AN ABSTRACT OF THE THESIS OF

Trevor J. Rose for the degree of Master of Science in Mechanical Engineering presented on August 13, 2021.

Title: Optimization & Qualification of a Liquid Engine Launch Vehicle Design Using 6DoF Simulations

Abstract approved: _____

Kyle E. Niemeyer

In this work, I developed a six degree of freedom flight simulator specific to liquid engine launch vehicles. This flight simulator improves upon available software by accounting for liquid engine specific phenomena such as tank pressurization, liquid engine performance, and center of gravity modeling. Additionally, I implemented methods for optimization and design qualification. I implemented models for atmosphere and wind, pressure-fed liquid engine performance, propellant tank pressures, aerodynamics, and center of gravity position with multiple changing masses. I conducted a design study on a liquid engine launch vehicle targeting the Kármán line (100 km) using these tools. I used Bayesian optimization to reduce required propellant mass by 7.0 %. I used Monte Carlo methods to qualify the design to be 95 % reliable with less than 10 % consumer risk.

©Copyright by Trevor J. Rose
August 13, 2021
All Rights Reserved

Optimization & Qualification of a Liquid Engine Launch Vehicle Design Using 6DoF Simulations

by

Trevor J. Rose

A THESIS

submitted to

Oregon State University

in partial fulfillment of
the requirements for the
degree of

Master of Science

Presented August 13, 2021
Commencement June 2022

Master of Science thesis of Trevor J. Rose presented on August 13, 2021.

APPROVED:

Major Professor, representing Mechanical Engineering

Head of the School of Mechanical, Industrial, and Manufacturing Engineering

---

Dean of the Graduate School

I understand that my thesis will become part of the permanent collection of Oregon State University libraries. My signature below authorizes release of my thesis to any reader upon request.

Trevor J. Rose, Author

ACKNOWLEDGEMENTS

First, I'd like to acknowledge the late Dr. Nancy Squires. Dr. Squires cherished her students, encouraged us, and believed in us even when we didn't believe in ourselves. Her endless optimism and enthusiasm was contagious and inspired me to pursue dreams I never thought possible. She inspired a generation of Oregon State engineers to aim for the stars.

I'd also like to acknowledge the work of the Oregon State HALE team. This work was built on the foundation and contributions of each and every HALE team member spanning multiple years of hard work and dedication. I'd like to specifically acknowledge the significant contributions of Devon Burson, Olivia Clark, Conner Flaherty, Colt Harms, Muriel Hayden, and Adam Ragle.

Additionally, I'd like to acknowledge my many family and friends for their loving support, patience, and understanding. Particularly my parents, Malcolm and Susan Rose, and to my siblings Kim, Alan, and Nicholas. I'd like to thank Kyle O'Brien for his friendship and always being available to talk through coding problems.

Lastly, I'd like to thank Dr. Niemeyer and Dr. Blunck for their efforts in supporting the OSU AIAA program.

TABLE OF CONTENTS

TABLE OF CONTENTS (Continued)

LIST OF FIGURES

LIST OF FIGURES (Continued)

Figure Page

LIST OF TABLES

LIST OF ALGORITHMS

LIST OF ACRONYMS

1DoF One DoF. 19

3DoF Three DoF. 21, 61

6DoF Six DoF. 11, 19, 21, 121

CEA Chemical Equilibrium with Applications. 12, 17, 34, 36, 39, 119

DoF Degree of Freedom. xiii, 11, 19, 21, 61, 121

GRAM Global Reference Atmospheric Model. ix, 12, 17, 18, 32, 34, 35, 89, 90, 113

JANNAF Joint Army Navy NASA Air Force. 12, 34, 119

k-NN k-Nearest Neighbor. ix, xii, 14, 92–96, 118, 120, 122

KKT Karush—Kuhn—Tucker. 13, 66, 67

MMH Monomethylhydrazine. 4, 63

NACA National Advisory Committee For Aeronautics. 10, 54, 56

NASA National Aeronautics & Space Administration. 2, 11, 12, 14, 17, 31–34, 89, 90, 119

NOAA National Oceanic & Atmospheric Administration. 31

O/F Oxidizer : Fuel. x, 37, 59, 107, 108, 113

RHC Random Hill Climb. 13, 74–76, 79, 86

RP-1 Rocket Propellant 1. 4

RPA Rocket Propulsion Analysis. 12, 34

SVM Support Vector Machine. 95

TDK Two-Dimensional Kinetic. 12, 34, 119

LIST OF ACRONYMS (Continued)

TVC Thrust Vector Control. 7

UDMH Unsymmetrical Dimethylhydrazine. 4

USAF United States Air Force. 31

This work is dedicated to the late Dr. Nancy Squires and to all educators who inspire their students to pursue their dreams.

All models are wrong, but some are useful.

George E. P. Box

Chapter 1: Introduction

1.1 Historical Perspective

An important topic in rocketry is to understand and acknowledge the previous work and contributions that have led to modern day liquid propellant rockets. Understanding what decisions were made in designing rockets, and what failures have resulted from those decisions, is critical to a successful design.

The first major contributor to modern rocket development was Konstantin Tsiolkovsky around the turn of the 20th century. He developed the famous rocket equation, describing the idealized change in velocity of a vehicle based on its exhaust velocity and mass ratio of the vehicle [1]. He calculated velocities required to maintain steady orbit above Earth's atmosphere and developed the concept of multi-stage liquid propellant vehicles that could achieve this [1]. He also developed the theory of escape velocity, determining the minimum speed required to leave the orbit of Earth [1]. Tisolkovsky's work was largely theoretical, and gained little traction or testing to prove his theories at the time he published them.

In the United States, Dr. Robert Goddard began to apply the theory to experimentation. Despite being largely unaware of Tsiolkovsky's work, the two developed similar principles. Goddard obtained patents for his invention of the liquid propelled rocket, and the multi-stage rocket (both previously theorized by Tsiolkovsky, unbeknownst to Goddard) [1]. While Tsiolkovsky's work was largely theoretical, Goddard began experimenting. A critical breakthrough came when he applied the de Laval nozzle to his rocket engines, allowing him to increase efficiencies and exhaust velocities to supersonic speeds. Similar to Tsiolkovsky, Goddard faced push-back and did not see acceptance of his work immediately. In 1920, The New York Times went as far as to publish a story labeling to Goddard's theory on space travel as impossible [2].[1] Goddard's rockets, while critical for research and development, never reached space.

World War II became the next major driver of research and development into rockets. Wernher von Brahn, working with the Nazi Socialist Party, led the development of the

[1] Shortly after the launch of Apollo 11 to the moon, The New York Times issued a correction to their 1920 article.

V-2 rocket [1]. Immediately after the conclusion of the war, von Brahn was recruited by the United States government to assist in their development of rockets. Von Brahn led National Aeronautics & Space Administration (NASA) through the development of the Apollo program, culminating in what still remains the pinnacle of space flight: sending humans to the surface of the moon and back [1].

Simultaneously, Sergey Korolev worked with the USSR to develop their space program. In 1957, he achieved what Tsiolkovsky had theorized over 60 years earlier: placing an object in orbit around the Earth [1]. During the heat of the Cold War, the Sputnik satellite sparked the space race [1]. In the next monumental achievement for space travel, the Soviet space program led by Korolev successfully sent Yuri Gargarin to orbit for the first time.

Another monumental achievement in space travel was the development of the International Space Station. The International Space Station was an incredible achievement for space travel, but also signaled the collaboration between the United States and Russia in space after the fall of the Soviet Union [3]. As of the writing of this work, the International Space Station has had people living continuously on-board for over 20 years.

The pursuit of rockets has historically been exorbitantly expensive. According to the Office of Management and Budget, NASA spending during the Apollo program peaked at 4.41 % of the federal spending of the United States government—over eight times the share of federal spending NASA had in 2010 [4]. These unsustainable costs contributed to the end of the Apollo program. Now that rocket technology has been developed to perform these achievements, the focus of the industry has shifted. A blossoming commercial sector of space travel has begun to thrive, with a much larger emphasis placed on the economics and profitability of space travel.

One notable effort to change the economics of space travel has been Space Exploration Technologies (or SpaceX), led by CEO Elon Musk. The company has pushed the boundaries on space travel, developing reusable orbital class rockets. Similar to many of the previous contributors to space exploration, Musk received critiques and was often told his ideas were impossible—until they weren't. As of the writing of this work, SpaceX has landed their first stage boosters 84 times with 66 re-flights, and is now working on development of an even more cost effective vehicle [5]. SpaceX is joined by United Launch Alliance, Northrop Grumman, Rocket Lab, Sierra Nevada Corporation, and long list of startups who are trying to push even further in the work of developing cost effective

rockets.

It is clear from the trends seen in the aerospace industry, the future development in rocketry is in cost effective rockets. This thesis aligns well with the aerospace industry trends. My intent is the design and development of a low-cost vehicle capable of reaching space that can be manufactured by students at Oregon State University. This would serve as both a feasibility study and a technology demonstrator, allowing future researchers to develop more innovative vehicles and payloads while retaining the cost-effective approaches.

1.2 Flight Simulation

In design of liquid engine rockets, comprehensive simulations and analyses are important for functionality, mission success, and personnel safety. A few key outputs of the flight simulation are: engine thrust, propellant tank sizing, expected forces of flight, and projected recovery locations. Comparison between flight simulation and test data is critical for verification of the accuracy of the simulation.

Liquid engine rockets have existed for nearly a century, so the science behind these vehicles is well known. This thesis combines the existing research in liquid engine rockets with advances in model development for different aspects of flight simulation that improve upon available flight simulations.

1.3 Liquid Engine Rocket Design

This section is intended to provide a brief introduction to common aspects of design in liquid engine rockets as many of these components will be referred to throughout this work.

Typically, liquid engine rockets are long, cylindrical vehicles. When standing vertically, the top of these vehicles is referred to as the fore end, and the bottom as the aft end. These vehicles have a payload affixed to the top of the vehicle, followed by a propellant tanks, and then engines. These can be stacked to create multi-stage vehicles. The propellant tanks include an oxidizer and a fuel tank. Notable exceptions to this pattern are the Space Transportation System (commonly referred to as the Shuttle) and the Buran—these vehicles feature very different shapes based on unique mission requirements.

1.3.1 Propellants

Choice of propellants places limitations on the missions a vehicle is capable of. An example is the use of cryogenic propellants eliminates the vehicle from being rapidly used, such as a defense missile that may need to be stored with propellants for months or years after it is manufactured. Additionally, the choice of propellants may limit the materials available for use in the design of the vehicle. A brief summary of propellant choices are summarized in this section. A common oxidizer for commercial rockets is liquid oxygen (LOx), which can be paired with several common fuels. Fuels that are commonly used with LOx include liquid hydrogen, liquid methane, and Rocket Propellant 1 (RP-1) (a refined kerosene-based fuel) [6].

Many other rocket fuels and oxidizers have been tested. Within the missile industry, it is highly desirable to have storable propellants for readiness at any time. Therefore, LOx is not a favorable choice because it would boil off if held at room temperature for long periods of time. A couple of common choices for an oxidizer with storable propellants are dinitrogen tetroxide ($N_2\,O_4$) and nitric acid (HNO_3). A couple of prominent fuel choices to pair with dinitrogen tetroxide include hydrazine ($N_2\,H_4$), Monomethylhydrazine (MMH) ($CH_3\,(NH)NH_2$) and Unsymmetrical Dimethylhydrazine (UDMH) ($H_2\,NN\,(CH_3)_2$). These propellants are hypergols: fuel and oxidizer combinations that combust on contact, which can be particularly useful for cases such as the lunar descent engine for its simplicity and reliability.

An additional notable oxidizer and fuel combination is triethylaluminum ($Al_2(C_2H_5)_6$) and triethylborane ($(C_2H_5)_3B$), or TEA-TEB. This combination is particularly common for ignition procedures. The Saturn V F1 engines used TEA-TEB for ignition, as do the SpaceX Merlin engines [5, 6]. TEA-TEB is hypergolic, which makes it useful for engine ignition. After combustion with TEA-TEB has begun, both the F1 and Merlin transfer to a LOx-RP-1 propellant combination for the remainder of the engine burn.

1.3.2 Liquid Engine Components

A liquid engine is composed of several common components across all variants and designs. Working from upstream to downstream, the primary components are:

- Injector

- Combustion chamber
- Nozzle

A secondary component of the engine is called the power pack. The power pack is not required for engine functionality, but commonly used to increase engine performance.

1.3.2.1 Injector

Injectors can vary significantly between designs, but the primary function of an injector is to mix the fuel and oxidizer as they enter the engine to prepare for combustion. At the exit of the injector, the oxidizer and fuel can be liquid, gaseous, or a combination of the two. Sutton and Biblarz [7] provide a more in depth overview.

A common injector type is an impinging injector, where two or more streams of propellants are directed at each other to meet at a specific point, causing mixing of the propellants. Many of these impinging streams are typically placed on an injector face. The concept is simple, but requires many precisely drilled holes in the injector face. If the holes are not perfect, the streams will not meet at the target impinging point and propellant mixing will be poor. A notable use of this style of injector is the F-1 engine.

A coaxial swirl injector relies on directing flow around a hollow cylinder such that it gains an angular velocity. Upon exiting the bottom of the cylinder, the propellant will be propelled out in a cone. By placing an annulus surrounding the first, the second propellant is also directed out in a cone. The two cones of a coaxial swirl injector can be aligned, allowing mixing of propellants to occur on the entire surface of the cone for combustion. Engines can have many coaxial swirl injector elements on a single injector plate. The SpaceX Raptor engine uses a coaxial swirl injector.

Another common injector type is a pintle injector. Originally developed by TRW, this type of injector features a hollow cylinder-like element called a pintle that is closed at the end. Orifices are placed just before the end, forcing propellant radially outward when it reaches the pintle tip. On the outside of the pintle is an annulus that forces a cylinder of propellant along the pintle. When the cylinder of propellant reaches the radial flow at the pintle tip, the two mix to form a cone. This type of injector was used on the Lunar descent engine and by the SpaceX Merlin engine. The pintle injector was also selected for use in the liquid engine of focus in this work. Details of selection are outside

the scope of this work, but are provided by Burson et al. [8, 9] and Rose et al. [10].

1.3.2.2 Combustion Chamber

The next downstream component of the injector is the combustion chamber. The goal of this region is for fuel and oxidizer to combust completely. In reality, there will always be some losses due to incomplete combustion. Higher temperatures and pressures can increase the maximum thrust potential of the engine. Due to the combustion temperatures of rocket fuels, higher combustion temperatures can result in cooling challenges to prevent the walls of the combustion chamber from melting. Two solutions are commonly used to solve this: regenerative cooling and ablative cooling. In regenerative cooling, propellant flows through the walls of the combustion chamber before it is directed to the injector. Ablative cooling uses a material designed to slowly ablate at the combustion temperatures to protect the combustion chamber walls from the extreme temperatures.

1.3.2.3 Nozzle

Following the combustion chamber is the nozzle. The purpose of the nozzle is to convert the potential energy of the high pressure and high temperature gas in the combustion chamber into the maximum achievable exhaust velocity to increase thrust. The basic nozzle design of all liquid rocket engines is a de Laval nozzle, or a converging-diverging nozzle. The converging section of the nozzle increases the fluid velocity, up to sonic velocity at the throat (narrowest point of the nozzle). After the exhaust gasses reach sonic velocity, further reduction in area will not continue to increase the exhaust velocity. After reaching sonic velocity, increasing area can continue to accelerate the velocity of the fluid. Therefore, the diverging section of the nozzle allows for the exhaust velocity to reach supersonic speeds.

1.3.2.4 Power Pack

An optional component of a liquid rocket engine is the power pack. This component increases the pressure of the propellants before it is directed to the engine, increasing performance. The power pack utilizes turbomachinery to increase the pressure. This

has most commonly been done by burning a small fraction of the propellant to drive the turbomachinery, though Rocket Lab has successfully used electric motors and batteries to drive their turbopumps.

Using a power pack can increase performance, but also increases vehicle dry weight and complexity. For the engine of focus in this work, no power pack is used. Therefore, a complete explanation of the power packs and variants are considered outside the scope of this work.

Alternatively, the propellant tanks can be pressurized using a pressurant gas. This provides more simplicity by eliminating turbomachinery, but requires thicker tank walls to account for higher tank pressures required to maintain the same engine performance. Additionally, a pressurant tank is required to store the pressurant gas. This method has been selected for focus in this work, primarily for simplicity. More details of selection are provided by Burson et al. [8, 9] and Rose et al. [10].

1.3.3 Stablization

The most simple method for stabilizing a launch vehicle is to maintain the center of pressure below the center of gravity for the vehicle, referred to as passive stabilization. The aerodynamic forces on the vehicle will correct the vehicles orientation to align with the relative wind. This is most commonly done by placing fins at the rear of the vehicle. A major disadvantage of passive stabilization only is there is no direct control of the orientation or trajectory of the vehicle, it is dictated by the direction of relative wind. Variation in wind or wind gusts can cause the same vehicle to fly very different trajectories. Additionally, placing fins on the vehicle increases vehicle drag.

In commercial launch vehicles, Thrust Vector Control (TVC) is used almost ubiquitously for vehicle stabilization. TVC works by changing the axis of thrust to adjust the direction of the force vector applied to the vehicle and ensure it maintains the target heading. This is typically required for commercial launch vehicles, because payloads must be placed precisely into the correct target orbit. A combination of passive stabilization and TVC has been used historically such as on the Saturn V, though it is not commonly seen today. Passive stabilization has been selected for focus in this work, primarily for simplicity. More details of selection can be found in [8–10].

1.4 Motivation

Oregon State University is competing in the Base 11 challenge to develop a liquid fueled launch vehicle capable of crossing the Kármán line (100 km). The goal of this challenge is to develop cost effective space technology and increase the presence of liquid engine research at universities around the world. The Oregon State University team is currently testing and refining the design of our first liquid engine.

While there are many collegiate rocketry teams across the world, few liquid engine research teams exist at the collegiate level. This is primarily due to two reasons: the significant increase in complexity of liquid engines and the significant increase in cost. While work may be limited, some universities have been heavily involved in liquid engine development since the beginning of liquid engine development. Notably, Goddard performed much of his early research and experimentation at Clark University. A few of the notable successful liquid engine rocket teams at the collegiate level include: University of California, Los Angeles (12,500 ft apogee with an ethanol and LOx engine), San Diego State University (13000 ft apogee with a kerosene and LOx engine - a Rocketdyne LR101 engine was used), and University of Michigan (4000 ft apogee with a nitrous oxide and ethanol engine) [11, 12].

Accurate simulation of vehicle performance is a safety critical aspect of launching a liquid engine vehicle. The available simulation software do not accurately predict the center of gravity of a liquid engine because the mass flow from two propellant tanks and a pressurant tank differs from the mass flow from a single solid rocket motor. Additionally, there is no inclusion of propellant slosh models or tank pressurization, because these phenomena are not relevant to solid motor rockets. The available simulation software only allow for use of constant wind speeds, which is not applicable across the atmosphere from ground level to the Kármán line. Each of these limitations are addressed and improved upon by the flight simulation developed in this work.

At the time of writing this thesis, no commercially available software existed that could perform the required simulations without notable limitations. Many flight simulations are proprietary, so the majority of software available has been developed for use in the high-powered rocketry community. High-powered rocketry vehicles typically use solid propellant and do not fly as high as the Kármán line. Therefore, the available software has been tailored towards the majority of use cases for these applications. Because of

these limitations, I developed a flight simulation software specific to liquid engine vehicles.

After development of a flight simulation that addressed these limitations, two natural questions arise for how best to use it:

1. How can I use a flight simulation to assist in optimizing the design of a liquid engine launch vehicle?

2. How can I use a flight simulation to verify the vehicle design will reliably achieve the target mission despite uncertainties?

This work seeks to address limitations in available software and to answer both questions by developing a single method for optimizing the high level design parameters of a liquid engine vehicle design for reliably achieving a specified target mission. The validity of the method is demonstrated through application to the Oregon State University launch vehicle designed to reach the Kármán line, though the method could be applied to any target mission. By adjusting the flight simulation to a new application, the methods presented in this work could be extended to additional complex systems that are computationally expensive to model such as satellites or aircraft.

Chapter 2: Literature Review

2.1 Flight Modeling

In this section, I introduce relevant works related to development of models for a flight simulation software. I discuss several available simulation software, numerical methods commonly used, dynamics of a vehicle in flight, methods for tracking orientation between reference frames, atmospheric models, and engine performance models.

A major contribution to flight modeling used in this work is the OpenRocket software [13] and RASAero II software [14]. OpenRocket and RASAero II were both used heavily for reference in selection and development of models in the flight simulator for this work. Both are based around the Barrowman method for calculating aerodynamic coefficients [15]. In addition to the Barrowman method, Hilton provides more details on calculating aerodynamic coefficients in supersonic flow [16]. Zucker and Biblarz provide details on compressible flow, specifically flow through normal and oblique shocks [17]. The nosecone tip of a launch vehicle experiences a conical shock in supersonic flow, the Taylor—Maccoll solution for conical shocks is described by Zucrow and Hoffman [18] and Anderson [19]. Syverston and Dennis outline a second-order method for determining nosecone aerodynamic coefficients using the conical shock and a series of Prandtl—Meyer expansions for National Advisory Committee For Aeronautics (NACA) [20]. Pitts et al. outline methods for computing tail-body interference factors for NACA [21].

An important aspect of flight modeling is the numerical method used to solve the initial value problem. Euler [22] provided initial works in *Institutionum Calculi Integralis*. The Euler method works by taking small time steps successively to approximate the solution to an initial value problem. However, the Euler method only provides first-order accurate solutions, meaning a halving the step size will only half the error. To greatly reduce the error in a solution becomes computationally expensive, therefore it is not an ideal solution.

A summary of the most important works relating to the Runge—Kutta methods was provided by Butcher [23]. In 1895, Runge [24] developed several methods which

improved upon the classical Euler method using the midpoint rule and the trapezoidal rule. Heun improved on Runge's work, developing methods to allow for fourth-order accurate integration. Kutta [25] generalized the work and developed the classifications for all fourth-order Runge—Kutta methods, while also introducing the RK4 method commonly used today. Nyström extended the work to systems of differential equations, allowing for higher order differential equations to be solved as a system of first-order equations. Lastly, Butcher fully classified the Runge—Kutta equations and developed the Butcher tableau which is now the standard method for displaying any Runge—Kutta method. The work of these individuals is commonly referred to as the Runge—Kutta methods today.

Fehlberg developed an approach to adaptively control step size of Runge—Kutta methods to maintain highly accurate solutions by using two Runge—Kutta methods with different orders of accuracy [26]. In unpublished internal technical report for NASA, Sarafyan used a similar approach to Fehlberg obtain both fourth- and fifth-order accurate solutions for the same time step, allowing the error of the time step to be evaluated [27]. Later, Fehlberg improved on Sarafyan's work and published the a fourth- and fifth-order method, now commonly referred to as the Runge—Kutta—Fehlberg fourth- and fifth-order method or RKF45 [27]. Fehlberg's work was also for NASA and specifically intended for algorithms that could limit the maximum error on a time step during integration, resizing as necessary. This allows for fewer time steps to be used and significantly reduces computation time. Press et al. provide a summary of the RKF45 method with example implementation details [28]. The RKF45 method has gained widespread use and is commonly applied to initial value problems in physics simulations. The RKF45 method is the primary numerical method used in this work.

Different forces and moments are present at different phases throughout a flight. Peng et al. summarize the dynamics of a sounding rocket in flight [29]. In a Six Degree of Freedom (DoF) (6DoF) flight simulation, forces and moments are typically easiest to calculate in a body reference frame, but results are most useful in a world reference frame. This makes maintaining accurate orientations between different reference frames critical to accuracy of the simulation. Quaternions have been successfully implemented as a standard method for maintaining orientation in aerospace simulations [13]. Quaternions were originally developed by Hamilton as an extension to complex numbers. Unit quaternions are useful and follow many similar rules to complex numbers [30–32]. Quaternions have

shown to be more computationally efficient than using Euler angles [33].

In 1976, NASA developed a standard atmospheric model that is still in use today: the US Standard Atmospheric Model (1976) [34]. The US Standard Atmospheric Model (1976) is based on both theoretical models and empirical data and is very useful as a baseline in flight modeling [35]. However, the US Standard Atmospheric Model (1976) does not account for temporal or spatial variations. Additionally, the US Standard Atmosphere (1976) does not make predictions about wind, or about noise and variation from the mean value.

After release of the US Standard Atmosphere (1976), NASA continued development of atmospheric models. A software called Earth-Global Reference Atmospheric Model (GRAM) has been under continual development and refinement since its first release in 1990 until the current version, Earth-GRAM 2016 [36–41]. This version accounts for spatial and temporal variations, along with noise variation from the mean value on an hourly or monthly basis. This model is used by NASA for Monte Carlo simulations.

Several models have been studied for analyzing engine performance. NASA developed the Chemical Equilibrium with Applications (CEA) software for combustion analysis [42, 43]. Additional alternatives include a Two-Dimensional Kinetic (TDK) model developed by Joint Army Navy NASA Air Force (JANNAF) [44] and Rocket Propulsion Analysis (RPA) software [45]. The original Fortran code [42, 43] has been accessed via the *RocketCEA* Python library by Taylor [46]. Additionally, isentropic flow through nozzles is used as outlined by Zucker and Biblarz [17] and Suttona and Biblarz [7].

2.2 Design

In this section, I discuss works related to the design of a liquid engine vehicle. I cover existing work that has been done at Oregon State University, which my work directly builds off of. Additionally, I introduce classical optimization techniques and several optimization algorithms which I used to perform design optimization.

Many of the initial design decisions were made as a part of the Oregon State University liquid engine rocket team and were not exclusive to this work. Three major design reports completed by the team, including the author, were submitted to a panel of aerospace experts for critical review [8–10]. Harms et al. performed design work on a thrust vector control system [47]. The most important topics from these as related to this work are

summarized, though many additional details and considerations regarding design decisions that do not effect the methods presented have not been included for conciseness.

Optimization is an active field of research. One early work in optimization was done by William Karush in his masters thesis, where conditions for establishing optimality of a point with constraints were derived [48]. Upon finishing his thesis, Karush quickly changed focus to his doctoral dissertation and his work went largely unnoticed. Years later, the work was independently derived and further generalized by Kuhn and Tucker [49, 50]. This became widely known as the Karush—Kuhn—Tucker (KKT) Theorem.

The design problem of interest can be modeled as a tree with different layers for each discrete design decision to be made. Design bias can be eliminated by searching the tree to find the best design. Short et al. [51] explored tree search methods which could be implemented for this problem. A couple options include algorithms such as Dijkstra's method and A* as presented by Dijkstra [52] and Hart et al. [53] respectively. Implementing a tree search algorithm was not fully implemented in this work due to increased model complexity and computation time required. Instead, two design decision methods that have been implemented are Pugh charts and trade off studies as outlined by Otto and Wood [54], see [8–10] for implementation details.

A useful and simple search algorithm for optimization is Random Hill Climb (RHC). The RHC algorithm does not require gradient information, which makes it a good choice for black-box optimization problems [55].

Kirkpatrick et al. developed an algorithm that was analogous to annealing process in statistical mechanics to grow crystals [56]. Arora also summarizes the method and provides examples of implementation [57]. The method builds on a method developed for statistical mechanics to determine the probability of a particle to be in a higher energy state based on the temperature, called the Metropolis criteria. Kirkpatrick et al. related this concept to design optimization by implementing an effective temperature and a artificial cooling schedules to function as a stochastic global optimization procedure. Kirkpatrick et al. showed this algorithm to work well in the design of computer chips, but also showed the algorithm could be extended to more broad problems if transitions within the design space can be constructed. An implementation of simulated annealing is available via the *SciPy* Python library [58].

Kennedy and Eberhart developed a particle swarm optimization algorithm [59]. Arora also summarizes the method and provides examples of implementation [57]. The algorithm

is intended to mimic nature, similar to how a flock of birds can quickly find a food source based on the benefit of the collective group searching a large area at once. An implementation of particle swarm is available via the *pyswarms* Python library [60].

The Bayesian optimization algorithm was explored by Mockus as a global optimization method [61]. Mockus et al. further explore the method and outline the branch and bound method for problems where design variables are discrete [62]. This method fits a stochastic process on a set of observations, which can be used to determine the ideal sampling point by optimizing an acquisition function. Common acquisition functions include expected improvement, upper confidence bound, and probability of improvement [63]. Specifically, Gaussian process regression is commonly used to develop a surrogate model, though other surrogate models are possible. The optimization of the acquisition function and building a surrogate model makes the method ideal for optimizing expensive black-box objective functions. An implementation of Bayesian optimization is available as the *bayesian-optimization* Python library [64].

2.3 Design Qualification

Hanson and Beard outlined a method for performing Monte Carlo simulations that is used as the NASA standard [65]. The Monte Carlo method in this application relies on randomly sampling parameters with uncertainty and evaluating mission performance repeatedly. After many simulations, the vehicle performance subject to uncertainty can be estimated. The methods outlined by Hanson and Beard are directly applicable and followed closely in this work to match accepted industry standards. This method uses consumer risk to evaluate the confidence in results produced from a set of Monte Carlo simulations, which is commonly used for designing sampling plans for reliability engineering in mass production. Birolini outlines a standard method for computing consumer risk to evaluate a sampling plan [66].

A method that can be used for classification of similar points was introduced by Fix and Hodges [67] called the k-Nearest Neighbor (k-NN) method. This method assumes the design space is smooth and nearby neighboring points can be used to reasonably approximate the point of interest [68].

Chapter 3: Methods

3.1 Flight Modeling

The first step to development of a simulation of the vehicle flight is to establish a representative model of the vehicle and all forces that will act on it. The primary forces that act on the vehicle are: gravity, thrust, drag, and a normal force. Thrust comes from combustion of the liquid propellants and results in a force vector along the axis of thrust. The axis of thrust is typically aligned with the axis of the vehicle, except when active stabilization is used. Drag is aligned with the axis of the vehicle and depends on the velocity of the vehicle and the density of the atmosphere. The direction of relative wind is the velocity of the fluid in the reference frame of the vehicle, which represents the sum of wind vector and the negative of the vehicles velocity vector. If the relative wind vector is not aligned with the axis of the vehicle, the vehicle will have an angle of attack. When flying at a non-zero angle of attack, the nosecone and fins produce lift that results in a net normal force on the vehicle. The drag and normal force act at the center of pressure of the vehicle, while gravity acts at the center of gravity. Shown in Figure 3.1 is a basic free body diagram to represent these forces.

The sum of the forces and sum of the moments equations can be performed in each coordinate directions, for a total of six equations. Using Newton's law and Euler's equations, each of these can be written as the second-order differential equations:

$$\Sigma F_x = m\ddot{x} \tag{3.1}$$

$$\Sigma F_y = m\ddot{y} \tag{3.2}$$

$$\Sigma F_z = m\ddot{z} \tag{3.3}$$

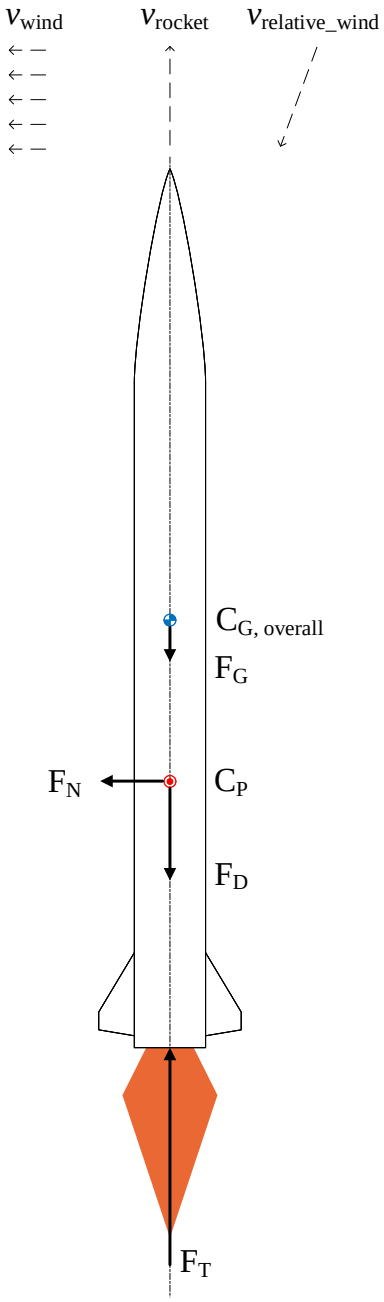

Figure 3.1: Free body diagram of a rocket.

$$\Sigma M_x = I_x\ddot{\Omega}_x + (I_z - I_y)\Omega_y\Omega_z \tag{3.4}$$

$$\Sigma M_y = I_y\ddot{\Omega}_y + (I_x - I_z)\Omega_x\Omega_z \tag{3.5}$$

$$\Sigma M_z = I_z\ddot{\Omega}_z + (I_y - I_x)\Omega_x\Omega_y \tag{3.6}$$

The starting point of the vehicle is at zero altitude and zero velocity on the launch rail, so I can solve Equations (3.1)-(3.6) for the trajectory throughout flight by integrating as an initial value problem.

3.1.1 Simulation Structure

Each of the elements described within Section 3.1 were coded in Python using an object oriented approach. The overall structure of the code is shown in Figure 3.2, where each block represents an object within the code. A simulation can be called by running the *Flight_MASTER* script and passing in two *JSON* files which contain relevant engine and rocket properties. The *JSON* files are populated with properties such as component weights, lengths within the airframe, engine burn time, vehicle outer diameter, propellant mass fraction, rail properties, design chamber pressures, nozzle throat and exit diameters, nozzle diverging half angle, and more. The intent of using these *JSON* files is to develop vehicle files that correspond to a specific vehicle configuration that are easy to modify. This means the simulation could quickly and easily be run for entirely different vehicles. While the *JSON* method is flexible and easy to integrate with Python, it is less user friendly than alternatives such as OpenRocket [13] and RASAero II [14] which feature a very intuitive user interfaces. Future work could take inspiration from OpenRocket and RASAero II to improve the user interface.

