INTERNATIONAL UNIVERSITY LIAISON INDONESIA (IULI)

BACHELOR'S THESIS

POTENTIAL ANALYSIS OF F-16 FOR HOMELAND INTERCEPTOR MISSIONS USING SUAVE

By

Muhammad Rizky Permana
11202101007

Presented to the Faculty of Engineering and Life Sciences
In Partial Fulfillment Of the Requirements for Degree of

BACHELOR OF ENGINEERING
In
AVIATION ENGINEERING

FACULTY OF ENGINEERING AND LIFE SCIENCES

Associate Tower Intermark 15310
Indonesia
February 2025

APPROVAL PAGE

POTENTIAL ANALYSIS OF F-16 FOR HOMELAND INTERCEPTOR MISSIONS USING SUAVE

MUHAMMAD RIZKY PERMANA
11202101007
Presented to the Faculty of Engineering
In Partial Fulfillment of the Requirements for the Degree of

BACHELOR OF ENGINEERING
In
AVIATION ENGINEERING

FACULTY OF ENGINEERING AND LIFE SCIENCES

Dr. Eng. Ressa Octavianty	____________________
Thesis Advisor	Date (MM/DD/YYYY)
Triwanto Simanjuntak, PhD	____________________
Thesis Co-Advisor	Date (MM/DD/YYYY)
Dipl.-Ing. Sentot Wahjoe Georitno, M.Si.	____________________
Dean of Faculty of Engineering and Life Sciences	Date (MM/DD/YYYY)

EXAMINERS APPROVAL PAGE

M. Ilham Adhynugraha, PhD	____________________
Examiner 1	Date (MM/DD/YYYY)

Dewi Habsari Budiarti, PhD	____________________
Examiner 2	Date (MM/DD/YYYY)

STATEMENT BY THE AUTHOR

I hereby declare that this submission is my own work and to the best of my knowledge, it contains no material previously published or written by another person, nor material which to a substantial extent has been accepted for the award of any other degree or diploma at any educational institution, except where due acknowledgement is made in the thesis.

Muhammad Rizky Permana
Student

02/18/2025
Date (MM/DD/YYYY)

ABSTRACT

Potential Analysis of F-16 for Homeland Interceptor Missions Using SUAVE

by

Muhammad Rizky Permana

Dr. Eng. Ressa Octavianty, Advisor

Triwanto Simanjuntak, PhD, Co-Advisor

Using Stanford University Aerospace Vehicle Environment (SUAVE), this thesis analyzes the feasibility F-16 Fighting Falcon for Homeland Defense Interceptor (HDI) missions. With an emphasis on Defensive Counter-Air (DCA), Point Defense Intercept (PDI), and Intercept/Escort (IE) mission profiles, the study evaluates the aircraft's performance. No modifications to the avionics, propulsion, and aerodynamic systems except its weaponry are investigated to satisfy AIAA requirements. The findings of SUAVE simulations show that although F-16 satisfies the requirements for PDI and IE missions, its endurance capabilities make aircraft unsuitable for DCA mission. In particular, without more fuel-efficient and operational range upgrades, F-16 can not sustain the necessary patrol time for DCA (Defensive Counter-Air) mission, due to the aircraft do not have anymore usable fuel on-board, based on the simulation. This analysis emphasizes a strategic balance between increasing operational capabilities and resource efficiency, highlighting both the possibilities and difficulties of converting current multi-role aircraft to specialized interceptor duties.

Keyword: *Homeland Defense Interceptor, SUAVE, OpenVSP, F-16*

ACKNOWLEDGEMENTS

I extend my heartfelt gratitude to all those who provided guidance and support throughout the completion of this thesis. Foremost, I express my deepest appreciation to Allah SWT for bestowing countless blessings upon me, granting health, resilience, and wisdom throughout the duration of this project.

Immense gratitude is directed towards my thesis advisor and co-advisor, Dr. Triwanto Simanjuntak and Dr. Eng. Ressa Octavianty, for their unwavering mentorship, support, and motivation. Their profound insights, suggestions, and expertise significantly refined both my personal and academic endeavors. Additionally, I want to say thank you to my examiners, M. Ilham Adhynugraha, PhD and Dewi Habsari Budiarti, PhD, for being my examiners and providing valuable feedback on my thesis manuscript.

I am immensely grateful to my friends in Taiwan—Lukman, Kevin, Irvan, Talal, Bimo, Fabian, Ryan, and Sandy. Special thanks to my seniors, Samudra and Ali, for their invaluable guidance. Heartfelt appreciation goes to Akbar, who joined me for many late nights and meals, providing essential companionship and academic support. And to my best buddy Akang (Samuel)—though miles apart, your readiness to answer my calls and tackle questions was a game changer.

I would also like to acknowledge someone from my past who, though no longer part of my journey, provided invaluable life lessons that have profoundly shaped my personal development. The wisdom gained from our time together continues to influence my perspectives and resilience.

My family deserves special thanks, especially my parents, for their enduring love and support, and my cats, Micu and Rosita, for their comforting presence. These regular video calls from Abu Dhabi to Taiwan have been a source of immense joy and comfort.

This accomplishment could not have been achieved without the collective support and faith of everyone involved. I am profoundly thankful to each individual who played a role in this journey. Thank you all for your outstanding and invaluable support and contributions.

Contents

List of Figures

List of Tables

List of Abbreviations

HDI	**H**omeland **D**efense **I**nterceptor
OPV	**O**ptionally **P**iloted **V**ehicle
UAV	**U**nmanned **A**erial **V**ehicle
MTOW	**M**aximum **T**ake-**O**ff **W**eight
UCAV	**U**nmanned **C**ombat **A**erial **V**ehicle
J-UCAS	**J**oint **U**nmanned **C**ombat **A**ir **S**ystem
INS	**I**nertial **N**avigation **S**ystems
GPS	**G**lobal **P**ositioning **S**ystems
EW	**E**lectronic **W**arfare
AI	**A**rtificial **I**ntelligence
C2	**C**ommand and **C**ontrol
DCA	**D**efensive **C**ounter **A**ir Patrol
PDI	**P**oint **D**efense **I**nterception
IE	**I**ntercept/**E**scort
NMi	**N**autical **M**iles
SEP	**S**pecific **E**xcess **P**ower
ECM	**E**lectronic **C**ounter **M**easures
BVR	**B**eyond **V**isual **R**ange
QRA	**Q**uick **R**eaction **A**lert
EVTOL	**E**lectronic **V**ertical **T**akeoff **L**anding
SUAVE	**S**tandford **U**niversity **A**erospace **V**ehicle **E**nvironment
NDARC	**N**ASA **D**esign and **A**nalysis of **R**otorcraft
GTOW	**G**ross **T**ake-**O**ff **W**eight
RPAS	**R**emotely **P**iloted **A**ircraft **S**ystems
UAM	**U**rban **A**ir **M**obility
MRW	**M**aximum **R**amp **W**eight
MCI	**M**issoin **C**overage **I**ndex

MDO	**M**ultidisciplinary **D**esign **O**ptimization
DOC	**D**irect **O**perating **C**ost
AIAA	**A**merican **I**nstitute of **A**eronautics and **A**stronautics
OpenVSP	**O**pen **V**isual **S**ketch **P**ad
FSAT	**F**ull **S**cale **A**erial **T**arget
LGPL	**L**esser **G**eneral **P**ublic **L**icense
RAM	**R**apid **A**ircraft **M**odeler
SGI	Silicon **G**raph**i**cs

Dedicated to my parents

CHAPTER 1
INTRODUCTION

1.1 Background

The growth of advanced offensive aircraft technology and the changing nature of global threats have increased the strategic and tactical demands on air defense systems. In order to effectively combat high-speed airborne threats and maintain national security, the American Institute of Aeronautics and Astronautics (AIAA) [1] has recognized the urgent need for a new Homeland Defense Interceptor (HDI). Among the aircraft being considered for such improvements is F-16 Fighting Falcon, as seen in Figure 1.1, which is renowned for its agility, reliability, and long operating history in many international air forces.

FIGURE 1.1: F-16 Fighting Falcon [2]

Further demonstrating the aircraft's adaptability is the conversion of retired F-16s into unmanned aerial target known as QF-16 [3]. By imitating enemy tactics and confirming the efficiency of air defense tactics without endangering pilot lives, these QF-16 drones play vital roles in defense system testing and training. The program emphasizes F-16's long term usefulness and versatility while highlighting the possibility of additional changes to satisfy AIAA's homeland security requirements.

As shown in Figure 1.2, using modern simulation tools like SUAVE (Stanford University Aerospace Vehicle Environment) [4] and OpenVSP (Open Visual Sketch Pad) [5], this study aims to analyze the feasibility of converting F-16 into an HDI aircraft, guided by the mission profiles and performance standards specified by AIAA. A thorough examination of possible aerodynamic, propulsion and mission profiles required to analyze F-16's capabilities and satisfy the demanding standards of homeland air defense would be made possible by these two tools.

(A) SUAVE [6]

(B) OpenVSP [7]

FIGURE 1.2: Tool utilized in this analysis

1.2 Problem Statement

F-16 is an excellent option to be converted into an HDI due to its numerous uses and flexibility. However, it takes serious consideration to convert an aircraft that was originally built for multi-role operations into a specialized interceptors. Optimizing airframe and propulsion systems for superior high-altitude performance, improving radar systems for better target acquisition, and integrating new avionics are some of the challenges. Furthermore, there is a lot of interest in re-using jet aircraft that are already in service, in order to maximize expenditure on defense and extend their operational lifespan.

This study is to investigate the possibility of converting F-16 into an HDI using advanced simulation tools like SUAVE and OpenVSP guided by the mission profiles specified by AIAA. This tool will make it possible to thoroughly analyze the avionics integrations, propulsion improvements, and aerodynamic changes that could be required to analyze F-16's capabilities and satisfy the demanding standards of homeland air defense.

1.3 Research Objectives

This study aims to evaluate F-16's capability as an HDI focusing on:

- Performance in performing three mission profiles given by AIAA:
    1. Defensive Counter-Air Patrol Mission (DCA).
    2. Point Defensive Intercept (PDI).
    3. Intercept/Escort (IE).

1.4 Research Scope and Limitation

The research scope and limitations are:

- Evaluating performance metrics such as fuel burnt, range of F-16 based on mission profiles given by AIAA.
- OpenVSP is used to evaluate F-16 model constructed in SUAVE framework.
- SUAVE can not accommodate two-wing configurations, as shown in Figure 1.3.
- Absence of takeoff & landing segments.
- Simplified climb modeling with rate of climb (ROC), and thrust modeling.
- All mission phases/segments are assumed in trim condition.

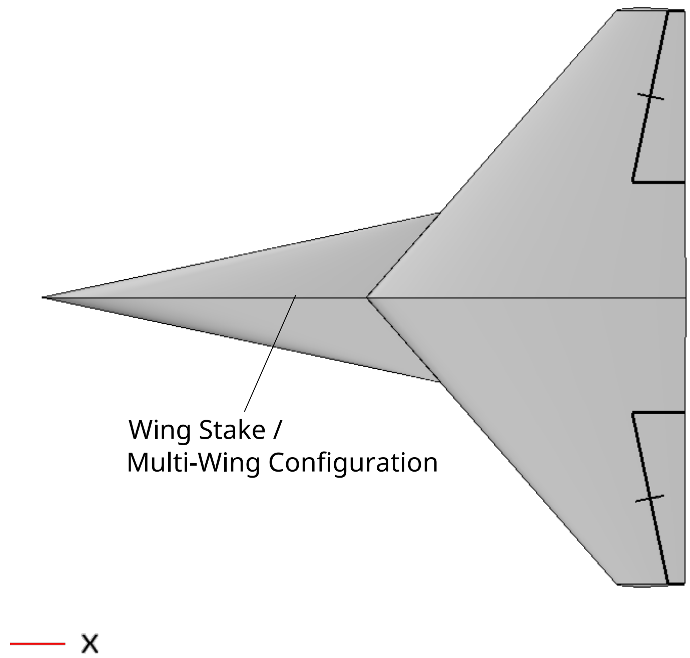


FIGURE 1.3: Wing stake or multi-wing configuration.

1.5 Significance of the Study

The results of this research are expected:

- Evaluates F-16 as possible platform for HDI development, which has an impact on air defense strategy.
- Helps engineer to update interceptor aircrafts by effectively utilizing current aircraft and technologies.
- Examines SUAVE's potential as a flexible tool for upcoming aerospace engineering projects, emphasizing how well it can simulate and analyze complex aircraft systems in theoretical frameworks.

CHAPTER 2
LITERATURE REVIEW

2.1 Homeland Defense Interceptor (HDI)

Homeland defense interceptors by definition is an fighter aircraft designed specifically to intercept and destroy enemy aircraft, particularly bombers, before they can reach their target, and has primary purpose of homeland defense interceptors is to intercept and take out enemy aircraft. These fighters have a compact design that improves performance and keeps operating costs low, making them ideal for close-range combat. Homeland defense interceptors are a more affordable option than heavier fighter aircraft like F-22 or F-35, which are primarily designed for air-to-air combat and come with greater cost and maintenance. They are vital components of national defense plans because of their armament and design, which are suited to successfully engage and defeat or dissuade invading fighters.

HDI's unique weapons and designs are suited effectively in order to counter and prevent enemies aircraft, making it crucial aircraft of the nation's defense program. In the past, HDI or interceptor aircraft like F-106 Delta Dart Mig-25 Foxbat and Tornado ADV, as illustrated in Figures 2.1a, 2.1b, 2.6a respectively, were known for their dash capabilities and advanced avionics that allows them to close up enemies at far distances. However, in modern tactics, a lot of countries are trying to make multi-purpose fighters jets to perform both offensive and defensive missions.

(A) F-106 Delta Dart [8]   (B) Tornado ADV [9]

Basic fundamental aircraft design of HDI emphasis more on rapid climbing, dash, supersonic capabilities, and quick response times. These traits enable the aircraft to effectively manage time-critical threats, which enables HDI as one of key aircraft for national defense plans. Facilitate with lighter airframes body and technical weaponry, HDI are proficient at engaging enemies within close to medium ranges. HDI commonly carry short to medium range air-to-air missiles and are supported by advanced sensory systems and avionics for quick targeting and engagement.

Integrated with powerful ground-based radar systems, command-and-control units, and early warning networks. HDI can quickly responds to unidentified or hostile aircrafts, making sure the security of the airspace with minimum time of reaction.

The increased interest in determining whether current existed multi-role fighters are effective is supported by this changing threat situation, such as the F-16s which could be adapted as HDI aircrat. If its practicable, re-purposing or re-using the aircraft to be HDI, it could representing new strategically balanced, optimizing capability, quickness, and budget compensation in modern defense infrastructure.

2.1.1 History of Interceptor Aircraft

From the Cold War to the Modern War era, interceptors have been developed and manufactured to meet evolving air defense demands. The American-developed Northrop F-89 Scorpion, which one of the most famous interceptors from the Cold War. One of the first jet-powered aircraft of its type, this all-weather interceptor was outfitted with air-to-air missiles and unguided rockets, which improved its capacity to successfully counter enemy threats. Below figure is the picture of F-89 Scorpion as shown in Figure 2.2a.

The Swedish Saab 37 Viggen is another outstanding interceptor from this era. In addition to being an interceptor, this multi-role aircraft was intended to be a flexible combat aircraft. High-speed interception capability, short runway operation, and a homeland defense-specific design are some of its key characteristics. During the Cold War, the Viggen was essential in protecting Sweden's airspace, highlighting the strategic value of interceptors. Below figure is the pictures of Saab 37 Viggen as shown in Figure 2.2b.

(A) F-89 Scorpion [10]

(B) Saab 37 Viggen [11]

FIGURE 2.2: Cold War Era Interceptors

2.1.2 Next-Gen Interceptor Aircraft

Interceptor aircraft have been significantly enhanced in many aspects in the modern era, and several nations are working to create the *state-of-the-art* interceptors, both manned and unmanned. One of the newer possibilities is the Optional Piloted Vehicle (OPV), which offers deployment flexibility by acting as a drone or a Unmanned Aerial Vehicle (UAV).

The Lockheed Martin F-22 Raptor is one of the most well-known American interceptors. This aircraft excels at interception and may be used as both an air superiority fighter and a multirole fighter. Stealth technology, supercruise capability (which enables sustained supersonic flight without the need for afterburners), an advanced radar system, and long-range missile weapons intended for air-to-air combat are some of the F-22 Raptor's primary characteristics. The F-22 is a powerful tool in contemporary air defense operations because of these characteristics. Below figure is the picture of Lockheed Martin F-22 Raptor as shown in Figure 2.3.

FIGURE 2.3: Lockheed Martin F-22 Raptor [12]

The safety of fighter pilots is a primary concern and a pilot's limitation in the current aerospace environment. As a result, engineers are developing technologies like Optionally Piloted Vehicles (OPV) to advance interceptor technology. This technology increases mission flexibility by enabling aircraft to be operated remotely by a pilot on the ground or by a pilot on board.

The Northrop Grumman Firebird is a well-known example of OPV technology in use. While not intended as an interceptor, the Firebird demonstrates the capabilities of OPV systems. The fact that this aircraft can be flown both manned and unmanned shows how unmanned technologies are becoming more and more popular in aviation. The use of OPV systems in interceptors is anticipated to grow as they develop, opening the door for more self-sufficient air defense capabilities. Below figure is the picture of Northrop Grumman Firebird as shown in Figure 2.4a.

There are currently no unmanned aerial vehicles (UAVs) specifically designed for the interceptor function, despite the fact that engineers have made significant advances in unmanned aircraft technology. Nonetheless, a number of UAVs with interception capabilities are being developed and tested. The Northrop Grumman MQ-9 Reaper, which was initially intended as a surveillance and strike drone, is a significant example. In order to investigate the MQ-9's potential for interception missions, testing have equipped it with air-to-air missiles. The testing demonstrates the increasing adaptability of UAVs in contemporary air defense tactics, even though its main purpose is still surveillance and strike operations. Below figure is the picture of General Atomics MQ-9 Reaper 2.4b.

(A) Northrop Grumman Firebird [13] (B) General Atomics MQ-9 Reaper [14]

FIGURE 2.4: Unmanned Interceptor Aircraft

2.2 Necessity for Homeland Defense Interceptors

There is a serious risk of aerial attacks on the United States given the changing global security environment, which is characterized by rising political tensions between nations. A diverse range of threats, from small autonomous cruise missiles to huge hijacked aircraft, could be involved in these strikes. A major obstacles to protecting national airspace authority is the projected 2045 end of service life for the majority of current Air Force and Navy Combat Aircraft. Although these jet fighters are notable at overcoming high-performance aircraft and complex air defenses, the sophisticated, stealth-capable F-22 and F-35 fighters are not cheap and cannot be purchased in large enough quantities to guarantee homeland defense and comprehensive force projection. The creation of a customized Homeland Defense Interceptor (HDI) is required in this situation [1].

Interceptors aircraft are designed specifically to intercept and destroy enemy aircraft, particulary bombers, before they can reach their target. Since, interceptors are basically a jet fighter aircraft, interceptors aircraft are also designed with armament and technological features that will help these aircrafts is suited and prepared for domestic homeland defense.

Interceptors aircraft are already been established since Cold War Era, the trends continues up to now, where conventional jet aircraft is combine with UAV technology. However, for now there are no drones or UAVs are currently dedicated solely to the interceptor role, several aircrafts are being developed or even tested with air-to-air combat and interception capabilities, but these aircrafts are often multi-role or support system, nonetheless these aircrafts have all the potential for future interceptor missions.

Understanding the interceptor aircraft design and mission requires comparing its features to those of other aircraft types often employed in military aviation. These consists of fighters, air superiority fighters, bombers, multirole aircraft, and interceptors. There are significant differences in responsibilities performance goals, and mission profiles, despite certain capabilities being comparable across all categories. The following table provide a thorough comparison to emphasize the key features and primary role of each aircraft class, setting the stage for a more thorough examination of interceptor aircraft designs as presented in Table 2.1 & 2.2.

Interceptor vs Fighter vs Multirole vs Air-Superiority vs Bomber			
Aircraft Type	Primary Role/ Focus	Key Features/Capabilities	Examples
Interceptor	Intercept and destroy enemy aircraft (bombers, threats) before they reach their target.	• High speed and climb rate • Long-range radar and missiles • Quick reaction time	• MiG-25 "Foxbat" • F-106 Delta Dart • MiG-31 Foxhound
Fighter	Engage enemy aircraft in direct combat (air-to-air engagements).	• High maneuverability and speed • Focus on dogfight and air-to-air combat	• F-16 Fighting Falcon • MiG-29 Fulcrum
Multirole	Perform multiple roles: • Air-to-Air missions • Air-to-Ground mission	• Verstaile which capable of both air-to-air combat and ground strikes • Can be adapted to various mission profiles	• F/A-18 Hornet • Dassault Rafale • F-35 Lightning II

Table 2.1: Comparison of Aircraft Types

Interceptor vs Fighter vs Multirole vs Air-Superiority vs Bomber			
Aircraft Type	Primary Role/ Focus	Key Features/Capabilities	Examples
Air-Superiority	Achieve and maintain dominance in airspace by defeating enemy aircraft.	• Extended range, advanced radar, and beyond visual range (BVR) • Designed for sustained air-to-air combat	• F-22 Raptor • Su-35 Flanker-E • Eurofighter Typhoon
Bomber	Deliver large quantities of explosives to ground targets (strategic or tactical missions).	• Heavy payload capability • Long-range flight capability • Typically slower and less maneuverable than fighters	• B-52 Stratofortress • Tu-160 Blackjack • B-2 Spirit

Table 2.2: Comparison of Aircraft Types

The future trends of these interceptor aircraft are currently dedicated to solely to the interception, the development of autonomous and optionally piloted vehicle (OPV) systems for air-to-air combat and interception is actively being researched. As UAVs technology advances, specialized interception drones may become a reality in the future.

There are several must have features for an interceptor aircraft. Interceptor aircraft are designed and specialized to detect, intercept, and destroy enemy aircraft before they reach their target, below are the essential features required for an effective interceptor.

1. High Speed and Climb Rate: This features allows the interceptors to quickly reach enemy aircraft, often covering long distances in a short amount of time, while high climb rate ensures it can ascend rapidly to intercept high-altitude targets, where MiG-25 ”Foxbat” and English Electric Lightning are the best example of this features.
2. Powerful Radar and Sensor Systems: Advanced radar and sensors are essential to detect and track enemy aircraft at long ranges, especially in difficult conditions such as night or poor weather, and other radar or sensor systems, as outlined in Section 4.
3. **Long-Range Air-to-Air Missiles**: Interceptors engage targets from long distances, thus making long-range air-to-air missiles allow them to strike enemy aircraft before getting too close, which has been discussed in Section 3(e)i.
4. **High Operational Ceiling**: Interceptors need a high operational ceiling to engage enemy aircraft, such as bombers, which often fly at high altitudes, where MiG-25 is designed to fly at altitudes above 20,000 meters (65,000 feet).
5. **Quick Reaction Time**: Interceptors must scramble and reach enemy aircraft with minimal delay , quick time is crucial in surprise attack or intrusion scenarios, where Eurofighter Typhoon used for quick reaction alert (QRA) missions.
6. **All-Weather and Day/Night Capabilities**: Interceptors must operate effectively in all weather conditions and at any time of day to ensure continuous airspace protection, where F-106 Delta Dart and MiG-31 are the best examples for this features.
7. **High Maneuverability (Optional)**: Interceptors may need to engage in dogfights or evade enemy missiles, making high maneuverability is important, though it is less critical for Beyond Visual Range (BVR) engagements, which is the ability where pilot can see their enemy with the naked eye, examples aircraft that have this features are F-15 Eagle and Su-35
8. **Extended Range or In-Flight Refueling**: Interceptors need sufficient range to cover large areas or have the ability to refuel mid-air to extend their operational reach, examples aircraft that is utilizing this features are F-22 Raptor and MiG-31.

9. **Stealth (Modern Interceptors)**: Stealth allows interceptors to approach enemy aircraft undetected, reducing the chances of being intercepted or avoided, one of exemplary that utilizing this technology is F-22 Raptor.
10. **Electronic Warfare (EW) Systems**: This systems help interceptors jam enemy radar and communications, protect themselves from missiles, and enhance survivability. Almost every modern interceptors are equipped with electronic countermeasures (ECM).

2.2.1 Challenges in Designing HDI

Designing HDI for this case is little bit tricky and faces unique challenges, since its going need balancing every aspects like, speed, agility, and even combat readiness, for all three missions profiles given in Section 2.7.1, these aspects and challenges include:

1. **Aerodynamics and High Speed Performance**
   (a) Main objective of this aspect is to reduce drag and increase lift across a variety of speed ranges, especially transonic and supersonic ranges, interceptors need to have an aerodynamically efficient design. All three missions profiles require tremendous speeds, making minimizing drag and preserving stability are important for this interceptors aircraft.
   (b) **Critical Aspects**
      i. **Wing Design**:
         A. **Swept Wings or Delta Wings**: At high speeds these two configurations play crucial role in order to get lower drag. **Delta Wing** improves stability and performance in supersonic flight conditions, while **Swept Wing** delays the onset of shock waves and smoothes airflow during transonic flight conditions. Several examples of interceptors aircraft with variable sweep wings are Mikoyan MiG-27 which an attack variant of the MiG-23 with variable-sweep wings, which designed for ground attack missions, another example is Boeing X-45, which an experimental Unamnned Combat Aerial Vehicle (UCAV) that features variable sweep wings, developed as part of the Joint Unmanned

Combat Air Systems (J-UCAS) program, example aircraft with sweep wing are Illustrated in Figure 2.5.

(A) Boeing X-45 [15]

(B) Mikoyan MiG-27 [16]

FIGURE 2.5: Aircraft Examples of Variable Sweep Wings

   B. **Variable Geometry (Swing Wings)**: This configurations is way complex compare to other configurations, but despite its complexity this configuration maximize lift and reduce drag at subsonic and supersonic speeds by allowing the wing to adapt throughout different speed regimes.

ii. **Area Rule (Coke-Bottle Design)**: This method is essential for minimizing wave drag at transonic speeds by establishing seamless cross-sectional transitions across the fuselage. It is very useful for controlling drag when reaching supersonic speeds.

iii. **Leading Edge and Control Surfaces**:

   A. **Thin, Sharp Leading Edges**: These kind of design of leading edges helps lowering wave drag during supersonic flight, which is esssential for interceptors that must perform effectively at high speed conditions.

   B. **Optimized Control Surfaces**: To maintain versatility at supersonic speeds without compromising aerodynamic efficiecy, high-speed stability needs precise optimized control surfaces.

(c) Since interceptors are designed to quickly and frequently reach and engage enemy targets over long distances, high-speed performance and rapid climb rates are critical. Strong climb rates ensure the interceptors can quickly reach high-altitude threats, while high-speed capabilities, especially at supersonic speeds, enable them to close to targets efficiently.

  i. **High-powered engines**: Interceptors are typically equipped with powerful engines that provide the thrust required for high-speed interception, allowing them to achieve supersonic dash and rapid climbs.

  ii. **Aerodynamic Shape Optimization**: This optimization is crucial for seamless transition from subsonic to transonic and supersonic flight regime. Interceptors can continue to operate efficiently during these flight regime changes by focusing on drag reduction techniques, such as area rules and shockwave management, as previously discussed in Section 1(b)i.

  iii. **Examples**:

    A. **MiG-25 "Foxbat"** is known for its exceptional speed and climb rate, which reflects its emphasis on intercepting targets at high speeds and altitudes, Mig-25 Foxbat is shown in Figure 2.6a.

    B. **English Electric Lightning** is designed to quickly intercept enemy aircraft in high-altitude combat, this interceptor is another iconic example of an interceptor aircraft with superior speed and climb capabilities, English Electric Lightning is shown in Figure 2.6b.

(A) MiG-25 "Foxbat" [17] (B) English Electric Lightning [18]

FIGURE 2.6: Examples of high-speed interceptors

2. **Weight Management and Structural Integrity**: Optimizing weight and structural integrity is crucial in designing high-performance supersonic interceptor aircraft. This includes aspects such as thermal control, material selection, and the impact of folding wings on carrier-based combat, as seen in Figure 2.7.

FIGURE 2.7: F-4 Folded Wing [19]

   (a) **Material Selection for Structural Integrity**:
      i. The main objective is to tolerate the extreme heat, pressure, and strain of supersonic flight.
      ii. The materials that are used for developing supersonic interceptor aircrafts are those material that can withstand high-temperature-resistant such as titanium, carbon composites, and aluminum alloys. These materials has a characteristics of strength, heat resistance,

and weight reduction, which are essential for overall functionality and structural integrity.

iii. Last constraint is heat management, in order to sustain heat from high-velocity aerodynamic on vital component, manufacturers apply specialized thermal coating or shield to protect the aircraft.

(b) **Foldable Wings** Foldable wings are mostly used on carrier-based aircraft to help save space, for example is F-4 as represented in Figure 2.7. However, these folded wings have several weight and structural disadvantages:

i. **Structural Complexity and Weight Increase** : The additional parts needed for folding wings such as hinges, hydraulic/electrical systems, and locking mechanism, result in a significant weight increase. THis additional weight has an impact on overall aircraft's performance such as aircraft speed, fuel efficiency, and maneuverability, particularly while cruising at supersonic flight regime.

ii. **Reduced Structural Intergrity**: In comparison to fixed wings, the hing and folding mechanism always damage the wing's structural integrity, which makes the aircraft more vulnerable to stress and strain, particularly during high-G maneuvers, requiring the addition of weight and more reinforcement.

iii. **Maintenance Requirements** Is more expensive and necessary due to mechanism of folding wings that includes moving parts that are more vulnerable to wear and damage. Regular inspection and maintenance are needed to guarantee dependable folding mechanism performance have an impact on operational preparation.

iv. **Risk of Mechanical Failure**: The folding mechanism has a significant potential of experiencing mechanical failure. Insufficient wing locking after deployment can result in fatal in-flight failure.

(c) **Aerodynamic and Load-Bearing**

i. **Aerodynamic Penalties**: Gaps or irregularities created by hinges and folding mechanism on the airfoil can increase drag and decrease aerodynamic efficiency, particularly at high speed regime.

ii. **Complex Wing Load Distribution**: The aerodynamic load distribution on the wing is altered by the presence of hinge points. Because weight distribution must be carefully managed to avoid restricting maximum G-force capabilities–which are essential for durability and maneuverabilityl, which are needed for interceptor aircrafts.

iii. **Impact on Weapon Systems** : The placement of weapon hardpoints may be constrained by the addition of foldable mechanism. This could restrict the aircraft's payload capacity or the kinds of armaments it can carry, as well as the amount of room it has for attaching external weapons.

3. **Engine Performance and Afterburners**: THe engine and propulsion system of interceptor aircrafts are essential for achieving the required mission performance under supersonic flight regime. Under a variety of operating situations, these aircrafts engines need to provide high thrust while remaining efficient and flexible.

(a) **Key features of the Engine System**

i. **Thrust for Supersonic Flight**:

A. The engines must produce enough thrust to achieve and maintain supersonic speeds, thus the interceptors can maneuver and intercept quickly in combat situations.

B. During crucial stages like takeoff, climb, and combat maneuvers, the engines' afterburners significantly increase thrust by injecting additional fuel into the exhaust system

ii. **Afterburners for Peak Performance**:

A. Interceptors can execute challenging combat maneuvers and high-speed dashes, like the 200 nautical miles dashes required for point defense interception missions, due to its afterburners.

B. The use of afterburners are strictly controlled, due to its massive fuel consumption, which indicates how crucial effective fuel management and mission planning.

iii. **High Thrust-to-Weight Ratio**

A. When executing combat maneuvers like 5-g sustained rotation, the engine system's high thrust-to-weight ratio enables it to climb freely, accelerate swiftly and support huge loads.

iv. **Variable-Geometry Intakes**

A. Variable geometry inlets are utilized to optimize engine performance at different speeds, from subsonic cruising to supersonic dashes. These intake ducts continuously modify their shape to give the engine the best airflow possible, which increasing efficiency and sustaining thrust at all speeds.

(b) **Balancing Thrust and Drag** The engine system is coupled with interceptors low-drag design to maintain a high thrust-to-weight ratio. Since the significant increase in air resistance at high speeds, this balance is essential for effective supersonic flight. To meet the demands of maneuverability and particular missions, the engines must not only withstand this drag but also supply extra power.

(c) **Fuel Efficiency and Supercruise Capability** Considering their poor fuel efficiency, afterburners are not the ideal option for long haul missions, even though they are necessary for short periods of high-speed performance. Interceptors supercruise capabilities allows it to maintian supersonic flight without the ned for afterburners, for example when doing mission like four hour combat patrols, this capability gives the aircraft more endurance by improving its combat range and fuel efficiency.

4. **Advanced Avionics and Targeting Systems** Interceptors aircraft's avionics and targeting systems, which are built to deliver precise navigation, threat identification and attack capabilities in a range of operating situations are an essential component of a successful mission. These *state-of-the-art* systems guarantee superior situational awareness and quick reaction to changing threats, allowing the interceptor to operate with maximum efficiency, precision, and survivability. Threats can be immediately neutralized in a variety of combat situations because to their integration, which also offers strong defense capabilities.

(a) **High Speed Radar System**

i. Modern radar that can detect and follow numerous targets over great distances is part of the high speed radar system.
ii. The ability to provide real time updates on target locations in order to quickly analyze and react to threats.
iii. Low latency supports low-visibility and all-weather operations.

(b) **Intertial Navigation Systems (INS) and Global Positioning System (GPS) Integration**

i. This system combines GPS and inertial navigation systems to provide precise positioning and navigation, which is essential for intercepting missions.
ii. Provide dependable navigation information even in locations lacking GPS to guarantee mission continuity.

(c) **Electronic Warfare (EW) Capabiliteis**

i. Preventive defense against threat is achieved through integrated EW systems.
ii. It is capable of facing off electronic attacks due to its spoofing defection and jamming resistance.
iii. It helps identify hostile radar or communication signals.

(d) **AI-Assisted Threat Detection and Targeting**

i. This technology automatically classifies potential threats by analyzing sensors data in real time using artificial intelligence (AI).
ii. Providing the ideal intercept trajectories and engagement strategies it makes target selection easier.
iii. Pilots can more focused on combat tactics, and it reduce the workload of pilots

(e) **Onboard and Offboard Sensor Fusion**

i. Combining sensors from onboard with offboard intelligence such as data from airborne sensors like radar, infrared, and electro-optical systems.
ii. Provides the Command and Control (C2) Center a comprehensive situational picture and permits effective targeting and strike decisions even in intricate tactical situations, A diagram of a Command

and Control (C2) center can be found in Figure 2.8.

