (Helium Heater) On
0:02:15.0 0:02:38.3	T ₂ +0:00:00.0 T ₂ +0:00:23.3	S-IC Center Engine Cutoff Begin Tilt Arrest	2:38:24.0 2:45:58.1 2:45:58.6 2:47:11.8 2:47:14.9 2:47:19.8	T ₆ +0:00:42.2 T ₆ +0:08:16.3 T ₆ +0:08:16.8 T ₆ +0:09:30.0 T ₆ +0:09:33.1 T ₆ +0:09:38.0	LH ₂ Continuous Vent Closed S-IVB APS Ullage Ignition Helium Heater Off S-IVB Engine Restart Sequence S-IVB APS Ullage Cutoff S-IVB Reignition (Start Tank Discharge Valve Opens) S-IVB Engine at 90% Thrust MR Shift (First Opportunity Only) S-IVB Engine Cutoff, Second Burn
0:02:42.2 0:02:42.7 0:02:42.9	T ₃ +0:00:00.0 T ₃ +0:00:00.5 T ₃ +0:00:00.7	S-IC Outboard Engine Cutoff S-II Ullage Rocket Ignition Signal to Separation Devices and S-IC Retro-rockets	2:47:22.3 2:49:02.3 2:53:04.8	T ₆ +0:09:40.5 T ₆ +0:11:20.5 T ₇ -0:00:00.2	S-IC/S-II First Plane Separation Complete S-II Engine Start Sequence Initiated S-II Ignition (Start Tank Discharge Valve Opens) S-II Engines at 90% Thrust S-II Ullage Thrust Cutoff S-II Aft Interstage Drop (Second Plane Separation) LET Jettison (Crew Action) Initiate IGM S-II Lox Tank Pressurization Flowrate Step S-II Center Engine Cutoff S-II Fuel Tank Pressurization Flowrate Step
0:02:43.0 0:02:43.6 0:02:44.6 0:02:46.6 0:02:47.2 0:03:12.9	T ₃ +0:00:00.8 T ₃ +0:00:01.4 T ₃ +0:00:02.4 T ₃ +0:00:04.4 T ₃ +0:00:05.0 T ₃ +0:00:30.7		2:53:05.0 2:53:05.5 2:53:05.7 2:53:05.8 2:53:08.6 2:53:10.0 2:53:14.8 2:53:25.0	T ₇ +0:00:00.0 T ₇ +0:00:00.5 T ₇ +0:00:00.7 T ₇ +0:00:00.8 T ₇ +0:00:03.6 T ₇ +0:00:05.0 T ₇ +0:00:09.8 T ₇ +0:00:20.0	Set Time Base 7 LH ₂ Continuous Vent Open Lox Nonpropulsive Vent Open LH ₂ Nonpropulsive Vent Open Flight Control Coast Mode On Enable SC Control of LV Translunar Injection Initiate Maneuver to and Maintain Local Horizontal Alignment (CSM Forward, Heads Down) Lox Nonpropulsive Vent Closed LH ₂ Nonpropulsive Vent Closed LH ₂ Continuous Vent Closed Initiate Maneuver to and Maintain TD&E Attitude CSM Separation (Variable) CSM/LM Docking (Variable) LH ₂ Nonpropulsive Vent Open LH ₂ Nonpropulsive Vent Closed SC/LV Final Separation (Variable) Initiate Maneuver to and Maintain S-IVB Evasive Attitude (Variable)
0:03:18.4 0:03:23.0 0:04:22.2	T ₃ +0:00:36.2 T ₃ +0:00:40.8 T ₂ +0:01:40.0		2:55:35.7 3:08:04.8 3:08:04.8 3:08:05.0	T ₇ +0:02:30.7 T ₇ +0:14:59.0 T ₇ +0:14:59.8 T ₇ +0:15:00.0	
0:07:41.2 0:07:42.2	T ₃ +0:04:58.0 T ₃ +0:05:00.0				
0:09:10.7 0:09:11.5 0:09:11.6	T ₄ +0:00:00.0 T ₄ +0:00:00.8 T ₄ +0:00:00.9	S-II Outboard Engine Cutoff S-IVB Ullage Ignition Signal to Separation Devices and S-II Retro-rockets	3:18:05.0 3:28:05.0 3:53:05.4 4:08:04.0 4:18:05.0 4:21:05.0	T ₇ +0:25:00.0 T ₇ +0:35:00.0 T ₇ +1:00:00.4 T ₇ +1:14:59.0 T ₇ +1:25:00.0 T ₇ +1:28:00.0	
0:00:11.7 0:09:11.7	T ₄ +0:00:01.0 T ₄ +0:00:01.0	S-II/S-IVB Separation S-IVB Engine Start Sequence, First Burn			
0:09:14.7	T ₄ +0:00:04.0	S-IVB Ignition (Start Tank Discharge Valve Opens)			
0:09:17.2 0:09:19.3 0:09:23.5 0:11:23.1 0:11:29.6	T ₄ +0:00:06.5 T ₄ +0:00:08.6 T ₄ +0:00:12.8 T ₄ +0:02:12.4 T ₅ -0:00:00.2	S-IVB Engine at 90% Thrust S-IVB Ullage Thrust End S-IVB Ullage Case Jettison Begin Chi Freeze S-IVB Cutoff, First Burn	4:29:05.0 4:29:06.2 4:30:26.2 4:38:45.0	T ₈ +0:00:00.0 T ₈ +0:00:01.2 T ₈ +0:01:21.2 T ₈ +0:09:40.0	Set Time Base 8 S-IVB APS Ullage Ignition S-IVB APS Ullage Cutoff Initiate Maneuver to and Maintain Slingshot Attitude LH ₂ Continuous Vent Open Start Lox Dump End Lox Dump Lox Nonpropulsive Vent Open LH ₂ Nonpropulsive Vent Open S-IVB APS Ullage Ignition S-IVB APS Ullage Cutoff Initiate Maneuver to and Maintain Communications Attitude
0:11:29.8 0:11:30.1 0:11:39.6 0:11:49.8	T ₅ +0:00:00.0 T ₅ +0:00:00.3 T ₅ +0:00:09.8 T ₅ +0:00:20.0	Set Time Base 5 S-IVB APS Ullage Ignition Parking Orbit Insertion Initiate Maneuver to and Maintain Local Horizontal Alignment (CSM Forward, Heads Down) Begin Orbital Guidance	4:38:45.2 4:50:25.0 4:51:23.0 4:52:25.2 4:52:30.0 5:25:45.0 5:30:45.0 5:39:05.0	T ₈ +0:09:40.2 T ₈ +0:21:20.0 T ₈ +0:22:18.0 T ₈ +0:23:20.2 T ₈ +0:23:25.0 T ₈ +0:56:40.0 T ₈ +1:01:40.0 T ₈ +1:10:00.0	
0:11:50.3	T ₅ +0:00:20.5				

Figure 2.4 : Event sequence of Saturn V Apollo 12 mission (NASA, 1969).

2.2.1 First stage

2.2.1.1 Vertical Lift-Off or tower clearance

Lift-off phase is the first flight phase of our discussion. In our simulation, the model for the vertical climb of a launch vehicle involves ascending vertically to a specific height, necessary to clear service towers, and other structures surrounding the launch pad. This vertical phase is crucial in the early moments after liftoff and it is applied until the pitch-over program starts. To accurately replicate this in our simulation, the vertical lift-off phase is physically enforced by applying thrust exclusively in the upward direction within the Up-East-North (UEN) coordinate frame, thereby ensuring a realistic and controlled vertical ascent.

2.2.1.2 Roll program

The roll program is a maneuver typically executed shortly after a launch vehicle clears the tower. It involves rotating the vehicle around its longitudinal axis to adjust its heading or azimuth along the before the pitch-over. It is not mandatory since a vehicle can directly pitch over to the necessary azimuth without a roll, though it is essential in certain scenarios. For instance, in the case of the Space Shuttle, the roll program was done for orienting antennas towards the ground as well. Essentially, this maneuver is primarily employed to ensure that the pitch axis plane of the vehicle aligns with the correct flight heading. In the context of the vectorized simulator in this thesis, the roll program phase is omitted. Instead, the simulator is designed to directly pitch towards the desired azimuth angle's plane, simplifying the initial phase of the flight path.

2.2.1.3 Pitch over

As previously discussed in the chapter 2.1.3, achieving orbit requires the launch vehicle to both gain horizontal velocity and exit the Earth's atmosphere. This requires a transition from the initial vertical or near-vertical ascent to a more horizontal trajectory. Such a transition is done through a pitch program. Here, 'pitching' involves adjusting the long axis of the vehicle, either tilting it upwards or downwards, to control its ascent or descent. This adjustment leads to an angle of attack and the start of a gradual turning process. Various techniques can be implemented during this turn, all aiming to minimize aerodynamic loads and fine-tune the trajectory while the vehicle is still within the Earth's atmosphere. Generally, an open-loop approach is employed for the pitch-over maneuver. In the initial phase of the flight, open-loop steering

instructions are established through parameter optimization. (Hanson, Shrader, & Cruzen, 1995)

The pitch-over maneuver generally starts early in the flight, often within the first 10-20 seconds when the vehicle's speed is still relatively low. This early initiation is strategic since it enables the vehicle to start gaining horizontal velocity before reaching the point of maximum dynamic pressure (max-q), where maneuverability is significantly limited due to high aerodynamic loading. Delaying the gain of horizontal velocity could make trajectory corrections more challenging in later flight stages, as it would require substantial force to alter the velocity vector's direction, leading to increased steering loss. Additionally, vehicle would have to spend a significant amount of time flying vertically which increases the gravity losses as well. Conversely, the vehicle must also avoid accelerating too rapidly while still in the atmosphere for avoiding aerodynamic loading and also for a healthy stage separation.

This phase represents a critical aspect of the vectorized simulator discussed in this thesis. The simulator employs a brute-force approach to explore various parameters of the pitch program, searching for the most efficient way to conduct this maneuver within the given constraints. Understanding and optimizing this phase is important to the success of launch vehicle missions, as it significantly impacts the overall trajectory and efficiency of the ascent.

The photo taken during the ascent of the space shuttle can be seen in **Figure 2.5**.



Figure 2.5 : Space Shuttle performing roll and pitch maneuver after lift-off (Shuttlesource, 2023).

2.2.1.4 Gravity turn

The Gravity Turn is the phase that follows after the initial pitch-over maneuver. During this phase, vehicle typically maintains a zero angle of attack, an approach that helps minimize aerodynamic loading. This phase is characterized by the vehicle following a trajectory that allows gravity to naturally turn it towards the horizontal, aligning it with the desired orbit. Since the vehicle is already tilted, the gravity will make the thrust vector more horizontal assisting to gain more horizontal velocity without a significant angle of attack. The effect can be examined in **Figure 2.6**.

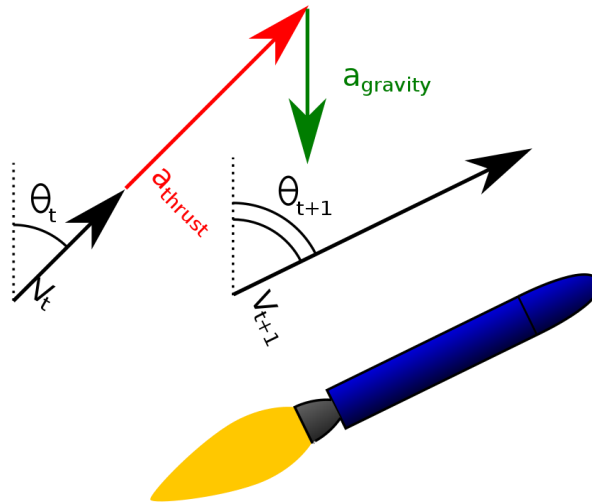


Figure 2.6 : Gravity turn vector visualization.

One of the most significant moments during the Gravity Turn phase is the passage through maximum dynamic pressure, commonly referred to as max-q. This typically occurs around 70-80 seconds after liftoff. Max-q represents the point during the ascent where the vehicle experiences the highest level of aerodynamic stress. The combination of increasing velocity and atmospheric density at this stage results in the greatest force exerted on the vehicle's structure.

$$q = \frac{1}{2} \rho V^2$$

During max-q, launch vehicles with a high lift-off thrust-to-weight ratio might need to reduce engine thrust, a process known as throttling down. This action is necessary to limit the aerodynamic loading and prevent structural damage. Managing the forces during max-q is critical for the integrity and safety of the vehicle, as the stresses encountered can have significant impacts on its performance and structural health.

2.2.1.5 Main engine cut off (MECO) and stage separation

The transition from the first to the second stage of a launch vehicle, known as stage separation, is a complex process in the ascent phase. This sequence begins once the first stage has spent its propellant and is ready to be jettisoned, allowing the vehicle to continue its journey with a significantly reduced weight. Managing stage separation effectively is vital for the success of the mission.

The altitude at which stage separation occurs is a critical factor. If the vehicle has followed a more horizontal trajectory, the separation may happen at a lower altitude

within the atmosphere. However, conducting stage separation too low can be hazardous due to the significant drag forces still present. These drag forces disproportionately affect the upper stage, as its leading surface area faces the oncoming airflow. During the brief phase when the engines are not firing, this drag can cause a deceleration, potentially bringing the second stage dangerously close to the first stage, risking collision.

Another concern during stage separation is the effect of this deceleration on the propellant inside the second stage. The absence of thrust can lead to propellant sloshing, pushing the liquid to the top of the tank. This sloshing can disrupt the proper functioning of the propellant feed system and engine might not be able to be ignited because of the choking of the propellant feed system

Given these considerations, stage separation is typically performed at an altitude above 60-65 kilometers for most launch vehicles. This ensures that the separation occurs outside the denser layers of the atmosphere, where drag forces are significantly reduced. In this thesis, this requirement is considered as a minimum stage separation altitude constraint. By simulating and applying this constraint, the program helps ensure a safer and more efficient transition between stages, a key aspect of successful launch vehicle operation.

2.2.2 Second Stage

2.2.2.1 Fairing jettisoning

The jettisoning of the payload fairing is usually initiated once the vehicle has ascended to a point where the aerodynamic loads from atmospheric passage fall below a critical level. The payload fairing, designed to shield the payload from aerodynamic forces and aerodynamic heating during the initial ascent, becomes functionless as these atmospheric effects diminish.

Typically, the jettisoning of the payload fairing happens shortly after the Main Engine Cut-Off (MECO) and the subsequent stage separation. This timing ensures that the payload is adequately protected during the most vulnerable phases of the ascent, where atmospheric interaction is most intense, and is exposed only when it is safe to do so.

The decision to jettison the fairing is based on various factors, including altitude, velocity, and external atmospheric conditions. Removing the fairing at the appropriate

time is crucial – too early, and the payload might be subjected to residual atmospheric forces; too late, and the additional weight of the fairing unnecessarily burdens the vehicle, consuming more fuel and potentially affecting the trajectory.

In the context of this thesis, a heat flux constraint scheme has been applied, which ensures that the fairing won't be separated until the heat flux drops below a certain threshold.

Figure 2.7 demonstrates the fairing separation process where pneumatic actuators used for jettisoning the two plates which composes the fairing.

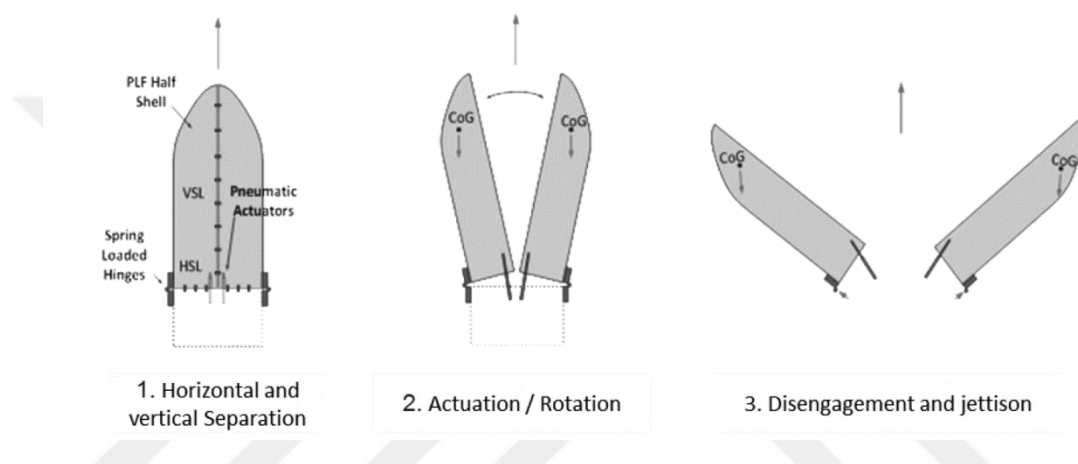


Figure 2.7 : Demonstration of a traditional fairing jettison (ESA, 2019).

2.2.2.2 Orbit insertion

Upon the successful jettisoning of the payload fairing, the launch vehicle's ascent transitions into a critical phase of orbit insertion. This segment of the mission is usually accompanied with a closed-loop guidance, a sophisticated control strategy that ensures the vehicle's trajectory is continuously adjusted based on real-time data to achieve the intended orbit with high precision.

Closed-loop guidance relies on a robust feedback system that continuously monitors the vehicle's current state against its desired trajectory. Inputs from inertial measurement units, star trackers, and other navigational sensors feed into the onboard computer, which calculates any necessary corrections. The guidance system accounts for variables such as remaining propellant, gravitational influences, and the vehicle's current velocity and altitude.

The orbit insertion phase's success is judged by its accuracy in achieving the intended orbit and the stability of the final trajectory. This requires an understanding of orbital mechanics and the execution of a refined guidance algorithm.

There are multiple strategies for achieving the mission's final orbit. It is not always the case that the target orbit is reached directly at the end of the initial powered ascent. Trajectory designers may opt for an intermediate step, placing the vehicle into a lower energy parking or transfer orbit. From there, after coasting to the desired apogee altitude, the vehicle can be maneuvered into the mission orbit. This approach allows for various advantages, such as maximizing payload capacity, ensuring a safe crash point in the event of an engine failure, and maintaining communication links.

This maneuver, commonly referred to as an apogee kick, is a critical phase in the mission profile. It is executed by a subsequent engine burn that adjusts the vehicle's trajectory from the parking orbit to the intended final orbit. The difficulties of planning an apogee kick, including its timing, duration, and the impact on mission success, are complex and will be explored in detail later in this thesis. The apogee kick is an essential part, very often used in missions, and offers flexibility and control over the spacecraft's path to its operational orbit.

2.3 Previous Research

2.3.1 Trajectory performance assessment

As mentioned before, the place of trajectory performance assessment is crucial in the conceptual and preliminary phase of design cycles of launch vehicles. POST is a program that is used extensively in the US since the 80's for this purpose.

There are examples studies that leverages POST to create faster and more automated tools, mainly done by the Advanced Concepts Office (Waters, Garcia, B., A., & Threet, 2013). Holt et al (Holt, Dee, & Diaz, 2015) propose an automated method for executing the POST, a python program called "multiPOST", aiming to generate data for a surrogate model for efficient trade space exploration. By developing a heuristic-based approach to automatically close cases in POST, they demonstrate its effectiveness in achieving the initial goals. The study utilizes an example problem involving an SLS-like vehicle with a parameterized upper stage, successfully

completing a substantial number of repetitions within a reasonable time frame, aligning with typical conceptual design studies.

Surrogate models find widespread application in engineering across various domains, including launch vehicles as described in this section. The term “metamodel” describes a model of a model. The primary objective of a metamodel is to provide a computationally efficient approximation for the analysis of interest. These approximations enable exploration of the design space during conceptual design, a task that would be impractical using more resource-intensive engineering analyses. By down-selecting from a large pool of potential designs, surrogate models facilitate informed decisions, which can then be refined using detailed engineering simulations.

In a subsequent study by the same team from the Advanced Concepts Office, Steffens et al (Steffens, Edwards, Mavris, Dees, & Diaz, 2016) investigated various methods for generating trajectory data, specifically for later use in surrogate modeling. By comparing these methods, they identified optimal criteria for data generation. In a subsequent step, Dees et al. (Dees, Zwack, Edwards, & Steffens, 2016) explored graph theory for surrogate model analysis. Their updated research enhances the existing “multiPOST” tool, designed for parallel analysis executions, by integrating graph theory. This novel approach not only reduces analysis time but also enhances the quality and quantity of generated data, surpassing the effectiveness of the original methodology.

2.3.2 Guidance

The optimization of a launch vehicle's trajectory and the precision of its guidance systems are paramount to the success of a space mission. This section of the thesis will explore the current methodologies and technologies employed in this critical aspect.

Historically, launch vehicle guidance systems have utilized open-loop control during atmospheric ascent, switching to closed-loop control in the post-atmospheric phase (Gath & Calise, 2001), or after payload fairing separation, as demonstrated by Saturn V in **Figure 2.4** (NASA, 1969).

NASA has traditionally leveraged open-loop guidance method in the atmospheric phase of launch due to their simplicity and reliability. (Hanson, Shrader, & Cruzen, 1995) This method utilizes pre-optimized pitch angle look-up tables that prescribe the

vehicle's orientation based on time or velocity during ascent. This method is particularly important in the early stages of launch when the vehicle is subject to high atmospheric pressures and dynamic forces. One of the systems that employed this approach was the Space Shuttle, which used a predefined set of pitch maneuvers to control the ascent trajectory until the Solid Rocket Boosters (SRBs) were jettisoned. (NASA, 1973)

The reason for using open-loop control in this phase is the predictability of atmospheric conditions up to a certain altitude, allowing for a robust and straightforward guidance process without the need for real-time adjustments. This approach minimizes the risk of guidance errors due to sensor inaccuracies or delays in control commands, which are more likely to occur within the dense atmosphere. It also simplifies the vehicle's design by reducing the dependency on complex sensing and computing systems during the most turbulent phase of flight.

Agencies like NASA and companies like SpaceX employ closed-loop guidance algorithms to navigate the critical exo-atmospheric phases of a launch. These algorithms enable the vehicle to make real-time adjustments based on current state data, adapting to the variances in vehicle performance and environmental conditions encountered in space. The Iterative Guidance Mode (IGM) and Powered Explicit Guidance (PEG) are examples of such algorithms. Closed-loop guidance is favored outside the atmosphere due to the reduced atmospheric interference, allowing for more accurate sensor readings and control responses. This approach is essential for fine-tuning the trajectory for orbit insertion, rendezvous maneuvers, and landing sequences where precise control is crucial.

The focus in this thesis is to assess the launch vehicle performance by scanning all plausible flight corridors (with minimal steering loss) by exploiting vectorized methods. There is no study like this in the current literature that uses vectorized simulators for launch vehicle simulations.

2.3.3 Vectorized simulations

In aerospace engineering, in the simulation of launch vehicle trajectories, there appears to be a lack of research directly applying vectorization techniques. However, the concept of vectorization has been widely adopted in various other scientific domains to efficiently simulate expansive search spaces. This computational strategy has

proven invaluable in fields requiring the analysis of large datasets or complex systems, such as high-energy physics (Gheata, 2019), where it's used to simulate particle interactions and model phenomena at the quantum level. Similarly, in the realm of artificial intelligence, vectorization plays a crucial role in agent-based simulations for reinforcement learning, enabling the rapid processing of numerous potential actions and their outcomes to optimize decision-making algorithms (Bettini, Kortvelesy, Blumenkamp, & Prorok, 2022).

Furthermore, the application of vectorization extends to fluid dynamics (Curtis, Niemeyer, & Sung, 2022), facilitating the detailed simulation of flow patterns, turbulence, and other fluid interactions across vast computational grids. Vectorization has been utilized in computational fluid dynamics since the 80s. (Gentzsch, 1984) In the field of chemistry, vectorization aids in the modeling of chemical reactions (Zirwes, et al., 2019), allowing for the simultaneous computation of multiple reaction pathways and the dynamic interactions of molecules in large systems.

Given the success of vectorization in these varied fields, its application in the simulation of launch vehicle trajectories holds significant promise. By leveraging vectorization, it's conceivable to dramatically increase the efficiency of trajectory simulations, enabling the exploration of a much broader range of trajectories and vehicle configurations within a shorter timeframe. This could lead to more optimized mission profiles and a deeper understanding of the impact of various design choices on a vehicle's performance. Thus, while direct applications in launch vehicle trajectory simulations may currently be limited, the potential benefits of incorporating vectorization into simulations are clear and need further investigation.



3. FLIGHT SIMULATOR & MODELS

3.1 Assumptions

The trajectory simulation program utilized in this thesis operates under a 3-Degree of Freedom (3-DOF) framework, requiring a set of simplifying assumptions across the models. This approach is typically used for determining optimal trajectories and evaluating the overarching performance characteristics of a given configuration. Detailed phenomena such as propellant sloshing, structural flexing, advanced aerodynamic interactions, wind effects, and the complex interactions of subsystems, along with other linear and non-linear dynamic factors, are typically considered in later stages of the launch vehicle design process using 6-Degree of Freedom (6-DOF) simulations. Since this is very time intensive for early phases of the design process, optimized trajectories are designed using 3-DOF simulators. (Hanson, Shrader, & Cruzen, 1995). The assumptions used in the simulator can be seen in **Table 3.1**.

Table 3.1 : Assumptions of the simulator.

Model	Assumption
General	Differential equations are integrated on Earth Centered Inertial (ECI) Frame.
	Launch vehicle is considered a rigid body.
Atmosphere and Aerodynamics	Atmospheric parameters like density, temperature, and speed of sound are derived from altitude-specific lookup tables based on the US Standard Atmosphere Model.
	Wind effects are not included.
	A linear model is utilized to approximate aerodynamic forces.
Propulsion	Thrust and mass flow rate are assumed constant.
	Engine dynamics are excluded.

Other engine related mass ejections such as, unburnt fuel ejection at motor ignition and cut-off, or fuel ejections from vents are not modeled.

Earth and Gravity	J2 gravity model has been employed.
	An oblate, rotating Earth model is adopted.
Guidance	Open loop guidance models used for both stages.
	The rocket's nose aligns with velocity vector if no pitch is made.
	First stage guidance operates up to Mach 0.8; beyond this, the vehicle flies with zero angle of attack for the rest of the first stage of flight to constraint aerodynamic loads.
	Launch azimuth is pre-calculated; pitch-over occurs in the target azimuth plane without side slip maneuvers

3.2 Reference Coordinate Frames

3.2.1 ECI and ECEF frames

The ECEF (Earth-Centered, Earth-Fixed) coordinate system is a geocentric framework, with its origin at the Earth's center of mass. The x-axis of this system is aligned with the intersection of the Greenwich meridian and the equator, while the z-axis is oriented toward the North Pole. This system is dynamic, moving with the Earth's rotation, hence, its orientation relative to sun always changes.

In contrast, the ECI (Earth-Centered Inertial) coordinate system, although sharing the same central point at the Earth's center of mass, differs in orientation. Its x-axis is aligned towards the vernal equinox. This particular alignment makes the ECI system useful for tasks in celestial navigation and in establishing the Earth's positioning in relation to other objects in space. The ECI system's z-axis, similar to that of the ECEF, points towards the North Pole. However, the defining characteristic of the ECI system is its inertial provides a consistent frame of reference.

The ECI system, with its stable orientation, is preferred for the trajectory analysis of satellites and other space objects. The necessity for a fixed frame of reference in space dynamics means that calculations for the motion of these objects are either calculated directly in the ECI system or converted to it for accuracy and reliability.

The coordinate transformations between ECI and ECEF frames is somewhat straightforward. One just needs to find the angle difference between their z axes and rotate along the z direction. In the equation below the angle $\theta - \lambda_E$ is the difference between ECI and ECEF at t_0 the conversions after t_0 should be done by incorporating the Earth's rotation, $\omega(t - t_0)$. (Chobotov, 2002)

$$ECI \rightarrow ECEF: C^{ECEF/ECI} = \begin{bmatrix} \cos[(\theta - \lambda_E) + \omega(t - t_0)] & \sin[(\theta - \lambda_E) + \omega(t - t_0)] & 0 \\ -\sin[(\theta - \lambda_E) + \omega(t - t_0)] & \cos[(\theta - \lambda_E) + \omega(t - t_0)] & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad (3.1)$$

In **Figure 3.1** the visualization of the ECI and ECEF frames can be examined. While they share the y and z axes, ECI's x axis is towards the vernal equinox while the ECEF's x axis is on the Greenwich median or x_g direction in the figure.

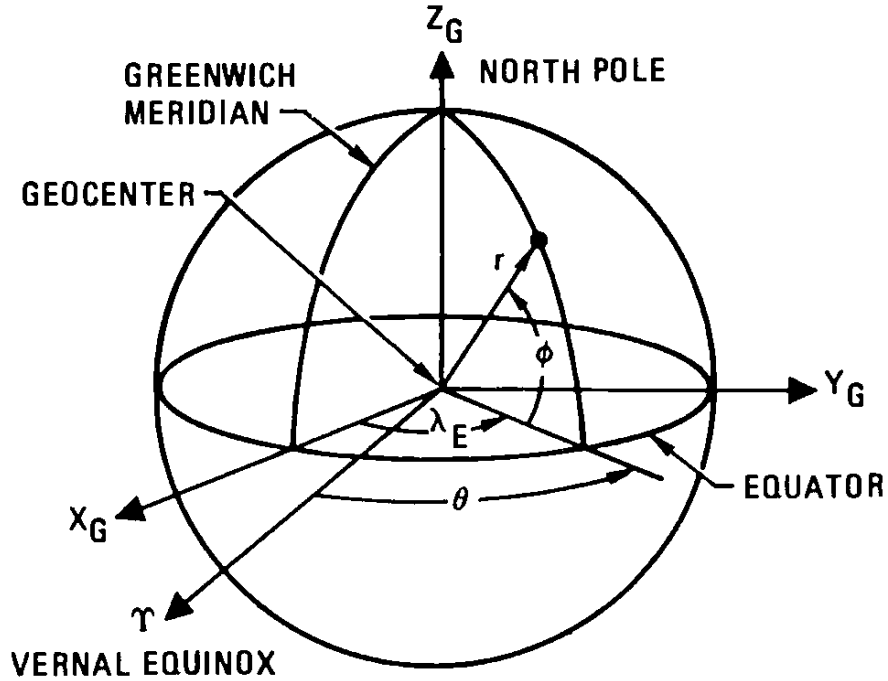


Figure 3.1 - ECI and ECEF frames (Chobotov, 2002).

To summarize, both the ECEF and ECI coordinate systems, originating from the Earth's center, have unique applications. The ECEF is linked to the Earth's surface and is optimal for Earth-centric applications, while the ECI, with its inertial and stationary nature, is ideal for space-oriented calculations.

For this study, the differential equations of motions are integrated along the ECI reference frame while ECEF and LLA (Latitude, Longitude and Altitude) are used for easy human interpretation.