The *Flight_MASTER* script initializes a *Simulation* object, which contains both a *Rocket* object and a *Physics* object. The *Rocket* object contains all relevant methods and properties specific to the rocket, while the *Physics* object contains all of the relevant methods and properties to calculate the vehicle's state at a future time step. Additional objects, such a *LiquidEngine* object, were created to encapsulate all relevant properties and methods specific to a liquid engine within the *Rocket* object. Not all levels of objects within the code are represented in Figure 3.2 for simplicity.

Additionally, the code interfaces with external code for calculations such as the NASA CEA Fortran code through the *RocketCEA* package. The NASA Earth-GRAM code is

another external code to interface with, due to current implementation issues this is simply done by running the Earth-GRAM code and logging data separately for a maximum number of simulations to be conducted, then passing a database into the flight simulation. Future work will include calling the Earth-GRAM code directly from the simulation.

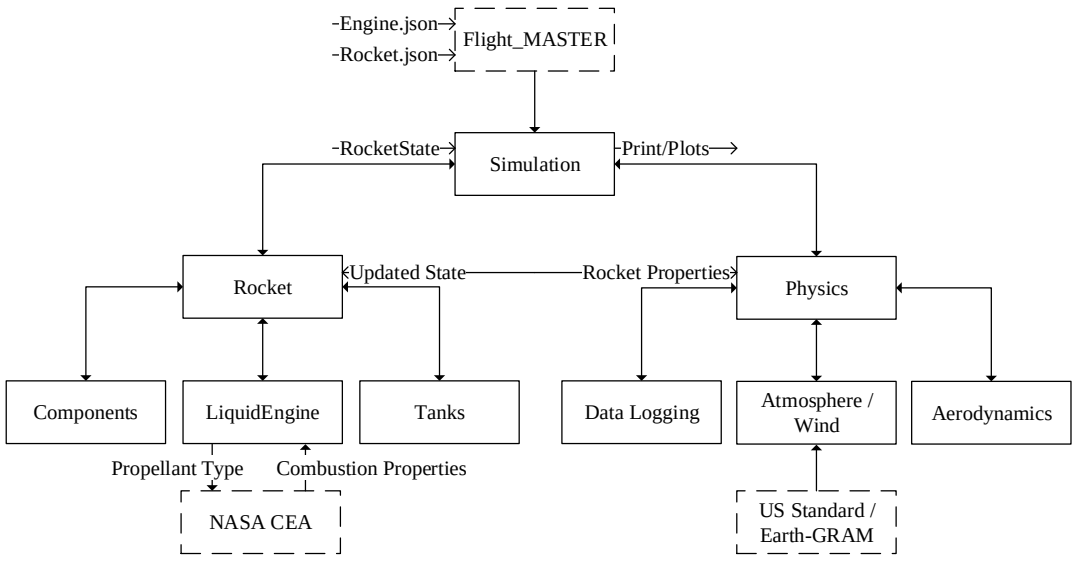

Figure 3.2: Structure of simulation code to combine simulation elements.

Additionally, a state machine is used to allow simple transitions between different phases of flight. In each state, the vehicle may have different characteristics. The possible states in the state machine are described in Section 3.1.2. This is important, because the physics in each of these phases may be unique. Use of a state machine makes the distinct differences between the different phases of flight simple. Due to this setup, adding additional launch phases, such as a second stage engine is simple by adding a transition for the change.

To call a simulation, the user simply provides *JSON* files with the properties of interest and executes the *Flight_MASTER* script.

In Section 3.2, the *Simulation* initialization was modified slightly to allow optimization methods to adjust high level vehicle parameters. Additionally, in Section 3.3, the *Simulation* initialization is again modified to allow properties with uncertainty to be

modified based on sampling a distribution. With the existing simulation, modifying this simulation initialization for both of these use cases is straightforward, though care must be taken that all of the *Rocket* object properties are properly initialized. To do this, *getter* and *setter* methods are used frequently in the code to modify applicable parameters safely. For example, it is useful to vary the outer diameter of the vehicle during design optimization, however simply adjusting the OD value would have no effect if only the A_{ref} value is used in drag calculations. A diameter *setter* method would simply update both the diameter and area values at the same time.

This object oriented approach allows great flexibility to be able to run standard flight simulations, implement optimization algorithms, or estimate design reliability by simply modifying how the *Simulation* object is initialized.

3.1.2 Launch Phases

The vehicle passes through several distinct phases throughout the flight which dictate specific parameters and place constraints on the launch vehicle. Figure 3.3 shows a visual representation of each of the distinct phases and milestones in a standard mission for a single stage launch vehicle with a parachute recovery. For more than one stage or different recovery methods, there are variations on the standard mission profile. Each of these phases are easily represented in a flight simulation through the use of a state machine. For this work, I broke the state machine into the following phases:

1. **Launch Rail:** The vehicle is restricted to a launch rail, allowing only linear motion in the direction of the launch rail in One DoF (1DoF). There is a non-zero propellant mass flow rate during this phase, because the engine is firing. All moments are set to zero and the equations of motion follow Equations (3.1)-(3.3), including a frictional force between the vehicle and rail. Peng et al. [29] proposed a model also that also includes a planar motion phase of flight after the fore rail attachment point has exited the rail, but the aft is still on the rail. Due to the very short span of the planar motion phase, I considered it to be part of the powered ascent phase.

2. **Powered Ascent:** After exiting the launch rail, the engine is producing thrust and the vehicle is capable of moving in 6DoF. There also is a non-zero propellant mass flow rate during this phase. The equations of motion are described by

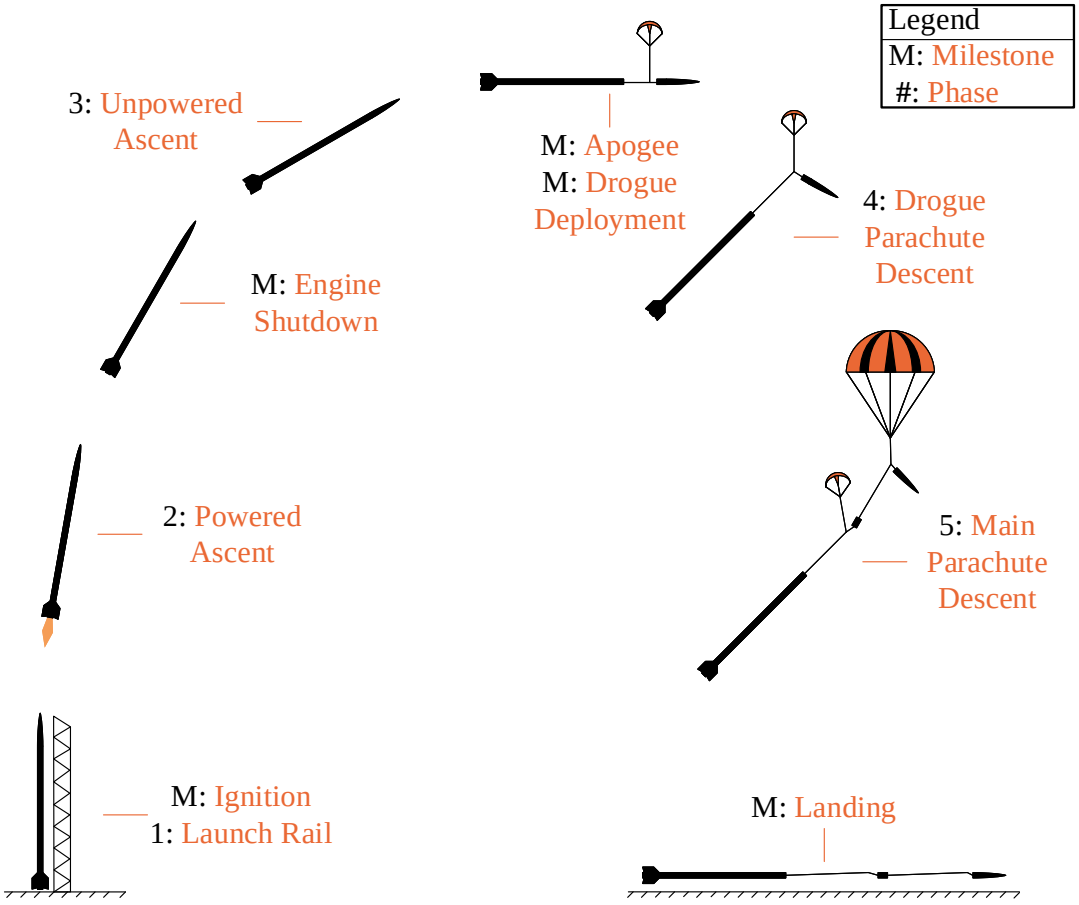

Figure 3.3: Representation of a standard mission profile.

Equations (3.1)-(3.6). This phase ends when the a propellant mass reaches zero.

3. **Unpowered Ascent:** After main engine cutoff, the vehicle is no longer producing thrust but is still subject to motion in 6DoF. For part of this phase, pressurant gas will continue to flow as the tanks depressurize. The vehicle will continue in this phase until it reaches a peak at apogee and begins descending. The equations of motion are described by Equations (3.1)-(3.6).

4. **Drogue Parachute Descent:** Immediately following apogee, the drogue parachute is deployed. The vehicle begins descending in multiple tethered sections. For simplicity, I modeled this phase of flight as a Three DoF (3DoF) system, only accounting for translation. This eliminates the need to model the complex multi-body dynamics, but should still provide reasonable estimations of recovery performance. The coefficient of drag and reference area are modified based on the drogue parachute parameters. The equations of motion follow Equations (3.1)-(3.3). More advanced models including multi-body dynamics and parachute inflation models are available, such as those proposed by Peng et al. [29]. My primary focus in this work was on the ascent phase, so I assumed a simple 3DoF translational model to be sufficient.

5. **Main Parachute Descent:** Upon descending to a set altitude, a main parachute is deployed to slow the vehicle to a safe landing velocity. Again, the vehicle is falling in multiple tethered sections during this phase and I chose to modeled it as a 3DoF system to avoid unnecessary complexities. The coefficient of drag and reference area are modified based on the main parachute parameters. The equations of motion follow Equations (3.1)-(3.3).

3.1.3 Numerical Integration Techniques

I used Runge—Kutta numerical methods in this work because they offer high orders of accuracy which allow a coarser grid of time steps to be used. A first-order differential equation,

$$\frac{dx}{dt} = f(t, x), \tag{3.7}$$

is what Runge—Kutta methods are intended to solve. Specifically, Runge—Kutta methods are used for initial value problems, where $f(t_0, x_0)$ is known. The Runge—Kutta

fourth-order method to find a solution at a point $n+1$ is shown in Equations (3.8)- (3.13).

$$k_1 = f(t_n, x_n) \tag{3.8}$$

$$k_2 = f(t_n + \frac{1}{2}\Delta t, x_n + \frac{1}{2}\Delta t k_1) \tag{3.9}$$

$$k_3 = f(t_n + \frac{1}{2}\Delta t, x_n + \frac{1}{2}\Delta t k_2) \tag{3.10}$$

$$k_4 = f(t_n + \Delta t, x_n + \Delta t k_3) \tag{3.11}$$

$$x_{n+1} = x_n + \frac{1}{6}\Delta t\,(k_1 + 2k_2 + 2k_3 + k_4) + \mathcal{O}(\Delta t^5) \tag{3.12}$$

$$t_{n+1} = t_n + \Delta t \tag{3.13}$$

The Runge—Kutta fourth-order method could be expressed in a Butcher tableau as shown in Table 3.1.

Table 3.1: Runge—Kutta fourth-order Butcher tableau

0				
1/2	1/2			
1/2	0	1/2		
1	0	0	1	
	1/6	1/3	1/3	1/6

The general form of a Butcher tableau is shown in Table 3.2 for m stages.

Table 3.2: General Butcher tableau

b_1				
b_2	a_{21}			
b_3	a_{31}	a_{32}		
$\vdots$	$\vdots$	$\vdots$	$\ddots$	
b_m	a_{m1}	a_{m2}	$\cdots$	a_{mm}
	c_1	c_2	$\cdots$	c_m

Using the variable definitions presented in Table 3.2, general forms of the Runge—

Kutta method of order P are expressed as

$$k_i = f(t_n + b_i \Delta t, x_n + \Delta t \sum_{j=1}^{i-1} a_{ij} k_j) \tag{3.14}$$

$$x_{n+1} = x_n + \Delta t \sum_{j=1}^{m} c_j k_j + \mathcal{O}^{P+1}. \tag{3.15}$$

While Runge—Kutta schemes can provide high orders of accuracy, they lack any information regarding the optimal step size for any given iteration. A method for evaluating the accuracy of each step size would be to solve the same initial value problem twice, once with a coarse grid and once with a grid spacing of half the size. For a fourth-order Runge—Kutta method, the finer grid size should have error reduced by 16 times the coarse grid size. When comparing the two grids, if the solutions are nearly the same then the solution can be said to be grid independent. Because this method requires solving the same problem twice with two different grid sizes, it is computationally inefficient and not desirable.

Fehlberg was able to obtain both a fourth- and fifth-order accurate solution requiring only one additional function evaluation between the two [27]. Table 3.3 is a Butcher table for the methods, where the bottom two rows correspond to the fourth- and fifth-order accurate methods respectively.

Table 3.3: Runge—Kutta—Fehlberg 4th- and 5th-order Butcher tableau

0						
$1/4$	$1/4$					
$3/8$	$3/32$	$9/32$				
$12/13$	$1932/2197$	$-7200/2197$	$7296/2197$			
1	$439/216$	-8	$3680/513$	$-845/4104$		
$1/2$	$-8/27$	2	$-3544/2565$	$1859/4104$	$-11/40$	
	$25/216$	0	$1408/2565$	$2197/4104$	$-1/5$	0
	$16/135$	0	$6656/12825$	$28561/56430$	$-9/50$	$2/55$

When applied to a initial value problem, Fehlberg's technique allows for each successive step to be evaluated for its error. The error between the two evaluations can be expressed

as

$$\epsilon = x_{n+1} - x^*_{n+1} = \sum_{i=1}^{m} (c_i - c^*_i) k_i, \tag{3.16}$$

where x_{n+1} is obtained from the higher order computation and x^*_{n+1} from the lower order. By assuming the error in the higher order approximation is small compared to the lower order approximation, ϵ is approximately the error in the time step. Because the order of accuracy of the method is known, an expression can be used to obtain error of an arbitrary size by adjusting the size of the step. I used the relation

$$\frac{\Delta t^P}{(\Delta t^+)^P} = \frac{\epsilon}{\epsilon^+} \tag{3.17}$$

to find a target step size Δt^+ for a specified desired accuracy ϵ^+ [28]. For a system of first-order differential equations, Equation (3.16) will return a vector $\boldsymbol{\epsilon}$. In this case, Equation (3.17) uses the element of the largest magnitude in the $\boldsymbol{\epsilon}$ vector [28].

To use the relation in Equation (3.17) efficiently, simple logic can be used. If the error of the current step is larger than the desired accuracy, then reduce Δt and repeat the step. If the error of the current step is smaller than the desired accuracy, then accept the step and adjust Δt for the next step. The value obtained from higher order of accuracy calculation is always the one retained for future steps. Shown in Algorithm 3.1 is sample psuedo-code for RKF45. Press et al. proposes that when reducing step size using RFK45, the exponent $1/P = 1/5$ should be replaced with $1/4$ to prevent the cases where the reduced step size is still not below the desired accuracy ϵ^+ [28].

This method allows a coarser grid spacing to be used in regions where the solution is not changing rapidly. However, when the solution is changing rapidly, a smaller grid spacing can be used. As applied to the vehicle simulation, this is particularly useful in scenarios where the vehicle is not experiencing many changing forces. In the upper atmosphere after engine cutoff, drag forces are near zero. This means the only force acting on the vehicle is gravity, which is constant and does not produce a moment. Coarse grid spacing in this region is expected and allows for quicker computation.

Most equations being numerically solved in this simulation are second-order differential equations. Second-order differential equations are decomposed to a pair of first-order differential equations. For example, Equation (3.1) is decomposed as

Algorithm 3.1: Runge—Kutta—Fehlberg fourth- & fifth-order method

Input: $t_0, \boldsymbol{x}_0, f(t, \boldsymbol{x}), \Delta t_0, \epsilon^+$
Result: $\boldsymbol{x}(t)$, t
$n = 0$
$t = \{t_0\}$
$x = \{\boldsymbol{x}_0\}$
$\Delta t = \Delta t_0$
while $t_n < t_f$ **do**
  $k_1 = f(t_n, \boldsymbol{x}_n)$
  $k_2 = f\left(t_n + \frac{1}{4}\Delta t, \boldsymbol{x}_n + \frac{1}{4}k_1\Delta t\right)$
  $k_3 = f\left(t_n + \frac{3}{8}\Delta t, \boldsymbol{x}_n + \left(\frac{3}{32}k_1 + \frac{9}{32}k_2\right)\Delta t\right)$
  $k_4 = f\left(t_n + \frac{12}{13}\Delta t, \boldsymbol{x}_n + \left(\frac{1932}{2197}k_1 - \frac{7200}{2197}k_2 + \frac{7296}{2197}k_3\right)\Delta t\right)$
  $k_5 = f\left(t_n + \Delta t, \boldsymbol{x}_n + \left(\frac{439}{216}k_1 - 8k_2 + \frac{3680}{513}k_3 - \frac{845}{4104}k_4\right)\Delta t\right)$
  $k_6 = f\left(t_n + \frac{1}{2}\Delta t, \boldsymbol{x}_n + \left(\frac{-8}{27}k_1 + 2k_2 - \frac{3544}{2565}k_3 + \frac{1859}{4104}k_4 - \frac{11}{40}k_5\right)\Delta t\right)$
  $\boldsymbol{x}^*_{\text{new}} = \boldsymbol{x}_n + \Delta t\left(\frac{25}{216}k_1 + \frac{1408}{2565}k_3 + \frac{2197}{4104}k_4 - \frac{1}{5}k_5\right)$
  $\boldsymbol{x}_{\text{new}} = \boldsymbol{x}_n + \Delta t\left(\frac{16}{135}k_1 + \frac{6656}{12825}k_3 + \frac{28561}{56430}k_4 - \frac{9}{50}k_5 + \frac{2}{55}k_6\right)$
  $\epsilon = \boldsymbol{x}_{\text{new}} - \boldsymbol{x}^*_{\text{new}}$
  if $\epsilon < \epsilon^+$ **then**
    $x_{n+1} = \boldsymbol{x}_{\text{new}}$
    $t_{n+1} = t_n + \Delta t$
    $\Delta t = \Delta t\left(\frac{\epsilon}{\epsilon^+}\right)^{1/P}$
    $n = n + 1$
  else
    $\Delta t = \Delta t\left(\frac{\epsilon}{\epsilon^+}\right)^{1/P}$
$t = t$
$\boldsymbol{x}(t) = x$

$$\frac{dx}{dt} = \dot{x}(t) \tag{3.18}$$

$$\frac{d\dot{x}}{dt} = \Sigma F_x/m \tag{3.19}$$

and solved explicitly using Runge—Kutta—Fehlberg techniques. This can be done for each coordinate direction and similarly for angular position and velocity. An equation for change of mass of the overall vehicle can be related to the mass flow rate of propellant. The mass flow rate

$$\frac{dm}{dt} = -\dot{m}_{\text{propellant}}(t) \tag{3.20}$$

is a function of time, after engine cutoff propellant mass flow rate goes to zero.

3.1.4 Orientation & Reference Frames

In flight modeling, a simple method of tracking the orientation of the vehicle is to use Euler angles. Three Euler angles can be used to specify the desired rotation. A common choice for tracking orientation is to use a fixed reference frame with respect to Earth (North, East, Down is a common in aerospace). However, an inertial reference frame aligned with the body of the vehicle is easiest for calculating forces and moments on the vehicle. Euler angles provide a method of rotating between reference frames. This allows the vehicle to be tracked in a fixed Earth frame, but physics calculations to be performed in the inertial frame aligned with the body of the vehicle.

Euler angles specify the angle between each axis of the two reference frames, resulting in three angles. Rotation matrices can be used to convert between reference frames. Rotation matrices are shown in Equations (3.21)-(3.23):

$$R_x(\phi) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\phi & -\sin\phi \\ 0 & \sin\phi & \cos\phi \end{bmatrix} \tag{3.21}$$

$$R_y(\theta) = \begin{bmatrix} \cos\theta & 0 & \sin\theta \\ 0 & 1 & 0 \\ -\sin\theta & 0 & \cos\theta \end{bmatrix} \tag{3.22}$$

$$R_z(\psi) = \begin{bmatrix} \cos\psi & \sin\psi & 0 \\ -\sin\psi & \cos\psi & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{3.23}$$

To convert a vector from world frame to inertial frame, the matrices are multiplied. The order of rotation matrix multiplication matters, so a common standard is to define $R = R(\phi, \theta, \psi) = R_x(\phi)R_y(\theta)R_z(\psi)$.

$$\begin{aligned} R &= \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\phi & -\sin\phi \\ 0 & \sin\phi & \cos\phi \end{bmatrix} \begin{bmatrix} \cos\theta & 0 & \sin\theta \\ 0 & 1 & 0 \\ -\sin\theta & 0 & \cos\theta \end{bmatrix} \begin{bmatrix} \cos\psi & \sin\psi & 0 \\ -\sin\psi & \cos\psi & 0 \\ 0 & 0 & 1 \end{bmatrix} \\ &= \begin{bmatrix} \cos\theta\cos\psi & \cos\theta\sin\psi & -\sin\theta \\ \cos\psi\sin\theta\sin\phi - \cos\phi\sin\psi & \cos\phi\cos\psi + \sin\theta\sin\phi\sin\psi & \cos\theta\sin\phi \\ \cos\phi\cos\psi\sin\theta + \sin\phi\sin\psi & -\cos\psi\sin\phi + \cos\phi\sin\theta\sin\psi & \cos\theta\cos\phi \end{bmatrix} \end{aligned} \tag{3.24}$$

Using the matrix in Equation (3.24), a general rotation can be performed. Equation (3.25) shows an example of how to use rotation matrices to change a position vector, $\boldsymbol{p}$, from a fixed frame to an inertial frame:

$$\boldsymbol{p}' = R\boldsymbol{p}\ . \tag{3.25}$$

While Euler angles are a simple and useful method for converting between reference frames, they are subject to singularities. Two Euler angles can become aligned when a vehicle passes through an angle of $\pi/2$ radians. This reduces the degrees of freedom of the system to two. This loss in degrees of freedom is not represented by physical forces

and moments on the vehicle, instead it is a failure in the Euler angles to provide the required orientation and position tracking throughout flight.

Quaternions can be used instead of Euler angles for rotations and tracking the vehicle throughout flight. Particularly, unit quaternions are useful for rotations. A unit quaternion is a four dimensional hypersphere with a radius of unity. Because they have four dimensions, additional information about the desired rotation can be stored in a quaternion. An example of a quaternion is shown in Equation (3.26).

$$q = \begin{bmatrix} q_0 \\ q_1 \\ q_2 \\ q_3 \end{bmatrix} \tag{3.26}$$

A quaternion has one real part and three orthonormal imaginary parts. The quaternion in Equation (3.26) can be rewritten as:

$$q = q_0 + q_1\hat{i} + q_2\hat{j} + q_3\hat{k} \tag{3.27}$$

Where $\hat{i}, \hat{j}, \hat{k}$ are principal coordinate axes for imaginary numbers. Similar rules to complex numbers apply when multiplying complex parts of quaternions, $i^2 = j^2 = k^2 = ijk = -1$. Additionally, $ij = k, ji = -k, jk = i, kj = -i, ki = j, ik = -j$ [30, 31]. From these rules, quaternion multiplication is defined as shown in Equation (3.28). The $\otimes$ symbol is used to indicate quaternion multiplication.

$$q_A \otimes q_B = \begin{bmatrix} a \\ b \\ c \\ d \end{bmatrix} \otimes \begin{bmatrix} e \\ f \\ g \\ h \end{bmatrix} \equiv \begin{bmatrix} ae - bf - cg - dh \\ af + be + ch - dg \\ ag - bh + ce + df \\ ah + bg - cf + de \end{bmatrix} \tag{3.28}$$

Other important properties of quaternions are their norm, conjugate, and inverse. These are defined according to Equations (3.29)-(3.31) [30] as:

$$|q| = \sqrt{q_0^2 + q_1^2 + q_2^2 + q_3^2} \tag{3.29}$$

$$q^* = \begin{bmatrix} q_0 \\ -q_1 \\ -q_2 \\ -q_3 \end{bmatrix} \tag{3.30}$$

$$q^{-1} = \frac{q^*}{|q|} \tag{3.31}$$

A unit quaternion is expressed by specifying a rotation angle, θ, and an axis of rotation as a unit vector, $\boldsymbol{\beta}$. This information would be stored in a unit quaternion,

$$q = \begin{bmatrix} \cos\frac{\theta}{2} \\ \sin\frac{\theta}{2}\beta_x \\ \sin\frac{\theta}{2}\beta_y \\ \sin\frac{\theta}{2}\beta_z \end{bmatrix} . \tag{3.32}$$

A rotation is performed using quaternions as

$$p' = q \otimes p \otimes q^{-1} \tag{3.33}$$

where p represents a quaternion that is rotated to a new coordinate system by the quaternion q.

As applied to the vehicle simulation, quaternions are useful in preventing singularities during the simulation. It is difficult to visualize how a four dimensional quaternion relates to changes in the real world. Euler angles are extracted from quaternions using

$$\begin{bmatrix} \phi \\ \theta \\ \psi \end{bmatrix} = \begin{bmatrix} \tan^{-1}\left(\frac{2(q_0q_1+q_2q_3)}{1-2\left(q_1^2+q_2^2\right)}\right) \\ \sin^{-1}\left(2\left(q_0q_2-q_3q_1\right)\right) \\ \tan^{-1}\left(\frac{2(q_0q_3+q_1q_2)}{1-2\left(q_2^2+q_3^2\right)}\right) \end{bmatrix} , \tag{3.34}$$

where $\tan^{-1}$ should be evaluated using a four-quadrant inverse tangent [32].

It is also convenient and useful to store position and velocity as quaternions for easy rotation. In this case, position and velocity are not stored as unit quaternions. They instead can be stored as quaternions with a zero real component and the vector component as the imaginary components of the quaternion. For example, the position vector in world

frame, $\boldsymbol{p}$, can be stored as a quaternion as

$$q_p = \begin{bmatrix} 0 \\ p_x \\ p_y \\ p_z \end{bmatrix} . \tag{3.35}$$

The position vector can then easily be rotated to inertial frame by

$$q'_p = \Phi \otimes q_p \otimes \Phi^{-1} = \begin{bmatrix} 0 \\ p'_x \\ p'_y \\ p'_z \end{bmatrix} \tag{3.36}$$

where $\boldsymbol{p}'$ represents the position vector in the inertial frame and Φ represents the orientation quaternion of the vehicle.

The orientation quaternion is updated based on the angular velocity vector. The direction of the angular velocity vector represents the axis of rotation, and the magnitude of the rotation is the the norm of the angular velocity vector multiplied by the size of the time step. The angular velocity vector is updated based on the moments calculated in the inertial frame.

An additional benefit of quaternions is that computing orientation with quaternions requires only 42 operations, while computing orientation with Euler angles requires 116 computations [33].

3.1.5 Atmospheric Modeling

Due to the significant change in conditions of the atmosphere between ground level and apogee, I needed a model of the atmosphere to accurately simulate a rocket flight. Two atmospheric models are used for separate purposes in this work. The first is a baseline standard atmosphere used primarily for optimization techniques, while the second is a model developed for more refined simulation and flight predictions that is used in Monte Carlo simulations.

3.1.5.1 US Standard Atmosphere (1976)

The US Standard Atmosphere (1976) model was developed by NASA, National Oceanic & Atmospheric Administration (NOAA), and the United States Air Force (USAF) to define the standard atmosphere as a function of altitude [34].

Drag is a function of air density, thrust of a rocket engine is a function of ambient pressure, and coefficient of drag is a function of the speed of sound. These properties can all be calculated using the US Standard Atmosphere (1976) model. NASA defined the US Standard Atmosphere (1976) based on a lapse rate of temperature. The lapse rate of temperature is different for different layers of atmosphere. Shown in Table 3.4 are the lapse rates used.

Table 3.4: Lapse rates in US Standard Atmosphere.

Altitude Range, Z (km)	Lapse Rate, L (K/m)
0-11,000	-0.0065
11,000-20,000	0.0
20,000-32,000	0.028
32,000-47,000	0.0
51,000-71,000	-0.0028
71,000-86,000	-0.002
86,000-91,000	0.0

The gravity at sea level, g_o, is defined as $9.807\,\mathrm{m/s^2}$. Gravity is a function of the radius of the Earth and the altitude of the vehicle,

$$g = g_o \left(\frac{R_{\text{earth}}}{R_{\text{earth}} + z} \right)^2 . \tag{3.37}$$

Based on the lapse rate and gravity as a function of altitude, the pressure of the atmosphere is calculated as

$$P = P_o \left(\frac{T_o}{T_o + L_o(Z - Z_o)} \right)^{\left(\frac{g_o \mathfrak{M}_o}{\Re L_o} \right)} \tag{3.38}$$

where $\mathfrak{M}$ is the molecular weight of to avoid confusion with M for Mach number. Additionally, $\Re$ is used to differentiate universal gas constant from specific gas constant, R.

The notation i_o is used to indicate that the variable i should be evaluated at the base of the layer of atmosphere, as specified by the altitude range in Table 3.4.

With both temperature and pressure known, an equation of state can be used to solve for the air density as a function of altitude,

$$\rho = \frac{P\mathfrak{M}}{\mathfrak{R}T} \,. \tag{3.39}$$

Lastly, the speed of sound can be calculated as

$$c = \sqrt{\gamma \mathfrak{R} T / \mathfrak{M}} \,, \tag{3.40}$$

where γ indicates the ratio of specific heats.

Evaluation of the atmosphere from a range of 0-100 km is shown in Figure 3.4. I differed from the US Standard Atmospheric model in the region above 91 km. In this region, the US Standard Atmosphere proposes a temperature model based on a segment of an ellipse instead of a constant lapse rate. Additionally, the atmospheric model at these altitudes is dependent on solar activity and the current state in the solar cycle [34]. For simplicity, I used a constant lapse rate of 0.0 K/m for the region above 91 km. The drag forces on the vehicle are less than 1.0 N at this altitude due to the near zero density of the atmosphere (air density at 91 km is 0.0002 % of sea level air density). The drag force is insignificant in comparison the the force due to gravity (on the order of 1000 N). This assumption would not be applicable to orbiting satellites where velocity is high and the atmospheric drag force is applied for a long duration, but is appropriate for a suborbital vehicle near the Kármán line.

3.1.5.2 Earth-GRAM 2016

While the US Standard Atmosphere (1976) provides a complete model of the atmosphere for determining baseline performance of a design, it notably lacks wind and gust models and does not incorporate uncertainty well. When performing Monte Carlo simulations, it is important to have an accurate representation of conditions the vehicle could actually experience during flight. I achieved this by using the NASAs Earth-GRAM 2016 software [36].

Earth-GRAM began development shortly after the release of the US Standard At-

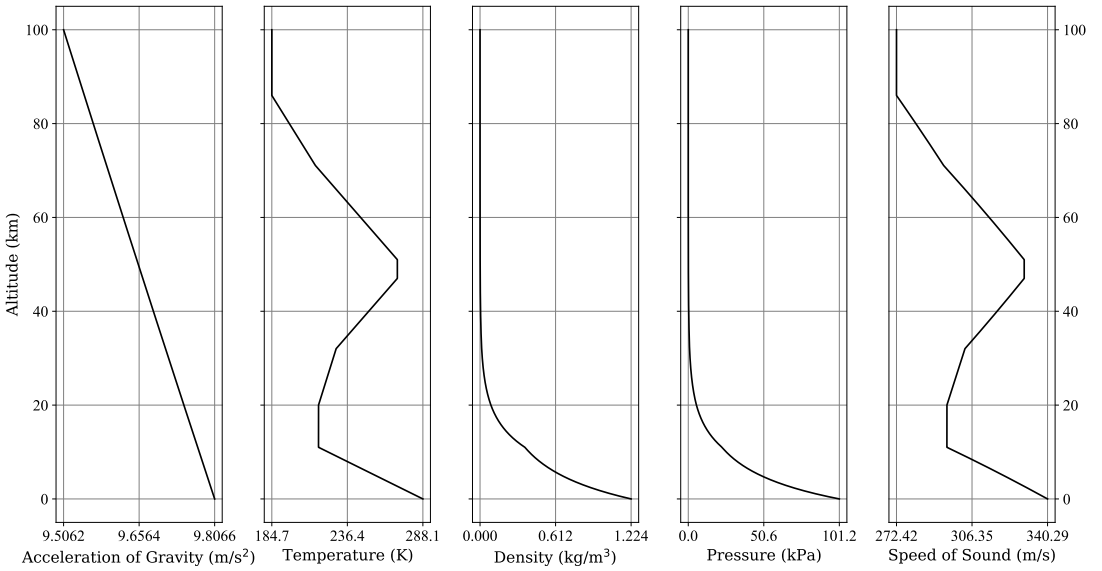

Figure 3.4: US Standard Atmosphere

mosphere (1976) as a Fortran program and has undergone numerous re-releases as atmospheric models are improved and refined [36–41]. The version used in this work was obtained from NASA as C++ version of the original Fortran code that is capable of conducting Monte Carlo simulations and reproducing close approximations of actual atmospheric environment conditions that a vehicle may experience. This model is a reputable standard which makes it a good choice for use in Monte Carlo simulations over implementation of alternative models. Similar models also exist for other planets in the solar system such as Mars and Venus that could be integrated for simulating flights on these planets if desired.