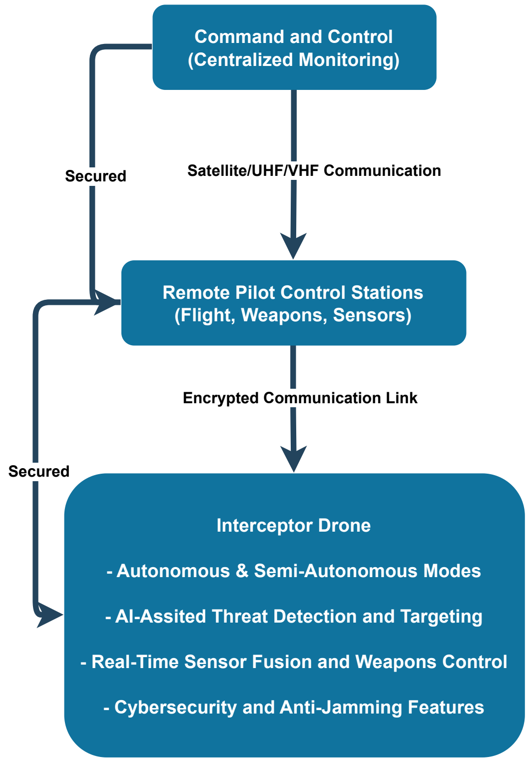


FIGURE 2.8: Command and Control diagram

(f) **Autonomous Target Engagement Support**

i. The system is designed to easily switch between manual and semi-manual modes to ensure combat variety.

ii. If contact with the aircraft is lost, the drone can still strike targets independently according to predetermined priorities.

2.3 Approach to Developing HDI Capabilities

Defense organizations typically three primary approaches for designing a new HDI aircraft, capabilities within the larger framework of force modernization; each has its own benefits, difficulties, and strategic consequences.

Re-Using an Existing Aircraft

This approach requires choosing fighter aircraft that are already in service and re-using or re-purposing into interceptor aircraft without any modifications on the aircraft, except weaponry.

- Advantages:
  - Rapid Integration: By avoiding the lengthy research and testing stages involved with new designs, using existing or aircraft that already in service speeds up the deployment process.
  - Cost Savings: Utilizing current resources reduces the requirement for substantial upfront investment in new platforms. Due to the RFP's tight budgetary constraints, this aircraft will only be constructed if it is incredibly cheap.
  - Leveraging Training and Logistics: Integration into present operations is made easier by the established training programs and logistical supports that come with existing manned or remotely piloted vehicle (RPV).
- Disadvantages:
  - Performance Limitations: The ideal speed, agility, or endurance needed for HDI missions may not be present in aircraft that were not built for interception.
  - Sub-Optimal System: Upgrades may be necessary because current avionics and weapons systems might not be able to handle the sophisticated demands of specialized interception tasks.

Modifying an Existing Aircraft

This approach requires structural, aerodynamic, or even avionics upgrades, such as enhancing radar capabilities, adding conformal fuel tanks for specific mission profiles that need more range than the normal mission of the current aircraft, reinforcing several airframe points to make the aircraft more durable, in order to fulfill the HDI aircraft effectively.

- Advantages:

  - Tailored Capabilities: By modifying existing platforms, the Air Force can maintain continuity in maintenance and support structures.
  - Retention of Maintenance Infrastructure: The Air Force can preserve continuity in maintenance and support structures by altering current platforms.
- Disadvantages:
  - Upfront Cost: Changes can be expensive, particularly if they require considerable retrofitting and complex technology.
  - Complexities in Design and Certification: Every change needs to be carefully examined and approved, which might cause delays and extra complications.

Sizing and Designing a New Interceptor Aircraft

Based on Raymer Aircraft Design book, sizing is the most important calculation in aircraft design, more so than drag, or stress, or even cost (well, maybe not cost) . Sizing literally determines the size of the aircraft, specifically the weight that the aircraft must be designed to so that it can perform its intended mission carrying its intended payload. An airplane that is too small just cannot carry enough fuel to do its job. How do we know? We know by sizing [20].

- Advantages:
  - Optimized Design: By designing an aircraft especially for HDI missions, engineers can use latest design concepts and technology to optimize efficiency and performance.
  - Incorporation of Advanced Technologies: Modern designs allow for cutting-edge technologies that may not be practical to retrofit into earlier platforms, including as next-generations avionics, sophisticated propulsion systems, and stealth capabilities.
- Disadvantages:
  - High Research and Development Costs: Creating a new aircraft from the ground up may be very expensive, frequently reaching billions of dollars, Thus it will not meet the requirements that is given from AIAA.

  - Longer Timelines: Decades may pass between design and actual deployment.
  - Uncertain Success: Since their effectiveness in real-world combat scenarios has not been shown until extensive testing, new aircraft designs are inherently risky.

Following the budget constraints, re-using existing aircraft that are already in service frequently seems to be the most practical strategy, especially country that already use advanced jet fighters. Given its extensive operational fleet, lengthy service history, and continuous upgrade, F-16 is a strong candidate for the new HDI interceptor aircrat. With possible improvements, this aircraft might fulfill HDI missions' demands for quick response and adaptability without spending the high expenses and time required to create a new aircraft from the ground up.

2.4 Real-World Applications and Efficacy of F-16

Particular incidents that demonstrate F-16's interception capabilities in real world situations illustrate the aircraft's importance in preserving airspace integrity. Notably, Indonesian F-16s were crucial in two important events:

- The 2003 Bawean Incident: Indonesian F-16s successfully intercepted a civilian aircraft over Bawean Island, showcasing the aircraft's operational readiness and rapid response capabilities in practical situations [21].
- The 2019 Ethiopian Airlines Flight 3728 Interception: By stopping and escorting the flight that had entered Indonesian territory without the required authority, this incident further demonstrated F-16's capacity to uphold national airspace sovereignty [22].

These situations demonstrate F-16's potential fit for adoption into HDI roles, enforcing strict compliance to airspace laws and quickly engaging unlawful competitors, in addition to validating its effectiveness in interception roles.

2.5 Re-Using Legacy and Existing Fighters for Interceptor Roles

There are several good aircraft that is worth of exploring and analysis re-using existed fighter jets aircraft to convert into HDI aircraft, such as:

- F-15C Eagle, as shown in Figure 2.9a, utilized by the U.S. Air Force for quick reaction alerts due to its high thrust, long range, and robust radar system.
- F/A-18, as shown in Figure 2.9b, adapted by some Air Forces for defensive combat air patrol missions, relying on its carrier-based design strengths such as strong climb and agility.
- Mirage 2000, as shown in Figure 2.9c, leveraged by several countries for defensive interception owing to its supersonic dash and delta-wing maneuverability which has been discussed in Section 1(b)i.

(A) F-15C Eagle [23]

(B) F/A-18 [24]

(C) Mirage 2000 [25]

The following critical components are necessary for the successful conversion of current aircraft into interceptors:

- Sensors and Avionics Upgrades: Modern interception require modern radar and data-link technologies are necessary for operations in order to sustain efficient situational awareness.
- Engine Performance and Fuel: Supporting the high-speed, high-altitude, and fast turnaround requirements of interceptors operations requires improved engine capability and sufficient fuel reserves.
- Maintenance and Sustainable: The maintenance requirements or airplanes rise with age, potentially affecting their availability for high-readiness tasks.

2.5.1 Focusing on the F-16 for Interceptor Roles

As for F-16 specifically, modernization programs such as the F-16V "Viper" or also referred to as the F-16 Block 70/27, as shown in Figure 2.10, is the latest variant of the F-16 Fighting Falcon fourth-generation multi-role fighter aircraft manufactured by Lockheed Martin. this aircraft integrates advanced capabilities as part of an upgrade package to better inter-operate with fifth-generation fighters, including the F-35 and the F-22. The fighter jet can be deployed in the suppression of enemy air defense missions, air-to-air ground and air-to-air combat, and deep interdiction and maritime interdiction missions. These upgrades or modernization are crucial to ensures and shows that older airframes can fulfill the requirements of HDI mission profiles that are given by AIAA which are Defensive Counter Air (DCA), Interception and Escort (IE), and Point Defense Interception (PDI) [26].

FIGURE 2.10: F-16V "Viper" [27]

Furthermore, F-16 has also been repurposed into **QF-16 Full Scale Aerial Target (FSAT)** as shown in Figure 2.11, innovative program that modernization or re-use retired F-16s into unmanned target drones. These drones are being used for training and testing weapons systems, which providing realistic conditions for live-fire training and combat training without endangering pilot lives, similar RPV aircraft, that enable pilot to stay on ground while controlling the aircraft remotely. This program not only increases the F-16s airframes operational lifespan but also shows how versatile and useful they are for a variety of defense-related [28].

The modernization or re-use of F-16s into target drones has demonstrated air-frame operational lifespan and the innovative approaches to preserve military capabilities while controlling costs. In order to evaluate the preparedness and efficacy of interceptor methods and missile systems, the QF-16 program has gained recognition for its ability to mimic enemy tactics and technologies. But as the program comes to an end, there is a noticeable interest in creating more sophisticated stealthy target drones that might emulate the QF-16 by utilizing newer technologies and taking note of its operating insights [3].

FIGURE 2.11: QF-16 Full Scale Aerial Target (FSAT) [3], [28]

This emphasis on the F-16 in Interception and training roles highlights its adaptability and ongoing worth to air forces around the world, demonstrating that through innovative re-using and technological advancements, older models can stil make a substantial contribution to defense strategies even as they are phased out of front-line service

2.6 AIAA Performance Requirements For Homeland Defense Interceptor

2.6.1 Performance Requirements

The interceptor aircraft for this case has to operate by thorough performance requirements to its defense and interception operations in order to fulfill mission requirements. These standards serve as the foundation for operational planning, design, and sizing.

1. Mission Performance Goals
   (a) Defensive Counter-Air (DCA) Patrol Mission Endurance: The aircraft must be able to maintain a combat air patrol (CAP) within 300 nautical miles (nmi) for 4 hours, as shown in Figure 3.2.
   (b) Point Defense Intercept Mission: This mission requires the aircraft to fly at least 400 nmi, as shown in Figure 3.3.
   (c) Intercept Mission Radius: In this mission, the aircraft must cover a minimum of 200 nmi, as shown in Figure 3.4.
2. Performance at Maneuver Weight (50% Internal Fuel)
   (a) Maximum Mach Number:
      i. The aircraft should achieve a top speed of at least Mach 1.6 at an altitude of 35,000 feet.
   (b) 1-g Specific Excess Power (SEP):
      i. Military Thrust:
         A. At 0.9 Mach/Sea Level: 200 ft/sec
         B. At 0.9 Mach/15,000 feet: 50ft/sec
      ii. Maximum Thrust
         A. At 0.9 Mach/Sea Level: 700 ft/sec
         B. At 0.9 Mach/15,000 feet: 400 ft/sec
   (c) 5-g Specific Excess Power (SEP)
      i. Maximum Thrust
         A. At 0.9 Mach/Sea Level: 300 ft/sec

B. At 0.9 Mach/15,000 feet: 50 ft/sec

(d) Sustained Load Factor

i. The aircraft must sustain a load factor of 5.0 g's at 0.9 Mach and 15,000 feet using maximum thrust.

(e) Maximum Instantaneous Turn Rate

i. Achieve a turn rate 18.0 degrees per second at 35,000 feet.

3. Additional Performance Metrics

(a) Climb Performance

i. The aircraft must climb from sea level to 35,000 feet within 1 Minute, covering a distance of 4.8 Nautical miles.

(b) Take-off and Landing

i. Operate efficiently on standard NATO 8,00-foot runways, ensuring quick response during mission deployment.

(c) Service Ceiling

i. Capable of operating at altitudes exceeding 60,000 feet to intercept high-altitude threats effectively.

(d) Range and Endurance

i. Operational range varies by mission but typically falls within 500-800 nm, balancing speed and endurance for defensive scenarios.

(e) Weapons Integration

i. Support air-to-air missile configuration, including internal and external carriage options, while minimizing drag to maintain performance, the weapons that will be carry are AIM-120 and AMRAAM and M61A1 20 mm Cannon, armament can be seen in Figure 2.12.

(A) AIM-120 AMRAAM [29]

(B) M61A1 20 mm Cannon [30]

(C) AIM-9 [31]

FIGURE 2.12: Air-to-Air Weapon

2.6.2 Potential of the F-16 as a Homeland Defense Interceptor

Analyzing the F-16s potential as HDI aircraft depends on several performance and operational considerations:

Climb Rate and Acceleration

1. Homeland interceptors frequently depend on quick climbs to high altitudes where potential threats may be present. Although it must be verified against more stringent interceptor periods of time, the F-16s single-engine design with a strong thrust-to-weight ratio provides competitive climb performance.

Loiter and Endurance

1. The aircraft could have stay on station for extended periods of time, for DCA and IE operations. Although the F-16 has a moderate internal fuel capacity,

endurance can be increased with additional external tanks or conformal fuel tanks.

Operational Readiness

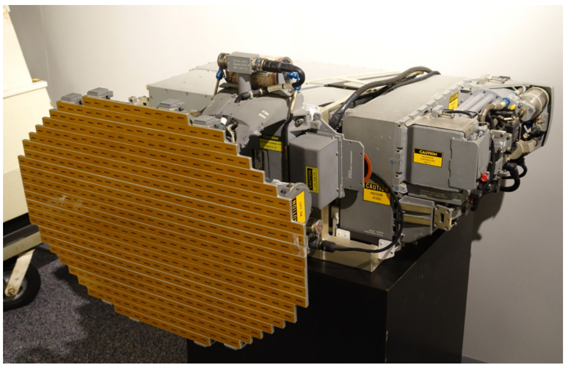
1. High fleet availability is required for quick reaction alerts. Although F-16 has extensive maintenance facility around the world, dependability evaluations must account for the additional wear and tear caused by frequent high-speed dashes.

Ammunition and Payload

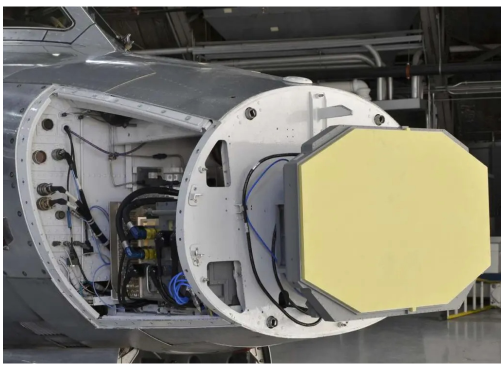
1. The F-16 is capable of carrying air-to-air missiles with short and medium ranges (such as the AIM-9 and AIM-120 can be seen in Figure 2.12c, 2.12a respectively), which are usually enough for interception tasks. Nonetheless, the ideal load-out arrangement for minimizing drag and maximizing agility could not be the same as typical multi-role mission configurations.

Avionics and Detection Capabilities

1. The F-16 can detect, track, and engage airborne threats at ranges appropriate for interception tasks due to advanced radar systems (AN/APG-68, AN/APG-83 AESA Upgrades can be seen in Figure 2.13a, 2.13b respectively). Situational awareness is further improved via data-link integration and electronic support methods.



(A) AN/APG-68 [32]



(B) AN/APG-83 AESA Upgrades [33]

FIGURE 2.13: Radar System

2.7 AIAA Mission Profiles for Homeland Defense Interceptor

2.7.1 Mission Profile and Requirements

Homeland Defense Interceptor (HDI) is designed to perform a range of high-demand air defense missions, ensuring readiness in a variety of scenarios. For this case the mission for the HDI are; defensive counter-air (DCA) patrol mission, point defense intercept mission, intercept/escort mission. These missions configurations deeteme-ine the aircraft's performance, maneuverability, endurance and weapons capabilites under various combat conditions.

1. **Defensive Counter-air (DCA) Patrol Mission**: This mission is focus on counter incoming threats for extended periods of time, protecting national assets or high-value zones (cities, infrastructure, military bases) against incoming hostile aircraft or missiles. This mission also focuses on sustaining air defense within a radius of the base, given figure is the picture of DCA Patrol Mission Profile Figure 3.2.

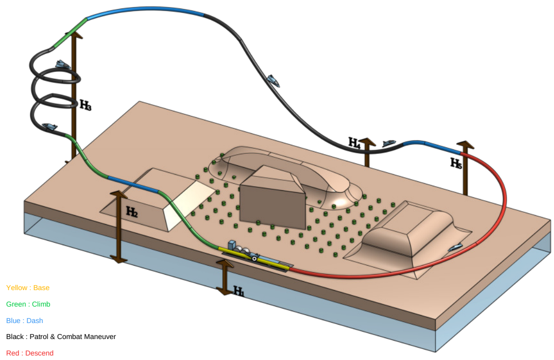


FIGURE 2.14: Defensive Counter-air (DCA) Patrol Mission Profile

2. **Point Defense Intercept Mission**: This mission focus on quick response, the interceptors needs to be able to responds threats that are within a short distance of the base, concentrated protection of a specific location or asset from airborne threats, often involving restricted operating areas and quick dashes and intercepts cycles. High speed and combat maneuvers are the main

focus of this mission in order to destroy approaching targets, given figure is the picture of PDI Mission Profile Figure 3.3.

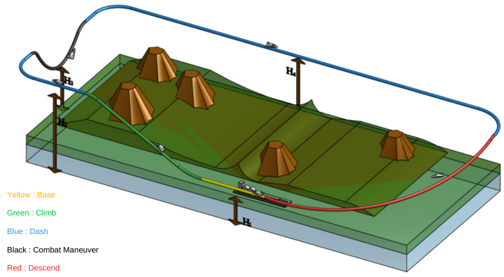


FIGURE 2.15: Point Defense Intercept Mission Profile

3. **Intercept/Escort Mission**: This mission need the interceptors to reach high speed faster, followed by an extended low-speed escort before safely returning to base as part of the interception/escort missions, given figure is the picture of Intercept/Escort Mission Profile Figure 3.4.

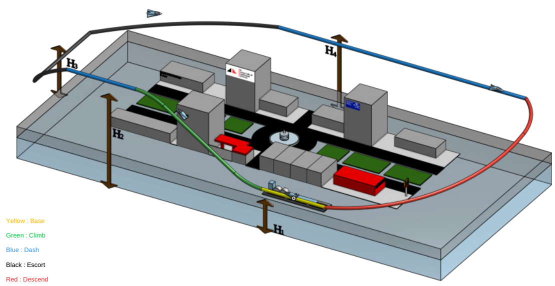


FIGURE 2.16: Intercept/Escort Mission Profile

The main focus of the study topic is determining whether F-16 can perform these mission profiles with no modification, except weaponry. If successful, it would confirm that re-suing a proven multi-role jet fighters into specific homeland defense intercept missions is feasible.

2.8 Previous Studies

2.8.1 Current Capabilities and Challenges of NDARC and SUAVE for eVTOL Aircraft Design and Analysis

This study compares two conceptual design and analysis tools for electric Vertical Takeoff and Landing (VTOL) aircraft designs. NDARC (NASA Design and Analysis of Rotorcraft) and SUAVE (Standford Aircraft Vehicle Environment). this paper was a collaboration between the U.S. Army Combat Capabilities Development Command and Standford University, which was presented at the 2019 AIAA Propulsion and Energy Forum [34].

Using publicly available data, the study involved modeling the Cora design and assessing how well it performed on reference tasks. Both analysis tools were used to determine the aircraft's structural weight, performance characteristics, and maximum takeoff weight (GTOW). The modeling approach's sensitivity was examined by analyzing the trade-offs between variables like rotor and wingspan.

NDARC use calibrated semi-empirical techniques to generate potent predictions based on historical data, whereas SUAVE uses physics-based simulations to obtain insights into complicated settings. The primary finding demonstrates that while the tools' GTOW estimations are similar, there are significants differences in the estimations of structural weight and aerodynamic performance. Due of its more cautious aerodynamic assumptions, SUAVE predicts higher drag but reduced structural weight.

The results fo these tools indicate that the motor and rotor weights are crucial design considerations. The optimization of rotor radius and wingspan exhibits contradictory trends, underscoring the necessity of meticulous assumption validation. This study emphasizes the value of using a variety of cross-validation techniques to address uncertainty in eVTOL design.

2.8.2 Preliminary Correlations for Remotely Piloted Aircraft Systems Sizing

This journal provides rapid sizing technology for H-tail Unmanned Remotely Piloted Aircraft Systems (RPAS), covering the light and medium categories based on comprehensive database of 398 aircrafts. Researchers from the Universidad Politécnica de Madrid conducted the study, which combines statistical correlatoin analysis with geometric and aerodynamic design principles to simplify the technology at the conceptual design stage [35].

Due to its mission specific versatility and structural benefits, the study highlights the use of H-tail designs in larger UAV systems. H-tail design may accommodate engine and cargo systems without requiring significant modifications to the airframe. This is accordance with trends observed in RPAS and other unconventional aircraft designs where quick prototyping and mission adaptability are essential.

The study developed a quick correlation based sizing method to determine aerodynamic and structural parameters such as wing area, tail dimensions, and volume coefficients. This method is similar to modern tools like SUAVE, which continuously resizes and optimizes aircraft layouts using physics-based models. By giving dependable beginning estimates, statistical correlation can be included to early designs to improve iterative and thorough simulations while drastically cutting down on computing time and speeds up convergence.

It was highlighted how crucial it is to construct an RPAS with mission flexibility, specifically payload capacity, endurance, and cruising speed. This aligns with optimization goals, especially when it comes to performance parameters like MTOW, fuel efficiency, and thrust-to-weight ratio for different aircraft configurations like supersonic or even hybrid electric aircrafts.

Based on MTOW. engine type and tail configurations, this study categorizes RPAS to guarantee that designs relevance is representative of a range of mission configurations and operation conditions. This categorization method can enhance multi-objective optimization processes that effectively balance performance and environmental impact trade-offs in a range of mission requirements, such as supersonic aircraft and urban air mobility (UAM) designs.

The results of the rapid sizing process' validation using a thorough RPAS database showed an average error range of less than 10%. For conceptual design tools that verify the accuracy of predictions by contrasting them with real data or pre-existing designs, this highlights the significance of cross-validation.

Based on data methods can be integrated into iterative optimization tools using correlation-based methodologies as a foundation. By using past performance data, this method can increase model accuracy and provide robust initial conditions. The design of some parts, such wings and tails, can be guided by statistical correlations in the early phases of development, which simplifies the procedure and boosts the effectiveness of trade-off analysis in designs. Furthermore, especially for supersonic or hybrid aircraft configurations, the results can be used to customize optimization algorithms to match mission-specific requirements like fuel efficiency or aerodynamic stability.

This study shows how rapid-sizing methods can connect the gap between early stage designs and thorough optimization. The concepts discusses can be used for various unconventional aircraft designs, such as hybrid electric and supersonic aircraft, even though RPAS is the primary focus in this study. Future research can combine quick initial sizing processes with advanced simulation tools to improve the efficiency and accuracy of aircraft conceptual design, particularly for unconventional designs.

2.8.3 Simultaneous Aircraft Sizing and Multi-Objective Considering Off-Design Mission Performance During Early Design

Important conclusions show how non-design factors influence ideal design parameters, for instance, larger configurations may be needed to optimize the aircraft for non-design missions, such as extended range capabilities, this configuration will increase its maximum ramp weight (MRW) and it will also affect mission coverage index (MCI). Finding the right balance between operational productivity and fuel economy requires making these trade-offs, which Pareto focus into these trade-offs, which allows designers to prioritize specific performance metrics based on task requirement [36].

Significant finding demonstrate how optimal design parameters are influenced by non-design elements, for instance, in order to optimize the aircraft for non-design objectives, such as extended range capabilities, larger configurations could be required. This will improve the MCI and MRW. These trade-offs must be made in order to strike the ideal balance between fuel efficiency and operational productivity. By providing these information on trade-offs, Pareto front under study enables designers to categorize particular performance measures according to mission profiles.

The study also emphasizes the necessity of using subsystem-level analysis and high-fidelity models to increase the accuracy of optimization outcomes. This is parallel with the growing focus on employing thorough simulation to verify design decisions and guarantee that the ideal configuration satisfies technical and operational requirements. Strong framework goals is ensured, allowing for a more thorough and precise analysis of potential aircraft design possibilities.

2.8.4 Preliminary Hybrid-Electric Aircraft Design with Advancements on The Open-Source Tool SUAVE

This study is about enhancing SUAVE modeling tool to take into the consideration the complexity of hybrid electric propulsion systems, which investigates the initial design of hybrid electrical aircraft. This study integrates additional components, such as electric motors, batteries and thermal management systems, into the design process to forecast how these components will interact and affect the overall aircraft plan. In order to combine operational effectiveness and environmental sustainability, this iterative method optimizes the installed power mix and supports sophisticated energy management strategies [37].

The updated version of SUAVE framework on this study incorporates a modular energy network to capture intricate interactions including aerodynamic effects, thrust vector control, and mass flow dynamics. Methodology of this study incorporates energy management strategies, with a focus on trade-offs between mission profiles, energy storage systems and propulsion system components. When designing or sizing an aircraft that can meet strict performance and environmental goals, these aspects must be taken into account.

Studies on SUAVE optimization have revealed trade-offs between aircraft weight, energy efficiency and propulsion architecture. These findings emphasize how crucial hybridization components are to attaining peak performance, such as finding a balance between electrical and conventional fuel. The broader objective of enhancing the aviation system's sustainability aligns with the inclusion of environmental parameters like noise and emissions.

Particularly for unconventional aircraft designs, the study demonstrates how modern tools such as SUAVE can be utilized to explore design trade-offs at the concept stage. Design features including wing loading, thrust-to-weight ratio and energy storage capacity can be chosen with previous data of the interaction between propulsion systems and aerodynamics. Complex physics-based models and optimization approaches can be used to improve the design of hybrid-electric and other advanced aircraft, according to research.

This study emphasizes how crucial iterative simulation methods are for bridging the gap between detailed performance analysis and early design conception. The ideas introduced in this paper could be useful for a range of aircraft configurations, such as those intended for supersonic flight or supersonic flight or urban air mobility.By utilizing the SUAVE optimization frameworks, this study offers insight into how to balance environmental consequences, performance needs, and operational efficiency in creative aircraft desigs.

2.8.5 Influence of Novel Airframe Technologies on the Feasibility of Fully-Electric Regional Aviation

The research focuses on the incorporation of modern materials, hybrid laminar control and load shedding technologies into aircraft design to investigate the impact of novel airframe technologies on the viability of fully-electric regional aviation. This study illustrates how these technologies can be used to improve aircraft energy efficiency and reduce emissions using the SUAVE framework for conceptual design and performance analysis. While load-shedding technologies reduce structural weight by better managing aerodynamics loads, hybrid laminar flow control minimizes drag by extending the laminar flow to critical aerodynamic surfaces. Advanced materials such as thin laminates are used to further reduce aircraft weight and make electric propulsion systems more viable [38].

This research also emphasizes the importance of battery energy density in establishing the viability of electric regional jets. It is shown that a combination of these promising airframe technologies can satisfy the performance capabilities which begin from the battery energy density of 700 Wh/kg. However it also emphasizes that until there is major changes in the batteries, the fully-electric regional aviation model cannot be fully realized. Furthermore, although these airframe developments have achieved significant emission reduction of 81% over the baseline ATR-72 emissions, these developments have some limitations like increase in Direct Operating Cost (DOC) and greater maintenance difficulties. These highlights the need to optimize the environmental benefits against the economic trade-off.

The conclusions of this research are consistent with the larger measure of balancing performance needs, energy efficiency, and environmental impacts in creative aircraft design. This research focuses on Multidisciplinary Design Optimization (MDO), demonstrating how these cutting technologies can be integrated into iterative frameworks such as SUAVE to improve aircraft configurations. Thorough a systematic study of the interactions between propulsion systems, aerodynamic and energy networks, the study illustrates scalable strategies to improve the performance and sustainability of upcoming regional aviation. It also emphasizes the need to combine cutting-edge energy storage systems with innovative airframe technologies to achieve the twin goals of market viability and environmental sustainability.

CHAPTER 3
RESEARCH METHODOLOGY

3.1 Research Methodology

3.1.1 Simulation Modeling

The performance of F-16 was modeled using SUAVE framework, with an emphasis on changes such as payload to improve its mission-specific capabilities based on AIAA RFP. The purpose of these simulations was to theoretically validate the aircraft's ability to satisfy HDI roles.

3.1.2 Validation Techniques

In order to make sure the simulation findings met requirements, the validation process was mostly indirect validation. F-16 model had to configured in SUAVE, results had to be replicated from given SUAVE examples in their repository and website, and results had to be compared with those of similar verified models whether it is give the same results as SUAVE examples but in different missions profiles.

3.1.3 Overview

High readiness levels may be maintained at a reasonable cost by using the current F-16 platform for HDI roles. This strategy makes use of the aircraft's extensive service experience and continuous improvements to satisfy modern defense requirements.

3.2 Research Approach

With a particular focus on the F-16 model utilizing **SUAVE** (Stanford University Aerospace Vehicle Environment) and **OpenVSP** (Open Vehicle Sketch Pad), this chapter describes the systematic approach used in the analysis of aircraft design and performance evaluation through computational tools. Following the guidelines established by the **AIAA** (American Institute of Aeronautics and Astronautics). the process is intended to thoroughly examine the aircraft's performance and design capabilities. In order to accomplish the objectives of the study, it covers every step of the process, from data collection and instrument selection to research design and analytical techniques. To make sure the aircraft design satisfies or surpasses the predetermined criteria, the structured approach that has been selected moves through requirement analysis, baseline design and benchmarking, iterative design refinements, and rigorous validation.

This systematic methodology that follows a set of predetermined stages to ensure that every stage of the aircraft design and analysis is carried out accurately and precisely. These progress consist of:

1. Requirement Analysis: This fundamental stage entails a thorough examination of the project's requirement, including precise mission characteristics, performance standards. The objective is to determine and specify the essential specifications that the aircraft model needs to fulfill in order to direct the following stages of the design process.
2. Benchmarking and Baseline Design: After the initial study, this stage concentrates on creating benchmarks using current jet fighters in service. Key characteristics that are used as benchmarks for the aircraft design in SUAVE and OpenVSP, such as vehicle level properties, aircraft structural assemblies, mission profiles, weaponry, must be compared. This stage ensures that the suggested design is practical and competitive in the present economic and technological environment.
3. Design Iteration: In this stage of study, F-16 model is systematically constructed using SUAVE and OpenVSP. Numerous considerations contributed to the F-16's selection, highlighting how suitable it was for the purpose of this study. F-16, which is wel-known for being multi-role fighter, is still in use

today and is undergoing significant improvements that guarantee its technological features and operational relevance, as detailed earlier in Chapter2.5.1. A thorough database of aircraft specs that was created during the first design process further supports the adoption of the F-16 for this study. As a vital resource for the duration of the analysis, this table provides comprehensive details on dimensions, propulsion type, performance metrics, and operational capacities. It guarantees that every design iteration is in line with validated, empirical data, enabling accurate simulations and analyses. This method not only makes it easier to fully understand the F-16's baseline capabilities, but it also makes it possible to figure out how well it might fulfill the AIAA's mission requirements for a homeland defense interceptor.

In order to improve performance, evaluate feasibility, and guarantee compliance to current operational standards. The design iteration are grounded in a context that blends historical data with forward-looking modifications by utilizing the table's full specifications and integrating insights from the QF-16 modernization. This thorough procedure highlights to feasibility of suggested improvements and the aircraft's capacity to carry out changing defensive roles.

4. Validation: Verifying the design to the original specifications is the last phase in the process. This stage includes thorough simulation and comparative analysis using SUAVE and OpenVSP to make sure the design meets or exceeds the configuration of the aircraft. The first step in the procedure is building the F-16 as a vehicle in the fully Python-based SUAVE environment. To determine whether the aircraft is well-constructed within the theoretical framework.

   In order to make this confirmation simpler, the structure and integration of the model are assessed using OpenVSP after the code has been written in SUAVE. This tool makes it possible to visually evaluate how the model is put together in the SUAVE environment, giving users the chance to examine and confirm the model's configuration accuracy and make sure all parameters are set up correctly. The design's theoretical integrity, practicality, and executability under real-world limitations is ensured by this dual-check with SUAVE for performance metrics and OpenVSP for structural integrity.

3.3 Research Flowchart

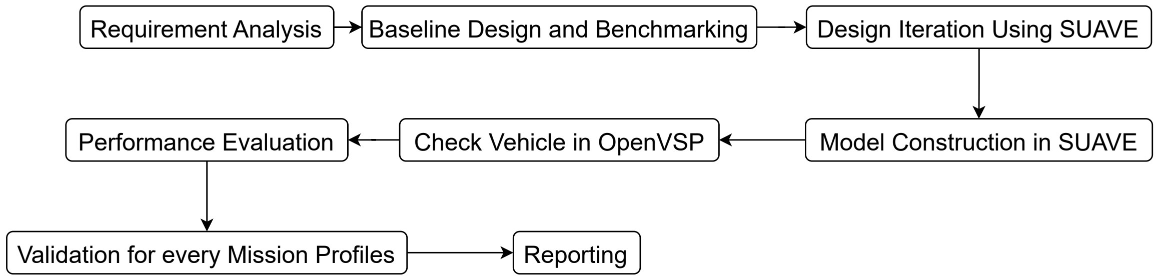


FIGURE 3.1: Flow Chart of Analyzing F-16

The entire analysis process for the examination of an F-16 airplane model using SUAVE and OpenVSP is shown in Figure 3.1. The flowchart shows the methodical procedures followed from the first concept to the last reporting stage. Below is summary of every step:

1. Requirement Analysis: The first step in the process is analysis of the requirements, with a focus on the requirements of the project, such as mission profiles, performance standards.
2. Baseline Design and Benchmarking: In this phase, a baseline is created by examining current interceptor aircraft to establish standards for weight and performance.
3. Design Iteration Using SUAVE: The aircraft design is iteratively constructed through the use of SUAVE. This involves modifying the design specifications in light of the benchmarking and baseline design phase's result.
4. Model Building in SUAVE: F-16 model is built in the SUAVE environment, taking into account all required configurations and parameters.
5. Check Construction in OpenVSP: OpenVSP is used to verify the model's structural and geometrically validity following its construction in SUAVE. This stage guarantees the accuracy and integrity of the model.
6. Performance Evaluation: To determine whether the model satisfies the requirement and to pinpoint areas for improvement, performance simulations are carried out in SUAVE.

7. Validation for Each Mission Profile: To make sure the design satisfies all performance and operational requirements given in the project specifications, each mission profile created during the requirement analysis is validated.
8. Reporting: The last stage is gathering and disseminating the results, recording the entire procedure, and summarizing the conclusions reached during the study.

3.4 Overview Comparison with Other Aircraft

A thorough analysis of jet aircraft specifications was the first step in this study, which was essential for comprehending the competitive environment and important performance standards, all jet fighters that can carry out the HDI mission profile are listed in the Table 3.1. This review took into account compliance with military requirements, technical improvements, and operational capabilities. Particularly, not all people had access to complete data. Because an OpenVSP model was available, allowing for more thorough analysis, the F-16 was chosen. F-16, which is well known for its versatility and constant modernization, was the perfect choice to assess if it might satisfy AIAA's homeland defense interceptor requirements. As discussed earlier in Section 2.5.1, this highlight the F-16 as a top choice that successfully satisfies cutting-edge military requirements.