3.2.2 Local horizon frame

During simulations, the Local Horizon, or UEN (Up, East, North) coordinate framework, is employed as an intermediate step when transitioning between the wind axis system and the ECI (Earth-Centered Inertial) and ECEF (Earth-Centered, Earth-Fixed) systems. In this context, the angle of the flight path and the azimuth are derived from the relative speed components within the UEN system. The conversion between the ECEF system and the UEN system can be carried out with just the latitude and longitude data like in the equation (Tewari, 2007) below with 2 rotations, first along z axis by longitude and then along the y axis in negative latitude direction.

$$ECEF \rightarrow UEN: C^{ECEF/UEN} = \begin{bmatrix} \cos \delta \cos \lambda & \cos \delta \sin \lambda & \sin \delta \\ -\sin \lambda & \cos \lambda & 0 \\ -\sin \delta \cos \lambda & -\sin \delta \sin \lambda & \cos \delta \end{bmatrix} \quad (3.2)$$

The visualization of the Local Horizon Frame can be seen in **Figure 3.2**.

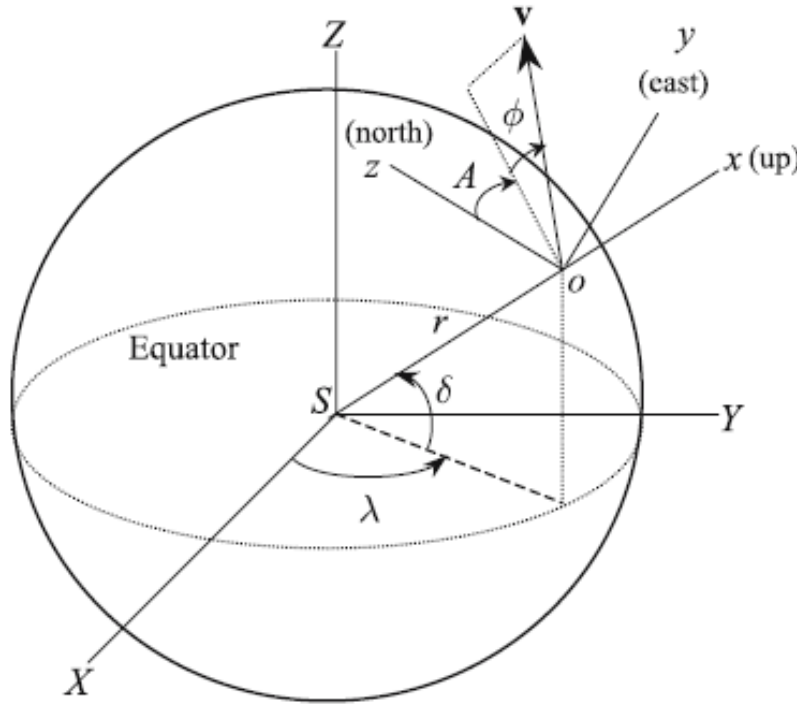


Figure 3.2 : Local horizon frame (Tewari, 2007).

3.2.3 Wind axis frame

In this axis system, established with its origin at the center of the vehicle, the x-axis is aligned with the vehicle's direction of velocity. This alignment ensures a direct correlation between the vehicle's forward motion and the x-axis orientation. The

remaining axes are defined and oriented in relation to the vehicle's physical structure, ensuring a comprehensive spatial reference framework. Within this framework, critical forces such as thrust, which propels the vehicle forward, drag, which acts against the vehicle's motion, and lift, crucial for aerial vehicles, are all applied and analyzed. Additionally, the guidance maneuver commands are applied in this axis system providing a reference for direction.

Wind Axes are useful since the forces of thrust, drag and lift forces can be easily interpreted and calculated. All of the forces mentioned are applied in this axis to be converted later to ECI frame.

3.3 Force and Environment Models

3.3.1 Thrust – mass model

In the conducted simulations, a constant thrust model has been used. The simulation starts with the input of key parameters: the engine's vacuum thrust, mass flow rate and vacuum-specific impulse (I_{sp}). To accurately transition from the vacuum thrust to the net thrust in atmospheric conditions, it is essential to input the nozzle exit section's diameter or area. This transition process integrates the atmospheric pressure at the respective altitude being considered.

$$T = (T_{vacuum} - T_{pressure}) \quad (3.3)$$

$$T_{pressure} = P_{atm} * A_{nozzle} \quad (3.4)$$

$$A_{nozzle} = \pi \left(\frac{D_{nozzle}}{2} \right)^2 \quad (3.5)$$

Furthermore, the calculation of the thrust vector, as described, initially takes place within the wind axis system before getting moved to ECI frame. Since the x axis of the wind axis reference aligns with rockets main body, if there is no guidance maneuver present, α , the angle of attack will be zero and the only component will be in the velocity direction. There is no component of thrust in the banking angle direction in the simulation since the pitch maneuver will start in the plane of interest with an initial given azimuth angle, which satisfies the target inclination angle in the terminal

conditions. Therefore, thrust is never applied in the y direction. Additionally, given that the simulator operates on a 3 Degree of Freedom (DOF) basis, it does not utilize thrust vector control angles within the simulation. Thus, we operate under the assumption that thrust is exerted in the direction of the angle of attack, consistent with the assumptions mentioned earlier.

$$T_{wind_x} = T * \cos(\alpha) \quad (3.6)$$

$$T_{wind_y} = 0 \quad (3.7)$$

$$T_{wind_z} = -T * \sin(\alpha) \quad (3.8)$$

The simulation also includes features that allow for the modulation of thrust during critical phases such as the maximum dynamic pressure transition or for managing axial acceleration. However, it is important to note that the implementation of thrust throttling, though feasible within the simulation framework, is not encompassed within the scope of this particular study. This ensures a focused approach on specific aspects of the program while acknowledging potential areas for future development and possible refinement.

3.3.2 Aerodynamics model

Process of determining the aerodynamics coefficients is an important process since this will affect the launch vehicle's behavior throughout the atmospheric portion of the flight.

Missile DATCOM, an abbreviation for Missile Data Compendium (Vukelich, Stoy, Burns, & Castillo, 1988), is an aerodynamic software tool used extensively in the aerospace industry for the analysis of missile and rocket designs. Developed by the United States Air Force, it tries to estimate the aerodynamic stability and control characteristics of missile-like bodies throughout their flight. Missile DATCOM operates by utilizing semi-empirical, empirical, and engineering methods based on wind tunnel tests and flight test data to provide estimates for aerodynamic derivatives. These include coefficients of lift, drag, and moment across a wide range of angles of attack, Mach numbers, and side-slip angles. The robustness of the software lies in its ability to handle various vehicle geometries, from simple bodies to complex configurations with multiple stages, fins, and control surfaces. As such, Missile DATCOM is an invaluable resource in the preliminary design and analysis stages,

enabling engineers to predict how changes to design parameters will affect a vehicle's performance long before costly prototypes are built and tested.

In this thesis, this process starts with creating an estimated geometry of the vehicle, which includes detailed contouring reflective of the vehicle's physical form, (Rosema, Doyle, Auman, & Underwood, 2011). This geometric model serves as the input for the Missile DATCOM software, which is tasked with the analysis of aerodynamic characteristics over a range of flight conditions.

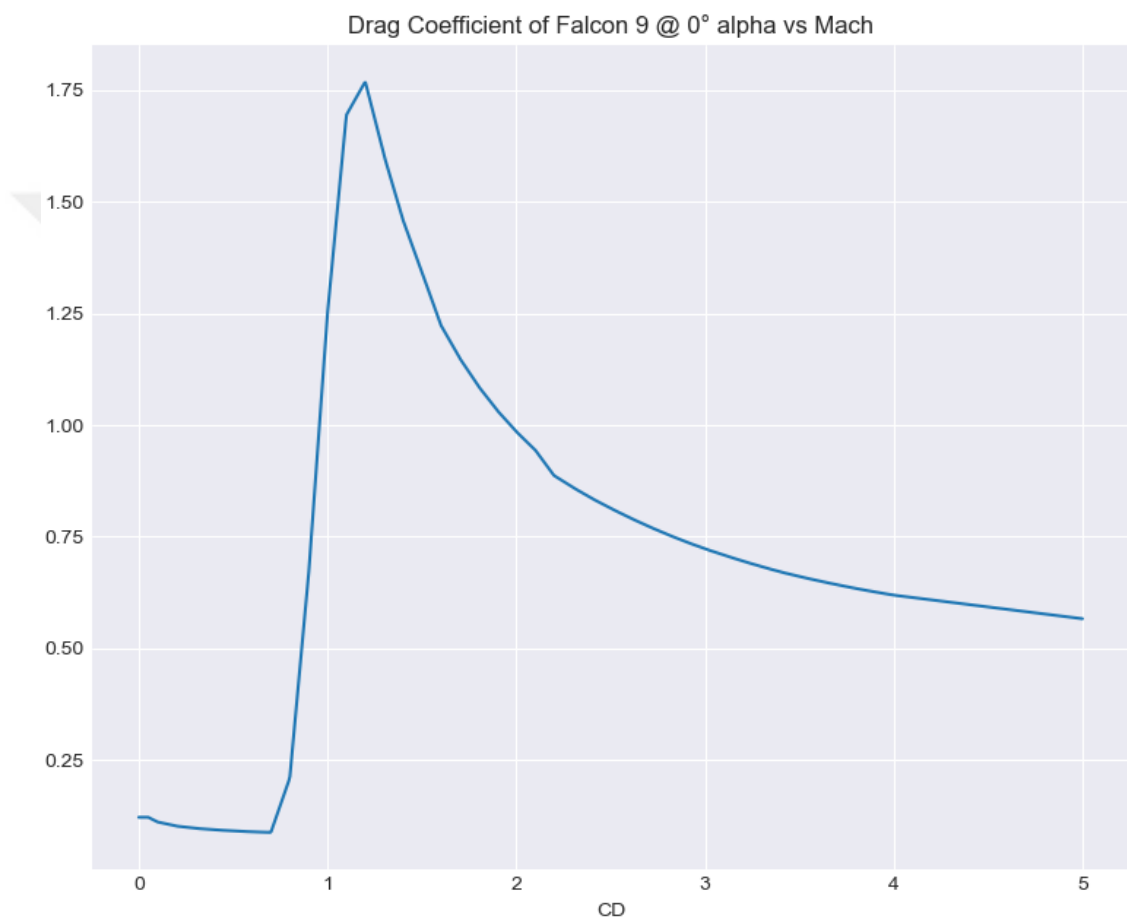


Figure 3.3 : An example CD graph vs Mach for x axis (Missile DATCOM).

Missile DATCOM was employed to process the vehicle's geometry. The software computed the drag coefficient (CD) and lift coefficient (CL) across a spectrum of Mach numbers and angles of attack (alpha values). By systematically varying these two parameters, a grid of CD and CL values, encapsulating the vehicle's aerodynamic profile throughout its operational envelope. The data produced by Missile DATCOM was then consolidated into an aerodynamic lookup table. This table is structured to enable two-dimensional interpolation, allowing for the determination of aerodynamic

coefficients at given Mach and alpha angles in the vehicle's flight trajectory. By interpolating between the data points, the simulation predicts the CD and CL values for intermediate flight conditions not explicitly covered by the DATCOM output for fast execution. An example CD graph versus mach number can be seen in **Figure 3.3**.

After the estimation of coefficient of drag and lift, calculating the drag force relatively straightforward. Firstly, the dynamic pressure is calculated with velocity (relative to air) and air density, which is calculated in the atmosphere model.

$$q = \frac{1}{2} \rho V_{rel}^2 \quad (3.9)$$

Then the dynamic pressure is multiplied with the reference area and coefficient of drag which is the biggest cross-sectional area of the rocket in that phase of the flight.

$$A = \pi r^2 \quad (3.10)$$

$$F_{drag} = C_D A q \quad (3.11)$$

3.3.3 Gravity model

Within the simulation program, options are available to select either a spherical or geoid model of Earth's gravity. For the purposes of this thesis, the geoid model, incorporating the J2 term, has been adopted. The computation of gravitational force is carried out in the Earth-Centered Inertial (ECI) frame of reference.

For a perfectly spherical Earth, the expression for gravity is straightforward and can be represented as follows:

$$\vec{g}_{ECI} = g_x \hat{i} + g_y \hat{j} + g_z \hat{k} \quad (3.12)$$

$$\vec{g}_{ECI} = -\frac{\mu}{r^3} \vec{r} = -\frac{\mu}{r^3} \begin{bmatrix} x \\ y \\ z \end{bmatrix} \quad (3.13)$$

$$\vec{F}_g = \begin{bmatrix} F_{g_x} \\ F_{g_y} \\ F_{g_z} \end{bmatrix} = m * \begin{bmatrix} g_x \\ g_y \\ g_z \end{bmatrix} \quad (3.14)$$

However, this representation is a simplification, as the shape of the Earth more closely resembles an oblate spheroid. To model the Earth's oblateness, zonal harmonics can

be applied seen in **Figure 3.4**. Of these, the J2 harmonic exerts the most influence, with the impact of higher-order harmonics diminishing progressively.

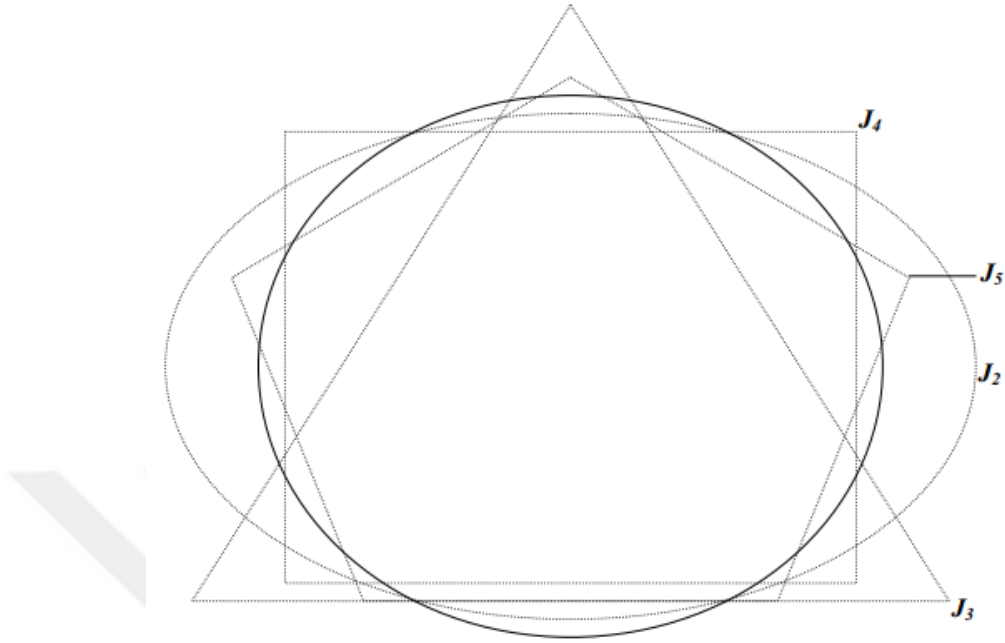


Figure 3.4 : Spherical harmonics of an axisymmetric planet (Tewari, 2007).

As a result, for an optimal balance between accuracy and computational efficiency, this study has chosen to incorporate only the J2 zonal harmonic in the gravity model. The inclusion of Earth's oblateness through the J2 term modifies the gravitational accelerations as follows:

$$\begin{bmatrix} g_x \\ g_y \\ g_z \end{bmatrix} = \begin{bmatrix} -x \left(\frac{\mu}{r^3} \right) \left[1 + 1.5j_2 \left(\frac{r_E}{r} \right)^2 \left(1 - \frac{5z^2}{r^2} \right) \right] \\ -y \left(\frac{\mu}{r^3} \right) \left[1 + 1.5j_2 \left(\frac{r_E}{r} \right)^2 \left(1 - \frac{5z^2}{r^2} \right) \right] \\ -z \left(\frac{\mu}{r^3} \right) \left[1 + 1.5j_2 \left(\frac{r_E}{r} \right)^2 \left(3 - \frac{5z^2}{r^2} \right) \right] \end{bmatrix} \quad (3.15)$$

3.3.4 Atmosphere model

US Standard Atmosphere (NASA, 1976) model has been used for determining the ambient pressure, temperature, density and speed of sound.

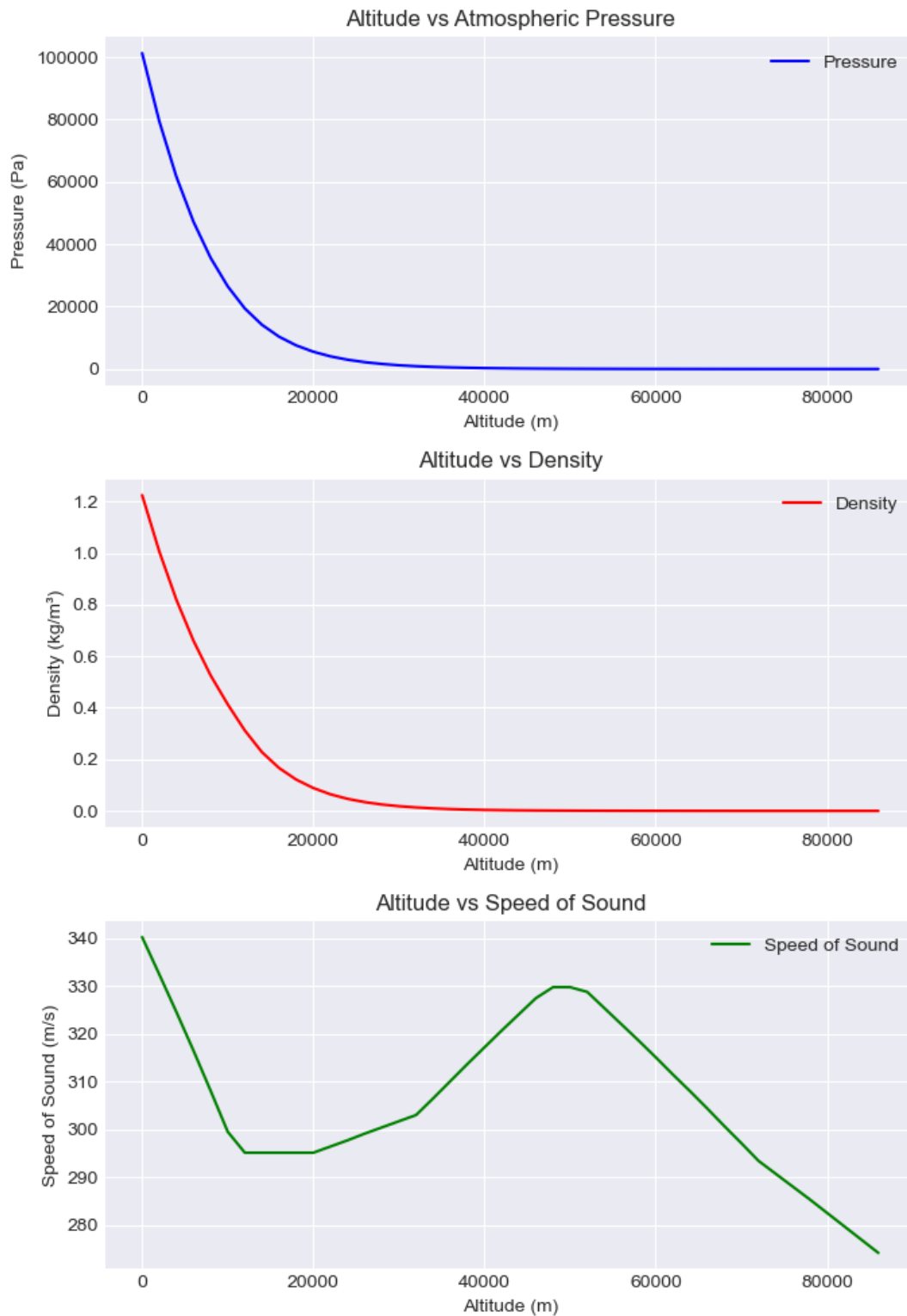


Figure 3.5 : Atmosphere parameters by altitude.

The US Standard Atmosphere Model is a statistical model that was developed and refined through extensive meteorological data collection and analysis. It is

internationally recognized and widely used in aerospace engineering and flight dynamics to standardize and predict atmospheric conditions for various altitudes.

The model separates the atmosphere into several layers, each with its distinct temperature gradient. These layers include the troposphere, stratosphere, mesosphere, and thermosphere, each characterized by unique thermal behaviors which significantly impact the aforementioned atmospheric properties.

In this study, the model is used to interpolate atmospheric properties at varying altitudes during a launch vehicle's ascent. By employing interpolation methods, the program can accurately determine values for atmospheric pressure, density, and speed of sound at any given altitude, which are essential for calculating aerodynamic forces on the vehicle.

Figure 3.5 visually illustrates the correlation between altitude and the atmospheric parameters of pressure, density, and speed of sound, showcasing how each parameter diminishes or varies with increasing altitude.

3.3.5 Guidance model

One of the primary objectives of this study is to determine the optimal pitch-over maneuver for a controlled ascent during a maneuvered flight. The inputs of the guidance model constitute a four-dimensional design space for scanning the optimal maneuver. For the first stage of the flight, there are three inputs for the pitch-over maneuver. During the second stage of flight, no maneuvers are applied until the payload fairing is jettisoned. Following the jettisoning of the fairing, a constant alpha rate maneuver is initiated, characterized by a fixed angle change in the angle of attack. This is another parameter that is included in the scanning process.

The pitch-over maneuver which is employed in this study can be shortly defined by a function dependent on three parameters. These parameters include the initiation moment of the guidance (which could be determined by the starting Mach number or altitude), the amplitude of the guidance angle, and the termination moment of the guidance (which could also be defined by the ending Mach number).

The pitch over maneuver should be continuous to avoid large loads on the vehicle. Therefore, appointing a sinusoidal function depending on these three parameters is intuitive, which was implemented in the first stages of this study. However, after some

experimenting, another sinusoidal function which is more tilted to the left, meaning a more aggressive angle change at first but smoother towards the end, has been chosen to represent the pitch-over angle. This was done solely to start gaining horizontal speed earlier in the flight, as a design choice. The comparison between these two sinusoidal functions can be seen in **Figure 3.6**.

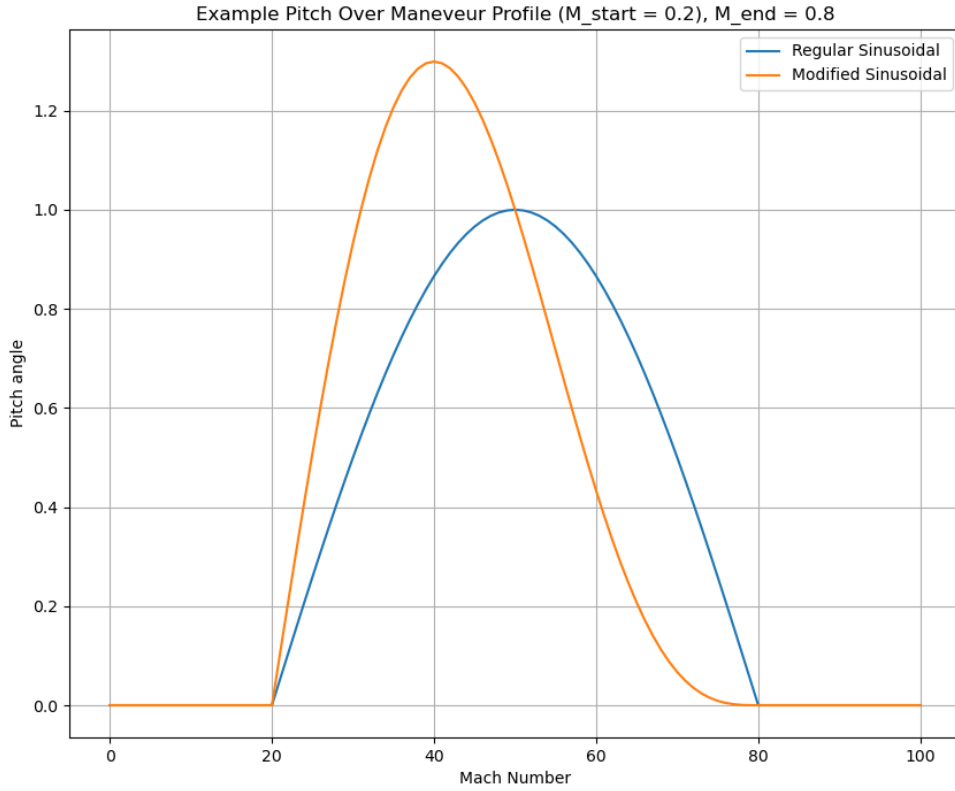


Figure 3.6 : Sinusoidal profiles pitch-over maneuvers.

However, it is crucial to express that the figure above will be much narrower during the flight depending on the launch vehicle's T/W ratio. If the vehicle exceeds the specified Mach values very quickly, the angle change might be dramatic and unacceptable in terms of aerodynamic loads and thrust vector control rate limitations. In order to prevent this, a filter in the search space has been implemented to remove the combinations which have less than 0.25 difference between their starting and ending Mach numbers.

The coefficient of the sinusoidal term and the functions can be seen below. This expression is only valid and active during the simulation when the flight is in the required Mach range of M_{start} and M_{end} .

$$C(t) = \frac{\pi}{(M_{end} - M_{start})} * (M(t) - M_{start}) \quad (3.16)$$

$$\alpha_{sinusoidal}(t) = \alpha_{amplitude} * \sin(C(t)) \quad (3.17)$$

$$\alpha_{modified\ sinusoidal}(t) = \alpha_{amplitude} * \left(\sin(C(t)) + \frac{\sin(2C(t))}{2} \right) \quad (3.18)$$

The primary goal of this study is to optimize these values through brute-force methods for the optimal ascent flight, ensuring compliance with flight constraints and terminal requirements. The search space range for the initial pitch-over maneuver is detailed in **Table 3.2**. More than twelve thousand combinations of these parameters will be scanned for each launch vehicle configuration, as will be discussed in next chapters.

Table 3.2 : An example search space for vectorized simulator.

Guidance Parameter	Min and Max Values	Description
M_{start}	0.05-0.4 Mach	Starting Mach number
M_{end}	0.2 – 0.8 Mach	Ending Mach number
$\alpha_{amplitude}$	0°- 3°	Amplitude of the pitch over

Since the pitch-over maneuver should occur in the subsonic regime, the highest value for the third parameter in the search space has been set at 0.8 Mach. This limitation ensures that the maneuver remains within the aerodynamically efficient range, avoiding the problems associated with transonic and supersonic flight.

In this section of the thesis, the focus is on elaborating the guidance model's structure and the rationale behind choosing specific parameters for the pitch-over maneuver. The chosen parameters are crucial for optimizing the vehicle's trajectory, especially considering the aerodynamic and mechanical constraints of early flight stages. Understanding the interplay of these parameters is essential for achieving an optimal ascent profile, which is a key aspect of this research.

3.4 Dynamic Equations and ODE's

Since the program that is used in this thesis only consists of three degrees of freedom dynamics, only the translational motion equations will be given under this chapter.

Newton's second law of motion asserts that the change of linear momentum of a body is proportional to the external force.

$$F = \frac{d(mv)}{dt} \quad (3.19)$$

Linear momentum can change in both ways, the mass might change while the speed changes as well. If there is no force acting, the body will stay undisturbed and the linear momentum will stay constant.

For an instantaneous moment we can assume the mass is constant and we get:

$$F = m \frac{d(v)}{dt} = ma \quad (3.20)$$

Then the acceleration in this context is the rate change of velocity V over time.

$$a = \dot{V} = \frac{F}{m} \quad (3.21)$$

We can write these equations in vector form for velocity and position by simply integrating the acceleration by time.

$$\vec{v}(t) = \vec{v}_0 + \int_0^t \frac{\vec{F}(t')}{m} dt' \quad (3.22)$$

$$\vec{r}(t) = \vec{r}_0 + \int_0^t \vec{v}(t') dt' \quad (3.23)$$

This is done via numerical methods in the simulation. The time derivatives of the state variables are numerically integrated by equating them to the right-hand sides of the differential equation set, which are calculated and expressed in the Earth-Centered Inertial (ECI) frame. This approach negates the necessity to deal with additional terms such as Coriolis and centrifugal acceleration when transitioning from relative to absolute acceleration since acceleration is computed in absolute terms directly.

The thrust force and aerodynamic force, collectively referred to as the partial net force, are initially calculated in the wind axis frame. Afterward, necessary coordinate transformations are performed to express these forces in the ECI frame. The partial net

force in the wind axis frame is computed as the sum of drag (D), lift (L), and thrust (T) forces in this frame:

$$\vec{F}_{wind_{partial}} = \vec{D}_{wind} + \vec{L}_{wind} + \vec{T}_{wind} \quad (3.24)$$

This partial net force vector in the wind axis frame is then transformed first to the Up-East-North (UEN) frame, followed by transformations to the Earth-Centered Earth-Fixed (ECEF) frame and finally to the ECI frame. The transformations are represented as:

$$\begin{aligned} \vec{F}_{wind_{partial}} &\rightarrow DCM_{Wind \rightarrow UEN} \rightarrow DCM_{UEN \rightarrow ECEF} \rightarrow DCM_{ECEF \rightarrow ECI} \\ &\rightarrow \vec{F}_{ECI_{partial}} \end{aligned} \quad (3.25)$$

$$\vec{F}_{ECI} = \vec{F}_{ECI_{partial}} + \vec{F}_{ECI_{gravity}} \quad (3.26)$$

The total force in the ECI frame $\vec{F}_{ECI}$, is the sum of the partial net force in the ECI frame and the gravitational force in the ECI frame.

The computation of state variable values at successive simulation steps employs the Runge-Kutta 4th order (RK4) integration method. This approach is selected for its enhanced accuracy compared to the Euler method, and its better computational efficiency relative to other multi-step integration techniques.

The Runge-Kutta 4th order method abbreviated as RK4, is an advanced numerical technique for solving ordinary differential equations. It calculates four estimates of the slope at different points within each time step and then combines these to produce a final estimate of the solution visualized in **Figure 3.7**. This process involves taking the initial slope, two slopes at the midpoint, and the slope at the end of the interval, which are then weighted and averaged to obtain a highly accurate approximation of the next value in the solution.

$$k_1 = f(t_n, X_n) \quad (3.27)$$

$$k_2 = f\left(t_n + \frac{\Delta t}{2}, X_n + k_1 * \frac{\Delta t}{2}\right) \quad (3.28)$$

$$k_3 = f\left(t_n + \frac{\Delta t}{2}, X_n + k_2 * \frac{\Delta t}{2}\right) \quad (3.29)$$

$$k_4 = f(t_n + \Delta t, X_n + k_3 * \Delta t) \quad (3.30)$$

$$X_{n+1} = X_n + \frac{\Delta t}{6} (k_1 + 2k_2 + 2k_3 + k_4) \quad (3.31)$$

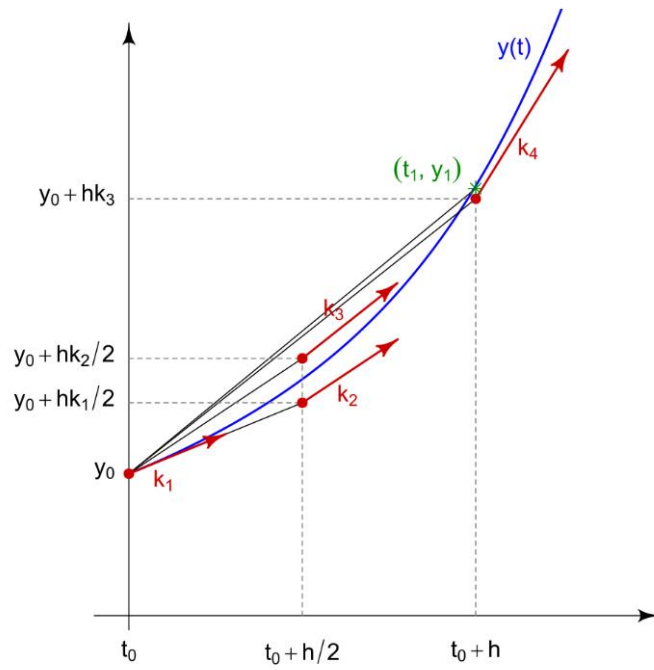


Figure 3.7 : RK4 method.