Another important note in the development and selection of wind modeling is limitations of the measurement devices used. For most typical wind measurement applications, an anemometer is used. Standard anemometers utilize propellers or cups that move as a function of wind speed. The propeller and cup style anemometers are subject to limitations in gust measurements, because the entire apparatus must be accelerated, slowing the response time. Higher temporal resolution wind measurement can be performed with techniques such as hot wire anemometery, though the availability of this data is not

as common. Therefore, the wind speeds measured have a loss of fidelity in temporal resolution. It is assumed that changes in wind speed occurring more rapidly than can be captured through a standard anemometer will not significantly effect the launch vehicle. This assumption has been successfully used previously in the design and development of launch vehicles [35]. Therefore, I assumed it is also appropriate for this work.

Figure 3.5 shows a sample of 100 random wind profiles produced by the Earth-GRAM software. The lines are plotted with some transparency, so darker regions are more common results. These are plotted for constant latitude and longitude of (32.21°, −106.38°) with an eight hour launch window starting at 8:00 AM local time (UTC-6) on December 1st, 2021.[1] Other parameters were left as defaults specified according to Leslie and Justus [41].

3.1.6 Engine Modeling

The NASA CEA software [42, 43] can be used for combustion modeling. This software is particularly useful for determining combustion properties of the engine. High-level parameters of engine performance, such as the chamber pressure, P_0; the mixture ratio, MR; and the expansion ratio, A_{ratio} are passed to the CEA software which calculates important engine parameters. For this work, I used a Python-wrapped version of the original CEA software written in Fortran [46].

Alternative software such as TDK produced by Sierra Engineering & Software or RPA produced by Rocket Propulsion Software+Engineering UG [44, 45] could have been used. Particularly, the TDK software accounts for kinetic losses in the nozzle according to the JANNAF standard. While the results from these software may have provided an improved theoretical engine model, the CEA software was chosen because it has already been adapted for use in Python and is readily available. Future work could include a more detailed comparison of the advantages of each of these software and integration with the current work.

Based on the output from the CEA software, I calculated the performance by assuming isentropic flow through the nozzle [7, 17]. Given a known nozzle area ratio A_{ratio}, the

[1] This corresponds to coordinates of a potential launch site at Spaceport America, NM and possible launch window of interest. These choices are arbitrary and should be specific to an actual launch.

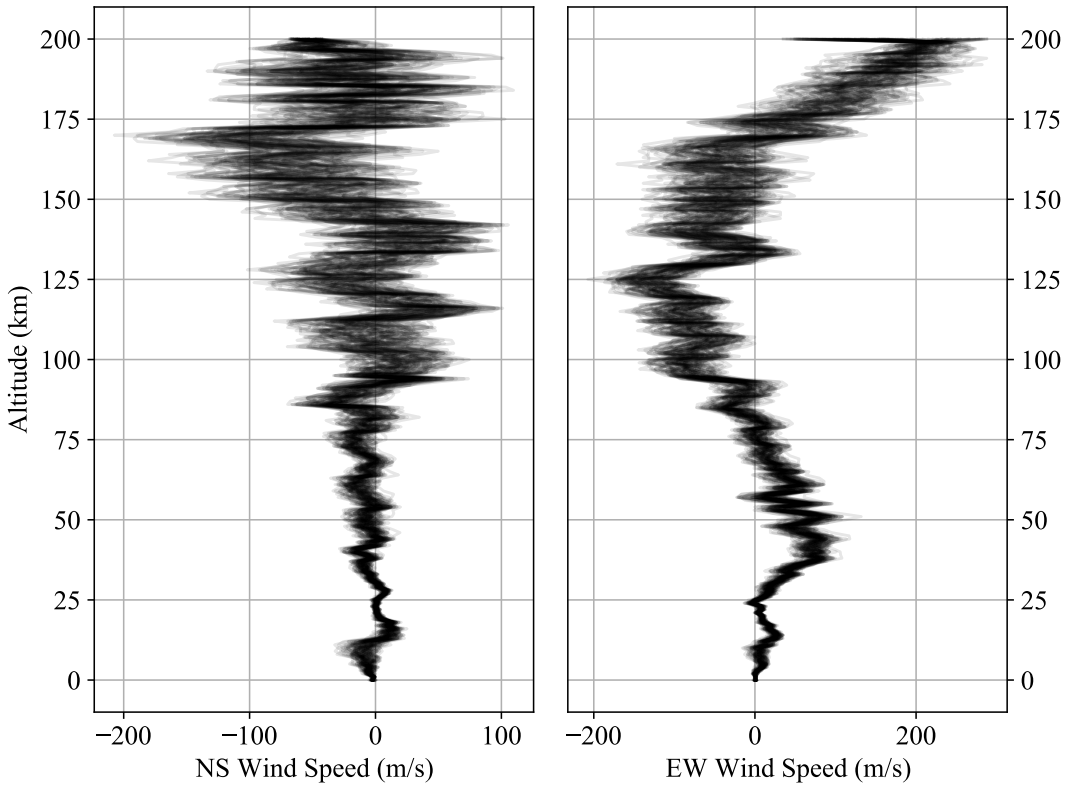

Figure 3.5: Monte Carlo results for 100 random wind profiles in North-South and East-West directions produced by Earth-GRAM.

exit Mach number is solved with

$$A_{\text{ratio}} = \frac{1}{M}\left(\frac{1+M^2(\gamma-1)/2}{(\gamma+1)/2}\right)^{\frac{\gamma+1}{2(\gamma-1)}}, \tag{3.41}$$

where M is the exit Mach number and γ is the ratio of specific heats from CEA. The exit temperature can be solved for as

$$T_{\text{exit}} = \frac{T_t}{1+M^2(\gamma-1)/2} \tag{3.42}$$

and exit velocity can be found with

$$v_{\text{exit}} = M\sqrt{\gamma R T_{\text{exit}}} \tag{3.43}$$

where R is the specific gas constant from CEA. The exit pressure is solved as

$$P_{\text{exit}} = P_t\left(\frac{1}{1+M^2(\gamma-1)/2}\right)^{\frac{\gamma}{\gamma-1}} \tag{3.44}$$

and the propellant mass flow rate as

$$\dot{m}_{\text{propellant}} = A_{\text{exit}} P_t M\left(1+\frac{M^2(\gamma-1)}{2}\right)^{\frac{-(\gamma+1)}{2(\gamma-1)}}\sqrt{\frac{\gamma}{RT_t}}. \tag{3.45}$$

A momentum efficiency is associated with non-axial motion of the exhaust gasses due to nozzle geometry. For a conical nozzle, Sutton and Biblarz [7] express this momentum efficiency as

$$\eta_m = \frac{1}{2}\left(1+\cos\alpha\right), \tag{3.46}$$

where α is the diverging half angle of the nozzle. Finally, I solved for the thrust with

$$F_{\text{thrust}} = \eta_k\left(\eta_m \dot{m}_{\text{propellant}} v_{\text{exit}} + (P_{\text{exit}} - P_\infty)A_{\text{exit}}\right), \tag{3.47}$$

where η_k is the efficiency of the nozzle in converting to kinetic energy and P_∞ is the external atmospheric pressure.

The engine performance is not always constant. During engine startup and shutdown

it goes through transient phases as the combustion chamber is pressurized. For the scope of this work, the transient phases are modeled only based on the tank pressures, assuming instantaneous pressurization of the combustion chamber. The duration of the startup and shutdown phases are very brief, so I assumed them to be negligible.

3.1.7 Tank Pressure Modeling

In a pressure-fed system, a high-pressure gas drives propellant into the combustion chamber. To do this in a controlled manner, the design uses a separate pressurant tank which has a pressure regulator between the pressurant and propellant tank, maintaining a nearly constant pressure throughout the engine burn. However, each of the tanks operate at different target pressures. To individually control the pressure in each tank, I used sonic venturis for mass flow regulation of the pressurant gas.

Operating Pressure: The target tank pressures are determined based on achieving the target Oxidizer : Fuel (O/F) ratio, propellant mass flow rate, and chamber pressure in the engine. The pressure losses through each component between the tanks and combustion chamber are summed to determine the required tank pressure. Equations (3.48)-(3.54) show all relevant calculations, where there are n bends in the pipe. The relation for Darcy friction factor in Equation (3.54) is only valid for Re $>$ 4000, which is true for both fuel and oxidizer lines in this work.

$$P_{\text{tank}} = P_{\text{chamber}} + P_{\text{losses}} \tag{3.48}$$

$$P_{\text{losses}} = \Delta P_{\text{injector}} + \Delta P_{\text{pipes}} + \Delta P_{\text{valves}} \tag{3.49}$$

$$\Delta P_{\text{valves}} = \frac{\dot{V}}{C_f} SG_{\text{propellant}} \tag{3.50}$$

$$\Delta P_{\text{pipes}} = \Delta P_{\text{minor}} + \Delta P_{\text{major}} \tag{3.51}$$

$$\Delta P_{\text{minor}} = \sum_{i=1}^{n} k_i \left(\frac{1}{2} \rho v^2 \right) \tag{3.52}$$

$$\Delta P_{\text{major}} = \frac{f L \rho v^2}{2D} \tag{3.53}$$

$$\frac{1}{\sqrt{f}} = -2 \log \left(\frac{\epsilon}{3.7D} + \frac{2.51}{\text{Re}\sqrt{f}} \right) \tag{3.54}$$

Pressurant Flow Modeling: To model the pressurant, I tracked three distinct gasses over time: gas within the pressurant tank, and gas within each propellant tank. In Equations (3.55)-(3.58), variables with the subscript *ullage* are within a propellant tank while variables with the subscript *pressurant* are within the pressurant tank. The ullage volume within the tank increases as propellant is expelled according to

$$\frac{dV_{\text{ullage}}}{dt} = \frac{\dot{m}_{\text{propellant}}}{\rho_{\text{propellant}}} \, . \tag{3.55}$$

I assumed the gas within both the propellant and the pressurant tanks is ideal and follows the ideal gas law,

$$PV = mRT \, . \tag{3.56}$$

Assuming reversible adiabatic expansion is applicable within the pressurant tank [7], the relation

$$P_{\text{pressurant}}^{1-\gamma} T_{\text{pressurant}}^{\gamma} = \text{constant} \tag{3.57}$$

can be used to find temperature of the gas within the pressurant tank as the pressure decreases. I solved for the temperature within the propellant tanks by assuming a quasi-equilibrium of the gas as it flows into the propellant tank from the pressurant tank. For a small time step, a small mass of pressurant Δm enters the propellant tank at temperature

$T_{\text{pressurant}}$. This comes to an equilibrium temperature T_{eq} with the mass of gas within the propellant tank, m_{ullage}, at an initial temperature T_{ullage}. For an adiabatic process, the equation

$$m_{\text{ullage}} c_V (T_{\text{eq}} - T_{\text{ullage}}) + \Delta m c_V (T_{\text{eq}} - T_{\text{pressurant}}) = 0 \tag{3.58}$$

can be used to solve for the average temperature within the propellant tank. This is used with Equation (3.56) to solve for the current propellant tank pressure. In reality during engine burn, the system is constantly changing and never comes fully into equilibrium. However, the average temperature found using Equation (3.58) is a better approximation than assuming the ullage temperature is constant or assuming it is equal to the pressurant tank temperature.

The design includes a pressure regulator, a pressurant valve, and two sonic venturis (one for each tank) to control the mass flow rate of pressurant gas to the propellant tanks. Although two independent pressure regulators and two independent valves could be used, sonic venturis are small and lightweight and can provide mass savings while allowing different mass flow rates of gas to each tank. I specified the operating pressure of the regulator at a fixed pressure that will cause flow through the venturis to the tanks to remain choked. As shown by Zucker and Biblarz [17], the mass flow rate of the pressurant gas through a choked sonic venturi is determined by

$$\dot{m}_{\text{pressurant}} = A_{\text{throat}} P_t \left(1 + \frac{\gamma - 1}{2}\right)^{\frac{-(\gamma+1)}{2(\gamma-1)}} \sqrt{\frac{\gamma}{R T_t}} \, . \tag{3.59}$$

Combustion Chamber Pressure Modeling: As described in Section 3.1.6, I use the CEA software to calculate combustion properties based on chamber pressure, mixture ratio, and the expansion ratio of the engine. In Section 3.1.6, I assume the combustion chamber pressure is known; this is not generally true. However, I can solve for the combustion chamber pressure if the propellant tank pressures based on the ullage pressurant mass, temperature, and ullage volume are known at a given time using Equations (3.55)-(3.59).

Given two arbitrary propellant tank pressures, I applied the conservation of mass to the system to solve for combustion chamber pressure. I guess an initial combustion chamber pressure, then apply the following procedure:

1. Given propellant tank pressures and combustion chamber pressure, calculate mass flow rate through each propellant line.

2. Use propellant mass flow rates to determine updated mixture ratio, MR. Use CEA and equations (3.41)-(3.45) to calculate updated mass flow rate through the engine.

3. If the sum of the mass flow rate through the propellant lines is not equal to the mass flow rate through the engine, adjust combustion chamber pressure and repeat from Step 1. I used a root-finding algorithm with a small allowable numerical error to perform the combustion chamber pressure adjustments efficiently.

3.1.8 Vehicle Aerodynamics Modeling

The Rogers modified Barrowman method [14] is primarily used in this work for calculating aerodynamic forces on the vehicle. Based on the Barrowman method [15], this analysis is performed by dividing the vehicle into a body and tail for separate analyses, followed by analysis of tail and body interference, and recombination for summary of aerodynamic performance. Rogers added onto this work by improving the normal force slope calculation and center of pressure calculation. These calculations are available as an output of the RASAero II software.

The Barrowman method provides calculation of: drag coefficient, C_D; center of pressure location, $\bar{X}_{C_P}$; normal force coefficient derivative, C_{N_α}; roll forcing moment coefficient derivative, C_{l_δ}; roll damping moment coefficient derivative, C_{l_p}; pitch moment coefficient, C_m; and pitch damping moment coefficient derivative, C_{m_q}. Each of these have different relations depending on if the vehicle is in the subsonic, transonic, or supersonic regime. In this section, I will present the major equations and processes to calculate each of these. The full derivation of each of the coefficients is extensive, so only the reader is referred to Barrowman [15] for more details. I attempt to use the same notation as Barrowman to wherever possible for easy reference.

The Barrowman method has several major assumptions that apply to all calculations: a near zero angle of attack, steady irrotational flow, the vehicle is a rigid body, and the nose tip is a sharp point [15].

In this section, the subscript 1 will indicate a value only applicable to a single fin, the subscript T will only apply to the tail section, the subscript B will only apply to the

body section, and no subscript will apply to the entire vehicle.

3.1.8.1 Tail Section

The vehicle is divided into a tail consisting only of the fin assembly with no airframe considered. I assume that the tail section consists of three (3) or four (4) trapezoidal fins, which represent the vast majority of slender finned vehicles. Not all relations presented in this section are applicable for tail sections with different numbers or shapes of fins.

A standard trapezoidal fin geometry is presented in Figure 3.6. The trapezoidal fin geometry can be fully defined with: root chord, c_r; tip chord, c_t; span S, sweep distance X_{sweep}. A few important parameters of the fin are the fin area

$$A_{\text{fin}} = \frac{c_t + c_r}{2} S \ , \tag{3.60}$$

the aspect ratio

$$AR = \frac{S^2}{A_{\text{fin}}} \ , \tag{3.61}$$

and the fin taper ratio

$$\lambda = \frac{c_t}{c_r} \ . \tag{3.62}$$

The leading edge sweep angle, Γ_L, the mid chord sweep angle, Γ_c, and the quarter chord sweep angle, $\Gamma_{1/4}$, are also useful quantities that can be easily derived from the geometry of the fin.

Subsonic Flow: A single fin is analyzed using thin airfoil theory. For a single fin, the normal force coefficient derivative can be expressed as

$$(C_{N_\alpha})_1 = \frac{2\pi AR \, (A_{\text{fin}}/A_{\text{ref}})}{2 + \sqrt{4 + \left(\frac{\beta AR}{\cos \Gamma_c}\right)^2}} \tag{3.63}$$

where α is the angle of attack and β is the Prandtl—Glauert correction $\sqrt{|M^2 - 1|}$.

The tail section consists of several fins placed at different dihedral angles. A relation based on the dihedral angle of the fin can be used to determine the normal force coefficient for the total number of fins. For three and four fin configurations, this simplifies based

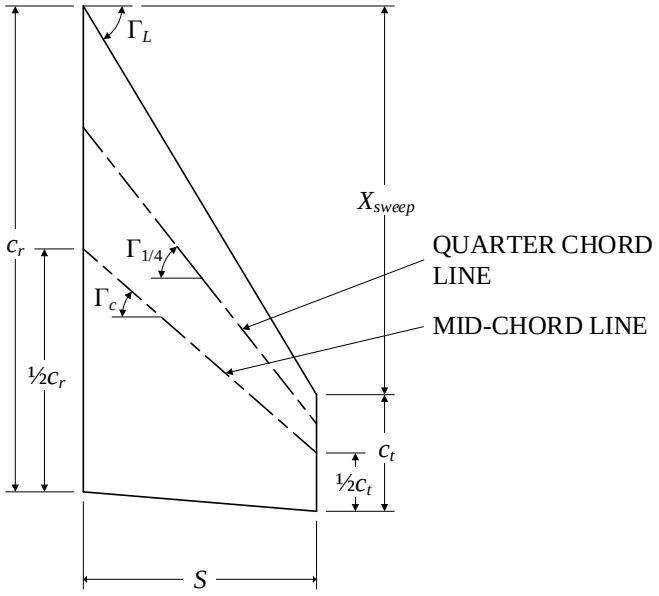

Figure 3.6: Notation for a standard trapezoidal fin geometry [15].

on the dihedral angles of the N fins as

$$(C_{N_\alpha})_T = \frac{N\pi AR\,(A_{\text{fin}}/A_{\text{ref}})}{2+\sqrt{4+\left(\frac{\beta AR}{\cos\Gamma_c}\right)^2}}\,. \tag{3.64}$$

For other fin configurations, Equation (3.64) does not apply and must be corrected based on the dihedral angle of each of the fins.

For a single fin, the center of pressure location has two components: longitudinally along the axis of the vehicle $\left(\bar{X}_{C_P}\right)_1$, and in the spanwise direction $\left(\bar{Y}_{C_P}\right)_1$. In the spanwise direction, this is determined as

$$Y_{MA} = \frac{S}{3}\left(\frac{c_r+2c_t}{c_r+c_t}\right) \tag{3.65}$$

$$\left(\bar{Y}_{C_P}\right)_1 = r_t + Y_{MA} \tag{3.66}$$

where r_t is the radius of the tail section to the base of the fin. For axisymmetric fins, the

spanwise center of pressure location sums to zero. The mean aerodynamic chord can be expressed as

$$c_{MA} = \frac{2}{3}\left(c_r + c_t - \frac{c_r c_t}{c_r + c_t}\right) . \tag{3.67}$$

The center of pressure in the longitudinal direction can be expressed as

$$\left(\bar{X}_{C_P}\right)_1 = l_T + Y_{MA}\tan\Gamma_L + \frac{1}{4}c_{MA} . \tag{3.68}$$

The location of the longitudinal center of pressure is the same for each fin, therefore $\left(\bar{X}_{C_P}\right)_T = \left(\bar{X}_{C_P}\right)_1$.

The roll coefficient derivative is

$$C_{l_\delta} = N\left(C_{N_\alpha}\right)_1 \frac{\bar{Y}_T}{L_{\text{ref}}} . \tag{3.69}$$

The roll damping moment coefficient derivative is

$$C_{l_p} = -\frac{N c_r S}{6L_{\text{ref}}^2}\left((1+3\lambda)S^2 + 4(1+2\lambda)Sr_t + 6(1+\lambda)r_t^2\right)\left(C_{N_\alpha}\right)_1 . \tag{3.70}$$

The drag force coefficient relations are from both empirical correlations and from theoretical predictions. In laminar flow over a flat plate the theoretical skin friction coefficient is

$$C_f = \frac{1.328}{\sqrt{\text{Re}}} \tag{3.71}$$

where Re is the Reynolds number [15]. In turbulent flow with $\text{Re} > 5\times10^5$, the relation

$$C_f = \frac{1}{(3.46\log\text{Re} - 5.6)^2} - \frac{1700}{\text{Re}} \tag{3.72}$$

is used. For rough surfaces, a critical Reynolds number is

$$\text{Re}_{cr} = 51\left(\frac{R_s}{L_r}\right)^{-1.039} . \tag{3.73}$$

For Reynolds numbers above Re_{cr}, I use the relation

$$C_f = 0.032\left(\frac{R_s}{L_r}\right)^{0.2} . \tag{3.74}$$

The skin friction coefficient is corrected for compressibility effects in laminar flow with

$$C_{f_c} = C_f(1 - 0.09M^2) \ , \tag{3.75}$$

and in turbulent flow with

$$C_{f_c} = C_f(1 - 0.12M^2) \ . \tag{3.76}$$

The skin friction coefficient is converted to a friction drag coefficient with

$$(C_{D_f})_T = 2NC_{f_c}\frac{A_{\text{fin}}}{A_{\text{ref}}} \ . \tag{3.77}$$

The pressure drag is calculated by assuming the fin has a leading edge with a radius r_L that can be approximated as a cylinder in crossflow with no base drag. Barrowman computes this based on the difference between total drag and base drag of a cylinder in different regimes of flight, the drag from a cylinder with no base drag is computed as

$$\Delta C_D = \begin{cases} (1 - M^2)^{-.417} - 1 & (M < 0.9) \\ 1 - 1.5(M - 0.9) & (0.9 \leq M \leq 1.0) \\ 1.214 - \frac{0.502}{M^2} + \frac{0.1095}{M^4} + \frac{0.0231}{M^6} & (M > 1.0) \end{cases} \tag{3.78}$$

which are applied in the equation

$$(C_{D_L})_T = 2N\left(\frac{Sr_L}{A_{\text{ref}}}\right)\cos^2\Gamma_L \Delta C_D \ . \tag{3.79}$$

Barrowman expresses the subsonic base drag of the fins as

$$(C_{D_B})_T = \frac{0.135N\left(\frac{A_{B_f}}{A_{\text{ref}}}\right)}{C_{f_B}^{1/3}(K - M^2\cos^2\Gamma_c)^{1/2}} \tag{3.80}$$

where A_{B_f} is the base area of a fin with thickness t, K is a constant to prevent the denominator from becoming undefined, and C_{f_B} in compressible flow is

$$C_{f_B} = 2C_{f_c}\left(\frac{c_r}{t}\right) \ . \tag{3.81}$$

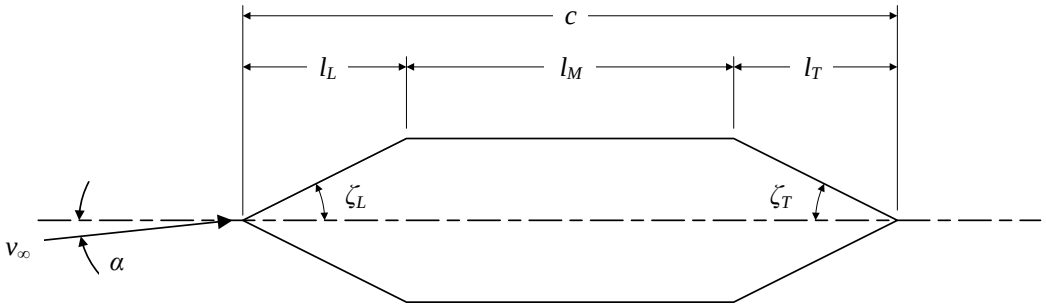

Figure 3.7: Notation for a standard double wedge fin [15].

Lastly, I found the drag due to the thickness of the fin as

$$(C_{D_T})_T = 4NC_{f_c}\left(\frac{t_r}{c_r}\right)\cos\Gamma_c + \frac{30\left(\frac{t_r}{c_r}\right)^4\cos^2\Gamma_c}{(K - M^2\cos^2\Gamma_c)^{3/2}}\ , \tag{3.82}$$

with K again a correction factor to prevent the denominator from being undefined, though it is different from the value of K used in Equation (3.80). See Barrowman [15] for the computation of the Prandtl factors K.

The drag from the entire section is computed by summing each of the components,

$$(C_D)_T = (C_{D_f})_T + (C_{D_L})_T + (C_{D_B})_T + (C_{D_T})_T \tag{3.83}$$

Supersonic Flow: Busemann's second-order expansion is applied to solve for the aerodynamic coefficients [15, 16]. This theory analyzes a single fin in supersonic flow by dividing it into spanwise strips and analyzing each as a two dimensional supersonic airfoil. This section focus only Busemann's second-order expansion for a double wedge airfoil shape, as shown in Figure 3.7, though other shapes can be analyzed similarly.

The double wedge shape in supersonic speeds has a weak oblique shock wave extending from the tip, followed by Prandtl—Meyer expansions around the corners. Busemann's theory is only applicable for an attached shock at the leading edge. With the leading edge of the double wedge at an angle from the velocity vector $\delta = \zeta_L \pm \alpha$, the equation

$$\frac{\tan\theta}{\tan(\theta-\delta)} = \frac{(\gamma+1)M_1^2\sin^2\theta}{(\gamma-1)M_1^2\sin^2\theta+2} \tag{3.84}$$

is solved for the angle of the oblique shock θ [17, 18]. For non-zero angles of attack, δ is different on the top and bottom panels. Equation (3.84) has two solutions, corresponding to a weak and a strong shock. For external flows, I am only interested in the weak shock solution [18]. The Mach number downstream of the shock can be calculated with the component of Mach number normal to the shock wave, M_{1_N}, as

$$M_{1_N} = M_1\sin\theta \tag{3.85}$$

$$M_{2_N}^2 = \frac{M_{1_N}^2+2/(\gamma+1)}{2\gamma/(\gamma-1)M_{1_N}^2-1}\ , \tag{3.86}$$

$$M_2 = \frac{M_{2_N}}{\sin(\theta-\delta)}\ . \tag{3.87}$$

Prandtl—Meyer expansion is based around the Prandtl—Meyer function

$$\nu = \left(\frac{\gamma+1}{\gamma-1}\right)^{1/2}\tan^{-1}\left(\frac{\gamma-1}{\gamma+1}(M^2-1)\right)^{1/2} - \tan^{-1}(M^2-1)^{1/2} \tag{3.88}$$

for a given flow [17]. For a flow to turn a small angle $\Delta\nu$, I use the relation

$$\nu_2 = \nu_1 + \Delta\nu \tag{3.89}$$

where a positive turning angle is a Prandtl—Meyer expansion and negative for compression.

A single strip of the double wedge fin has six panels, as shown by Figure 3.7. The Busemann second-order expanded shock theory is applied to find the coefficient of pressure for a single panel,

$$C_{P_i} = \frac{P_i - P_\infty}{1/2\rho v_\infty^2} = K_1\eta + K_2\eta^2 + K_3\eta^3 + B_3\eta^3\ , \tag{3.90}$$

where η is the angle of incidence with respect to the free stream flow and K_i, B_i are

constants defined by

$$K_1 = \frac{2}{\beta} \tag{3.91}$$

$$K_2 = \frac{(\gamma+1)M^4 - 4\beta^2}{4\beta^4} \tag{3.92}$$

$$K_3 = \frac{(\gamma+1)M^8 - (2\gamma^2 - 7\gamma - 5)M^6 + 10(\gamma+1)M^4 - 12M^2 + 8\beta^2}{6\beta^7} \tag{3.93}$$

$$B_3 = \frac{(\gamma+1)M^4\left((5-3\gamma) + 4(\gamma-3)M^2 + 8\right)}{48} \tag{3.94}$$

At supersonic speeds, the fin tip will have a Mach cone extending from it with an angle

$$\mu = \tan^{-1}\left(\frac{1}{\beta}\right) . \tag{3.95}$$

Barrowman assumes that the effects of the Mach cone extending from the fin tip half the value of the coefficient of pressure found in Equation (3.90). Ratios are set up based on the location the Mach cone intersects a strip to account for this effect, termed r_i. When no Mach cone effects are present, r_i is equal to zero. When the Mach cone completely envelopes a panel, r_i is one. See Barrowman [15] for a full description of all conditions of r_i when the Mach cone partially covers a panel.

The normal force of a panel can be expressed as

$$F'_{N_i} = C_{P_i} d_i \left(1 - \frac{r_i}{2}\right) \tag{3.96}$$

and the wave drag force as

$$F'_{DW_i} = C_{P_i} n_i \left(1 - \frac{r_i}{2}\right) , \tag{3.97}$$

where the prime is used to indicate that the force values are normalized by dynamic pressure and strip width Δy. The pitching and roll moments are

$$M'_{p_i} = F'_{N_i} \bar{X}_i , \tag{3.98}$$

$$M'_{r_i} = y F'_{N_i} \tag{3.99}$$

with the location of the center of pressure of a single strip equal to

$$\bar{X}_i = \frac{\frac{1}{2}\left(1 - r_i + \frac{r_i}{2} + \frac{X_P}{l_i}\left(2 - r_i\right)\right)}{1 - \frac{r_i}{2}} \ . \tag{3.100}$$

These are summed for each strip to determine the normal force, wave drag, pitching moment, and roll coefficients of a single fin as

$$\left(C_N\right)_1 = \frac{\Sigma F'_{N_i} \Delta y}{A_{\text{ref}}} \ , \tag{3.101}$$

$$\left(C_{DW}\right)_1 = \frac{\Sigma F'_{DW_i} \Delta y}{A_{\text{ref}}} \ , \tag{3.102}$$

$$\left(C_m\right)_1 = \frac{\Sigma M'_{p_i} \Delta y}{A_{\text{ref}} L_{\text{ref}}} \ , \tag{3.103}$$

$$\left(C_l\right)_1 = \frac{\Sigma M'_{r_i} \Delta y}{A_{\text{ref}} L_{\text{ref}}} \ . \tag{3.104}$$

The center of pressure location of a single fin is calculated as

$$\left(\bar{X}_{C_P}\right)_1 = \frac{\Sigma M'_{p_i}}{\Sigma F'_{N_i} \cos\alpha} \tag{3.105}$$

$$\left(\bar{Y}_{C_P}\right)_1 = \frac{\Sigma M'_{r_i}}{\Sigma F'_{N_i}} \ . \tag{3.106}$$

The roll damping moment coefficient derivative in supersonic flow is found by

$$C_{l_p} = 1000 N \left(C_l\right)_1 \tag{3.107}$$

The skin friction coefficient of the laminar region in supersonic flow is

$$C_{f_c} = \frac{C_f}{\left(1 + 0.045 M^2\right)^{1/4}} \tag{3.108}$$

and in turbulence is estimated by

$$C_{f_c} = \frac{C_f}{\left(1 + k M^2\right)^{0.58}} \tag{3.109}$$

where k depends on the wall temperature of the fin. I assumed no fin cooling. When there is no fin cooling, a value of $k = 0.15$ is used. For rough surfaces, the correction

$$C_{f_c} = \frac{C_f}{1 + 0.18M^2} \tag{3.110}$$

is used. The supersonic skin friction coefficient is used in Equation (3.77) to calculate $(C_{D_f})_T$. The value of $(C_{D_L})_T$ is computed using Equations (3.78) and (3.79). The base drag is computed as

$$(C_{D_B})_T = \frac{0.135N\left(\frac{A_{B_f}}{A_{\text{ref}}}\right)}{C_{f_B}^{1/3}(K - M^2\cos^2\Gamma_c)^{1/2}} \ . \tag{3.111}$$

Each of the drag components are summed to calculate drag from the tail

$$(C_D)_T = (C_{D_f})_T + (C_{D_L})_T + (C_{D_B})_T + (C_{DW})_T \ . \tag{3.112}$$

3.1.8.2 Body Section

The body of the vehicle is analyzed for three types of components: a nosecone, a cylindrical airframe, and a set of conical frustums that can be used to transition between airframe diameters.

Subsonic Flow: In subsonic flow for low angles of attack, Barrowman derives the equation

$$(C_{N_\alpha})_i = \frac{2}{A_{\text{ref}}}\left(A_{\text{aft}} - A_{\text{for}}\right), \tag{3.113}$$

for the ith body section, where A_{fore} and A_{aft} represent cross sectional areas of the foremost and aft-most ends of a component [15]. The cross sectional area does not change for a cylindrical component, therefore the normal force of these components are zero for small angles of attack. For a nosecone, the cross sectional area at the tip is zero and the resulting derivative of the normal force coefficient is

$$(C_{N_\alpha})_i = \frac{2A_{\text{BN}}}{A_{\text{ref}}} \tag{3.114}$$

where A_{BN} is the area of the base of the nosecone. By convention, this area is typically selected as the reference area, though not required. The resulting normal force coefficient of conical frustums with a smaller diameter at the aft end will be negative.

Barrowman also shows that the center of pressure location of a component is

$$\bar{X}_i = l_i - \frac{V_i}{A_i} \tag{3.115}$$

where l_i is the distance of the leading edge of the ith component from the nosecone tip.