3.5 Request for Proposal From AIAA

3.5.1 Defensive Combat-Air Patrol Mission Profile

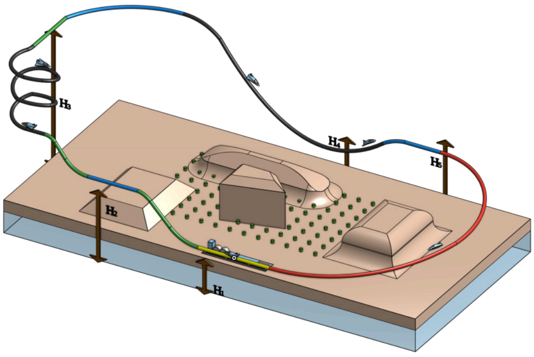


FIGURE 3.2: Defensive Counter-air (DCA) Patrol Mission Profile

TABLE 3.2: Defensive Counter-Air Patrol Mission Phases Description

Phase	Description
1	Take-off and acceleration
2	Climb from sea level to optimum cruise altitude
3	Cruise out 300 nm at optimum speed and altitude
4	Combat air patrol for 4 hours at best loiter speed at 35,000 ft
5	Dash 100 nm at maximum speed at 35,000 ft
6	Combat maneuvers at 35,000 ft, maximum thrust and fuel flow. Fire all missiles and retain gun ammunition
7	Climb/accelerate to optimum speed and altitude
8	Cruise back 400 nm at optimum speed and altitude
9	Descend to sea level
10	Reserves: Fuel for 30 minutes at sea level at maximum endurance speed

3.5.2 Point Defense Intercept Mission Profile

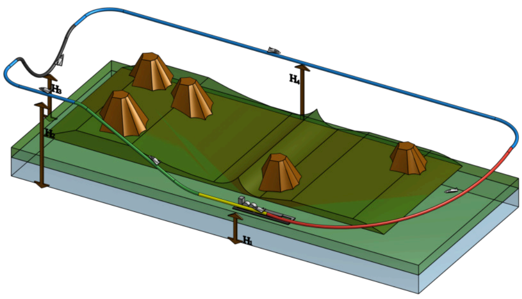


FIGURE 3.3: Point Defense Intercept Mission Profile

TABLE 3.3: Point Defnse Intercept Mission Phases Description

Phase	Description
1	Take-off and acceleration
2	Climb from sea level to 35,000 ft and accelerate to maximum speed
3	Dash 200 nm at maximum speed at 35,000 ft
4	Combat maneuvers at 35,000 ft, maximum thrust and fuel flow. Fire all missiles and retain gun ammunition
5	Climb/accelerate to optimum speed and altitude
6	Cruise back 200 nm at optimum speed and altitude
7	Descent to sea level
8	Reserves: Fuel for 30 minutes at sea level at maximum endurance speed

3.5.3 Intercept/Escort Mission Profile

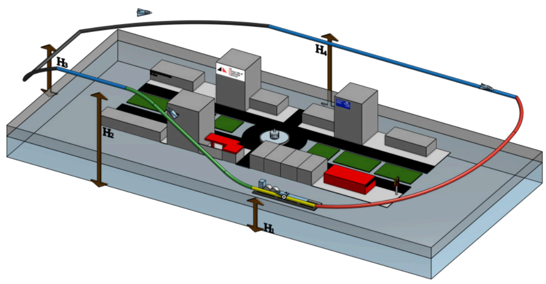


FIGURE 3.4: Intercept/Escort Mission Profile

TABLE 3.4: Intercept/Escort Mission Phases Description

Phase	Description
1	Take-off and acceleration
2	Climb from sea level to 35,000 ft and accelerate to maximum speed
3	Dash out at maximum speed at 35,000 ft
4	Escort for 300 nm at minimum practical airspeed. Retain all weapons
5	Climb/accelerate to optimum speed and altitude
6	Cruise back at optimum speed and altitude
7	Descend to sea level
8	Reserves: Fuel for 30 minutes at sea level at maximum endurance speed

3.5.4 Minimum Performance Requirements/Constraints

Table 3.5: Mission Performance and Requirements

Mission Performance	**Performance Requirement**
Intercept Radius	200 nm
DCA Mission CAP endurance	At 300 nm radius
Maximum Mach Number	At 35,000 ft: M = 1.6
Maneuver Weight (50% Internal Fuel)	
1-g Specific Excess Power – Military Thrust	0.9M/Sea Level = 200 ft/s, 0.9M/15,000 ft = 50 ft/s
1-g Specific Excess Power – Maximum Thrust	0.9M/Sea Level = 700 ft/s, 0.9M/15,000 ft = 400 ft/s
5-g Specific Excess Power – Maximum Thrust	0.9M/Sea Level = 300 ft/s, 0.9M/15,000 ft = 50 ft/s
Sustained Load Factor – Maximum Thrust	0.9M/15,000 ft
Maximum Instantaneous Turn Rate	At 35,000 ft = 18.0 deg/s

3.5.5 Government Furnished Equipment

TABLE 3.6: Weight Component List

No.	Items	Weight (lb)	Volume (ft^3)
Avionics			
1	ICNIA	100	3
2	3 X MFDs	20	1.5
3	Head-up Display	35	1.6
4	Data Bus	10	0.5
5	INEWS	100	3
Flight and Propulsion Control System			
6	Vehicle Management System	50	1
Fire Control Systems			
7	IRSTS	50	1
8	Active Array Radar	450	6
System and Equipment			
9	Electrical Systems (2 Engines)	300	4
10	Auxiliary Power Unit (APU)	100	1
11	Ejection Seat	160	2
12	OBOGS	35	1
13	OBIGGS	35	1
Air-to-Air Weapons			
14	AIM - 120 AMRAAM	327	3
15	M61A1 20 MM Cannon	275	5
16	Ammunition feed system (500 rounds)	200	1
Total Weight:			2,347

3.6 Software Setup and Configuration

3.6.1 Overview of SUAVE

SUAVE (Stanford University Aerospace Vehicle Environment) is a conceptual level aircraft design environment built with the ability to analyze and optimize both conventional and unconventional designs runs in Python-based framework. This capability is achieved in part by allowing analysis information for aircraft to be drawn from multiple sources. Many others software tools for aircraft conceptual design rely on fixed empirical correlations and other handbook approximation. SUAVE instead provides a framework that can be used to design aircraft featuring advanced technologies by augmenting relevant correlations with physics-based methods [4].

A strong framework for optimizing aircraft parameters is offered by SUAVE, especially for intricate designs like supersonic interceptors. It is a crucial technique for contemporary aircraft design since it can combine several performance goals, including speed, agility, fuel economy, and environmental effect, into a single optimization framework. Engineers can balance the advantages and disadvantages between performance and operational constraints utilizing SUAVE to make sure the aircraft's final configuration satisfies mission requirements. Tools like SUAVE will be essential in pushing the limits of what is possible in supersonic flight as future interceptor designs continue to develop. SUAVE do supports tasks such as:

1. Mission Performance Simulation
2. Aircraft Sizing and Optimization
3. Detailed Aerodynamics and Propulsion Analysis
4. Environmental Performance Evaluation (Noise and Emissions)

3.6.2 Getting Started with SUAVE

Installation Options

SUAVE can be downloaded from its official website [39], it offers two installation options:

1. Standard Install: For users who intended to use the current functionalities without any modifications

2. Development Install: For users that want to contribute or do modifications to SUAVE source code, since SUAVE is open-source and released in the GNU Lesser General Public License (LGPL) 2.1.

3.6.3 SUAVE Installation and Configuration

To use SUAVE effectively, make sure the program is installed and configured correctly. The prerequisites, installation steps, and verification process.

Prerequisites

SUAVE is developed primarily on Python 3, and its work on Python versions 3.6 and above. However it is known that SUAVE won't work on the latest versions, due to latest update from Python 3.10 and above, that makes it not backward compatible with earlier versions, thus it is advisable to use previous Python versions before 3.10.

Using a Python distribution such as Anaconda is highly recommended by SUAVE. Anaconda simplifies the installation procedure by offering the bundled libraries that SUAVE needs. Additionally, it facilitates the creation and management of virtual environments, which makes switching between Python versions easy for users, moreover Anaconda able to import current virtual environment and it can be use in other devices or even as a backup.

Installing SUAVE

Step 1: SUAVE can be download from its official website [39] or its GitHub repository [40]. After downloading, extract the SUAVE file into a preferred directory.

Step 2: Create Virtual Environment Using Anaconda Use Anaconda to build a virtual environment with a compatible Python version to run SUAVE. This virtual environment ensures that necessary dependencies are kept separate and do not affect other projects. To build a virtual environment using Ubuntu terminal, below is the command to create the virtual environment

```
conda create -n py36 python=3.6

```

Step 3: Verifying the Installation In order to verify whether SUAVE has been installed correctly, use one of the provided sample scripts. On its GitHub tutorial page [40], SUAVE provides a series of tutorial scripts. For example, to test "tut_mission_B737.py" file in Tutorials-master folder. First locate the correct folder after that run the script using the following command:

```
python tut_mission_B737.py

```

If the file executes successfully and produces output, SUAVE is ready for use, refer to Figure 3.5 below. Otherwise, any execution errors need to be fixed before continuing.

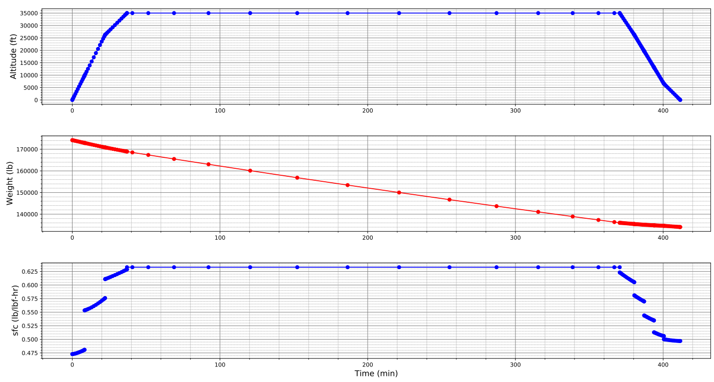


FIGURE 3.5: Altitude, SFC, Weight

3.6.4 Introduction to OpenVSP

A common open-source parametric geometry and analytic tool for conceptual aviation desing is called **OpenVSP** (Open Vehicle Sketch Pad). It enables a thorough approach to aircraft design and optimization by allowing designers to produce intricate 3D models that can be combined with other analysis tools.

OpenVSP was first created at NASA Ames in the early 1990s and has undergone substantial development since then. Originally called the Rapid Aircraft Modeler (RAM), it was mostly utilized on computers made by Silicon Graphics (SGI). The software has advanced from a basic modeling tool to a powerful geometry and analysis engine essential to many aircraft design frameworks, with significant contributions from a range of aerospace professionals. Wider adoption and more adaptability across several operating systems were made possible by the shift from platform-specific libraries to more universal ones like OpenGL and Xforms [5].

3.6.5 Features and Capabilities

By offering a collection of parametric components and combinations that may be altered to model intricate aircraft designs, OpenVSP makes the process of creating airplane models easier. Among its abilities are:

- Parametric Modeling: Users may easily modify and iterate their designs by defining geometric forms with parameters.
- Integration with Analysis Tools: The output files from OpenVSP models are compatible with a number of engineering analysis tools, making it possible to move smoothly between the design and analysis stages.
- Wide Adoption: The tool's versatility and broad usefulness are demonstrated by the fact that it is used by startups, business, government, and academia in the aerospace sector.

3.6.6 Configuring OpenVSP for SUAVE

Prerequisites

The following step is to download the most recent version of OpenVSP from their official website [41], after downloading SUAVE in the device. Since SUAVE has problems with Python 3.10 and later versions, make sure to get the version that is compatible which is Python 3.9.

Step 1

Go to the Python folder in the OpenVSP directory (\OpenVSP-3.41.2-win64\python) after downloading OpenVSP. ”python=3.6” needs to be changed to ”python=3.9” when you open the environment.yml file.

Step 2

Create a new environment based on Python 3.9 by going to the Environment tab in Anaconda after upgrading the Python version in the requirements.

Step 3

Open the terminal when the environment has been set up. Open the main OpenVSP folder (\OpenVSP-3.41.2-win64\python) and navigate to README.md file, to read through the steps for getting the API, there will be two methods of installation are available:

- ```
  pip install -r requirements.txt # If you are not going to modify the packages
  ```

- ```
  pip install -r requirements-dev.txt # If you want to modify the python packages
  ```

Once the API has been installed successfully, user may use the example included in the OpenVSP directory (\OpenVSP-3.41.2-win64\python\openvsp\openvsp\tests). The successful completion of the tests verifies that the OpenVSP API is configured correctly. At this point, user are prepared to use OpenVSP to analyze the vehicle constructed in the SUAVE.

3.7 Configuring Workflow in SUAVE

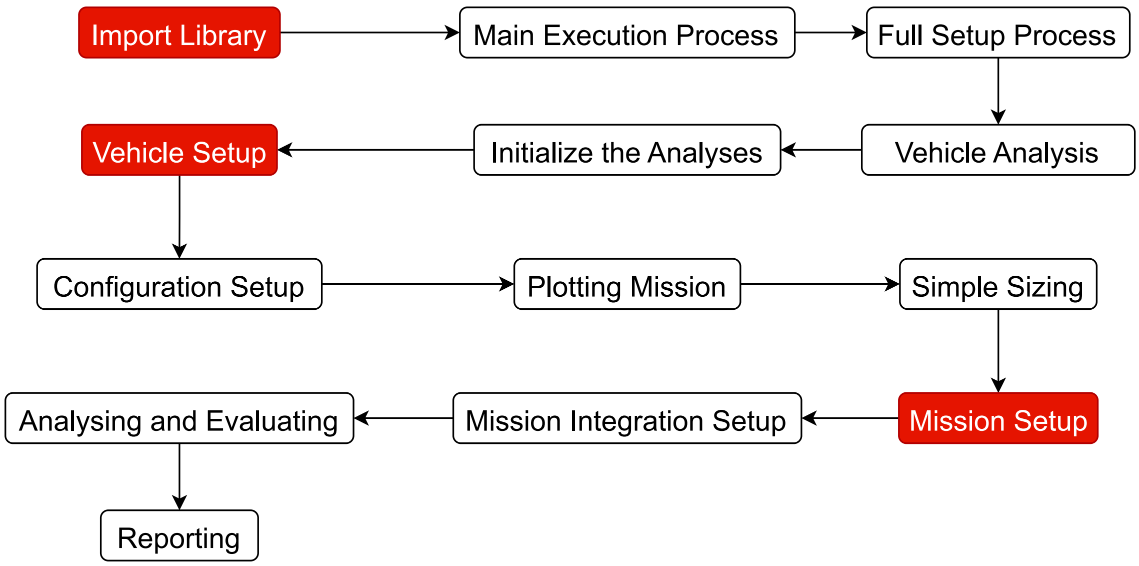


FIGURE 3.6: SUAVE Flowchart

Given figure 3.6, is the flowchart when users want to use SUAVE frameworks.

Import Library

```
import SUAVE
assert SUAVE.__version__=='2.5.2', 'These tutorials only work with
  the SUAVE 2.5.2 release'
from SUAVE.Core import Units
from SUAVE.Plots.Performance.Mission_Plots import *
from SUAVE.Methods.Propulsion.turbofan_sizing import turbofan_sizing
from SUAVE.Methods.Geometry.Two_Dimensional.Planform import (
wing_planform,
segment_properties,
)

import pylab as plt

from copy import deepcopy
```

This section is to set up the SUAVE script initially, with an emphasis on importing libraries and modules needed to execute the F-16 to do simulation using SUAVE.

- import SUAVE: Import to access all of the features and modules associated with the simulation
- assert SUAVE.___version___ == '2.5.2': Ensures that the appropriate SUAVE version is being used.
- from SUAVE.Core import Units: Used throughout the script to manange unit conversions and preserve dimensional consistency in computations.
- from SUAVE.Plots.Performance.Mission_Plots import *: Used to visualize several aspects of the aircraft's performance during the simulation
- from SUAVE.Methods.Propulsion.turbofan_sizing import turbofan_sizing: Used to properly set up and size the aircraft's propulsion systems.
- from SUAVE.Methods.Geometry.Two.Dimensional.Planform import (wing_planform, segment_properties): Used to defining the aircraft's wing geometries.
- import pylab as plt: Used for plotting the analysis.
- from copy import deepcopy: Helpful for replicating configurations or settings that need to be changed without the original data.

Main Execution Process

The complete simulation process for the F-16 vehicle in SUAVE is coordinated by the main() function. Here is a brief explanation of every step:

```
def main():
    configs, analyses = full_setup()
    simple_sizing(configs)

    configs.finalize()
    analyses.finalize()

    weights = analyses.configs.base.weights
    breakdown = weights.evaluate

    mission = analyses.mission.base
    results = mission.evaluate()

```

```
    plot_mission(results)

    plt.show()

    return
```

- Setup Initialization:
  - This function begins by using full_setup() to initialize the aircraft configurations and analysis settings. This sets up all of the aircraft's required models and settings.
- Sizing Adjustments:
  - To maximize the aircraft's configurations for the simulation, simple_sizing(configs) modifies important sizing parameters.
- Finalizing Configurations and Analyses:
  - configs.finalize and analyses.finalize() prepares the models for simulation by locking in all setups and analytical parameters.
- Weight Analysis:
  - This segment gives information on the weight characteristics of the aircraft.
- Mission Analysis:
  - For simulating the flight and compute performance across different segments are defined in this part.
- Results Visualization:
  - This part presents the plots of important flight parameters that are produced by plot_mission(result), providing a graphic representation of the aircraft's performance during the mission.

Full Setup Process

The complete setup needed to analyze the F-16 within the SUAVE framework is coordinated by the full_setup(). This function creates a single, simulation-ready model by combining the aircraft's configuration, analysis parameters, and mission profiles. The steps in this function are broken down as follows:

```
def full_setup():

  vehicle = vehicle_setup()
  configs = configs_setup(vehicle)

  configs_analyses = analyses_setup(configs)

  mission = mission_setup(configs_analyses)
  missions_analyses = missions_setup(mission)

  analyses = SUAVE.Analyses.Analysis.Container()
  analyses.configs = configs_analyses
  analyses.missions = missions_analyses

  return configs, analyses
```

- Vehicle Configuration:
  - This function establishes the fundamental properties and geometries of the aircraft, establishing variables such as aerodynamics, weight, and dimensions.
- Configuration Setup:
  - This function initializes the aircraft's base, cruise, takeoff, and landing configurations, among other flight settings, customizing the aircraft model for each stage of the missions.
- Analysis Configurations:
  - This function organizes analyses involving sizing, weights, aerodynamics, and propulsion system to prepare the analytical models for every aircraft configuration.
- Mission Setup:
  - This function will defines aircraft mission phases such as climbs, cruises, and descents.
  - These missions are compiled into a structured analysis sequence that is prepared for the execution.
- Integration of Configuration and Analyses:

- This function places all missions and configurations inside the SUAVE container. the configured analyses and mission profiles are linked into the primary analysis framework in this function.

Vehicle Analysis

In order to ensure that every mission phases has customized analytical model to evaluate its performance, this function is made to set up and customize analyses for various aircraft configurations. Accurate simulations under a range of flight situations depends on this procedure.

```
def analyses_setup(configs):
  for tag, config in configs.items():
    analysis = base_analysis(config)
    if tag == "cruise_spoiler":
      analysis.aerodynamics.settings.spoiler_drag_increment = 0.004
      aerodynamics.settings.drag_coefficient_increment = 0.002
    analyses[tag] = analysis
  return analyses
```

- Setup Process:
  - First, analyses container created in this function. All of the analysis settings for various aircraft configuration will be stored in this container.
- Configuration-Specific Analysis:
  - The function repeatedly goes over every aircraft configuration that is kept in configs. This function is used to set up a base analysis for each configuration, configuring fundamental analytical models including weights, propulsion systems, and aerodynamics.
- Customization for Specific Configuration: A verification for particular tags, such as "cruise_spoiler" is included within the loop. Additional aerodynamic modifications are applied if this tag is found. Settings to meet specific needs of the configuration:
  - Increment for Spoiler Drag: The drag for the spoiler configuration is increased in this function by 0.004, which helps in examining how the aircraft behaves when spoilers are deployed.

  - Drag Coefficient Increment: In the same way, the drag also increased by 0.002, which improves the aerodynamic profile for precision in simulations including the cruise spoiler by adjusting the total drag coefficient.
- Storing Analysis Setting:
  - Each customized analysis object is put back into the analysis container with the key that corresponds to its tag. This arrangement makes it simple to find and use the appropriate analytic parameters during the simulation.

Initialize the Analyses

This function is crucial for setting up the core analyses required to evaluate the vehicle's performance across multiple areas, from basic geometry to energy management. This function constructs a comprehensive suite of analyses for a given vehicle configuration, ensuring that all necessary evaluations can be performed effectively.

```
def base_analysis(vehicle):
  analyses = SUAVE.Analyses.Vehicle()

  sizing = SUAVE.Analyses.Sizing.Sizing()
  sizing.features.vehicle = vehicle
  analyses.append(sizing)

  weights = SUAVE.Analyses.Weights.Weights_Transport()
  weights.vehicle = vehicle
  analyses.append(weights)

  aerodynamics = SUAVE.Analyses.Aerodynamics.Supersonic_Zero()
  aerodynamics.geometry = vehicle
  aerodynamics.settings.drag_coefficient_increment = 0.0000
  aerodynamics.settings.span_efficiency            = .8

  stability = SUAVE.Analyses.Stability.Fidelity_Zero()
  stability.geometry = vehicle
  analyses.append(stability)

  energy= SUAVE.Analyses.Energy.Energy()
```

```
    energy.network = vehicle.networks #what is called throughout the mission (at every time step))
    analyses.append(energy)

    planet = SUAVE.Analyses.Planets.Planet()
    analyses.append(planet)

    atmosphere = SUAVE.Analyses.Atmospheric.US_Standard_1976()
    atmosphere.features.planet = planet.features
    analyses.append(atmosphere)

    return analyses
```

- Initialize Analyses:
  - This function creates a vehicle-specific analysis, which ensures vehicle-related analyses will be kept in this container.
- Sizing Analysis:
  - This vehicle object is attached to the sizing analysis to ensure that all geometric and physical properties, after that the analyses container is then supplemented with the sizing module.
- Weight Analysis:
  - The weight aspects of the vehicle are analyzed in this function. In order to obtain structural and payload data, this module also sets a direct connection with th vehicle data which is a part of the main container for analyses.
- Aerodynamic Analysis:
  - Aerodynamic properties are handled in this function, for computations the module is set up to use the vehicle's geometry. Specific aerodynamic settings, like as drag coefficient and span efficiency are adjusted to customize the study to supersonic conditions. The analyses container is then updated with this aerodynamic analysis.
- Stability Analysis:
  - The vehicle's stability performance can be assessed using the stability analysis module. It also references the vehicle's geometry for detailed

evaluations and is appended to the container.

- Energy Analysis:
  - To control the vehicle's energy network throughout different mission segment. It links directly to the vehicle's eneryg systems, ensuring that all energy flows are accurately modeled.
- Planet and Atmosphere Analysis:
  - This function establishes environmental conditions.It also configured with Earth-like conditions provided by the planet module and is responsible for calculating atmospheric properties affecting flight. For thorough environmental modeling, both analyses are incorporated to the container.

Vehicle Setup

In order to guarantee accurate performance simulations, this function initializes the F-16 model in SUAVE.

```
def vehicle_setup():

    vehicle = SUAVE.Vehicle()
    vehicle.tag = "HI-2025"

    vehicle.mass_properties.max_takeoff = 19187.0 * Units.kg # kg # WEIGHT
    vehicle.mass_properties.operating_empty = (8570.0 * Units.kg) + (90.0 * Units.kg) # kg
    vehicle.mass_properties.takeoff = 19187.0 * Units.kg  # kg
    vehicle.mass_properties.max_zero_fuel = 10360 * Units.kg  # kg
    vehicle.mass_properties.max_payload = 1700 * Units.kg  # kg
    vehicle.mass_properties.max_fuel = 17000 * Units.kg  # kg
    vehicle.mass_properties.cargo = 0.0 * Units.kg  # kg

    vehicle.envelope.ultimate_load = 9.0
    vehicle.envelope.limit_load = 7.0

    vehicle.reference_area = 34.40808
    vehicle.passengers = 0
    vehicle.systems.control = "fully powered"
```

```
    vehicle.systems.accessories = "medium range"
    vehicle.maximum_cross_sectional_area = 1.8 * Units.meter**2
    vehicle.total_length = 15.06 * Units.meter

    wing = SUAVE.Components.Wings.Main_Wing()
    wing.tag = "main_wing"
    wing.areas.reference = 34.40813 * Units.meter**2
    wing.aspect_ratio = 2.96104
    wing.chords.root = 5.61458 * Units.meter
    wing.chords.tip = 1.20312 * Units.meter
    wing.sweeps.quarter_chord = 33.1 * Units.deg
    wing.thickness_to_chord = 0.10000
    wing.taper = wing.chords.tip / wing.chords.root
    wing.dihedral = 0.0 * Units.deg
    wing.spans.projected = 10.09375
    wing.origin = [[6.304 * Units.meter, 0, 0.290 * Units.meter]]
    wing.vertical = False
    wing.symmetric = True
    wing.high_lift = True
    wing.vortex_lift = True
    wing.high_mach = True
    wing.areas.exposed = 0.80 * wing.areas.wetted
    wing.twists.root = 0.0 * Units.degrees
    wing.twists.tip = 0.0 * Units.degrees
    wing.dynamic_pressure_ratio = 1.0

    flap = SUAVE.Components.Wings.Control_Surfaces.Flap()
    flap.tag = "flap"
    flap.span_fraction_start = 0.402439
    flap.span_fraction_end = 1.0
    flap.deflection = 0.0 * Units.deg
    flap.chord_fraction = 0.243902
    flap.configuration_type = "trailing_edge"
    wing.append_control_surface(flap)

    wing = wing_planform(wing)

    wing.areas.exposed = 0.90 * wing.areas.wetted
    wing.twists.root = 0.0 * Units.degrees
    wing.twists.tip = 0.0 * Units.degrees
```

```
    wing.dynamic_pressure_ratio = 1.0

    # add to vehicle
    vehicle.append_component(wing)

    wing = SUAVE.Components.Wings.Stabilator()  # Horizontal_Tail()
    wing.tag = "stabilator"
    wing.areas.reference = 12.32273 * Units.meter**2  #  #6.16136
    wing.aspect_ratio = 2.08396  # 1.04198
    wing.sweeps.quarter_chord = 41.48750 * Units.deg
    wing.thickness_to_chord = 0.10000
    wing.taper = 0.36360
    wing.dihedral = 0.0 * Units.degrees
    wing.origin = [[12.391 * Units.meter, 0.500 * Units.meter, 0.290 *
     Units.meter]]
    wing.vertical = False
    wing.symmetric = True
    wing.high_lift = False
    wing = wing_planform(wing)
    wing.areas.exposed = 0.9 * wing.areas.wetted
    wing.twists.root = 2.0 * Units.degrees
    wing.twists.tip = 2.0 * Units.degrees
    wing.dynamic_pressure_ratio = 0.90

    wing = SUAVE.Components.Wings.Vertical_Tail()
    wing.tag = "vertical_stabilizer"
    wing.sweeps.quarter_chord = 0.0 * Units.deg
    wing.thickness_to_chord = 0.03
    wing.areas.reference = 8.46748 * Units.meter**2
    wing.spans.projected = 2.24609 * Units.meter + 0.89063 * Units.
   meter
    wing.chords.root = 5.833 * Units.meter
    # wing.chords.tip = 2.881 * Units.meter
    # wing.taper = wing.chords.tip / wing.chords.root
    wing.aspect_ratio = wing.spans.projected**2.0 / wing.areas.
   reference
    wing.twists.root = 0.0 * Units.degrees
    wing.twists.tip = 0.0 * Units.degrees
    wing.origin = [[9.783 * Units.meter, 0, 0.850 * Units.meter]]
    wing.vertical = True
```

```
    wing.symmetric = False
    wing.high_lift = False
    wing.dynamic_pressure_ratio = 1.0

    # Wing Segments
    segment = SUAVE.Components.Wings.Segment()
    segment.tag = "Root"
    segment.percent_span_location = 0.0
    segment.twist = 0.0 * Units.deg
    segment.root_chord_percent = 1  # 1.0
    segment.thickness_to_chord = 0.03
    segment.dihedral_outboard = 0.0 * Units.degrees
    segment.sweeps.quarter_chord = 71.2 * Units.degrees  # 75.16023
    wing.append_segment(segment)

    segment = SUAVE.Components.Wings.Segment()
    segment.tag = "Break"
    segment.percent_span_location = 0.2839
    segment.twist = 0.0 * Units.deg
    segment.root_chord_percent = 0.49

    segment.thickness_to_chord = 0.03
    segment.dihedral_outboard = 0 * Units.degrees
    segment.sweeps.quarter_chord = 44.6 * Units.degrees
    wing.append_segment(segment)

    segment = SUAVE.Components.Wings.Segment()
    segment.tag = "Tip"
    segment.percent_span_location = 1.0
    segment.twist = 0.0 * Units.degrees
    segment.root_chord_percent = 0.2439
    segment.thickness_to_chord = 0.1
    segment.dihedral_outboard = 0.0
    segment.sweeps.quarter_chord = 44.6 * Units.degrees  # 49.66591
    wing.append_segment(segment)

    # Fill out more segment properties automatically
    wing = segment_properties(wing)
    wing = SUAVE.Methods.Geometry.Two_Dimensional.Planform.
   wing_planform(wing)
```

```

    # # add to vehicle
    vehicle.append_component(wing)

    fuselage = SUAVE.Components.Fuselages.Fuselage()
    fuselage.tag = "fuselage"
    fuselage.origin = [[0, 0, 0]]
    fuselage.number_coach_seats = 1
    fuselage.seats_abreast = 1
    fuselage.seat_pitch = 0.0

    fuselage.fineness.nose = 2.0  # 1.28 * Units.meter
    fuselage.fineness.tail = 4.626  # 3.48

    fuselage.lengths.nose = 4.569 * Units.meter  # 3.748
    fuselage.lengths.tail = 9.0207 * Units.meter  #  8.549
    fuselage.lengths.cabin = 1.4803 * Units.meter  # 2.845
    fuselage.lengths.total = 15.070 * Units.meter
    fuselage.lengths.fore_space = 0.0
    fuselage.lengths.aft_space = 0.0

    fuselage.width = 1.95000

    fuselage.heights.maximum = 1.38201 * Units.meter
    fuselage.heights.at_quarter_length = 1.38201 * Units.meter
    fuselage.heights.at_three_quarters_length = 1.18182 * Units.meter
    fuselage.heights.at_wing_root_quarter_chord = 1.18182 * Units.
   meter

    fuselage.areas.side_projected = 6.08548 * Units.meter**2  # 22.27
    fuselage.areas.wetted = 65.334 * Units.meter**2  # 51.083
    fuselage.areas.front_projected = 1.496 * Units.meter**2

    fuselage.effective_diameter = 1.38 * Units.meter

    fuselage.differential_pressure = (
    7.4e4 * Units.pascal
    )  # Maximum differential pressure

    # # Segment
```

```
    segment = SUAVE.Components.Lofted_Body_Segment.Segment()
    segment.tag = "segment_0"
    segment.percent_x_location = 0.0
    segment.percent_z_location = 0.03000
    segment.height = 0.0
    segment.width = 0.0
    fuselage.Segments.append(segment)

    # Segment
    segment = SUAVE.Components.Lofted_Body_Segment.Segment()
    segment.tag = "segment_1"
    segment.percent_x_location = 0.30341
    segment.percent_z_location = 0.04000
    segment.height = 1.38201 * Units.meter
    segment.width = 1.21036 * Units.meter
    fuselage.Segments.append(segment)

    # Segment
    segment = SUAVE.Components.Lofted_Body_Segment.Segment()
    segment.tag = "segment_2"
    segment.percent_x_location = 0.40171
    segment.percent_z_location = 0.04000
    segment.height = 1.38201 * Units.meter
    segment.width = 1.37582 * Units.meter
    fuselage.Segments.append(segment)

    # Segment
    segment = SUAVE.Components.Lofted_Body_Segment.Segment()
    segment.tag = "segment_3"
    segment.percent_x_location = 0.61651
    segment.percent_z_location = 0.03261
    segment.height = 1.18182 * Units.meter
    segment.width = 1.95000 * Units.meter
    fuselage.Segments.append(segment)

    # Segment
    segment = SUAVE.Components.Lofted_Body_Segment.Segment()
    segment.tag = "segment_5"
    segment.percent_x_location = 1.00000
    segment.percent_z_location = 0.03237
```

```
    segment.height = 0.91971 * Units.meter
    segment.width = 0.91971 * Units.meter
    fuselage.Segments.append(segment)

    # Segment
    segment = SUAVE.Components.Lofted_Body_Segment.Segment()
    segment.tag = "segment_6"
    segment.percent_x_location = 1.0
    segment.percent_z_location = 0.03237
    segment.height = 0.0
    segment.width = 0.0
    fuselage.Segments.append(segment)

    # add to vehicle
    vehicle.append_component(fuselage)

    nacelle = SUAVE.Components.Nacelles.Nacelle()
    nacelle.diameter = 0.76
    nacelle.tag = "nacelle"
    nacelle.origin = [[14.0 * Units.meter, 0, 0.490 * Units.meter]]
    nacelle.length = 1.4
    nacelle.inlet_diameter = 0.50
    nacelle.areas.wetted = 20.0
    vehicle.append_component(nacelle)

    # initialize the gas turbine network
    gt_engine = SUAVE.Components.Energy.Networks.Turbofan()
    gt_engine.tag = "turbofan"
    gt_engine.origin = [[14.0 * Units.meter, 0, 0.490 * Units.meter]]
    gt_engine.number_of_engines = 1.0
    gt_engine.bypass_ratio = 0.76  # F110-GE-129

    # add working fluid to the network
    gt_engine.working_fluid = SUAVE.Attributes.Gases.Air()

    # Component 1 : ram,  to convert freestream static to stagnation
   quantities
    ram = SUAVE.Components.Energy.Converters.Ram()
    ram.tag = "ram"
    # add ram to the network
```

```
    gt_engine.ram = ram

    # Component 2 : inlet nozzle
    inlet_nozzle = SUAVE.Components.Energy.Converters.
   Compression_Nozzle()
    inlet_nozzle.tag = "inlet nozzle"
    inlet_nozzle.polytropic_efficiency = 0.98
    inlet_nozzle.pressure_ratio = 0.98
    # add inlet nozzle to the network
    gt_engine.inlet_nozzle = inlet_nozzle

    # Component 3 :low pressure compressor
    low_pressure_compressor = SUAVE.Components.Energy.Converters.
   Compressor()
    low_pressure_compressor.tag = "lpc"
    low_pressure_compressor.polytropic_efficiency = 0.91
    low_pressure_compressor.pressure_ratio = 3.1
    # add low pressure compressor to the network
    gt_engine.low_pressure_compressor = low_pressure_compressor

    # Component 4 :high pressure compressor
    high_pressure_compressor = SUAVE.Components.Energy.Converters.
   Compressor()
    high_pressure_compressor.tag = "hpc"
    high_pressure_compressor.polytropic_efficiency = 0.91
    high_pressure_compressor.pressure_ratio = 5.0
    # add the high pressure compressor to the network
    gt_engine.high_pressure_compressor = high_pressure_compressor

    # Component 5 :low pressure turbine
    low_pressure_turbine = SUAVE.Components.Energy.Converters.Turbine
   ()
    low_pressure_turbine.tag = "lpt"
    low_pressure_turbine.mechanical_efficiency = 0.99
    low_pressure_turbine.polytropic_efficiency = 0.93
    # add low pressure turbine to the network
    gt_engine.low_pressure_turbine = low_pressure_turbine

    # Component 6 :high pressure turbine
```

```
    high_pressure_turbine = SUAVE.Components.Energy.Converters.Turbine()
    high_pressure_turbine.tag = "hpt"
    high_pressure_turbine.mechanical_efficiency = 0.99
    high_pressure_turbine.polytropic_efficiency = 0.93
    # add the high pressure turbine to the network
    gt_engine.high_pressure_turbine = high_pressure_turbine

    # Component 7 :combustor
    combustor = SUAVE.Components.Energy.Converters.Combustor()
    combustor.tag = "combust"
    combustor.efficiency = 0.99
    combustor.alphac = 1.0
    combustor.turbine_inlet_temperature = 1500
    combustor.pressure_ratio = 0.98
    combustor.fuel_data = SUAVE.Attributes.Propellants.Jet_A()
    # add the combustor to the network
    gt_engine.combustor = combustor

    # Component 8 :core nozzle
    core_nozzle = SUAVE.Components.Energy.Converters.Expansion_Nozzle()
    core_nozzle.tag = "core nozzle"
    core_nozzle.polytropic_efficiency = 0.95
    core_nozzle.pressure_ratio = 0.99
    # add the core nozzle to the network
    gt_engine.core_nozzle = core_nozzle

    # Component 9 :fan nozzle
    fan_nozzle = SUAVE.Components.Energy.Converters.Expansion_Nozzle()
    fan_nozzle.tag = "fan nozzle"
    fan_nozzle.polytropic_efficiency = 0.95
    fan_nozzle.pressure_ratio = 0.99
    # add the fan nozzle to the network
    gt_engine.fan_nozzle = fan_nozzle

    # Component 10 : fan
    fan = SUAVE.Components.Energy.Converters.Fan()
    fan.tag = "fan"
    fan.polytropic_efficiency = 0.93
```

```
    fan.pressure_ratio = 1.7
    # add the fan to the network
    gt_engine.fan = fan

    # Component 11 : thrust (to compute the thrust)
    thrust = SUAVE.Components.Energy.Processes.Thrust()
    thrust.tag = "compute_thrust"
    # total design thrust (includes all the engines)

    thrust.total_design = 131000.0 * Units.N  # afterburner, 76310 kN
normal

    # design sizing conditions
    altitude = 50000.0 * Units.ft
    mach_number = 1.6

    # add thrust to the network
    gt_engine.thrust = thrust

    # size the turbofan
    turbofan_sizing(gt_engine, mach_number, altitude)

    # add  gas turbine network gt_engine to the vehicle
    vehicle.append_component(gt_engine)

    fuel = SUAVE.Components.Physical_Component()
    vehicle.fuel = fuel
    fuel.mass_properties.mass = (
    vehicle.mass_properties.max_takeoff - vehicle.mass_properties.
max_fuel
    )
    fuel.origin = vehicle.wings.main_wing.mass_properties.
center_of_gravity
    fuel.mass_properties.center_of_gravity = vehicle.wings.main_wing.
aerodynamic_center

    return vehicle
```

- General Properties:

  - The aircraft has been initialized with the tag ”HI-2025” and set up with basic mass characteristics like fuel capacity, operational empty weight, and maximum takeoff weight. These parameters specify the aircraft's operational capability and physical boundaries.