4. VECTORIZED SIMULATION PROGRAM AND METHODOLOGY

4.1 Vectorization

In upcoming chapters, the program's method of simulating trajectories efficiently without the use of 'for loops' will be examined. This process is designed to handle up to 7000 simulations at once, with the program requiring only a single run for any number of simulations. This is achieved through the development of custom functions capable of vectorized operations. An important aspect of this approach is the introduction of an additional dimension in each variable, which enables simultaneous storage and processing of data from multiple simulations. For instance, the variable representing thrust is transformed from a singular value into a vector that contains thrust data for all the trajectories being simulated at once. The process of vectorizing functions varies in complexity; while some functions adapt to vectorization easily, others, such as interpolation functions, present significant challenges, and some may even be impractical if the computation of one variable influence another in the same tensor.

Following sections will provide details on vectorization, which is a core principle in this thesis. The main innovation presented in this thesis is the use of vectorization to simulate a large number of launch vehicle trajectories, enabling swift and thorough near optimal trajectory optimization and performance evaluation of launch vehicles.

4.1.1 SIMD Units

SIMD, or Single Instruction, Multiple Data, is a parallel computing paradigm that has become important in computational tasks requiring high efficiency. This technology enables a single operation to be performed simultaneously on multiple data, significantly accelerating the computational process. A basic visualization of working mechanism of SIMD can be examined in **Figure 4.1**.

SIMD involves executing the same instruction on different pieces of data in parallel. This is in contrast to traditional sequential processing, where each instruction operates

on a single data point at a time. By processing multiple data elements simultaneously, SIMD can greatly reduce the time required for large-scale computations. For instance, in vector arithmetic operations, such as adding two large arrays, SIMD allows for the elements of these arrays to be added in parallel across several computational units, rather than one by one.

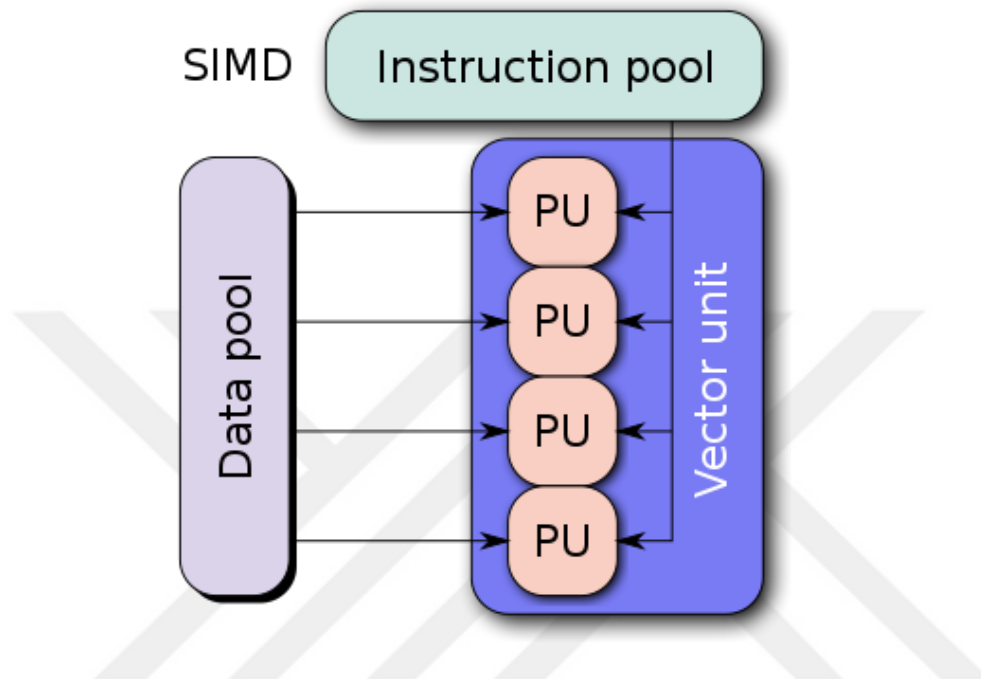


Figure 4.1 : Working mechanism of and SIMD unit.

The working mechanism of SIMD is often mistaken for parallelization across multiple cores, but its mechanics are more intricate. SIMD operates by utilizing multiple computational units within each core, as modern and even older CPUs typically have several SIMD units. This approach is akin to vector processing. For the purposes of this thesis, the program runs on a single core but multiple instances of program can be executed across multiple cores as well, if there is no memory problem.

A very basic visual depicting the difference between SISD (Single Instruction Multiple Data) and SIMD can be seen in **Figure 4.2**.

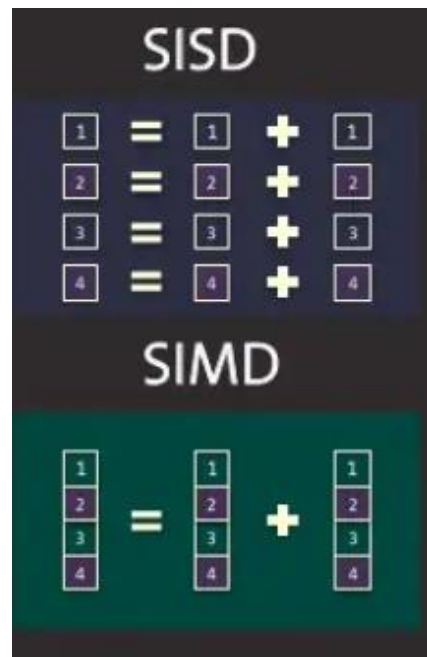


Figure 4.2 : SISD vs SIMD.

While SIMD initially appears advantageous, programming efficient SIMD routines can be challenging due to the need for low-level programming languages like Assembly. Nonetheless, SIMD instructions have been implemented in numerous scientific computing and linear algebra libraries across various programming languages. Libraries such as BLAS (Lawson, Hanson, Kincaid, & Krogh, 1979), LAPACK, and MKL (Intel, 2024) are notable examples of these low-level scientific computing libraries. They are often integrated into higher-level linear algebra packages of languages like Python, with NumPy being a prime example. In this study, NumPy is selected for its seamless usage and efficiency. NumPy harnesses several low-level, SIMD-supported linear algebra libraries, making it an ideal choice for rapid prototyping and quick implementation without sacrificing much of the performance.

In the next chapters about vectorization, all of the calculations and vectorization methods mentioned are executed on SIMD units since specific NumPy functions are chosen for speed improvements.

4.1.2 Array programming and NumPy module

Array programming has significantly changed the way data is handled in both scientific and industrial areas, using a clear and effective syntax to work with data in various structures like vectors or tensors. “NumPy”, a key library in Python for array programming is widely used in several scientific areas. One example is astronomy,

where it contributed to major achievements such as the discovery of gravitational waves and the first imaging of a black hole. It can be examined in **Figure 4.3**.

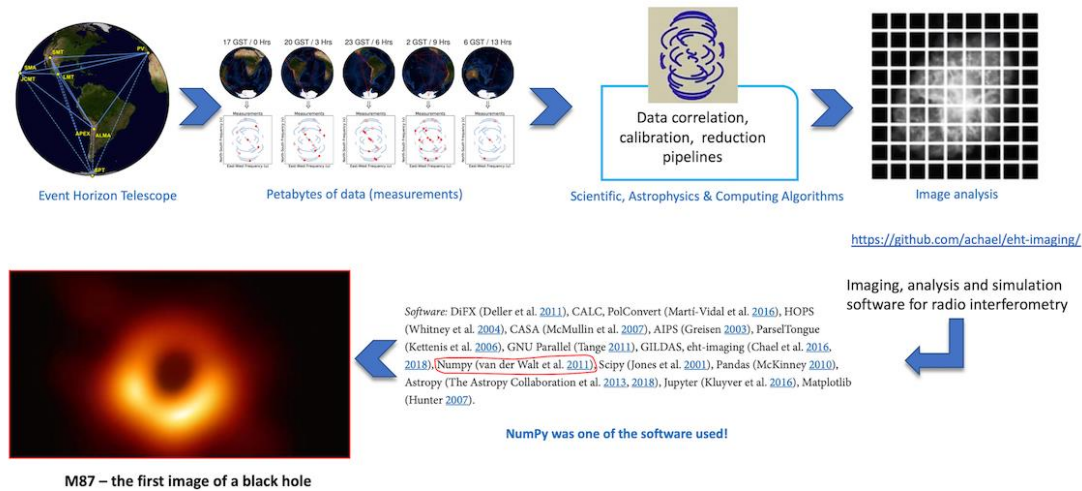


Figure 4.3 : Usage of NumPy in black hole imagery (NumPy, 2019).

The influence of NumPy goes beyond these areas. It is useful in chemistry for tasks like molecular modeling, in geoscience for interpreting seismic data, in biology for sequencing genetic data, and in psychology for statistical analysis. In applied sciences like materials science and engineering, NumPy is used for simulating and analyzing new materials. In finance and economics, its capacity to manage large datasets and perform advanced mathematical calculations makes it very valuable for data analysis and economic forecasting.

NumPy's impact is also seen in how it simplifies the programming approach, making it more user-friendly and efficient for organizing, exploring, and analyzing large sets of data. Its approach centered on arrays is both simple and powerful, allowing researchers and scientists to concentrate more on their scientific questions rather than on data manipulation complexities. Moreover, NumPy's popularity has led to the development of a broad ecosystem that includes specialized projects, all adopting interfaces and array objects similar to NumPy to ensure compatibility and ease of use. This widespread adoption highlights NumPy's foundational role not only in Python's scientific environment but in the wider field of scientific computing.

NumPy is essentially a very powerful data manipulation tool that can be used for any area where the data can be expressed as numbers. The backbone of the software that is used in this thesis has been designed with NumPy functions. In the upcoming chapters of this study, it will be discussed how NumPy's array operation functions

stand out by offering significant performance improvements compared to conventional programming methods.

4.1.2.1 Vectorization and Broadcasting

Vectorization is a method of programming that involves performing operations on entire arrays or large data simultaneously, rather than using loops to iterate over individual elements. This is made possible by NumPy's ability to perform element-wise operations. In the **Figure 4.4** section d Vectorization demonstrates this concept with the addition of two arrays. Instead of adding elements one by one, NumPy adds corresponding elements of each array in a single operation with multiple computational units.

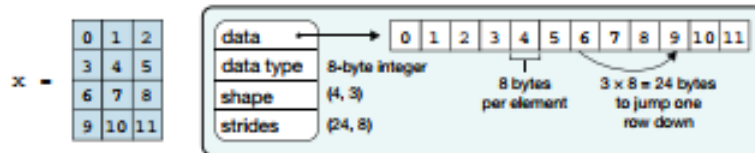
4.1.2.2 Vectorization of “if” conditions

Vectorizing complex programs presents several challenges, one of which is the handling of "if" conditions. In scenarios where the program is required to calculate parameters for every moment across multiple simulations, such as running 6000 simulations simultaneously, these challenges become pronounced. Consider a state vector X_{JD} with dimensions (6000, t, 6), where '6' represents various elements like geodetic latitude, longitude, altitude, non-inertial speed, flight path angle, and azimuth. Here, '6000' denotes the number of simulations, and t represents the total number of time steps. For every time step the program will calculate a state vector with a size of (6000, 6). Let's say, if a pitch-over maneuver is to be initiated once the altitude exceeds 1000 meters, this would necessitate checking each altitude element in this large tensor.

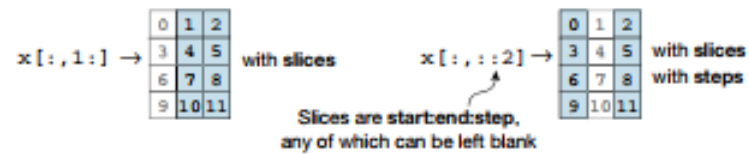
The intuitive approach might involve using a for-loop. However vectorized programming offers a more efficient solution known as "Boolean masks." This method allows for the comparison of every element in the altitude section of the array simultaneously without a need for the for-loop in Python, resulting in a Boolean array that indicates whether each simulation's altitude has surpassed the 1000-meter threshold or not. The resulting array, of size (6000, 1), will consist of True or False values corresponding to whether the altitude criterion is met or not. Speed comparison of this technique is illustrated in the code snippets in **Figure 4.5**, showing Boolean masks are roughly 64 times faster than regular if conditions for 6000 simulation data points.

Review

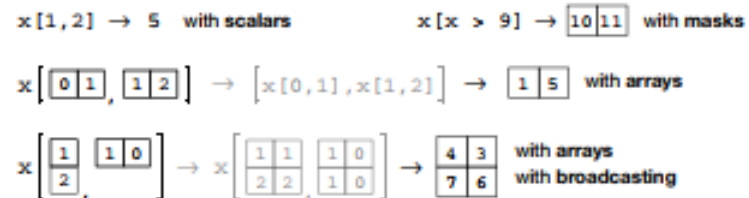
a Data structure



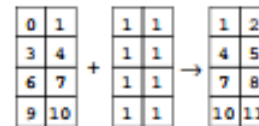
b Indexing (view)



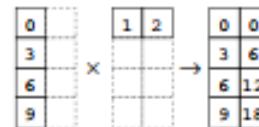
c Indexing (copy)



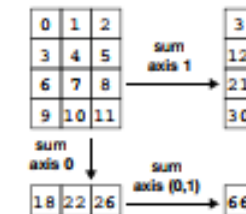
d Vectorization



e Broadcasting



f Reduction



g Example

```
In [1]: import numpy as np
In [2]: x = np.arange(12)
In [3]: x = x.reshape(4, 3)

In [4]: x
Out[4]:
array([[ 0,  1,  2],
       [ 3,  4,  5],
       [ 6,  7,  8],
       [ 9, 10, 11]])

In [5]: np.mean(x, axis=0)
Out[5]: array([4.5, 5.5, 6.5])

In [6]: x = x - np.mean(x, axis=0)

In [7]: x
Out[7]:
array([[ -4.5,  -4.5,  -4.5],
       [ -1.5,  -1.5,  -1.5],
       [  1.5,   1.5,   1.5],
       [  4.5,   4.5,   4.5]])
```

Figure 4.4 : Basic vectorized operations in NumPy (Harris, et al., 2020).

```

import numpy as np
import time

# Sample size to average the timings
n_sample = 1000
# Sample size and dimensions for demonstration
n_simulations = 6000
n_elements = 6

# Generate random data for demonstration
X_JD = np.random.random((n_simulations, n_elements)) * 2000 # Random values between 0 and 2000
pitch_angle = np.zeros(n_simulations)
alpha = 5 # Example value for alpha

def regular_if(X_JD, pitch_angle):
    for i in range(X_JD.shape[0]):
        if X_JD[i, 2] > 1000:
            pitch_angle[i] = alpha

def boolean_mask(X_JD, pitch_angle):
    mask = X_JD[:, 2] > 1000
    pitch_angle[mask] = alpha

# Timing the regular_if function
start_time = time.time()
for i in range(n_sample):
    regular_if(X_JD, pitch_angle)
regular_if_time = (time.time() - start_time)/n_sample

# Timing the boolean_mask function
start_time = time.time()
for i in range(n_sample):
    boolean_mask(X_JD, pitch_angle)
boolean_mask_time = (time.time() - start_time)/n_sample

print(f"Average time taken by regular_if function: {regular_if_time:.8f} seconds")
print(f"Average time taken by boolean_mask function: {boolean_mask_time:.8f} seconds")
x = regular_if_time/boolean_mask_time
print(f"Boolean mask is {x:.2f} times faster than regular if statement for {n_simulations} simulations.")

Average time taken by regular_if function: 0.00284101 seconds
Average time taken by boolean_mask function: 0.00004400 seconds
Boolean mask is 64.57 times faster than regular if statement for 6000 simulations.

```

Figure 4.5 : Speed comparison of if vs Boolean masks.

4.1.2.3 Speed Improvement on Matrix Multiplication and Algebraic Computations

One of the most computationally intensive tasks in mathematical operations is matrix multiplication. NumPy offers various functions to streamline this process, along with techniques to enhance their execution speed. This section will explore the performance gains achieved through vectorization in matrix multiplication and general algebraic computations. As previously noted, vectorized operations effectively utilize every available computing unit in the processor, assuming NumPy's core packages are optimized for the specific CPU being used. A code snippet in **Figure 4.6** demonstrates the speed enhancement between a standard dot product and a vectorized multiplication

of two transformation matrices. The results show a remarkable tenfold increase in speed.

```
import numpy as np
import time
n_sample = 1000 # Sample size to measure performance
n_simulations = 6000 # Sample size and dimensions for demonstration

DCM_1 = np.random.random((n_simulations, 3, 3)) # Random transformation matrices for all the simulations
DCM_2 = np.random.random((n_simulations, 3, 3)) # Random transformation matrices for all the simulations

def regular_dot(matrix_1, matrix_2):
    result = np.zeros_like(matrix_1)
    for i in range(matrix_1.shape[0]): # Akin to outer loop of running n simulations
        result[i, :, :] = np.dot(matrix_1[i, :, :], matrix_2[i, :, :])

def vectorized_dot(matrix_1, matrix_2):
    return np.matmul(matrix_1, matrix_2) # Running all the simulations at once

# Timing the regular_dot function
start_time = time.time()
for i in range(n_sample):
    regular_dot(DCM_1, DCM_2)
regular_dot_time = (time.time() - start_time)/n_sample

# Timing the vectorized_dot function
start_time = time.time()
for i in range(n_sample):
    vectorized_dot(DCM_1, DCM_2)
vectorized_dot_time = (time.time() - start_time)/n_sample

print(f"Average time taken by regular_dot function: {regular_dot_time:.8f} seconds")
print(f"Average time taken by vectorized_dot function: {vectorized_dot_time:.8f} seconds")
x = regular_dot_time/vectorized_dot_time
print(f"Vectorized dot product is {x:.2f} times faster than regular dot product for {n_simulations} simulations.")

Average time taken by regular_dot function: 0.01561518 seconds
Average time taken by vectorized_dot function: 0.00143856 seconds
Vectorized dot product is 10.85 times faster than regular dot product for 6000 simulations.
```

Figure 4.6 : Demonstration of 10x speed improvement for the dot product.

Moreover, this example also illustrates the simplification of syntax in the vectorized approach. In this context, the vectorized function is imagined to be processing transformation matrices for all 6000 simulations concurrently, whereas the for loop in the regular dot product represents the sequential execution of 6000 simulations in the outer loop of the simulation program.

A tenfold increase in speed for the dot product is noteworthy, however the most significant performance enhancement actually comes from computing algebraic equations. In the code snippet in **Figure 4.7**, functions are presented for a random algebraic equation, both in a vectorized format and a conventional format. This particular equation, comprising two divisions, one squaring, three additions, and one subtraction, is representative of the types of operations commonly encountered throughout the program. Remarkably, as shown in the lower part of the figure, there is a staggering 453x improvement in speed.

```

import numpy as np
import time

a = np.random.random(6000)
b = np.random.random(6000)
c = np.random.random(6000)

def vectorized_algebraic_eq(a, b, c):
    return a + b / c - b/a + 2*b + c**2

def regular_algebraic_eq(a, b, c):
    result = np.zeros_like(a)
    for i in range(a.shape[0]):
        result[i] = a[i] + b[i] / c[i] - b[i]/a[i] + 2*b[i] + c[i]**2
    return result

# Timing the regular_algebraic_eq function
start_time = time.time()
for i in range(n_sample):
    regular_algebraic_eq(a, b, c)
regular_algebraic_eq_time = (time.time() - start_time)/n_sample

# Timing the vectorized_algebraic_eq function
start_time = time.time()
for i in range(n_sample):
    vectorized_algebraic_eq(a, b, c)
vectorized_algebraic_eq_time = (time.time() - start_time)/n_sample

print(f"Average time taken by regular_dot function: {regular_algebraic_eq_time:.8f} seconds")
print(f"Average time taken by vectorized_dot function: {vectorized_algebraic_eq_time:.8f} seconds")
x = regular_algebraic_eq_time/vectorized_algebraic_eq_time
print(f"Vectorized dot product is {x:.2f} times faster than regular dot product for {n_simulations} simulations.")

Average time taken by regular_dot function: 0.01278483 seconds
Average time taken by vectorized_dot function: 0.00002820 seconds
Vectorized dot product is 453.28 times faster than regular dot product for 6000 simulations.

```

Figure 4.7 : Demonstration of 450x speed improvement for algebraic expressions.

4.1.2.4 BLAS, LAPACK and MKL

In high-performance scientific computing, the efficiency of numerical computations is extremely important. Basic Linear Algebra Subprograms (BLAS) (Lawson, Hanson, Kincaid, & Krogh, 1979) and Linear Algebra Package (LAPACK) (Dongarra, Demmel, & Ostrouchov, 1992) are foundational libraries in this domain, known for their efficiency in performing a variety of linear algebra operations. Dating back to the late 1970s, these libraries were originally developed by the Jet Propulsion Laboratory using the FORTRAN programming language. Today, they are widely adopted in the backend of numerous platforms. Notable examples include R, Python's NumPy, and MATLAB, all of which utilize BLAS and LAPACK to enhance their capabilities.

While BLAS focuses on lower-level vector and matrix operations, which are the building blocks for some higher-level algorithms. LAPACK, built on top of BLAS, provides routines for more complex tasks like matrix factorizations, solving linear systems, eigenvalue problems, and more. These libraries are optimized for

performance on a wide range of hardware, making them indispensable tools in computational software development.

As mentioned in the previous paragraph, NumPy, being a central package for numerical computations in Python, integrates BLAS and LAPACK for efficient execution of array operations. However, when NumPy is built with Math Kernel Library (MKL) (Intel, 2024) support, it leverages the enhanced performance capabilities of MKL, leading to significant speed improvements in various numerical tasks. This integration is crucial for computationally intensive applications like launch vehicle trajectory simulations, where processing large datasets and performing complex calculations rapidly is essential.

As can be seen in some of the benchmarks online, MKL offers 4x speed improvement in some cases (Science, n.d.). However, it is a necessity to mention that MKL is optimized for Intel CPU's only.

4.1.3 Handling memory issues

The memory consumption of the Vectorized Simulation Program is a critical aspect of its operation. For every variable that is progressed through time and recorded, a memory allocation of at least 'Number of Simulations' multiplied by 'Number of Time Steps' is required. In this context, the number of such variables will be referred to as 'channels'. Given that each variable is a 64-bit (or 8-byte) float, the total memory requirement for a single session can be calculated using the following equation:

$$\begin{aligned} \text{Min RAM Requirement} \\ = n_{simulations} * n_{timesteps} * n_{channels} * 8 [\text{Bayt}] \end{aligned} \quad (4.1)$$

Let's consider an example scenario: Approximately 6000 simulations can be run in one process of the program, and up to 8 parallel processes (assuming an 8 core CPU) can be initiated simultaneously. This means the total number of simulations running at the same time would be 48,000.

$$\sum_{n_{process}} n_{simulations} = 8 * 6000 = 48000 \quad (4.2)$$

Let's say we have chosen time step as 0.1:

$$n_{timesteps} = \frac{500}{0.1} = 5000 \quad (4.3)$$

The matrices corresponding to the vectors "X_ECI", "X_LDBARV", and "X_JD", each having six channels, are progressed through time. Three channels are allocated for the gravity vector written in Cartesian coordinates and two channels for drag and lift. Additionally, mass, thrust, and guidance angle are represented with one channel each. Thus, the minimum RAM requirement is calculated as follows:

$$n_{channels} = \underbrace{6 + 6 + 6}_{\text{State Variables}} + \underbrace{3}_{\text{Gravity Vector}} + \underbrace{2}_{\text{Drag Lift}} + \underbrace{1 + 1 + 1}_{\text{Thrust, Mass, alpha}} = 26 \quad (4.4)$$

$$\begin{aligned} \text{Minimum RAM Requirement} &= 48000 \times 5000 \times 26 \times 8 \text{ [Byte]} \\ &= 49.92 \text{ GigaByte} \end{aligned} \quad (4.5)$$

A requirement of approximately 49 GB of RAM is considerably high, even by modern computer standards. Consequently, the program has been designed to operate in two modes. In the low RAM mode, only the essential state variables that must be recorded during the simulation are kept in memory. This mode allows users to either use smaller time steps or run more simulations simultaneously. Variables not recorded can be easily reconstructed by the post-processing algorithm, albeit at the cost of extended processing time. In the low RAM mode, only three state variable tensors are recorded, reducing the number of progressed channels to 18 and achieving a 30% savings in memory and storage capacity.

4.2 Core Software Concept

As previously highlighted, the primary objective of developing the vectorized simulator is to swiftly evaluate the performance of a launch vehicle. This evaluation leverages the rapid processing capabilities of the simulator to explore a vast amount of flight trajectories. The program investigates potential trajectories, flight paths, and various transfer orbits to achieve the specified target orbit. This section will further explain the individual components of the program and their operational principles, separately from the ascent problem.

The vectorized simulator is architected to be a semi Object-Oriented Program (OOP). While it incorporates object-based components for data encapsulation, it also utilizes modules and functions for simplicity and ease of use. The design is centered around using objects primarily as data storage entities to maintain a clean and organized structure.

In this simulator, all variables, with the exception of certain constants like the gravitational constant, are stored within arrays. This structure is vital as it allows the simulator to perform operations across all simulations simultaneously. However, this approach poses inherent risks common in programming, where a minor error can lead to a misalignment in the arrays' ordering. To mitigate this, the simulator underwent rigorous validation against already established trajectory simulators utilized within the author's company, which is not in scope of this thesis.

The heart of the simulator lies in the flight propagator function, tasked with solving the equations of motion. This function, which accepts various differential equations, is also designed to handle event triggers. To avoid the computational issues that discontinuities might cause during the Runge-Kutta 4 (RK4) integration process, events are programmed to be triggered only at time steps, not in interval steps of Runge Kutta. Events such as the vertical ascent phase, pitch-over program, gravity turn, max Q, stage separation, fairing jettison, and coasting phases are detailed in the flight phases section of this thesis.

One distinctive capability of the simulator is its use of different timesteps during different simulation stages. This allows for heightened accuracy in critical flight segments, should the user require it for specific analytical purposes.

Figure 4.8 illustrates the vectorized propagation scheme. In this depiction, segments of the X_{ECI} tensor, labeled in Pythonic notation, are processed through the flight propagator. The tensor dimensions, represented by $n_{\text{simulation}} \times n_{\text{timesteps}} \times 6$, encapsulate both position and velocity. Consequently, at each timestep, $n_{\text{simulation}} \times 6$ parameters are computed in unison. This means that, 36,000 parameters are calculated simultaneously, negating the need for iterative loops.

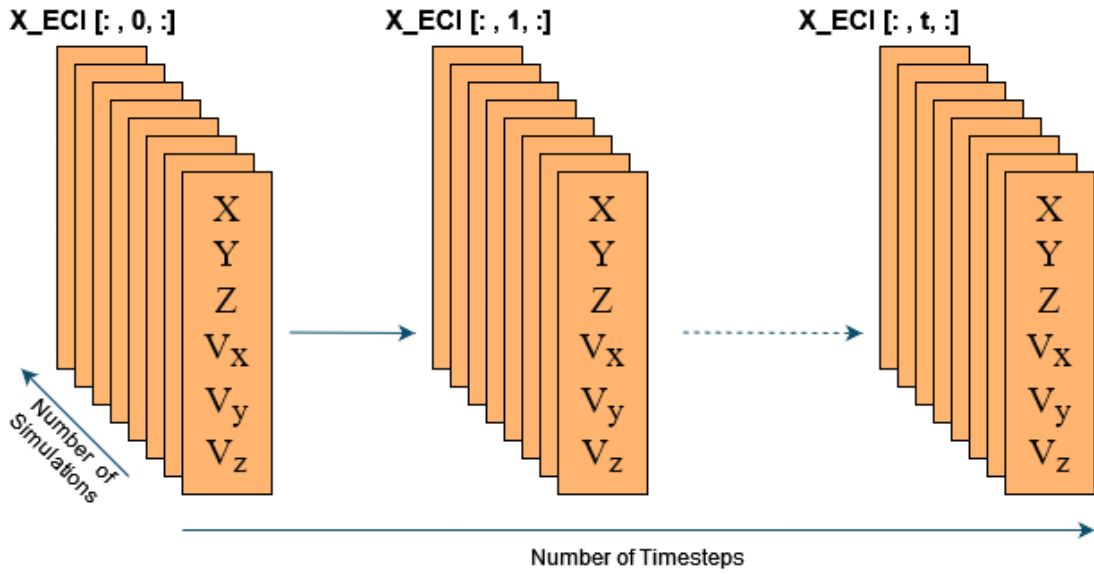


Figure 4.8 : Propagation of state variables, written in their native Python array notation.

The core structure of the program is built around three principal classes: Scenario, Stage, and Simulator. The process begins with initializing the Scenario class, which can be done through a JSON input file or a user-friendly script that simplifies the JSON file creation. Once the Scenario is set up, it becomes the foundation for the Simulator object, which then generates various Stage objects tailored to the specific scenario requirements. The Stage class primarily functions as a repository for the data associated with different segments of the launch vehicle's flight within the simulation. Further elaboration on these classes will be provided in the subsequent chapters. It should be noted that the functional models for aerodynamics, thrust and mass, gravity, atmosphere, and guidance are not class-based but are instead constructed as functions, which have been previously detailed in section 3.3.