The pitch moment coefficient in subsonic flow for a body component is derived by Barrowman as

$$(C_m)_i = \frac{2\alpha}{A_{\mathrm{ref}} L_{\mathrm{ref}}} \left(l_i A_{\mathrm{aft}} - V_i\right) \ . \tag{3.116}$$

Subsonic skin friction for the body is found the same as for the tail using Equations (3.71)-(3.76), except a correction must be applied because the body is not a flat plate. The relation

$$(C_{D_f})_B = \left(1 + \frac{0.5}{f_B}\right) C_{f_c} \frac{A_{W_B}}{A_{\mathrm{ref}}} \tag{3.117}$$

is used to correct for the body not being flat, where f_B is the fineness ratio [15].

The body pressure drag is expressed as

$$(C_{D_P})_B = \frac{6 A_{W_B} C_{f_c}}{f_B^3 A_r (K - m^2)^{.6}} \ , \tag{3.118}$$

where K is again a constant Prandtl factor, the calculation of K is provided by Barrowman [15]. The subsonic base drag is given by

$$(C_{D_B})_B = \frac{0.29 \left(\frac{A_B}{A_{\mathrm{ref}}}\right)}{\sqrt{C_{f_c} \left(\frac{A_W}{A_B}\right) (K - M^2)}} \ . \tag{3.119}$$

Finally the total drag from the body section in subsonic flow is the sum of the components,

$$(C_D)_B = (C_{D_f})_B + (C_{D_P})_B + (C_{D_B})_B \ . \tag{3.120}$$

Supersonic Flow: Similar to the analysis of the fins, determining the aerodynamic coefficients of the body in supersonic flow requires use of the panel method. A few

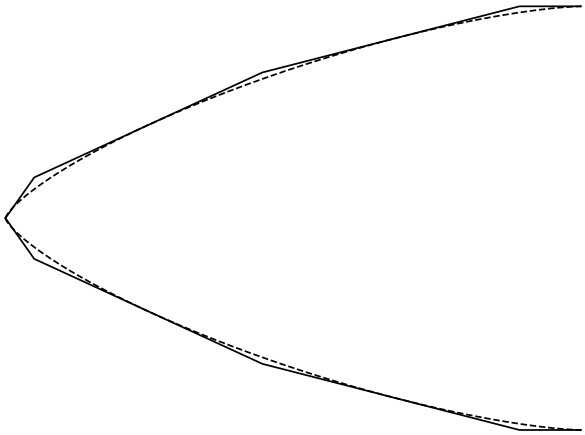

Figure 3.8: Fitting tangent panels to an arbitrary nosecone curve.

common nosecone shapes are: parabolic, conical, ogive, power series, and Haack series. The method for analysis of the nosecone is independent of the shape chosen, but an equation for the profile must be known. In this work, I analyzed a Von Kármán style nosecone, which is a specific type of Haack series nosecones.

To perform the analysis, the nosecone is represented by conical tip followed by a series of conical frustums. Each of the panels are drawn tangent to the nosecone. It is important that the panels be tangent to the nosecone because the angle of the conical shock attacked to the tip is a function of the angle of the nosecone vertex. Simply selecting points along the nosecone profile as the ends of each panel will cause the accuracy to be dependent on the size of the panel chosen [20]. An example of fitting the tangent panels to a nosecone profile is shown in Figure 3.8. Figure 3.8 is for visual purposes, many more panels are used in the actual analysis.

The conical nosecone tip is analyzed as a Taylor—Maccoll flow assuming a steady, irrotational flow. In spherical coordinates with θ as the angle from the body axis, the

Taylor—Maccoll equation is defined as

$$\begin{aligned} 0 =& \frac{\gamma-1}{2}\left(1-V_r^2-\left(\frac{dV_r}{d\theta}\right)^2\right)\left(2V_r+\frac{dV_r}{d\theta}\cot\theta+\frac{d^2V_r}{d\theta^2}\right)- \\ & -\frac{dV_r}{d\theta}\left(V_r\frac{dV_r}{d\theta}+\frac{dV_r}{d\theta}\frac{d^2V_r}{d\theta^2}\right) \end{aligned} \tag{3.121}$$

with velocities non-dimensionalized by the free stream Mach numbers. This is a second-order ordinary differential equation [18, 19]. After some algebra, the Taylor—Maccoll equation can be expressed as

$$\frac{d^2V_r}{d\theta^2} = -V_r + \frac{(a/a^+)^2(V_r+\frac{dV_r}{d\theta}\cot\theta)}{\left(\frac{dV_r}{d\theta}\right)^2-(a/a^*)^2} \tag{3.122}$$

with the quantity

$$(a/a^+)^2 = \frac{\gamma+1}{2}-\frac{\gamma-1}{2}\sqrt{V_r^2+\left(\frac{dV_r}{d\theta}\right)^2}\,. \tag{3.123}$$

Presented in the form of Equation (3.122), the Taylor—Maccoll equation can be solved numerically as a set of first-order differential equations using a Runge—Kutta fourth-order method. The quantity $\frac{dV_r}{d\theta}$ is equivalent to velocity normal to a ray extending from the nosecone vertex. At the surface of the nosecone, I know that the only velocity component can be in along the surface, therefore $\frac{dV_r}{d\theta}$ must be zero at the surface [18, 19].

To solve the Taylor—Maccoll equation, I followed the procedure outlined by Anderson [19]:

1. Guess an initial angle of the conical shock extending from the nosecone tip. Anderson suggests to use the angle

$$\delta_s^{(1)} = \delta_c + \frac{\mu}{2}\,, \tag{3.124}$$

where μ is the free stream Mach angle given by $\mu = \sin^{-1}\left(\frac{1}{M_1}\right)$ and δ_c is the half angle of the nosecone tip.

2. Calculate the parameters immediately downstream of an oblique shock wave at the

angle δ_s. To do this, I used the relations:

$$\frac{p_2}{p_1} = \frac{2}{\gamma+1}\left(M_1^2 \sin^2 \delta_s - \frac{\gamma-1}{2\gamma}\right) \tag{3.125}$$

$$\frac{\rho_1}{\rho_2} = \frac{\tan\psi}{\tan\delta_s} = \frac{2}{\gamma+1}\left(\frac{1}{M_1^2 \sin^2 \delta_s} + \frac{\gamma-1}{2}\right) \tag{3.126}$$

$$\frac{M_2^+}{M_1^+} = \frac{\sin\delta_s}{\sin\psi}\left(\frac{2}{(\gamma+1)M_1^2 \sin^2 \delta_s} + \frac{\gamma-1}{\gamma+1}\right) \tag{3.127}$$

$$M_1^+ = \sqrt{\frac{(\gamma+1)M_1^2}{2+(\gamma-1)M_1^2}} \tag{3.128}$$

$$V_r = M_2^+ \cos\psi \tag{3.129}$$

$$\frac{dV_r}{d\theta} = -M_2^+ \sin\psi \tag{3.130}$$

where the subscript 1 indicates a point just prior the oblique shock wave and 2 is just after the shock wave.

3. Starting with the initial conditions at the oblique shock, integrate by taking small steps $\Delta\theta$ until the value of $\frac{dV_r}{d\theta}$ equals zero. The angle where this occurs, $\delta_c^{(1)}$, represents the angle of cone that would cause the shock angle $\delta_s^{(1)}$.

4. It is likely that the angle $\delta_c^{(1)}$ is not equal to the actual cone angle δ_c. The shock angle should be updated according to

$$\delta_s^{(i)} = \delta_s^{(i-1)} + \left(\delta_c - \delta_c^{(i-1)}\right) \tag{3.131}$$

and the process repeated from step 2 until the value $\left(\delta_c - \delta_c^{(i-1)}\right)$ is within a sufficiently small tolerance.

In a supersonic conical flow, the properties along a ray line extending from the vertex of the cone are assumed to be constant [18, 19], therefore I can determine the flow properties just prior to the first corner on the nosecone approximated by tangent panels.

After analysis of the nosecone tip, each subsequent conical frustum is analyzed with a Prandtl—Meyer expansion fan at the corner. Here, I evaluate three critical points on each conical frustum: just upstream of the corner of the previous frustum, immediately downstream of the corner, and at the very end of the frustum. These points are denoted

in subscripts as 1-3 respectively.

With the ability to solve for this flow information at any point using Taylor—Maccoll for the tip and Prandtl—Meyer expansions for subsequent sections, Syvertson and Dennis [20] established a method for NACA to calculate the normal force coefficient. This method is followed by Barrowman [15]. To do this, the pressure gradient along the surface just after the corner is evaluated as

$$\left(\frac{\partial p}{\partial s}\right)_2 = \frac{B_2}{r}\left(\frac{\Omega_1}{\Omega_2}\sin\delta_2 - \sin\delta_2\right) + \frac{B_2}{B_1}\frac{\Omega_1}{\Omega_2}\left(\frac{\partial p}{\partial s}\right)_1 \tag{3.132}$$

with the quantities

$$B = \frac{\gamma p M^2}{2(M^2-1)}\ , \tag{3.133}$$

$$\Omega = \frac{1}{M}\left(\frac{1+\left(\frac{\gamma-1}{2}\right)M^2}{\frac{\gamma+1}{2}}\right)^{\frac{\gamma+1}{2(\gamma-1)}}\ . \tag{3.134}$$

The pressure at any point along the frustum is determined by

$$p = p_c - (p_c - p_2)e^{-\eta}\ , \tag{3.135}$$

$$\eta = \left(\frac{\partial p}{\partial s}\right)_2 \frac{x - x_2}{(p_c - p_2)\cos\delta_2}\ . \tag{3.136}$$

where p_c is the pressure on an equivalent cone tangent to the frustum at the same local radius. This relation is used to find the pressure gradient along the surface at point 3 with

$$\left(\frac{\partial p}{\partial s}\right)_3 = \frac{p_c - p_3}{p_c - p_2}\left(\frac{\partial p}{\partial s}\right)_2\ . \tag{3.137}$$

For subsequent frustums, Equation (3.137) will be equivalent to $\left(\frac{\partial p}{\partial s}\right)_1$.

Finally, the nondimensional loading on the frustum is evaluated as

$$\Lambda = (1 - e^{-\eta})\tan\delta\ \left.\frac{dC_N}{d\alpha}\right|_{tcx} + \frac{\lambda_2}{\lambda_1}e^{-\eta}\Lambda_1 \tag{3.138}$$

where Λ_1 is the nondimensional loading of the nosecone tip,

$$\Lambda_1 = \tan\delta_v\ \left.\frac{dC_N}{d\alpha}\right|_{tcv}\ . \tag{3.139}$$

Using the same notation as Sylverston and Dennis [20], the subscript tcx indicates the derivative is evaluated for the tangent cone and tcv is the value at the tangent cone vertex.

The nondimensional loading is used to find the normal force coefficient derivative of the body

$$(C_{N_\alpha})_B = \frac{2\pi}{A_B}\int_0^l \Lambda r dx \tag{3.140}$$

and the pitching moment coefficient

$$(C_m)_B = -2\pi\left(\frac{A_{\text{ref}}}{L_{\text{ref}}}\right)^{-1}\int_0^l \Lambda r x dx\ . \tag{3.141}$$

The pressure drag caused by the body is calculated as

$$(C_{D_P})_B = \frac{4}{\gamma M^2}\int_0^L\int_0^\pi P\sin\delta d\phi dx\ . \tag{3.142}$$

The supersonic friction drag, $(C_{D_f})_B$ can be found using the supersonic skin friction coefficient from Equations (3.108)-(3.108) with the body friction drag in Equation (3.117).

The supersonic base drag is a piecewise function with two components. Below a critical Mach number, M_{cr}, the base drag follows an exponential form. Above the critical Mach number, the base drag follows an inverse square form,

$$(C_{D_B})_B = \begin{cases} C^*_{D_B}(0.88 + 0.12e^{-3.58(M-1)}) & (1.0 \le M \le M_{\text{cr}}) \\ 0.7\frac{A_B}{A_{\text{ref}}}/M^2 & (M > M_{\text{cr}}) \end{cases} \tag{3.143}$$

where $C^*_{D_B}$ is a constant defined by Barrowman [15]. The critical Mach number is

$$M_{\text{cr}} = \frac{0.892}{\sqrt{C^*_{D_B}}}\ . \tag{3.144}$$

Finally, the total drag from the body section in supersonic flow is expressed as the sum of the components,

$$(C_D)_B = (C_{D_f})_B + (C_{D_P})_B + (C_{D_B})_B\ . \tag{3.145}$$

3.1.8.3 Tail-Body Interference

The tail in the presence of the body has a correction coefficient defined given by

$$K_{T(B)} = \frac{2}{\pi\left(1-1/\tau\right)^2}\left[\left(1+\frac{1}{\tau^4}\right)\left(\frac{1}{2}\tan^{-1}\frac{1}{2}\left(\tau-\frac{1}{\tau}\right)\right) - \right. \\ \left. -\frac{1}{\tau^2}\left(\left(\tau-\frac{1}{\tau}\right)+2\tan^{-1}\left(\frac{1}{\tau}\right)\right)\right] \tag{3.146}$$

with the center of pressure located at $\bar{X}_{T(B)} = \bar{X}_T$. This is used to compute the normal force coefficient $(C_{N_\alpha})_{T(B)} = (C_{N_\alpha})_T K_{T(B)}$.

The body in the presence of the tail has the correction coefficient

$$K_{B(T)} = \frac{\left(1-1/\tau^2\right)^2}{\left(1-1/\tau\right)^2} - K_{T(B)} \; . \tag{3.147}$$

This is used to compute the normal force coefficient $(C_{N_\alpha})_{B(T)} = (C_{N_\alpha})_T K_{B(T)}$. If the condition

$$\beta AR(1+\lambda)\left(\frac{1}{\beta m}+1\right) > 4 \tag{3.148}$$

where $m = \cot\Gamma_L$ is true, then a correction for the Mach cone extending from the fin tips is required. In this case, I used

$$(C_{N_\alpha})_{B(T)} = K'_{B(T)}\beta(1+\lambda)(\tau-1) \; , \tag{3.149}$$

where $K'_{B(T)}$ is defined by Pitts et al. in a NACA report [21].

To find the center of pressure location, I use the definition

$$\bar{X}_{B(T)} = \left(\frac{(C_{m_\alpha})_{B(T)}}{(C_{N_\alpha})_{B(T)}}\right) d \tag{3.150}$$

and calculate the pitching moment coefficient as

$$(C_{m_\alpha})_{B(T)} = \begin{cases} \frac{-8(m\beta)^{3/2}}{A_{\text{ref}}\pi\beta(\beta m+1)}\int_0^d dy\int_{\beta_y}^{c_r} x\cos^{-1}\left(\frac{x/\beta-\beta m y}{mx-y}\right)dx & m\beta<1 \\ \frac{-4m\beta}{A_{\text{ref}}\pi\beta\sqrt{\beta^2m^2-1}}\int_0^d dy\int_{\beta_y}^{c_r}\left(\frac{x\sqrt{x/\beta+y}}{\sqrt{mx-y}}\right)dx & m\beta>1 \end{cases} \; . \tag{3.151}$$

Pitts et al. [21] compute the integrals in Equation (3.151) for several common fin shapes.

The interference factor caused by fin cant can be computed as

$$k_{T(B)} = \frac{1}{\pi^2}\left[\frac{\pi^2(\tau+1)^2}{4\tau^2} + \frac{\pi(\tau^2+1)^2}{\tau^2(\tau-1)^2}\sin^{-1}\left(\frac{\tau^2-1}{\tau^2+1}\right) - \frac{2\pi(\tau+1)}{\tau(\tau-1)} + \right.$$
$$+ \frac{(\tau^2+1)^2}{\tau^2(\tau-1)^2}\left(\sin^{-1}\frac{\tau^2-1}{\tau^2+1}\right)^2 - \frac{4(\tau+1)}{\tau(\tau-1)}\sin^{-1}\frac{\tau^2-1}{\tau^2+1} + \tag{3.152}$$
$$\left. + \frac{8}{(\tau-1)^2}\ln\left(\frac{\tau^2+1}{2\tau}\right)\right] .$$

This is used to compute the roll moment coefficient as $(C_{l_\delta})_{T(B)} = (C_{l_\delta})_T \, k_{T(B)}$.

The interference factor for roll damping in the presence of the body is

$$k_{R(B)} = 1 + \frac{\frac{\tau-\lambda}{\tau} - \frac{1-\lambda}{\tau-1}\ln\tau}{\frac{(\tau+1)(\tau-1)}{2} - \frac{(1-\lambda)(\tau^3-1)}{3(\tau-1)}} . \tag{3.153}$$

This is used to compute the roll damping moment coefficient $C_{l_p} = \left(C_{l_p}\right)_T k_{R(B)}$.

With the vehicle traveling at a velocity V and a pitch rate q, a local velocity is produced across the tail from the rotation. This produces a damping moment,

$$C_{m_q} = (C_{N_\alpha})_{T(B)} \frac{q(\Delta x)^2}{V L_{\text{ref}}} \tag{3.154}$$

where Δx is the distance from $\bar{X}_{T(B)}$ to X_{C_G}.

3.1.8.4 Implementation

The normal force coefficient derivatives are combined to find the vehicle normal force coefficient,

$$C_{N_\alpha} = (C_{N_\alpha})_B + (C_{N_\alpha})_{T(B)} + (C_{N_\alpha})_{B(T)} . \tag{3.155}$$

The vehicle coefficient of drag is the sum of the components,

$$C_D = (C_D)_B + (C_D)_T . \tag{3.156}$$

Similarly, the pitch moment coefficients is

$$C_m = (C_m)_T + (C_m)_B \ . \tag{3.157}$$

The pitch damping moment coefficient, roll moment coefficient, and roll damping moment coefficient have only a single component and are the overall vehicle coefficients.

By applying a sum of the moments produced by each normal force component, the location of the vehicle center of pressure can be solved by

$$\bar{X}_{C_P} = \frac{\left(\bar{X}_{C_P}\right)_B (C_{N_\alpha})_B + \left(\bar{X}_{C_P}\right)_{T(B)} (C_{N_\alpha})_{T(B)} + \left(\bar{X}_{C_P}\right)_{B(T)} (C_{N_\alpha})_{B(T)}}{C_{N_\alpha}} \ . \tag{3.158}$$

The standard Barrowman method for subsonic and supersonic flow was implemented for aerodynamic coefficients. Barrowman makes no attempt to model the transonic region, where many of the coefficients become undefined as β goes to zero. Niskanen does not state the method used to correct for the transonic regime in OpenRocket [13]. By viewing the OpenRocket source code, it appears that a fifth-order polynomial is fit in the transonic regime; however, it is unclear where the coefficients in this polynomial come from. OpenRocket provides warnings in the transonic and supersonic regimes that the software may not be perfectly accurate, so I did not adopt this method for modeling the transonic regime. Rogers and Cooper [14] do not state the model used in RASAero II for the transonic regime, and the source code for RASAero II is proprietary. However, I analyzed the aerodynamic coefficients output from RASAero II and it appears to be a linear model through the transonic region for normal force coefficient derivative and coefficient of drag. Because the vehicle spends very little time in the transonic regime, I assumed this linear fit to be reasonably accurate in my implementation. I compared my implementation to the Rogers Modified Barrowman method [14] for reliability.

3.1.9 Center of Gravity Modeling

The unloaded, or dry center of gravity can easily be calculated given locations and masses of the vehicle using

$$\bar{X}_{C_G} = \frac{\Sigma m_i X_i}{\Sigma m_i}. \tag{3.159}$$

Shown in Equation (3.159) is how the center of gravity location can be computed for i components where the component masses are m_i and the component center of gravity is located a distance X_i from the base of the vehicle.

During the engine burn, the center of gravity will change based on the volume of fuel, oxidizer, and pressurant gas present in the tanks. This is clearly shown in Figure 3.9. With the amount of oxidizer, fuel, and pressurant gas in the tanks known at any given point, the overall vehicle center of gravity can be solved for at all points during flight using Equation (3.159) with the structures center of gravity and the center of gravity of both volumes of fluid. Note that while Figure 3.9 displays the center of gravity consistently moving up, there is a period of time at the start of flight where the center of gravity will move down. This is caused by a larger volume of oxidizer being displaced from the top of the oxidizer tank (due to O/F ratio).

This method of calculating center of gravity does not account for changes in center of gravity due to propellant slosh or vortices in the tank. I assumed the effects to be relatively small because they only affect the surface of the fluid.

3.1.10 Recovery Modeling

I implemented a simplistic recovery model in this work. The design uses a dual-deploy parachute system for recovery. A small drogue parachute is deployed at apogee via a carbon dioxide (CO_2) ejection system. The CO_2 ejection system uses a small canister of compressed CO_2 that is released into the parachute bay, increasing the pressure until several shear pins have been sheared and the two sections of the vehicle are separated. This ejection is considered to be instantaneous, as is inflation of the parachute.

The vehicle descends from apogee under drogue parachute at a safe and controlled descent speed until it reaches a lower altitude for deployment of the main parachute. The main parachute is similarly ejected using CO_2 and assumed to inflate instantaneously.

Each of the parachutes selected are sized to achieve a desired terminal velocity, where the force of gravity pulling the vehicle down is balanced by the force of drag pulling the vehicle up. Rewriting these forces to solve for terminal velocity provides

$$v_{\text{terminal}} = \sqrt{\frac{2mg}{C_D A_{\text{parachute}} \rho}} \,. \tag{3.160}$$

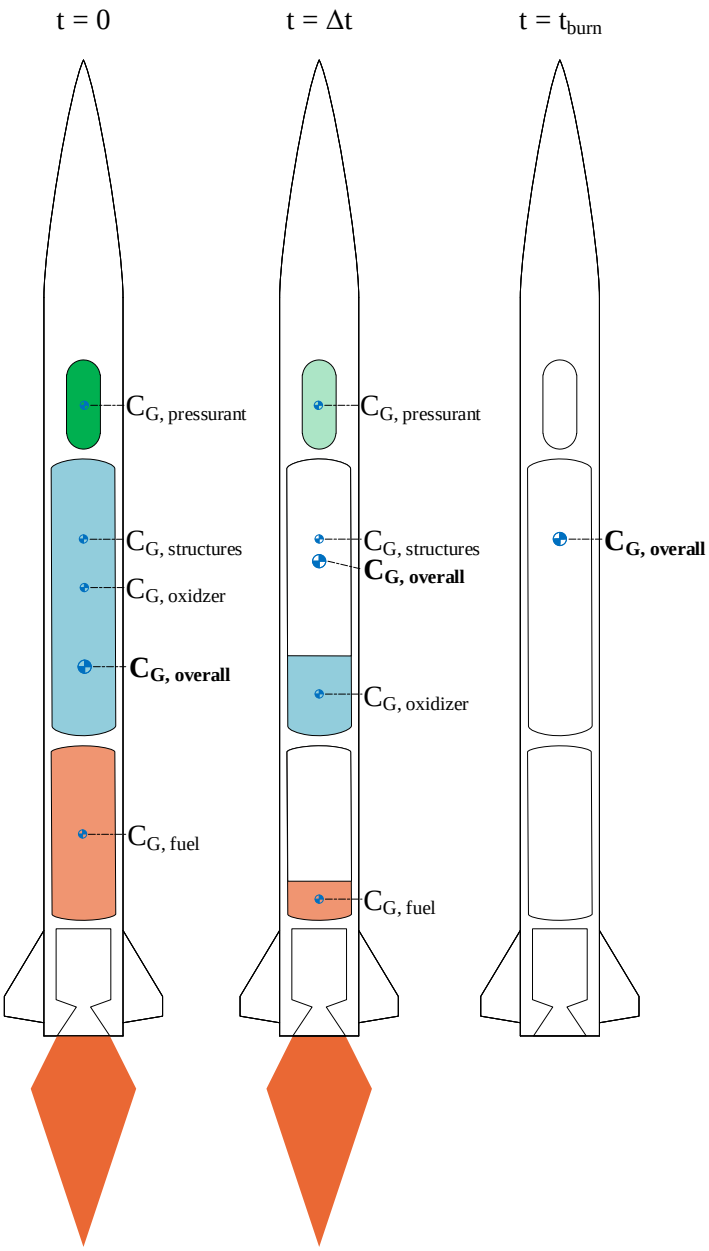

Figure 3.9: Changing center of gravity during engine burn.

The sizes of the parachutes is directly correlated to the mass of the parachute, the parachute bay size, and therefore the CO_2 ejection charge size. Therefore, proper sizing of the parachutes has a directly impacts the performance of the vehicle during ascent. Specifics of parachute sizing is outlined in [8, 10].

For a vehicle that leaves the atmosphere, a drogue parachute deployed at apogee will not begin to slow the descent of the vehicle until it reaches a denser atmosphere. At this point, the vehicle could be traveling at supersonic speeds. This will significantly change the drag coefficient of the parachute. However, parachute drag coefficients were considered constants across all Mach numbers within the scope of this work. This adds error in descent times and drift radii, however these are not primary objectives of the flight simulations performed and therefore I considered this acceptable. Additionally, the tethered sections of the vehicle are a multi-body dynamics problem [29]. Again, precise recovery modeling is not a primary objective. Therefore, multi-body dynamics were neglected in favor of a simple 3DoF model of the entire tethered system using only gravity and wind forces during recovery.

3.2 Vehicle Design

This section will discuss the design of a launch vehicle. Specifically, I focus on developing methods for designing the optimal vehicle to reliably complete a mission. Section 3.2.1 focuses on methods developed to optimize several high level vehicle parameters. However, the optimization performed requires numerous high level vehicle decisions to be made that are not considered in the optimization. These early design decisions mean that even if the parameters are able to be perfectly optimized, the resulting vehicle design is not necessarily the optimal design.

A complex system design, like a launch vehicle, can be modeled as a tree. Each layer of the tree represents a specific high level design decision to be made. These choices may have cascading effects on the resultant design. To complete a high level design, a selection must be made at each level of the tree until there are no deeper levels. Shown in Figure 3.10 is an example of how a designer might layout a tree for a launch vehicle design.

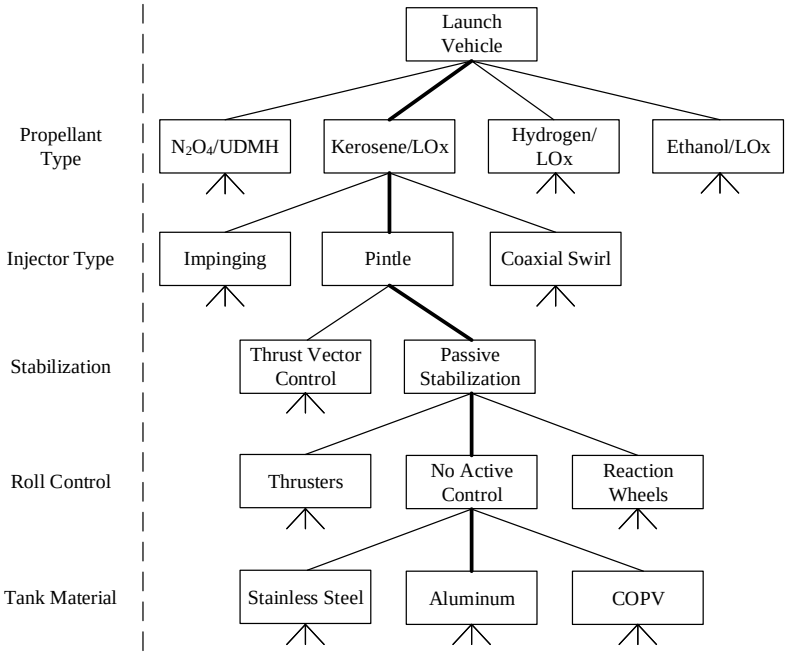

Figure 3.10: An example design tree for a launch vehicle with many distinct choices that could be made at each level. The tree is not fully expanded at any level.

The size of the tree shown in Figure 3.10 is for a discrete number of high level design decisions and the number of alternatives associated with each design. However, a designer could feasibly continue to add many additional layers of detail on each design decision. The number of states in the tree follows the relationship

$$\text{states} = b_{\text{eff}}^{N} \tag{3.161}$$

for an effective branching factor b_{eff} and number of layers in the tree N. This quickly becomes too large to feasibly explore the entire tree to find the best design. Often in the case of complex system design, component selection trees are indefinite and search methods must be used to explore them [51].

Even a finite tree can quickly become large enough that an exhaustive search of all possible combinations within the tree can become intractably large. Methods such as Dijkstra's algorithm [52] and A^* [53] can search a finite tree and guarantee an optimal

result without requiring a full search of the tree. However, both Dijkstra's algorithm and A^* require establishing a transition cost to go between each node in the graph. Additionally, A^* can reduce computation time further but requires establishing an admissible heuristic. If transition costs and admissible heuristics are established, the A^* method is guaranteed to find the most optimal solution within the tree.

Simple transition costs such as expected financial cost or weight added to the vehicle are possible for developing heuristic and transition costs. However, there are many trade-offs which are difficult to quantify by these simple metrics in complex system design. When deciding a propellant type for example, N_2O_4 and MMH may be less desirable due to the safety concerns associated with hypergolic propellant despite any performance benefits it may have over alternatives. A safety concern would not easily translate to a transition cost. Short et al. refers to this concept as *opaqueness* in the tree [51]. They demonstrated that genetic algorithms can be useful in searching opaque design decision trees.

Another method to compare alternatives where multiple characteristics may influence the decision is weighted Pugh charts [54]. Weighted Pugh charts were used in the context of this work for their simplicity and ease of implementation, though implementation of a tree search method could improve the overall design.

Lastly, some design decisions are not subject to further evaluation due to sunk cost. This is a poor choice for finding an optimal design. However, there is a time in the design process where the design is too mature and costly to change the design, despite possible benefits. In this case, settling for a sub-optimal solution may result in a more successful final product. An example of this could be propellant choice in the Oregon State liquid engine development program, where significant time and resources have gone into developing engine test infrastructure specific to a kerosene and LOx propellant combination. Switching propellants may feasibly provide benefit on paper, but would result in significant rework of the test infrastructure, loss of useful test data, and become a major schedule risk that could ultimately jeopardize the success of the project.

I considered implementing tree search optimization for high level vehicle decisions, but ultimately decided that it is outside of the scope of this work because of the complexities associated with developing clear transition and heuristic costs for the tree, accompanied with the sunk cost challenges for a program that is already underway. Instead of conducting tree search optimization, we used Pugh charts to compare trade-offs between

alternatives and detailed analyses were conducted where appropriate in [8–10]. Despite not implementing it in this work, I felt tree search warranted brief discussion for possible future work on search algorithms within the design tree. A few design choices are briefly described, though these choices are far from exhaustive and not covered in significant detail within the scope of this work.

Thrust Vector Control System: A thrust vector control system uses a pair of linear actuators to gimbal the engine and change the direction of thrust. This can be used to actively steer the vehicle during flight. Thrust vector control systems are applied to all commercial launch vehicle designs to actively guide the vehicle and ensure the payload is placed in the proper orbit. An additional benefit of thrust vector control systems is active stability control. Instead of relying on fins to lower the center of pressure and maintain stability, the thrust vector control system can actively maintain the orientation of the vehicle and eliminate the need for fins, which reduces drag and provides mass savings for the vehicle.

For the mission of focus in this work, thrust vector control is not needed because the vehicle does not have to fly on a specific trajectory to reach an orbit—it merely needs to reach a target apogee altitude. However, if thrust vector control is not used, the passively stable vehicle will rotate into the direction of relative wind. As the flight path angle deviates from vertical, apogee altitude is significantly reduced.

At the time of this writing, work on a thrust vector control system for the vehicle of focus in this report has begun at Oregon State University [47]. However, the thrust vector control system will require significant development testing, which could lag behind the rest of the development of the vehicle. Therefore, it is beneficial to develop a design which is functional without reliance on a thrust vector control system. If the thrust vector control system testing is completed in time, it can be integrated into both the vehicle design and the simulations presented in this work. I do not present details within this work, simply noting it presents an interesting branch in the design tree of Figure 3.10 that could warrant future study.

Reaction Control System: A reaction control system is used to react to system changes as they deviate from nominal trajectory. Reaction control systems as applied to the design of focus in this work is useful for controlling vehicle roll. Vehicle roll is

particularly important with liquid propellants, because liquid motion caused by tank rotation can allow gas bubbles to enter propellant lines and reach the engine during combustion. This can cause combustion instability with the possibility of damaging the engine, therefore it may need to be mitigated.

Two common forms of reaction control systems are reaction wheels and gas thruster systems (cold and hot). Reaction wheels operate by have a set of three wheels with discrete mass that are spun to a high speed prior to launch. If roll is experienced during flight, the reaction wheels can speed up or slow down which will change the roll rate of the vehicle due to conservation of angular momentum in the entire system (reaction wheels and overall vehicle). Alternatively, gas thruster systems work by storing some form of propellant which is directed out of a nozzle in the direction to be controlled. Cold gas thruster systems can be very simple, consisting only of compressed gas, a valve, an actuator, and a nozzle. Meanwhile, hot gas thruster systems can be more complicated as they rely on combustion of a fuel and oxidizer propellant to accelerate the fluid further, getting more roll control capability for the same mass of propellant than the cold gas thruster variant. Both cold and hot gas thrusters must be used in pairs on opposite sides of the vehicle to apply a moment.

At the time of writing this thesis, minimal work has been conducted by Oregon State University to design a reaction control system, but it is not required for mission success. Again, I do not present details of roll control within this work besides noting the branches in Figure 3.10 present significant opportunity for further study.

3.2.1 Design Optimization

Typically in a design process, the designer would always prefer to find the optimum design. In many cases, the optimum is related to the cost which will directly impact the profit of producing the design. There are often cases where finding the perfect optimum is not desired, because cost of analysis will begin to exceed the opportunity of cost reduction. Though many optimization algorithms have been developed, real world design problems are rarely well suited to apply these algorithms simply. According to Arora, up to 50 % of time spent optimizing a design may be in properly establishing the exact specifics of the design problem [57]. However, there are cases where determining the optimum design will provide significant improvement in the design that justifies performing the analysis.