- Aerodynamic Properties:
  - Wing Configuration: The area, aspect ratio, sweep, and chords length of the main wing are all set up according to precise requirements. To precisely replicate the wing's performance aerodynamic characteristics such as high lift and vortex lift capabilities are defined.
  - Fuselage Configuration: The fuselage length's, width, and height measurements are provided, as well as aerodynamic characteristics like projected area and fineness ratio. These add to the aircraft's overall aerodynamic profile.
- Stabilizers and Control Surfaces:
  - F-16 has a vertical stabilizer and a horizontal stabilizer (stabilator), each of which is set up with particular aerodynamic characteristics including area, aspect ratio, and sweep angles. The main wing's flaps and other control surfaces are arranged to improve the aircraft's lift and control qualities at various stage of flight.
- Propulsion System:
  - A thorough turbofan engine configuration including parts like combustor, turbines, compressors, and nozzles are provided. The configuration establishes the engine's performance parameters, including thrust capacity, pressure ratios, and efficiency. In order to accurately mimic the engine's thrust and overall energy efficiency, the network also includes energy conversion elements like a fan and ram.
- Additional Components:
  - Specifications regarding the sizes, locations, and capabilities of additional vehicle components, such as nacelles and fuel systems are provided. These elements are necessary to accurately model the performance of the aircraft

Configuration Setup

F-16's operational configurations are created and arranged in this function, which customizes the aircraft's setup for independent SUAVE flight phases. This is essential for specifying certain flying circumstances and ensuring precise simulation outcomes.

```
def configs_setup(vehicle):

  configs = SUAVE.Components.Configs.Config.Container()

  base_config = SUAVE.Components.Configs.Config(vehicle)
  base_config.tag = 'base'
  configs.append(base_config)

  config = SUAVE.Components.Configs.Config(base_config)
  config.tag = 'cruise'

  configs.append(config)

  config = SUAVE.Components.Configs.Config(base_config)
  config.tag = 'takeoff'

  config.V2_VS_ratio = 1.21
  config.maximum_lift_coefficient = 2.

  configs.append(config)

  config = SUAVE.Components.Configs.Config(base_config)
  config.tag = 'landing'

  config.Vref_VS_ratio = 1.23
  config.maximum_lift_coefficient = 2.

  configs.append(config)
```

- initialization:
  - The functions begins by constructing a configuration container. All of the aircraft's unique flying configurations will be stored in this container.
- Base Configuration:

  - This function used to create a base, which replicates the original configuration of the vehicle. This function acts as a basis for adjustments tailored to various flight circumstances.
- Cruise Configuration:
  - The base configuration is used to create a particular configuration for cruising conditions. The 'cruise' setting keeps the default settings but allows cruising-specific modifications, like fuel consumption and aerodynamic alterations.
- Takeoff Configuration:
  - This function is to maximize the aircraft's performance during takeoff. To improve takeoff performance, it modifies parameters such as the maximum lift coefficient and the V2/VS ratio, which represents the safety speed in proportion to stall speed.
- Landing Configuration:
  - This function ensures landing operations are both safe and effective. To enable lower landing speeds and shorter runways, it raises the maximum lift coefficient and modifies the Vref/VS ratio, which is related to the landing approach speed vs stall speed.

Plotting Mission Results

```
def plot_mission(results,line_style='bo-'):
  # Plot Altitude, sfc, vehicle weight
  plot_altitude_sfc_weight(results, line_style) #DONE

  # Plot Velocities
  plot_aircraft_velocities(results, line_style) #DONE

  plot_fuel_use(results, line_style) #DONE

  # Plot Aerodynamic Coefficients
  plot_aerodynamic_coefficients(results, line_style)  #DONE

  # Plot Aerodynamic Forces
  plot_aerodynamic_forces(results, line_style) #DONE
```

```

    # Drag Components
    plot_drag_components(results, line_style) #DONE

    # Plot Flight Conditions
    plot_flight_conditions(results, line_style) #DONE

    plot_flight_trajectory(results, line_style) #DONE

    plot_stability_coefficients(results, line_style) #DONE

    return
```

This function is an essential phase in the simulation process, which is intended to display the overall performance outcomes of the F-16 during the SUAVE simulated mission. An intuitive comprehension of the aircraft's behavior under various operating conditions is made possible by this function, which offers a graphical representation of various flight performance parameters.

The function makes use of a number of plotting procedures that together provide important performance metrics for the aircraft, including altitude, fuel consumption, aerodynamic forces, and stability characteristics. By showing how the aircraft reacts to the specified flight circumstances and maneuvers, each plot adds to a comprehensive plot of the operation.

The function enables a thorough assessment of the aircraft's capabilities and operational effectiveness by combining these graphs into a single presentation. The visual output is a crucial tool for both analysis and reporting since it helps in locating any possible problems or places where the aircraft's performance strategy and design need to be improved.

Simple Sizing

This function applies a number of size changes that standardize the aircraft's operational and physical properties directly to the basic configuration. By making these changes, the simulation is guaranteed to depict consistent and realistic aircraft characteristics.

```
def simple_sizing(configs):

```

```
    base = configs.base
    base.pull_base()

    base.mass_properties.max_zero_fuel = 0.9 * base.mass_properties.max_takeoff

    for wing in base.wings:
      wing.areas.wetted   = 2.0 * wing.areas.reference
      wing.areas.exposed  = 0.8 * wing.areas.wetted
      wing.areas.affected = 0.6 * wing.areas.wetted

    base.fuselages['fuselage'].number_coach_seats = base.passengers

    base.store_diff()

    return
```

- Sizing Steps:
  - Retrieving the aircraft's base configuration from the configuration container is the first step in this function. The most recent configuration settings are synchronized, ensuring all changes are the latest.
- Adjusting Mass Properties
  - To replicate operational weight constraints without fuel, the zero fuel weight is set to 90% of the maximum takeoff weight.
- Configuring Wing Areas:
  - The function modifies a number of area characteristics for every wing component in the vehicle, including:
    * Wetted Area: This is the overall surface area impacted by aerodynamic forces and is set to twice the reference area.
    * Exposed Area: 80% of the wetted surface is the exposed area, which is the portion of the wing that is immediately exposed to airflow.
    * Affected Area: 60% of the wetted area is the affected area, which is used to represent areas that are damaged by environmental conditions and operational wear.
- Fuselage Configuration:

  - The number of coach seats in the fuselage corresponds to the aircraft's passenger capacity.
- Data Synchronization:
  - This function called to document the modifications following these modifications. By tracking changes made to the base configuration, this technique makes sure that all data utilized in ensuring analysis accurately reflects these modifications.

Mission Setup

```
def mission_setup(analyses):

    #   Initialize the Mission
    mission = SUAVE.Analyses.Mission.Sequential_Segments()
    mission.tag = 'DCA test mission'

    # atmospheric model
    atmosphere = SUAVE.Attributes.Atmospheres.Earth.US_Standard_1976()
    planet = SUAVE.Attributes.Planets.Earth()

    # airport
    airport = SUAVE.Attributes.Airports.Airport()
    airport.altitude   =  0.0  * Units.ft
    airport.delta_isa  =  0.0
    airport.atmosphere = SUAVE.Attributes.Atmospheres.Earth.
   US_Standard_1976()

    mission.airport = airport

    # unpack Segments module
    Segments = SUAVE.Analyses.Mission.Segments

    # base segment
    base_segment = Segments.Segment()

    #   First Climb Segment: Constant Speed, Constant Rate

    segment = Segments.Climb.Constant_Speed_Constant_Rate()
```

```
    segment.tag = "climb_1"

    # connect vehicle configuration
    segment.analyses.extend(analyses.base)

    # define segment attributes
    segment.atmosphere = atmosphere
    segment.planet = planet

    segment.altitude_start = 0.0 * Units.km
    segment.altitude_end = 3.048 * Units.km
    segment.air_speed = 144.0 * Units["m/s"]
    segment.climb_rate = 14.0 * Units["m/s"]

    # add to misison
    mission.append_segment(segment)

    #   Second Climb Segment: Constant Speed, Constant Rate

    segment = Segments.Climb.Constant_Speed_Constant_Rate()
    segment.tag = "climb_2"

    # connect vehicle configuration
    segment.analyses.extend(analyses.cruise)

    # segment attributes
    segment.atmosphere = atmosphere
    segment.planet = planet

    segment.altitude_end = 4.57 * Units.km
    segment.air_speed = 165.0 * Units["m/s"]
    segment.climb_rate = 9.0 * Units["m/s"]

    # add to mission
    mission.append_segment(segment)

    #   Third Climb Segment: Constant Speed, Constant Climb Rate

    segment = Segments.Climb.Constant_Speed_Constant_Rate()
    segment.tag = "climb_3"
```

```

    # connect vehicle configuration
    segment.analyses.extend(analyses.cruise)

    # segment attributes
    segment.atmosphere = atmosphere
    segment.planet = planet

    segment.altitude_end = 7.6 * Units.km
    segment.air_speed = 230.0 * Units["m/s"]
    segment.climb_rate = 4.5 * Units["m/s"]

    # add to mission
    mission.append_segment(segment)

    #   Fourth Climb Segment: Constant Speed, Constant Rate

    segment = Segments.Climb.Constant_Speed_Constant_Rate()
    segment.tag = "climb_4"

    # connect vehicle configuration
    segment.analyses.extend(analyses.cruise)

    # segment attributes
    segment.atmosphere = atmosphere
    segment.planet = planet

    segment.altitude_end = 8.5 * Units.km
    segment.air_speed = 240.0 * Units["m/s"]
    segment.climb_rate = 4.0 * Units["m/s"]

    # add to mission
    mission.append_segment(segment)

    #   Fifth Climb Segment: linear Mach

    segment = Segments.Climb.Constant_Speed_Constant_Rate(base_segment
   )
    segment.tag = "climb_5"

```

```
    segment.analyses.extend( analyses.base )

    segment.altitude_end    = 10.67    * Units.km
    segment.air_speed       = 230.0   * Units['m/s']
    segment.climb_rate      = 4.0   * Units['m/s']

    # add to mission
    mission.append_segment(segment)

    #   Cruise Segment: Constant Speed, Constant Altitude

    segment = Segments.Cruise.Constant_Speed_Constant_Altitude(
base_segment)
    segment.tag = "cruise"

    segment.analyses.extend( analyses.cruise )

    segment.air_speed   = 230.412 * Units['m/s']
    segment.distance    = 2490. * Units.nautical_miles

    # Add to mission
    mission.append_segment(segment)

    #   First Descent Segment

    segment = Segments.Descent.Constant_Speed_Constant_Rate(
base_segment)
    segment.tag = "descent_0"

    segment.analyses.extend( analyses.base )

    segment.altitude_end = 8.5    * Units.km
    segment.air_speed     = 305. * Units['m/s']
    segment.descent_rate = 5.0    * Units['m/s']

    # append to mission
    mission.append_segment(segment)

    #   Second Descent Segment: Constant Speed, Constant Rate

```

```
    segment = Segments.Descent.Constant_Speed_Constant_Rate(
base_segment)
    segment.tag = "descent_2"

    segment.analyses.extend( analyses.landing )

    segment.altitude_end = 6.8   * Units.km
    segment.air_speed    = 195.0 * Units['m/s']
    segment.descent_rate = 5.0   * Units['m/s']

    # Add to mission
    mission.append_segment(segment)

    #   Third Descent Segment: Constant Speed, Constant Rate

    segment = Segments.Descent.Constant_Speed_Constant_Rate(
base_segment)
    segment.tag = "descent_3"

    segment.analyses.extend( analyses.landing )

    analyses.landing.aerodynamics.settings.spoiler_drag_increment =
0.00

    segment.altitude_end = 4.0   * Units.km
    segment.air_speed    = 170.0 * Units['m/s']
    segment.descent_rate = 5.0   * Units['m/s']

    # Add to mission
    mission.append_segment(segment)

    #   Fourth Descent Segment: Constant Speed, Constant Rate

    segment = Segments.Descent.Constant_Speed_Constant_Rate(
base_segment)
    segment.tag = "descent_4"

    segment.analyses.extend( analyses.landing )
    analyses.landing.aerodynamics.settings.spoiler_drag_increment =
0.00
```

```
    segment.altitude_end = 2.0   * Units.km
    segment.air_speed    = 150.0 * Units['m/s']
    segment.descent_rate = 5.0   * Units['m/s']

    # Add to mission
    mission.append_segment(segment)

    #   Fifth Descent Segment: Constant Speed, Constant Rate

    segment = Segments.Descent.Constant_Speed_Constant_Rate(
   base_segment)
    segment.tag = "descent_5"

    segment.analyses.extend( analyses.landing )
    analyses.landing.aerodynamics.settings.spoiler_drag_increment =
   0.00

    segment.altitude_end = 0.0   * Units.km
    segment.air_speed    = 145.0 * Units['m/s']
    segment.descent_rate = 3.0   * Units['m/s']

    # Append to mission
    mission.append_segment(segment)

    #   Mission definition complete

    ###         Reserve mission

    #   First Climb Segment: Constant Speed, Constant Throttle

    segment = Segments.Climb.Constant_Speed_Constant_Rate()
    segment.tag = "reserve_climb"

    # connect vehicle configuration
    segment.analyses.extend(analyses.base)

    # define segment attributes
    segment.atmosphere = atmosphere
    segment.planet = planet
```

```
    segment.altitude_start = 0.0 * Units.km
    segment.altitude_end = 18000.0 * Units.ft
    segment.air_speed = 138.0 * Units["m/s"]
    segment.climb_rate = 15.3 * Units["m/s"]

    # add to misison
    mission.append_segment(segment)

    #   Cruise Segment: constant speed, constant altitude

    # segment = Segments.Cruise.Constant_Mach_Constant_Altitude(
   base_segment)
    # segment.tag = "reserve_cruise"

    # segment.analyses.extend(analyses.cruise)

    # segment.mach = 0.5
    # segment.distance = 140.0 * Units.nautical_mile
    # mission.append_segment(segment)

    #   Loiter Segment: constant mach, constant time

    segment = Segments.Cruise.Constant_Mach_Constant_Altitude_Loiter(
   base_segment)
    segment.tag = "reserve_loiter"

    segment.analyses.extend(analyses.cruise)

    segment.mach = 0.5
    segment.time = 30.0 * Units.minutes

    mission.append_segment(segment)

    #  Final Descent Segment: consant speed, constant segment rate

    segment = Segments.Descent.Linear_Mach_Constant_Rate(base_segment)
    segment.tag = "reserve_descent_1"

    segment.analyses.extend(analyses.landing)
```

```

    segment.altitude_end = 0.0 * Units.km
    segment.descent_rate = 5.0 * Units["m/s"]
    segment.mach_end = 0.25
    segment.mach_start = 0.4

    # append to mission
    mission.append_segment(segment)

    ###         Reserve mission completed

    return mission
```

This function starts by defining the basic airport circumstances, initializing a mission profiles, and configuring the planet and atmospheric environmental model. Accurate simulation of flight segments under typical atmospheric and planetary conditions depends on following conditions:

- Segment Configuratoin:
  - Different flight segments are set up in a sequential order, including climbs, cruises, descents, reserve loiters. Specific elements of the mission profile including ascending to cruising altitude, returning to base, are reflected in each segment
- Environmental Setup:
  - Common atmospheric model, in this function. To guarantee that the flight conditions are accurate and in line with actual operations.
- Airport Configuration:
  - To serve as a reference point for takeoff and landing simulations, the base airport configuration sets the temperature offset (delta_isa) and altitude offset to standard values.
- Mission Phases: There are several different stages of the mission, including:
  - Climb Segment: Model the aircraft reaching operating altitude at pre-determined climb rates and speeds.
  - Cruise Segment: Essential for long-distance flights, depicts the aircraft moving at certain speed and altitude.

    - Descent Segments: Discuss how the plane descends back to lower altitude in order to land.
    - Reserve Missions: Extra segments such loiters and reserve climbs ensures compliance to legal specifications for emergency protocols and reserve fuel.
- Analysis Integration:
    - To ensures that the aerodynamic, energy, and weight analyses are appropriately applied based on the segments phases, each segments are connected to a particular configuration analysis.

Mission Integration Setup

```python
def missions_setup(base_mission):
    missions = SUAVE.Analyses.Mission.Mission.Container()
    missions.base = base_mission
    return missions

if __name__ == '__main__':
    main()
```

This functions are arranged and ready for simulation by the mission_setup() function. This function si essential for overseeing how the specified mission scenarios are carried out.

- Initialization: A mission container is created for the simulation's mission scenarios are stored and managed in this container.
- Integration of Base Mission: This function adds the previously set-up base mission to the mission container. In the simulation framework, this prepares the mission for execution.
- Program Execution: This function executes as part of the overall simulation workflow to make sure the mission is configured properly prior to the simulation commencing.
- At the end of script if___name___ == '___main___' determines that the script is the main module, it then calls the main() method. After integrating the mission using mission_setup(). this function launches the simulation.

3.8 Exporting OpenVSP from SUAVE

3.8.1 Introduction

The smooth transfer of aircraft geometry data from SUAVE's analytical and simulation environment to OpenVSP's visual and interactive environment is made possible by the combination of SUAVE and OpenVSP, given figure 3.7, is the flowchart when users want to export SUAVE vehicle configuration into OpenVSP file (.vsp3).

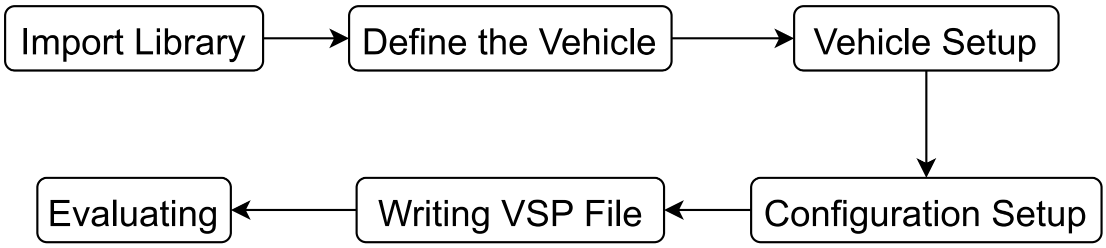


FIGURE 3.7: Exporting SUAVE file into OpenVSP file

3.8.2 Overview of SUAVE and OpenVSP Integration

Aircraft geometry can be exported from SUAVE to OpenVSP for visualization and additional analysis. This feature is essential for confirming design hypotheses and modifying in real time in response to visual input. The goal is to use OpenVSP's visualization capabilities to verify and improve airplane geometry created in SUAVE.

3.8.3 Configuration Setup

First we need to install the OpenVSP API's first, as discussed in chapter 3.6.6, after installation it can be continued as below:

Import Library

```
import numpy as np
import matplotlib.pyplot as plt
```

```python

import SUAVE
from SUAVE.Core import Units
from SUAVE.Methods.Propulsion.turbofan_sizing import turbofan_sizing
from SUAVE.Methods.Geometry.Two_Dimensional.Planform import (
wing_planform,
segment_properties,
)

from SUAVE.Input_Output.OpenVSP import write, get_vsp_measurements
from SUAVE.Input_Output.OpenVSP.vsp_read import vsp_read

from copy import deepcopy
```

This section is similar to previous Python file in Section 3.7, there are only two differences between these codes which are:

- from SUAVE.Input_Output.OpenVSP import write, get_vsp_measurements: Its used to export an SUAVE aircraft configuration to a format that can be read by OpenVSP. It enables OpenVSP to visualize and further analyze the airplane model defined in SUAVE and imports geometric measurements back into SUAVE after retrieving them from the OpenVSP model.
- from SUAVE.Input_Output.OpenVSP.vsp_read import vsp_read: An OpenVSP model can be simply imported into SUAVE using this function. This makes it possible for OpenVSP to seamlessly integrate detailed airplane geometry that have been produced or altered.

Define the Vehicle

```python
def setup():
  base_vehicle = base_setup()

  vsp_write_read(base_vehicle)

  configs = configs_setup(base_vehicle)

  return configs, base_vehicle
```

- base_vehicle = base_setup(): function to set up the aircraft's basic configuration.
- vsp_write_read(base_vehicle): This function manages the aircraft model's import and export to and from OpenVSP.
- configs = configs_setup(base_vehicle) :This function integrates the OpenVSP modifications and then uses the upgraded base vehicle to set up several flight configurations.

Vehicle Setup

This function is similar as discussed in Section 3.7

Defining the Configurations

```
def configs_setup(vehicle):
  configs = SUAVE.Components.Configs.Config.Container()

  base_config = SUAVE.Components.Configs.Config(vehicle)
  base_config.tag = "base"
  configs.append(base_config)

  config = SUAVE.Components.Configs.Config(base_config)
  config.tag = "cruise"

  configs.append(config)

  config.maximum_lift_coefficient = 1.2

  config = SUAVE.Components.Configs.Config(base_config)
  config.tag = "cruise_spoilers"

  configs.append(config)

  config.maximum_lift_coefficient = 1.2

  config = SUAVE.Components.Configs.Config(base_config)
  config.tag = "takeoff"
  config.wings["main_wing"].control_surfaces.flap.deflection = 20.0
 * Units.deg
```

```
    # config.wings["main_wing"].control_surfaces.slat.deflection = 25.0 * Units.deg
    config.V2_VS_ratio = 1.21
    configs.append(config)

    config = SUAVE.Components.Configs.Config(base_config)
    config.tag = "landing"
    config.wings["main_wing"].control_surfaces.flap.deflection = 30.0 * Units.deg
    # config.wings["main_wing"].control_surfaces.slat.deflection = 25.0 * Units.deg
    config.Vref_VS_ratio = 1.23
    configs.append(config)

    config = SUAVE.Components.Configs.Config(base_config)
    config.tag = "short_field_takeoff"
    config.wings["main_wing"].control_surfaces.flap.deflection = 20.0 * Units.deg
    # config.wings["main_wing"].control_surfaces.slat.deflection = 25.0 * Units.deg
    config.V2_VS_ratio = 1.21

    configs.append(config)

    return configs
```

This function is similar as discussed in Chapter 3.7.

Writing VSP file

```
def vsp_write_read(vehicle):
    """
    Function to read and write into OpenVSP
    """
    write(vehicle, "F-16")

    return
```

The purpose of this function is to export the SUAVE aircraft configuration to OpenVSP so that further geometric analysis or adjustments can be carried out. The main goal is to improve the airplane model based on SUAVE's initial configuration by utilizing OpenVSP's advanced visualization and customization features.

3.9 Limitations

3.9.1 Simulation of Combat Maneuvers

One notable drawback of SUAVE is its lack of ability to replicate precise combat maneuvers required in the AIAA RFP, which mostly affects the DCA and PDI mission profiles. Among these restrictions are:

- Combat Maneuvers Fuel Requirements: SUAVE is unable to precisely model how much fuel used during complex battle maneuvers like:
  - A sustained 360° turn at 35,000 ft at Mach 1.2 with maximum thrust and fuel flow.
  - A sustained 360° turn at 35,000 ft at Mach 0.9 under similar conditions.
- Weapon Deployment Post-Maneuvers: After executing above maneuvers, SUAVE does not allow for the simulation of firing every missile while holding onto gun ammunition. Accurately evaluating the aircraft's operating capability in combat scenarios typical of DCA and PDI missions depends on this limitations.

Its more likely due to SUAVE's inability to support this configuration consistently, most likely in the weight analysis or aerodynamic modules, the wing stake has to be deactivated.

3.9.2 Limitations on Software Installation

Due to its lack of native support for Windows operating systems, SUAVE can only be used effectively in certain circumstances, requiring Ubuntu or virtual environment using Anaconda in Windows.

3.10 Indirect Validation through SUAVE Boeing 737 Simulation Example

3.10.1 Overview of Boeing 737 Simulation Example

Boeing 737, a popular narrow-body commercial airplane, is simulated to demonstrated within SUAVVE framework. This example is especially important since it indirectly validates SUAVE's modeling abilities by showing that it can effectively mimic and predict the performance parameters of an aircraft. The Boeing 737 simulation's SUAVE tutorials is accessible at [42].

3.10.2 Simulation Setup

Wing area, engine thrust, fuselage dimension, and flight environment are just a few of the parameters that are set up in the simulation to replicate Boeing 737's known physical and operational characteristics. These settings are essential for accurately simulating the behavior and flight dynamics of the aircraft.

3.10.3 Execution of the Simulation

Python script that initializes the aircraft model and processes a normal mission profile, which includes the usual flying phases of takeoff, climb, cruise, and descent.