4.3 OOP Structure

4.3.1 Scenario

The initialization of the Scenario object marks the start of the program. This can be achieved in two ways: users can either create a custom Scenario object by uploading a JSON file with their specific parameters or opt for the more straightforward method of utilizing pre-built scenario initiation scripts. The Scenario object is designed to read and incorporate all program options, defaulting to built-in values for any parameters not specified in the file or manually by the user. An initial step involves the Scenario

object attempting to capture all provided variables to determine whether the input file sufficiently covers the necessary parameters. This can be seen in **Figure 4.9** This evaluation is necessary as it ensures the program is aware of the available variables for subsequent calculations. If the inputs fall short of the requirements, a 'sanity check' function, executed later in the process, will identify this and stop the program.

```
for phase in range(parameters['Stage']):
    print(phase + 1, '. phase is initializing.')

    self.Defined_M0[phase] = self.isGiven(parameters, 'm_0', phase, islist=True)
    self.Defined_MF[phase] = self.isGiven(parameters, 'm_f', phase, islist=True)
    self.Defined_ISP[phase] = self.isGiven(parameters, 'I_sp', phase, islist=True)
    self.Defined_M_DOT[phase] = self.isGiven(parameters, 'm_dot', phase, islist=True)
    self.Defined_FAIRING = self.isGiven(parameters, 'm_fairing', phase, islist=False)
    self.Defined_STAGE = self.isGiven(parameters, 'stage', phase, islist=False)
    self.Defined_THRUST[phase] = self.isGiven(parameters, 'thrust', phase, islist=True)
    self.Defined_BUNDLE[phase] = self.isGiven(parameters, 'bundle', phase, islist=True)
    self.Defined_T_PROFILE[phase] = self.isGiven(parameters, 'thrust_profile', phase, islist=True)
    self.Defined_T_MOD[phase] = self.isGiven(parameters, 'thrust_mod', phase, islist=True)
    self.Defined_T_PROGRAM = self.isGiven(parameters, 'thrust_program', phase, islist=False)
    self.Defined_GUDUM_MOD[phase] = self.isGiven(parameters, 'guidance_mod', phase, islist=True)
    self.Defined_LULE_CAP[phase] = self.isGiven(parameters, 'D_exit', phase, islist=True)
    self.Defined_TW[phase] = self.isGiven(parameters, 'T_W', phase, islist=True)

    self.Defined_LAUNCH_STATE = self.isGiven(parameters, 'X_Start', phase, islist=False)
    self.Defined_LAUNCH_TIME = self.isGiven(parameters, 'launch_time_utc', phase, islist=False)
    self.Defined_LAUNCH_AZIMUTH = self.isGiven(parameters, 'launch_azimuth', phase, islist=False)
    self.Defined_AERODB[phase] = self.isGiven(parameters, 'aero_database', phase, islist=True)
    self.Defined_INTERP_MOD[phase] = self.isGiven(parameters, 'interpolation_mod', phase, islist=True)
    self.Defined_AERO_MOD[phase] = self.isGiven(parameters, 'Aero_Mod', phase, islist=True)

    self.Defined_TARGET_APOGEE = self.isGiven(parameters, 'target_apogee', phase, islist=False)

    self.Defined_NAME = self.isGiven(parameters, 'name', phase, islist=False)
    self.Defined_DEL_T[phase] = self.isGiven(parameters, 'del_t', phase, islist=True)
    self.Defined_TC_ALTITUDE = self.isGiven(parameters, 'TC_Altitude', phase, islist=False)
    self.Defined_T_START[phase] = self.isGiven(parameters, 't_start', phase, islist=True)
    self.Defined_T_SIM[phase] = self.isGiven(parameters, 't_sim', phase, islist=True)
    self.Defined_PHASE_NUM = self.isGiven(parameters, 'phase_num', phase, islist=False)
    self.Defined_SEPARATION_OFFSET[phase] = self.isGiven(parameters, 'separation_offset', phase, islist=True)
    self.Defined_RECORD_MOD[phase] = self.isGiven(parameters, 'record_mod', phase, islist=True)
    self.Defined_OUTPUT_FOLDER = self.isGiven(parameters, 'output_folder', phase, islist=False)
    self.Defined_OPTIM_PARAMS_CSV[phase] = self.isGiven(parameters, 'optim_params_csv', phase, islist=True)
    self.Defined_FALL_SIMULATION[phase] = self.isGiven(parameters, 'fall_simulation', phase, islist=True)
```

Figure 4.9 : Variable checking flags for sanity check.

Upon loading the inputs, the Scenario object organizes them into a Python dictionary, streamlining easier access and reference throughout the simulation. The next step involves a compilation process where the program evaluates the provided inputs, specifically focusing on parameters related to thrust, mass, simulation dynamics, launch positioning, guidance systems, physics, and general settings. This compilation phase is not only about gathering data but also involves a thorough validation of the input values. The program systematically checks to ensure that all inputs are within

acceptable ranges and conform to expected formats. This process is vital for preempting potential issues that might arise during the simulation, thereby safeguarding against errors that could compromise the integrity of the results and also prevents the user spending time waiting for a faulty simulations result.

Upon receiving and structuring the necessary raw parameters within a Python dictionary, the Scenario object executes preliminary calculations needed for preparing the simulation. This preparatory step, referred to as compiling the scenario, is shown in **Figure 4.10**. The essence of this function is to decipher the specifics of the simulation scenario.

```
def compileScenario(self):
    self.compileThrust()
    self.compileMass()
    self.compileSimulation()
    self.compileLaunch()
    self.compileGuidance()
    self.compilePhysics()
    self.compileSettings()
    self.COMPILE_FLAG = True
    self.n = self.parameters['optim_params'][0].shape[0]
    self.events = { # CONTAINERS FOR EVENTS
        "t_FYK_ejection": np.zeros(self.n),
        "h_FYK_ejection": np.zeros(self.n),
        "MACH_GUD_BAS": np.zeros(self.n),
        "t_pitch_start": np.zeros(self.n),
        "t_pitch_end": np.zeros(self.n),
        "h_pitch_start": np.zeros(self.n),
        "Drag Acc @ Sep": np.zeros(self.n),
        "MECO Altitude": np.zeros(self.n),
        "Separation Alt": np.zeros(self.n),
        "SE Burn": np.zeros(self.n),
        "misc": {}
    }
    print('Compilation Finished')
```

Figure 4.10 : Scenario compilation function.

For instance, a user may define a nominal engine burn time of 150 seconds with a stage separation duration of 5 seconds, coupled with instructions for a thrust throttle down once the dynamic pressure hits 35 kPa. A throttle down indicates a reduced fuel consumption rate during this phase, resulting in a surplus of fuel for subsequent burning, which in turn extends the engine's burn time. Such complexities necessitate pre-calculations to determine the total simulation time for the first stage flight, taking

into account the nominal burn time, throttling activity, and separation timing. These pre-calculations are a fundamental part of the scenario compilation process.

As an additional example, the guidance parameters' search space for input via a JSON file can make the JSON overcrowded and unhandy, often containing over 6000 lines and four columns. Therefore, these parameters are stored in a separate CSV file, which are more manageable for tabular data. The scenario JSON specifies the file path for these values. The guidance compilation within the Scenario object verifies the guidance modes and the file path for the guidance parameter search space, subsequently loading this data into a designated dictionary for easy access during the simulation. A small code snippet of the guidance compilation code is presented in **Figure 4.11**.

```
def compileGuidance(self):
    self.parameters['optim_params'] = [False] * self.parameters['Stage']
    for phase in range(self.parameters['Stage']):
        if self.Defined_Optim_Params_CSV[phase]:
            optim_params_csv = self.parameters['optim_params_csv'][phase]
            self.parameters['optim_params'][phase] = (
                pd.read_csv(optim_params_csv, dtype=np.float64, header=None)).values
            self.parameters['optim_params'][phase][:, 2] = np.deg2rad(self.parameters['optim_params'][phase][:, 2])
            if self.parameters['optim_params'][phase].shape[1] >= 4:
                self.parameters['optim_params'][phase][:, 3] = np.deg2rad(
                    self.parameters['optim_params'][phase][:, 3])
```

Figure 4.11 : Code snippet from guidance scenario compiler.

After the compilation of the scenario, the simulation is ready to be executed and the compiled scenario is inputted into a Simulator object defined in the next chapter.

4.3.2 Simulator

The Simulator object functions as the central processor in the simulation framework. Its primary role encompasses the instantiation of Stage objects in line with the predefined Scenario and orchestrating the simulation's progression through each stage. Furthermore, the Simulator is responsible with encapsulating the generated data and events, efficiently storing them in a binary format optimized for swift post-processing access. An example of the Simulator object's functionality, as shown in the provided code snippet in **Figure 4.12**, demonstrates how it constructs the stages and runs the simulation forward stage by stage.

```

def runSimulation(self):
    X = deepcopy(self.X_Start)
    t_bas = deepcopy(self.Scene.parameters['t_bas'][0])

    if self.Scene.COMPILE_FLAG == False:
        assert ValueError('Scenario must be compiled first!')
    for i in range(1, self.phase + 1):
        self.buildStage(X = X, t_bas = t_bas, phase = i)
        self.runStage()
        X = self.current_stage.X_JD[:, -1, :]
        t_bas = self.Scene.parameters['t_sim'][i - 1]
        print('t_bas:', t_bas)

```

Figure 4.12 : Code snippet from Simulator object.

The Simulator object is responsible for activating the essential physical models that drive the simulation forward. This involves key elements like aerodynamics, thrust, mass, gravity, guidance, atmosphere, and flight mechanics, all of which are outlined in section 3.3. Initially, the Stage objects start with state variables set to zero, but as the simulation progresses, these variables are populated with data by the Simulator.

Once the simulation concludes, the Simulator merges the separate state vectors, which represent the states of each stage, into one comprehensive array each. These arrays are then saved using Numpy's binary format for efficient data handling later. The code snippet for this process can be seen in **Figure 4.13**.

```

def concatFlight(self):
    Stages = self.Stages
    X_ECI = [None] * len(Stages)
    X_JD = [None] * len(Stages)
    X_LDBARV = [None] * len(Stages)
    m = [None] * len(Stages)
    gud_aci = [None] * len(Stages)
    D = [None] * len(Stages)
    L = [None] * len(Stages)
    T = [None] * len(Stages)
    q = [None] * len(Stages)
    a_a = [None] * len(Stages)
    i = 0
    for stage in Stages:
        X_ECI[i] = stage.X_ECI
        X_JD[i] = stage.X_JD
        X_LDBARV[i] = stage.X_LDBARV
        m[i] = stage.m[:, 0:-1]
        gud_aci[i] = stage.gud_aci
        D[i] = stage.D
        L[i] = stage.L
        T[i] = stage.T
        q[i] = stage.q
        a_a[i] = stage.axial_acc
        i = i + 1

    self.container['X_ECI'] = np.concatenate(X_ECI, axis = 1)
    self.container['X_JD'] = np.concatenate(X_JD, axis = 1)
    self.container['X_LDBARV'] = np.concatenate(X_LDBARV, axis = 1)
    self.container['m'] = np.concatenate(m, axis = 1)
    self.container['gud_aci'] = np.concatenate(gud_aci, axis = 1)
    self.container['D'] = np.concatenate(D, axis = 1)
    self.container['L'] = np.concatenate(L, axis = 1)
    self.container['T'] = np.concatenate(T, axis = 1)
    self.container['q'] = np.concatenate(q, axis = 1)
    self.container['Ax_Acc'] = np.concatenate(a_a, axis = 1)

    self.saveContainer()

```

Figure 4.13 : Data arranging code snippet from Simulator object.

4.3.3 Stage

The last object in the vectorized simulation software is the Stage. It is used as data storage for the state vectors of a stage. Some of the many state variables that are stored in the stage object can be seen in the **Figure 4.14**.

```

self.gvd_aci = np.zeros(shape=(self.n, self.n_2))

self.T = np.zeros(shape=(self.n, self.n_2)) # [N]
self.m_at_yak = np.zeros(shape=(self.n, self.n_2)) # [kg]

if self.RECORD_MOD == 'full':
    # Aerodinamik Kuvvetler acisindan ("AERODYNAMICS")
    self.D = np.zeros(shape=(self.n, self.n_2)) # [N]
    self.L = np.zeros(shape=(self.n, self.n_2)) # [N]
    self.q = np.zeros(shape=(self.n, self.n_2))
    self.axial_acc = np.zeros(shape=(self.n, self.n_2))

    print('Full record mode is activated: Aero, mass, thrust, guidance states will be saved.')
self.m = np.zeros(shape=(self.n, self.n_2 + 1)) # [kg]
self.X_JD = np.zeros(shape=(self.n, self.n_2 + 1, 6)) # [rad;rad;m/m/sn;rad;rad]
self.X_LDBARV = np.zeros(shape=(self.n, self.n_2 + 1, 6)) # [rad;rad;m/m/sn;rad;rad]
self.X_ECI = np.zeros(shape=(self.n, self.n_2 + 1, 6)) # [m/sn;m/sn;m/sn;m/m]
self.X_start = X

self.initStateVars()

self.mach_current = np.zeros(self.n)
self.density_current = np.zeros(self.n)
self.atmpressure_current = np.zeros(self.n)

```

Figure 4.14 : Code snippet from Stage object initialization.

The Stage object, similar to the Scenario object, is frequently passed throughout different parts of the software for easier access overall. Passing it over to a function streamlines the process of reading data consistently across various functions. The object carries not only physical variables like ECI position, ECI velocity, drag, lift, and dynamic pressure, but it also holds flag variable arrays. These flags indicate the occurrence of events or the initiation of different phases during the flight. Some of these flag state variables can be seen in **Figure 4.15**.

Passing this object into functions is efficient and it doesn't hinder the program's speed. Unlike in MATLAB and certain other environments, Python does not duplicate the object's data when it's passed to a function. Instead, Python passes the object's reference—a pointer to its memory address—allowing the function to access the object directly. This method, known as 'pass by reference,' requires careful use since it makes the object mutable within any function it is passed to, which can be advantageous for flexibility but risky if unintended changes are made.

```

self.STATES = {'COAST': False}
if phase == 1:
    self.STATES['LIFT_OFF_STATE'] = np.zeros(self.n, dtype=bool)
    self.STATES['THRUST_PROGRAM_STATE'] = np.zeros(self.n, dtype=bool)
    self.t_bas = 0
    self.STATES['GUDUM_STATE'] = np.zeros(self.n, dtype=bool)
    self.STATES['GUD_BAS'] = np.zeros(self.n, dtype=float)

if phase != 1:
    self.t_bas = Scene.parameters['t_bas'][idx]

if case == 2 and self.Scene.DEFINED_FYK:
    self.STATES['FYK_ATILDI'] = np.zeros(self.n, dtype=bool)

print('\n\n', self.name, "kademe ", case, " baslatiliyor.")

if case == 1 and self.Scene.DEFINED_T_PROGRAM:
    if self.Scene.parameters['T_program'] == 'Under_Throttle_At_Q_lim':
        # if ITKI_PROGRAMI == True:
        self.STATES['THRUST_PROGRAM_STATE_2'] = np.zeros(self.n, dtype=int)
        self.STATES['THRUST_PROGRAM_RECORDER'] = np.zeros(self.n,
                                                            dtype=float)
        self.STATES['THRUST_PROGRAM_Q_T_END'] = np.zeros(self.n, dtype=float)

```

Figure 4.15 : Code snippet depicting state flags.

4.4 User Interaction

4.4.1 Search space input

Inputting the search space for the program is conducted through a Comma-Separated Values (CSV) file. This file format uses commas to separate columns and each new line represents a different entry. The total number of lines in the CSV file dictates how many simulations will be executed in a given session. Presently, the program requires four inputs for the search space: three are for detailing the first stage's pitch-over maneuver, and the fourth input specifies the maneuver for the second stage, which commences after the payload fairing is jettisoned.

A standard CSV file is provided for user convenience. Given the relatively fixed range of options for the first stage maneuver, users don't need to modify these parameters for each new session. The CSV file with search space parameters is saved in a dedicated folder, named after the session, to enhance trackability. This file location is noted in the session's scenario file.

Additionally, a supportive spreadsheet file has been created to help users explore the search space and gauge the scale of the simulation beforehand. This file displays the anticipated number of simulations and the expected RAM usage, assisting in situations where system memory might be limited. Users can then adjust the search space accordingly. The spreadsheet, illustrated in **Figure 4.16**, requires users to input data in the blue boxes, while the orange boxes display the outcomes. It calculates all possible combinations of the three given parameters, considering their ranges and step sizes, and excludes any combinations where the MACH_PITCHOVER_START is greater than the MACH_PITCHOVER_END parameter.

	A	B	C	D	E	F
1		First Value	Last Value	Step size	Dimension	Unit
2	MACH_PITCHOVER_START	0,05	0,25	0,05	5	-
3	MACH_PITCHOVER_END	0,1	0,8	0,05	15	-
4	ALPHA_MAGNITUDE	0	4	0,05	81	deg
5				Pseudo_n	6075	-
6	Number of vectorized simulations			Real n	5265	-
7						
8						
9		Value	Unit			
10	Number of parallel Process	1	-			
11	dt	0,1	sec			
12	num_channel_propagated	25	-			
13	Simulation time	600	sec			
14	Number of timesteps	6000	-			
15	Estimated minimum RAM usage	7290	MB			

Figure 4.16 : Helper spreadsheet for specifying the search space.

Typically, the Excel spreadsheet is unnecessary due to the program's robust default pitch-over parameters.

However, it's worth noting that the last parameter, governing the second stage flight maneuver, is easily adjustable by the user. The dynamic exploitation of this space search will be further detailed in Chapter 5.

4.4.2 Launch vehicle specification input

The user inputs launch vehicle specifications through a JSON file, referred to as the scenario file. JSON, known for its user-friendly format and ease of readability, is a popular choice for data interchange between different software applications. **Figure 4.17** shows a snippet of a scenario JSON file, with each vehicle's stage defined by the user in a sequential list format, as indicated by the square brackets. The ordering within these lists is stage-specific: the first item corresponds to the first stage, the second to

the next, and so on. Within this file, users specify crucial parameters like the mass range for each stage, thrust capacities, aerodynamic characteristics, designated output directories, and paths to search space data files and so on. This structured approach not only simplifies the input process but also aids in maintaining clarity and trackability of simulation data for later review.

```

1 - {
2 -   "Vehicle": {
3 -     "m_0": [
4 -       12750,
5 -       2800
6 -     ],
7 -     "m_f": [
8 -       3750,
9 -       550
10 -    ],
11 -    "I_sp": [
12 -      311,
13 -      343
14 -    ],
15 -    "m_dot": [
16 -      7.142857142857143,
17 -      6.818181818181818
18 -    ],
19 -    "Stage": 2,
20 -    "Thrust": [
21 -      24000,
22 -      24000
23 -    ],
24 -    "bundle": [
25 -      9,
26 -      1
27 -    ],
28 -    "T_profile": [
29 -      null,
30 -      null
31 -    ],
32 -    "T_mod": [
33 -      "CONSTANT",
34 -      "CONSTANT"
35 -    ],
36 -    "nozzle_exit_diameter": [
37 -      0.2,
38 -      1
39 -    ],
40 -    "stage_diameter": [
41 -      1.2,
42 -      1.2
43 -    ]
44 -  },
45 -   "Simulation": {
46 -     "name": "Electron_new",
47 -     "del_t": [
48 -       0.1,
49 -       0.1
50 -     ],
51 -     "Tower_Clearance_Alt": 400,
52 -     "t_start": [
53 -       0,
54 -       null
55 -     ],
56 -     "t_sim": [
57 -       null,
58 -       null
59 -     ],
60 -     "phase_num": 2,
61 -     "phase_triggers": "Default",
62 -     "seperation_offset": [
63 -       0,
64 -       0
65 -     ],
66 -     "record_mod": [
67 -       "full",
68 -       "full"
69 -     ],
70 -     "output_folder": "outputs",
71 -     "optim_params_csv": [
72 -       "input/Electron_new/_p_1.csv",
73 -       null
74 -     ]
75 -   },
76 -   "Physics": {
77 -     "w": 0.0000729212,
78 -     "f": 0,
79 -     "R_D": 6378137,
80 -     "MU": 3986000000000000,
81 -     "g_0": 9.8065,
82 -     "Atm_Table": "databases/US_Athmo",
83 -     "Physics_Mod": "Custom"
84 -   },
85 -   "Launch_Init": {
86 -     "X_Start": [
87 -       -39.2615,
88 -       177.864876,
89 -       0,
90 -       0,
91 -       90,
92 -       0
93 -     ],
94 -     "launch_Date": {
95 -       "AN_Fir": [
96 -         2027,
97 -         10,
98 -         29,
99 -         1,
100 -        1,
101 -        1
102 -       ]
103 -     },
104 -     "Aerodynamics": {
105 -       "aerodb": [
106 -         "databases/aero_stage_1.mat",
107 -         "databases/aero_stage_2.mat"
108 -       ],
109 -       "interp_dim": [
110 -         3,
111 -         3
112 -       ],
113 -       "Aero_Mod": [
114 -         "LOOKUP",
115 -         "LOOKUP"
116 -       ]
117 -     },
118 -     "Guidance_Program": {
119 -       "guidance_program": [
120 -         "tiltedsine2",
121 -         "CPR"
122 -       ],
123 -       "guidance_on": [
124 -         true,
125 -         true
126 -       ]
127 -     }
128 -   }
129 - }

```

Figure 4.17 : Snippet from a Scenario JSON file.

Additionally, there is a script available that translates data from an Excel spreadsheet into this JSON format, providing an automated and user-friendly option for creating these scenario files. This process, including the script's functionality, will be further explained in Chapter 5.

4.4.3 Flight data output

The comprehensive flight data, containing state vectors like ECI coordinates and velocities, as well as forces including gravity, thrust, aerodynamics, dynamic pressure, and angle of attack profiles, are all stored in Numpy's binary format. This format includes the values from every simulation at each timestep, which, due to the extensive nature of the data, requires an efficient storage method to facilitate quick and manageable read and write operations. These data are stored to be post processed later on to examine the results of the flight simulations.

5. BRUTE FORCE LAUNCH VEHICLE ASSESMENT METHODOLOGY

5.1 Exploring Trajectories

Considering the endless number of potential trajectories available to reach a given orbit, particularly when involving an intermediate transfer orbit, the selection process becomes complex. For instance, to attain a circular orbit of 1000 x 1000 km, one might choose a direct route, which tends to be costly and inefficient, or alternatively, enter a transfer orbit with a perigee of -1000 km, then perform an apogee kick burn at the apogee. Through scanning a broad spectrum of flight corridor, we also explore potential transfer orbits that precede the final desired orbit. Predicting the most efficient transfer orbit is challenging; thus, employing a vectorized simulator is advantageous for pinpointing these viable trajectories.

The vectorized simulation program executes a vast number of distinct maneuvers for the initial stage of flight, leading the vehicle to unique flight trajectories for every simulation that is run. The optimal maneuvers that satisfy the specified constraints are then systematically paired with thousands of varied maneuvers for the second stage. This pairing extends the successful flight trajectories initiated by the first stage.

For the second stage, maneuvers are designed to minimize steering losses, mirroring the guidance profiles seen in conventional launch vehicles. Optimal thrust direction in the vacuum is typically a near-linear function of time. A passage from the Saturn V flight manual (NASA, 1969) underscores this linearity:

“The optimizing techniques of the calculus of variations were used to develop the iterative path adaptive guidance mode. Experience with hundreds of minimum propellant trajectories for various orbital injection missions has demonstrated that the optimum thrust direction relative to the local vertical is very nearly a linear function of time during vacuum flight. Moreover, the size of the angle between the optimum thrust direction and the local horizon is never very large for an orbital mission.”

This supports the notion that a linearly evolving thrust angle is highly effective for orbital missions, which can also be further explored in **Figure 5.1**.

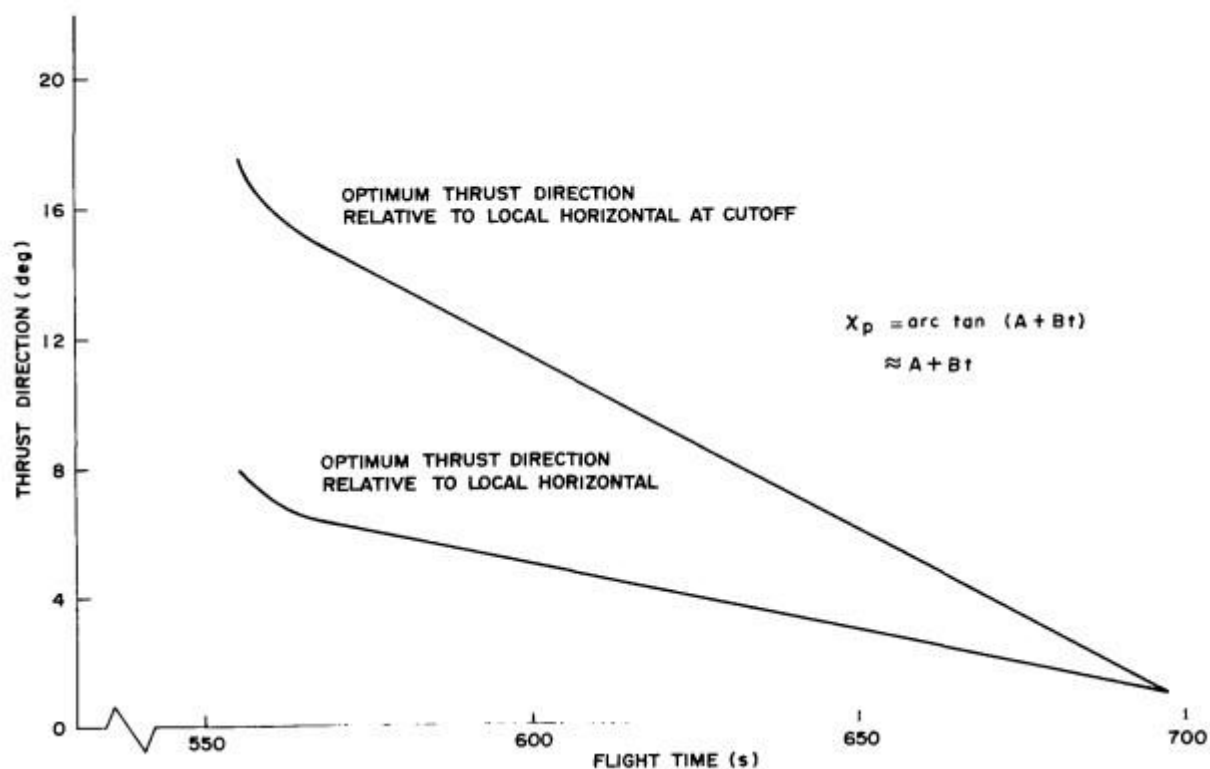


Figure 5.1 : Optimum thrust direction in Saturn V vacuum flight (Haeussermann, 1970).

Table 5.1 lays out the comprehensive search space, cataloging maneuvers that creates the flight trajectories.

Table 5.1 : Maneuver search space.

PITCH-OVER MANEUVEUR	Pitch over start Mach number	0.08 – 0.4
	Pitch over finish Mach number	0.1 – 0.8 Mach
	Pitch over alpha amplitude	0 – 4 degrees
SECOND STAGE MANEUVEUR	Constant alpha change	-0.15 – 0.15 deg/s
TOTAL	12000 different trajectories.	

5.2 Assessment Methodology

5.2.1 Flight constraints

Numerous flight constraints, often referred to as filters in this thesis, are imposed on the extensive array of simulation outcomes generated by the vectorized simulator. In the case of the first stage, four standard constraints for launch vehicles are employed. As will be evident in the results section, the most efficient configurations will be significantly far from these constraint thresholds.

Regarding the second stage, the filters used for the first stage are reapplied due to the possibility of the vehicle performing lofting maneuvers and approaching the boundaries of the atmosphere. Subsequently, orbital constraints are enforced to assess whether the vehicle has effectively achieved insertion into the transfer orbit or directly reached the intended orbit. The mission constraints can be examined in **Table 5.2**.

Table 5.2 : Mission constraints.

Mission Constraints	
1st Stage Constraints	
Dynamic Pressure (Max Q)	$\leq 50 \text{ KPA}$
Q*alpha product	$\leq 40 \text{ KPA}^\circ$
Axial Acceleration (Max g)	$\leq 8g$
MECO Altitude	$\geq 65 \text{ km for Falcon 9}$ $\geq 72 \text{ km for Electron}$
2nd Stage Constraints	
Dynamic Pressure (Max Q)	$\leq 50 \text{ KPA}$
Q x alpha product	$\leq 40 \text{ KPA x }^\circ$
Axial Acceleration (Max g)	$\leq 8g$
Successful Transfer Orbit Injection	1.Perigee $\geq 160 \text{ km}$ or 2.Altitude at orbit injection $\geq 160\text{km}$ and the vehicle flies to the apogee
Transfer Orbit Apogee	Target apogee error $\leq \%2$
Transfer Orbit Inclination	Target inclination error $\leq \%1$

5.2.2 Approach

Utilizing the vectorized simulator for searching transfer orbits is advantageous, as this is where the program's rapid processing capability shines, allowing for the examination of numerous potential transfer orbits.

In this approach, the user starts by detailing the launch vehicle's specifications in a format compatible with the scenario file. Subsequent steps involve defining the flight and final orbit injection constraints, using an initiation file that matches the input data format. Scripts then automate the preparation of scenario and maneuver space input files, placing them into their respective directories, readying the simulator for its dynamic search for the optimal payload and maneuver combinations.

The objective is to explore a wide array of trajectories through the application of various maneuvers, calculating the maximum payload that can be delivered to the target orbit while satisfying established constraints within the generated trade space.

Initially, the simulator assesses 6079 different first-stage maneuver combinations. The outputs are then processed, applying filters for flight constraints such as maximum dynamic pressure, maximum G-forces, and minimum stage separation altitudes, as outlined in the previous chapter. Flights that do not meet these criteria are discarded, leaving a subset of trajectories that comply with the first-stage requirements. Then, these remaining paths are ranked based on their 'gained specific mechanical energy' at the end of the first stage, a metric indicative of the flight's efficiency, as the aim is to minimize energy losses throughout the ascent.

$$E_{gained} = \left(\frac{v_{initial}^2}{2} - \frac{\mu}{r_{initial}} \right) - \left(\frac{v_{final}^2}{2} - \frac{\mu}{r_{final}} \right) \quad (5.1)$$

Upon evaluating the first stage's flight performance, the program selects a representative sample, typically 20 samples from the first 300 most efficient trajectories. These are then combined with 300 distinct second-stage maneuvers, culminating in 6000 new simulations ready for execution. The program proceeds to simulate these combined trajectories in the second phase of its process.

Choosing 20 distinct trajectories from the top 300 most efficient ones during the first stage trajectory selection is a strategic decision. It's possible to simply pick the top 20

first-stage trajectories, a method that has been explored in this thesis. Yet, casting a broader selection over the design space offers several advantages. When first-stage constraints are applied and the outcomes are ranked by the mechanical energy attained at the end of the first stage, the leading results typically exhibit the least gravity losses, indicating a more horizontal flight path compared to less efficient trajectories. A more horizontal trajectory implies higher speed but lower altitude, which could pose challenges for launch vehicles with a less powerful second stage. Such second stages might struggle to achieve the necessary altitude for orbit since they primarily contribute to increasing horizontal velocity. In these cases, the first stage needs to adopt a more vertical trajectory to secure both the required altitude and upward momentum. This approach allows the vehicle, even if it begins to descend during the second stage, to afford the second stage sufficient time to acquire the necessary horizontal speed to avoid re-entry, a strategy known as "lofting." As demonstrated in the results section, the Falcon 9 LEO case study yielded lofted trajectories where the altitude first increased then decreased, providing the second stage an opportunity to compensate. Although the trajectories with the highest mechanical energy at MECO (Main Engine Cut Off) reached an altitude of 65 kilometers, the overall performance was better in trajectories with a MECO altitude of approximately 72-74 kilometers.