Optimization of a launch vehicle is one such case; the opportunity for improvement can simplify designs, reduce probability of failure, and reduce costs.

A standard formulation of optimization problems is to identify the best set of design variables $\boldsymbol{x}$ to minimize an objective function $f(\boldsymbol{x})$ [57]. Maximization problems can easily be re-written as minimization problems, therefore I focus primarily on minimization. The objective function may additionally be subject to a set of constraints that impose restrictions on the feasible design space. Constraints are typically formulated in negative-null form. For the nth constraint imposed on the design problem, this is typically written as $g_n(\boldsymbol{x}) \leq 0$. In some cases, equality constraints may be expressed as $h_n(\boldsymbol{x}) = 0$. Equality constraints were unnecessary in the context of this work, so I do no treat them in significant detail.

For the design space, $\boldsymbol{x}$, each design variable x_i has a feasible set that may be evaluated. For many engineering problems, the feasible set of a design variable is $0 \leq x_i \leq \infty$ as negative values for length, time, mass, etc. do not make physical sense. However, this is not a strict requirement imposed on most optimization techniques. Additionally, some design variables may be discrete. Examples of discrete values in engineering design are material properties, which are fixed for a specific material but many options may be available.

The result of optimization methods are the optimizer of a set of design variables, $\boldsymbol{x}^*$, which results in the optima of the objective function such that $f(\boldsymbol{x}^*) \leq f(\boldsymbol{x})$ for all values of $\boldsymbol{x}$ in the feasible design space [48, 49, 57]. In practice, optimization methods typically fall into two categories: local and global optimization methods. In a convex problem, a local optima is guaranteed to be the global optima. For certain problems, convexity can be analytically determined. However, it is often difficult or impossible to determine the convexity of a problem [57].

In certain problems, a local optima may be satisfactory. An example of a problem where local optima may be satisfactory could be packing numerous odd shaped objects into the smallest area or volume, which can have many near-optimal solutions based on the large number of combinations of rotations and placements of each object. Sometimes, the difference between many of the local optima and the global optima may be only a few percent, therefore the additional complexity of implementation of global optimization methods may be determined to be unnecessary by the designer. Local minimization has a significant benefit that if an arbitrary point $\boldsymbol{x}$ does not meet the KKT conditions, it is

guaranteed to not be a local minima [48, 49].

Though local optima may be satisfactory in some cases, designers typically prefer to find the global optima. However, if the problem cannot be shown to be convex, it is difficult to guarantee a global optima. Unlike the KKT conditions for local optima, there is no general expression that can be written to guarantee an optimization method has found a global optima [57]. This makes determining convergence of global optimization methods difficult. The added complexities of global optimization methods may make the designer favor local optimization methods where global optimization methods are unnecessary. If the problem cannot be shown to be convex and the designer attempts to find the global optima, it would be good practice for a designer to compare the solution of a fully converged local optimization technique to the final result of the global optimization methods used.

An additional optimization challenge arises in trying to verify KKT conditions even for a local minima. The KKT conditions depend on gradient information of the objective function. In this case, the objective function is considered a black-box and I cannot express gradient information analytically. The gradient of the objective functions could be computed using finite differences, but I did not implement this in the current work.

One of the few approaches to guarantee a global optima is to conduct an exhaustive search on the entirety of the feasible design space. If the entire design space is searched, then the best result is guaranteed to be the global optima. For continuous design variables, there are infinite values that each design variable may theoretically have which makes an exhaustive search impossible. In engineering contexts, discretization of the design variables may still make sense. However, exhaustive methods quickly become intractably large based on the number of the design variables and the size of each set.

This section is intended to detail methods which can be used to search for a globally optimal high level design for a launch vehicle, subject to several important constraints. Determining the optimal high level vehicle design parameters is important because these will have cascading effects on all lower level designs as they continue to be refined and improved. This optimization assumes design performance can be approximated using the methods developed in Section 3.1. Note that the vehicle optimization is for performance subject to the models I previously selected, therefore my optimization results are critically dependent on the accuracy of previous model selection.

3.2.1.1 Objective Function

Before beginning an optimization, it is important to establish a clear objective. There are many potential objectives that could provide results of interest, such as minimizing cost of the vehicle or minimizing the dynamic pressure to reduces stresses within the vehicle. One objective is particularly interesting is minimization of propellant mass. By reducing the propellant mass, the tank sizes can be reduced, which lowers the overall mass of the vehicle, therefore requiring even less propellant. This phenomenon is sometimes referred to as the tyranny of the rocket equation. Minimizing propellant was selected as the optimization problem of interest, however the same methods could be applied to differing objective functions if desired. The unconstrained objective function can be expressed as

$$\min f(\boldsymbol{x}) = m_{\text{propellant}} \ . \tag{3.162}$$

3.2.1.2 Design Variables

Three high level design variables are selected as the focus of the optimization problem. These variables represent key parameters to the vehicle that have cascading effects on all subsystems within the vehicle, some of which could become their own optimization problems. Three design variables have been selected: engine burn time, propellant mass fraction, and vehicle outer diameter. The engine burn time is the time before a propellant is fully expended. For an engine with constant propellant mass flow rate, engine burn time can be expressed as

$$t_{\text{burn}} = \frac{m_{\text{propellant}}}{\dot{m}_{\text{propellant}}} \ . \tag{3.163}$$

The propellant mass fraction is the ratio of the mass of propellant to the mass of the entire vehicle,

$$\zeta = \frac{m_{\text{propellant}}}{m_{\text{total}}} = \frac{m_{\text{propellant}}}{m_{\text{structures}} + m_{\text{propellant}}} = \frac{m_{\text{propellant}}}{m_{\text{fixed}} + m_{\text{scaling}} + m_{\text{propellant}}} \ . \tag{3.164}$$

And lastly the outer diameter, OD, is simply the outer diameter of the airframe. I assumed the outer diameter of the propellant tanks is equal to the outer diameter of the

vehicle, where the tanks serve as a critical structural component in the airframe. Changes to this assumption would result in changes to the OD variable.

In accordance with common notation for optimization, the design variables are expressed as a vector,

$$\boldsymbol{x} = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} t_{\text{burn}} \\ \zeta \\ OD \end{bmatrix} . \tag{3.165}$$

3.2.1.3 Design Constraints

After establishing an objective, I established constraints for the vehicle. These constraints relate to requiring the mission objective to be met as well as performance parameters of the vehicle. I evaluated each of the constraints using a penalty method applied to the objective function. I used a square penalty method, where constraint violation is squared and added to the objective function. Alternatives such as infinite barrier penalty method are not ideal, due to limitations on optimization methods which rely on the design space being smooth and differentiable. Throughout this section, the *target* subscript is placing an upper or lower limit on constraint value, it does not correspond to a value that candidate designs are trying to meet.

One constraint imposed on the vehicle is related to engine performance. This optimization was completed assuming it is for an engine of known performance. There are strict limitations placed on engine development at Oregon State University that limit maximum thrust of the engines tested, therefore it is unnecessary and not useful to design a vehicle for an engine that cannot be developed and tested. To enforce this constraint, it is simply imposed through simulation parameters relating to the engine performance that are unchanging. In design studies where engine performance is unknown, engine parameters could be added as design variables with the same process repeated.

An important constraint is that the apogee altitude must meet or exceed the Kármán line (100 km). Due to the computational expense of evaluating mission reliability under varying wind with methods such as Monte Carlo simulations, I decided to adjust the apogee altitude constraint to allow margin for wind losses. The constraint target altitude was adjusted to be $h_{\text{target}} = 120\,\text{km}$. For consistency between samples, I used the US

Standard Atmosphere (1976). The value of h_{target} was made somewhat arbitrarily and could be adjusted based on design reliability results if necessary.

Additionally, the mass margin is the allowable mass the vehicle can increase by and still achieve the mission goals. From Equations (3.163) and (3.164), a maximum allowable structures mass (where structures mass is the mass of everything that is not propellant) can be defined as

$$m_{\text{structures,max}} = (\zeta - 1) m_{\text{propellant}} \ . \tag{3.166}$$

The mass margin is defined by

$$m_{\text{margin}} = m_{\text{structures,max}} - m_{\text{fixed}} - m_{\text{scaling}} \tag{3.167}$$

where m_{fixed} represents masses that do not change based on the design parameters, such as avionics or parachutes, and m_{scaling} represents masses that change based on the design parameters, such as the size of propellant tanks. The primary mass considered as a scaling mass are fuel and oxidizer tanks, though the airframe and nosecone do change in mass as well. Any design with a negative mass margin is infeasible, so the target mass margin is selected as $m_{\text{margin,target}} = 1\,\text{kg}$ which allows for slight violation of the constraints to still result in a feasible design. This value could be adjusted for preliminary analyses and assessed as a much larger target margin when other subsystem masses are only rough estimates.

Lastly, the aspect ratio (used interchangeably with fineness ratio in the context of this work) of the vehicle is

$$AR = \frac{L}{OD} \ . \tag{3.168}$$

For high values of aspect ratio, structural issues become highly relevant and can introduce failure modes. Particularly, the first mode of the natural frequency of the vehicle tends to lower as the aspect ratio is increased. To eliminate analysis and constraints on the natural frequency, an aspect ratio constraint is simply placed as a maximum of $AR_{\text{target}} = 20$.

Each of these can be re-written as a constraint in negative-null form. Though the design variables $\boldsymbol{x}$ do not directly appear in any constraints, each of these constraints are explicitly evaluated as a function of the design variables and become an output to the

flight simulation. The apogee altitude reached must be greater than the target apogee, the mass margin must be positive, and the aspect ratio must not be too large. In negative null form, these constraints are summarized as

$$g_1(\boldsymbol{x}) : h_{\text{target}} - h_{\text{apogee}} \leq 0 \tag{3.169}$$

$$g_2(\boldsymbol{x}) : m_{\text{margin,target}} - m_{\text{margin}} \leq 0 \tag{3.170}$$

$$g_3(\boldsymbol{x}) : AR - AR_{\text{target}} \leq 0 \tag{3.171}$$

To enforce these constraints, the simple objective function described in Equation (3.162) is adjusted as a square penalty method. Therefore, Equation (3.162) is re-written as

$$\begin{aligned} \min f(\boldsymbol{x}) = & m_{\text{propellant}} + \max(0, h_{\text{target}} - h_{\text{apogee}})^2 + \\ & + \max(0, m_{\text{margin}} - m_{\text{margin,target}})^2 + \max(0, AR - AR_{\text{target}})^2 \ . \end{aligned} \tag{3.172}$$

When no constraints are violated, all values of the constraint values are negative and Equation (3.172) simplifies to Equation (3.162).

3.2.1.4 Problem Bounds

Before testing any optimization algorithms, I checked to see that the problem is well bounded. If the problem is not bounded, the design variables will go to zero or infinity, neither is physically meaningful in this case. To do this, I must be able to show that each design variable must have some minimum non-zero value or some maximum non-infinity value.

The apogee altitude is enforced as a constraint which represents a change in potential energy from launch to the apogee altitude based on the mass of the vehicle. This change in energy must come from the stored chemical potential energy in the propellant,

$$\Delta E_{\text{chemical}} = \Delta E_{\text{potential}} + E_{\text{losses}} \tag{3.173}$$

where E_{losses} is the energy loss due to phenomena such as friction, non-vertical flight, angular momentum, and combustion efficiency. The stored chemical potential energy

monotonically increases as the propellant mass increases. Therefore, the stored chemical potential energy must be minimized to minimize the propellant mass, but the required change in potential energy to reach the target altitude imposes a strict minimum on the stored chemical potential energy. Therefore, I have imposed a strict minimum on burn time for a constant mass flow rate based on Equation (3.163).

The propellant mass fraction shown in Equation (3.164) varies between zero and one. There is some non-zero minimum propellant mass to achieve the mission based on change in chemical potential energy. Additionally, there are structures masses from the weight of both scaling and fixed components. Therefore, the propellant mass fraction is bounded between zero and one, but can never reach either zero or one. Referencing Equation (3.164), it is apparent that the propellant mass fraction monotonically decreases as the objective function decreases. Therefore, the global optima will minimize mass fraction.

The last design variable to consider is outer diameter of the vehicle. This is less straightforward to determine if it is bounded. By considering the mass of a thin-wall cylindrical tank[2] increases proportional to the diameter of the tank,

$$m_{\text{cylinder}} = \pi D L t_{\text{wall}} \rho_{\text{material}}. \tag{3.174}$$

However, the internal volume of a cylinder increases with the square of the diameter of the tank,

$$V_{\text{cylinder}} = \pi \left(\frac{D}{2}\right)^2 L. \tag{3.175}$$

Therefore, increasing tank diameter will reduce the tank mass for the same volume of propellant because the tank length is able to be reduced. Figure 3.11 is the result of varying tank diameter for 112 kg of propellant mass with 0.19 in. thick tank walls (arbitrary choices). In Figure 3.11, end cap masses are included that are not represented in Equations (3.174) and (3.175). By increasing the diameter of the tanks, $\Delta E_{\text{potential}}$ in Equation (3.173) is able to be reduced which means less propellant required for the

[2]Equations are presented for a cylindrical tank with no end caps for simplicity in presenting problem bounds. The actual optimization was performed assuming hemispherical end caps on each tank with the same thickness as the tank wall. Hemispherical end caps are a reasonable approximation of the flight vehicle tank design.

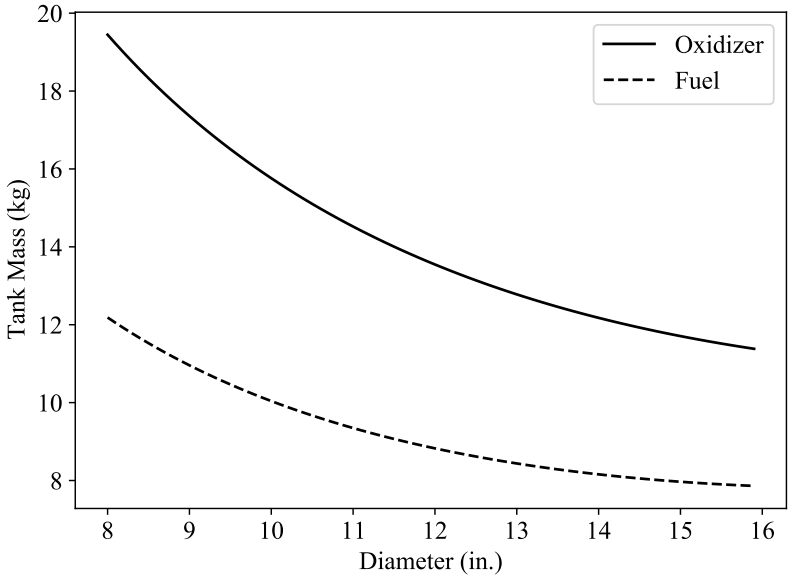

Figure 3.11: Tank masses with varying diameter.

same losses.

Vehicle drag also increases with the square of diameter, therefore reducing tank diameter will minimize drag losses. Vehicle drag increases energy losses and is described by

$$F_D = \frac{1}{2} C_D \pi \left(\frac{D}{2}\right)^2 \rho v^2. \tag{3.176}$$

Therefore, reducing tank diameter will minimize the E_{losses} in Equation (3.173) which would reduce the propellant required for the same change in potential energy. These two competing effects from varying outer diameter bound the variable and prevent it from ever going to zero or infinity.

Each of the design variables have shown to be bounded either through monotonicity or through competing effects on the objective function. Therefore, I can conclude before proceeding into the optimization that the problem is well bounded.

3.2.1.5 Optimization Method Selection

Each optimization problem can have a method that is most appropriate for the problem. Selection of the most appropriate optimization method could result not only in significant reduction in computational expense, but also improved optima. My first optimization method selection criteria is to only use methods which do not rely on gradient information. The objective function is treated as a black-box and no gradient information is known. The gradient could be approximated using finite differences, however this becomes computationally expensive by requiring many neighboring objective function evaluations. Instead, optimization methods such as direct search methods are favorable.

As with most optimization problems, I am interested in finding the global optima. However, because performance of each design is tested through running an entire flight simulation, it is computationally expensive to get a single function evaluation and gradient information about the objective function is unknown. I also do not know if the objective function is multi-modal or convex, a convex problem would greatly simplify the optimization and narrow the optimization method selection. These characteristics limit the scope of applicable optimization methods primarily to stochastic optimization methods and methods which optimize surrogate models.

I selected four optimization methods that met these criteria for comparison: Random Hill Climb (RHC), simulated annealing, particle swarm optimization, and Bayesian optimization.

Simplifications & Assumptions: To identify the best optimization method for the problem, I used a simplified version of the flight simulation that reduced computation time of a single function evaluation from approximately 30-40 seconds down to approximately 1-2 seconds. These simplifications only apply to the method selection of this work. Because of these simplifications, the optimization method selection is only useful in identifying the optimization method for the context of this problem. Some of the most notable simplifications included:

- Vertical flight only
- Constant thrust
- Constant drag coefficient

Table 3.5: Initial vehicle design parameters.

Parameter	**Symbol**	**Value**
Propellant Mass Fraction	ζ	63 %
Outer Diameter	OD	12 in.
Burn Time	t_{burn}	35 s
Propellant mass	$m_{\text{propellant}}$	112 kg
Apogee Altitude	h_{apogee}	136.2 km
Aspect Ratio	AR	18.9
Mass Margin	m_{margin}	3.3 kg

Another assumption applicable to both the optimization method selection and the optimization results is that each of the models selected closely match the actual performance and environment conditions the vehicle will undergo during flight. Therefore it is important to compare models to test data whenever possible to verify they accurately reflect reality. Each of the models described in Section 3.1 are subject to assumptions of their own, which cascade to the optimization performed.

Initial Design: During development and testing of the flight simulation, an initial design was found which had good performance and did not violate any constraints. I found this design through trial and error, so it is likely there are designs with lower objective functions. It is useful to have an initial feasible design for some optimization methods tested, this design is used as the starting point for all optimization techniques which require an initial starting point. The initial design parameters and performance are summarized in Table 3.5.

Random Hill Climb: The RHC method starts at a specific point and attempts to transition to a random neighbor to find an improved point in the design space [55]. If the neighbor point is an improvement on the original point, the algorithm accepts the transition. If the neighbor point is worse, the algorithm rejects the transition and stays in the same space. This method is subject to finding only local optima and to getting stuck in flat regions of the design space.

This method involves establishing discrete transition sizes which can be considered. In testing RHC, I used a total of four transition sizes and applied to each design variable.

An initial starting point is given, then a random number is generated, which corresponds to a specific transition that represents the new search direction, $\boldsymbol{d}$. The objective function of this new point is evaluated, if it is improved from the current point then the algorithm moves to this point. This process is repeated many times, which results in movement towards a local optima.

Because transition sizes are independent of the design variable, it is important to have design variables that are approximately on the same order of magnitude. The design variables are normalized by converting engine burn time to minutes and vehicle outer diameter to meters, which places all design variables in the range of zero to one for any reasonable design alternative. The transition sizes were then established as: [0.05, 0.01, 0.001, 0.0001] with search directions of positive or negative in each design variable. This totals 24 possible transitions that RHC could evaluate from any specific point, stored in a vector $\boldsymbol{\tau}$. If RHC fails to find an improved neighbor after 24 failed evaluations, I exhaustively search each of the neighboring points and a transition is made if any improved neighbors are found. Otherwise, I assume it has reached the optima and returns the design variables and the objective function.

A convergence criteria for RHC is the maximum number of function evaluations. Another criteria is to perform an exhaustive search of all neighbors after a set number of successive failed evaluation in a row. If the exhaustive search finds an improved neighbor, the transition is performed and the algorithm continues. Shown in Algorithm 3.2 is sample pseudo-code for RHC.

I first tested RHC from the initial design point in Table 3.5 and found an improved optima with relatively few function evaluations and short computation time. Several more random start points were generated and tested which found different solutions, this demonstrated that the design space was not convex. Table 3.6 displays the optima found by each RHC test.

The benefits of RHC are that it is a very simple algorithm which is capable of quickly finding a local optima. However, RHC has no ability to find a global optima unless it is able to start nearby. Testing several points demonstrated that the design space is multi-modal when they converged to different solutions. A significant weakness of RHC is that I do not know how many random points should be tested to thoroughly explore the space, and each additional point increases computation time and function evaluations.

Algorithm 3.2: Random Hill Climb

Input: $\boldsymbol{x}_0$
Result: $\boldsymbol{x}^*$, $f(\boldsymbol{x})^*$ $k = 0$
$\boldsymbol{x} = \boldsymbol{x}_0$
$f = f(\boldsymbol{x})$
$\boldsymbol{\tau} = \{\boldsymbol{\tau}_0, \boldsymbol{\tau}_1, ..., \boldsymbol{\tau}_n\}$
while not converged **do**
    $\boldsymbol{d} = \text{rand}(\boldsymbol{\tau})$
    $\boldsymbol{x}_{\text{new}} = \boldsymbol{x} + \boldsymbol{d}$
    $f_{\text{new}} = f(\boldsymbol{x}_{\text{new}})$
    $k = k + 1$
    if $f_{\text{new}} < f$ **then**
        $\boldsymbol{x} = \boldsymbol{x}_{\text{new}}$
        $f = f_{\text{new}}$
    check for convergence
$\boldsymbol{x}^* = \boldsymbol{x}$
$f(\boldsymbol{x})^* = f$

Table 3.6: Random hill climb performance with different starting points.

Optimization Starting Point	$f(\boldsymbol{x})$	$m_{\text{propellant}}$ (kg)	ζ	t_{burn} (s)	OD (in.)	Function Evaluations	Time (min)
Initial Design	106.0	106.0	0.630	33.13	11.61	122	2.52
Random Start Point 1	**103.3**	**103.3**	**0.610**	**32.28**	**10.68**	**164**	**2.96**
Random Start Point 2	129.1	128.6	0.674	40.20	15.25	162	3.30
Random Start Point 3	118.6	118.6	0.653	37.06	13.55	177	2.86

Simulated Annealing: The simulated annealing algorithm is analogous to annealing and crystal formation in macroscopic materials [56, 57]. At temperatures far above the freezing point, the materials are typically in a high energy state such as austenite. In this state, the atoms are more readily able to move about the material. The materials are then cooled according to a precise cooling schedule. As the materials are slowly cooled, the probability of each atom to remain in a high energy state begins to decrease and crystals are formed. If the materials are cooled slowly enough, the atoms are more readily able to move throughout the material to form large crystals while they are still at a higher energy state. This lowers the final energy state after the material has cooled. Rapid cooling of the material prevents this movement and creates many smaller crystals with a higher final energy state of the cooled material.

For large numbers of atoms, the probability of a transition to a different energy state in the material can be approximated according to the Metropolis criteria,

$$Pr = e^{\frac{-\Delta E}{k_B T}} \ , \tag{3.177}$$

where ΔE is the increased energy above the lowest energy state, k_B is Boltzmann's constant, and T is the temperature of the material [56]. As the temperature decreases, the probability of all of the atoms remaining at a high energy state decreases.

Kirkpatrick et al. [56] developed a method to apply the Metropolis criteria in a stochastic optimization algorithm. The algorithm has an effective temperature T_{eff}. A random transition within the design space is generated and the objective function is evaluated. If the objective function at the new point is less than the previous point, the transition is accepted. If the objective function is higher than the current evaluation (correlates to a higher energy state), the probability of accepting the point is computed as

$$Pr = e^{\frac{-\Delta f(\boldsymbol{x})}{T_{\text{eff}}}} \ , \tag{3.178}$$

where $\Delta f(\boldsymbol{x})$ is the difference in function evaluations, $f_{k+1}(\boldsymbol{x}) - f_k(\boldsymbol{x})$. A random number between 0 and 1 is generated, if this number is less than Pr then the transition is accepted. Otherwise, the transition is rejected and the algorithm remains at the previous point.

For high temperatures, the probability of accepting a transition to a higher objective function is nearly 100 %. This allows the algorithm to explore the design space. A cooling schedule is applied, slowly reducing the effective temperature [55]. As T_{eff} tends towards

zero, the probability of accepting a higher objective function becomes nearly 0 % and the algorithm exploits the best region it has previously found. In the limit where T_{eff} goes to 0, simulated annealing simplifies to RHC where only improved transitions are accepted.

The cooling schedule implemented will have significant effects on the performance of the algorithm. Kirkpatrick et al. used a simple cooling schedule, $T_{k+1} = T_k\mu$ where μ is a constant slightly below unity to provide slow cooling with each iteration [56]. Many alternative cooling schedules have been explored with varying success, but are outside the scope of this work. The cooling schedule is left as a function $T_{k+1} = \text{cooling}(T_k)$ for generality.

Several convergence criteria are possible for simulated annealing. These include maximum iterations, number of iterations without successful transition [57]. Shown in Algorithm 3.3 is pseudo-code for the simulated annealing algorithm.

While simulated annealing has beneficial aspects, it is highly dependent on both the starting temperature and the cooling schedule selected. It is very possible to cool too quickly and be stuck in a local optima or to cool too slowly and waste many function evaluations after the global optima has been found. These parameters are problem dependent and difficult to estimate beforehand.

The implementation of simulated annealing I used for this problem was from the *scipy.optimize* library [58]. I tested initially with the default parameters, though I stopped early because the algorithm spent significant time evaluating infeasible regions of the design space ($\zeta < 0$, $\zeta > 1$, $t_{\text{burn}} < 0$, or $OD < 0$). A maximum step size was added an re-run. Even with a maximum step size, the algorithm spent significant function evaluations in the infeasible region and there was no implemented method of establishing bounds on the design space. Despite some infeasible function evaluations, simulated annealing found a very good optima, but took over 2.8×10^4 function evaluations. Further work could have been done on implementing bounds, adjusting initial temperature, and alternative cooling schedules to improve the method for this problem. I choose not to pursue this because of the wasted function evaluations in the infeasible region.

Particle Swarm Optimization: Kennedy and Eberhart [59] developed particle swarm optimization as a method that is analogous to real life phenomena: the movement characteristics of many animals that travel in swarms, such as flocks of birds or schools of fish [57]. Flocks of birds can rapidly find food sources at random locations, despite

Algorithm 3.3: Simulated Annealing

Input: $\boldsymbol{x}_0$, T_0
Result: $\boldsymbol{x}^*$, $f(\boldsymbol{x})^*$ $k = 0$
$\boldsymbol{x} = \boldsymbol{x}_0$
$f = f(\boldsymbol{x})$
$T = T_0$
$\boldsymbol{\tau} = \{\boldsymbol{\tau}_0, \boldsymbol{\tau}_1, ..., \boldsymbol{\tau}_n\}$
while not converged **do**
    $\boldsymbol{d} = \text{rand}(\boldsymbol{\tau})$
    $\boldsymbol{x}_{\text{new}} = \boldsymbol{x} + \boldsymbol{d}$
    $f_{\text{new}} = f(\boldsymbol{x}_{\text{new}})$
    if $f_{\text{new}} < f$ **then**
        $\boldsymbol{x} = \boldsymbol{x}_{\text{new}}$
        $f = f_{\text{new}}$
    else
        $Pr = e^{(-\Delta f/T)}$
        $r = \text{rand}(0, 1)$
        if $r < Pr$ **then**
            $\boldsymbol{x} = \boldsymbol{x}_{\text{new}}$
            $f = f_{\text{new}}$
    $T = \text{cooling}(T)$
    $k = k + 1$
    check for convergence
$\boldsymbol{x}^* = \boldsymbol{x}$
$f(\boldsymbol{x})^* = f$

each bird only knowing about what it has seen recently. This is because the birds follow a simple set of general rules that dictate their movement that allow the flock to search large areas in a short amount of time to find the best sources of food. The rules that the flock follows have been optimized over eons through evolution, which limits the rules they follow to those that have been demonstrated to be beneficial.

To define the motion within the population, $\{\boldsymbol{x}^0, ..., \boldsymbol{x}^n\}$, Kennedy and Eberhart established several rules [59]. Each particle has some existing momentum, and cannot instantly change directions. Additionally, each particle has knowledge of the best position it has previously found, $\boldsymbol{x}^n_{\text{best}}$. Lastly, each particle has knowledge of the global best position any agent in the population has found, $\boldsymbol{x}_G$.

Each of the particles in the population has a velocity defined by a simple rule for the ith particle,

$$\boldsymbol{v}^i_{k+1} = \boldsymbol{v}^i_k + c_1 \text{rand}(0,1)\left(\boldsymbol{x}^i_{\text{best}} - \boldsymbol{x}^i_k\right) + c_2 \text{rand}(0,1)\left(\boldsymbol{x}_G - \boldsymbol{x}^i_k\right). \tag{3.179}$$

The $\boldsymbol{v}^i_k$ term gives the particle momentum based on its previous direction of travel. The other terms causes the particle to tend to exploit local optima and explore global optima. Kennedy and Eberhart originally used $c_1 = c_2 = 2$ in [59], however other values for these have been empirically shown to be useful as long as they sum to four [57].

In this method, each of the n particles are initially seeded randomly throughout the design space. Seeding can be user specified, random, or through methods such as Latin hypercube sampling. I used random sampling, but for generality I express seeding as the function sample(). Function evaluations are performed, then each of the particles tend towards the particle with the best function evaluation with some inherent randomness. As the particles move closer, they may find improved optima and shift the direction of the rest of the swarm. This allows for many regions of the design space to be explored which increases the likelihood of finding the global optima.

Two primary convergence criteria exist for the particle swarm algorithm: maximum iterations and all particles converging to the same point. If too low of a value is set for c_2, particles will favor local optima and the swarm will never converge to the same point. However, if too high of a value is set for c_2, particles will converge to a single local optima before exploring the design space enough to find the true global optima. Shown in Algorithm 3.4 is pseudo-code for the particle swarm optimization algorithm.

Algorithm 3.4: Particle Swarm Optimization

Input: n, $v_{\max}$, c_1, c_2
Result: $\boldsymbol{x}^*$, $f(\boldsymbol{x})^*$
$k = 0$
$x = \{\}$
$v = \{\}$
$f = \{\}$
for $i = 0$ **to** n **do**
    $\boldsymbol{x}_k^i = \text{sample}()$
    $\boldsymbol{v}_k^i = 0$
    $f_k^i = f(\boldsymbol{x}_k^i)$
    $f_{\text{best}}^i = f_k^i$
    $k = k + 1$
$f_G, \boldsymbol{x}_G = \min(\boldsymbol{f}_{\text{best}})$
while not converged **do**
    for $i = 0$ **to** n **do**
        $\boldsymbol{v}_k^i = \boldsymbol{v}_{k-1}^i + c_1 \text{rand}(0,1)\left(\boldsymbol{x}_{\text{best}}^i - \boldsymbol{x}_{k-1}^i\right) + c_2 \text{rand}(0,1)\left(\boldsymbol{x}_G - \boldsymbol{x}_{k-1}^i\right)$
        if $|\boldsymbol{v}_k^i| > v_{\max}$ **then**
            $\boldsymbol{v}_k^i = \boldsymbol{v}_k^i (v_{\max}/|\boldsymbol{v}_k^i|)$
        $\boldsymbol{x}_k^i = \boldsymbol{x}_{k-1}^i + \boldsymbol{v}_k^i$
        $f_k^i = f(\boldsymbol{x}_k^i)$
        if $f_k^i < f_{\text{best}}^i$ **then**
            $\boldsymbol{x}_{\text{best}}^i = x_k^i$
            $f_{\text{best}}^i = f_k^i$
    if $\min(f_k) < f_G$ **then**
        $f_G, \boldsymbol{x}_G = \min(f_k)$
    $k = k + 1$
    check for convergence
$\boldsymbol{x}^* = \boldsymbol{x}_G$
$f(\boldsymbol{x})^* = f_G$

Similar to the initial temperature and cooling schedule in simulated annealing, particle swarm optimization performance is very subject to the swarm size, swarm velocities, and other parameters such as c_1 and c_2. It is difficult to know what each of these parameters should be before starting an optimization problem.

The benefits of particle swarm are its ability to find global optima and search large portions of the design space. Additionally, function evaluations for each particle can be run in parallel to decrease computation time. Particle swarm was tested using the *pyswarms* library [60]. However, similar to simulated annealing, parameters such as swarm size and particle velocity are difficult to know at the start of a problem. This problem was tested with the default parameters for particle swarm and was able to find a very good optima, equivalent to simulated annealing, in less than 90 minutes and 4.6×10^3 function evaluations. Further work could be done to improve this through adjusting the parameters such as swarm size, maximum velocity, c_1, and c_2.

Bayesian Optimization: In this method, each function evaluation is used to build a surrogate model [61–63]. Prior to an evaluation, an estimate of the objective function is known as the *a priori*. Each function evaluation, the objective function is evaluated and the surrogate model updated with the new information to become the *a posteriori*. Using Gaussian process regression, error on the surrogate model can be calculated at any point. An acquisition function can be used to explore points that either have high uncertainty or a high probability of finding an improved point. Bayesian optimization is typically formulated to maximize an objective function, while the other methods explored are formulated to minimize a function. I accounted for this simply by maximizing the negative of the objective function expressed in Equation (3.172).