SUAVE Code for the Simulation

```
# tut_mission_B737.py
#
# Created:  Aug 2014, SUAVE Team
# Modified: Aug 2017, SUAVE Team
#           Mar 2020, E. Botero

#
 ----------------------------------------------------------------------

#   Imports
```

```python
# ----------------------------------------------------------------------

# General Python Imports
import numpy as np
# Numpy is a commonly used mathematically computing package. It contains many frequently used
# mathematical functions and is faster than native Python, especially when using vectorized
# quantities.
import matplotlib.pyplot as plt
# Matplotlib's pyplot can be used to generate a large variety of plots. Here it is used to create
# visualizations of the aircraft's performance throughout the mission.

# SUAVE Imports
import SUAVE
assert SUAVE.__version__=='2.5.2', 'These tutorials only work with the SUAVE 2.5.2 release'
from SUAVE.Core import Data, Units
# The Data import here is a native SUAVE data structure that functions similarly to a dictionary.
#   However, iteration directly returns values, and values can be retrieved either with the
#   typical dictionary syntax of "entry['key']" or the more class-like "entry.key". For this to work
#   properly, all keys must be strings.
# The Units import is used to allow units to be specified in the vehicle setup (or elsewhere).
#   This is because SUAVE functions generally operate using metric units, so inputs must be
#   converted. To use a length of 20 feet, set l = 20 * Units.ft . Additionally, to convert to SUAVE
#   output back to a desired units, use l_ft = l_m / Units.ft
from SUAVE.Plots.Performance.Mission_Plots import *
# These are a variety of plotting routines that simplify the plotting process for commonly
```

```
# requested metrics. Plots of specifically desired metrics can also be manually created.
from SUAVE.Methods.Propulsion.turbofan_sizing import turbofan_sizing
# Rather than conventional sizing, this script builds the turbofan energy network. This process is
# covered in more detail in a separate tutorial. It does not size the turbofan geometry.

from copy import deepcopy

# ----------------------------------------------------------------------
#   Main
# ----------------------------------------------------------------------

def main():
"""This function gets the vehicle configuration, analysis settings, and then runs the mission.
Once the mission is complete, the results are plotted."""

# Extract vehicle configurations and the analysis settings that go with them
configs, analyses = full_setup()

# Size each of the configurations according to a given set of geometry relations
simple_sizing(configs)

# Perform operations needed to make the configurations and analyses usable in the mission
configs.finalize()
analyses.finalize()

# Determine the vehicle weight breakdown (independent of mission fuel usage)
weights = analyses.configs.base.weights
breakdown = weights.evaluate()
```

```

# Perform a mission analysis
mission = analyses.missions.base
results = mission.evaluate()

# Plot all mission results, including items such as altitude profile and L/D
plot_mission(results)

return

# ----------------------------------------------------------------------
#   Analysis Setup
# ----------------------------------------------------------------------

def full_setup():
"""This function gets the baseline vehicle and creates modifications for different
configurations, as well as the mission and analyses to go with those configurations."""

# Collect baseline vehicle data and changes when using different configuration settings
vehicle  = vehicle_setup()
configs  = configs_setup(vehicle)

# Get the analyses to be used when different configurations are evaluated
configs_analyses = analyses_setup(configs)

# Create the mission that will be flown
mission  = mission_setup(configs_analyses)
missions_analyses = missions_setup(mission)

# Add the analyses to the proper containers
analyses = SUAVE.Analyses.Analysis.Container()
```

```
analyses.configs  = configs_analyses
analyses.missions = missions_analyses

return configs, analyses

# 
 ----------------------------------------------------------------------

#   Define the Vehicle Analyses
# 
 ----------------------------------------------------------------------


def analyses_setup(configs):
"""Set up analyses for each of the different configurations."""

analyses = SUAVE.Analyses.Analysis.Container()

# Build a base analysis for each configuration. Here the base
 analysis is always used, but
# this can be modified if desired for other cases.
for tag,config in configs.items():
analysis = base_analysis(config)
analyses[tag] = analysis

return analyses

def base_analysis(vehicle):
"""This is the baseline set of analyses to be used with this vehicle
 . Of these, the most
commonly changed are the weights and aerodynamics methods."""

# ------------------------------------------------------------------
#   Initialize the Analyses
# ------------------------------------------------------------------
analyses = SUAVE.Analyses.Vehicle()

# ------------------------------------------------------------------
#  Weights
weights = SUAVE.Analyses.Weights.Weights_Transport()
```

```
weights.vehicle = vehicle
analyses.append(weights)

# ------------------------------------------------------------------
#  Aerodynamics Analysis
aerodynamics = SUAVE.Analyses.Aerodynamics.Fidelity_Zero()
aerodynamics.geometry = vehicle
analyses.append(aerodynamics)

# ------------------------------------------------------------------
#  Stability Analysis
stability = SUAVE.Analyses.Stability.Fidelity_Zero()
stability.geometry = vehicle
analyses.append(stability)

# ------------------------------------------------------------------
#  Energy
energy = SUAVE.Analyses.Energy.Energy()
energy.network = vehicle.networks
analyses.append(energy)

# ------------------------------------------------------------------
#  Planet Analysis
planet = SUAVE.Analyses.Planets.Planet()
analyses.append(planet)

# ------------------------------------------------------------------
#  Atmosphere Analysis
atmosphere = SUAVE.Analyses.Atmospheric.US_Standard_1976()
atmosphere.features.planet = planet.features
analyses.append(atmosphere)

return analyses

#
  ----------------------------------------------------------------------

#   Define the Vehicle
```

```
# ----------------------------------------------------------------------

def vehicle_setup():
"""This is the full physical definition of the vehicle, and is designed to be independent of the
analyses that are selected."""

# ------------------------------------------------------------------
#   Initialize the Vehicle
# ------------------------------------------------------------------

vehicle = SUAVE.Vehicle()
vehicle.tag = 'Boeing_737-800'

# ------------------------------------------------------------------
#   Vehicle-level Properties
# ------------------------------------------------------------------

# Vehicle level mass properties
# The maximum takeoff gross weight is used by a number of methods, most notably the weight
# method. However, it does not directly inform mission analysis.
vehicle.mass_properties.max_takeoff               = 79015.8 * Units.kilogram
# The takeoff weight is used to determine the weight of the vehicle at the start of the mission
vehicle.mass_properties.takeoff                   = 79015.8 * Units.kilogram
# Operating empty may be used by various weight methods or other methods. Importantly, it does
# not constrain the mission analysis directly, meaning that the vehicle weight in a mission
# can drop below this value if more fuel is needed than is available.
vehicle.mass_properties.operating_empty           = 62746.4 * Units.kilogram
# The maximum zero fuel weight is also used by methods such as weights
```

```
vehicle.mass_properties.max_zero_fuel             = 62732.0 * Units.kilogram
# Cargo weight typically feeds directly into weights output and does not affect the mission
vehicle.mass_properties.cargo                     = 10000.  * Units.kilogram

# Envelope properties
# These values are typical FAR values for a transport of this type
vehicle.envelope.ultimate_load = 3.75
vehicle.envelope.limit_load    = 2.5

# Vehicle level parameters
# The vehicle reference area typically matches the main wing reference area
vehicle.reference_area         = 124.862 * Units['meters**2']
# Number of passengers, control settings, and accessories settings are used by the weights
# methods
vehicle.passengers             = 170
vehicle.systems.control        = "fully powered"
vehicle.systems.accessories    = "medium range"

# ------------------------------------------------------------------
#  Landing Gear
# ------------------------------------------------------------------

# The settings here can be used for noise analysis, but are not used in this tutorial
landing_gear = SUAVE.Components.Landing_Gear.Landing_Gear()
landing_gear.tag = "main_landing_gear"

landing_gear.main_tire_diameter = 1.12000 * Units.m
landing_gear.nose_tire_diameter = 0.6858 * Units.m
landing_gear.main_strut_length  = 1.8 * Units.m
landing_gear.nose_strut_length  = 1.3 * Units.m
landing_gear.main_units   = 2    # Number of main landing gear
landing_gear.nose_units   = 1    # Number of nose landing gear
landing_gear.main_wheels = 2    # Number of wheels on the main landing gear
```

```
landing_gear.nose_wheels = 2    # Number of wheels on the nose landing gear
vehicle.landing_gear = landing_gear

# ------------------------------------------------------------------
#   Main Wing
# ------------------------------------------------------------------

# This main wing is approximated as a simple trapezoid. A segmented wing can also be created if
# desired. Segmented wings appear in later tutorials, and a version of the 737 with segmented
# wings can be found in the SUAVE testing scripts.

# SUAVE allows conflicting geometric values to be set in terms of items such as aspect ratio
# when compared with span and reference area. Sizing scripts may be used to enforce
# consistency if desired.

wing = SUAVE.Components.Wings.Main_Wing()
wing.tag = 'main_wing'

wing.aspect_ratio            = 10.18
# Quarter chord sweep is used as the driving sweep in most of the low fidelity analysis methods.
# If a different known value (such as leading edge sweep) is given, it should be converted to
# quarter chord sweep and added here. In some cases leading edge sweep will be used directly as
# well, and can be entered here too.
wing.sweeps.quarter_chord    = 25 * Units.deg
wing.thickness_to_chord      = 0.1
wing.taper                   = 0.1
wing.spans.projected         = 34.32 * Units.meter
wing.chords.root             = 7.760 * Units.meter
wing.chords.tip              = 0.782 * Units.meter
wing.chords.mean_aerodynamic = 4.235 * Units.meter
wing.areas.reference         = 124.862 * Units['meters**2']
wing.twists.root             = 4.0 * Units.degrees
```

```
wing.twists.tip                 = 0.0 * Units.degrees
wing.origin                     = [[13.61, 0, -1.27]] * Units.meter
wing.vertical                   = False
wing.symmetric                  = True
# The high lift flag controls aspects of maximum lift coefficient calculations
wing.high_lift                  = True
# The dynamic pressure ratio is used in stability calculations
wing.dynamic_pressure_ratio     = 1.0

# ------------------------------------------------------------------
#   Main Wing Control Surfaces
# ------------------------------------------------------------------

# Information in this section is used for high lift calculations and when conversion to AVL
# is desired.

# Deflections will typically be specified separately in individual vehicle configurations.

flap                          = SUAVE.Components.Wings.Control_Surfaces.Flap()
flap.tag                      = 'flap'
flap.span_fraction_start      = 0.20
flap.span_fraction_end        = 0.70
flap.deflection               = 0.0 * Units.degrees
# Flap configuration types are used in computing maximum CL and noise
flap.configuration_type       = 'double_slotted'
flap.chord_fraction           = 0.30
wing.append_control_surface(flap)

slat                          = SUAVE.Components.Wings.Control_Surfaces.Slat()
slat.tag                      = 'slat'
slat.span_fraction_start      = 0.324
slat.span_fraction_end        = 0.963
slat.deflection               = 0.0 * Units.degrees
slat.chord_fraction           = 0.1
```

```
wing.append_control_surface(slat)

aileron                             = SUAVE.Components.Wings.
  Control_Surfaces.Aileron()
aileron.tag                         = 'aileron'
aileron.span_fraction_start         = 0.7
aileron.span_fraction_end           = 0.963
aileron.deflection                  = 0.0 * Units.degrees
aileron.chord_fraction              = 0.16
wing.append_control_surface(aileron)

# Add to vehicle
vehicle.append_component(wing)

# ------------------------------------------------------------------
#  Horizontal Stabilizer
# ------------------------------------------------------------------

wing = SUAVE.Components.Wings.Horizontal_Tail()
wing.tag = 'horizontal_stabilizer'

wing.aspect_ratio                 = 6.16
wing.sweeps.quarter_chord         = 40.0 * Units.deg
wing.thickness_to_chord           = 0.08
wing.taper                        = 0.2
wing.spans.projected              = 14.2 * Units.meter
wing.chords.root                  = 4.7   * Units.meter
wing.chords.tip                   = 0.955 * Units.meter
wing.chords.mean_aerodynamic = 3.0   * Units.meter
wing.areas.reference              = 32.488    * Units['meters**2']
wing.twists.root                  = 3.0 * Units.degrees
wing.twists.tip                   = 3.0 * Units.degrees
wing.origin                       = [[32.83 * Units.meter, 0 , 1.14 *
  Units.meter]]
wing.vertical                     = False
wing.symmetric                    = True
wing.dynamic_pressure_ratio       = 0.9

# Add to vehicle
vehicle.append_component(wing)
```

```python

# ------------------------------------------------------------------
#   Vertical Stabilizer
# ------------------------------------------------------------------

wing = SUAVE.Components.Wings.Vertical_Tail()
wing.tag = 'vertical_stabilizer'

wing.aspect_ratio             = 1.91
wing.sweeps.quarter_chord     = 25. * Units.deg
wing.thickness_to_chord       = 0.08
wing.taper                    = 0.25
wing.spans.projected          = 7.777 * Units.meter
wing.chords.root              = 8.19  * Units.meter
wing.chords.tip               = 0.95  * Units.meter
wing.chords.mean_aerodynamic = 4.0   * Units.meter
wing.areas.reference          = 27.316 * Units['meters**2']
wing.twists.root              = 0.0 * Units.degrees
wing.twists.tip               = 0.0 * Units.degrees
wing.origin                   = [[28.79 * Units.meter, 0, 1.54 *
  Units.meter]] # meters
wing.vertical                 = True
wing.symmetric                = False
# The t tail flag is used in weights calculations
wing.t_tail                   = False
wing.dynamic_pressure_ratio   = 1.0

# Add to vehicle
vehicle.append_component(wing)

# ------------------------------------------------------------------
#  Fuselage
# ------------------------------------------------------------------

fuselage = SUAVE.Components.Fuselages.Fuselage()
fuselage.tag = 'fuselage'

# Number of coach seats is used in some weights methods
fuselage.number_coach_seats     = vehicle.passengers
```

```
# The seats abreast can be used along with seat pitch and the number
  of coach seats to
# determine the length of the cabin if desired.
fuselage.seats_abreast          = 6
fuselage.seat_pitch             = 1     * Units.meter
# Fineness ratios are used to determine VLM fuselage shape and
  sections to use in OpenVSP
# output
fuselage.fineness.nose          = 1.6
fuselage.fineness.tail          = 2.
# Nose and tail lengths are used in the VLM setup
fuselage.lengths.nose           = 6.4   * Units.meter
fuselage.lengths.tail           = 8.0   * Units.meter
fuselage.lengths.total          = 38.02 * Units.meter
# Fore and aft space are added to the cabin length if the fuselage
  is sized based on
# number of seats
fuselage.lengths.fore_space     = 6.    * Units.meter
fuselage.lengths.aft_space      = 5.    * Units.meter
fuselage.width                  = 3.74  * Units.meter
fuselage.heights.maximum        = 3.74  * Units.meter
fuselage.effective_diameter     = 3.74     * Units.meter
fuselage.areas.side_projected  = 142.1948 * Units['meters**2']
fuselage.areas.wetted           = 446.718  * Units['meters**2']
fuselage.areas.front_projected = 12.57    * Units['meters**2']
# Maximum differential pressure between the cabin and the atmosphere
fuselage.differential_pressure = 5.0e4 * Units.pascal

# Heights at different longitudinal locations are used in stability
  calculations and
# in output to OpenVSP
fuselage.heights.at_quarter_length          = 3.74 * Units.meter
fuselage.heights.at_three_quarters_length   = 3.65 * Units.meter
fuselage.heights.at_wing_root_quarter_chord = 3.74 * Units.meter

# add to vehicle
vehicle.append_component(fuselage)


# ------------------------------------------------------------------
```

```
#   Nacelles
# ------------------------------------------------------------------
nacelle                              = SUAVE.Components.Nacelles.Nacelle()
nacelle.tag                          = 'nacelle_1'
nacelle.length                       = 2.71
nacelle.inlet_diameter               = 1.90
nacelle.diameter                     = 2.05
nacelle.areas.wetted                 = 1.1*np.pi*nacelle.diameter*nacelle.
  length
nacelle.origin                       = [[13.72, -4.86,-1.9]]
nacelle.flow_through                 = True
nacelle_airfoil                      = SUAVE.Components.Airfoils.Airfoil()
nacelle_airfoil.naca_4_series_airfoil = '2410'
nacelle.append_airfoil(nacelle_airfoil)

nacelle_2                            = deepcopy(nacelle)
nacelle_2.tag                        = 'nacelle_2'
nacelle_2.origin                     = [[13.72, 4.86,-1.9]]

vehicle.append_component(nacelle)
vehicle.append_component(nacelle_2)


# ------------------------------------------------------------------
#   Turbofan Network
# ------------------------------------------------------------------

turbofan = SUAVE.Components.Energy.Networks.Turbofan()
# For some methods, the 'turbofan' tag is still necessary. This will
   be changed in the
# future to allow arbitrary tags.
turbofan.tag = 'turbofan'

# High-level setup
turbofan.number_of_engines = 2
turbofan.bypass_ratio      = 5.4
turbofan.origin            = [[13.72, 4.86,-1.9],[13.72,
  -4.86,-1.9]] * Units.meter

# Establish the correct working fluid
```

```
turbofan.working_fluid = SUAVE.Attributes.Gases.Air()

# Components use estimated efficiencies. Estimates by technology
 level can be
# found in textbooks such as those by J.D. Mattingly

# ------------------------------------------------------------------
#   Component 1 - Ram

# Converts freestream static to stagnation quantities
ram = SUAVE.Components.Energy.Converters.Ram()
ram.tag = 'ram'

# add to the network
turbofan.append(ram)

# ------------------------------------------------------------------
#  Component 2 - Inlet Nozzle

# Create component
inlet_nozzle = SUAVE.Components.Energy.Converters.Compression_Nozzle
 ()
inlet_nozzle.tag = 'inlet_nozzle'

# Specify performance
inlet_nozzle.polytropic_efficiency = 0.98
inlet_nozzle.pressure_ratio        = 0.98

# Add to network
turbofan.append(inlet_nozzle)

# ------------------------------------------------------------------
#  Component 3 - Low Pressure Compressor

# Create component
compressor = SUAVE.Components.Energy.Converters.Compressor()
compressor.tag = 'low_pressure_compressor'

# Specify performance
compressor.polytropic_efficiency = 0.91
```

```
compressor.pressure_ratio        = 1.14

# Add to network
turbofan.append(compressor)

# ------------------------------------------------------------------
#  Component 4 - High Pressure Compressor

# Create component
compressor = SUAVE.Components.Energy.Converters.Compressor()
compressor.tag = 'high_pressure_compressor'

# Specify performance
compressor.polytropic_efficiency = 0.91
compressor.pressure_ratio        = 13.415

# Add to network
turbofan.append(compressor)

# ------------------------------------------------------------------
#  Component 5 - Low Pressure Turbine

# Create component
turbine = SUAVE.Components.Energy.Converters.Turbine()
turbine.tag='low_pressure_turbine'

# Specify performance
turbine.mechanical_efficiency = 0.99
turbine.polytropic_efficiency = 0.93

# Add to network
turbofan.append(turbine)

# ------------------------------------------------------------------
#  Component 6 - High Pressure Turbine

# Create component
turbine = SUAVE.Components.Energy.Converters.Turbine()
turbine.tag='high_pressure_turbine'

```

```
# Specify performance
turbine.mechanical_efficiency = 0.99
turbine.polytropic_efficiency = 0.93

# Add to network
turbofan.append(turbine)

# ------------------------------------------------------------------
#  Component 7 - Combustor

# Create component
combustor = SUAVE.Components.Energy.Converters.Combustor()
combustor.tag = 'combustor'

# Specify performance
combustor.efficiency                = 0.99
combustor.alphac                    = 1.0
combustor.turbine_inlet_temperature = 1450 # K
combustor.pressure_ratio            = 0.95
combustor.fuel_data                 = SUAVE.Attributes.Propellants.
  Jet_A()

# Add to network
turbofan.append(combustor)

# ------------------------------------------------------------------
#  Component 8 - Core Nozzle

# Create component
nozzle = SUAVE.Components.Energy.Converters.Expansion_Nozzle()
nozzle.tag = 'core_nozzle'

# Specify performance
nozzle.polytropic_efficiency = 0.95
nozzle.pressure_ratio        = 0.99

# Add to network
turbofan.append(nozzle)

# ------------------------------------------------------------------
```

```
#  Component 9 - Fan Nozzle

# Create component
nozzle = SUAVE.Components.Energy.Converters.Expansion_Nozzle()
nozzle.tag = 'fan_nozzle'

# Specify performance
nozzle.polytropic_efficiency = 0.95
nozzle.pressure_ratio        = 0.99

# Add to network
turbofan.append(nozzle)

# ------------------------------------------------------------------
#  Component 10 - Fan

# Create component
fan = SUAVE.Components.Energy.Converters.Fan()
fan.tag = 'fan'

# Specify performance
fan.polytropic_efficiency = 0.93
fan.pressure_ratio        = 1.7

# Add to network
turbofan.append(fan)

# ------------------------------------------------------------------
#  Component 11 - thrust (to compute the thrust)

thrust = SUAVE.Components.Energy.Processes.Thrust()
thrust.tag ='compute_thrust'

# Design thrust is used to determine mass flow at full throttle
thrust.total_design            = 2*24000. * Units.N #Newtons

# Add to network
turbofan.thrust = thrust

# Design sizing conditions are also used to determine mass flow
```

```
altitude       = 35000.0*Units.ft
mach_number    = 0.78

# Determine turbofan behavior at the design condition
turbofan_sizing(turbofan,mach_number,altitude)

# Add turbofan network to the vehicle
vehicle.append_component(turbofan)

# ------------------------------------------------------------------
#   Vehicle Definition Complete
# ------------------------------------------------------------------

return vehicle

# 
   ----------------------------------------------------------------------

#   Define the Configurations
# 
   ---------------------------------------------------------------------


def configs_setup(vehicle):
"""This function sets up vehicle configurations for use in different
   parts of the mission.
Here, this is mostly in terms of high lift settings."""

# ------------------------------------------------------------------
#   Initialize Configurations
# ------------------------------------------------------------------
configs = SUAVE.Components.Configs.Config.Container()

base_config = SUAVE.Components.Configs.Config(vehicle)
base_config.tag = 'base'
configs.append(base_config)

# ------------------------------------------------------------------
#   Cruise Configuration
# ------------------------------------------------------------------
```

```
config = SUAVE.Components.Configs.Config(base_config)
config.tag = 'cruise'
configs.append(config)

# ------------------------------------------------------------------
#   Takeoff Configuration
# ------------------------------------------------------------------
config = SUAVE.Components.Configs.Config(base_config)
config.tag = 'takeoff'
config.wings['main_wing'].control_surfaces.flap.deflection = 20. *
  Units.deg
config.wings['main_wing'].control_surfaces.slat.deflection = 25. *
  Units.deg
# A max lift coefficient factor of 1 is the default, but it is
  highlighted here as an option
config.max_lift_coefficient_factor    = 1.

configs.append(config)

# ------------------------------------------------------------------
#   Cutback Configuration
# ------------------------------------------------------------------
config = SUAVE.Components.Configs.Config(base_config)
config.tag = 'cutback'
config.wings['main_wing'].control_surfaces.flap.deflection = 20. *
  Units.deg
config.wings['main_wing'].control_surfaces.slat.deflection = 20. *
  Units.deg
config.max_lift_coefficient_factor    = 1.

configs.append(config)

# ------------------------------------------------------------------
#   Landing Configuration
# ------------------------------------------------------------------

config = SUAVE.Components.Configs.Config(base_config)
config.tag = 'landing'

```

```
config.wings['main_wing'].control_surfaces.flap.deflection = 30. * Units.deg
config.wings['main_wing'].control_surfaces.slat.deflection = 25. * Units.deg
config.max_lift_coefficient_factor    = 1.

configs.append(config)

# ------------------------------------------------------------------
#   Short Field Takeoff Configuration
# ------------------------------------------------------------------

config = SUAVE.Components.Configs.Config(base_config)
config.tag = 'short_field_takeoff'

config.wings['main_wing'].control_surfaces.flap.deflection = 20. * Units.deg
config.wings['main_wing'].control_surfaces.slat.deflection = 20. * Units.deg
config.max_lift_coefficient_factor    = 1.

configs.append(config)

return configs

def simple_sizing(configs):
"""This function applies a few basic geometric sizing relations and modifies the landing
configuration."""

base = configs.base
# Update the baseline data structure to prepare for changes
base.pull_base()

# Revise the zero fuel weight. This will only affect the base configuration. To do all
# configurations, this should be specified in the top level vehicle definition.
base.mass_properties.max_zero_fuel = 0.9 * base.mass_properties.max_takeoff
```

```

# Estimate wing areas
for wing in base.wings:
wing.areas.wetted   = 2.0 * wing.areas.reference
wing.areas.exposed  = 0.8 * wing.areas.wetted
wing.areas.affected = 0.6 * wing.areas.wetted

# Store how the changes compare to the baseline configuration
base.store_diff()

# ------------------------------------------------------------------
#   Landing Configuration
# ------------------------------------------------------------------
landing = configs.landing

# Make sure base data is current
landing.pull_base()

# Add a landing weight parameter. This is used in field length
  estimation and in
# initially the landing mission segment type.
landing.mass_properties.landing = 0.85 * base.mass_properties.
  takeoff

# Store how the changes compare to the baseline configuration
landing.store_diff()

return

#
  ----------------------------------------------------------------------
#   Define the Mission
#
  ----------------------------------------------------------------------

def mission_setup(analyses):
"""This function defines the baseline mission that will be flown by
  the aircraft in order
```

```
to compute performance."""

# ----------------------------------------------------------------------
#   Initialize the Mission
# ----------------------------------------------------------------------

mission = SUAVE.Analyses.Mission.Sequential_Segments()
mission.tag = 'the_mission'

# Airport
# The airport parameters are used in calculating field length and
  noise. They are not
# directly used in mission performance estimation
airport = SUAVE.Attributes.Airports.Airport()
airport.altitude   =  0.0  * Units.ft
airport.delta_isa  =  0.0
airport.atmosphere = SUAVE.Attributes.Atmospheres.Earth.
  US_Standard_1976()

mission.airport = airport

# Unpack Segments module
Segments = SUAVE.Analyses.Mission.Segments

# Base segment
base_segment = Segments.Segment()

# ----------------------------------------------------------------------
#   First Climb Segment: Constant Speed, Constant Rate
# ----------------------------------------------------------------------

# A constant speed, constant rate climb segment is used first. This
  means that the aircraft
# will maintain a constant airspeed and constant climb rate until it
   hits the end altitude.
# For this type of segment, the throttle is allowed to vary as
  needed to match required
# performance.
segment = Segments.Climb.Constant_Speed_Constant_Rate(base_segment)
```

```
    # It is important that all segment tags must be unique for proper evaluation. At the moment
    # this is not automatically enforced.
    segment.tag = "climb_1"

    # The analysis settings for mission segment are chosen here. These analyses include information
    # on the vehicle configuration.
    segment.analyses.extend( analyses.takeoff )

    segment.altitude_start = 0.0   * Units.km
    segment.altitude_end   = 3.0   * Units.km
    segment.air_speed      = 125.0 * Units['m/s']
    segment.climb_rate     = 6.0   * Units['m/s']

    # Add to misison
    mission.append_segment(segment)

    # ------------------------------------------------------------------
    #   Second Climb Segment: Constant Speed, Constant Rate
    # ------------------------------------------------------------------

    segment = Segments.Climb.Constant_Speed_Constant_Rate(base_segment)
    segment.tag = "climb_2"

    segment.analyses.extend( analyses.cruise )

    # A starting altitude is no longer needed as it will automatically carry over from the
    # previous segment. However, it could be specified if desired. This would potentially cause
    # a jump in altitude but would otherwise not cause any problems.
    segment.altitude_end   = 8.0   * Units.km
    segment.air_speed      = 190.0 * Units['m/s']
    segment.climb_rate     = 6.0   * Units['m/s']

    # Add to mission
    mission.append_segment(segment)

    # ------------------------------------------------------------------
```

```
#   Third Climb Segment: constant Speed, Constant Rate
# ------------------------------------------------------------------

segment = Segments.Climb.Constant_Speed_Constant_Rate(base_segment)
segment.tag = "climb_3"

segment.analyses.extend( analyses.cruise )

segment.altitude_end = 10.668 * Units.km
segment.air_speed    = 226.0  * Units['m/s']
segment.climb_rate   = 3.0    * Units['m/s']

# Add to mission
mission.append_segment(segment)

# ------------------------------------------------------------------
#   Cruise Segment: Constant Speed, Constant Altitude
# ------------------------------------------------------------------

segment = Segments.Cruise.Constant_Speed_Constant_Altitude(
  base_segment)
segment.tag = "cruise"

segment.analyses.extend( analyses.cruise )

segment.air_speed  = 230.412 * Units['m/s']
segment.distance   = 2490. * Units.nautical_miles

# Add to mission
mission.append_segment(segment)

# ------------------------------------------------------------------
#   First Descent Segment: Constant Speed, Constant Rate
# ------------------------------------------------------------------

segment = Segments.Descent.Constant_Speed_Constant_Rate(base_segment
  )
segment.tag = "descent_1"

segment.analyses.extend( analyses.cruise )
```

```

segment.altitude_end = 8.0   * Units.km
segment.air_speed    = 220.0 * Units['m/s']
segment.descent_rate = 4.5   * Units['m/s']

# Add to mission
mission.append_segment(segment)

# ------------------------------------------------------------------
#   Second Descent Segment: Constant Speed, Constant Rate
# ------------------------------------------------------------------

segment = Segments.Descent.Constant_Speed_Constant_Rate(base_segment
 )
segment.tag = "descent_2"

segment.analyses.extend( analyses.landing )

segment.altitude_end = 6.0   * Units.km
segment.air_speed    = 195.0 * Units['m/s']
segment.descent_rate = 5.0   * Units['m/s']

# Add to mission
mission.append_segment(segment)

# ------------------------------------------------------------------
#   Third Descent Segment: Constant Speed, Constant Rate
# ------------------------------------------------------------------

segment = Segments.Descent.Constant_Speed_Constant_Rate(base_segment
 )
segment.tag = "descent_3"

segment.analyses.extend( analyses.landing )
# While it is set to zero here and therefore unchanged, a drag
 increment can be used if
# desired. This can avoid negative throttle values if drag generated
  by the base airframe
# is insufficient for the desired descent speed and rate.
analyses.landing.aerodynamics.settings.spoiler_drag_increment = 0.00
```

```

segment.altitude_end = 4.0   * Units.km
segment.air_speed    = 170.0 * Units['m/s']
segment.descent_rate = 5.0   * Units['m/s']

# Add to mission
mission.append_segment(segment)

# ------------------------------------------------------------------
#   Fourth Descent Segment: Constant Speed, Constant Rate
# ------------------------------------------------------------------

segment = Segments.Descent.Constant_Speed_Constant_Rate(base_segment
 )
segment.tag = "descent_4"

segment.analyses.extend( analyses.landing )
analyses.landing.aerodynamics.settings.spoiler_drag_increment = 0.00

segment.altitude_end = 2.0   * Units.km
segment.air_speed    = 150.0 * Units['m/s']
segment.descent_rate = 5.0   * Units['m/s']

# Add to mission
mission.append_segment(segment)

# ------------------------------------------------------------------
#   Fifth Descent Segment: Constant Speed, Constant Rate
# ------------------------------------------------------------------

segment = Segments.Descent.Constant_Speed_Constant_Rate(base_segment
 )
segment.tag = "descent_5"

segment.analyses.extend( analyses.landing )
analyses.landing.aerodynamics.settings.spoiler_drag_increment = 0.00

segment.altitude_end = 0.0   * Units.km
segment.air_speed    = 145.0 * Units['m/s']
segment.descent_rate = 3.0   * Units['m/s']
```

```

# Append to mission
mission.append_segment(segment)

# ------------------------------------------------------------------
#   Mission definition complete
# ------------------------------------------------------------------

return mission

def missions_setup(base_mission):
"""This allows multiple missions to be incorporated if desired, but
  only one is used here."""

# Setup the mission container
missions = SUAVE.Analyses.Mission.Mission.Container()

# ------------------------------------------------------------------
#   Base Mission
# ------------------------------------------------------------------

# Only one mission (the base mission) is defined in this case
missions.base = base_mission

return missions

#
  ----------------------------------------------------------------------

#   Plot Mission
#
  ----------------------------------------------------------------------


def plot_mission(results,line_style='bo-'):

# Plot Altitude, sfc, vehicle weight
plot_altitude_sfc_weight(results, line_style) #DONE

# Plot Velocities
```

```
plot_aircraft_velocities(results, line_style) #DONE

plot_fuel_use(results, line_style) #DONE

# Plot Aerodynamic Coefficients
# plot_aerodynamic_coefficients(results, line_style)  #DONE

# Plot Aerodynamic Forces
# plot_aerodynamic_forces(results, line_style) #DONE

# Drag Components
plot_drag_components(results, line_style) #DONE

# Plot Flight Conditions
plot_flight_conditions(results, line_style) #DONE

plot_flight_trajectory(results, line_style) #DONE

# plot_stability_coefficients(results, line_style) #DONE

return

# This section is needed to actually run the various functions in the file
if __name__ == '__main__':
main()
# The show commands makes the plots actually appear
plt.show()
```

3.10.4 Analysis of Results

The simulation's output provides an extensive overview of how the aircraft performs under different circumstances. These consist of weight change, specific fuel consumption (SFC), altitude variations, drag components, fuel consumption, and velocity profiles.

Model Construction of Boeing 737 within SUAVE Framework

The Boeing 737's model assembly within SUAVE framework is shown in Figure 3.8. Fuselage size, wing arrangement with particular airfoil selection, empennage, and the location and kind of propulsion systems are important components that are shown. The labeling of each part emphasizes how it contributes to the overall aerodynamics and performance of the aircraft, providing a clear model of Boeing 737 is designed for simulation.

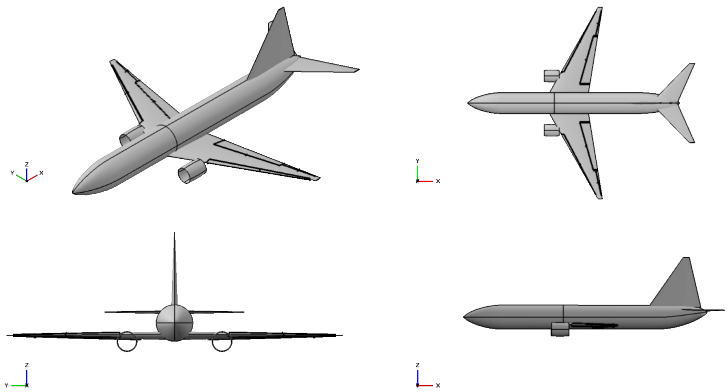
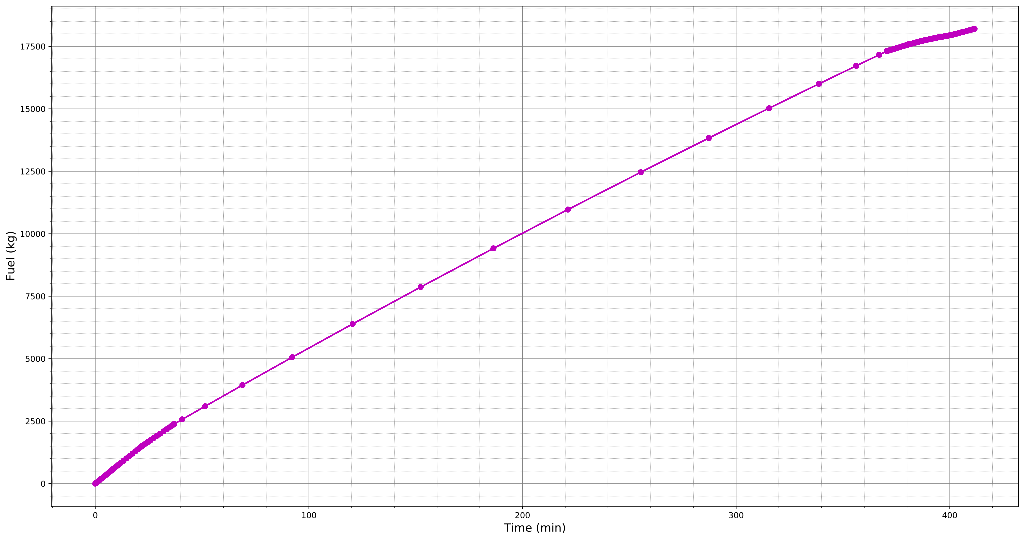


FIGURE 3.8: Boeing 737 Model Construction Using SUAVE

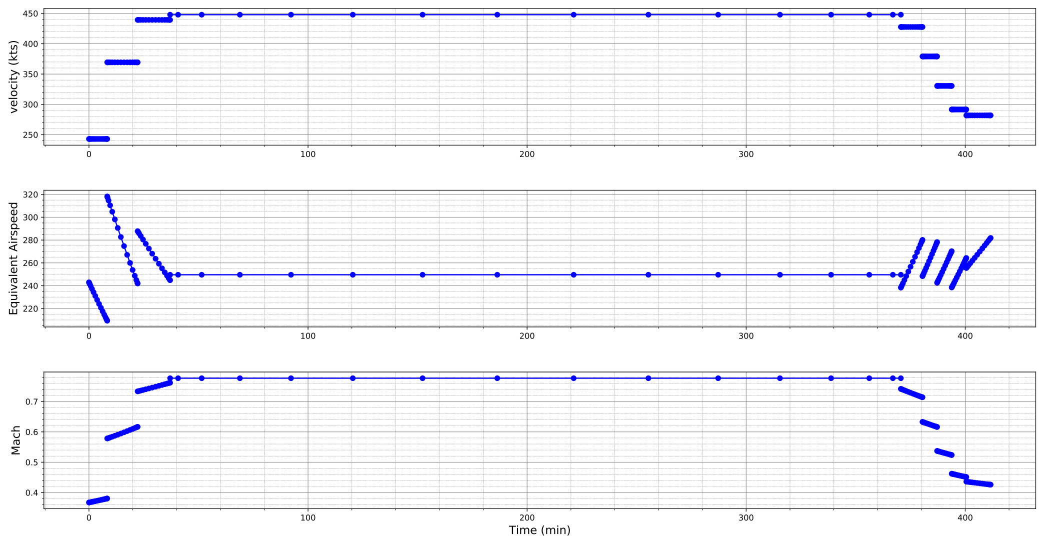
Result Interpretation

1. Fuel Consumption: According to the simulation, the aircraft would use about 18,200 kg of fuel overall during the flight, as shown in Figure 3.9a.
2. Velocity and Mach Number: Aircraft max speed of 450 knots and Mach values between 0.4 and 0.7 indicates that it was operating at average cruise speeds for commercial jet operations, as shown in Figure 3.9b.
3. Altitude and Specific Fuel Consumption: Effective high-altitude cruise operations were demonstrated by the aircraft's ability to maintain altitudes of up to 35,000 feet and its specific fuel consumption, which peaked at about 0.625 lb/lb-f hr, as shown in Figure 3.10a.
4. Weight Reduction: Fuel consumption was the main cause of the aircraft's weight dropping from 174,000 lb to 134,000 lb throughout the flight, as shown in Figure 3.10a.

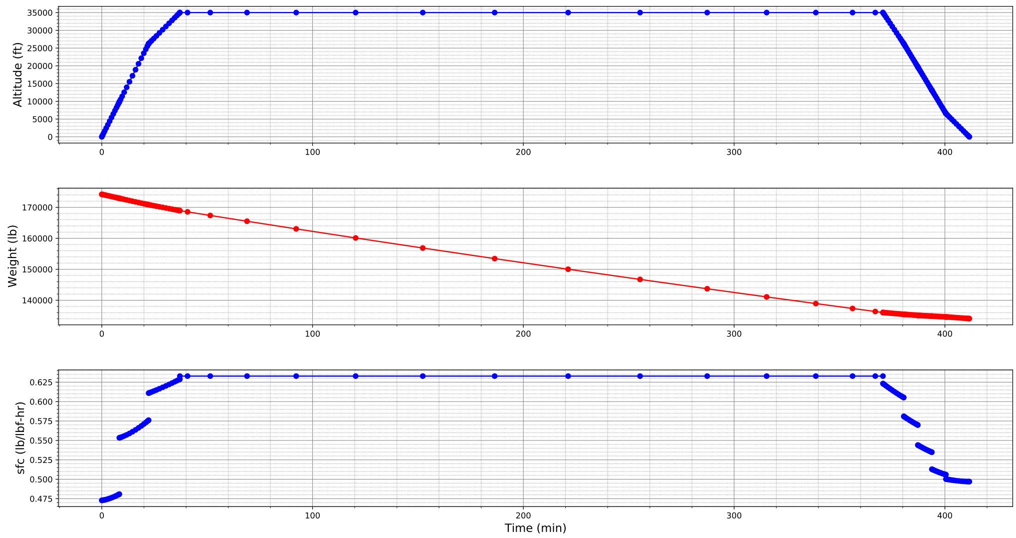
5. Drag Components: Total drag coefficients for this aircraft type stayed within the expected ranges, despite across many components, including compressibility, induced drag, and parasite drags, according to the drag analysis, as shown in Figure 3.10b.
6. Flight Conditions: The simulation achieves a cruising height of 35,000 feet and airspeeds between 300 and 500 mph, flight range of almost 3000 nautical miles, as shown Figure 3.11a
7. Flight Trajectory: A throughout flight trajectory was plotted by the simulation including positional coordinates and altitude variations during the mission, as shown in Figure 3.11b.