Following the second simulation run, a collection of 6000 potential trajectories is analyzed. At this phase, the program reapplies general constraints such as maximum dynamic pressure, G-force limits, and stage separation altitude. It's at this point that the approach diverges from the initial step, as orbit injection criteria are introduced, including constraints on transfer orbit injection altitude, perigee, and apogee ranges. The outcomes are then ranked based on their payload delivery potential.

If the simulated payload exceeds the user's initially inputted value, the program enters an iterative refinement phase, adjusting the payload and do the same steps. This iterative process, which also adjusts the launch azimuth for the desired orbital inclination, typically converges within two to three cycles at maximum, provided the initial payload estimate wasn't significantly off-target.

At the end of this iterative process, typically only requiring a few cycles, the final iteration's results are deemed to represent the true payload capacity deliverable to the target orbit.

The flow diagram of this process can be examined in **Figure 5.2**.

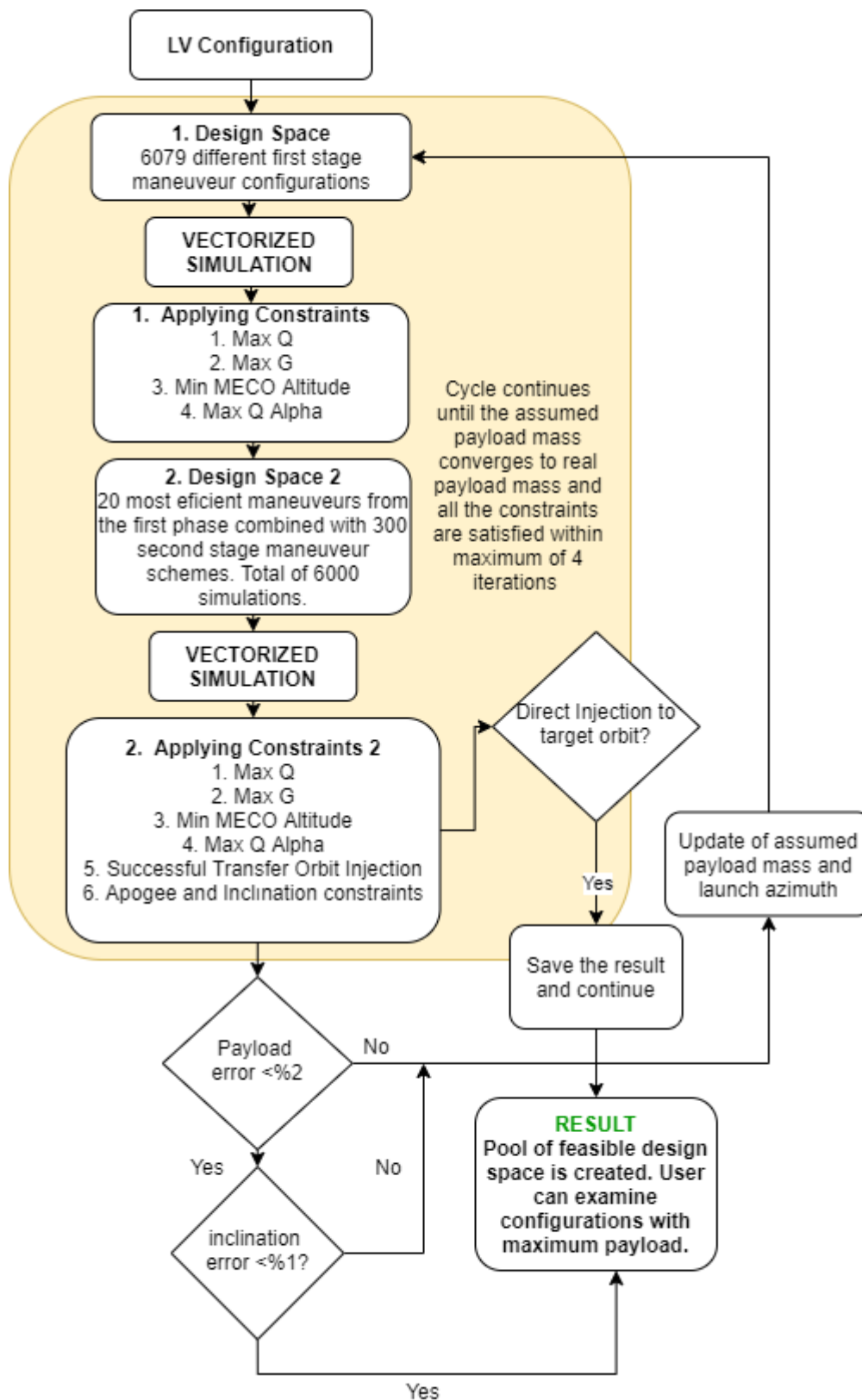


Figure 5.2 : Flow Diagram of the vectorized simulator.

6. RESULTS

6.1 Indirect Injection: Falcon 9

6.1.1 Vehicle properties

6.1.1.1 General properties

Gathering open-source information on the various iterations of the Falcon-9 rocket, such as Falcon 9 v1, v1.0, v1.1, and the subsequent Block 3, Block 4, and Block 5 versions, presents significant challenges. The most widely recognized among these is the Falcon FT (Full Thrust) Block 5. For the sake of simplicity, this research focuses on the Falcon 9 Block 5, leveraging the amount of flight videos and data available online. Nonetheless, pinpointing specific details like the Propellant Mass Fraction (PMF), fuel quantities for each stage, dry weights, and the first stage engines' nozzle exit diameters is a challenging task. The figures found online are often contradictory and confused with data from other versions of the vehicle. The data best matching the Falcon 9 Payload User's Guide (SpaceX, 2024) were used which are taken from Launcher (Launcher, 2023), Wevolver (Wevolver, 2023). However, to address some of the uncertain data, further assumptions have been made which are described in the paragraph below.

For instance, the nozzle's exit diameter was calculated by comparing the vacuum and sea-level thrusts of the engines and conducting a straightforward reverse engineering based on sea-level pressure thrust calculations. Estimating burn times is particularly challenging, as the Falcon-9's thrust can vary during flight, and the duration of the first stage burn can fluctuate between 140 to 163 seconds, influenced by factors such as the extent of throttle adjustment and whether the first stage is designed to land on a drone ship or return to the launch site or to be expended. Hence, burn times are deduced from video footage with similar missions and adjusted to align with the rocket's mass of 549 tons (excluding payload) and the delta-v requirements for the missions studied.

This chapter will delve into two specific missions involving the Falcon 9. Before detailing these missions, a table summarizing the Falcon 9's vehicle properties is provided for reference in **Table 6.1**. Within the table, certain attributes are marked as "pre-iteration" to indicate that these values are subject to change after the completion of the launch vehicle evaluation. Specifically, changes in payload capacity will lead to adjustments in the overall lift-off mass. Such changes in the gross lift-off mass will, in turn, affect the vehicle's Thrust to Weight (T/W) ratio as well as its delta-v capacity.

Table 6.1 : Falcon 9 assumed properties.

Falcon 9 FT Block 5 Assumed Properties			
	1 st mission	2 nd mission	Unit
Height	70	70	m
Diameter	3.66	3.66	m
Fairing Diameter	5.2	5.2	m
Assumed Payload (pre-iteration)	22000	7000	kg
Fairing mass	1700	1700	kg
GLOM (pre-iteration)	571322	556322	kg
T/W (pre-iteration)	1.3574	1.394	-
Delta V (pre-iteration)	9390	11934	m/s
1st stage			
Sea Level Thrust	7619	7619	kN
Single Engine Sea Thrust	845	845	kN
Vacuum Thrust	8226	8226	kN
Single Engine Vacuum Thrust	914	914	kN
Burn time	153	153	s
Vacuum ISP	313	313	s
PMF	0.948	0.948	-
Wet Mass	432520	432520	kg
Dry Mass	22491	22491	kg
Engine Bundle	9	9	#
Engine Exit Diameter	0.92	0.92	m
2nd Stage			
Vacuum Thrust	981	981	kN
Burn time	382	382	s
Vacuum ISP	348	348	s
PMF	0.954	0.954	-
Wet Mass	115102	115102	kg
Dry Mass	5291	5291	kg
Engine Bundle	1	1	#

6.1.1.2 Aerodynamic properties

To determine the aerodynamic characteristics of the vehicle, the Missile DATCOM (Rosema, Doyle, Auman, & Underwood, 2011) software was employed to generate an interpolation table based on Mach number and angle of attack (α).

This required modeling the vehicle's shape, as illustrated in **Figure 6.1**. The dimensions fed into the Missile DATCOM software's geometry input included the Falcon 9's 70-meter height, its fairing's maximum diameter of 5.2 meters, a detailed fairing profile, and the diameters of its stages, which measure around 3.66 meters.

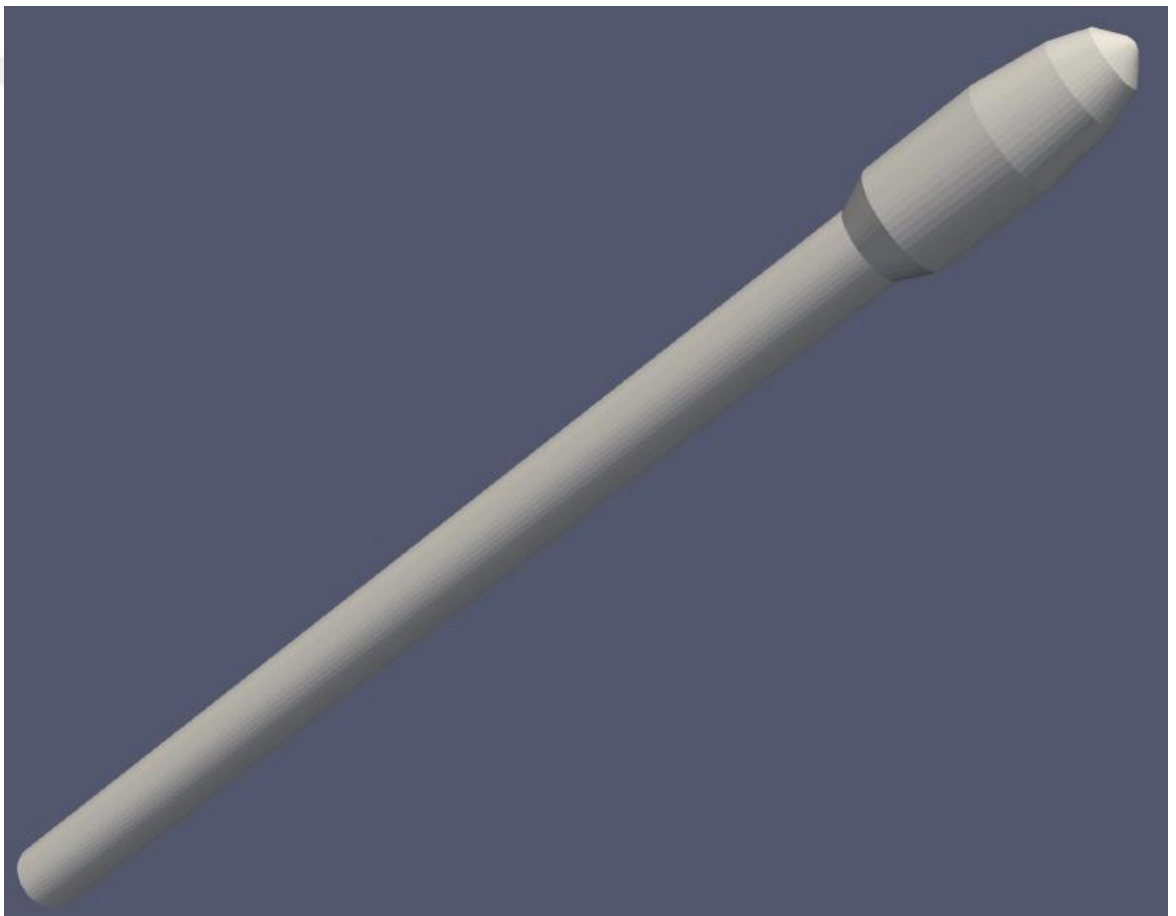


Figure 6.1 : Simple Falcon 9 geometry for aerodynamics.

While Missile DATCOM is a useful tool, it is not without its flaws and may produce some inaccurate values. Nevertheless, it serves as a practical approach during the initial stages of design for estimating aerodynamic coefficients. As illustrated in **Figure 6.3**, the drag coefficient for this model is notably higher than what is typical for conventional launch vehicles, possibly as a result of the fairing's design. Despite

this, the Falcon 9 is capable of withstanding higher drag coefficients because drag losses become less significant as the vehicle's mass increases. This increase in mass outpaces the growth in the vehicle's diameter according to the square-cube law.

Side by side comparison of the reference geometry model with real Falcon 9 can be seen in **Figure 6.2**.



Figure 6.2 : Falcon 9 reference geometry model comparison.

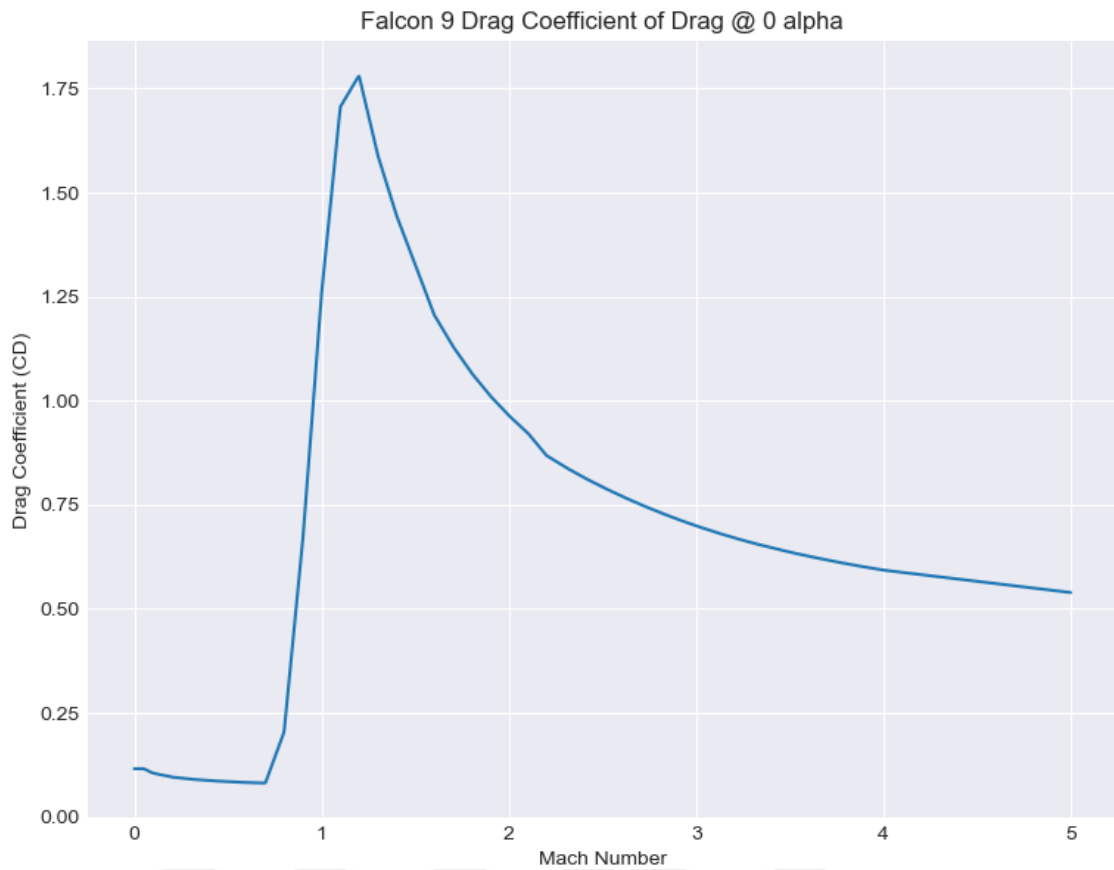


Figure 6.3 : Calculated drag coefficient of Falcon 9.

6.1.2 Missions

For the Falcon 9 simulations, we have selected two distinct missions for analysis: one targeting Low Earth Orbit (LEO) and the other targeting at a Super Geostationary Transfer Orbit (SGTO). The specifics of these missions' orbits are detailed in **Table 6.2**. These simulated missions are inspired by two actual Falcon 9 launches: the Starlink Group 6-35 mission (SpaceX, 2024) and Eutelsat 10B mission (SpaceX, 2022). It is important to note that these are designed as flights with expendable first stages.

Table 6.2 : Falcon 9 mission scenarios.

Falcon 9 FT Block 5 - Missions			
	1 st mission	2 nd mission	Unit
Orbit			
Orbit Type	LEO	SGTO	-
Target Apogee	293	59831	km
Target Perigee	284	261	km
Transfer Orbit Apogee	~284	~261	km
Transfer Orbit Perigee	Unbounded	Unbounded	km
Target Inclination	43	28.56	°
Launch Site			
Site	SLC-40, Cape Canaveral		-
Latitude	28.5621		°
Longitude	-80.5772		°
Altitude	0		m
Launch Azimuth	56.376	90	°

6.1.3 Statistics

Before moving into the results, it would be insightful to provide some statistics on the simulation runs and the impact of constraints on narrowing down the design space. The table below illustrates the step-by-step reduction of the design space, starting from the initial set and applying each constraint in sequence. It's important to note that a single iteration consists of initially conducting 6079 simulations focusing solely on the first stage maneuvers. Following this, designs are eliminated and ranked based on the orbital mechanical energy achieved, with sample from the top performers then combined with second stage maneuvers for further analysis with 6000 simulations as explained in section 5.2.2. This process explains the table's division into two parts: one detailing the eliminations after the first set of runs, and the other showing the reductions following the second set of simulations.

The statistical outcomes of the first mission simulations for the Falcon 9, aimed at achieving a Low Earth Orbit (LEO) with a desired perigee of 284 km and an apogee of 293 km, are presented in **Table 6.3**.

Table 6.3 : Falcon 9 LEO mission design space statistics.

LEO mission design space reduction	
1st run 1st stage eliminations	
Elimination	Number of simulations left
Pre-Elimination	6079
After Max Q	4545
After Max Q*alpha	4233
After minimum MECO altitude	3583
After max g	3396
2nd run Final Eliminations	
Elimination	Number of simulations left
Pre-Elimination	6000
After Max Q	5977
After Max Q*alpha	3742
After Max g	3742
Successful Transfer Orbit	1809
Injection	1665
After Transfer Orbit Apogee	1665
After Transfer Orbit	1665
Inclination	1665
Total Assessed Trajectories	12079
Total Successful Trajectories	1665

The data from the second set of mission simulations for the Falcon 9, targeting a Super Geostationary Transfer Orbit (SGTO) with a planned perigee of 261 km and an apogee of 59,831 km, are detailed in **Table 6.4**.

Table 6.4 : Falcon 9 SGTO mission design space statistics.

SGTO mission design space reduction	
1st run 1st stage eliminations	
Elimination	Number of simulations left
Pre-elimination	6079
After max Q	4775
After max Q*alpha	4427
After minimum MECO altitude	3878
After Max G	3779
2nd run final eliminations	
Elimination	Number of simulations left
Pre-Elimination	6000
After Max Q	6000
After Max Q*alpha	3758
After Max g	3758
Successful Transfer Orbit	2212
Injection	2030
After Transfer Orbit Apogee	2030
After Transfer Orbit	2030
Inclination	2030
Total Assessed Trajectories	12079
Total Successful Trajectories	2030

Figure 6.4 presents a trio of plots: the first displays the altitude profiles for all 6079 simulated flights, the second narrows down to those adhering to the constraints, and the third shows the trajectories selected for the subsequent stage of the process which are identified as having achieved the highest levels of mechanical energy during their ascent in the first stage.

Following this selection, each of the preferred first-stage trajectories is then combined with a set of 300 second-stage maneuvers. The altitude profiles for this refined design space are depicted in **Figure 6.5**. The patterns observed shows a tendency for the vehicle to fly a lofted trajectory—a strategy typically adopted when the second stage has a relatively lower power compared to the first stage (Edberg & Costa, 2022). This

lofting technique allows the first stage to reach a higher altitude, affording the second stage ample time to gain horizontal velocity and keep the vehicle on course above the atmosphere without the risk of re-entry.



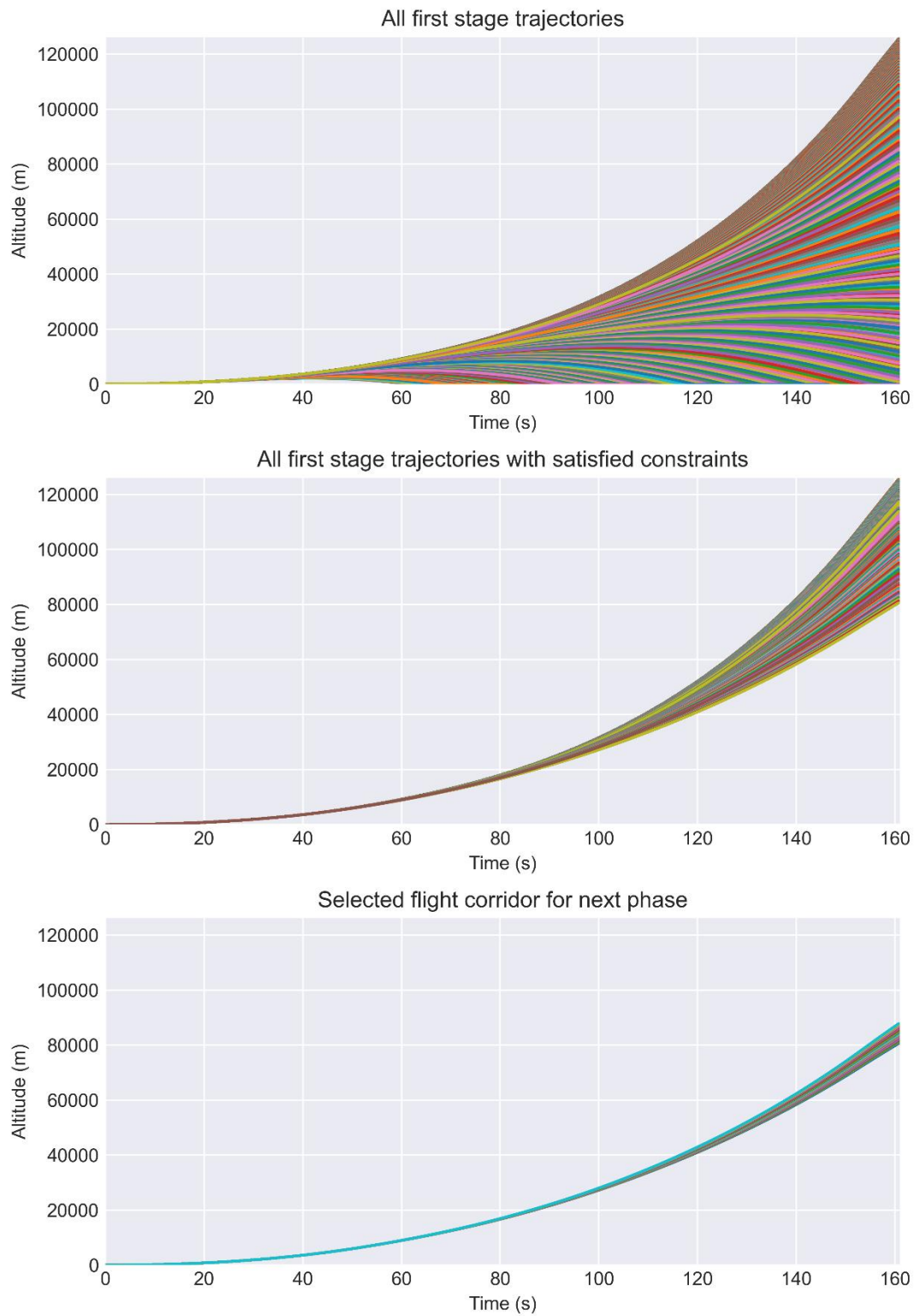


Figure 6.4 : Falcon 9 LEO mission first phase design space.

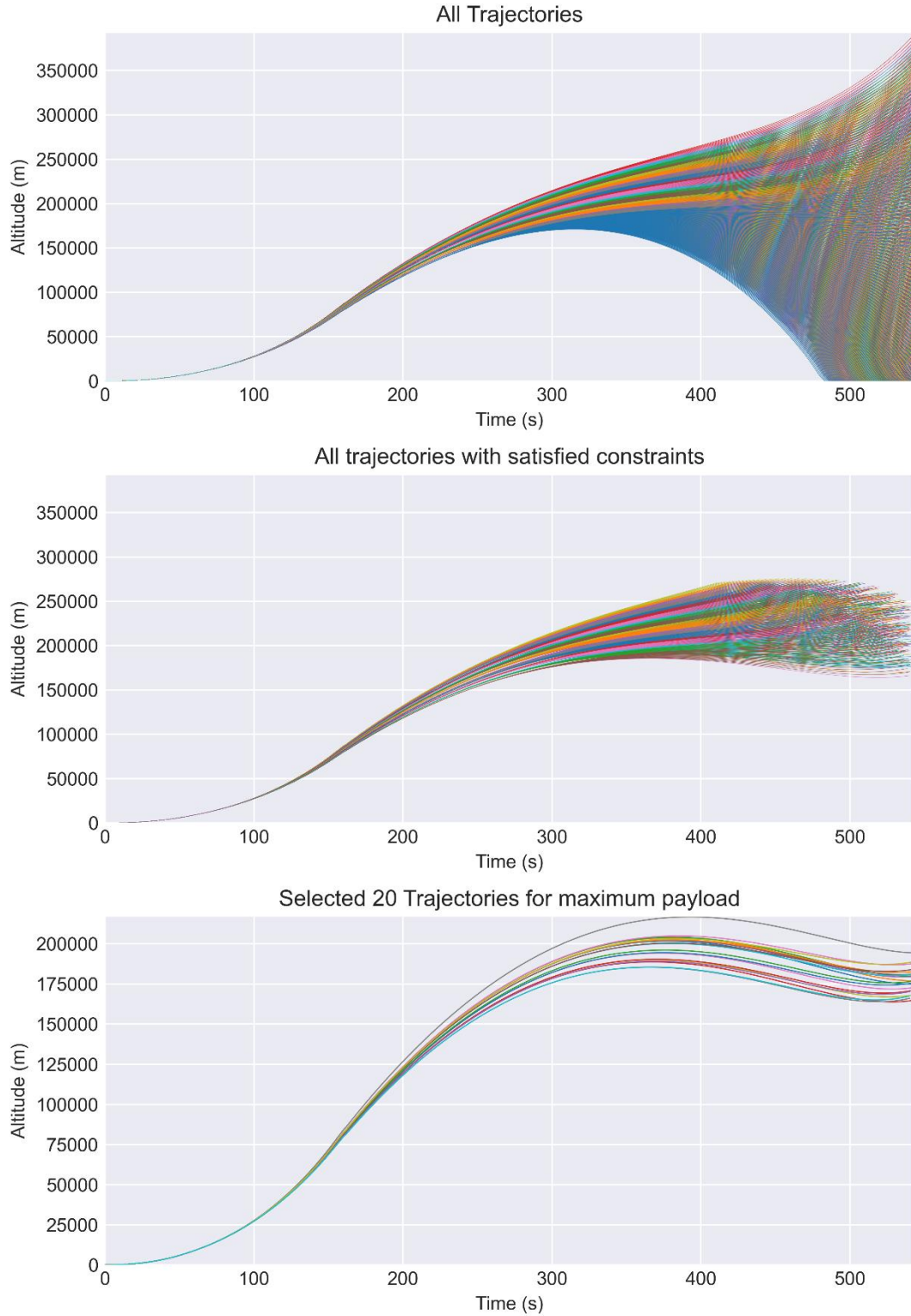


Figure 6.5 : Falcon 9 LEO mission final phase design space.

Lastly the same design space visualizations for the second mission, which is a super GTO mission, can be observed in **Figure 6.6** and **Figure 6.7**.

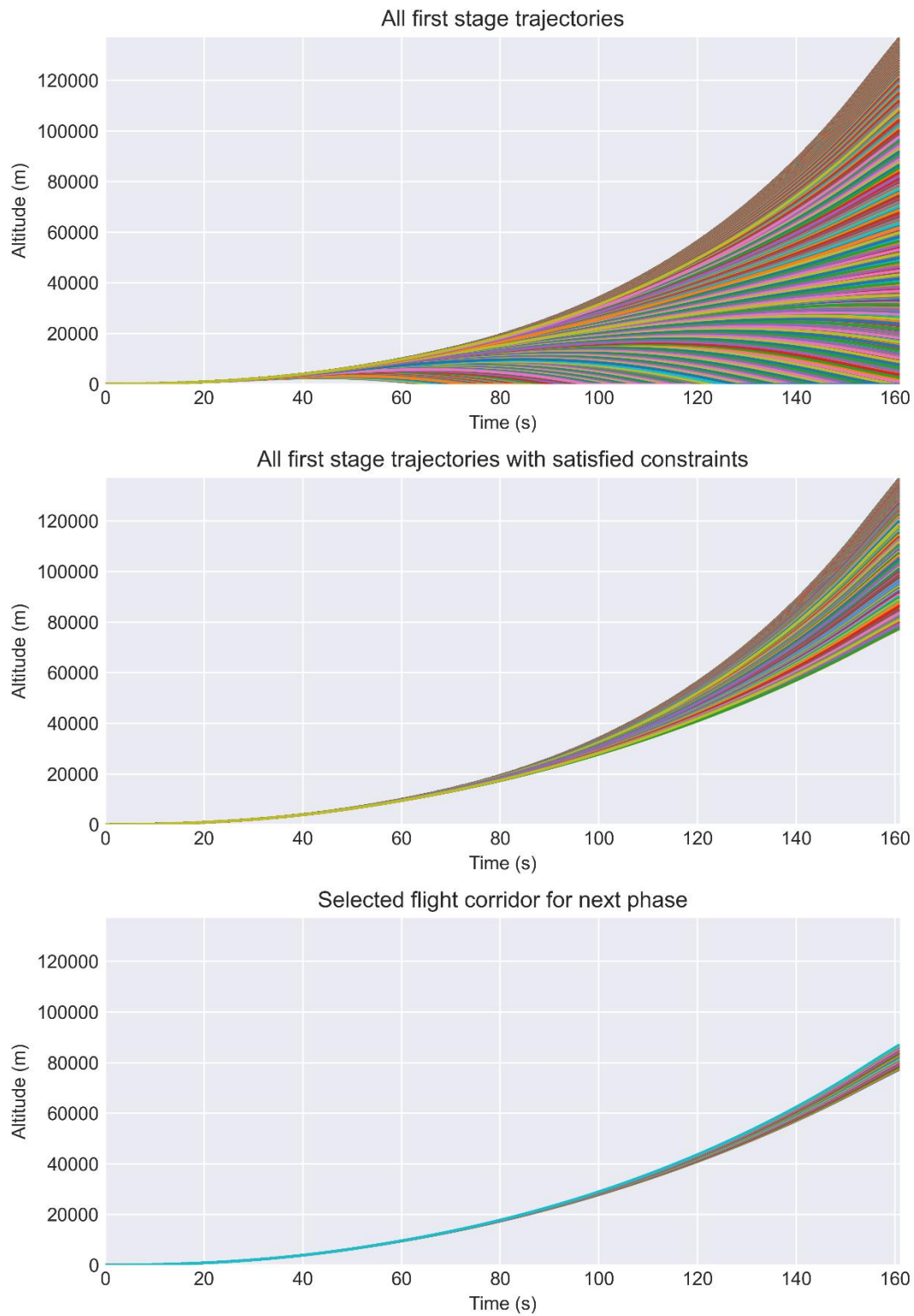


Figure 6.6 : Falcon 9 SGTO mission first phase design space.