Three common acquisition functions are used in Bayesian optimization: Probability of Improvement (PI), Upper Confidence Bound (UCB), and Expected Improvement (EI). I used the expected improvement acquisition function provided by Hoffman et al. [63], which is defined by

$$EI(\boldsymbol{x}) = \begin{cases} (\mu(\boldsymbol{x}) - f(\boldsymbol{x}^+) - \zeta)\Phi(Z) + \sigma(\boldsymbol{x})\phi(Z) & \text{if } \sigma(\boldsymbol{x}) > 0 \\ 0 & \text{if } \sigma(\boldsymbol{x}) = 0 \end{cases}. \tag{3.180}$$

In Equation (3.180), $\boldsymbol{x}^+$ is the current best point with an objective function evaluation

$f(\boldsymbol{x}^+)$, ζ is a small constant value, Φ is the normalized cumulative distribution function, ϕ is the normalized probability distribution function, and Z is

$$Z = \begin{cases} \frac{\mu(\boldsymbol{x}) - f(\boldsymbol{x}^+) - \zeta}{\sigma(\boldsymbol{x})} & \text{if } \sigma(\boldsymbol{x}) > 0 \\ 0 & \text{if } \sigma(\boldsymbol{x}) = 0 \end{cases}. \tag{3.181}$$

Within each iteration of Bayesian optimization, the acquisition function must be maximized to find the next point to sample. Many optimization techniques could be implemented to do this, including some of the other techniques explored in this work such as random hill climb, simulated annealing, or particle swarm. However, the optimization of the acquisition function I used is to randomly sample the acquisition function 10,000 times and use L-BFGS-B[3] with a multi-start method with 10 seeds. After finding the 10 local optima with L-BFGS-B, the maximum of the random samples and L-BFGS-B seeds is considered to be the global optima of the acquisition function which determines the next objective function evaluation explored. These parameters are the default of *bayesian-optimization* library [64] and could be modified as necessary. For the scope of this work, I considered the defaults to be appropriate, however it is important to understand exactly how the acquisition function is maximized to get an understanding of the computational expense of Bayesian optimization. If evaluating the objective function is not far more computationally expensive than evaluating the acquisition function, Bayesian optimization will be a poor choice due to the computational expense of maximizing the acquisition function in every iteration of Bayesian optimization.

Bayesian optimization starts by sampling n points and evaluating the objective function at each one. The initial sampling could be random, latin hypercube sampling, or user specified. I used random sampling, but for generality I write the sampling as a function sample(). After n samples, a surrogate model is fit to the observations with a mean and standard deviation at each point. Often a Gaussian process regression is used, but for generality I just refer to fitting the surrogate model as fit(). An acquisition function of the surrogate model can be evaluated at any point, for generality I refer to the acquisition function used as acquisition(). An optimization is performed on the acquisition

[3]L-BFGS-B is an additional optimization algorithm that is not covered within the scope of this work. It requires evaluation of gradients of the objective function and is therefore not applicable to a general expensive black-box function. For the surrogate model built using Gaussian process regression, computation of the gradient of the is far less expensive and allows L-BFGS-B to be used.

function of the surrogate model that returns the best point for the next sample, which I again leave general as the function optimization(). This is summarized by pseudo-code in Algorithm 3.5.

Algorithm 3.5: Bayesian Optimization

```
Input: n, k_max
Result: x*, f(x)* k = 0
x = {}
f = {}
for i = 0 to n do
    x_k = sample()
    f_k = f(x_k)
    k = k + 1
GP = fit(x, f)
x_G, f_G = min(f)
while not converged do
    x_k = optimize(acquisition(GP))
    f_k = f(x_k) GP = fit(x, f)
    k = k + 1
    if f_k < f_G then
        x_G = x_k
        f_G = f_k
    check for convergence
x* = x_G
f(x)* = f_G
```

I initially tried Bayesian optimization with the *bayesian-optimization* package by randomly sampling the design space 50 times, then proceeding with 500 iterations based on the maximum of the acquisition function for a total of 550 iterations. The initial test found all samples in constraint violation regions and did not find any improved optima over the baseline. Next, I probed the first point as the baseline presented in Table 3.5 followed by 49 random samples. Most of the sampling was far from the baseline and the result was a best optima of the initial point sampled. I restricted the design space more and tried again, this time Bayesian optimization found a very good optima in only 550 function evaluations and less than 20 minutes of computation time. This was repeated two more times, each with similar results and the same optima.

Method Selection: Each of the four methods tested did find improved designs from the initial design point. Therefore, they are all feasible paths forward for optimizing the non-simplified problem. Table 3.7 is a summary of the performance of each of the methods. I show RHC for its effectiveness as multi-start method in a multi-modal space, with the combined computation time and function evaluations from all points tested.

Table 3.7: Comparison of optimization methods.

Optimization Algorithm	$f(\boldsymbol{x})$	$m_{\text{propellant}}$ (kg)	ζ	t_{burn} (s)	OD (in.)	Function Evaluations	Time (min)
RHC (multi-start)	103.3	103.3	0.610	32.28	10.68	625	11.64
Simulated Annealing	99.2	97.6	0.616	30.49	10.33	28,409	364.1
Particle Swarm	99.1	97.5	0.608	30.46	10.14	4,600	85.1
Bayesian Optimization	**99.2**	**97.5**	**0.616**	**30.46**	**10.42**	**550**	**18.54**

Table 3.7 shows that particle swarm and simulated annealing both perform worse than Bayesian optimization to find the same optima, both take significantly more computation time and function evaluations. RHC finds an optima that is nearly as good as the other three methods in just over half the time of Bayesian optimization, so it is possible that it could be improved over Bayesian optimization with addition of a few more random starting points. However, even with only four starting points, RHC surpassed the number of function evaluations of Bayesian optimization and found a worse global optima. In application to the non-simplified optimization, each function evaluation increases in computation time from approximately 1-2 seconds to 30-40 seconds. Therefore, the additional time Bayesian optimization spent optimizing the acquisition function will be far less than the additional computation time of nearly a hundred extra function evaluations. Based on the results in Table 3.7, I selected Bayesian optimization as the best suited method of the four methods explored.

3.3 Design Qualification

In nearly all designs, there is at least one failure mode that has a non-zero probability. Failures should always be avoided, but in different contexts failures can have drastically different effects. In mass production, the primary effect of a failure may be the cost of labor and materials that impacts the overall profit of a company. To reduce the failure rate may increase the manufacturing costs, therefore there is typically some acceptable non-zero failure rate from balancing the increased costs of manufacturing and the losses due to failure. In the case of human rated spaceflight, failure could directly result in loss of life. In human rated spaceflight, failure must be minimized regardless of cost due to the effects of failure.

It may be possible to directly compute the probability of failure occurring for simple designs. An example could be an assembly of parts fitting together, where the failure mode is they do not fit. Given knowledge on the manufacturing tolerances, the probability of parts not fitting together could be calculated without too much difficulty. As system complexity increases, both the number of contributing factors to a failure occurring and the number of possible failures can greatly increase. In the case of complex systems, it may be difficult to directly compute the probability of failures occurring. The case of a launch vehicle design is an example of a complex system that is difficult to directly compute failure probabilities. To account for failure probability in launch vehicle design, a robust method for design qualification must be employed.

To qualify a vehicle design, the performance effects of uncertainties in the design and environment must be considered. A standard method of determining the effects is through the use of Monte Carlo simulations. In Monte Carlo simulations, a set of known uncertainties and associated distributions are defined. Each of these distributions are randomly sampled, and performance is simulated for the set of parameters. After performing many samples, statistical analysis can be done on the results to determine sensitivity to each of the uncertainties. This method can be used to quantify the confidence that a vehicle will meet the mission objective despite many uncertainties in the design.

3.3.1 Uncertainty

There are numerous aspects of vehicle design subject to uncertainty. I grouped uncertainty into two categories: epistemic and aleatoric uncertainty [65, 66].

Epistemic Uncertainty: Epistemic uncertainty is typically described as imprecise knowledge of a parameter with a fixed value. Epistemic uncertainty could be caused from issues such as manufacturing capabilities, a common epistemic uncertainty are dimensional uncertainties. A fixed tolerance range can be assigned to a part, which can be manufactured within the specification, however the value of the dimension cannot be known precisely prior to manufacturing. Additionally, epistemic uncertainty can be a result of measurement error. For the same example, the uncertainty on a dimension of a part which has been manufactured could be reduced by measuring it, but is still limited based on the accuracy of the instrument used.

Aleatoric Uncertainty: Aleatoric uncertainty is caused by inherent randomness in the system that prevents prior knowledge of the exact conditions. An example of aleatoric uncertainty that is particularly relevant to this work is weather. Measurements can be used to develop forecasts based on previous weather trends, however weather conditions at any future time cannot be known with perfect certainty. The most relevant weather phenomena in this context are wind and gust modeling, which cannot precisely be known prior to the flight of a rocket. Based on historical data, weather modeling can be developed and applied to simulations to determine expected results in varying wind conditions.

Distributions: There are numerous distributions that could be used to model a parameter of interest. Particularly for uncertainties which have not previously been measured repeatedly, selection of a distribution can often be difficult. Within the scope of this work, I only used normal distributions. In a normal, or Gaussian, distribution values have a certain probability of being a distance from the population mean. For a normal distribution with a standard deviation σ, there is approximately a 68 % probability of a random sample being within 1σ, approximately a 95 % probability of a random sample being within 2σ, and approximately 99.7 % probability of being within 3σ.

3.3.1.1 Design Uncertainty:

Important parameters and their associated distributions have been established. In the scope of this work, I focused on the development and testing of the tools used to qualify a design, therefore the ranges and distribution types warrant further refinement and exploration for complete qualification of the design. Shown in Table 3.8 is a list of each parameter with uncertainty and the standard deviation of the distribution.

Table 3.8: Summary of epistemic uncertainties.

Design Parameter	**Standard Deviation**
Pressurant regulator pressure	50 psi
Fuel sonic venturi throat diameter	0.0005 in.
Oxidizer sonic venturi throat diameter	0.0005 in.
Initial fuel tank pressure	10 psi
Initial oxidizer tank pressure	10 psi
Nozzle efficiency	1.0 %
Nozzle throat diameter	0.001 in.
Nozzle exit diameter	0.001 in.
Coefficient of drag[1]	5.1 %
Derivative of normal force coefficient[1]	5.1 %
Center of pressure location	0.5 calibers
Center of gravity location	0.5 in.
Launch rail angle from vertical[2]	1.5°

1 - Uncertainty is enforced as a scalar multiplier

2 - Launch rail angle from vertical is in random direction

The only aleatoric uncertainty I considered is associated with weather. Both the atmospheric conditions and the wind conditions are unknown and cannot be precisely predicted prior to a launch. The NASA Earth-GRAM software is capable of modeling this and accounts for spatial and temporal variation in weather [41]. Section 3.1.5.2 provides more detail on the Earth-GRAM model used in this work.

3.3.2 Statistical Analysis

I conducted Monte Carlo simulations using the uncertainties described in Section 3.3.1 by randomly sampling each distribution and using the NASA Earth-GRAM software to generate a random atmosphere and wind condition. There are two possible outcomes of each sample in a Monte Carlo simulation: success or failure. I defined success as achieving the mission objective by having an apogee altitude equal to or greater than the Kármán line. For this work, I defined a failure simply as a sample that did not meet the mission objective. Future work could also classify failures caused from parameters exceeding bounds such as structural stresses above the yield strength of the material or engine thermal failure.

For any vehicle design, there is a non-zero failure rate that is unknown. A limit should be placed on acceptable failure rates to qualify the design based on maximum allowable failure rate. NASA standards use a 99.865 % success rate [65]. The impacts of mission failure for the vehicle of focus in this work are far less significant than for a standard NASA mission. Therefore, I considered a 95 % success rate to be acceptable. Determination of a minimum allowable success rate should be performed carefully with knowledge of the resulting impacts of mission failure.

A simple approach to determining failure rate would be to divide the number of recorded failures by the total number of samples observed. As the number of samples observed approaches infinity, the observed failure rate will tend towards the actual failure rate. However, taking very large samples is undesirable due to computational expense of each sample. A confidence interval can be placed on failure rate with a finite number of samples using binomial statistics. The probability of the actual success rate being lower than the acceptable success rate is called consumer risk. The NASA standards are to use a maximum of 10 % consumer risk, which I adopted for this work. Stated in its entirety, the standard for a reliable design in this work is to have a 95 % success rate with less than 10 % consumer risk.

At any given point during Monte Carlo simulations, both the number of samples, N, and the of failures, k, observed are known. For a given failure rate, p, there is a known probability of observing exactly k failures. This probability can be calculated by

$$P\left(k|N,p\right) = \left(\begin{array}{c} N \\ k \end{array} \right) p^k \left(1-p\right)^{N-k}. \tag{3.182}$$

Additionally, the maximum likelihood estimation of the failure rate is the same as the current sample failure rate and can be calculated as

$$\hat{p} = \frac{k}{N}. \tag{3.183}$$

There is a probability density function associated with the failure rate. The probability of observing k or fewer failures given a maximum allowable failure rate p_{design} is calculated with

$$\beta = \sum_{j=0}^{k} \left(\begin{array}{c} N \\ j \end{array} \right) p_{\text{design}}^j \left(1-p_{\text{design}}\right)^{N-j}. \tag{3.184}$$

where β is the consumer risk [65, 66]. With established values for consumer risk and maximum acceptable failure rates of $\beta = 10\,\%$ and $p_{\text{design}} = 5\,\%$, Equation (3.184) can be solved for a given number of failures and number of samples observed.

An important note on design qualification through Monte Carlo simulations: it can only be as accurate as the models and distributions used. If the flight simulation models or the uncertainty distributions are inaccurate or incomplete, the results will be inaccurate. Therefore it is critically important to ensure both the flight simulation used and the uncertainty models are as accurate as possible for the results of Monte Carlo simulations to be meaningful. Relevant test data should be compared to models used in Monte Carlo simulations as components are manufactured wherever possible, and in some cases may be able to be directly substituted in place of a theoretical model.

3.3.3 Failure Classification

After performing a Monte Carlo simulation, the design was assessed for meeting an acceptable failure rate with a predefined consumer risk. However, analysis of the failures to try to gain more information is possible by attempting to classify similar failures. This can be useful for both improvement in design and prediction of failures. Sensitivity to certain parameters can be determined by grouping similar failures.

When a parameter is identified that the system is sensitive to, there are three possible decisions to improve the design: moving the mean value, reducing the uncertainty, or accepting the system sensitivity to the parameter. I identified sensitive parameters by finding a limit state which represents a threshold for success and failure. For the scope of this work, all mission failures are classified the same regardless of cause, therefore each simulation is able to be classified by two states,

$$s = \begin{cases} 1 & \text{if mission success} \\ 0 & \text{if not mission success} \end{cases}, \tag{3.185}$$

where the designations 1 and 0 are arbitrary integers. Further classifications are possible and in some cases may be useful for design decisions. For example, classification of a structural failure versus a failure to reach the target apogee altitude may have two very different design changes to reduce the likelihood of the failure occurring. This could make it useful to expand the number of failure states to determine how best to improve the design.

3.3.3.1 k-Nearest Neighbor Method

One method to classify failures in the design space[4] is the k-NN method. The k-NN method works by evaluating a point of interest based on the nearest surrounding points [67, 68]. For a specific failure mode, it is logical to assume that sampling points nearby points were observed to fail would also fail. The k-NN method for classification works by computing the distance from a point of interest to all neighboring points. Then, the points are sorted based on distance. The mode of the state (failed or successful) of the nearest k points is assumed to be the state of the point of interest. This allows failure regions to be distinctly observed and the factors that cause them can be further investigated. The value of k must not be a multiple of the number of possible states to prevent the mode of the neighbors from being a tie. I used a value of three for k. The observations from the Monte Carlo method provide many existing data points that the k-NN method can use to make predictions about the failure classification.

[4]The design space for failure classification is slightly different from the design space in optimization contexts as used in Section 3.2.1 In this case, the design space consists of all variables with associated uncertainty.

A further improvement on the k-NN method is to place a fixed maximum radius of neighboring points. This is important, because the further away the point of interest is from any neighbors, the less accurate the classification will be. Unexplored regions of the design space that are of interest should be explored via further sampling of the design space and simulating these areas. For a maximum radius to be effective, all design variables must be normalized. The choice of maximum radius is best found through empirical testing on the data set of interest, and is dependent on the value of k chosen. For high values of k, the maximum radius must be larger to classify the same amount of the design space. Increasing the value of k and the maximum radius will have a smoothing effect on the data set [68].

For two parameters, it is easy to look at the nearest points surrounding a point of interest and count the number of nearby failures and successes to determine which is most likely. As the number of variables with associated uncertainty increases, it becomes impossible to visualize the entire space at once. Therefore, the k-NN method has been developed to quantify the number of nearby failures.

I tested the method on two sample 2D problems with a theoretical boundary. The design space is randomly sampled, and failures and successes are recorded. The first problem classifies successes as sample points within a circle. The second problem classifies success as values of $f(x)$ less than an arbitrary boundary curve,

$$g(x) = x + \frac{\sin(4\pi x)}{\pi}. \tag{3.186}$$

For both problems, sampling distributions were normal distributions with mean values at the center of the plot. I classified 300 random points, then generated 100 random points and used the k-NN method was used to predict their classification. The results of the sampling problems, along with the theoretical boundary are shown in Figure 3.12.

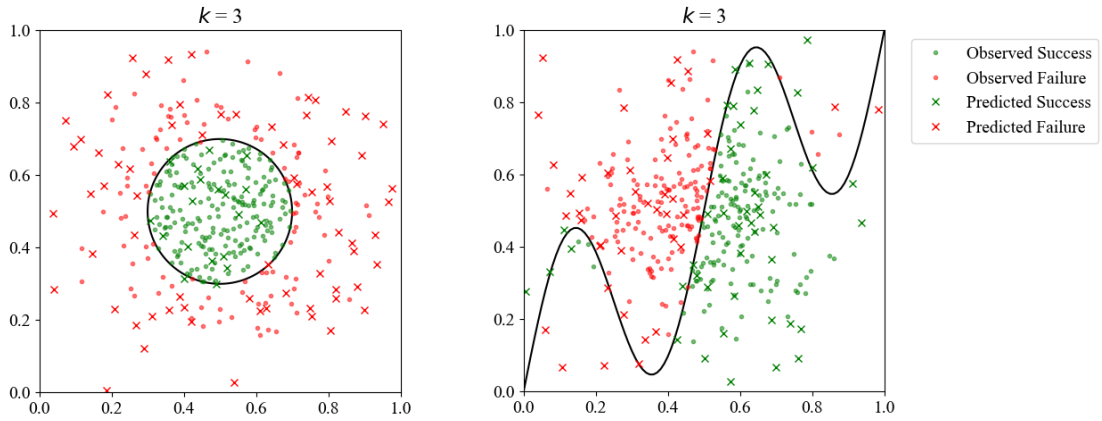

Figure 3.12: k-NN example

Figure 3.12 shows that the k-NN method is clearly capable of classifying failures with reasonable accuracy, however it is also worth illustration of areas where k-NN is weak. The circle problem correctly classifies all except two sample points that are just across the boundary, an error rate of 2 %. This could be used to develop a reasonably accurate boundary curve to denote failure regions. The second problem misidentifies ten points, an error rate of 10 %. Each of the misidentified points fall in regions where the boundary curve was not initially sampled many times (near the corners $(x, y) = (0, 0)$ and $(x, y) = (1, 1)$). The k-NN method could only provide a reasonably accurate boundary curve prediction near the center of the plot, towards the corners it is prone to significant error. This illustrates a weakness of classification, it is only reliable when the surrounding region has been reasonably well sampled.

To account for this weakness, I imposed a maximum distance for nearby neighbors. If there are not k neighbors less than the maximum distance from the point of interest, it is left unclassified. The same problems were re-tested with a maximum radius, $r_{\max} = 0.1$. This time, k-NN only misidentified one point for the circle problem and two for the second problem, as shown in Figure 3.13.

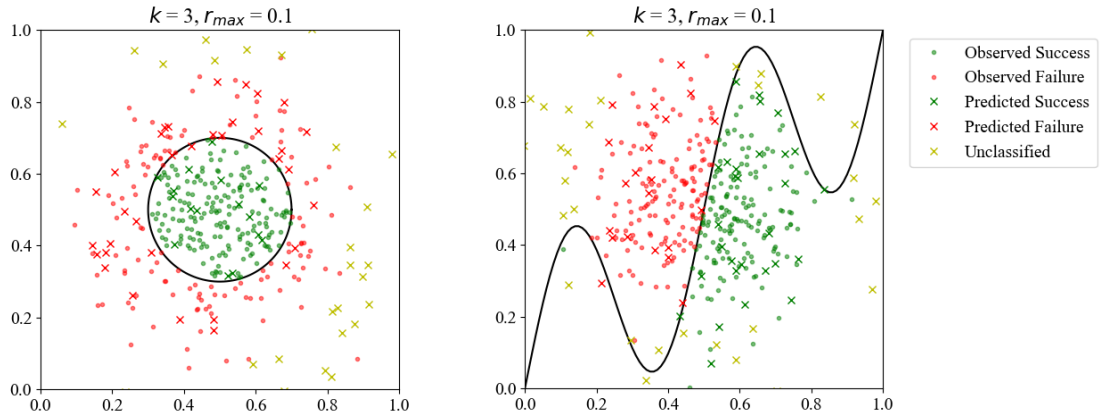

Figure 3.13: k-NN example with maximum radius

The maximum radius is helpful to reduces issues with inaccurate classifications, but introduces the problem of unclassified regions. To account for unclassified regions in areas of interest, directed sampling can be performed in the regions of interest prior to using the k-NN method in these areas. An example of when this could be important is if test data is far off from the predicted mean value, the nearby region may have been minimally sampled by the Monte Carlo method. Directed sampling could be implemented to determine if the results of the test data demonstrate the vehicle is in or nearby a failure region.

To implement a k-NN algorithm, a the parameter for k must be selected along with a maximum radius, $r_{\max}$. The inputs to the algorithm is a point of interest, $\boldsymbol{x}_0$, and a set of previously observed points $\boldsymbol{p}$ with their observed states $\boldsymbol{s}$. The algorithm returns an estimated state for the point, $\hat{s}$. The distance between the point of interest and a previously observed point, $\boldsymbol{x}_i$ is computed based on the Euclidian distance in the design space, d_i. The calculation of the Euclidian distance is represented by the norm() function. The states of the nearest neighbors, s_{nn}, are stored and the mode computed. Sample pseudocode is shown in Algorithm 3.6.

Other classification methods exist, but are not thoroughly explored or covered in this work. Many classification methods have been developed for machine learning including Support Vector Machine (SVM) and naive Bayes classification. The k-NN method was chosen over these for its relative simplicity in implementation while providing results with

Algorithm 3.6: k-NN Method

Input: $\boldsymbol{x}_0, \boldsymbol{p}, \boldsymbol{s}$, $r_{\max}$
Result: $\hat{s}$
$\boldsymbol{d} = \{\}$
for $\boldsymbol{x}_i$ *in* $\boldsymbol{p}$ **do**
    $d_i = \text{norm}(\boldsymbol{x}_i - \boldsymbol{x}_0)$
$\boldsymbol{d}_{\text{sorted}}, \boldsymbol{s}_{\text{sorted}} = \text{sort}(\boldsymbol{d}, \boldsymbol{s})$
$\boldsymbol{s}_{nn} = \boldsymbol{s}_{\text{sorted}}[0 : k-1]$ `// Choose states of first` k `neighbors`
if $\boldsymbol{s}_{nn}[-1] \leq r_{\max}$ **then**
    $\hat{s} = \text{mode}(\boldsymbol{s}_{nn})$
else
    $\hat{s} = \text{None}$ `// Unclassified if a neighbor is outside max radius`

reasonable accuracy. Future work could include comparison of each of these methods to determine the most suitable method for this application.

3.3.3.2 Variable Normalization

The examples shown in Section 3.3.3.1 worked well for establishing a maximum radius. This works well for problems where the variables of interest are the same scale, however the distance between points is based on the Euclidian distance. In the case of a real problem, the variables can have vastly different scales and a maximum radius becomes useless. Therefore, I normalized each of the variables of interest.

A simple method of normalizing variables is to subtract the sample mean, μ, and divide by the sample standard deviation, σ for each of the i sample points of a variable ϕ,

$$\phi_{i,\text{normalized}} = \frac{\phi_i - \mu}{\sigma} \,. \tag{3.187}$$

This method is best for distributions which are normal or not highly skewed.

Chapter 4: Results & Discussion

4.1 Flight Modeling

All of the models discussed in Section 3.1 were implemented in Python. A sample of the outputs generated by this implementation of a flight simulation are shown within this section for an arbitrary design. Figure 4.1 shows a standard profile for the mission of focus in this work. The phases of flight are generally notable on this diagram. The powered ascent phase is identifiable by the rapidly increasing velocity at the start of flight, until engine cutoff when the velocity begins to lower. During the unpowered ascent phase of flight, velocity continues to lower as it exits the atmosphere, until reaching its peak altitude at apogee where vertical velocity is zero. At apogee, the vehicle enters the drogue descent phase. However, the vehicle continues to rapidly accelerate downward under drogue until it re-enters the dense atmosphere where drag force is comparable to force from gravity. After re-entering the dense atmosphere, the vehicle slows to a terminal velocity and descends at a stable speed. Near the very end of flight, at 1000 ft, the main parachute is deployed for landing. The entire flight takes around 17 minutes from takeoff to landing.

Figure 4.2 shows the Mach number throughout the ascent of the vehicle, peaking at a maximum of Mach 4.92. This gives us a range of Mach numbers to evaluate aerodynamic coefficients from 0 to 5.

The Barrowman method for calculating the drag coefficient at varying Mach numbers is shown in Figure 4.3. Both the RASAero II output and Barrowman method implemented in Python agree well in this work, with differences less than 10 % for the majority of the profile. There is a spike in difference in the transonic region, which analysis of the RASAero II data appears to only be consider from Mach 0.9 to Mach 1.05. I considered the transonic region from Mach 0.8 to Mach 1.2, critically the Busemann second-order expanded shock theory is only applicable for attached oblique shock waves to the nosecone and fins. By assuming thin fins and a sharp nosecone, Hilton [16] considers the range of applicability of the Busemann second-order expanded shock theory to be applicable

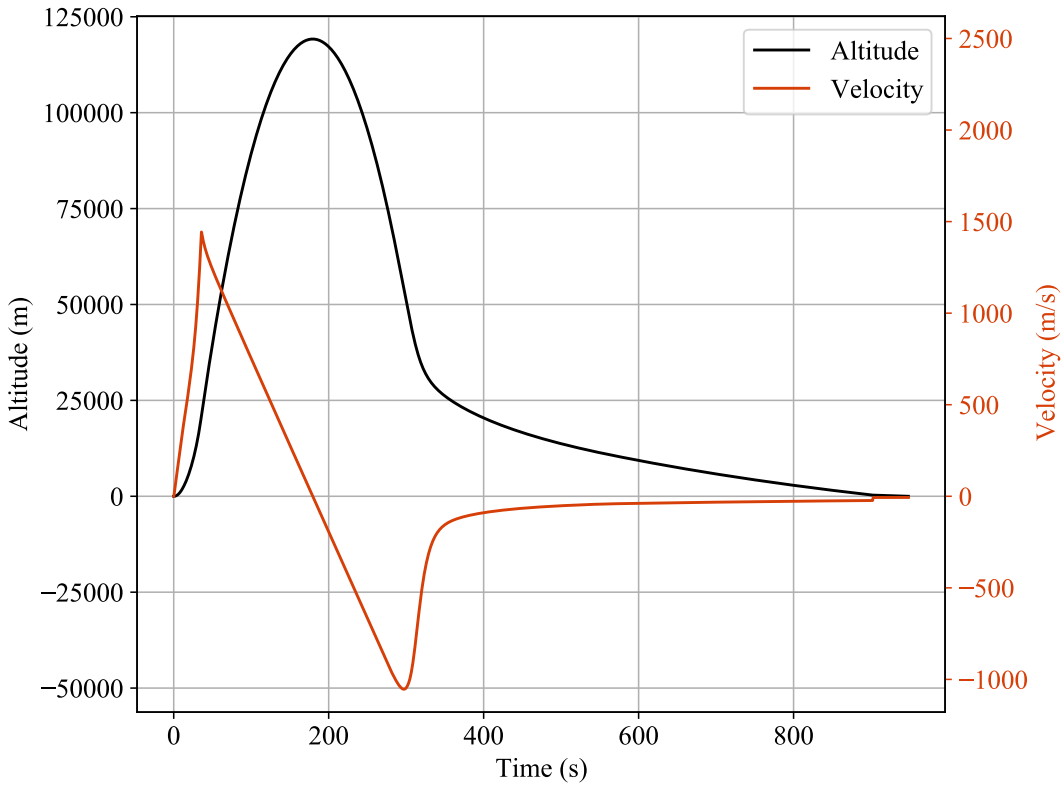

Figure 4.1: Flight profile for the duration of a mission to the Kármán line with design parameters of $\zeta = 61.25\,\%$ and $m_{\text{propellant}} = 112\,\text{kg}$.

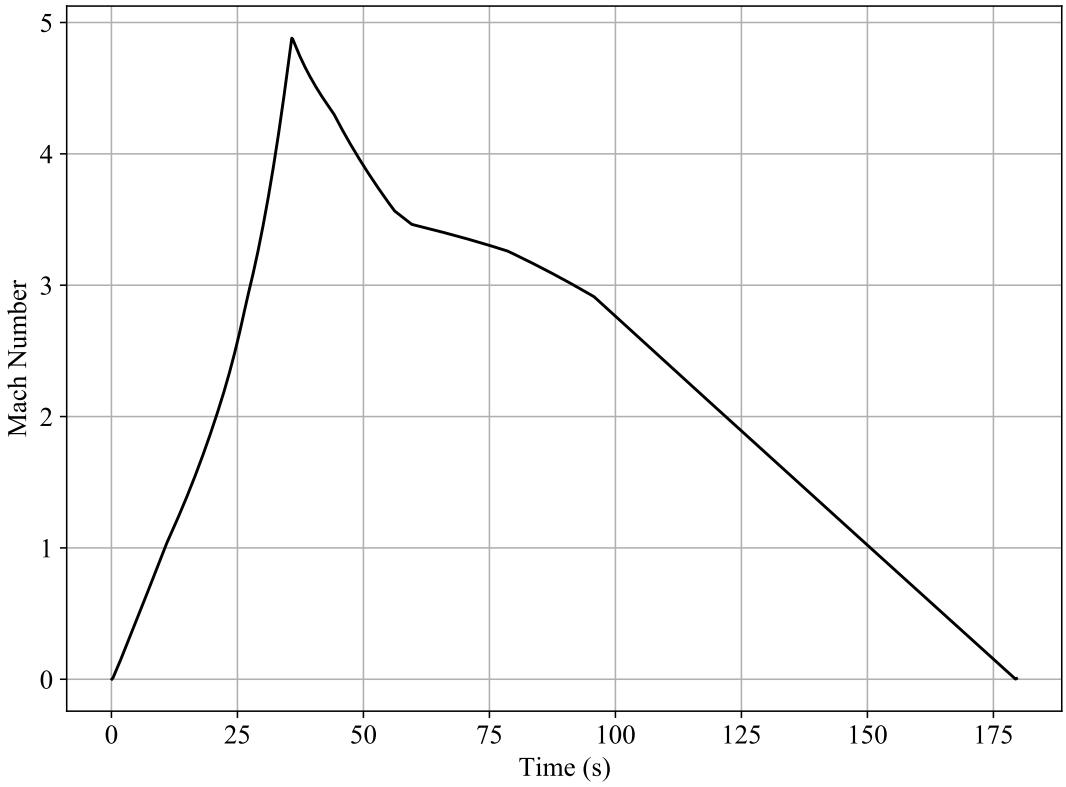

Figure 4.2: Mach number throughout ascent.

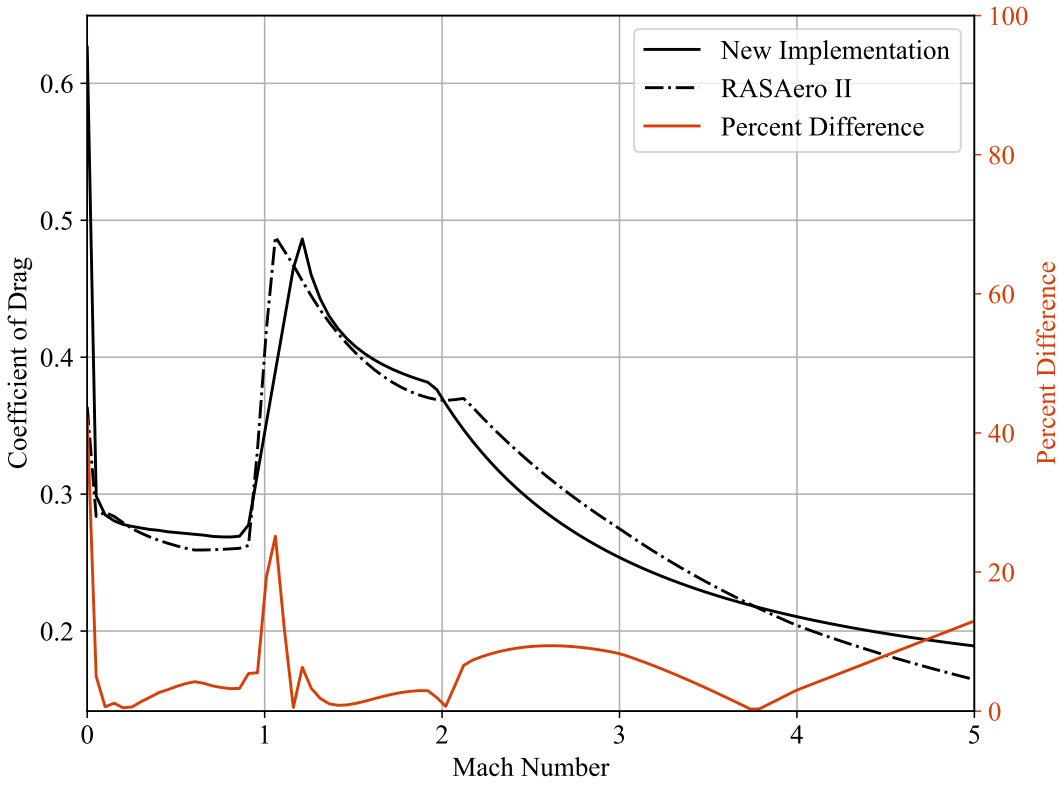

Figure 4.3: Vehicle drag coefficient for varying mach number on ascent.

at a minimum supersonic value of Mach 1.2. The Barrowman method does not address the transonic region, so I have followed the RASAero II implementation and fit a line between the last point where the flow is subsonic until the first point where flow is fully supersonic. There is some disagreement between the shape of the curves in the supersonic region, it's unclear what the source of this is. However, based on empirical data collected by Barrowman, he concludes that the method is accurate to within 10 % for calculating aerodynamic coefficients. This fits the uncertainty used in Monte Carlo simulations, therefore both implementations are within the accepted accuracy for almost the entire duration of flight.