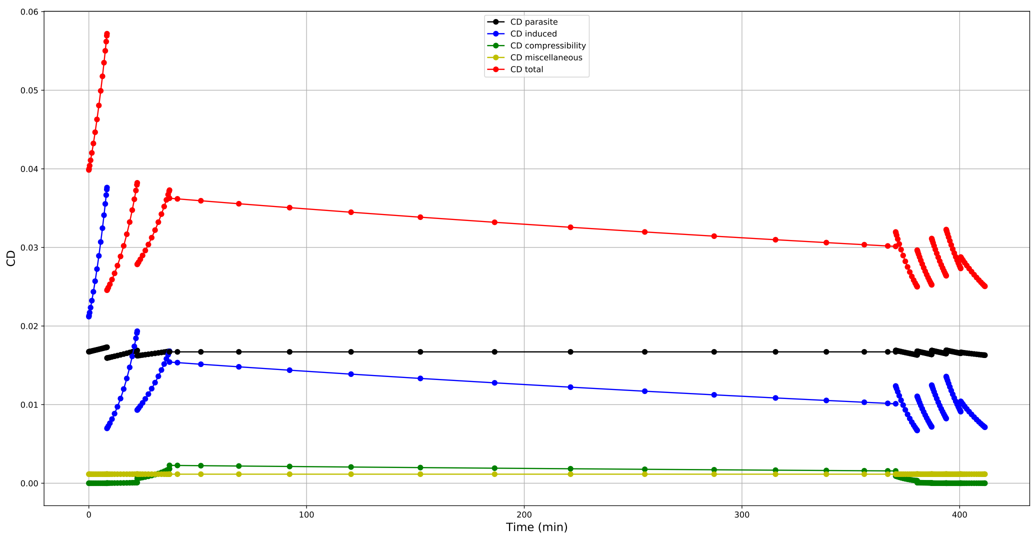
(A) Aircraft Fuel Burnt



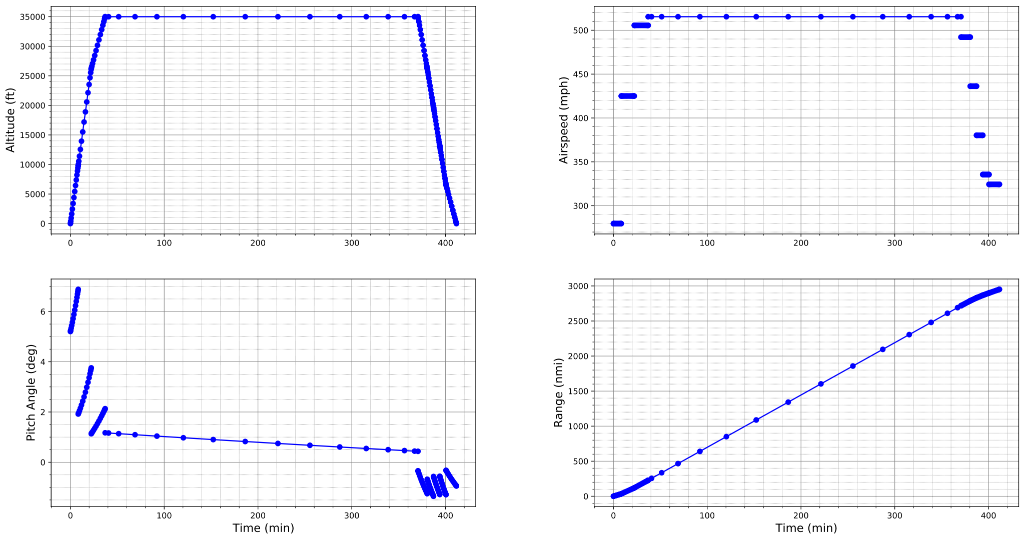
(B) Aircraft Velocities



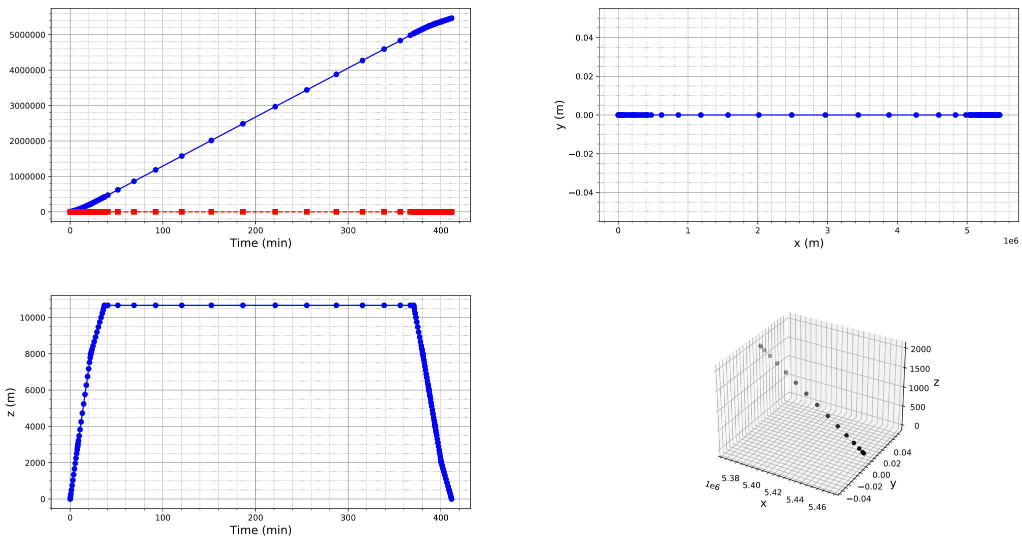
(A) Altitude, SFC, Weight



(B) Drag Components



(A) Flight Conditions



(B) Flight Trajectory

3.10.5 Conclusion

This thorough simulation of Boeing 737 using SUAVE highlights the software's ability and realistically model and analyze aircraft performance. The effectiveness of SUAVE as a useful tool for aircraft engineering applications is indirectly validated by the alignment of the simulated findings with know performance parameters, of the Boeing 737. In the following sections of this thesis, the insights obtained from this case are used to support the use of SUAVE in aircraft performance design and optimization.

TABLE 3.1: List of Aircraft and Specifications

Aircraft Name	**MTOW (kg)**	**EMTOW (kg)**	**Payload (kg)**	**Fuel Mass (kg)**	**Ferry Range (km)**	**Service Ceiling (m)**	**Max Speed**
F-16 Fighting Falcon	19,187	8,573	4,470	3,175 (Int) 2×5,334 (Ext)	4,217	15,000	Mach 2.05
MiG-29 Fulcrum	18,000	11,000	4,000	3,500	2,100	18,000	Mach 2.3+
Saab Gripen (JAS 39C)	14,000	6,800	5,300	2,340 (Int) 2,730 (Ext)	3,200	15,240	Mach 2
F-22 Raptor	38,000	19,700	2,270	8,200 (Int) 2×600 gal (Ext)	3,220	20,000	Mach 2.25
F-35A Lightning II	31,800	13,154	8,160	8,278	2,200	15,000	Mach 1.6
Sukhoi Su-57	37,000	18,500	8,000	10,300	4,500	20,000	Mach 2
Chengdu J-20	37,000	17,000	11,000	12,000	5,500	20,000	Mach 2
Eurofighter Typhoon	23,500	11,000	9,000	4,996	3,790	19,182	Mach 2.35
Dassault Rafale C	24,500	9,850	9,500	4,700	3,700	15,835	Mach 1.8
Sukhoi Su-35	34,500	19,000	8,000	11,500	3,600	18,000	Mach 2.25
Boeing F-15EX Eagle II	36,741	15,694	13,400	16,125	3,900	18,000	Mach 2.5

CHAPTER 4
RESULTS AND DISCUSSIONS

4.1 Exporting OpenVSP File

Exporting the aircraft model to OpenVSP, which is an application for additional aero-structural assessment and visualization, was a crucial step in confirming the F-16's design performance analysis. This Procedure made it easier to move from the SUAVE environment's to OpenVSP more easier and visual analysis.

4.1.1 Process Overview

Determining which vehicle or aircraft to be analyzed is the first step in SUAVE. F-16 was chosen because of its proven ability to be modernized, as shown by its modernization into the QF-16 for unmanned missions, as was covered in Section 2.5.1. The availability of reliable OpenVSP model, which is essential for guaranteeing precise data translation and parameter verification within SUAVE framework. The result of the study could be affected if any parameter in SUAVE is configured incorrectly, highlighting the significance of precise model building.

F-16 was carefully prepared for export following the definition of the aircraft model and the conclusion of simulation in SUAVE. In order to guarantee accuracy and consistency within OpenVSP framework, all aerodynamic, structural data had to be finalized.

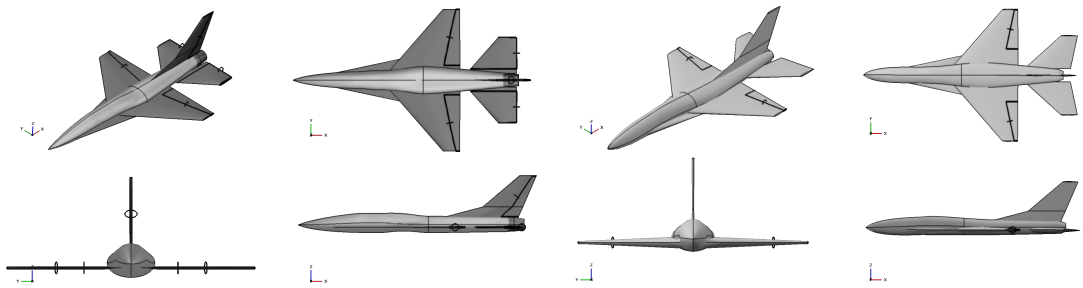
Technical Steps

1. Finding the Proper F-16 Model: It was essential to find an F-16 model in OpenVSP that had appropriate parameters. This guarantees that the simulations' baseline is accurate and follows established aerodynamic profiles.

2. SUAVE Parameter Adjustments: OpenVSP model is used to fine-tune and modify SUAVE's settings. To guarantee consistency between the two platforms, this stage involves comparing the OpenVSP model's parameters with SUAVE's.
3. SUAVE Model Preparation: Making sure F-16 model in SUAVE has all required adjustments and corresponded to OpenVSP compatible data formats.
4. Data Conversion and Export: By utilizing SUAVE's built-in features, the detailed model of the aircraft may be converted into an OpenVSP file while maintaining all geometrical and performance related characteristics.
5. Verification in SUAVE: It was crucial to confirm that the vehicle was properly built and that all parameters were applied after constructing the model with SUAVE.
6. Verification in OpenVSP: The model was inspected in OpenVSP after exporting to verify for any design inconsistencies or problems with data translation. Making sure the visual depiction matched the computational model required in this step.

Limitations Constructing Vehicle in SUAVE

There are a lot of limitations when using SUAVE to design a vehicle, especially when it comes to representing aerodynamics structures. For instance, ”skinning” the fuselage, a feature that SUAVE does not have but OpenVSP does, which allows for a more aerodynamically pointed and streamlined fuselage design. Modeling engine nacelles presents another difficulty. This study analyzing the shape and detail changes between the SUAVE build and the F-16 model from the original file to show how these differences could affect aerodynamic calculations. Additionally, there may be a number of differences when importing data from SUAVE to OpenVSP, including inthe control surfaces, wing segments, and fuselage segments. Due to these variations, users must switch from OpenVSP to SUAVE and modify the models by trial and error. Differences between the original OpenVSP model and model that is constructed using SUAVE framework can be seen in Figure 4.1.



(A) F-16 Generated by OpenVSP [43]

(B) F-16 Generated by SUAVE

FIGURE 4.1: Comparison F-16 Model

Importance of the Export

Figure 4.1, displays the F-16 Fighting Falcon's detailed three-dimensional model, which was obtained from the VSP Airshow, an official repository on the OpenVSP website [43]. Exporting to OpenVSP made it possible to perform further analysis including drag prediction, lift distribution, and structural integrity under simulated operating conditions, however in this study OpenVSP does not use any analysis from OpenVSP, in addition to improving the visualization of the F-16's model and operational envelopes.

By offering a thorough platform for in-depth design evaluation and improvement, this stage is essential for bridging the gap between theoretical simulations. This procedure highlights SUAVE and OpenVSP ability to work together and their mutual benefit in the iterative design and assessment cycle of aerospace engineering projects.

4.2 Mission Configuration

Setting particular aerodynamic and operational parameters that corresponds to the requirement of each mission type as specified by the AIAA was necesssary to configure the F-16 for different mission profiles inside the SUAVE framework. The setup proceudre and associated simulation results for the Defenseive Combat-Air Patrol (DCA), Point Defense Intercept (PDI), and Intercept/Escort missions.

4.2.1 Technical Adjustment and Simulation

To accurately represent the operational demands and guarantee that the model's was optimized for its intended mission, each mission profile required unique technological modifications in the SUAVE model. Altitude, air speed, climb rate, time, Mach number, distance, and descent rate were all carefully adjusted for this.

Aerodynamic Profiling

To guarantee that the aircraft could satisfy the various requirements of each mission type, modification to the aerodynamic profiles were necessary, paying special attention to:

- Altitude: Determining precise operating ceilings for every mission in order to optimize effectiveness and efficiency.
- Air Speed: Determining ideal speeds for takeoff, cruising, and among other mission segments.
- Climb Rate: This should be set up to acquire altitude quickly, which is essential for intercept mission.
- Time and Distance: Making sure the aircraft can go the required distances and loiter time.

Fuel and Weight Management

It was essential to strategically control the fuel load and distribution, especially to balance the aircraft's center of gravity, which influences speed and agility. Based on the mission profile, modifications were also made to account for different fuel requirements, including:

- Mach Number: Ensuring the aircraft can function well at high speeds by adjusting for ideal supersonic performance when necessary.
- Descent Rate: Carefully arranging the descent to provide a safe and effective transfer to lower operational altitudes or return to base.

Mission Validation

By simulating and assessing these parameters using SUAVE's built-in features, a thorough analysis of the aircraft's performance in various circumstances is provided, which includes:

- Simulation Runs: To guarantee the correctness and dependability of the outcomes, each configuration was put through several iterations.
- Error Analysis and Resolution: In order to ensure that every mission profile could be carried out effectively and in accordance with plan, any inconsistencies or errors in mission simulations required fast analysis to find and fix the problems.

Defensive Combat-Air Patrol (DCA) Mission

DCA mission profile was changed to remove complicated combat maneuvers from SUAVE's simulation due to its limitations, leaving only patrol endurance and response capabilities, Table 4.1 shown the missions segments that can simulated in SUAVE's framework. To compensate the removal, we offset the fuel mass equivalent to the combat maneuver budget as new constraint.

TABLE 4.1: Defensive Counter-Air Patrol Mission Phases Description

Phase	Description
1	Take-off and acceleration
2	Climb from sea level to optimum cruise altitude
3	Cruise out 300 nm at optimum speed and altitude
4	Combat air patrol for 4 hours at best loiter speed at 35,000 ft
5	Dash 100 nm at maximum speed at 35,000 ft
6	Climb/accelerate to optimum speed and altitude
7	Cruise back 400 nm at optimum speed and altitude
8	Descend to sea level
9	Reserves: Fuel for 30 minutes at sea level at maximum endurance speed

Point Defense Intercept (PDI) Mission

PDI mission profile was also altered to remove combat maneuvers from the simulation shown in Table 4.2. Similar to DCA mission reserves fuel will be use for the combat maneuver budget.

TABLE 4.2: Point Defense Intercept Mission Phases Description

Phase	Description
1	Take-off and acceleration
2	Climb from sea level to 35,000 ft and accelerate to maximum speed
3	Dash 200 nm at maximum speed at 35,000 ft
4	Climb/accelerate to optimum speed and altitude
5	Cruise back 200 nm at optimum speed and altitude
6	Descent to sea level
7	Reserves: Fuel for 30 minutes at sea level at maximum endurance speed

Intercept/Escort Mission

SUAVE framework successfully specified the IE missions, accounting for each mission component as a function in the framework, therefore there are no modifications to the original mission profile unlike DCA and PDI.

4.3 Mission Configuration

4.3.1 Limitations

Considering the limitations listed in Chapter 3.9:

- Combat Maneuvers: Fuel requirements and aerodynamic pressures of complex combat maneuvers are crucial for accurate interceptor mission simulations, but SUAVE's current version does not offer proper function for combat maneuvers given by AIAA.
- Multi-Wing Configurations: Analysis of aircraft like F-16 use complex wing shapes, which is the wing stake in front of the main wing, for improved

aerodynamic efficiency may be impacted by challenges in precisely configuring multiple wing configurations in SUAVE.

Defensive Combat-Air Patrol (DCA) Mission

1. Fuel Burn:

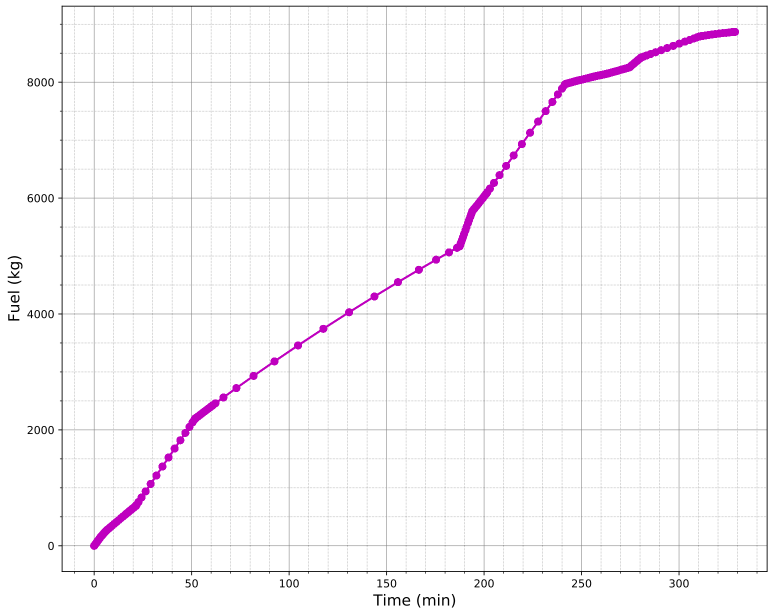


FIGURE 4.2: Aircraft Fuel Burnt

- Figure 4.2, which runs from 0 to 328 minutes, shows the trend of fuel usage during the flight. The steady increase in fuel consumption is indicative of the normal fuel burn profile during long flight phases, such as climbing, cruise, descent, and reserves loiter. The aircraft uses fuel in a nearly linear pattern from the start of the journey until the finish, using 8,869 kg of fuel in total.

2. Aircraft Velocities:

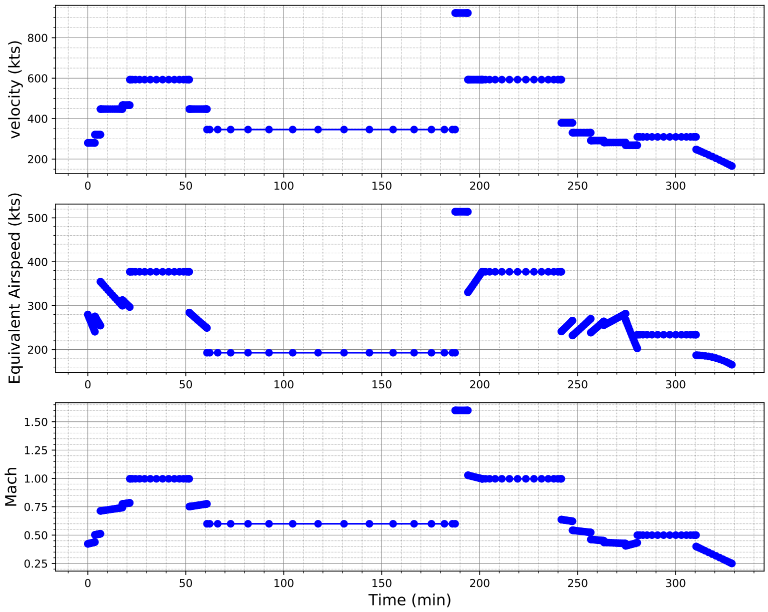


Figure 4.3: Aircraft Velocities

- Over the mission, Figure 4.3 shows true airspeed, equivalent airspeed, and Mach number. As the aircraft starts to descend, the true airspeed progressively drops after rising to a peak about 920 knots, which roughly Mach 1.4, which is probably indicates the dash conditions.

3. Altitude, Specific Fuel Consumption (SFC), and Weight Analysis:

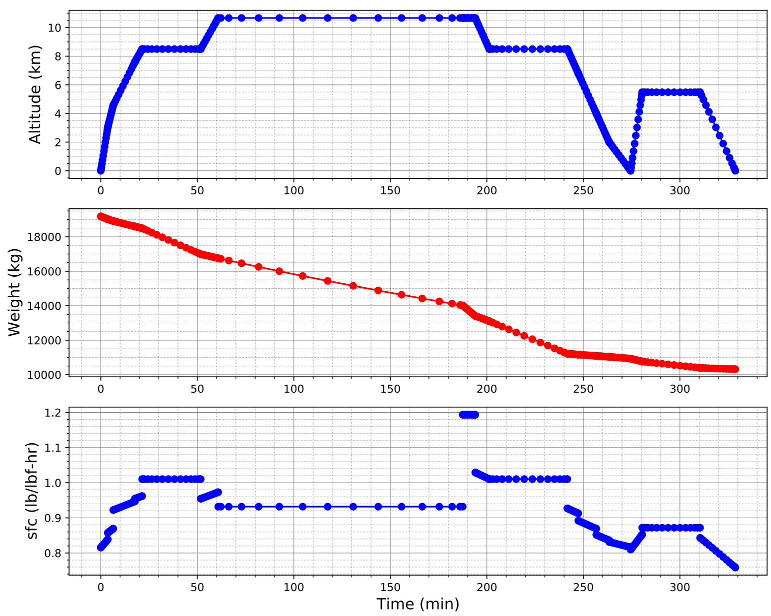


Figure 4.4: Altitude, SFC, Weight

- Figure 4.4 provied a thorough analysis of the altitude, peaking at about 10.6 km or 35,000 feet. Engine efficiency is shown by the SFC, which varies somewhat but stays at about 1.0lb/lb-hr throughout the flight profile. The vehicle function specifies 19,200 kg as the Maximum Takeoff Weight (MTOW) of the aircraft. By the en of the mission, this weight drops to about 10,320 kg, mostly as a result of fuel use. The Maximum Zero Fuel Weight (MZFW), which is 10,360 kg, is the limit weight that an aircraft can have and still complete its mission without refueling. Thus, the aircraft cannot be used for patrolling more than 7,600 seconds (2.1 hours).
- Since there are no useful fuel left onboard, it is not feasible for F-16 to perform combat maneuver when doing DCA mission.

4. Drag Analysis:

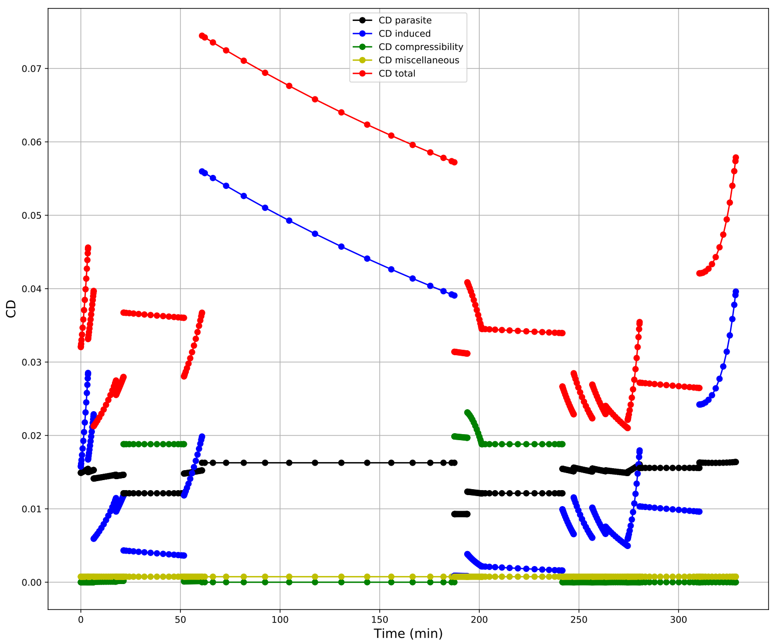


FIGURE 4.5: Drag Components

- Figure 4.5 shows the several types of drag, including parasite, induced, compressibility, miscellaneous, and total drag.

5. Flight Conditions Analysis:

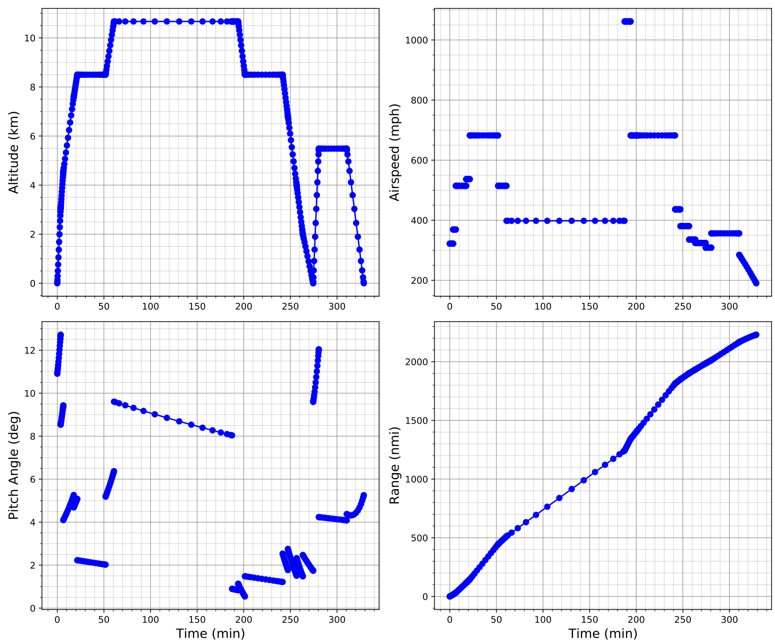


FIGURE 4.6: Flight Conditions

- Altitude, airspeed, pitch angle, and range are shown in Figure 4.6. The maximum airspeed is around 1,060 mph which is around 921 knots, and range of 2,220 nmi or around 4,100 km.

6. Flight Trajectory Analysis:

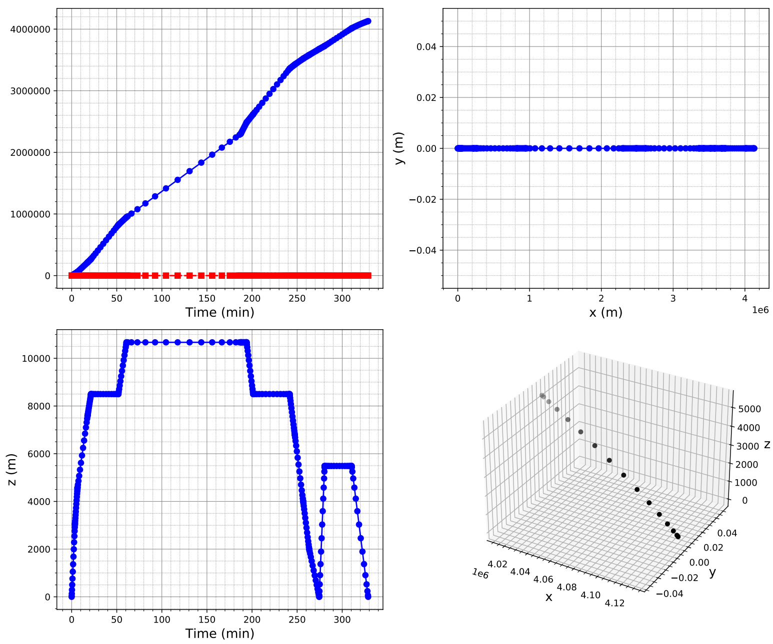


FIGURE 4.7: Flight Trajectory

- The aircraft's path represented by x,y,z coordinates over time and as well as 3D dimensions on the trajectory graph are plotted in Figure 4.7.

Extended Mission Feasibility Analysis

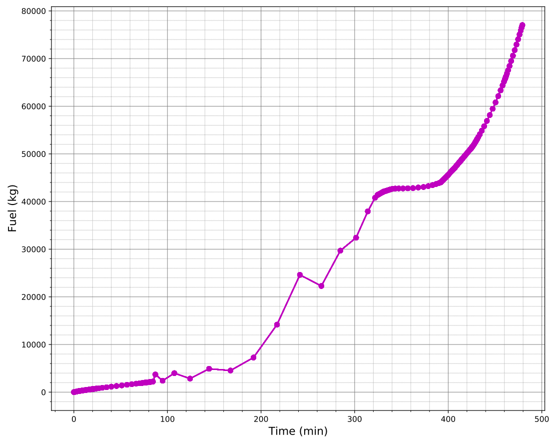
The possibility of increasing F-16's Defensive Counter-Air (DCA) mission duration to four hours is examined in this section. Due to fuel restrictions limit, the current operating settings do not permit such a long period. The purpose of this analysis is to suggest and assess possible changes to the mission profiles that would allow the aircraft to accomplish this longer mission duration.

Detailed Analysis of Each Scenario

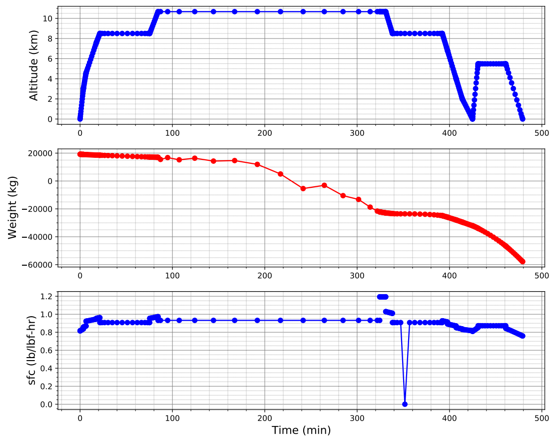
A brief overview of each scenario is given, highlighting the particular operational factors or objectives that define each, before going into details on specific results. This background information helps comprehension of the following analyses:

1. Scenario 1: Parameter changed in this scenario are:
   - max_payload = 850 kg

- air_speed = 171.5 m/s in cruise_out segment
- air_speed = 171.5 m/s in cruise_back segment
- distance = 300 nmi in cruise_back segment



(A) Aircraft Fuel Burnt



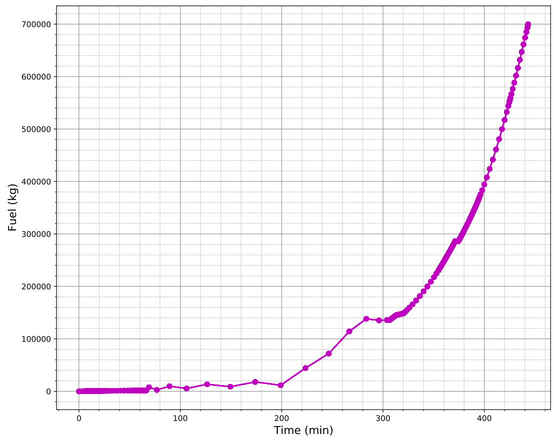
(B) Altitude, SFC, Weight

FIGURE 4.8: Results from Scenario 1

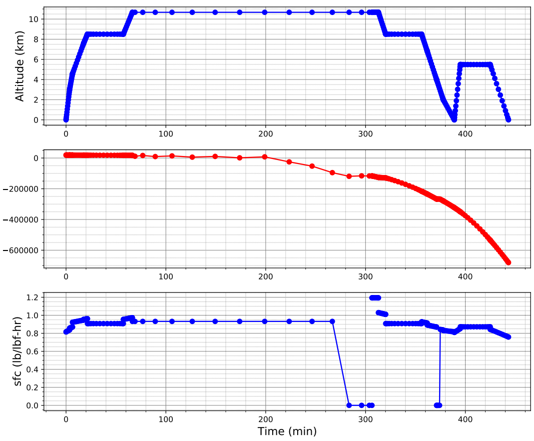
Based on Figure 4.8, the aircraft burns about 77,000 kg of fuel in 480 minutes. At the same time the weight decreases from 19,185 kg to -57,878 kg. This implies an impractical situation in which the computed weight is less than zero. Because it suggests that the aircraft would need more fuel than it could physically carry.

2. Scenario 2: Parameter changed in this scenario are:
   - max_payload = 850 kg

- air_speed = 171.5 m/s in cruise_out segment
- distance = 200 nmi in cruise_out segment
- air_speed = 171.5 m/s in cruise_back segment
- distance = 200 nmi in cruise_back segment



(A) Aircraft Fuel Burnt



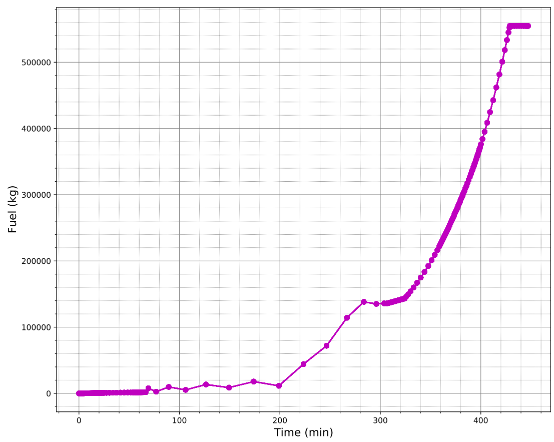
(B) Altitude, SFC, Weight

FIGURE 4.9: Results from Scenario 2

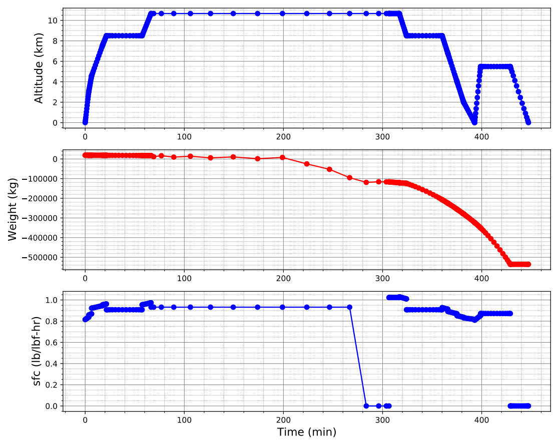
Based on Figure 4.9, the aircraft burns about 700,000 kg of fuel in 442 minutes. At the same time the weight decreases from 19,185 kg to -680,388 kg. Similar to the first scenario, the aircraft would need more fuel than it could physically carry.

3. Scenario 3: Parameter changed in this scenario are:
    - max_payload = 850 kg

- air_speed = 171.5 m/s in cruise_out segment
- distance = 200 nmi in cruise_out segment
- air_speed = 171.5 m/s in cruise_back segment
- distance = 200 nmi in cruise_back segment
- mach = 1 in dash segment



(A) Aircraft Fuel Burnt



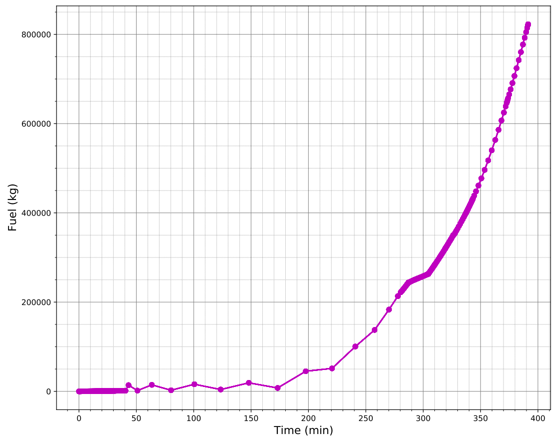
(B) Altitude, SFC, Weight

FIGURE 4.10: Results from Scenario 3

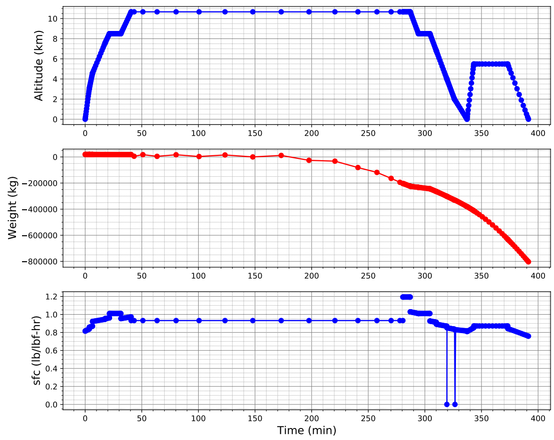
Based on Figure 4.10, the aircraft burns about 555,000 kg of fuel in 447 minutes. At the same time the weight decreases from 19,185 kg to -535,239 kg. Similar to the first scenario, the aircraft would need more fuel than it could physically carry.