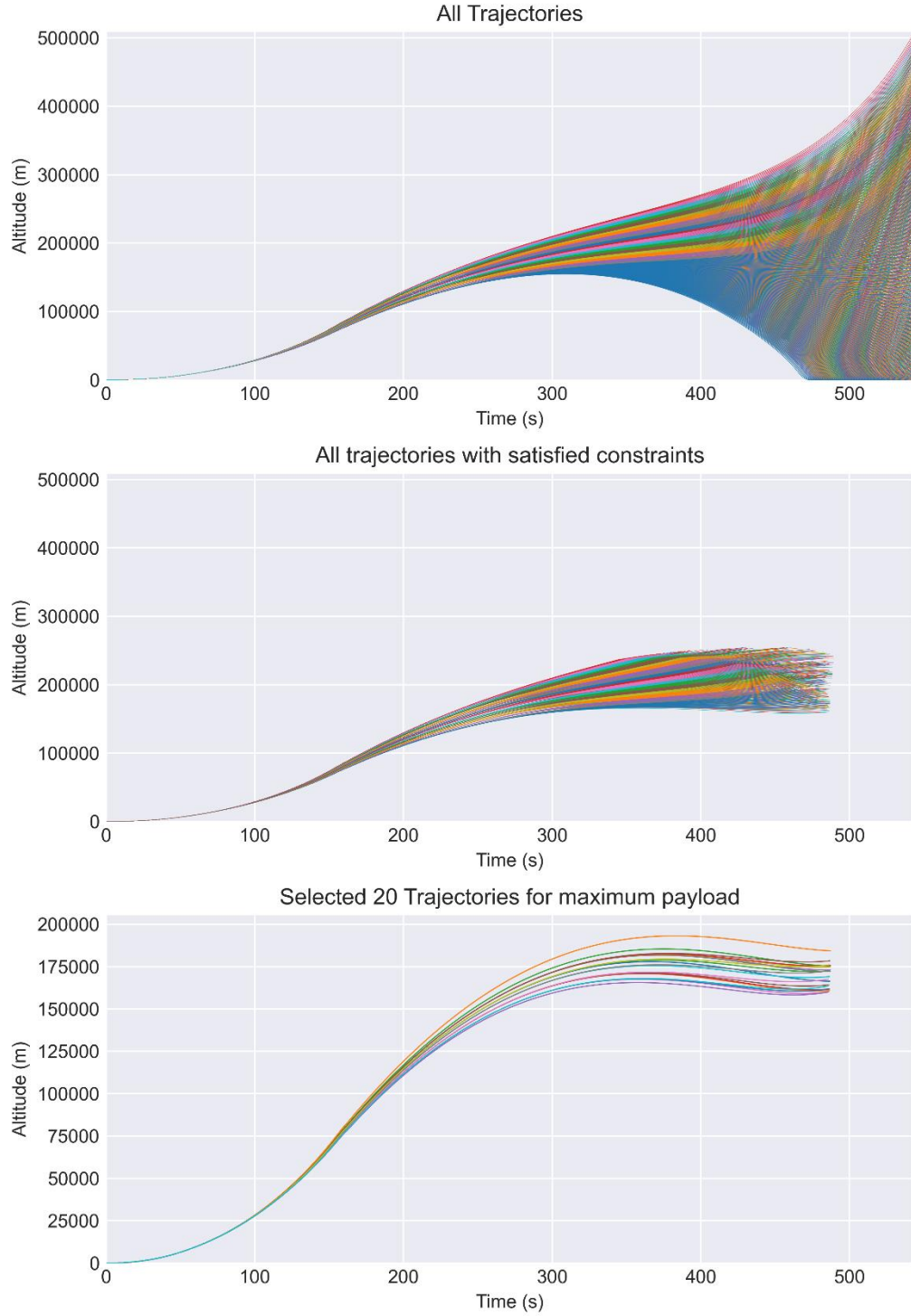


Figure 6.7 : Falcon 9 SGTO mission last phase design space.

6.1.4 Results

The summary of the top 20 performing flights results can be seen in **Table 6.5** while plots of the top performing flight can be examined in **Figure 6.8**.

Table 6.5 : Falcon 9 successful LEO missions - top 20 results.

Flight Index	Mach Start	Mach Finish	Alpha [°]	Constant Alpha Change Rate [°/s]	Delta V [m/s]	MECO Altitude [km]	Max Q [kPa]	Total Delta V Loss [m/s]	Payload Mass [kg]	Injection Altitude [km]	Transfer Orbit Apogee [km]	Transfer Orbit Perigee [km]	Transfer Orbit Inclination [°]	Inertial FPA @ Cut-off [deg]
4267	0.17	0.6036	1.4922	0.0425	9292	74.306	31.02	1731	22823	175.022	286.215	166.549	43.221	-0.094
4228	0.23	0.7345	2.6667	0.0411	9292	74.519	30.97	1735	22820	176.968	285.914	164.406	43.172	-0.211
4268	0.23	0.7345	2.6667	0.0425	9297	74.519	30.97	1745	22776	180.018	284.658	171.887	43.172	0.074
4582	0.14	0.8	1.1602	0.0532	9299	72.754	31.21	1731	22758	164.674	284.080	76.607	43.129	0.883
4327	0.17	0.6036	1.4922	0.0445	9303	74.306	31.02	1747	22731	179.585	286.822	159.866	43.221	0.325
4347	0.17	0.6036	1.4922	0.0452	9305	74.306	31.02	1753	22710	181.104	286.072	148.772	43.221	0.459
4504	0.11	0.6691	0.7149	0.0505	9305	73.385	31.14	1744	22708	172.630	286.009	90.230	43.178	0.830
4072	0.17	0.7345	1.4922	0.0358	9306	75.893	30.83	1771	22705	194.446	284.496	171.545	43.189	-0.337
4583	0.23	0.8	2.7436	0.0532	9306	72.873	31.16	1740	22703	167.955	285.167	57.282	43.130	0.992
4426	0.14	0.7345	1.1016	0.0478	9310	74.034	31.05	1759	22672	181.041	285.240	102.315	43.164	0.778
4544	0.11	0.6691	0.7149	0.0518	9311	73.385	31.14	1755	22665	175.542	284.202	39.207	43.177	1.072
4367	0.17	0.6036	1.4922	0.0458	9312	74.306	31.02	1758	22655	182.620	288.722	134.160	43.221	0.593
4505	0.2	0.6691	2.0769	0.0505	9314	73.562	31.10	1755	22642	176.714	287.592	68.071	43.180	0.956
4623	0.23	0.8	2.7436	0.0545	9314	72.873	31.16	1752	22633	170.822	285.924	3.448	43.129	1.230
4662	0.14	0.8	1.1602	0.0559	9316	72.754	31.21	1754	22622	170.373	285.223	-30.624	43.127	1.355
4466	0.14	0.7345	1.1016	0.0492	9317	74.034	31.05	1770	22617	183.998	284.921	50.266	43.163	1.024
4368	0.23	0.7345	2.6667	0.0458	9317	74.519	30.97	1772	22612	187.628	286.830	111.221	43.171	0.747
4741	0.2	0.6036	2.1282	0.0585	9321	72.304	31.26	1752	22574	166.123	287.861	-69.771	43.190	1.518
4447	0.17	0.6036	1.4922	0.0485	9323	74.306	31.02	1780	22568	188.591	285.631	32.888	43.218	1.091
4761	0.2	0.6036	2.1282	0.0592	9323	72.304	31.26	1758	22566	167.482	285.532	-106.869	43.189	1.622

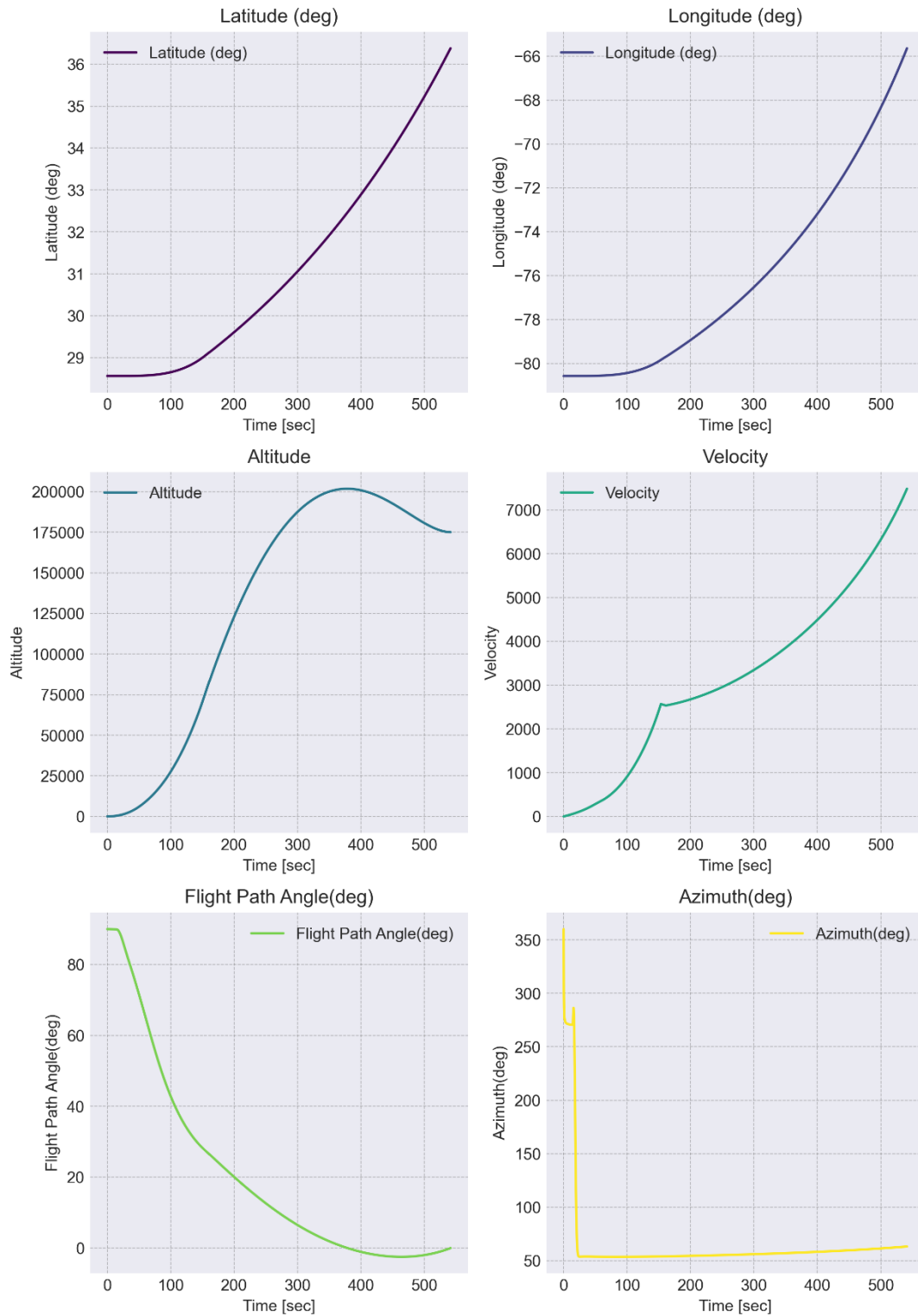


Figure 6.8 : Plots of the most successful flight.

Table 6.5 indicates that the most efficient payload delivery for a particular flight achieved 22823 kilograms. This figure aligns closely with SpaceX's declared payload capacity of 22800 kg to Low Earth Orbit (LEO) for flights where the first stage is not intended for recovery. (SpaceX, 2024). It's important to note that the mentioned

capacity is for the scenarios where the first stage is expended, not recovered. In cases where the first stage is retrieved—either via drone ship or return to the launch site, which falls outside the scope of this thesis due to insufficient mass and fuel data—the payload capacity is estimated to be around 18 tons, with potential reductions depending on the recovery location.

The table's rightmost columns detail the specifics of the transfer orbit required to reach the intended target orbit. Our target is an orbit with perigee and apogee altitudes of 284 km and 295 km, at an inclination of 43 degrees. All considered flights achieve a transfer orbit with an apogee exceeding 284 km and they navigate towards apogee before reaching their perigee which may be on the surface of the Earth or even lower because they reserve fuel for an apogee kick maneuver at around 284 km to elevate their perigee to 295 km, thus attaining the target orbit with the desired inclination. However, some of the flights have a negative inertial flight path angle which means they are headed to their perigee but since their perigee is higher than 160 kilometers, they are fit to perform and to be included in the design space. It's important to note that the outcomes listed in **Table 6.5** and **Table 6.7** all successfully achieve orbits with apogees and inclinations very close to their specified targets—within 2% and 1%, respectively. Additionally, they retain sufficient fuel to execute a Hohmann transfer maneuver at the designated apogee, facilitating the elevation of their orbit to the planned target orbit

Table 6.6 shows the sequence of events for the flight with the highest payload. Following the first cut-off of the second engine (SECO-1), the vehicle enters a coasting phase, continuing until it arrives at the apogee, at which point it commences the apogee kick maneuver, enduring for roughly 1.5 seconds.

Table 6.6 : Event sequence for Falcon 9 LEO mission.

LEO mission event sequence	
Time after launch	Event
00:00:00	Lift Off
00:00:14	Pitch Over Start
00:00:39	Pitch Over End
00:01:04	Max Q
00:02:33	MECO
00:02:41	Separation and SES-1
00:03:04	Fairing Ejection
00:09:01	SECO-1 (Transfer Orbit Injection)
00:45:57	SES-2 (Apogee Kick Start)
00:45:58	SECO-2 (Target Orbit Insertion)

Table 6.7 reveals a payload capacity of approximately 7847 kilograms for the super GTO mission. This contrasts with SpaceX's specification for the Falcon 9 FT, which is stated to deliver up to 8300 kilograms to a Geostationary Transfer Orbit (GTO) featuring a perigee around 35,786 kilometers. In the context of this thesis, however, an enhanced Super GTO orbit with an apogee extending to 59831 kilometers is studied.

It's evident that both the maximum dynamic pressure and the peak axial force (max g) experienced during this mission surpass those encountered in the initial mission. The reason behind this increase is the reduced mass of the vehicle, which, carrying roughly 7.8 tons as opposed to the previous 22.8 tons of payload, achieves greater acceleration. Consequently, the delta V capability of the vehicle is observed to be in the vicinity of 11700, aligning with the requirements for typical super GTO missions.

A note is that the apogee values in the transfer orbit effectively serve as the mission's target perigee values. At the apogee, the vehicle performs an apogee kick maneuver, which incrementally raises its perigee up to the desired apogee of 59831 kilometers, making the perigee the new apogee, thus completing the transition to the Super GTO orbit.

Table 6.7 : Falcon 9 successful SGTO missions - top 20 results.

Flight Index	Mach Start	Mach Finish	Alpha [°]	Constant Alpha Change Rate [°/s]	Delta V [m/s]	MECO Altitude [km]	Max Q [kPa]	Total Delta V Loss [m/s]	Payload Mass [kg]	Injection Altitude [km]	Transfer Orbit Apogee [km]	Transfer Orbit Perigee [km]	Transfer Orbit Inclination [°]	Inertial FPA @ Cut-off [deg]
4085	0.17	0.7345	1.9609	0.0365	11710	71.426	33.63	1611	7847	166.331	262.915	161.632	28.469	-0.039
4323	0.2	0.6036	2.641	0.0445	11710	70.492	33.77	1640	7845	160.868	261.798	128.482	28.470	0.473
3947	0.08	0.5382	0.5042	0.0318	11717	72.354	33.52	1613	7823	175.340	262.646	167.732	28.470	-0.147
4343	0.2	0.6036	2.641	0.0452	11716	70.492	33.77	1647	7823	161.728	265.173	118.730	28.470	0.562
4066	0.17	0.6691	1.9219	0.0358	11717	71.877	33.57	1626	7822	173.013	262.277	165.566	28.470	0.145
4403	0.2	0.6036	2.641	0.0472	11718	70.492	33.77	1667	7814	164.278	262.106	74.803	28.469	0.816
4502	0.17	0.7345	2	0.0505	11718	70.014	33.84	1677	7814	160.202	262.367	47.610	28.469	0.942
4046	0.17	0.6691	1.9219	0.0351	11720	71.877	33.57	1620	7810	172.109	266.141	167.260	28.470	0.051
4106	0.17	0.6691	1.9219	0.0371	11720	71.877	33.57	1638	7810	174.819	261.976	154.509	28.469	0.331
4304	0.11	0.6036	0.9062	0.0438	11721	71.043	33.70	1659	7804	169.147	263.519	97.936	28.470	0.709
4542	0.17	0.7345	2	0.0518	11722	70.014	33.84	1691	7800	161.841	261.958	7.484	28.468	1.102
3809	0.11	0.5382	0.8477	0.0271	11725	73.272	33.39	1618	7792	184.247	262.040	167.313	28.470	-0.278
4225	0.17	0.7345	1.9609	0.0411	11725	71.426	33.63	1654	7790	172.552	265.079	118.478	28.469	0.605
4126	0.17	0.6691	1.9219	0.0378	11725	71.877	33.57	1644	7790	175.722	264.862	145.823	28.469	0.424
4621	0.17	0.6691	1.9805	0.0545	11726	69.728	33.89	1704	7783	160.014	264.014	-27.721	28.468	1.244
4186	0.17	0.6691	1.9219	0.0398	11727	71.877	33.57	1663	7782	178.402	262.118	102.828	28.469	0.692
4483	0.2	0.6036	2.641	0.0498	11727	70.492	33.77	1695	7782	167.617	262.504	-6.194	28.469	1.142
4384	0.11	0.6036	0.9062	0.0465	11728	71.043	33.70	1686	7780	172.560	262.456	18.800	28.469	1.043
4305	0.17	0.7345	1.9609	0.0438	11728	71.426	33.63	1680	7779	176.031	261.316	42.313	28.469	0.945
4582	0.17	0.7345	2	0.0532	11729	70.014	33.84	1706	7774	163.466	264.126	-35.040	28.468	1.261

A breakdown of the most successful flight until the transfer orbit injection can be seen in **Figure 6.9**

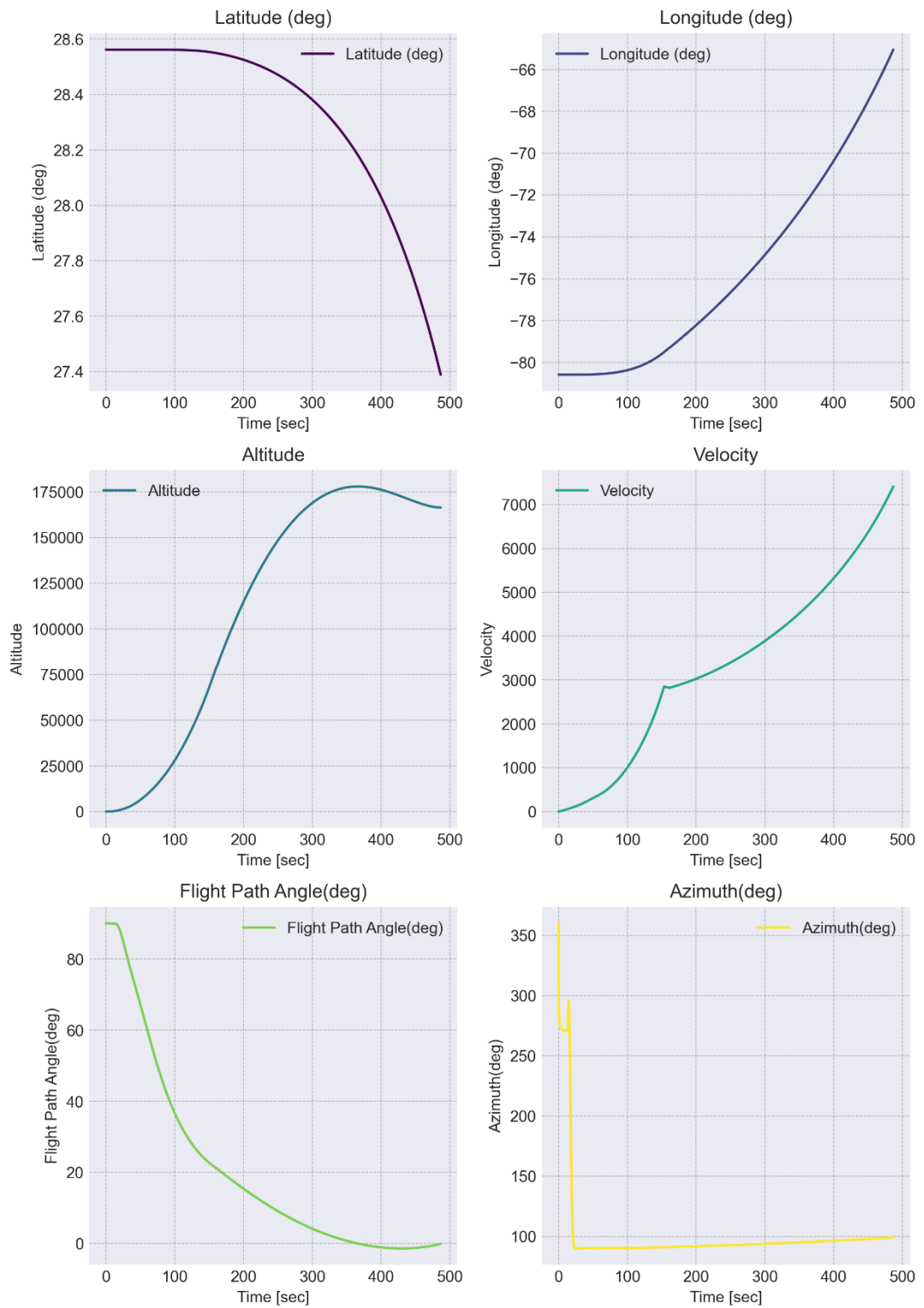


Figure 6.9 : Plots of the most successful flight.

Table 6.8 shows the sequence of events for the flight with the highest payload for the Super GTO mission. Following the first cut-off of the second engine (SECO-1), the vehicle enters a coasting phase, continuing until it arrives at the apogee, at which point it commences the apogee kick maneuver, raising its apogee to 59831 kilometers with a second engine firing of 56 seconds.

Table 6.8 : Event sequence for Falcon 9 Super GTO mission.

SGTO mission event sequence	
Time after launch	Event
00:00:00	Lift Off
00:00:13	Pitch Over Start
00:00:41	Pitch Over End
00:01:03	Max Q
00:02:33	MECO
00:02:41	Separation and SES-1
00:03:14	Fairing Ejection
00:08:06	SECO-1 (Transfer Orbit Injection)
00:45:55	SES-2 (Apogee Kick Start)
00:46:51	SECO-2 (Target Orbit Insertion)

6.2 Indirect Injection: Electron

6.2.1 Vehicle properties

6.2.1.1 General properties

The specifications utilized for the Electron Rocket (Rocket Lab, 2024) are outlined in the table. Given the challenge of sourcing comprehensive and up-to-date information on the vehicle, these parameters were compiled from various resources. The bulk of the data pertains to the original two-stage configuration of the Electron rocket, excluding the third stage, as per information from Space Launch Report (Space Launch Report, n.d.) and Launcher (Launcher, 2023). The collected data have been organized to ensure coherence in the delta-v calculations for the intended missions. It should be noted that the model does not incorporate the battery jettison event. According to Rocket Lab's 2022 Payload Users Guide (Rocket Lab, 2022), Electron is capable of

delivering payloads up to 200 kg to Sun-Synchronous Orbit (SSO) and up to 300 kg to Low Earth Orbit (LEO).

Table 6.9 : Rocket Lab Electron – assumed properties

Rocket Lab Electron assumed properties			
	1 st mission	2 nd mission	Unit
Height	18	18	m
Diameter	1.2	1.2	m
Fairing Diameter	1.2	1.2	m
Assumed Payload (pre-iteration)	200	300	kg
Fairing mass	44	44	kg
GLOM (pre-iteration)	14741	14841	kg
T/W (pre-iteration)	1.2848	1.3054	-
Delta V (pre-iteration)	9938	9368	m/s
1st stage			
Sea Level Thrust	185.74	185.74	kN
Single Engine Sea Level Thrust	20.638	20.638	kN
Vacuum Thrust	220.41	232.2	kN
Single Engine Vacuum Thrust	24.49	25.8	kN
Burn time	150	150	s
Vacuum ISP	311	311	s
PMF	0.91	0.91	-
Wet Mass	11912	11912	kg
Dry Mass	1072.118	1072.118	kg
Engine Bundle	9	9	#
Engine Exit Diameter	0.22	0.22	m
2nd Stage			
Vacuum Thrust	25.8	25.8	kN
Burn time	300	300	s
Vacuum ISP	343	343	s
PMF	0.89	0.89	-
Wet Mass	2585	2585	kg
Dry Mass	284.4	284.4	kg
Engine Bundle	1	1	#

6.2.1.2 Aerodynamic properties

Aerodynamics properties of the Electron has been obtained by Missile DATCOM, just like Falcon 9. The height, diameters and a simple fairing profile has been inputted, resulting in the geometry in **Figure 6.10** which can be seen side by side with real life model of Electron vehicle.



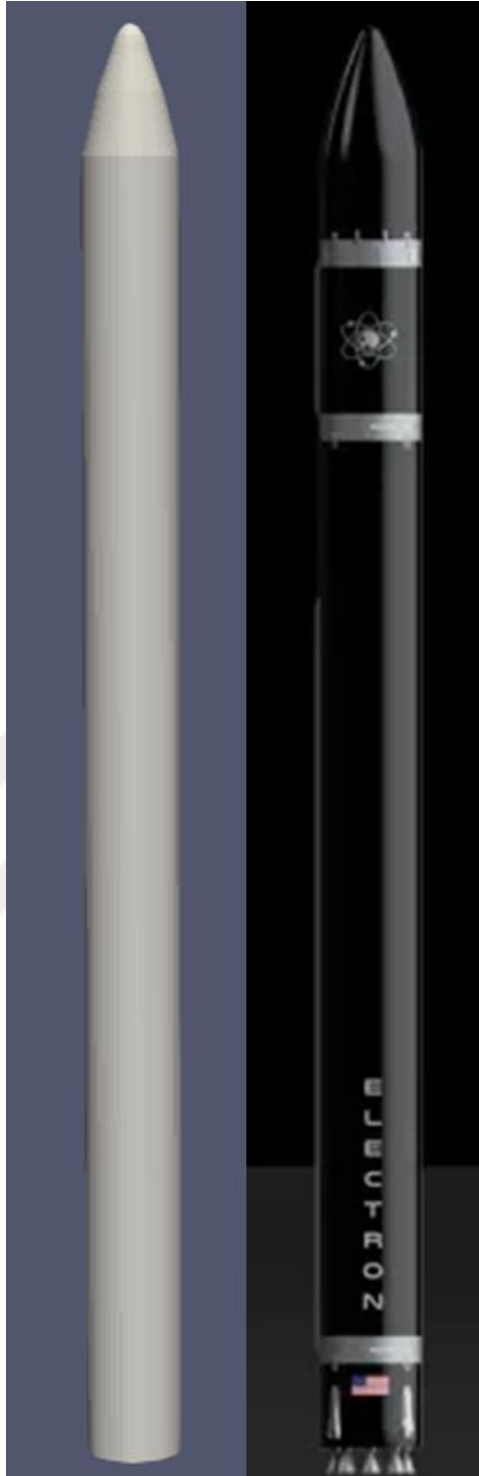


Figure 6.10 : Geometric comparison of reference model.

The resulting coefficient of drag profile can be seen in **Figure 6.11**. Like Falcon 9, Missile DATCOM seems to overestimate the CD at transonic regime.

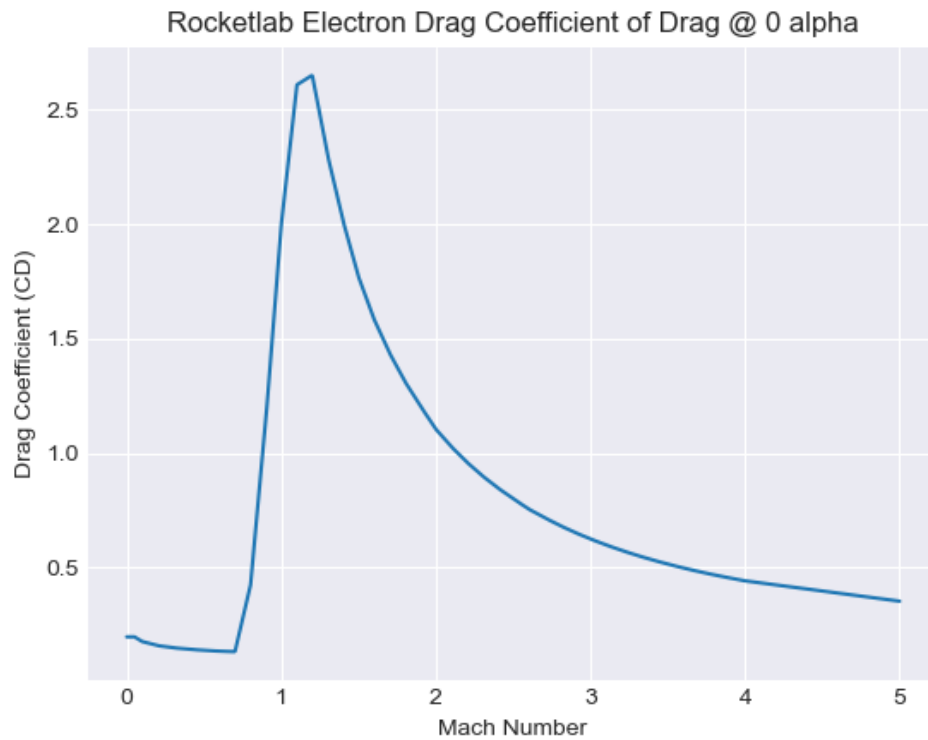


Figure 6.11 : CD of Electron obtained via Missile DATCOM.