My implementation of the Barrowman method is compared RASAero II for the center of pressure and derivative of the normal force coefficient in Figures 4.4 and 4.5. In the

subsonic region, RASAero II appears to use a constant derivative of the normal force coefficient, despite the Barrowman method being a function of the Prandtl—Glauert number, $\beta = \sqrt{1 - M^2}$, as shown in Equation (3.64). The constant value used in RASAero II agrees with my calculation at approximately Mach 0.8. During the transonic region, I followed the method of RASAero II to use a constant negative slope of the normal force coefficient, so both implementations agree very well. The high supersonic regions above Mach 3.0 appear to have both methods converging to nearly the same values. However, between the region Mach 1.2 and Mach 3.0, my implementation differs from RASAero II significantly. The center of pressure location has a significant spike in the low supersonic region around Mach 1.2 which would suggest the vehicle would become unstable. The Rogers Modified Barrowman implementation in RASAero II is known to give good results in this regime, therefore I believe this spike to be artificial. I do not currently know if the additional modifications made by Rogers in the Rogers Modified Barrowman method [14] are the source of these differences seen in Figures 4.4 and 4.5. I have verified the Busemann coefficients with those tabulated by Hilton [16]. Figure 4.6 shows the Busemann coefficients, which match Hilton's tabulated values and charts exactly. Additionally I compared my implementation with the source code in OpenRocket created by Niskanen [13], but the full supersonic Barrowman method appeared to be incomplete. Niskanen does provide warnings in the OpenRocket software for accuracy of the normal force coefficient in the supersonic regime. Without being able to reconcile the differences, I choose to use the Rogers Modified Barrowman method output from RASAero II for the aerodynamic coefficients used in the remaining simulations presented in this work. This ensures accurate aerodynamic coefficients which allows us to conduct accurate simulations. An area of future work is to resolve the differences between RASAero II and my implementation.

The stability margin is measured in calibers and is maintained above 1.5 calibers for the duration of flight. A caliber is defined as the distance between the center of gravity and center of pressure divided by the vehicle diameter at the base of the nosecone. A vehicle with stability of over 1.0 calibers is considered passively stable, though it is common practice to design for a minimum stability above 1.5 caliber to prevent instability due to inaccuracy in center of pressure or center of gravity calculations. Figure 4.7 shows the stability throughout flight.

The dynamic pressure during flight places constraints on the loads the vehicle must

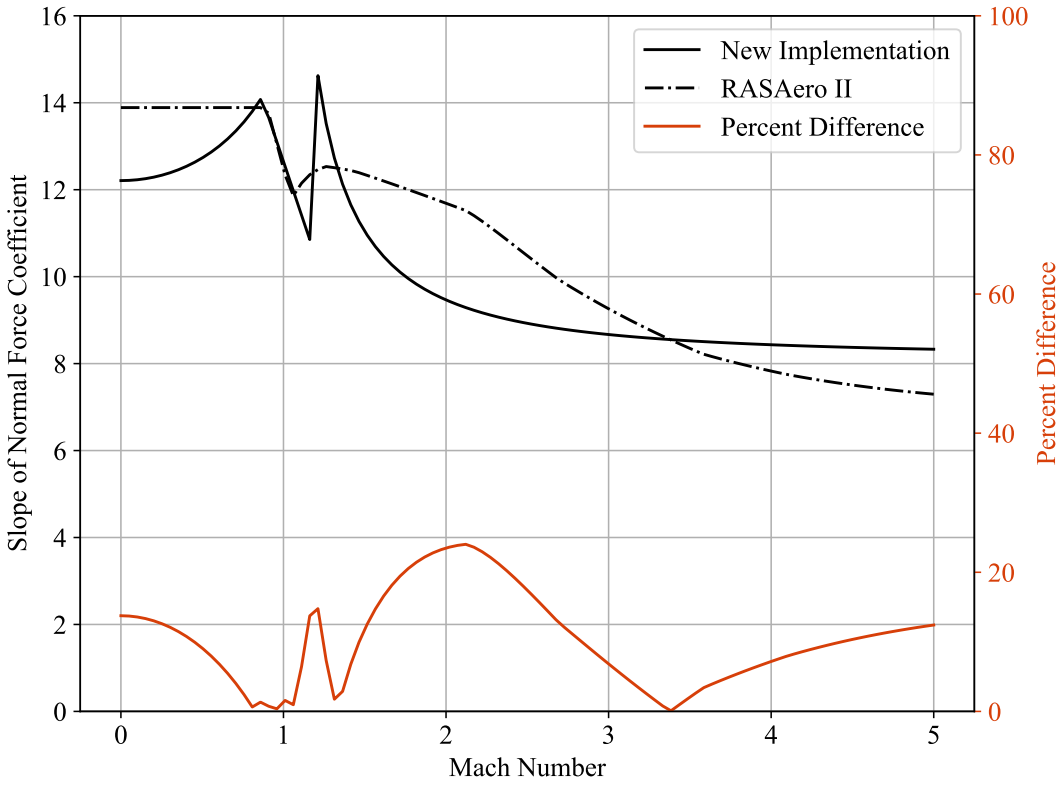

Figure 4.4: Derivative of the normal force coefficient for varying mach number on ascent.

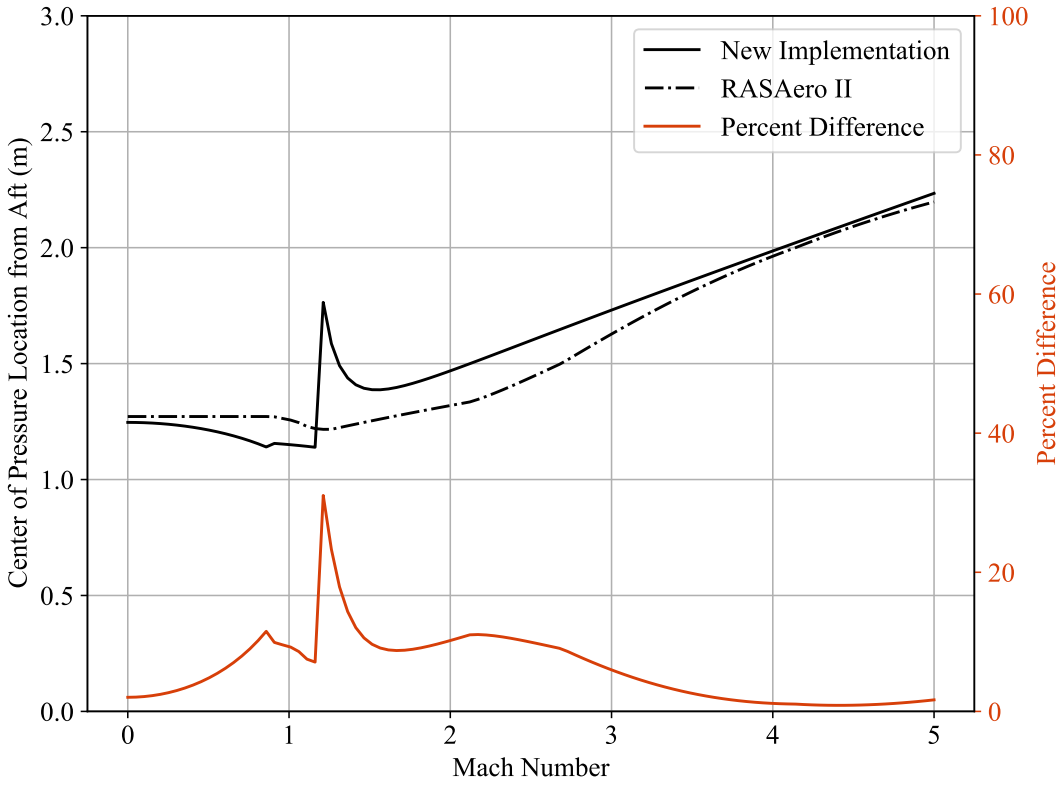

Figure 4.5: Center of pressure location for varying mach number on ascent.

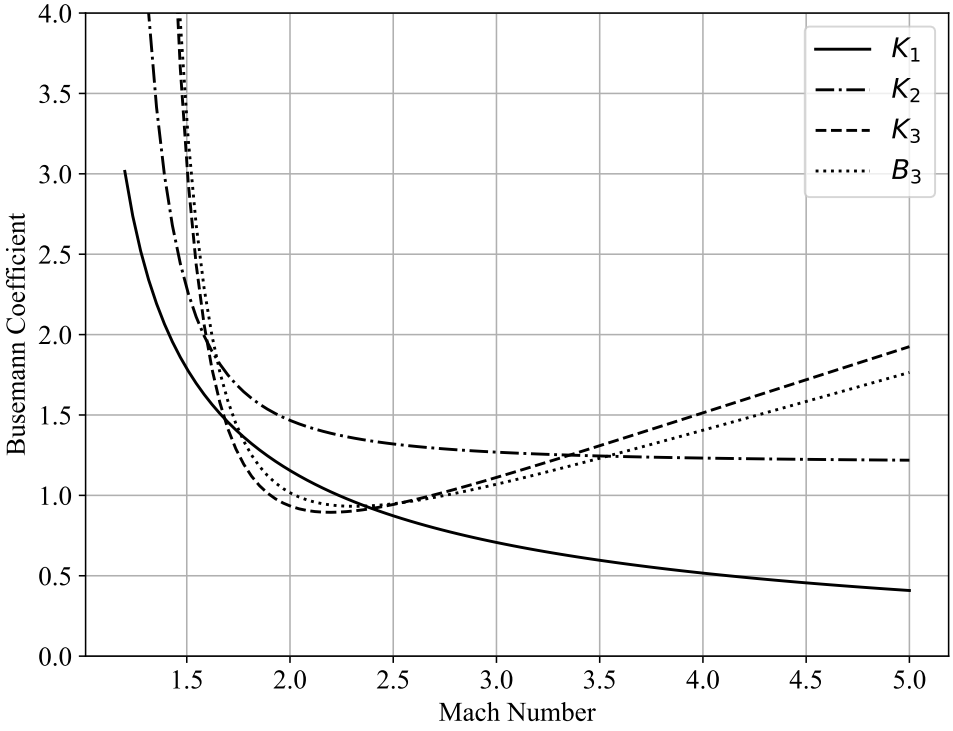

Figure 4.6: Busemann coefficients for varying Mach number

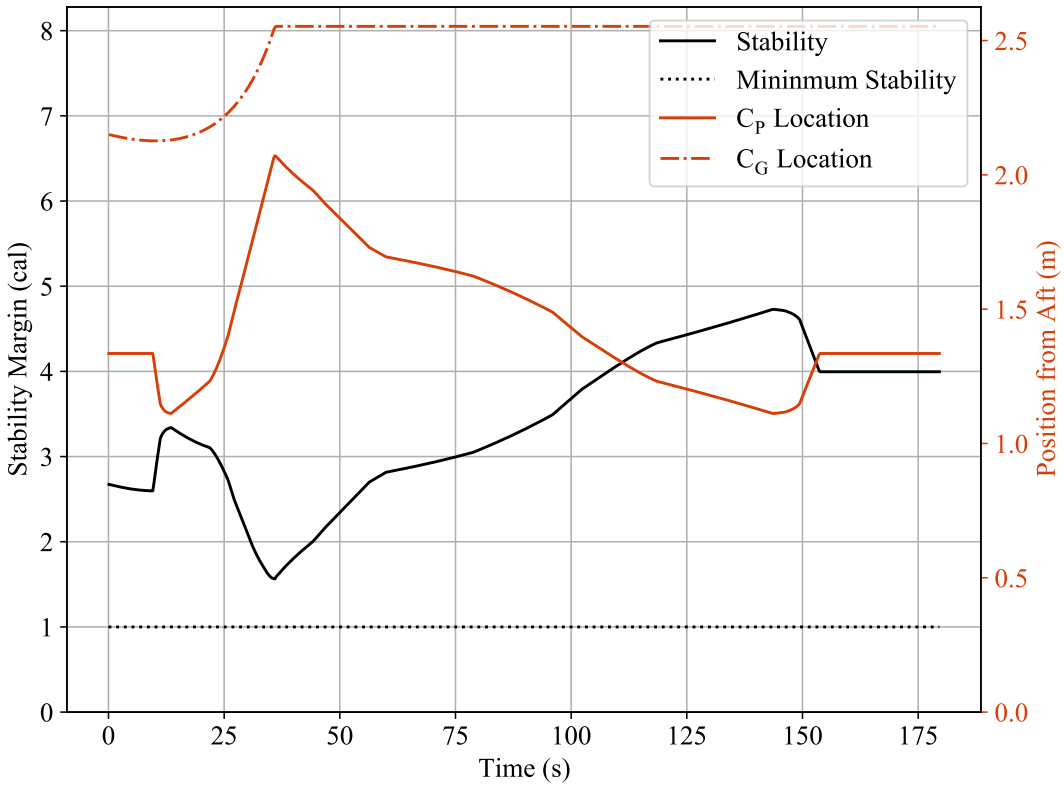

Figure 4.7: Stability margin throughout ascent based on distance from center of gravity to center of pressure.

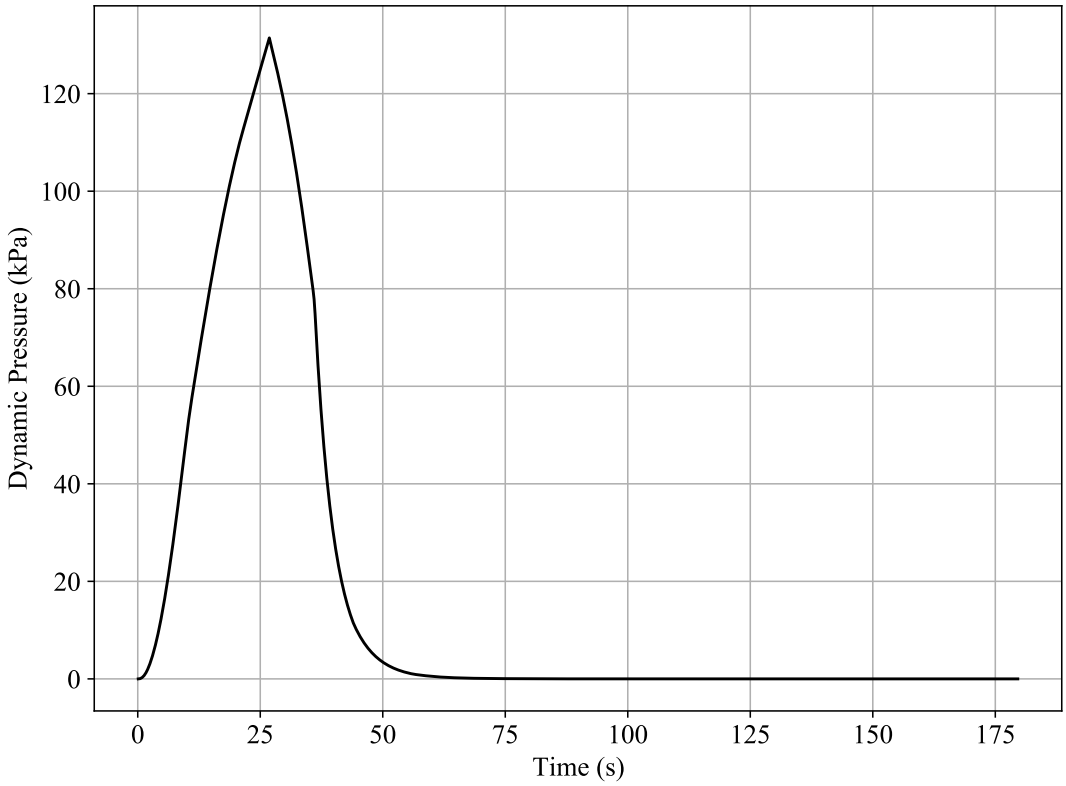

Figure 4.8: Dynamic pressure on the vehicle throughout ascent.

be able to withstand. Dynamic pressure is a function of both velocity and air density, as velocity increases air density decreases. Figure 4.8 shows the dynamic pressure throughout the ascent of the vehicle.

For recovery modeling it is important to verify some assumptions used during component selection. Recovery components were sized to withstand $500\,\mathrm{m/s^2}$ of acceleration in an off-nominal recovery event, but should only be expected to see around $100\,\mathrm{m/s^2}$ in nominal recoveries. This work assumes instantaneous parachute inflation, this is a conservative assumption that will produce slightly higher accelerations than experienced by the vehicle. Both the drogue and main parachute inflation events (the two peaks after engine burnout) show accelerations below this threshold. This verifies component weights assumed for recovery are reasonable.

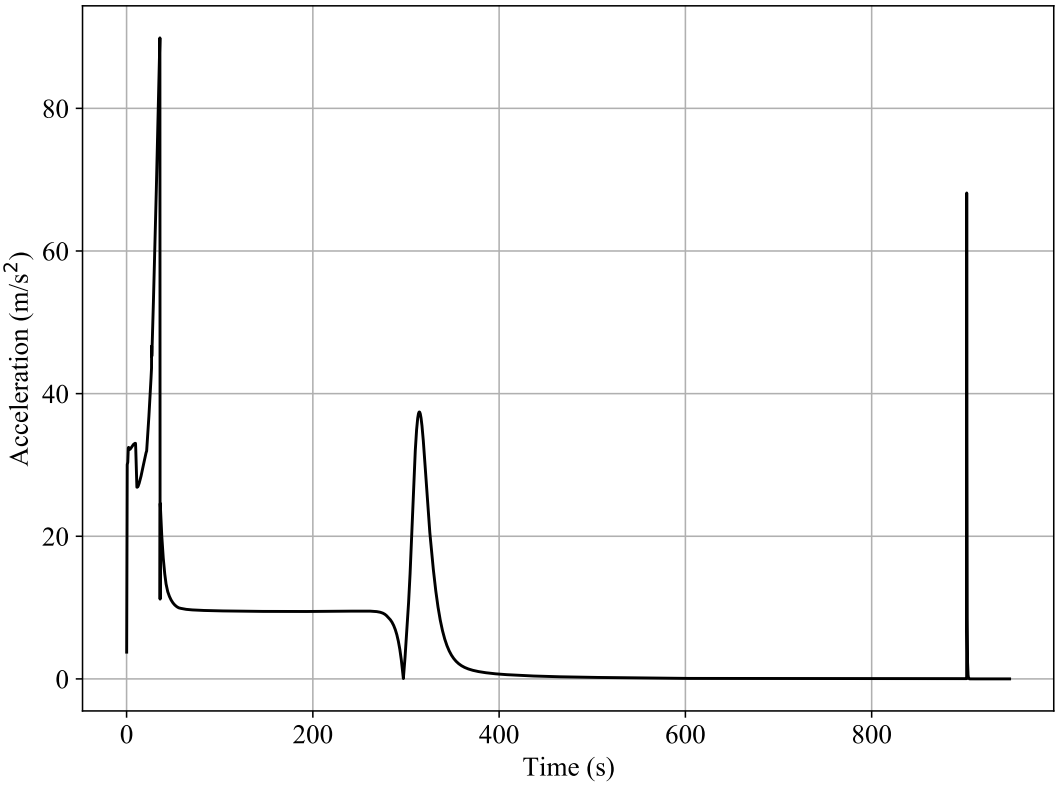

Figure 4.9: Magnitude of acceleration throughout the duration of flight.

As described in Section 3.1.7, sonic venturis are used to choke the mass flow of pressurant so a single pressure regulator can be used to independently control O/F ratio within the engine. Figure 4.10 shows sonic venturis maintaining constant O/F ratio by choking the mass flow rates of pressurant gas. I plotted tank pressures and O/F ratio for a single Monte Carlo sample in Figure 4.10. The design oxidizer-to-fuel ratio is 2.30, and the results in Figure 4.10 show that it rapidly steadies to a value of 2.294 for this Monte Carlo sample. This analysis assumed tanks are filled to 95 % full and start with 5 % ullage volume.

The thrust produced by the engine is dependent on the chamber pressure and the atmospheric pressure throughout flight. As shown in Figure 4.10, the chamber pressure decreases throughout most of the engine burn, which would cause the thrust to decrease.

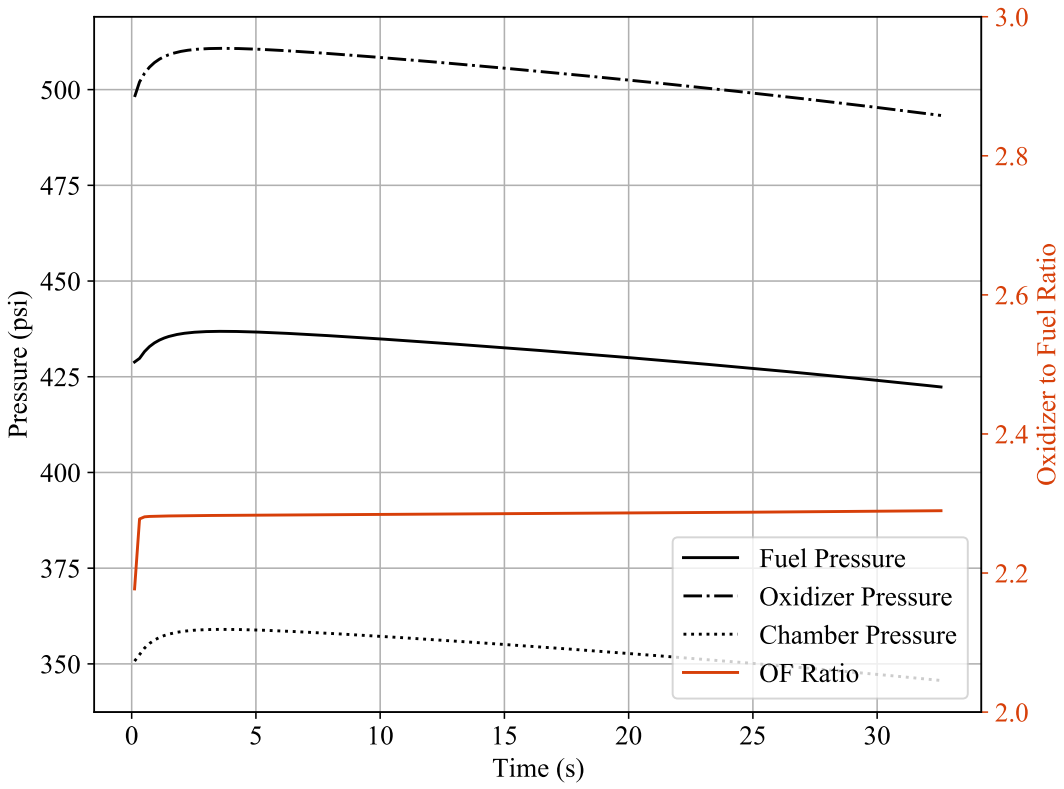

Figure 4.10: Pressures and O/F ratio throughout engine burn for a single Monte Carlo sample.

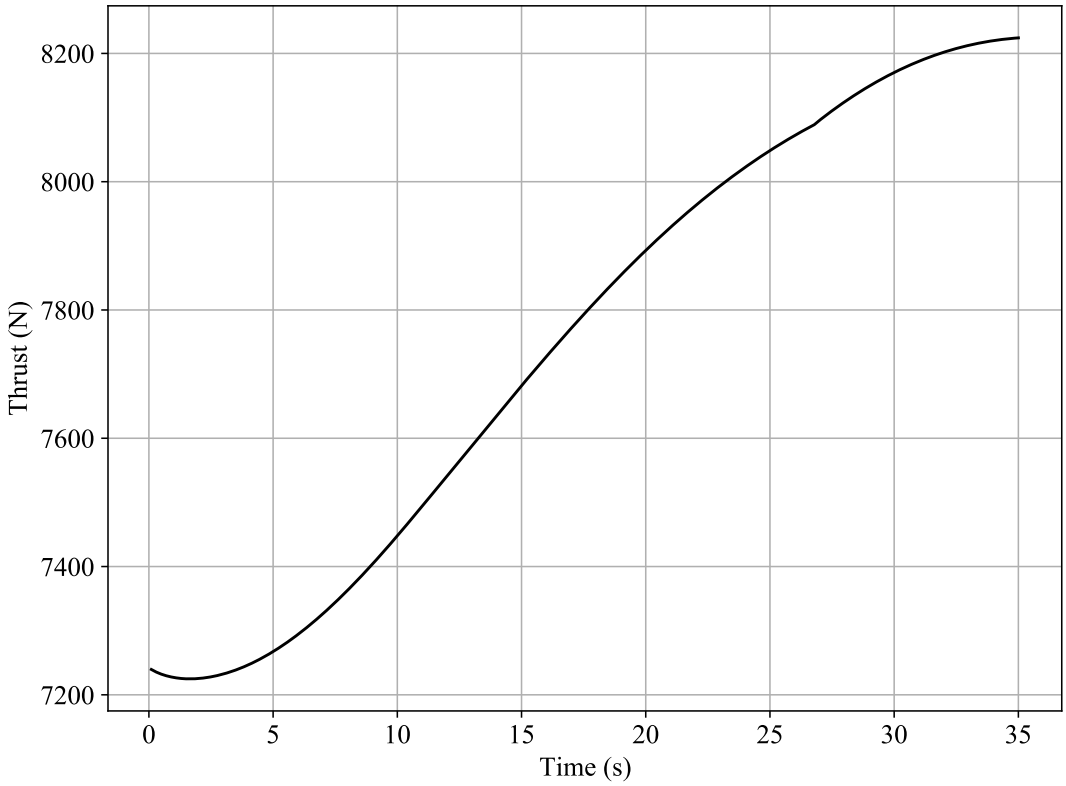

Figure 4.11: Thrust curve throughout engine burn

However, the atmospheric pressure also decreases throughout flight, which causes the thrust to increase. Shown in Figure 4.11 is the resulting thrust curve from the two of these effects.

I compared the flight simulation developed in this work against a RASAero II simulation for the same vehicle design. The results are summarized in Table 4.1. Both agree within 7.4 % for several constant wind conditions. Three notable differences in the simulations were identified that represent the most likely cause of differences. RASAero II uses standard motor files, which provide time varying information about the mass and thrust of a solid rocket motor that are available for many types of solid rocket motors. For the liquid engine performance modeled in this work, there is no such motor file, so a file was written for a thrust versus time curve that must then be interpolated by RASAero II.

Though the method is not stated by Rogers and Cooper [14], RASAero II likely performs linear interpolation that results in small differences between the thrust curves used in the two simulations. Additionally, RASAero II assumes a constant center of gravity for the motor about its center, resulting in a linear movement of the center of gravity of the vehicle during the engine burn for constant mass flow rate. This assumption is invalid for a liquid engine. This assumption in RASAero II overestimates the stability throughout flight, and results in an artificially large moment arm for aerodynamic forces to act. The artificially large moment arm would make the RASAero II simulations more susceptible to wind losses, which agrees with the results in Table 4.1. Lastly, RASAero II does not allow for moments of inertia to be specified or allow for component locations and masses to be implemented. Rogers and Cooper [14] do not state how RASAero II computes the moments of inertia, and differences here would also change simulation susceptibility to wind.

Table 4.1: A comparison of apogee altitudes for the custom flight simulation written in Python used for this work with RASAero II.

Wind Speed	0.0 mph	5.0 mph	10.0 mph	15.0 mph	20.0 mph
Python Apogee (m)	130,634	130,555	129,745	125,088	119,071
RASAero II Apogee (m)	127,681	123,931	120,852	116,669	111,978
Percent Difference	2.3 %	5.3 %	7.4 %	7.2 %	6.3 %

The reasonably small difference between the two simulations shown in Table 4.1 demonstrates the simulation results shown here compare well to those found using other flight simulation software that are widely used in the high-power rocketry community.

4.2 Vehicle Design

The Bayesian optimization algorithm was run on the vehicle design starting from an initial known design. Three constraints were enforced using a square penalty method: an aspect ratio less than 20, a mass margin greater than 1 kg, and an apogee altitude of greater than 120 km. The apogee altitude constraint was placed on the design to allow margin for wind losses. The optimization identified a design requiring 102.05 kg of propellant, but slightly violates the aspect ratio, mass margin, and apogee altitude

Table 4.2: Bayesian optimization results summary.

Parameter	Initial	Optimized	$OD \leq 9.625$ in.	$OD \geq 10.75$ in.
Propellant Mass Fraction	61.13 %	60.0 %	59.3 %	60.4 %
Outer Diameter (in.)	12.00	10.40	9.62	10.78
Burn Time (s)	35.00	31.89	30.64	32.55
Propellant mass (kg)	112.00	102.05	98.05	104.17
Apogee Altitude (km)	119.20	118.76	114.27	120.05
Aspect Ratio	18.96	23.18	26.01	21.96
Mass Margin (kg)	3.58	0.36	-0.70	0.61
Objective Function	113.51	103.89	106.50	104.56

constraints. This shows that all constraints are active for the optimization problem. The optimized design had a 10.40 in. outer diameter. This is an issue, because there is no standard raw materials found at this size.

To discretize the outer diameter, I used the branch and bound method. The nearest standard outer diameter of nominal pipe sizes are 10.75 in. and 9.625 in. I ran the optimization algorithm once with a high-bound set at $OD \leq 9.625$ in. and once with a low-bound set at $OD \geq 10.75$ in. If both of these optimizations converge at the bounds, the discretized optima is the best achievable discretized design. However, if the optimizations converge away from the bounds, I must discretize again by repeating a branch and bound step. The branch and bound method converged very near to both boundaries, with the outer diameters being 9.62 in. and 10.78 in. The 10.78 in. diameter result required a propellant mass of 104.17 kg and had less violation of the constraints than the 9.62 in. diameter result. I considered this close enough to be discretized as a nominal 10.75 in. diameter. Table 4.2 shows the results of each optimization. Figure 4.12 shows the algorithm closing on the optima for each trial, where every point shown is a new best.

Using Bayesian optimization, I was able to optimize the high level design parameters to reduce propellant mass by 7.0 % with a discretized outer diameter of 10.75 in. Because each of the constraints are active and enforced with a penalty method, the optimal design slightly violates the constraints. However, the constraint limits were specifically chosen to keep small violation allowable without resulting in an infeasible design.

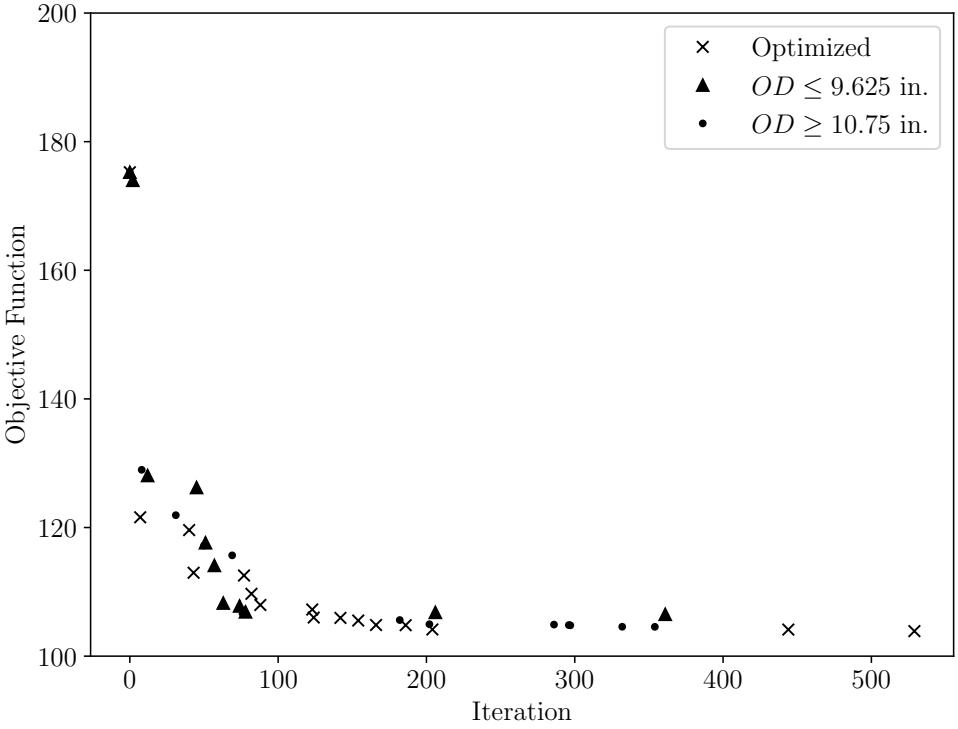

Figure 4.12: Bayesian optimization algorithm finding improved points.