4. Scenario 4: Parameter changed in this scenario are:

- distance = 100 nmi in cruise_out segment
- distance = 100 nmi in cruise_back segment



(A) Aircraft Fuel Burnt

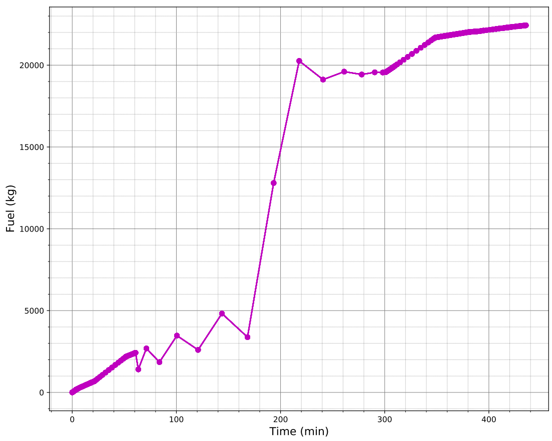


(B) Altitude, SFC, Weight

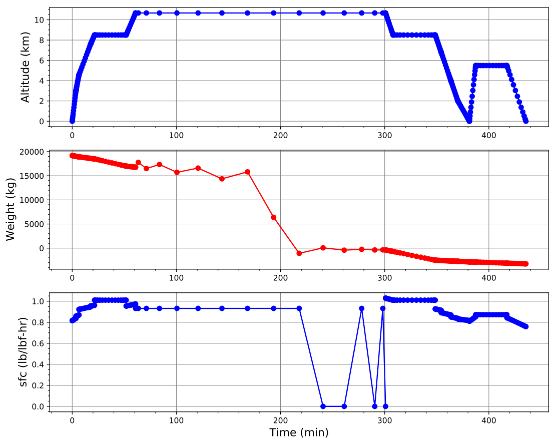
FIGURE 4.11: Results from Scenario 4

Based on Figure 4.11, the aircraft burns about 822,660 kg of fuel in 391 minutes. At the same time the weight decreases from 19,185 kg to -803,471 kg. Similar to the first scenario, the aircraft would need more fuel than it could physically carry.

5. Scenario 5: Parameter changed in this scenario are:
    - completely removed dash segment



(A) Aircraft Fuel Burnt

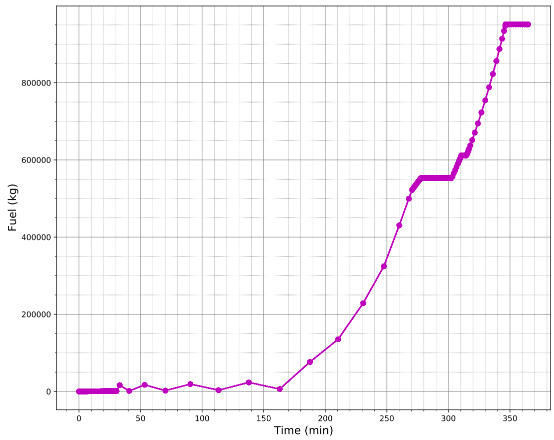


(B) Altitude, SFC, Weight

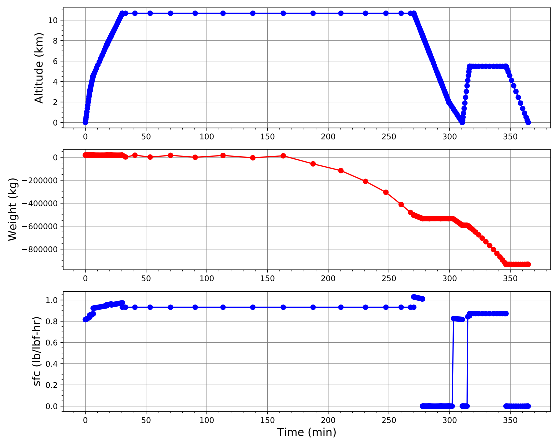
FIGURE 4.12: Results from Scenario 5

Based on Figure 4.12, the aircraft burns about 22,434 kg of fuel in 435 minutes. At the same time the weight decreases from 19,185 kg to -3,288 kg. Similar to the first scenario, the aircraft would need more fuel than it could physically carry.

6. Scenario 6: Parameter changed in this scenario are:
   - completely removed dash segment
   - completely removed cruise_out segment
   - completely removed cruise_back segment



(A) Aircraft Fuel Burnt



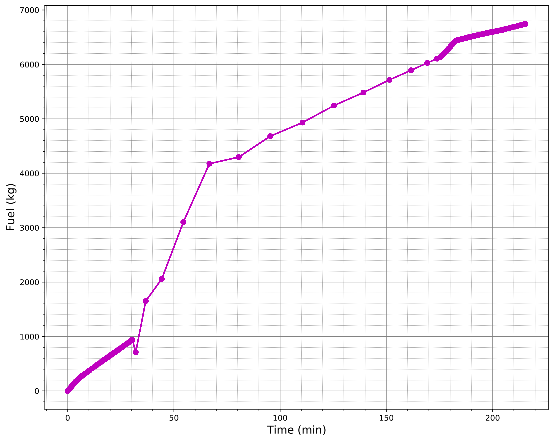
(B) Altitude, SFC, Weight

FIGURE 4.13: Results from Scenario 6

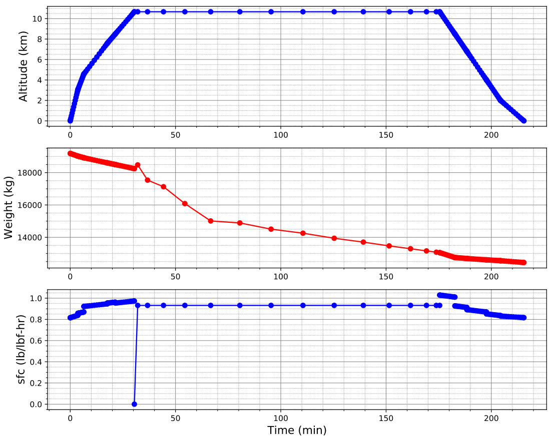
Based on Figure 4.13, the aircraft burns about 952,000 kg of fuel in 365 minutes. At the same time the weight decreases from 19,185 kg to -934,400 kg. Similar to the first scenario, the aircraft would need more fuel than it could physically carry.

7. Scenario 7: Parameter changed in this scenario are:
   - max_payload = 850 kg
   - completely removed dash segment
   - completely removed cruise_out segment
   - completely removed cruise_back segment
   - patrol time 8,700 seconds

- completely removed reserves segment



(A) Aircraft Fuel Burnt



(B) Altitude, SFC, Weight

FIGURE 4.14: Results from Scenario 7

Based on Figure 4.14, the aircraft burns about 6,758 kg of fuel in 215 minutes. At the same time the weight decreases from 19,185 kg to 12,428 kg. In this scenario there are around 2,068 kg fuel left, therefore we could get the fuel fraction and use it for combat maneuver. Fuel fraction for this scenario is 0.107, it is still not feasible to do combat maneuver because the usual fuel fraction to do combat maneuver is around 0.290 based on [44].

Point Defense Intercept (PDI) Mission

1. Fuel Burn:

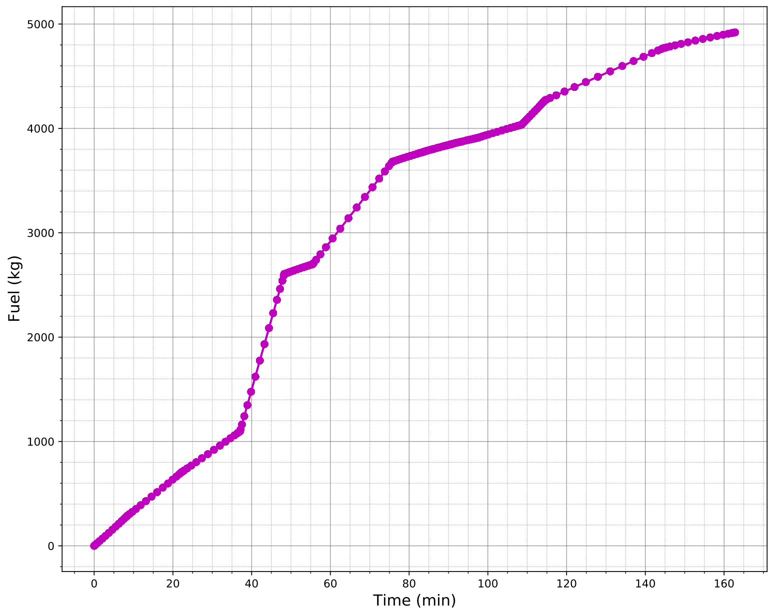


Figure 4.15: Aircraft Fuel Burnt

- Starting at 0 kg and rising to 4,920 kg in about 162 minutes, the fuel consumption graph shows a linear increase over time is shown in Figure 4.15.

2. Aircraft Velocities:

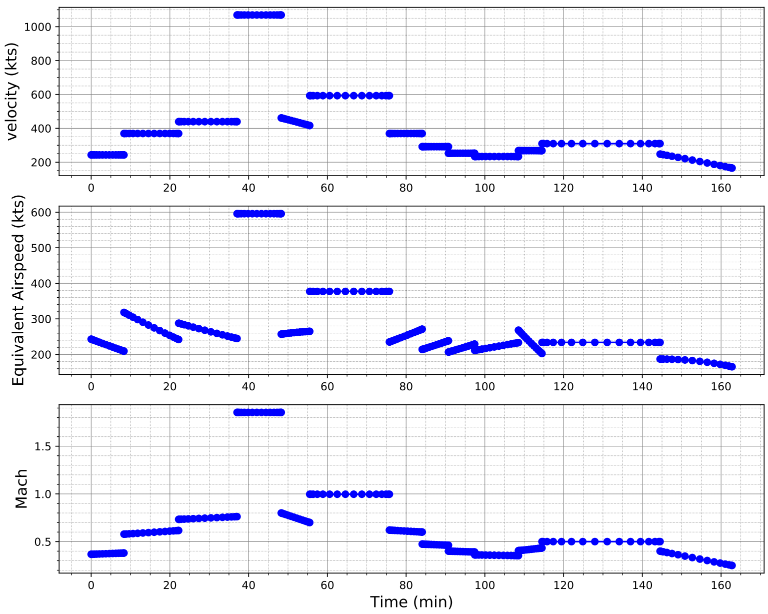


Figure 4.16: Aircraft Velocities

- Over the mission, the graph shows true airspeed, equivalent airspeed, and Mach number, is shown in Figure 4.16. When the aircraft reaches its dash segment, which is probably around 1,069 knots or Mach 1.6, the true airspeed displays a pattern of increasing velocity.

3. Altitude, Specific Fuel Consumption (SFC), and Weight Analysis:

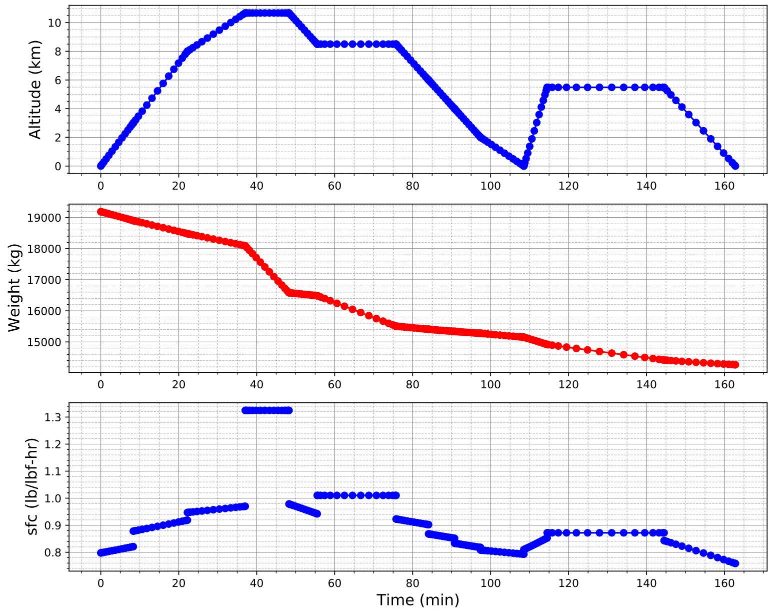


FIGURE 4.17: Altitude, SFC, Weight

- This graph shows the aircraft's weight and specific fuel consumption (SFC) shown in Figure 4.17. Highest altitude can reach 10.6 km or 35,000 feet. The vehicle function specifies the aircraft's initial MTOW which is 19,200 kg, as a result of the fuel burn, the aircraft the aircraft weight drops to about 14,267 kg by the end of the mission. The MZFW or the maximum weight that the aircraft can carry out the mission without useful fuel on board is 10,360 kg, as a result 3,900 kg of useful fuel are still on board.
- Reserves fuel for this mission is 3,900 kg and the fuel fraction is 0.203, getting from dividing 3,900 with the MTOW which is 19,200 kg.
- Typical fuel fraction of a jet fighter when doing combat maneuver is 0.290 [44]. Based on this information, it is not feasible for F-16 to perform PDI with combat maneuver, with current mission profile setup in SUAVE.

4. Drag Analysis:

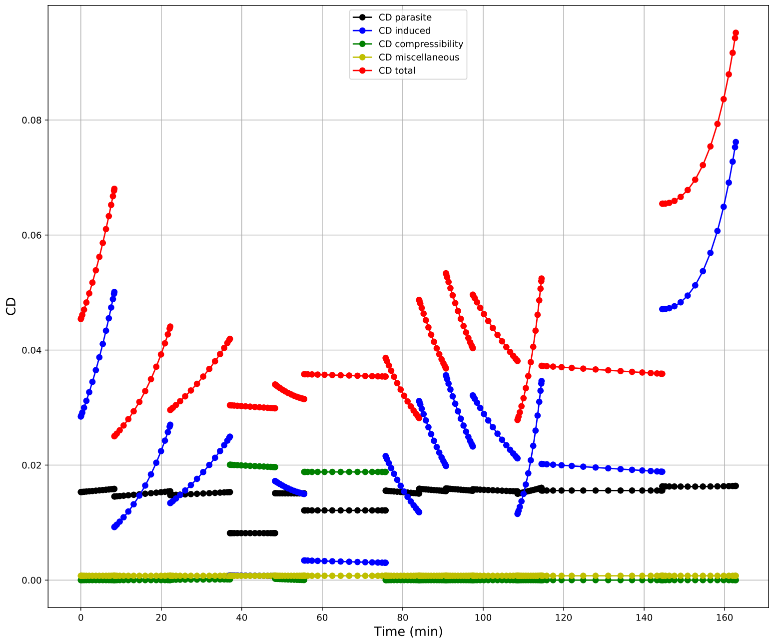


FIGURE 4.18: Drag Components

- The graph shows the overall drag, which is the sum of several drag components throughout the flight, like parasite, induced, compressibility, and miscellaneous drag shown in Figure 4.18.

5. Flight Conditions Analysis:

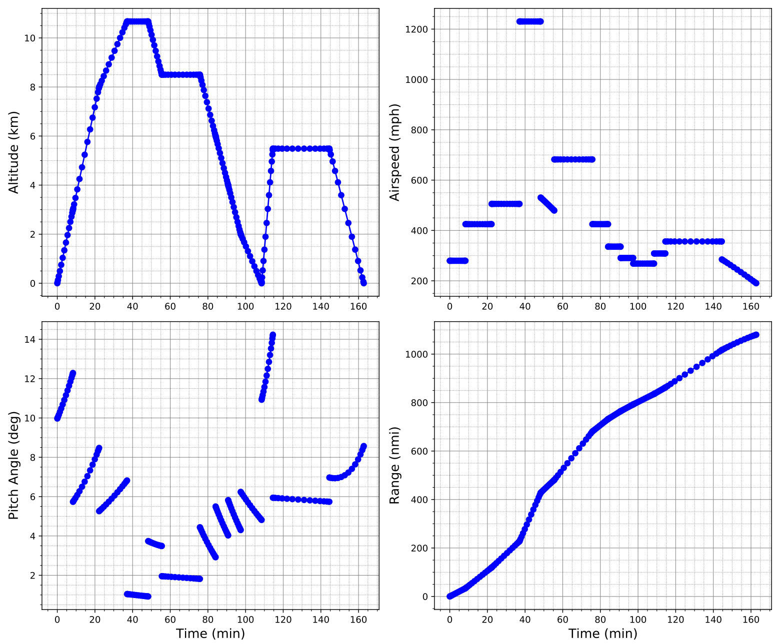


FIGURE 4.19: Flight Conditions

- Several flight metrics, including altitude, airspeed, pitch angle, and range over time are shown in Figure 4.19. The airspeed and altitude profiles match of those mission with different flight segments. The aircraft's operating capacity within a certain mission profile is highlighted by its range, which gradually to 1,080 nautical miles, or almost 2,000 km.

6. Flight Trajectory Analysis:

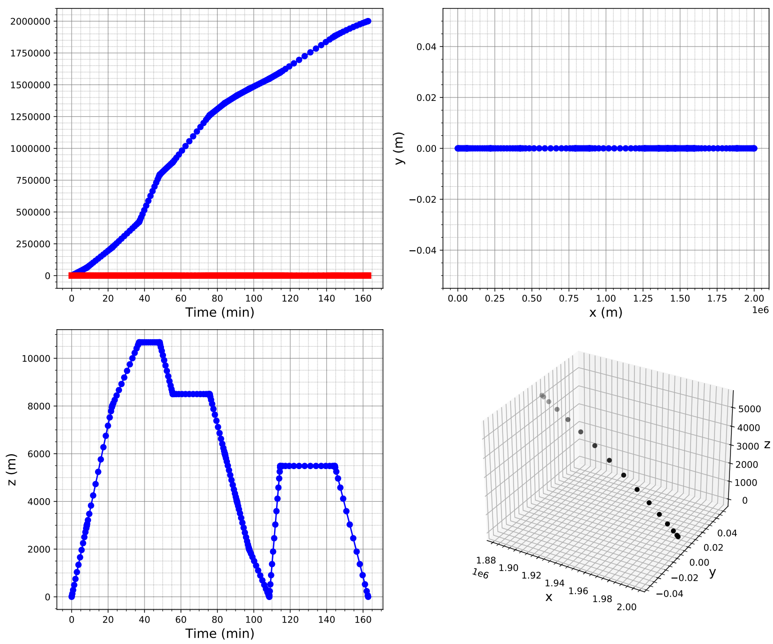


FIGURE 4.20: Flight Trajectory

- The aircraft's path represented by x,y,z coordinates over time and as well as 3D dimensions on the trajectory graph are plotted in Figure 4.20.

Intercept/Escort Mission

1. Fuel Burn:

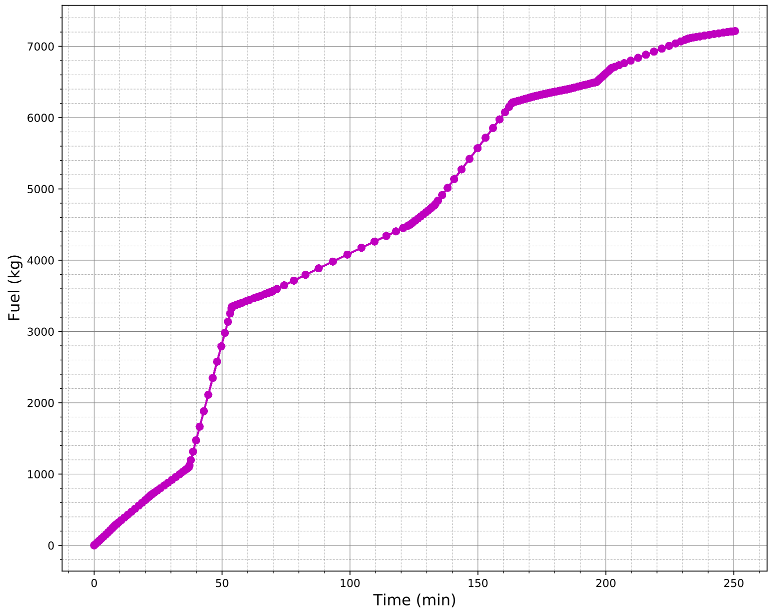


Figure 4.21: Aircraft Fuel Burnt

- Over the course of 250 minutes, the fuel consumption graph shows a steady increase from 0 kg to roughly is shown in Figure 7,214 kg 4.21.

2. Aircraft Velocities:

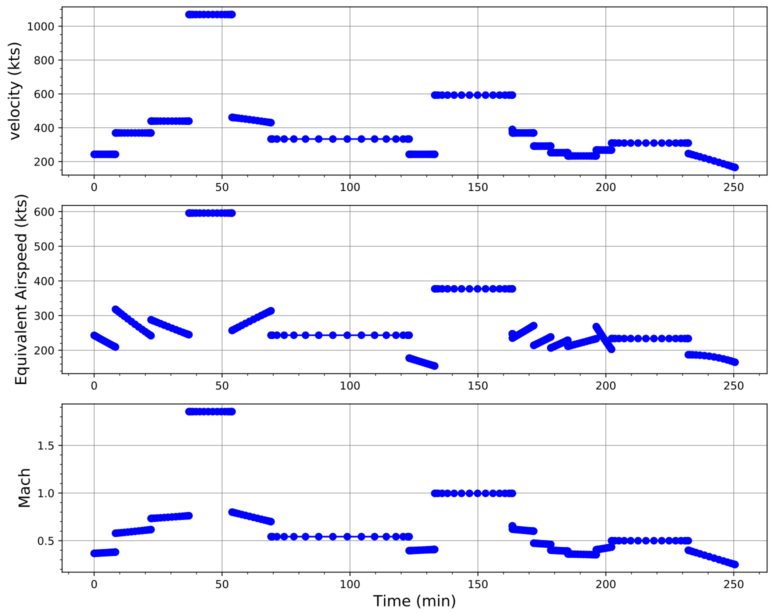


Figure 4.22: Aircraft Velocities

- Over a 250 minutes period, Mach number reached 1.6, and equivalent airspeed peaks at about 600 knots, which is around Mach 0.8 is shown in Figure 4.22.

3. Altitude, Specific Fuel Consumption (SFC), and Weight Analysis:

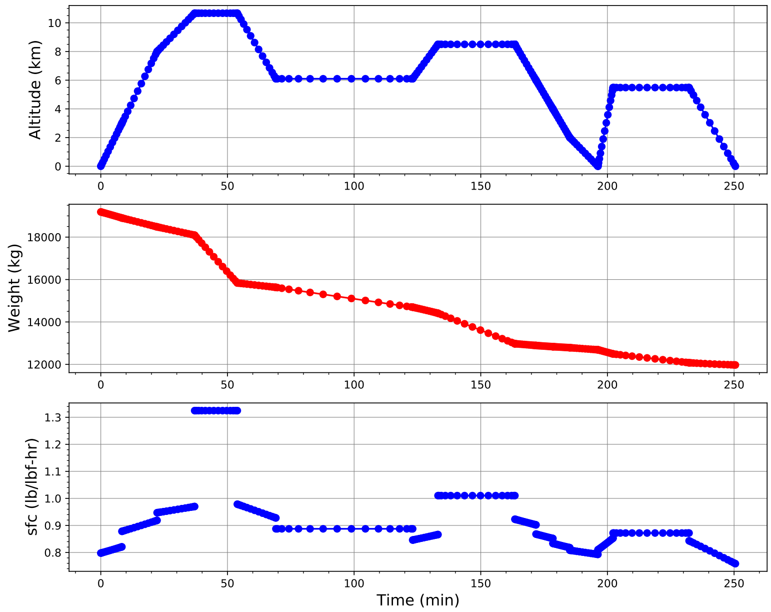


FIGURE 4.23: Altitude, SFC, Weight

- Given graphs shows that steady climbing to about 10.6 km or 35,000 feet, over the course 250 minutes, the weight decreases from roughly 19,200 kg to 11,973 kg, while the remaining useful fuel on board is 1,600 kg is shown in Figure 4.23.

4. Drag Analysis:

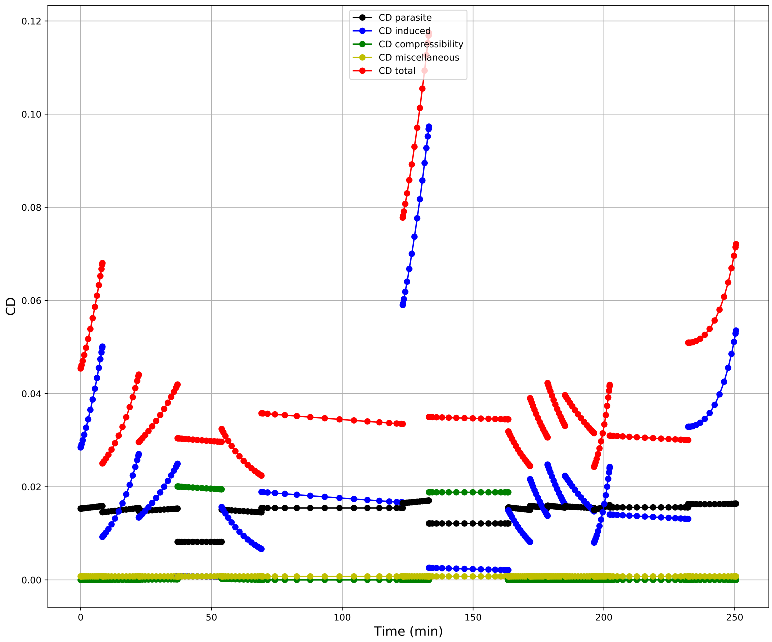


Figure 4.24: Drag Components

- Given plot shows that during the flight, the total drag coefficient fluctuates and peaks approximately above 0.12 is shown in Figure 4.24.

5. Flight Conditions Analysis:

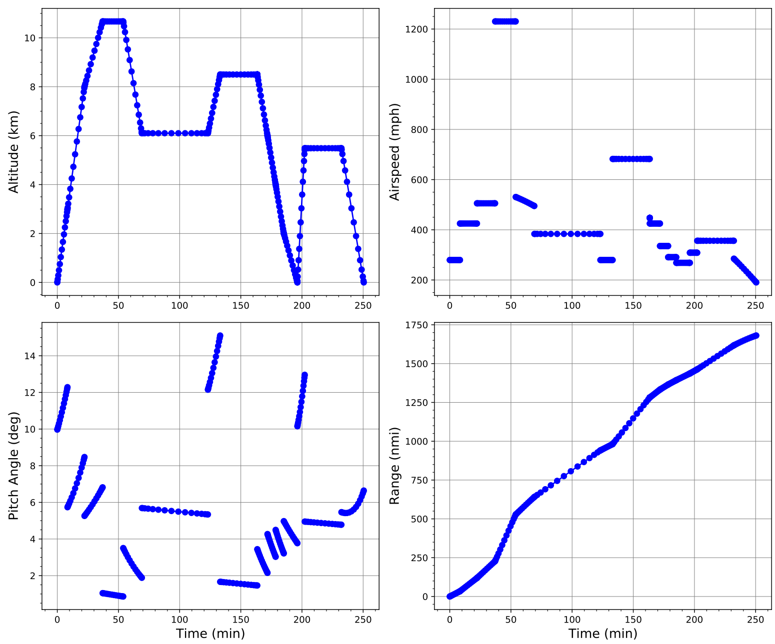


FIGURE 4.25: Flight Conditions

- According to the plot, the aircraft reaches the altitude of 10.6 km and continues to fly at up to 1,200 mph or about Mach 1.6, while the total range attained is around 1,750 nmi or roughly 3,214 km over 250 minute flight time is shown in Figure 4.25.

6. Flight Trajectory Analysis:

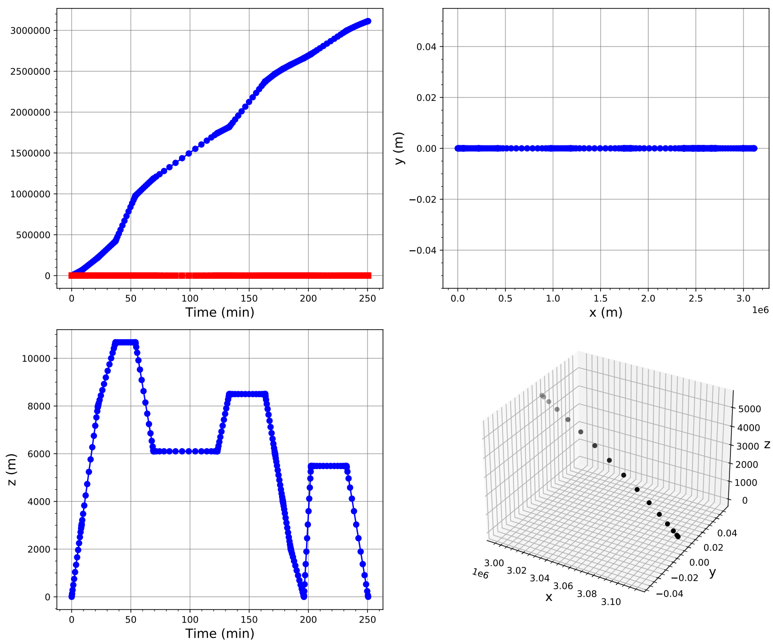


FIGURE 4.26: Flight Trajectory

- The plot shows a vertical displacement close to 10,000 meters (10km) and a lateral displacement of almost 3112,000 meters (3,112 km) is shown in Figure 4.26.

4.4 Conclusions

The conclusion of this study is that out of three mission profiles (DCA, PDI, IE) given by AIAA, only two (PDI, IE) which complied successfully when doing analyzing using SUAVE. Tables of compliance for all three mission profiles are presented in Table 4.3, 4.4, 4.5.

Table 4.3: Table of Compliance for Defensive Counter-Air Patrol Mission (DCA)

Phase	RFP Requirement	Met
1	Take-off and acceleration allowance	✓
2	Climb from sea level to optimm cruise altitude	✓
3	Cruise out 300 nm at optimum speed and altitude	✓
4	Combat air patrol 4 hours at best loite speed and 35,000 ft	✖
5	Dash 100 nm at maximum speed at 35,000 ft	✓
6	Combat maneuvers	✖
7	Climb/accelerate to optimum speed and altitude	✓
8	Cruise back 400 nm at optimum speed and altitude	✓
9	Descend to sea level	✓
10	Reserves loiter	✓

Table 4.4: Table of Compliance for Point Defense Intercept Mission (PDI)

Phase	RFP Requirement	Met
1	Take-off and acceleration allowance	✓
2	Climb from sea level to 35,000 ft and accelerate to maximum speed	✓
3	Dash 200 nm at maximum speed at 35,000 ft	✓
4	Combat maneuvers	✖
5	Climb/accelerate to optimum speed and altitude	✓
6	Cruise back 200 nm at optimum speed and altitude	✓
7	Descend to sea level	✓
8	Reserves loiter	✓

Table 4.5: Table of Compliance for Intercept/Escort Mission (IE)

Phase	RFP Requirement	Met
1	Take-off and acceleration allowance	✓
2	Climb from sea level to 35,000 ft and accelerate to maximum speed	✓
3	Dash out at maximum speed at 35,000 ft	✓
4	Escort for 300 nm at minimum practical airspeed. Retain all weapons	✓
5	Climb/accelerate to optimum speed and altitude	✓
6	Cruise back at optimum speed and altitude	✓
7	Descend to sea level	✓
8	Reserves loiter	✓

As summarized in Table 4.3, the table presents the compliance data for the Defensive Counter-Air Mission (DCA)

1. Phase 4: According to the requirements, patrol time has to be cut from 4 hours (14,400 seconds) to 2.1 hours (7,600 seconds) in order to finish all mission phases.
2. Phase 6: In order to finish the simulation, this phase was eliminated entirely because SUAVE lacked a feature that would have allowed for the execution of combat maneuvers.

However, in Table 4.4 which table represent the compliance data for Point Defensive Intercept Mission (PDI), demonstrates that only one mission phase—the combat maneuvers—did not follow the mission profile given by AIAA. This is because of the same SUAVE constraint that is occurred in DCA mission.

On the other hand, Table 4.5, which details about compliance for Intercept/Escort Mission (IE), shows that all mission phases established by AIAA were successfully completed, proving that the F-16 can successfully complete an IE mission.

This study concludes that the F-16 can successfully and carry out intercept/escort missions. Despite successfully completing point defense intercept missions—apart from combat maneuvers—it falls short of the defensive combat air patrol's 4-hour patrolling requirement. The F-16 therefore needs to make additional changes in order to completely comply to the DCA mission profile.

CHAPTER 5
SUMMARY, CONCLUSION, RECOMMENDATION

5.1 Summary

Using SUAVE, this thesis analyzes F-16 Fighting Falcon's performance as a Homeland Defense Interceptor (HDI) in three mission profiles: Defensive Counter-Air (DCA), Point Defense Intercept (PDI), and Intercept/Escort (IE). F-16's endurance and agility in supersonic flying conditions were evaluated through extensive simulations. The aircraft performed quite well in PDI and IE missions, but its limited fuel capacity prevented it from fulfilling DCA mission, underscoring the necessity of improving fuel management to completely satisfy HDI criteria.

According to the study, F-16 could need to have its aerodynamics redesigned and perhaps have hybrid propulsion systems in order to increase its operational range and performance. These result highlights the possibility of improving F-16 capabilities and modifying fighter aircraft tactics for defense of the homeland. In order to ensure that the aircraft satisfies the changing requirements of national security, this analysis establishes the foundation for future advancements in its design and operating strategies.

5.2 Conclusion

Detailed Analyses Outcome

F-16 may fulfill the mission for PDI and IE missions, according to the simulations. However, based on the analyses and the error showed during the simulation, due to useful fuel on-board, it was unable to meet DCA mission profile, highlighting the need for improved design or better fuel management to increase fuel capacity.

This deficiency is significant because its affects the aircraft's capacity to continue operating after completing its mission profiles.

Technical Challenges

1. SUAVE Limitations:
   - Combat Maneuvers: SUAVE currently lacks the ability to simulate complex combat maneuvers, which are crucial for accurately modeling interceptor missions.
   - Multi-Wing Configurations: SUAVE does not support the anlaysis of aircraft with complex wing configurations, which limits the accuracy of aerodynamic assessments for F-16 and any other unconventional aircraft.
   - As discussed in previous Section 4.1.1, SUAVE encountered difficulties in accurately visualizing complex aerodynamic components and systems. Among these were problems with modeling engine nacelles and "skinning" the fuselage, both of which are essential for accurate performance assessments.
   - Furthermore, there can be a variety of inconsistencies in the control surface, wing segments, and fuselage segments when importing data from SUAVE to OpenVSP. These differences force users to go from OpenVSP to SUAVE and adjust the models through trial and error.
2. Strategic Implications: There are important strategic issues when using F-16 as HDI, as was covered in previous Section 2.5.1. Highlighting by the QF-16 role in modernization and unmanned operations, but it also emphasizes the need for significant investment in upgrades and modifications to meet HDI requirement given by AIAA.
3. Operational Scenarios and Outcomes: The operational requirements for PDI and IE missions are satisfactorily met by F-16, however it performs poorly in DCA mission, as shown in Table 4.3. According to the mission analysis and error shown in the simulation, the main drawback was the absence of useful fuel on board. This suggests that the mission parameters may need to be reconsidered or that the aircraft's capacity and fuel efficiency be improved.

5.3 Recommendation

This study's thorough assessment of the F-16 as a HDI utilizing SUAVE has shown a number of areas that need more investigation and improvement in order to improve the aircraft's capabilities. In order to further increase F-16 efficiency in HDI missions, the following recommendations seek to overcome the shortcomings that have been found and build upon the preliminary findings:

Broader Simulations and Operational Analysis

To gain better understanding of F-16 performance under various operational situations, run simulations under a greater variety of flight conditions, such as different Mach numbers and atmospheric conditions.