6.2.2 Missions

For the Electron simulations, we have selected two distinct missions for analysis: one targeting Low Earth Orbit (LEO) and the other targeting at a Sun Synchronous Orbit (SSO). The specifics of these missions' orbits are detailed in **Table 6.10**.

Table 6.10 : Rocket Lab Electron missions.

Rocket Lab Electron - Missions			
	1 st mission	2 nd mission	Unit
Orbit			
Orbit Type	SSO	LEO	-
Target Apogee	500	300	km
Target Perigee	500	300	km
Transfer Orbit Apogee	~500	~300	km
Transfer Orbit Perigee	Unbounded	Unbounded	km
Target Inclination	98	42	°
Launch Site			
Site	Mahia Launch Complex		-
Latitude	-39.2609		°
Longitude	177.8659		°
Altitude	0		m
Launch Azimuth	-10.3550	73.6983	°

6.2.3 Statistics

Just like in the previous Falcon 9 missions, there were a lot of flight trajectories analyzed in the Electron missions as well. Design space reduction can be seen in **Table 6.11**.

Table 6.11 : Design space reduction Electron SSO mission.

1st Mission Design Space Reduction	
1st run 1st Stage Eliminations	
Elimination	Number of simulations left
Pre-Elimination	6079
After Max Q	4216
After Max Q*alpha	3942
After Minimum MECO altitude	3005
After Max g	3005
2nd run Final Eliminations	
Elimination	Number of simulations left
Pre-Elimination	6000
After Max Q	5666
After Max Q*alpha	5666
After Max g	5666
Successful Transfer Orbit Injection	3310
After Transfer Orbit Apogee	3085
After Transfer Orbit Inclination	2194
Total Assessed Trajectories	12079
Total Successful Trajectories	2194

Eliminations for the second mission can be seen in the table.

Table 6.12 : Design space reduction Electron LEO mission.

2nd Mission Design Space Reduction	
1st run 1st Stage Eliminations	
Elimination	Number of simulations left
Pre-Elimination	6079
After Max Q	4216
After Max Q*alpha	3946
After minimum MECO altitude	2989
After max g	2989
2nd run Final Eliminations	
Elimination	Number of simulations left
Pre-Elimination	6000
After Max Q	5890
After Max Q*alpha	5890
After Max g	5890
Successful Transfer Orbit	3845
Injection	3841
After Transfer Orbit Apogee	3841
After Transfer Orbit	2840
Inclination	2840
Total Assessed Trajectories	12079
Total Successful Trajectories	2840

Furthermore, the visualization of the flight trajectories and the effect of the constraints are presented in **Figure 6.12** for the first phase and in **Figure 6.13** for the final phase of the SSO mission while for the LEO mission it is presented in **Figure 6.14** and **Figure 6.15** respectively.

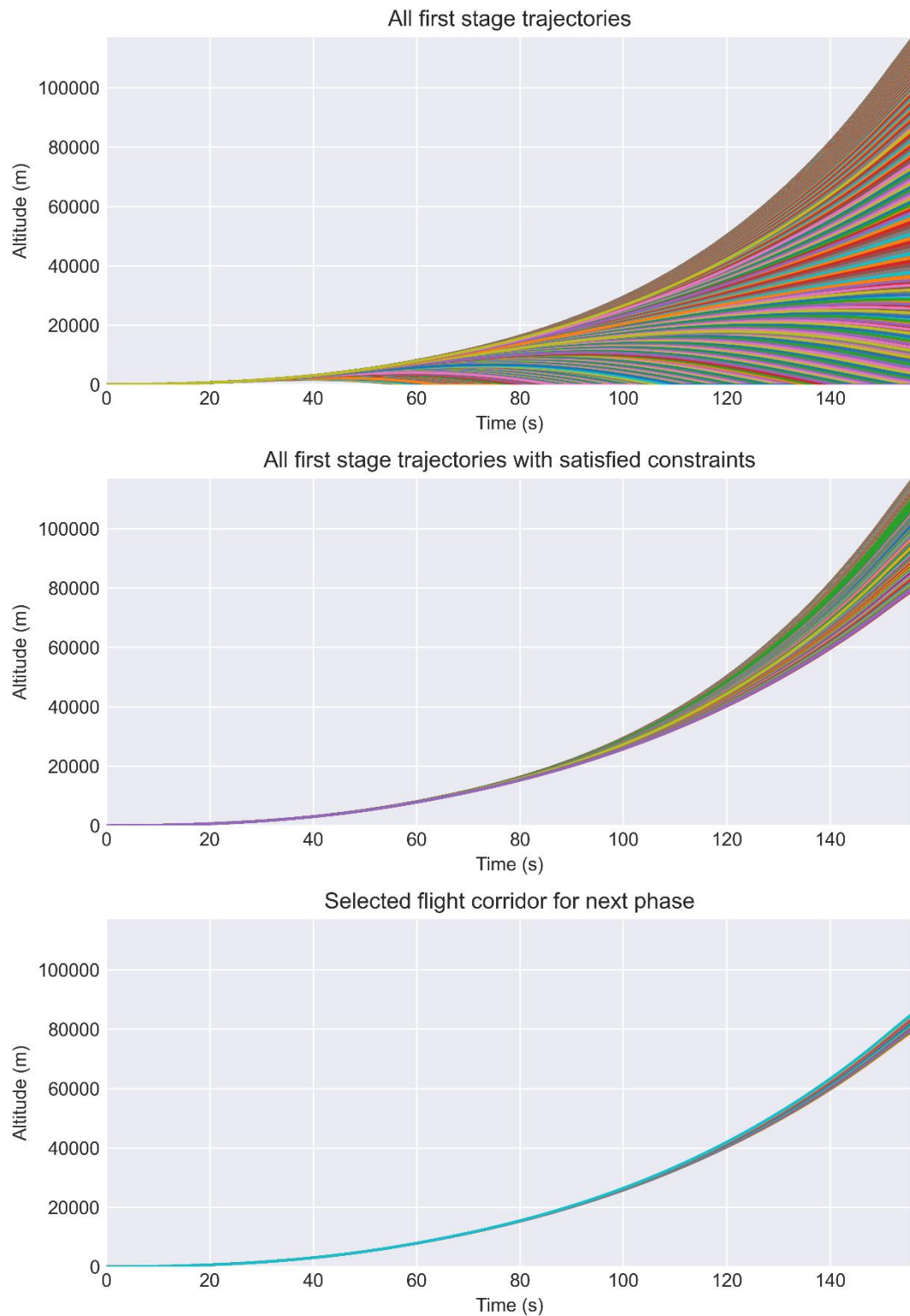


Figure 6.12 : Electron SSO mission first phase design space.

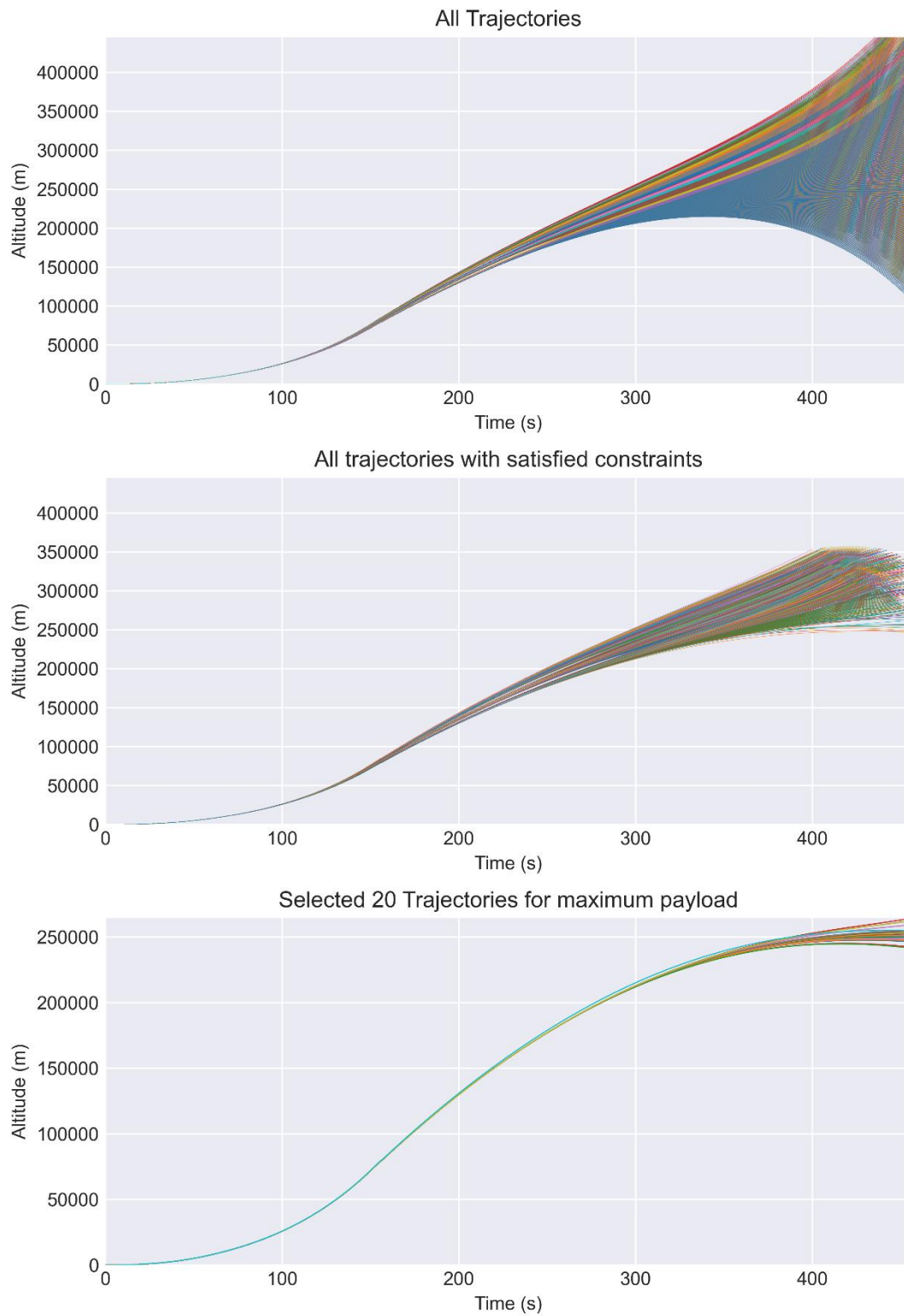


Figure 6.13 : Electron SSO mission final phase design space.

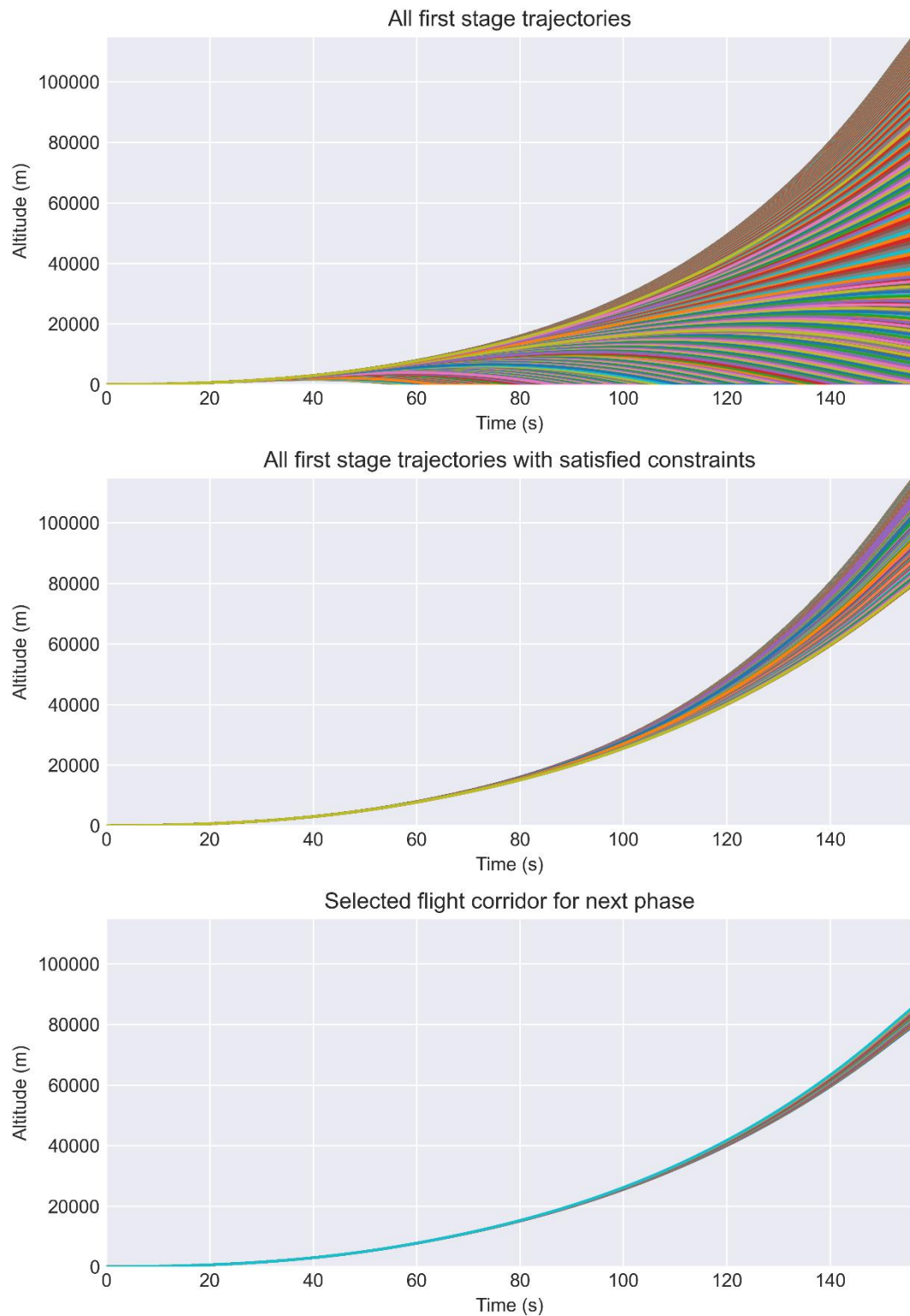


Figure 6.14 : Electron LEO mission first phase design space.

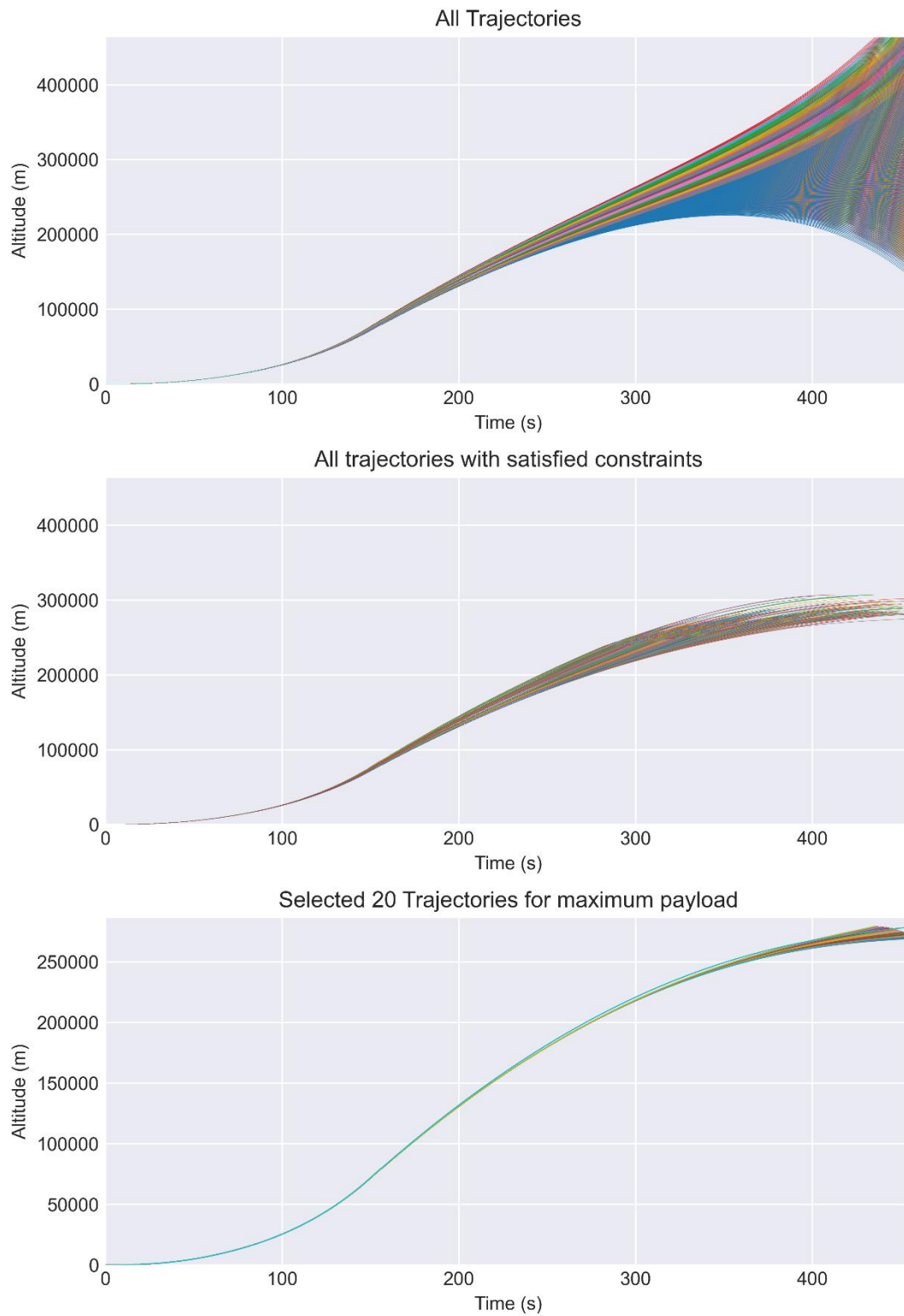


Figure 6.15 : Electron LEO mission final phase design space.

6.2.4 Results

A breakdown of the most successful flight of Electron for SSO mission can be seen in **Figure 6.16**.

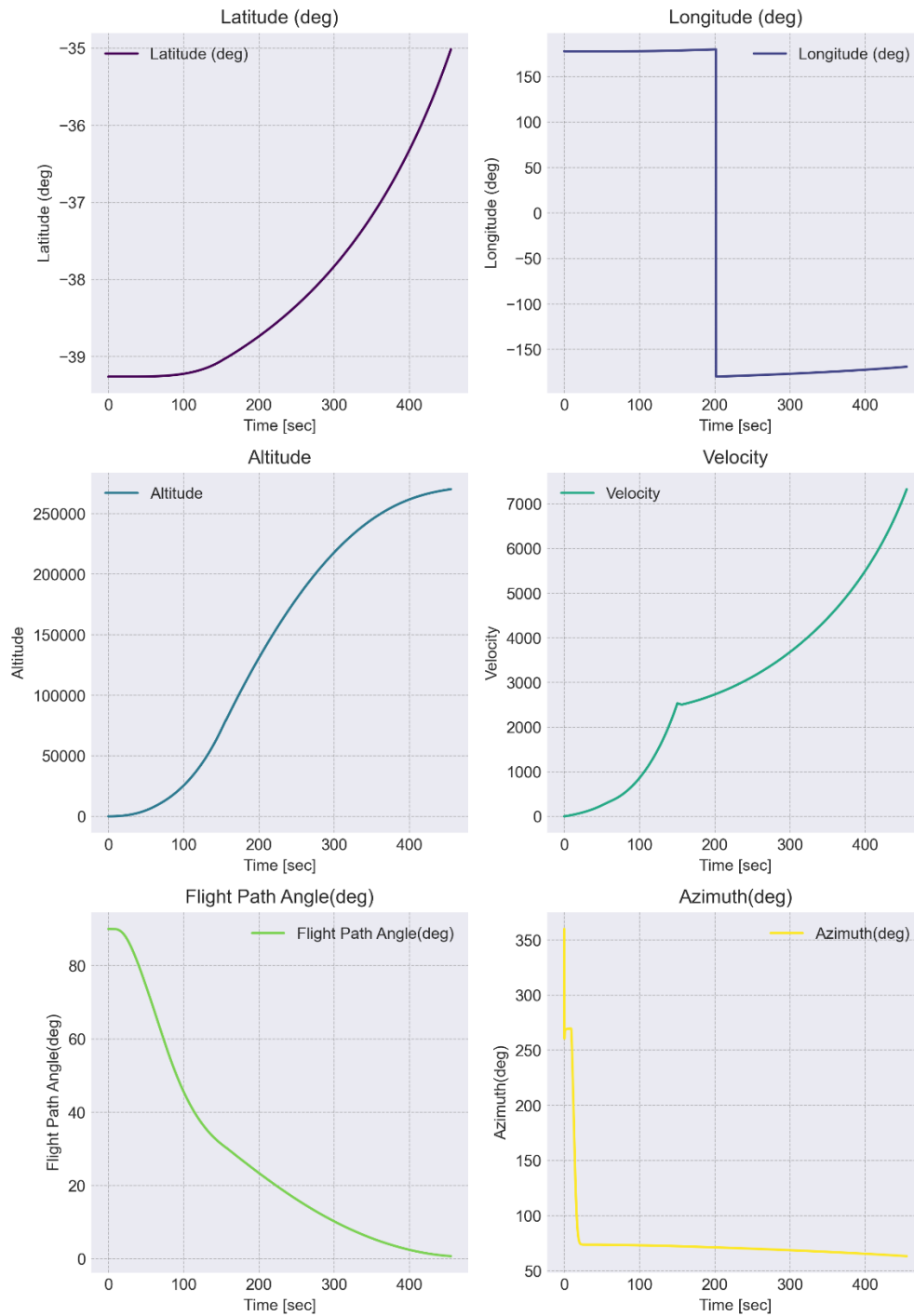


Figure 6.16 : Electron SSO mission plots of the best flight.

Likewise, the plots of the best flight can be examined in **Figure 6.17** for the LEO mission.

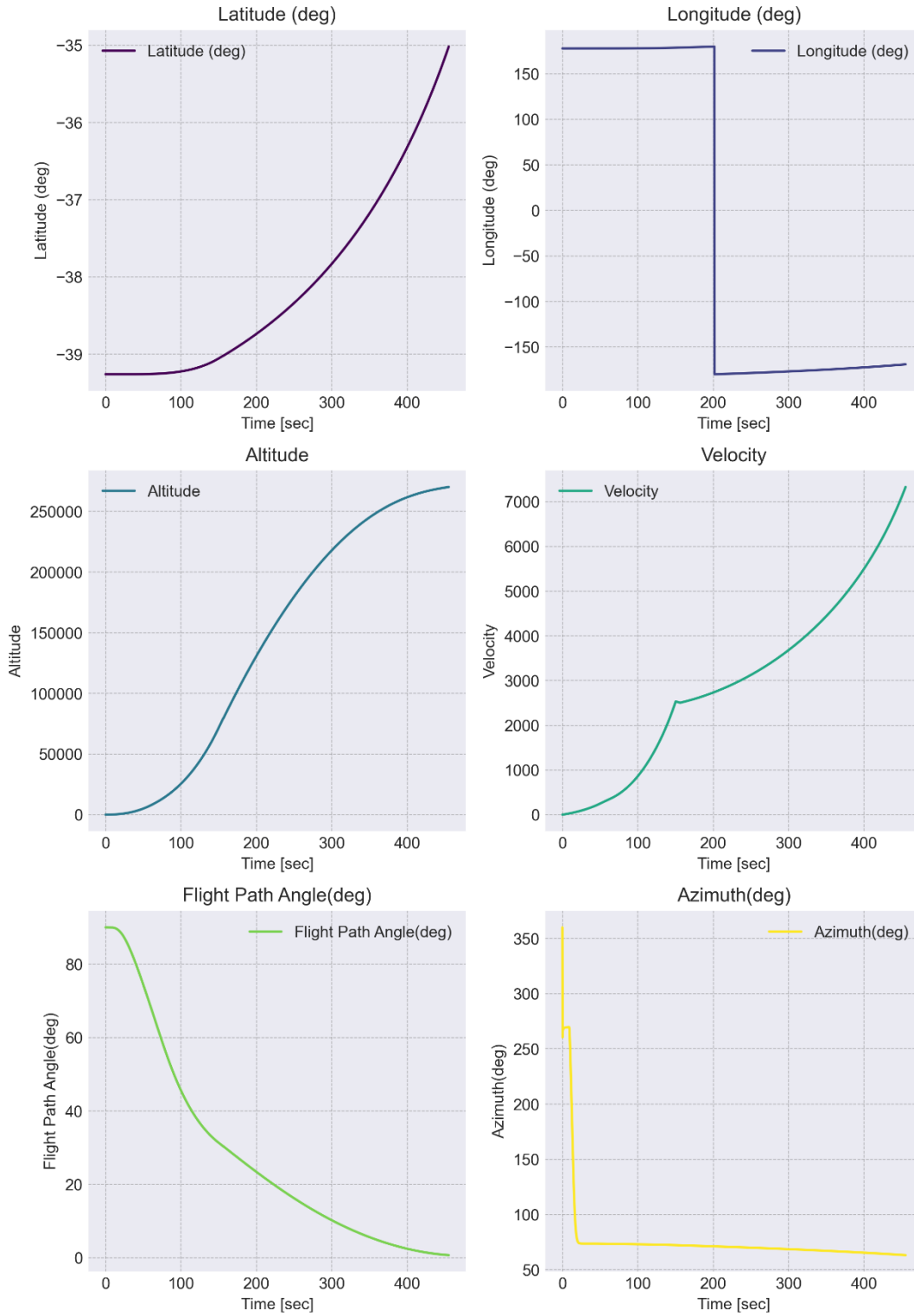


Figure 6.17 : Electron LEO mission plots of the best flight.

Event sequence for the SSO mission can be seen **Table 6.13**.

Table 6.13 : Electron SSO mission event sequence.

Electron SSO Mission Event Sequence	
Time after Launch	Event
00:00:00	Lift Off
00:00:19	Pitch Over Start
00:00:50	Pitch Over End
00:01:07	Max Q
00:02:30	MECO
00:02:36	Separation and SES-1
00:03:00	Fairing Ejection
00:07:35	SECO-1 (Transfer Orbit Injection)
00:49:26	SES-2 (Apogee Kick Start)
00:49:27	SECO-2 (Target Orbit Insertion)

Event sequence for the LEO mission for Electron can be seen in **Table 6.14**.

Table 6.14 : Electron LEO mission event sequence.

Electron SSO Mission Event Sequence	
Time after Launch	Event
00:00:00	Lift Off
00:00:09	Pitch Over Start
00:00:46	Pitch Over End
00:01:08	Max Q
00:02:30	MECO
00:02:36	Separation and SES-1
00:02:59	Fairing Ejection
00:07:35	SECO-1 (Transfer Orbit Injection)
00:17:50	SES-2 (Apogee Kick Start)
00:17:52	SECO-2 (Target Orbit Insertion)

Electron's SSO missions' top 20 results are presented in the **Table 6.15** while the top 20 results for the LEO mission are presented in **Table 6.16** which are sorted by the payload.

Table 6.15 : Electron SSO mission top 20 results.

Flight Index	Mach Start	Mach Finish	Alpha [°]	Constant Alpha Change Rate [°/s]	Delta V [m/s]	MECO Altitude [km]	Max Q [kPa]	Total Delta V Loss [m/s]	Payload Mass [kg]	Injection Altitude [km]	Transfer Orbit Apogee [km]	Transfer Orbit Perigee [km]	Transfer Orbit Inclination [°]	Inertial FPA @ Cut-off [deg]
2901	0.08	0.8	0.2819	-0.0030	9951	72.137	30.89	1862	197	250.371	500.002	245.130	97.818	0.110
2921	0.08	0.8	0.2819	-0.0023	9951	72.137	30.89	1863	197	251.354	500.148	243.888	97.818	0.230
2881	0.08	0.8	0.2819	-0.0037	9952	72.137	30.89	1860	197	249.388	501.323	244.807	97.818	-0.009
2741	0.08	0.8	0.2819	-0.0084	9953	72.137	30.89	1851	196	242.533	502.493	201.063	97.811	-0.841
2842	0.17	0.6036	1.2383	-0.0050	9953	72.143	30.89	1858	196	247.549	500.933	239.706	98.000	-0.246
2941	0.08	0.8	0.2819	-0.0017	9953	72.137	30.89	1864	196	252.336	501.740	241.105	97.818	0.349
2861	0.08	0.8	0.2819	-0.0043	9955	72.137	30.89	1859	196	248.405	504.082	242.948	97.818	-0.128
2982	0.17	0.6036	1.2383	-0.0003	9957	72.143	30.89	1868	196	254.428	502.211	230.771	98.001	0.589
2961	0.08	0.8	0.2819	-0.0010	9957	72.137	30.89	1866	196	253.318	504.703	236.851	97.819	0.468
3082	0.17	0.6036	1.2383	0.0030	9959	72.143	30.89	1876	196	259.296	501.539	176.542	97.997	1.185
2822	0.17	0.6036	1.2383	-0.0057	9959	72.143	30.89	1857	196	246.566	506.377	234.967	98.000	-0.366
2841	0.08	0.8	0.2819	-0.0050	9959	72.137	30.89	1857	196	247.421	508.202	239.632	97.817	-0.248
2722	0.17	0.6036	1.2383	-0.0090	9959	72.143	30.89	1850	196	241.680	506.564	190.073	97.994	-0.959
3181	0.08	0.8	0.2819	0.0064	9960	72.137	30.89	1884	195	263.929	500.488	82.876	97.805	1.772
3142	0.17	0.6036	1.2383	0.0050	9961	72.143	30.89	1881	195	262.173	501.222	124.891	97.993	1.540
2981	0.08	0.8	0.2819	-0.0003	9962	72.137	30.89	1867	195	254.300	508.929	231.239	97.819	0.588
3081	0.08	0.8	0.2819	0.0030	9963	72.137	30.89	1876	195	259.167	506.908	178.286	97.815	1.183
3002	0.17	0.6036	1.2383	0.0003	9963	72.143	30.89	1870	195	255.409	507.762	223.738	98.002	0.709
3141	0.08	0.8	0.2819	0.0050	9964	72.137	30.89	1881	195	262.045	505.714	127.443	97.810	1.538
2843	0.23	0.7345	2.3077	-0.0050	9965	72.522	30.82	1878	195	255.145	502.686	250.328	97.883	0.064

Table 6.16 : Electron LEO mission top 20 results.