4.3 Design Qualification

To qualify a design, I used Monte Carlo simulations for design uncertainties and then performed failure classification. The distributions for epistemic uncertainties used are shown in Table 3.8. Aleatoric uncertainties for atmosphere and weather conditions were performed using the Earth-GRAM software for coordinates of $(32.21°, -106.38°)$ with an eight hour launch window starting at 8:00 AM local time on December 1st, 2021 as described in Section 3.1.5.2.

4.3.1 Monte Carlo Simulation Results

I tested numerous different propellant mass and propellant mass fraction combinations to identify a feasible design that would meet my mission reliability target. The best combination tested was for a propellant mass of $m_{\text{propellant}} = 112\,\text{kg}$ and a propellant mass fraction of $\zeta = 61.25\,\%$. For this combination, I ran a set of 4000 Monte Carlo simulations which had 157 failures for a sample failure rate of $\hat{p} = 3.93\,\%$. For a 95 % reliability, this evaluates to a consumer risk of $\beta = 0.072\,\%$, well below the target of 95 % reliability with less than 10 % consumer risk. The average apogee altitude was 113,771 m with a standard deviation of 5156 m. Figure 4.13 shows a histogram of the apogee altitudes, where the vertical line represents the minimum apogee altitude that must be achieved for mission success.

Figure 4.14 shows a histogram of steady-state O/F ratios. The distribution of steady-state O/F ratios from all Monte Carlo samples is tightly grouped near the design point of 2.30, verifying that using sonic venturis to regulate pressurant mass flow rate will provide good combustion performance despite variation in initial tank pressures, sonic venturi throat diameters, and regulator pressure. It's useful to quantify the range of O/F ratios a future sample would fall in, so I fit a 95 % prediction interval to the O/F ratio samples assuming a normal distribution. I calculated a prediction interval of 2.248 to 2.349, with 95 % confidence. The upper end of this prediction interval could be used for finding maximum combustion temperatures, useful for sizing ablative liners or regenerative cooling channels.

Figure 4.15 is a histogram of the maximum dynamic pressures experienced by the vehicle in each simulation. A 95 % prediction interval on the maximum dynamic pressure

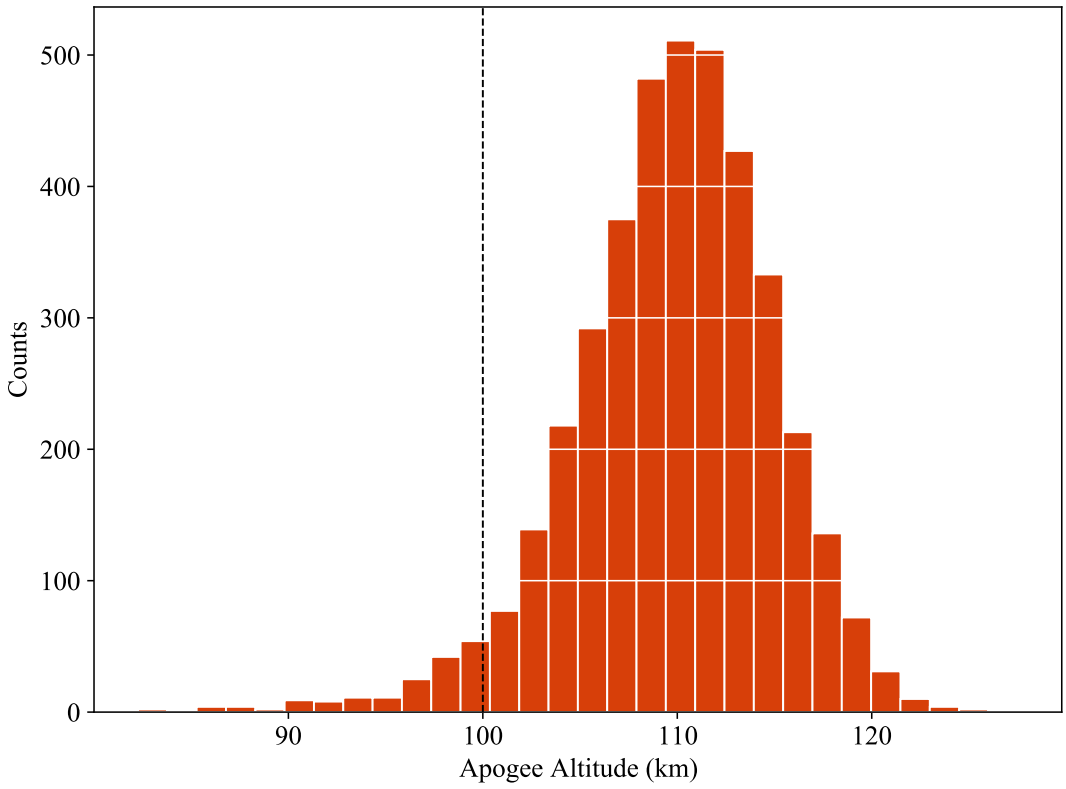

Figure 4.13: Distribution of apogee altitudes from Monte Carlo simulations. Vertical line represents minimum apogee altitude for mission success.

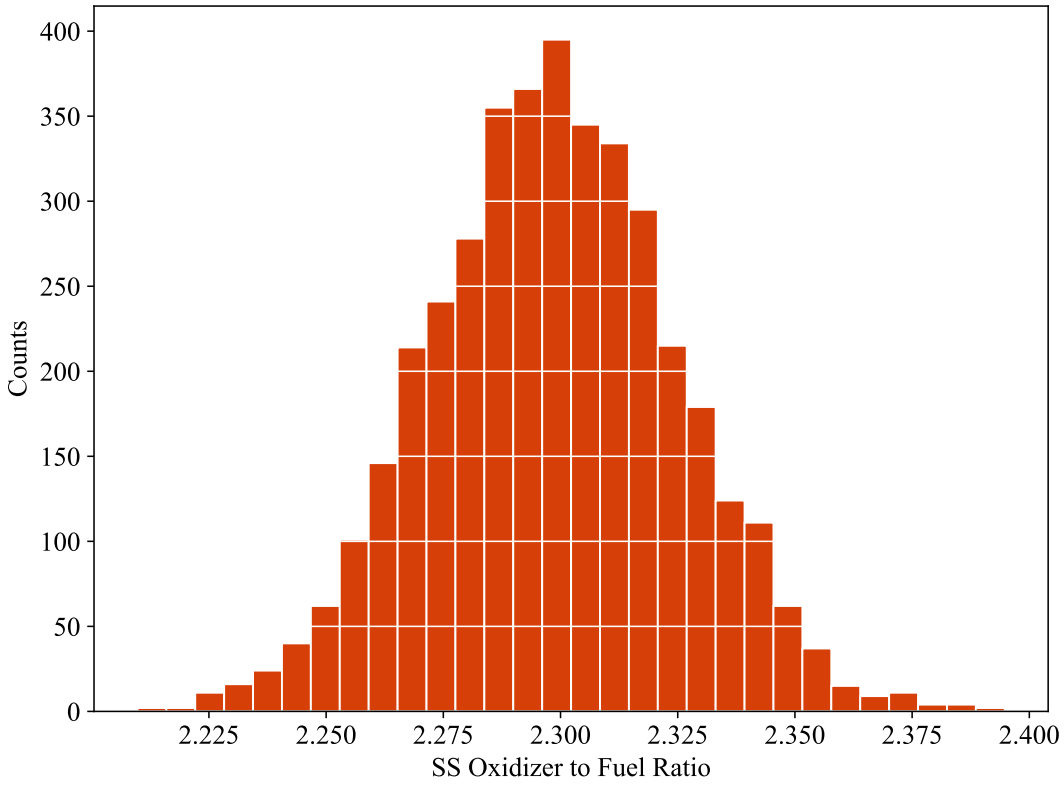

Figure 4.14: Distribution of steady-state oxidizer-to-fuel ratios.

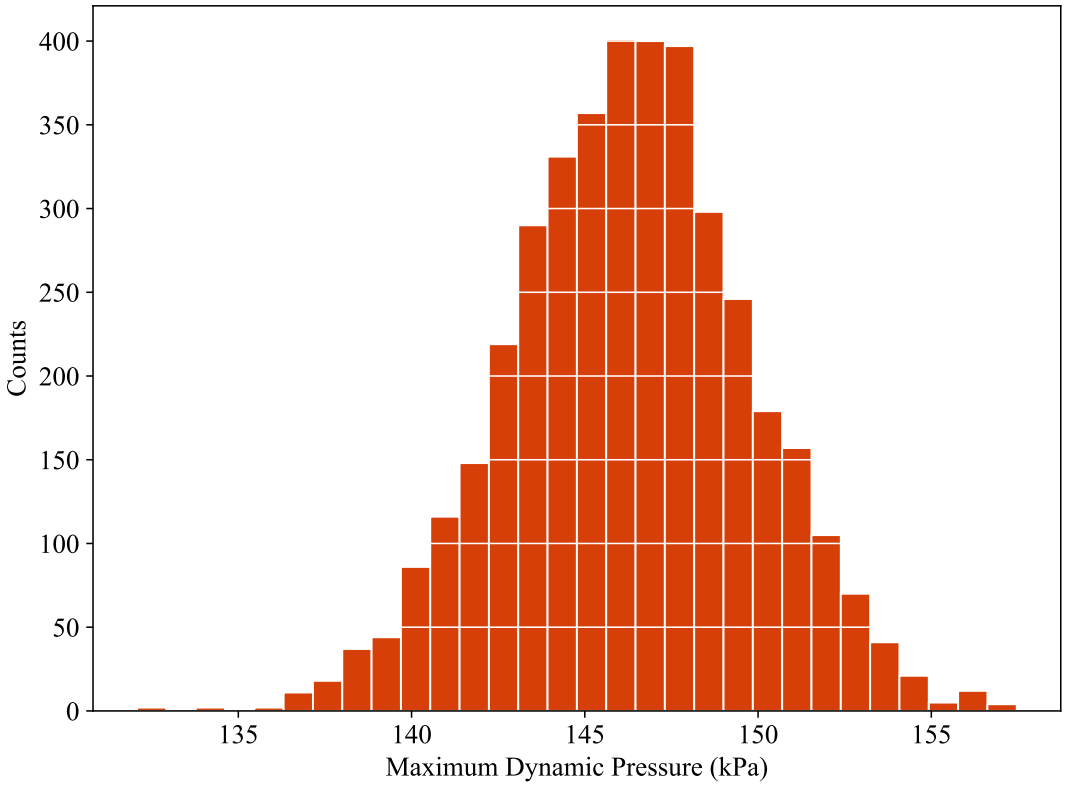

Figure 4.15: Distribution of maximum dynamic pressures

is from 139.5 kPa to 153.0 kPa. The upper end of this prediction interval could be used as a constraint for structural design.

I used the results of the Monte Carlo simulations to predict landing the recovery zone of the vehicle through dispersion analyses. I calculated the eigenvalues and eigenvectors of the covariance matrix. The direction of the eigenvector corresponding to the largest eigenvalue is corresponds to the direction of major axis of the ellipse. The critical values of a chi squared distribution for 1, 2, and 3 standard deviations are used with the eigenvalues to determine the major and minor axis lengths. Figure 4.16 is the landing locations of each Monte Carlo simulation with 1, 2, and 3 standard deviation elliptical confidence regions fit to the data. The landing locations shown in Figure 4.16 assume a nominal engine burn and recovery performance for the flight. I could use this method to generate

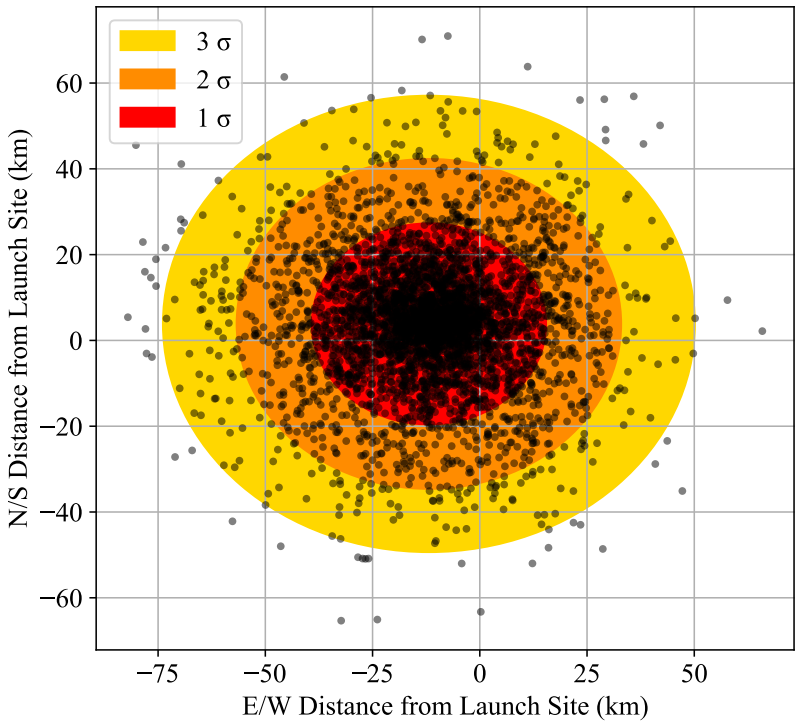

Figure 4.16: Dispersion analysis of the landing locations of the vehicle for nominal trajectory.

additional plots for failed recovery or drogue parachute deployment only if desired by slightly modifying the state machine discussed in Section 3.1.2.

I tested the optimized vehicle design with a 10.75 in. outer diameter summarized in Section 4.2 and found the vehicle to have an even lower failure rate of 1.82 % with 73 failures occurring in 4000 observations. For 95 % reliability, this evaluates to a consumer risk level of 5.6×10^{-26}, well below the maximum acceptance criteria of 10 % consumer risk. This indicates that the value I chose for the altitude constraint on the apogee altitude was chosen as a conservative value.

4.3.2 Classification Results

I applied a k-NN algorithm to the data from Monte Carlo simulations, as described in Section 3.3.3.1. My intent was to use data already collected to identify areas of improvement in the design that could increase reliability. I tested the k-NN algorithm on the first Monte Carlo data set collected, which had 4000 observations and a sample failure of 3.93 %.

Applying the k-NN algorithm to the normalized data set obtained from the Monte Carlo simulations had limited success. I attempted to classify 5000 randomly sampled points in the design space, the k-NN algorithm identified 4512 as success, 30 as failures, and left 458 points unclassified. Only 30 failures identified would suggest a failure rate of 0.6 %. This failure rate is not reflective of the data collected through Monte Carlo samples and suggests many of the unclassified points should have been classified as failures. In general, the k-NN algorithm was only able to classify points as successes, or leave them unclassified. I expect this is because over 95 % of the samples taken were successful, so the failure region was not nearly as heavily sampled as the success region.

Though I did not test any other algorithms, I expect that applying other classification algorithms to the same data set would suffer similar issues as the k-NN algorithm. The algorithm worked as intended, but it showed the data set from Monte Carlo simulations is not well suited for a classification algorithm to identify a limit state boundary. Because of this issue, I was unable to use the k-NN algorithm to identify areas of improvement in the design. Despite this, the designs still met my reliability criteria.

I was able to identify three promising paths for future work on failure classification. However, each method has challenges. The first is to greatly increase the number of samples taken during Monte Carlo simulations. This method is computationally expensive and requires more time for simulations to run than was available for this work. The second is to use a directed sampling routine nearby the limit state boundary. This method requires developing a directed sampling routine. Lastly, I could use the k-NN algorithm in its current state, but call the flight simulation each time a point is left unclassified. This is perhaps the most promising and easiest to implement method, but does not meet my original intent of this analysis to use the data set generated by Monte Carlo simulations. Future work could expand on the paths outlined here.

4.4 Limitations & Future Work

This section provides a summary of limitations for the current flight simulator I developed in Python and opportunities for future work. This is intended to be a reference for future development and refinement of the tool. Here I summarize the most notable assumptions that warrant further investigation and results which have demonstrated limitations of the flight simulator. More detailed explanations of these limitations are provided throughout this work.

I assumed the transient phases for engine startup and engine shutdown to be negligible and made no attempt to model these. For short duration engine burns, this assumption would be more significant. I used the NASA CEA software for combustion modeling based on availability, however the TDK model is an alternative that should be investigated to provide better modeling of the two dimensional flow through the engine to the JANNAF standard.

My current model for propellant tanks assumes propellant slosh within the propellant tanks to be negligible, I did an analysis in [10] that confirmed the vehicle will remain stable despite propellant slosh. Additionally, I assume ullage collapse of the pressurant gas within the cryogenic liquid oxygen tank to be negligible. Vortices produced by roll of the vehicle are not modeled, these could allow for gas bubbles to enter the propellant lines and potentially cause engine failure. Surface effects from vortices and propellant slosh would slightly effect the center of gravity location of the liquid propellant, I assumed this effect to be negligible. My propulsion model is specific to a pressure-fed engine with sonic venturis to regulate mass flow of pressurant gas to the propellant tanks, additional models would be required to simulate other engine cycles.

My implementation of the Barrowman method is inaccurate in the supersonic region. I have not been able to identify the source of this inaccuracy to date, it is the largest known limitation of the current software. To retain accuracy in my simulations, I used RASAero II to calculate aerodynamic coefficients and import them to my software.

The recovery modeling in this work assumed constant coefficient of drag for each parachute. This assumption is reasonable for flights within the dense portion of the atmosphere. However for the mission of focus in this work, the vehicle exits the atmosphere and re-enters at supersonic speeds. The drogue parachute drag coefficient would vary significantly in this region. Additionally, I assumed the body drag produced by the

airframe to be negligible. For more accurate dispersion analyses, I would correct these assumptions. The multi-body dynamics of recovery could also be modeled for improved accuracy, though I expect this would significantly increase computation time for only modest improvement in accuracy.

My optimization method assumes the aerodynamic coefficients are relatively constant as the vehicle design is scaled. After correcting the Barrowman method in the supersonic region, this assumption is no longer necessary. Additionally this would also allow for aerodynamic optimization of the vehicle fins for the mission profile. My optimization is also limited by the estimations used for component masses, which will be further refined as the component level design of the vehicle is completed. I did not consider engine optimization for the mission in this work because we have previously tested the engine of focus in this work and were interested in a vehicle design that could use this engine to complete the target mission. Future work could perform engine optimization for a mission. Lastly, I was unable to implement a tree search algorithm within this work because of the increased model complexity.

In this work, I was able to validate design reliability, but I unsuccessfully implemented a k-NN algorithm for identifying ways to improve the design. The k-NN algorithm was used on the data set produced by Monte Carlo sampling, which proved to not be a good data set for training a machine learning algorithm on failures because of the scarcity of these within the data set for a reliable design. I would need to implement a method for importance sampling to complete this.

Though not a limitation, an opportunity for future development of the tool would be to include the option to model vehicles flying with solid or hybrid rocket motors. This would be useful for the high-powered rocketry community which most commonly uses solid rocket motors, but currently has no access to tools for reliability analyses and limited tools for optimization.

Lastly, the simulator is currently limited to suborbital flights. Modeling orbital flights would require control of a vehicles trajectory and implementing orbital mechanics models. Additionally, I have not investigated atmospheric models for on-orbit atmospheric drag.

Chapter 5: Conclusions

This work identified limitations in available simulation software for liquid engine launch vehicles. I addressed these limitations through model selection specific to a liquid engine launch vehicle that targets the Kármán line (100 km), though the models could more generally be applied to an arbitrary suborbital mission. I developed a 6DoF simulation in Python that implemented these models to evaluate the performance of a liquid engine launch vehicle design.

I used optimization algorithms to improve vehicle designs by reducing propellant mass. The random hill climb, simulated annealing, particle swarm, and Bayesian optimization algorithms were compared for a simplified version of the problem and lead to the selection of Bayesian optimization. I used Bayesian optimization to find a set of high level design variables which minimize the required propellant mass, providing a propellant mass reduction of 7.0 % over the baseline design.

I did not know how parameters which have associated uncertainty would effect vehicle performance. I identified a set of uncertainties and conducted Monte Carlo simulations to determine the reliability of a vehicle design. Designs were qualified to meet a standard of 95 % reliability with less than 10 % consumer risk. I conducted 4000 Monte Carlo samples for an initial design and saw 157 failures for a sample failure rate of 3.93 %. The consumer risk of this design for 95 % reliability evaluates to 0.072 %. I additionally tested the result of my optimized design with 4000 Monte Carlo samples and observed 73 failures for a sample failure rate of 1.82 %, a consumer risk of 5.26×10^{-26} for 95 % reliability. Both designs well exceeded my established reliability criteria for qualifying a design is robust to uncertainties.

Both the results of the optimization and of Monte Carlo simulations are subject to the accuracy of the models used in my flight simulation. Inaccuracies or incomplete models of all relevant phenomena to vehicle performance would cause inaccuracy in both the optimization and design qualification results. The best way to verify the models used are applicable to the designs considered in this work is to compare to empirical test data for validation wherever applicable.

The primary contributions of this work are to: address limitations in accurately simulating liquid engine launch vehicles in available software by selecting applicable models and developing a new flight simulation, to apply optimization algorithms to high level vehicle parameters, and to implement methods to qualify design reliability in achieving a mission through Monte Carlo simulations. Available software prior to this work were unable to perform the same optimizations or assess design reliability. The methods for optimization and design qualification could be extended to any moderately computationally expensive black-box model, such as high fidelity simulations of aircraft, satellites, or automobiles by changing the simulation to incorporate applicable models.

Future work would include further verification and validation of the simulation and models used within this work through empirical test data wherever possible. Additionally, future work is needed to validate the distributions used in Monte Carlo simulations to insure they reflect actual uncertainties the vehicle could experience. Future optimization work could be applied through exploration of the design tree to determine an optimal component configuration. I had limited success with classification through application of a k-NN algorithm in this work, but this is an opportunity for future work to develop an automated process to improve reliability of a vehicle design. Future work could be done on each of the limitations explained in Section 4.4, most notably on addressing the limitations for calculating aerodynamic coefficients in the supersonic regime. Implementing models for solid and hybrid rocket motors could allow the high-powered rocketry community access to a flight simulator capable of performing reliability analyses and optimization of high level vehicle design parameters. Lastly, future work could be done to improve the usability of the flight simulation software for a broader community to be able to use these tools, such as development of a graphical user interface.

Two versions of the tools developed in this work are available: the version I used for the results contained within this work [69], and a open source version available online at `github.com/rosetrevor-osu/Liquid-Engine-Rocket-Simulations/`. Short term planned updates to the open source version include improvements the user interface, while longer term updates will seek to address the limitations discussed in this work.

Bibliography

[1] JC Newby, "The History of Rockets", Physics Education **23**, 266–271 (1988).

[2] T Kuntz, "150th Anniversary: 1851-2001; The Facts That Got Away", (2001).

[3] J Chladek, *Outposts on the Frontier: A Fifty-Year History of Space Stations* (ProQuest Ebook Central, Nebraska, US, 2017).

[4] S Rogers, *NASA Budgets: US Spending on Space Travel Since 1958*, Feb. 2010, Accessed Sept. 14, 2021 [Online]. Available: `http://www.theguardian.com/news/datablog/2010/feb/01/nasa-budgets-us-spending-space-travel`.

[5] *SpaceX*, 2021, Accessed Sept. 14, 2021 [Online]. Available: `http://www.spacex.com`.

[6] DK Huzel and DH Huang, *Design of Liquid Propellant Rocket Engines*, 2nd (Science & Technical Information Division of NASA, Washington, D.C., 1967).

[7] GP Sutton and O Biblarz, *Rocket Propulsion Elements*, 9th (Wiley, Hoboken, N.J, 2017).

[8] D Burson, C Harms, A Ragle, A Gulstrom, O Clark, M Hayden, T Rose, K Burson, and J Morgan, "HALE Preliminary Design Review", Unpublished (2018).

[9] D Burson, C Harms, A Ragle, A Gulstrom, O Clark, M Hayden, T Rose, K Burson, J Morgan, P Strohmaier, J Van de Lindt, K Gopal, J Schwartz, G Nourot, and C Young, "HALE Static Fire Review", Unpublished (2019).

[10] T Rose, C Harms, A Ragle, O Clark, M Hayden, J Morgan, D Burson, and K Burson, "HALE Critical Design Review", Unpublished (2021).

[11] C Byrd, *What's the Record for a University Liquid Bipropellant Engine?*, July 2018.

[12] MASA, *Laika - First Liquid-Bipropellant Rocket Launched at Spaceport America Cup*, 2021.

[13] S Niskanen, "Development of an Open Source Model Rocket Simulation Software", MA thesis (Helsinki University of Technology, May 2009).

[14] C Rogers and D Cooper, *RASAero II Aerodynamic Analysis and Flight Simulation Program* (Rogers Aeroscience, 2019).

[15] J Barrowman, "The Practical Calculation of The Aerodynamic Characteristics of Slender Finned Vehicles", MA thesis (Catholic University of America, Mar. 1967).

[16] WF Hilton, *High-Speed Aerodynamics*, 1st (Longman, Green and Co., New York, NY, 1951).

[17] R Zucker and O Biblarz, *Fundamentals of Gas Dynamics*, 2nd (John Wiley & Sons, Hoboken, NJ, 2002).

[18] M Zucrow and J Hoffman, *Gas Dynamics: Multidimensional Flow*, Vol. 2 (John Wiley & Sons, 1977).

[19] JD Anderson, *Modern Compressible Flow With Historical Perspective*, 3rd (McGraw-Hill, New York, NY, 2003).

[20] C Syvertson and D Dennis, "A Second-Order Shock-Expansion Method Applicable to Bodies of Revolution Near Zero Lift", NACA Technical Report (1957).

[21] W Pitts, J Nielsen, and G Kaattari, "Lift and center of pressur of wing-body-tail combinations at subsonic, transonic, and supersonic speeds", NACA Technical Report (1959).

[22] L Euler, *Institutionum Calculi Integralis* (Impensis Academiae Imperialis Scientiarum, Petropoli, 1824).

[23] J Butcher, "A history of Runge-Kutta methods", Applied Numerical Mathematics **20**, 247–260 (1996).

[24] C Runge, "Über die numerische Auflösung von Differentialgleichungen", Mathematische Annalen **46**, 167–178 (1895).

[25] W Kutta, "Beitrag zur näherungsweisen Integration totaler Differentialgleichungen", Zeitschrift für Mathematik und Physik **46**, 435–453 (1901).

[26] E Fehlberg, "Classical Fifth-, Sixth-, Seventh-, and Eighth-Order Runge-Kutta Formulas with Stepsize Control", NASA Technical Report **287**, 89 (1968).

[27] E Fehlberg, "Low-Order Classical Runge-Kutta Formulas with Stepsize Control and Their Application to Some Heat Transfer Problems", NASA Technical Report **315**, 49 (1969).

[28] W Press, S Teukolsky, W Vetterling, and B Flannery, *Numerical Recipes in C: The Art of Scientific Computing*, Second (Press Syndicate of the University of Cambridge, 40 West 20th Street, New York, NY, 2002).

[29] W Peng, Q Zhang, T Yang, and Z Feng, "A High-Precision Dynamic Model of a Sounding Rocket and Rapid Wind Compensation Method Research", Advances in Mechanical Engineering **9**, Publisher: SAGE Publications (2017).

[30] B Graf, "Quaternions and dynamics", arXiv:0811.2889 [math-ph] (2008).

[31] MA Unseren, "Derivation of Three Closed Loop Kinematic Velocity Models Using Normalized Quaternion Feedback for an Autonomous Redundant Manipulator with Application to Inverse Kinematics", Oak Ridge National Laboratory (1993).

[32] JL Blanco, *A Tutorial on SE(3) Transformation Parameterizations and On-Manifold Optimization* (Universidad De Malaga, Sept. 2013).

[33] JM Cooke, MJ Zyda, DR Pratt, and RB McGhee, "NPSNET: Flight Simulation Dynamic Modeling Using Quaternions", Presence: Teleoperators and Virtual Environments **1**, 404–420 (1992).

[34] J Pearson, H Rugge, J Ellis, A Hull, R Quiroz, T Shimazaki, T Van Zandt, P Mange, M Dubin, J Nisbet, L Jacchia, J Slowey, E Batten, G Schilling, F Bartman, L Jones, A Nier, and B Tinsley, "U.S. Standard Atmosphere, 1976", NASA Technical Memorandum **74335** (1976).

[35] G McDonough, W Murphree, J Blair, J Scoggins, T Reed, E Linsley, V Verderaime, J Lovingood, M Rheinfurth, and R Ryan, *Wind Effects on Launch Vehicles* (Advisory Group for Aerospace Research and Development of NATO, Feb. 1970).

[36] P White, "Earth Global Reference Atmospheric Model - 2016", NASA/MSFC Natural Environments (2016).

[37] C Justus, F Alyea, D Cunnold, W Jeffries III, and D Johnson, "The NASA/MSFC Global Referance Atmospheric Model - 1990 (GRAM-90)", NASA Technical Memorandum **4268** (1991).

[38] C Justus, W Jeffries III, S Yung, and D Johnson, "The NASA/MSFC Global Reference Atmospheric Model - 1995 (GRAM-95)", NASA Technical Memorandum **4715** (1995).

[39] C Justus and D Johnson, "The NASA/MSFC Global Reference Atmospheric Model - 1999 (GRAM-99)", NASA Technical Memorandum **209630** (1999).

[40] C Justus and F Leslie, "The NASA/MSFC Earth Global Reference Atmospheric Model - 2007", NASA Technical Memorandum **215581** (2008).

[41] F Leslie and C Justus, "The NASA/MSFC Earth Global Reference Atmospheric Model - 2010", NASA Technical Memorandum **216467** (2011).

[42] B McBride and S Gordon, "Computer Program for Calculation of Complex Chemical Equilibrium Compositions and Applications", NASA Reference Publication **1311** (1994).

[43] B McBride and S Gordon, "Computer Program for Calculation of Complex Chemical Equilibrium Compositions and Applications: Users Manual and Program Description", NASA Reference Publication **1311** (1996).

[44] I Sierra Engineering & Software, *Two Dimensional Kinetic (TDK) Code*, 2017.

[45] RPS UG, *Rocket Propulsion Analysis*, 2021.

[46] C Taylor, *RocketCEA 1.1.24*, Jan. 2021.

[47] C Harms, K Gopal, J Schwartz, G Nourot, and C Young, "HALE Thrust Vector Control - Technical Portfolio", (2020).

[48] W Karush, "Minima of Functions of Several Variables with Inequalities as Side Conditions", MA thesis (University of Chicago, Chicago, 1939).

[49] HW Kuhn and AW Tucker, "Nonlinear Programming", in Econometrica (1951), pp. 481–492.

[50] R Cottle, "William Karush and the KKT Theorem", Documenta Mathematica, 255–259 (2012).

[51] AR Short, BL DuPont, and MI Campbell, "A Comparison of Tree Search Methods for Graph Topology Design Problems", in Design Computing and Cognition '18, edited by JS Gero (2019), pp. 75–94.

[52] EW Dijkstra, "A Note on Two Problemsin Connexion with Graphs", Numerische Mathematik **1**, 269–271 (1959).

[53] P Hart, N Nilsson, and B Raphael, "A Formal Basis for the Heuristic Determination of Minimum Cost Paths", IEEE Transactions System Science and Cybernetics **4**, 100–107 (1968).

[54] K Otto and K Wood, *Product Design* (Prentice Hall, Upper Saddle River, N.J, 2001).

[55] SJ Russell and P Norvig, *Artificial Intelligence: A Modern Approach*, Prentice Hall series in artificial intelligence (Prentice Hall, Englewood Cliffs, N.J, 1995).

[56] S Kirkpatrick, CD Gelatt Jr, and MP Vecchi, "Optimization by Simulated Annealing", Science **220**, 671–680 (1983).

[57] JS Arora, *Introduction to Optimum Design*, 3rd ed (Academic Press, Boston, MA, 2011).

[58] E Burovski, M Brett, R Gommers, R Kern, T Oliphant, and R Tyler, *Scipy 1.6.2*, Apr. 2021.

[59] J Kennedy and R Eberhart, "Particle Swarm Optimization", in Proceedings of International Conference on Neural Networks, Vol. 4 (1995), pp. 1942–1948.

[60] LJ Miranda, *Pyswarms 1.3.0*, Apr. 2021.

[61] J Mockus, *Bayesian Approach to Global Optimization: Theory and Applications*, edited by M Hazewinkel, Vol. 37, Mathematics and Its Applications (Springer Netherlands, Dordrecht, 1989).

[62] J Mockus, W Eddy, A Mockus, L Mockus, and G Reklaitis, *Bayesian Heuristic Approach to Discrete and Global Optimization: Algorithms, Visualization, Software, and Applications*, edited by P Pardalos and R Horst, Vol. 17, Nonconvex Optimization and Its Applications (Springer US, Boston, MA, 1997).

[63] M Hoffman, E Brochu, and ND Freitas, "Portfolio Allocation for Bayesian Optimization", in Proceedings of uncertainty in artificial intelligence (2011).

[64] F Nogueira, *Bayesian-Optimization: Open Source Constrained Global Optimization Tool for Python (version 1.2.0)*, June 2020.

[65] JM Hanson and BB Beard, "Applying Monte Carlo Simulation to Launch Vehicle Design and Requirements Analysis", NASA Technical Publication **216447**, 134 (2010).

[66] A Birolini, *Reliability Engineering: Theory and Practice* (Springer Berlin Heidelberg, Berlin, Heidelberg, 2017).

[67] E Fix and J Hodges, *Nonparametric Discrimination Consistency Properties* (USAF School of Aviation Medicine, 1951).

[68] NS Altman, "An Introduction to Kernel and Nearest-Neighbor Nonparametric Regression", The American Statistician **46** (1992).

[69] T Rose, *Liquid Engine Rocket Optimization & Qualification*, Sept. 2021, `10.5281/zenodo.5513451`.