Flight Index	Mach Start	Mach Finish	Alpha [°]	Constant Alpha Change Rate [°/s]	Delta V [m/s]	MECO Altitude [km]	Max Q [kPa]	Total Delta V Loss [m/s]	Payload Mass [kg]	Injection Altitude [km]	Transfer Orbit Apogee [km]	Transfer Orbit Perigee [km]	Transfer Orbit Inclination [°]	Inertial FPA @ Cut-off [deg]
2761	0.08	0.6691	0.2468	-0.0077	9456	72.031	30.58	1931	282	270.224	300.067	60.079	42.147	0.756
2801	0.08	0.6691	0.2468	-0.0064	9457	72.031	30.58	1933	282	271.933	300.362	-86.468	42.143	0.986
2841	0.08	0.6691	0.2468	-0.0050	9459	72.031	30.58	1936	282	273.545	301.310	-262.106	42.137	1.215
2821	0.08	0.6691	0.2468	-0.0057	9459	72.031	30.58	1934	282	272.765	301.818	-158.947	42.141	1.100
2881	0.08	0.6691	0.2468	-0.0037	9459	72.031	30.58	1938	282	274.993	300.873	-491.904	42.130	1.445
2861	0.08	0.6691	0.2468	-0.0043	9460	72.031	30.58	1937	282	274.302	301.612	-363.089	42.134	1.330
2921	0.08	0.6691	0.2468	-0.0023	9460	72.031	30.58	1941	282	276.239	300.112	-765.164	42.121	1.674
2741	0.08	0.6691	0.2468	-0.0084	9460	72.031	30.58	1929	282	269.347	303.797	131.662	42.149	0.639
2901	0.08	0.6691	0.2468	-0.0030	9460	72.031	30.58	1940	282	275.656	300.801	-616.554	42.126	1.559
2781	0.08	0.6691	0.2468	-0.0070	9461	72.031	30.58	1932	281	271.113	304.136	18.682	42.146	0.869
2961	0.08	0.6691	0.2468	-0.0010	9461	72.031	30.58	1944	281	277.306	300.001	-1044.089	42.112	1.899
2941	0.08	0.6691	0.2468	-0.0017	9461	72.031	30.58	1943	281	276.844	300.758	-880.601	42.118	1.785
2981	0.08	0.6691	0.2468	-0.0003	9462	72.031	30.58	1946	281	277.791	300.271	-1174.939	42.107	2.009
3001	0.08	0.6691	0.2468	0.0003	9463	72.031	30.58	1947	281	278.180	300.071	-1324.082	42.101	2.120
3021	0.08	0.6691	0.2468	0.0010	9463	72.031	30.58	1949	281	278.533	300.041	-1466.115	42.095	2.230
3041	0.08	0.6691	0.2468	0.0017	9464	72.031	30.58	1950	281	278.850	300.122	-1601.502	42.089	2.339
3061	0.08	0.6691	0.2468	0.0023	9465	72.031	30.58	1952	281	279.133	300.272	-1730.678	42.083	2.447
3081	0.08	0.6691	0.2468	0.0030	9466	72.031	30.58	1954	281	279.307	300.061	-1873.483	42.077	2.555
3101	0.08	0.6691	0.2468	0.0037	9467	72.031	30.58	1955	280	279.520	300.301	-1990.571	42.071	2.661
2782	0.14	0.6036	0.8086	-0.0070	9468	72.440	30.52	1952	280	278.392	300.018	-358.846	42.150	1.214

6.3 Success of the Indirect Injection Case Studies

The apogee kick burn in the simulation is determined analytically, with the program concluding at the point of transfer orbit injection. Initially, a hypothetical payload mass is specified, which is adjusted based on the actual fuel consumption post-mission. If surplus fuel remains, it indicates potential for a higher payload capacity; conversely, increased fuel demand for orbit insertion suggests a reduced payload capability. The simulator iteratively refines the estimated payload mass to align closely with the final calculated value. **Table 6.17** provides details on these adjustments and the rates at which payload convergence is achieved.

Table 6.17 : Payload and mission success summaries.

Property	Falcon 9 LEO Mission	Falcon 9 SGTO Mission	Electron SSO Mission	Electron LEO Mission
Iteration Count	2	3	1	2
2nd stage assumed fuel mass	109807	109807	2301	2301
2nd stage first burn fuel mass	109304	93638	2293	2293
2nd stage second burn fuel mass	323.24	16098	10.45	12.1
2nd stage total used fuel mass	109627.8	109736	2303.8	2305.5
Fuel mass left in the tank	179.81	71.3	-2.77	-4.43
Assumed payload Mass at last iteration	22643	7776	200	287
Injected payload mass	22823	7847	197	282
Payload convergence rate	% 100.72	% 100.9	% 98.48	% 98.17

6.4 Direct Injection – Falcon 9 Super GTO Mission

The vectorized simulator excels in exploring transfer orbits and identifying the optimal ones for indirect injections into the target orbit, but it's also capable of handling direct injections. To showcase this functionality, a mission involving Falcon 9, previously analyzed for indirect injection, has also been assessed for direct injection into orbit. The specific mission selected for this demonstration targeted a super geostationary transfer orbit, which can be seen in **Table 6.18**, characterized by an apogee significantly above that of a standard geostationary orbit, inspired by the launch orbit of EUTELSAT 10B.

Table 6.18 : SGTO direct injection mission.

Falcon 9 FT Block 5 – Super GTO Mission		
Orbit Type	Super GTO	-
Target Apogee	59831	km
Target Perigee	261	km
Target Inclination	28.56	°
Launch Site		
Site	SLC-40, Cape Canaveral	-
Latitude	28.5621	°
Longitude	-80.5772	°
Altitude	0	m
Launch Azimuth	90	°

The vehicle's capacity for direct injection into the mission orbit has been determined to be 7400 kilograms, which is 447 kilograms less than the optimal outcome achieved through the indirect injection method. **Figure 6.18** illustrates the trajectory immediately following the launch from Space Launch Complex 40 at Cape Canaveral, showing the vehicle's initial ascent and gradual pitch towards the launch azimuth. **Figure 6.19** depicts the entire boost phase of the vehicle up to the point of achieving the final orbit, representing the powered portion of the launch.

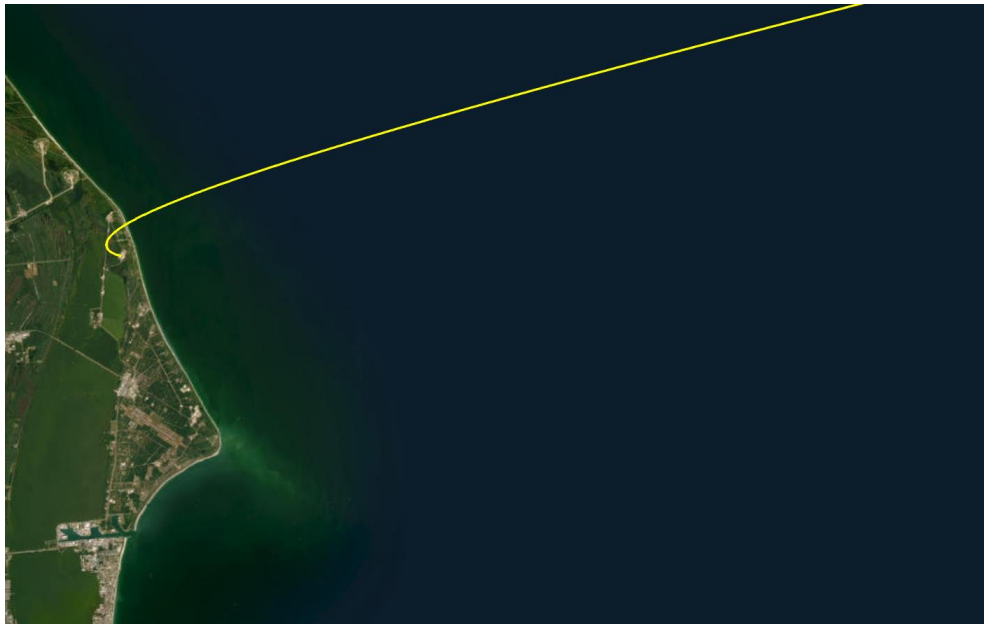


Figure 6.18 : Falcon 9 initial trajectory from Cape Canaveral.

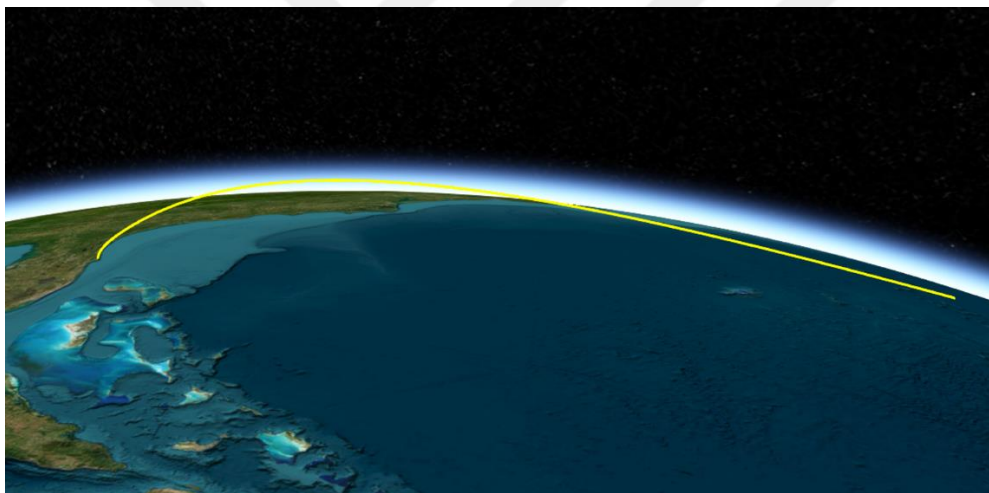


Figure 6.19 : Falcon 9 boost trajectory to orbit.

Illustrating the complexities of a super GTO orbit can be a challenging task, as its nuances are not easily understood at first glance. This orbit reaches an apogee of about 60000 kilometers, at which point the satellite's radial speed is slower than Earth's rotational velocity for quite some time, causing the satellite to appear to fall behind from an Earth-based viewpoint. Consequently, the satellite's ground track seems like bean shapes in **Figure 6.20**. This effect is partly due to the significant rotation of the Earth during the satellite's orbit.

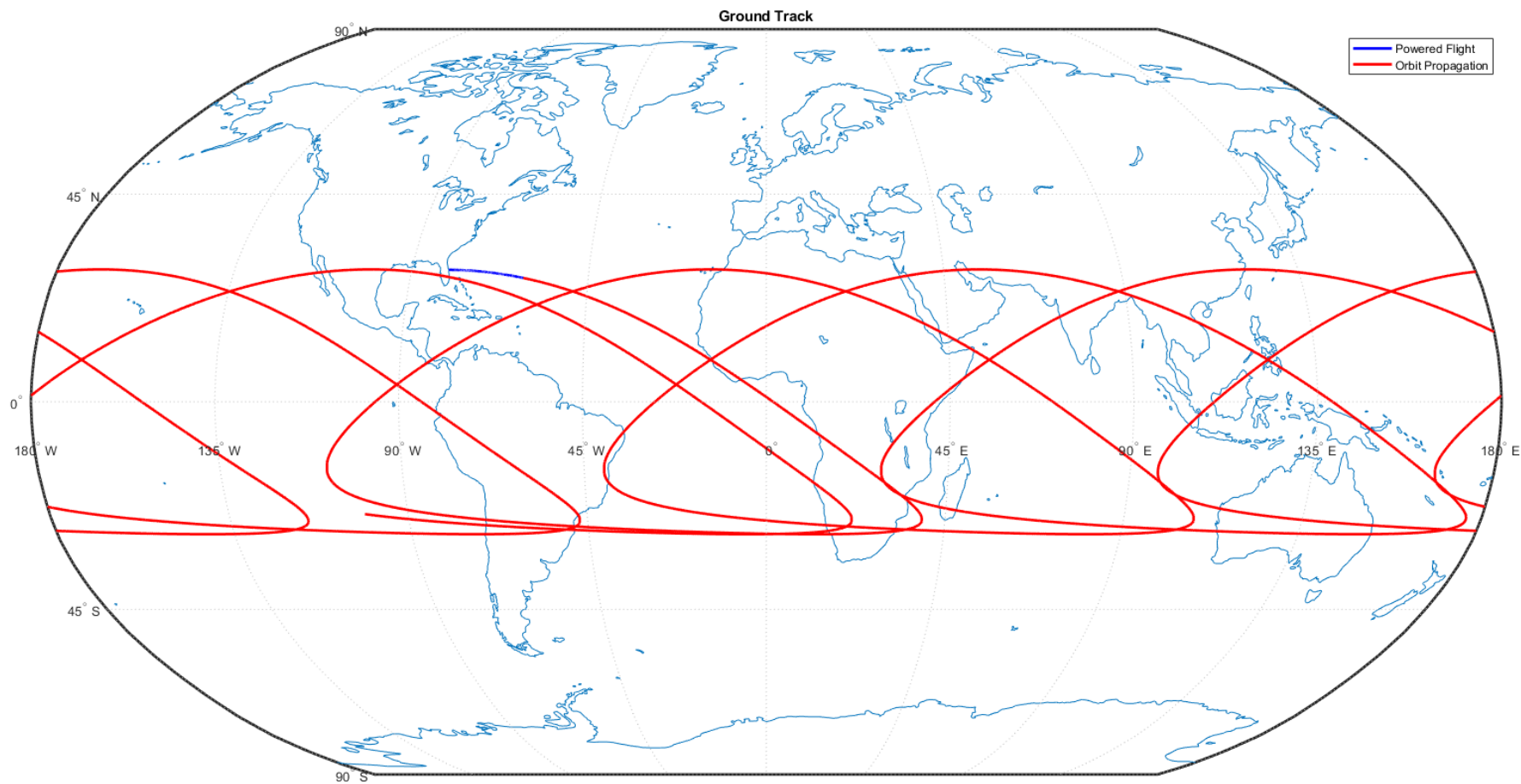


Figure 6.20 : Falcon 9 SGTO ground track.

The reached orbit for the Super GTO Mission can be seen below in **Table 6.19** with apogee and perigee values and their error with the target.

Table 6.19 : Falcon 9 SGTO direct injection mission success.

Falcon 9 FT Block 5 – Super GTO Mission			
	Target	Result	Error
Apogee	59831	59765	% 0.11
Perigee	261	260.71	% 0.11
Inclination	28.56	28.51	% 0.16

The orbital elements of the resulting orbit can be seen below in **Table 6.20**.

Table 6.20 : Falcon 9 SGTO direct injection orbital elements

Falcon 9 FT Block 5 – Super GTO Orbital Elements				
	Target	Result	Error	Unit
Semimajor Axis	36241	36391	% 0.09	km
Inclination	28.56	28.51	% 0.16	deg
Eccentricity	0.8177	0.8176	% 0.02	-
Right Ascension	-	183.56	-	deg
Argument of Perigee	-	104.61	-	deg
True Anomaly	-	5.41	-	deg

6.5 Direct Injection: Electron LEO Mission

A direct injection mission was also devised for the Electron rocket, targeting a circular low Earth orbit at an altitude of 250 kilometers, which is standard for such missions. Details of the mission are provided in **Table 6.21**.

Table 6.21 : Rocket Lab Electron LEO direct injection mission.

Rocket Lab Electron– Circular LEO mission		
Orbit Type	Circular LEO	-
Target Apogee	250	km
Target Perigee	250	km
Target Inclination	42	°
Launch site		
Site	Mahia Launch Complex	-
Latitude	-39.2609	°
Longitude	177.8659	°
Altitude	0	m
Launch Azimuth	73.6983	°

The vehicle's capacity for direct injection into the mission orbit has been determined to be 287 kilograms. **Figure 6.21** illustrates the trajectory immediately following the launch from Mahia Launch Complex, showing the vehicle's initial ascent and gradual pitch towards the launch azimuth. **Figure 6.22** depicts the entire boost phase of the vehicle up to the point of achieving the final orbit, representing the powered portion of the launch.

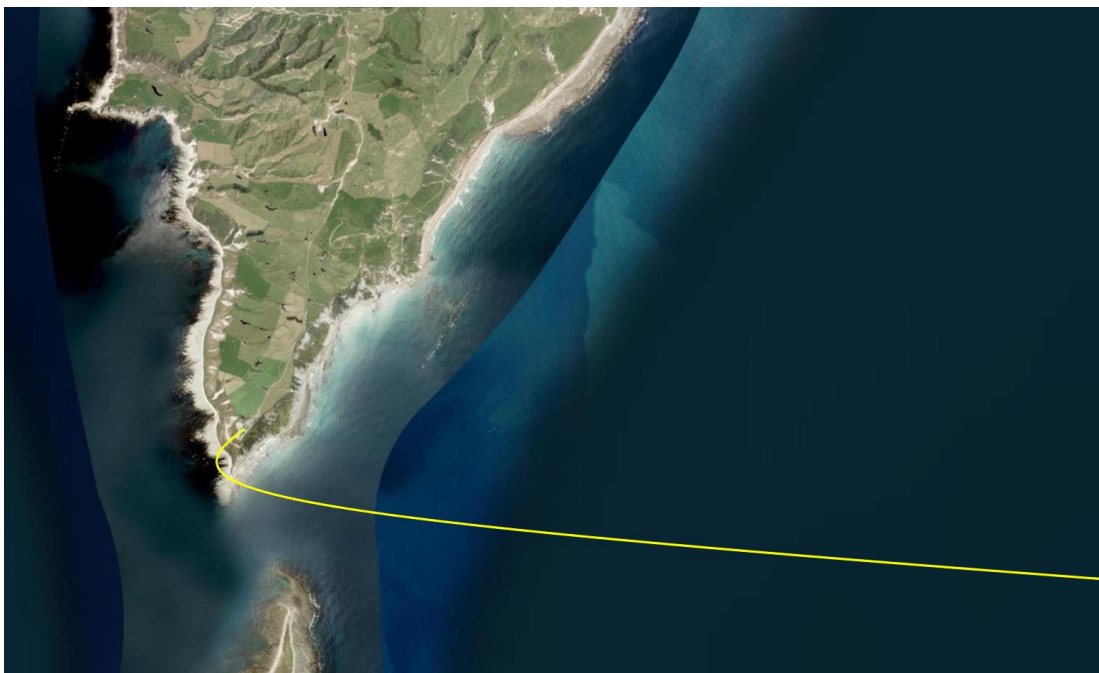


Figure 6.21 : Electron trajectory after lift-off.

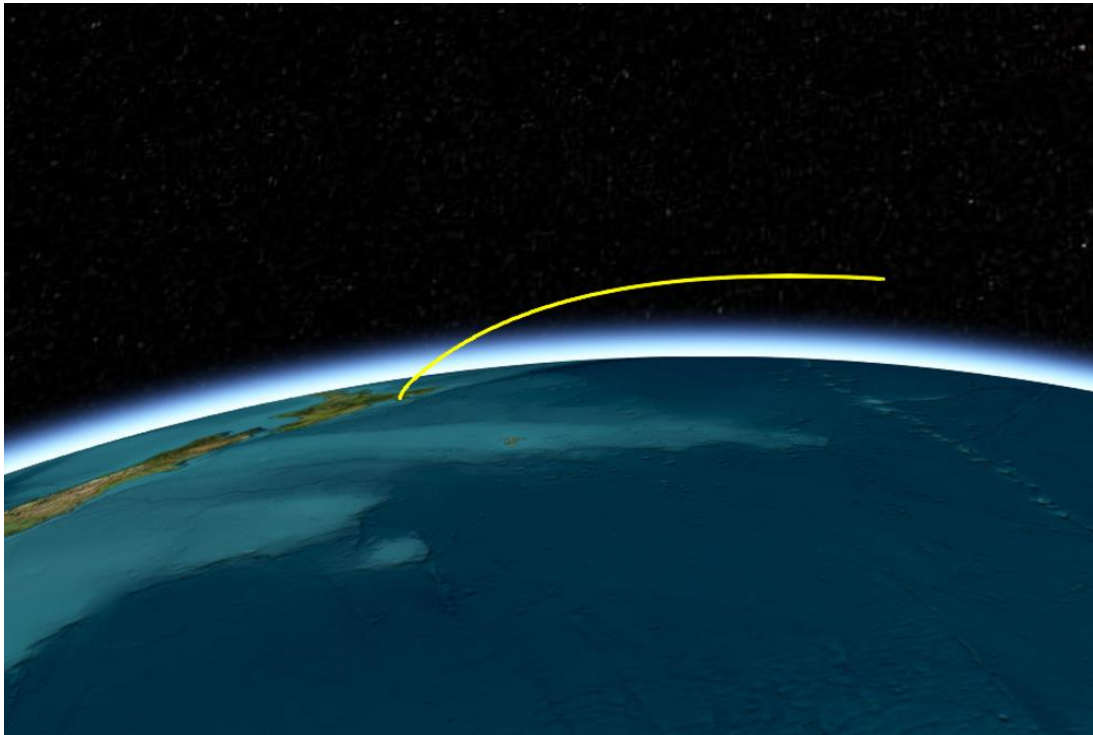


Figure 6.22 : Electron boost trajectory to orbit.



Figure 6.23 : Rocket Lab Electron LEO ground track.

The acquired orbit for the circular LEO mission can be seen below with apogee and perigee values and their error with the target in **Table 6.22**.

Table 6.22 : Electron direct injection mission success.

Electron – Circular LEO Mission			
	Target	Result	Error
Apogee	250	259.87	% 3.95
Perigee	250	252.37	% 0.95
Inclination	42	42.15	% 0.36

The orbital elements of the resulting orbit can be seen below in **Table 6.23**.

Table 6.23 : Electron direct injection orbital elements.

Electron – Circular LEO Mission Orbital Elements				
	Target	Result	Error	Unit
Semimajor Axis	6504	6634	% 2	km
Inclination	42	42.15	% 0.36	deg
Eccentricity	0	0.00056	~ 0	-
Right Ascension	-	238.3	-	deg
Argument of Perigee	-	42.34	-	deg
True Anomaly	-	259.39	-	deg

7. CONCLUSIONS

In this thesis, a 3D trajectory simulator leveraging vectorized computation has been created, enabling the efficient exploration of a vast number of launch vehicle trajectories. This simulator stands out by not only iterating through thousands of potential flight paths swiftly but also by showcasing its robustness across several realistic mission scenarios, including those for SpaceX's Falcon 9 and Rocket Lab's Electron. Unlike conventional optimizers that narrow down to a single solution, this tool presents a spectrum of viable maneuvers and trajectory options, each with associated payload capabilities. This feature is invaluable in the preliminary design of launch vehicles in which the designer has to quickly iterate through design space with sometimes very strict constraints. The simulator's ability to provide a diverse set of solutions enriches the decision-making process, offering insights into the trade-offs and benefits of various flight strategies.

In addition to exploring various transfer orbits for reaching the intended orbit, the tool also has the capability to identify opportunities for direct injections into the target orbit. This research analyzes 2 Falcon 9 and 2 Electron missions for indirect injections, alongside 1 Falcon 9 and 1 Electron mission for direct injection scenarios, making up a total of 6 distinct case studies.

Looking ahead, given that the primary objective of the tool is to swiftly simulate trajectories, it can be adapted for Monte Carlo analysis to model various launch trajectory scenarios. Moreover, the tool has the potential to be developed into a more comprehensive system that supports three-stage launch vehicle trajectories and includes a stage mass scanner feature for enhanced optimization of stage mass allocations.



REFERENCES

- Bettini, M., Kortvelesy, R., Blumenkamp, J., & Prorok, a. A.** (2022). *VMAS: A Vectorized Multi-Agent Simulator for Collective Robot Learning*. arXiv.
- Brauer, G., Cornick, D., & Stevenson, R.** (1977). *Capabilities and Applications of the Program to Optimize Simulated Trajectories (POST)*. NASA.
- Chobotov, V. A.** (2002). *Orbital Mechanics*. AIAA.
- Coşkun, E. C.** (2014). *Multistage Launch Vehicle Design with Thrust Profile and Trajectory Optimization, PhD thesis*. Middle East Technical University.
- Curtis, N. J., Niemeyer, K. E., & Sung, C.-J.** (2022). Accelerating reactive-flow simulations using vectorized chemistry integration. *Computer Physics Communications*.
- Dees, P. D., Zwack, M. R., Edwards, S., & Steffens, M. J.** (2016). Augmenting Conceptual Design Trajectory Tradespace Exporation with Graph Theory. *AIAA SPACE 2016*.
- Dongarra, J. J., Demmel, J. W., & Ostrouchov, S.** (1992). *LAPACK: A Linear Algebra Library for High-Performance Computers*. Heidelberg: Physica-Verlag HD.
- Edberg, D., & Costa, W.** (2022). *Design of Rockets and Space Launch Vehicles*. Reston, Virginia: American Institute of Aeronautics and Astronautics.
- ESA.** (2019). *ESA - Low shock fairing separation and jettison system*. Retrieved from ESA:
https://www.esa.int/ESA_Multimedia/Images/2019/10/Low_shock_fairing_separation_and_jettison_system
- Gath, P. F., & Calise, A. J.** (2001). Optimization of Launch Vehicle Ascent Trajectories with Path Constraints and Coast Arcs. *JOURNAL OF GUIDANCE, CONTROL, AND DYNAMICS*, 296-304.

- Gentzsch, W.** (1984). *Vectorization of Computer Programs with Applications to Computational Fluid Dynamics*. Springer.
- Gheata, A.** (2019). *Vectorization of simulation code*. Cern - GeantV.
- Haeussermann, W.** (1970). *Description and Performance of the Saturn Launch Vehicles Navigation, Guidance, and Control System*. NASA.
- Hanson, J. M., Shrader, M. W., & Cruzen, C. A.** (1995). Ascent Guidance Comparisions. *Journal of the Astronautical Sciences*, 43(3), 307-326.
- Harris, C. R., Millman, K. J., van der Walt, S. J., Gommers, R., Virtanen, P., Cournapeau, D., . . . Brett, M.** (2020). Array programming with Numpy. *Nature*, 357-362.
- Holt, J. B., Dee, P. D., & Diaz, M. J.** (2015). An Expert System-Driven Method for Parametric Trajectory Optimization During Conceptual Design. *AIAA SPACE 2015 Conference and Exposition*.
- Intel.** (2024). *Accelerate Fast Math with Intel® oneAPI Math Kernel Library*. Retrieved from Intel - MKL: <https://www.intel.com/content/www/us/en/developer/tools/oneapi/onemkl.html#gs.3jvn79>
- Launcher.** (2023). *Launcher Calculator*. Retrieved from Launcher Calculator: <https://launchercalculator.com/>
- Lawson, C., Hanson, R., Kincaid, D., & Krogh, F.** (1979). Basic Linear Algebra Subprograms for Fortran Usage. *ACM Transactions on Mathematical Software*, 308-323.
- NASA.** (1969). *Saturn V Flight Manual, SA-507*. Huntsville: George C. Marshall Space Flight Center.
- NASA.** (1973). *Space shuttle guidance, navigation, and control design equations : Guidance*. Houston: NASA.
- NASA.** (1976). *U.S. Standard Atmosphere*. Washington D.C.: NASA.
- NumPy.** (2019). *Case Study: First Image of a Black Hole*. Retrieved from NumPy - Case Studies: <https://numpy.org/case-studies/blackhole-image/>

- Rocket Lab.** (2022, June). *Electron Payload User Guide 7.0*. Retrieved from Rocket Lab: <https://www.rocketlabusa.com/assets/Uploads/Electron-Payload-User-Guide-7.0.pdf>
- Rocket Lab.** (2024). *Electron - Rocket Lab*. Retrieved from Rocket Lab: <https://www.rocketlabusa.com/launch/electron/>
- Rosema, C., Doyle, J., Auman, L., & Underwood, M.** (2011). *Missile DATCOM User's Manual*. U.S. Army Aviation & Missile Research, Development and Engineering Center.
- Science, T. D. (n.d.).** *Is your Numpy Optimized for Speed*. Retrieved from Towards Data Science: <https://towardsdatascience.com/is-your-numpy-optimized-for-speed-c1d2b2ba515>
- Shuttlesource.** (2023). *STS-29 Space Shuttle Discovery Countdown & Launch*. Retrieved from Youtube: <https://www.youtube.com/watch?v=9l5mXuTBfRU>
- Song, E.-J., Cho, S., & Roh, W.-R.** (2015). A comparison of iterative explicit guidance algorithms for space launch vehicles. *Advances in Space Research*, 463-476.
- Space Launch Report.** (n.d.). *Rocket Lab Electron Data Sheet*. Retrieved from Space Launch Report: <https://web.archive.org/web/20220302121231/http://spacelaunchreport.com/electron.html>
- SpaceX.** (2022). *Eutelsat 10B Mission*. Retrieved from Youtube: <https://www.youtube.com/watch?v=QCpfvj1eiLs&t=1085s>
- SpaceX.** (2024). *SpaceX - Falcon 9*. Retrieved from SpaceX: <https://www.spacex.com/vehicles/falcon-9/>
- SpaceX.** (2024, January 7). *SpaceX Falcon 9 Launches Starlink 6-35*. Retrieved from Youtube: https://www.youtube.com/watch?v=_rXq2YDdYyk
- Steffens, M. J., Edwards, S. J., Mavris, D. N., Dees, P., & Diaz, M.** (2016). A Method for Launch Vehicle Performance Analysis via Surrogate Modeling. *Modeling and Simulation Technologies Conference*. AIAA.

- Sun, B., Park, Y. K., & Rae Roh, W.** (2010). Attitude Controller Design and Test of Korea Space Launch Vehicle-I Upper Stage. *International Journal of Aeronautical and Space Sciences*, 303-312.
- Tewari, A.** (2007). *Atmospheric and Space Flight Dynamics*. Birkhauser.
- Vukelich, S., Stoy, S., Burns, K., & Castillo, J.** (1988). *Missile Datcom Volume I - Final Report*. AFWAL (USA) Air Force Wright Aeronautical Laboratories.
- Waters, E., Garcia, J. B., B., P., A., H. J., & Threet, G.** (2013). NASA Advanced Concepts Office, Earth-To-Orbit Team Design Process and Tools. *AIAA SPACE 2013 Conference & Exposition*. AIAA. Retrieved from NASA: <https://www.nasa.gov/wp-content/uploads/2023/07/advanced-concepts.pdf>
- Wevolver.** (2023). *Falcon 9 v1.2 or Full Thrust - Block 5*. Retrieved from Wevolver: <https://www.wevolver.com/specs/falcon-9-v12-or-full-thrust-block-5>
- Zirwes, T., Zhang, F., Denev, J. A., Habisreuther, P., Bockhorn, H., & Trimis, D.** (2019). Improved Vectorization for Efficient Chemistry Computations in OpenFOAM for Large Scale Combustion Simulations. *High Performance Computing in Science and Engineering*, 209-224.

CURRICULUM VITAE

Name Surname : Ahmet Enes Yüceyurt

E-Mail :

EDUCATION :

- **B.Sc.** : 2019, Istanbul Technical University, Faculty of Aeronautics and Astronautics, Astronautical Engineering

PROFESSIONAL EXPERIENCE AND REWARDS:

- 2020 - Now: ROKETSAN Inc.