System Integration and Multidisciplinary Studies

To make sure that alterations are not affecting F-16 speed and responsiveness, consider how the suggested aerodynamic changes will affect the aircraft's control surfaces and flight control systems.

Exploration of Alternative Aircraft Models

Future studies should examine more aircraft models in SUAVE framework. This comparison may highlight important compromises and improvements required for the best possible mission performance.

Bibliography

[1] AIAA, *AIAA 2024-2025 Team Homeland Defense Interceptor Request For Propsal*, https://www.aiaa.org/docs/default-source/uploadedfiles/membership-and-communities/university-students/design-competitions/2024-2025_team_homeland_defense_interceptor-updated-9-december-2024.pdf, 2024.

[2] *F-16 Fighting Falcon*, https://www.af.mil/About-Us/Fact-Sheets/Display/Article/104505/f-16-fighting-falcon/https%3A%2F%2Fwww.af.mil%2FAbout-Us%2FFact-Sheets%2FDisplay%2FArticle%2F104505%2Ff-16-fighting-falcon%2F. (visited on 01/31/2025).

[3] J. Trevithick, *Stealthy Target Drones Sought As QF-16 Program Winds Down*, https://www.twz.com/stealthy-target-drones-sought-as-qf-16-program-winds-down, Aug. 2022. (visited on 01/20/2025).

[4] T. MacDonald, M. Clarke, E. M. Botero, J. M. Vegh, and J. J. Alonso, "Suave: An open-source environment enabling multi-fidelity vehicle optimization," in *18th AIAA/ISSMO Multidisciplinary Analysis and Optimization Conference*. DOI: 10.2514/6.2017-4437. eprint: https://arc.aiaa.org/doi/pdf/10.2514/6.2017-4437. [Online]. Available: https://arc.aiaa.org/doi/abs/10.2514/6.2017-4437.

[5] R. A. McDonald and J. R. Gloudemans, "Open Vehicle Sketch Pad: An Open Source Parametric Geometry and Analysis Tool for Conceptual Aircraft Design," in *AIAA SCITECH 2022 Forum*, San Diego, CA & Virtual: American Institute of Aeronautics and Astronautics, Jan. 2022, ISBN: 978-1-62410-631-6. DOI: 10.2514/6.2022-0004. (visited on 01/25/2025).

[6] *Resources | Aerospace Design Lab*, https://adl.stanford.edu/resources. (visited on 01/29/2025).

[7] *OpenVSP/OpenVSP*, OpenVSP, Jan. 2025. (visited on 01/29/2025).

[8] *Convair F-106 Delta Dart - Wikipedia*, https://en.wikipedia.org/wiki/Convair_F-106_Delta_Dart. (visited on 01/29/2025).

[9] *Panavia Tornado ADV - Wikipedia*, https://en.wikipedia.org/wiki/Panavia_Tornado_ADV. (visited on 01/29/2025).

[10] *Northrop F-89 Scorpion - Wikipedia*, https://en.wikipedia.org/wiki/Northrop_F-89_Scorpion. (visited on 01/29/2025).

[11] *Saab 37 Viggen - Wikipedia*, https://en.wikipedia.org/wiki/Saab_37_Viggen. (visited on 01/29/2025).

[12] "Lockheed Martin F-22 Raptor," *Wikipedia*, Jan. 2025. (visited on 01/31/2025).

[13] *Northrop pitches optionally manned Firebird to monitor Australia's coasts*, https://www.defensenews.com/digital-show-dailies/avalon/2019/02/28/northrop-pitches-optionally-manned-firebird-to-monitor-australias-coasts/. (visited on 01/29/2025).

[14] "General Atomics MQ-9 Reaper," *Wikipedia*, Jan. 2025. (visited on 01/29/2025).

[15] "Boeing X-45," *Wikipedia*, Jan. 2025. (visited on 01/29/2025).

[16] "List of MiG-27 operators," *Wikipedia*, Jan. 2025. (visited on 01/29/2025).

[17] "Mikoyan-Gurevich MiG-25," *Wikipedia*, Jan. 2025. (visited on 01/29/2025).

[18] "English Electric Lightning," *Simple English Wikipedia, the free encyclopedia*, Jul. 2023. (visited on 01/29/2025).

[19] *What are these pipelines for between the folding wing mechanism of F4 Phantom II FGR2 and Buccaneer? - Aviation Stack Exchange*, https://aviation.stackexchange.com/questions/99049/what-are-these-pipelines-for-between-the-folding-wing-mechanism-of-f4-phantom-ii. (visited on 01/29/2025).

[20] D. P. Raymer, *Aircraft Design: A Conceptual Approach* (AIAA Education Series), Sixth edition. Reston, VA: American Institute of Aeronautics and Astronautics, Inc, 2018, ISBN: 978-1-62410-490-9.

[21] "2003 Bawean incident," *Wikipedia*, Feb. 2025. (visited on 02/13/2025).

[22] *Indonesia is intercepting aircraft - outside their airspace*, Jan. 2019. (visited on 02/13/2025).

[23] *McDonnell Douglas F-15 Eagle - Wikipedia*, https://en.wikipedia.org/wiki/McDonnell_Douglas_F-15_Eagle. (visited on 01/29/2025).

[24] "Boeing F/A-18E/F Super Hornet," *Wikipedia*, Jan. 2025. (visited on 01/31/2025).

[25] *France Confirms Upgraded Mirage 2000s Heading To Ukraine In Early 2025*, https://www.twz.com/air/france-confirms-upgraded-mirage-2000s-heading-to-ukraine-in-early-2025. (visited on 01/29/2025).

[26] *F-16V (Viper) Fighting Falcon Multi-role Fighter Aircraft*. (visited on 01/20/2025).

[27] "General Dynamics F-16 Fighting Falcon "Viper"," *Wikipedia*, Jan. 2025. (visited on 01/29/2025).

[28] *FY16 Air Force Programs QF-16 Full Scale Aerial Target (FSAT)*. [Online]. Available: https://www.dote.osd.mil/Portals/97/pub/reports/FY2016/af/2016qf16fsat.pdf?ver=2019-08-22-105431-373.

[29] *Further than R-73 and Sparrow. AIM-120 AMRAAM*. (visited on 01/29/2025).

[30] *Cannon, 20mm, M61A1 Vulcan | National Air and Space Museum*, https://airandspace.si.edu/collection-media/NASM-NASM2011-02417-000004. (visited on 01/29/2025).

[31] *AIM-9 Sidewinder - Wikipedia*, https://en.wikipedia.org/wiki/AIM-9_Sidewinder. (visited on 01/29/2025).

[32] "AN/APG-68," *Wikipedia*, Jan. 2025. (visited on 01/29/2025).

[33] *Radar fire-control F-16 | Military Aerospace*, https://www.militaryaerospace.com/sensors/article/14074509/radar-fire-control-f-16. (visited on 01/29/2025).

[34] J. M. Vegh, E. Botero, M. Clarke, J. Smart, and J. Alonso, "Current Capabilities and Challenges of NDARC and SUAVE for eVTOL Aircraft Design and Analysis," in *AIAA Propulsion and Energy 2019 Forum*, Indianapolis, IN: American Institute of Aeronautics and Astronautics, Aug. 2019, ISBN: 978-1-62410-590-6. DOI: 10.2514/6.2019-4505. (visited on 11/25/2024).

[35] Á. Gómez-Rodríguez, A. Sanchez-Carmona, L. García-Hernández, and C. Cuerno-Rejado, "Preliminary Correlations for Remotely Piloted Aircraft Systems Sizing," *Aerospace*, vol. 5, no. 1, p. 5, Jan. 2018, ISSN: 2226-4310. DOI: 10.3390/aerospace5010005. (visited on 11/25/2024).

[36] Y. Cai, D. Rajaram, and D. N. Mavris, "Simultaneous aircraft sizing and multi-objective optimization considering off-design mission performance during early design," *Aerospace Science and Technology*, vol. 126, p. 107 662, Jul. 2022, ISSN: 12709638. DOI: 10.1016/j.ast.2022.107662. (visited on 11/25/2024).

[37] J Mangold, D Eisenhut, F Brenner, N Moebs, and A Strohmayer, "Preliminary hybrid-electric aircraft design with advancements on the open-source tool SUAVE," *Journal of Physics: Conference Series*, vol. 2526, no. 1, p. 012 022, Jun. 2023, ISSN: 1742-6588, 1742-6596. DOI: 10.1088/1742-6596/2526/1/012022. (visited on 11/25/2024).

[38] S. Karpuk and A. Elham, "Influence of Novel Airframe Technologies on the Feasibility of Fully-Electric Regional Aviation," *Aerospace*, vol. 8, no. 6, p. 163, Jun. 2021, ISSN: 2226-4310. DOI: 10.3390/aerospace8060163. (visited on 11/25/2024).

[39] *SUAVE_Install*, https://suave.stanford.edu/download/standard_install.html. (visited on 01/25/2025).

[40] A. Wendorff, A. Variyar, C. Ilario, *et al.*, *Suave: An aerospace vehicle environment for designing future aircraft*, version 2.1, 2020. [Online]. Available: https://github.com/suavecode/SUAVE.

[41] *Openvsp_download*, https://openvsp.org/download.php. (visited on 01/25/2025).

[42] *Boeing 737-800 - SUAVE Tutorial*, https://suave.stanford.edu/tutorials/B737.html. (visited on 02/15/2025).

[43] D. Enriquez and J. Garcia, *F-16 "Fighting Falcon"*, https://vspairshow.com. (visited on 01/28/2025).

[44] mathscinotes, *WW2 Fighter Aircraft Fuel Fraction*, Mar. 2020. (visited on 02/16/2025).

Appendices

Appendix A: F-16 Model Created Using SUAVE Framework

```
#------------
#   Imports
#------------

import numpy as np
import matplotlib.pyplot as plt

import SUAVE
from SUAVE.Core import Units
from SUAVE.Methods.Propulsion.turbofan_sizing import turbofan_sizing
from SUAVE.Methods.Geometry.Two_Dimensional.Planform import (
wing_planform,
segment_properties,
)

from SUAVE.Input_Output.OpenVSP import write, get_vsp_measurements
from SUAVE.Input_Output.OpenVSP.vsp_read import vsp_read

from copy import deepcopy

#------------
#   Define the Vehicle
#------------


def setup():
base_vehicle = base_setup()

vsp_write_read(base_vehicle)

configs = configs_setup(base_vehicle)
```

```

    #print(configs)

    return configs, base_vehicle


def base_setup():
    #------------
    #   Initialize the Vehicle
    #------------

    vehicle = SUAVE.Vehicle()
    vehicle.tag = "F-16"

    #------------
    #   Vehicle-level Properties
    #------------

    vehicle.mass_properties.max_takeoff = 19187.0  # kg
    vehicle.mass_properties.operating_empty = 8570.0 +90.0 # kg
    vehicle.mass_properties.takeoff = 19187.0  # kg
    vehicle.mass_properties.max_zero_fuel = 8263.55  # kg
    vehicle.mass_properties.max_payload = 4470  # kg
    vehicle.mass_properties.max_fuel = 14061.0  # kg
    vehicle.mass_properties.cargo = 0.0  # kg

    vehicle.mass_properties.center_of_gravity = [[16.8, 0, 1.6]]
    vehicle.mass_properties.moments_of_inertia.tensor = [
    [32.222, -0.050, 1.599],
    [-0.050, 303.792, 0.076],
    [1.599, 0.076, 325.136],
    ]

    # envelope properties
    vehicle.envelope.ultimate_load = 13.5
    vehicle.envelope.limit_load = 9

    # basic parameters
    vehicle.reference_area = 34.40808
    vehicle.passengers = 0
```

```
vehicle.systems.control = "fully powered"
vehicle.systems.accessories = "medium range"
vehicle.total_length = 15.070
vehicle.maximum_cross_sectional_area = 8

#------------
#   Main Wing
#------------
wing = SUAVE.Components.Wings.Main_Wing()
wing.tag = "main_wing"
wing.areas.reference = 34.40813 * Units.meter**2  # 17.20406
wing.aspect_ratio = 2.96104  # 1.48052
wing.chords.root = 5.61458 * Units.meter
wing.chords.tip = 1.20312 * Units.meter
wing.sweeps.quarter_chord = 33.1 * Units.deg  # 41.04545
wing.thickness_to_chord = 0.10000
wing.taper = wing.chords.tip / wing.chords.root
wing.dihedral = 0.0 * Units.deg
wing.spans.projected = 10.09375
wing.origin = [[6.304 * Units.meter, 0, 0.290 * Units.meter]]
wing.vertical = False
wing.symmetric = True
wing.high_lift = True
wing.vortex_lift = True
wing.high_mach = True
wing.areas.exposed = 0.80 * wing.areas.wetted
wing.twists.root = 0.0 * Units.degrees
wing.twists.tip = 0.0 * Units.degrees
wing.dynamic_pressure_ratio = 1.0

# control surfaces -------------------------------------------
flap = SUAVE.Components.Wings.Control_Surfaces.Flap()
flap.tag = "flap"
flap.span_fraction_start = 0.402439
flap.span_fraction_end = 1.0
flap.deflection = 0.0 * Units.deg
flap.chord_fraction = 0.243902
flap.configuration_type = "trailing_edge"
wing.append_control_surface(flap)

```

```
wing = wing_planform(wing)

wing.areas.exposed = 0.90 * wing.areas.wetted
wing.twists.root = 0.0 * Units.degrees
wing.twists.tip = 0.0 * Units.degrees
wing.dynamic_pressure_ratio = 1.0

# add to vehicle
vehicle.append_component(wing)

#------------
wing = SUAVE.Components.Wings.Main_Wing()
wing.tag = "main_wing_1"
wing.areas.reference = 8.53846 * Units.meter**2  # 17.20406
wing.aspect_ratio = 1.03221  # 1.48052
wing.chords.root = 5.69775 * Units.meter
wing.chords.tip = 0.05447 * Units.meter
wing.sweeps.quarter_chord = 75.0 * Units.deg
wing.thickness_to_chord = 0.10000
wing.taper = wing.chords.tip / wing.chords.root
wing.dihedral = 0.0 * Units.deg
wing.spans.projected = 2.96875
wing.origin = [[0.6 * Units.meter, 0, 0.290 * Units.meter]]
wing.vertical = False
wing.symmetric = True
wing.high_lift = False
wing.vortex_lift = True
wing.high_mach = True
wing.areas.exposed = 0.80 * wing.areas.wetted
wing.twists.root = 0.0 * Units.degrees
wing.twists.tip = 0.0 * Units.degrees
wing.dynamic_pressure_ratio = 1.0

wing = wing_planform(wing)

# add to vehicle
vehicle.append_component(wing)

#------------
#   Main Wing
```

```
#------------

#------------
#  Horizontal Stabilizer (Tail Elevon)
#------------

wing = SUAVE.Components.Wings.Stabilator()
wing.tag = "stabilator"
wing.areas.reference = 12.32273 * Units.meter**2  #  #6.16136
wing.aspect_ratio = 2.08396  # 1.04198
wing.sweeps.quarter_chord = 41.48750 * Units.deg
wing.thickness_to_chord = 0.10000
wing.taper = 0.36360
wing.dihedral = 0.0 * Units.degrees
wing.origin = [[12.391 * Units.meter, 0.500 * Units.meter, 0.290 *
  Units.meter]]
wing.vertical = False
wing.symmetric = True
wing.high_lift = False
wing = wing_planform(wing)
wing.areas.exposed = 0.9 * wing.areas.wetted
wing.twists.root = 2.0 * Units.degrees
wing.twists.tip = 2.0 * Units.degrees
wing.dynamic_pressure_ratio = 0.90

# control surfaces -------------------------------------------
# rudder = SUAVE.Components.Wings.Control_Surfaces.Rudder()
# rudder.tag = "rudder"
# rudder.span_fraction_start = 0.372  # 0.155
# rudder.span_fraction_end = 1.0  # 1.0
# rudder.deflection = 0.0 * Units.deg
# rudder.chord_fraction = 1.0
# rudder.configuration_type = "trailing_edge"
# wing.append_control_surface(rudder)

# add to vehicle
vehicle.append_component(wing)

#------------
#   Vertical Stabilizer
```

```python
#------------

wing = SUAVE.Components.Wings.Vertical_Tail()
wing.tag = "vertical_stabilizer"
wing.sweeps.quarter_chord = 0.0 * Units.deg
wing.thickness_to_chord = 0.03
wing.areas.reference = 8.46748 * Units.meter**2
wing.spans.projected = 2.24609 * Units.meter + 0.89063 * Units.meter
wing.chords.root = 5.833 * Units.meter
# wing.chords.tip = 2.881 * Units.meter
# wing.taper = wing.chords.tip / wing.chords.root
wing.aspect_ratio = wing.spans.projected**2.0 / wing.areas.reference
wing.twists.root = 0.0 * Units.degrees
wing.twists.tip = 0.0 * Units.degrees
wing.origin = [[9.783 * Units.meter, 0, 0.850 * Units.meter]]
wing.vertical = True
wing.symmetric = False
wing.high_lift = False
wing.dynamic_pressure_ratio = 1.0

# Wing Segments
segment = SUAVE.Components.Wings.Segment()
segment.tag = "Root"
segment.percent_span_location = 0.0
segment.twist = 0.0 * Units.deg
segment.root_chord_percent = 1  # 1.0
segment.thickness_to_chord = 0.03
segment.dihedral_outboard = 0.0 * Units.degrees
segment.sweeps.quarter_chord = 71.2 * Units.degrees  # 75.16023
wing.append_segment(segment)

segment = SUAVE.Components.Wings.Segment()
segment.tag = "Break"
segment.percent_span_location = 0.2839
segment.twist = 0.0 * Units.deg
segment.root_chord_percent = 0.49
segment.thickness_to_chord = 0.03
segment.dihedral_outboard = 0 * Units.degrees
segment.sweeps.quarter_chord = 44.6 * Units.degrees
wing.append_segment(segment)
```

```

segment = SUAVE.Components.Wings.Segment()
segment.tag = "Tip"
segment.percent_span_location = 1.0
segment.twist = 0.0 * Units.degrees
segment.root_chord_percent = 0.2439
segment.thickness_to_chord = 0.1
segment.dihedral_outboard = 0.0
segment.sweeps.quarter_chord = 44.6 * Units.degrees  # 49.66591
wing.append_segment(segment)

# Fill out more segment properties automatically
wing = segment_properties(wing)
wing = SUAVE.Methods.Geometry.Two_Dimensional.Planform.wing_planform
 (wing)

# # add to vehicle
vehicle.append_component(wing)

#------------
#  Fuselage
#------------

fuselage = SUAVE.Components.Fuselages.Fuselage()
fuselage.tag = "fuselage"
fuselage.origin = [[0, 0, 0]]
fuselage.number_coach_seats = 1
fuselage.seats_abreast = 1
fuselage.seat_pitch = 0.0

fuselage.fineness.nose = 2.0  # 1.28 * Units.meter
fuselage.fineness.tail = 4.626  # 3.48

fuselage.lengths.nose = 4.569 * Units.meter  # 3.748
fuselage.lengths.tail = 9.0207 * Units.meter  #  8.549
fuselage.lengths.cabin = 1.4803 * Units.meter  # 2.845
fuselage.lengths.total = 15.070 * Units.meter
fuselage.lengths.fore_space = 0.0
fuselage.lengths.aft_space = 0.0

```

```
fuselage.width = 1.95000

fuselage.heights.maximum = 1.38201 * Units.meter
fuselage.heights.at_quarter_length = 1.38201 * Units.meter
fuselage.heights.at_three_quarters_length = 1.18182 * Units.meter
fuselage.heights.at_wing_root_quarter_chord = 1.18182 * Units.meter

fuselage.areas.side_projected = 6.08548 * Units.meter**2  # 22.27
fuselage.areas.wetted = 65.334 * Units.meter**2  # 51.083
fuselage.areas.front_projected = 1.496 * Units.meter**2

fuselage.effective_diameter = 1.38 * Units.meter

fuselage.differential_pressure = (
7.4e4 * Units.pascal
)  # Maximum differential pressure

# # Segment
segment = SUAVE.Components.Lofted_Body_Segment.Segment()
segment.tag = "segment_0"
segment.percent_x_location = 0.0
segment.percent_z_location = 0.03000
segment.height = 0.0
segment.width = 0.0
fuselage.Segments.append(segment)

# Segment
segment = SUAVE.Components.Lofted_Body_Segment.Segment()
segment.tag = "segment_1"
segment.percent_x_location = 0.30341
segment.percent_z_location = 0.04000
segment.height = 1.38201* Units.meter
segment.width = 1.21036* Units.meter
fuselage.Segments.append(segment)

# Segment
segment = SUAVE.Components.Lofted_Body_Segment.Segment()
segment.tag = "segment_2"
segment.percent_x_location = 0.40171
segment.percent_z_location = 0.04000
```

```
segment.height = 1.38201* Units.meter
segment.width = 1.37582* Units.meter
fuselage.Segments.append(segment)

# Segment
segment = SUAVE.Components.Lofted_Body_Segment.Segment()
segment.tag = "segment_3"
segment.percent_x_location = 0.61651
segment.percent_z_location = 0.03261
segment.height = 1.18182* Units.meter
segment.width = 1.95000* Units.meter
fuselage.Segments.append(segment)

# Segment
segment = SUAVE.Components.Lofted_Body_Segment.Segment()
segment.tag = "segment_5"
segment.percent_x_location = 1.00000
segment.percent_z_location = 0.03237
segment.height = 0.91971* Units.meter
segment.width = 0.91971* Units.meter
fuselage.Segments.append(segment)

# Segment
segment = SUAVE.Components.Lofted_Body_Segment.Segment()
segment.tag = "segment_6"
segment.percent_x_location = 1.0
segment.percent_z_location = 0.03237
segment.height = 0.0
segment.width = 0.0
fuselage.Segments.append(segment)

# add to vehicle
vehicle.append_component(fuselage)


#------------
# the nacelle
#------------

nacelle = SUAVE.Components.Nacelles.Nacelle()
```

```
nacelle.diameter = 0.76
nacelle.tag = "nacelle"
nacelle.origin = [[14.0, 0, 0.490]]
nacelle.length = 1.4
nacelle.inlet_diameter = 0.50
nacelle.areas.wetted = 20.0
vehicle.append_component(nacelle)


#------------
#  Turbofan Network
#------------

# initialize the gas turbine network
gt_engine = SUAVE.Components.Energy.Networks.Turbofan()
gt_engine.tag = "turbofan"
gt_engine.origin = [[12.0, 4.38, -2.1], [12.0, -4.38, -2.1]]
gt_engine.number_of_engines = 2.0
gt_engine.bypass_ratio = 5.4

# add working fluid to the network
gt_engine.working_fluid = SUAVE.Attributes.Gases.Air()

#------------
#   Component 1 - Ram

# to convert freestream static to stagnation quantities

# instantiate
ram = SUAVE.Components.Energy.Converters.Ram()
ram.tag = "ram"

# add to the network
gt_engine.append(ram)

#------------
#  Component 2 - Inlet Nozzle

# instantiate
```

```
inlet_nozzle = SUAVE.Components.Energy.Converters.Compression_Nozzle
  ()
inlet_nozzle.tag = "inlet_nozzle"

# setup
inlet_nozzle.polytropic_efficiency = 0.98
inlet_nozzle.pressure_ratio = 1.0

# add to network
gt_engine.append(inlet_nozzle)

#------------
#  Component 3 - Low Pressure Compressor

# instantiate
compressor = SUAVE.Components.Energy.Converters.Compressor()
compressor.tag = "low_pressure_compressor"

# setup
compressor.polytropic_efficiency = 0.91
compressor.pressure_ratio = 3.1

# add to network
gt_engine.append(compressor)

#------------
#  Component 4 - High Pressure Compressor

# instantiate
compressor = SUAVE.Components.Energy.Converters.Compressor()
compressor.tag = "high_pressure_compressor"

# setup
compressor.polytropic_efficiency = 0.91
compressor.pressure_ratio = 5.0

# add to network
gt_engine.append(compressor)

#------------
```

```
#  Component 5 - Low Pressure Turbine

# instantiate
turbine = SUAVE.Components.Energy.Converters.Turbine()
turbine.tag = "low_pressure_turbine"

# setup
turbine.mechanical_efficiency = 0.99
turbine.polytropic_efficiency = 0.93

# add to network
gt_engine.append(turbine)

#------------
#  Component 6 - High Pressure Turbine

# instantiate
turbine = SUAVE.Components.Energy.Converters.Turbine()
turbine.tag = "high_pressure_turbine"

# setup
turbine.mechanical_efficiency = 0.99
turbine.polytropic_efficiency = 0.93

# add to network
gt_engine.append(turbine)

#------------
#  Component 7 - Combustor

# instantiate
combustor = SUAVE.Components.Energy.Converters.Combustor()
combustor.tag = "combustor"

# setup
combustor.efficiency = 0.99
combustor.turbine_inlet_temperature = 1450.0
combustor.pressure_ratio = 1.0
combustor.fuel_data = SUAVE.Attributes.Propellants.Jet_A()

```

```
# add to network
gt_engine.append(combustor)

#------------
#  Component 8 - Core Nozzle

# instantiate
nozzle = SUAVE.Components.Energy.Converters.Supersonic_Nozzle()
nozzle.tag = "core_nozzle"

# setup
nozzle.polytropic_efficiency = 0.95
nozzle.pressure_ratio = 0.99

# add to network
gt_engine.append(nozzle)

# Component 8 :fan nozzle
fan_nozzle = SUAVE.Components.Energy.Converters.Expansion_Nozzle()
fan_nozzle.tag = "fan nozzle"
fan_nozzle.polytropic_efficiency = 0.95
fan_nozzle.pressure_ratio = 0.99
# add the fan nozzle to the network
gt_engine.fan_nozzle = fan_nozzle

# Component 9 : fan
fan = SUAVE.Components.Energy.Converters.Fan()
fan.tag = "fan"
fan.polytropic_efficiency = 0.93
fan.pressure_ratio = 1.7
# add the fan to the network
gt_engine.fan = fan

# Component 10 : thrust (to compute the thrust)
thrust = SUAVE.Components.Energy.Processes.Thrust()
thrust.tag = "compute_thrust"
# total design thrust (includes all the engines)
thrust.total_design = 120102.0 * Units.N  # Newtons

# design sizing conditions
```

```
altitude = 35000.0 * Units.ft
mach_number = 1.6
isa_deviation = 0.0
# add thrust to the network
gt_engine.thrust = thrust

# size the turbofan
turbofan_sizing(gt_engine, mach_number, altitude)

# add  gas turbine network gt_engine to the vehicle
vehicle.append_component(gt_engine)

fuel = SUAVE.Components.Physical_Component()
vehicle.fuel = fuel
fuel.mass_properties.mass = (
vehicle.mass_properties.max_takeoff - vehicle.mass_properties.
 max_fuel
)
fuel.origin = vehicle.wings.main_wing.mass_properties.
 center_of_gravity
fuel.mass_properties.center_of_gravity = vehicle.wings.main_wing.
 aerodynamic_center

#------------
#   Vehicle Definition Complete
#------------

return vehicle


#------------
#   Define the Configurations
#---------------


def configs_setup(vehicle):
#------------
#   Initialize Configurations
#------------

```

```
configs = SUAVE.Components.Configs.Config.Container()

base_config = SUAVE.Components.Configs.Config(vehicle)
base_config.tag = "base"
configs.append(base_config)

#------------
#   Cruise Configuration
#------------

config = SUAVE.Components.Configs.Config(base_config)
config.tag = "cruise"

configs.append(config)

config.maximum_lift_coefficient = 1.2

#------------
#   Cruise with Spoilers Configuration
#------------

config = SUAVE.Components.Configs.Config(base_config)
config.tag = "cruise_spoilers"

configs.append(config)

config.maximum_lift_coefficient = 1.2

#------------
#   Takeoff Configuration
#------------

config = SUAVE.Components.Configs.Config(base_config)
config.tag = "takeoff"
config.wings["main_wing"].control_surfaces.flap.deflection = 20.0 *
 Units.deg
# config.wings["main_wing"].control_surfaces.slat.deflection = 25.0
 * Units.deg
config.V2_VS_ratio = 1.21
configs.append(config)
```

```
    #------------
    #   Landing Configuration
    #------------

    config = SUAVE.Components.Configs.Config(base_config)
    config.tag = "landing"
    config.wings["main_wing"].control_surfaces.flap.deflection = 30.0 * Units.deg
    # config.wings["main_wing"].control_surfaces.slat.deflection = 25.0 * Units.deg
    config.Vref_VS_ratio = 1.23
    configs.append(config)

    #------------
    #   Short Field Takeoff Configuration
    #------------

    config = SUAVE.Components.Configs.Config(base_config)
    config.tag = "short_field_takeoff"
    config.wings["main_wing"].control_surfaces.flap.deflection = 20.0 * Units.deg
    # config.wings["main_wing"].control_surfaces.slat.deflection = 25.0 * Units.deg
    config.V2_VS_ratio = 1.21

    configs.append(config)

    return configs


def vsp_write_read(vehicle):
    """
    Function to read and write into OpenVSP
    """
    write(vehicle, "F-16")

    return
```

```
setup()
plt.show()
```

Appendix B: Defensive Counter-Air Patrol Mission Profile Python Code

```
#------------
#   Define the Configurations
#---------------

def configs_setup(vehicle):

#------------
#   Initialize Configurations
#------------

configs = SUAVE.Components.Configs.Config.Container()

base_config = SUAVE.Components.Configs.Config(vehicle)
base_config.tag = 'base'
configs.append(base_config)

#------------
#   Cruise Configuration
#------------

config = SUAVE.Components.Configs.Config(base_config)
config.tag = 'cruise'

configs.append(config)

#------------
#   Takeoff Configuration
#------------

config = SUAVE.Components.Configs.Config(base_config)
config.tag = 'takeoff'
```

```

config.V2_VS_ratio = 1.21
config.maximum_lift_coefficient = 2.

configs.append(config)

#------------
#   Landing Configuration
#------------

config = SUAVE.Components.Configs.Config(base_config)
config.tag = 'landing'

config.Vref_VS_ratio = 1.23
config.maximum_lift_coefficient = 2.

configs.append(config)

return configs

#------------
#   Plot Mission
#------------

def plot_mission(results,line_style='bo-'):

# Plot Altitude, sfc, vehicle weight
plot_altitude_sfc_weight(results, line_style) #DONE

# Plot Velocities
plot_aircraft_velocities(results, line_style) #DONE

plot_fuel_use(results, line_style) #DONE

# Plot Aerodynamic Coefficients
plot_aerodynamic_coefficients(results, line_style)  #DONE

# Plot Aerodynamic Forces
plot_aerodynamic_forces(results, line_style) #DONE

```

```
# Drag Components
plot_drag_components(results, line_style) #DONE

# Plot Flight Conditions
plot_flight_conditions(results, line_style) #DONE

plot_flight_trajectory(results, line_style) #DONE

plot_stability_coefficients(results, line_style) #DONE

return

def simple_sizing(configs):

base = configs.base
base.pull_base()

# zero fuel weight
base.mass_properties.max_zero_fuel = 0.9 * base.mass_properties.
    max_takeoff

# wing areas
for wing in base.wings:
wing.areas.wetted   = 2.0 * wing.areas.reference
wing.areas.exposed  = 0.8 * wing.areas.wetted
wing.areas.affected = 0.6 * wing.areas.wetted

# fuselage seats
base.fuselages['fuselage'].number_coach_seats = base.passengers

# diff the new data
base.store_diff()


# done!
return

#------------
  #   Define the Mission
#------------
```

```python

def mission_setup(analyses):
#------------
  #   Initialize the Mission
#------------
  mission = SUAVE.Analyses.Mission.Sequential_Segments()
  mission.tag = 'DCA test mission'

  # atmospheric model
  atmosphere = SUAVE.Attributes.Atmospheres.Earth.US_Standard_1976()
  planet = SUAVE.Attributes.Planets.Earth()

  # airport
  airport = SUAVE.Attributes.Airports.Airport()
  airport.altitude   =  0.0  * Units.ft
  airport.delta_isa  =  0.0
  airport.atmosphere = SUAVE.Attributes.Atmospheres.Earth.
   US_Standard_1976()

  mission.airport = airport

  # unpack Segments module
  Segments = SUAVE.Analyses.Mission.Segments

  # base segment
  base_segment = Segments.Segment()

#------------
  #   First Climb Segment: Constant Speed, Constant Rate
#------------

  segment = Segments.Climb.Constant_Speed_Constant_Rate()
  segment.tag = "climb_1"

  # connect vehicle configuration
  segment.analyses.extend(analyses.base)

  # define segment attributes
  segment.atmosphere = atmosphere
  segment.planet = planet
```

```

    segment.altitude_start = 0.0 * Units.km
    segment.altitude_end = 3.048 * Units.km
    segment.air_speed = 144.0 * Units["m/s"]
    segment.climb_rate = 14.0 * Units["m/s"]

    # add to misison
    mission.append_segment(segment)

#------------
    #   Second Climb Segment: Constant Speed, Constant Rate
#------------

    segment = Segments.Climb.Constant_Speed_Constant_Rate()
    segment.tag = "climb_2"

    # connect vehicle configuration
    segment.analyses.extend(analyses.cruise)

    # segment attributes
    segment.atmosphere = atmosphere
    segment.planet = planet

    segment.altitude_end = 4.57 * Units.km
    segment.air_speed = 165.0 * Units["m/s"]
    segment.climb_rate = 9.0 * Units["m/s"]

    # add to mission
    mission.append_segment(segment)

#------------
    #   Third Climb Segment: Constant Speed, Constant Climb Rate
#------------

    segment = Segments.Climb.Constant_Speed_Constant_Rate()
    segment.tag = "climb_3"

    # connect vehicle configuration
    segment.analyses.extend(analyses.cruise)

```

```
    # segment attributes
    segment.atmosphere = atmosphere
    segment.planet = planet

    segment.altitude_end = 7.6 * Units.km
    segment.air_speed = 230.0 * Units["m/s"]
    segment.climb_rate = 4.5 * Units["m/s"]

    # add to mission
    mission.append_segment(segment)

#------------
    #   Fourth Climb Segment: Constant Speed, Constant Rate
#------------

    segment = Segments.Climb.Constant_Speed_Constant_Rate()
    segment.tag = "climb_4"

    # connect vehicle configuration
    segment.analyses.extend(analyses.cruise)

    # segment attributes
    segment.atmosphere = atmosphere
    segment.planet = planet

    segment.altitude_end = 8.5 * Units.km
    segment.air_speed = 240.0 * Units["m/s"]
    segment.climb_rate = 4.0 * Units["m/s"]

    # add to mission
    mission.append_segment(segment)

#------------
    #   Cruise Out
#------------

    segment = Segments.Cruise.Constant_Speed_Constant_Altitude(
     base_segment)
    segment.tag = "cruise_out"

```

```
    segment.analyses.extend(analyses.cruise)

    # segment attributes
    segment.atmosphere = atmosphere
    segment.planet     = planet

    segment.air_speed = 305 * Units["m/s"]
    segment.distance = 300. * Units.nmi
    segment.altitude = 8.5 * Units.km
    mission.append_segment(segment)

#------------
    #   Fifth Climb Segment: linear Mach
#------------

    segment = Segments.Climb.Constant_Speed_Constant_Rate(base_segment)
    segment.tag = "climb_5"

    segment.analyses.extend( analyses.base )

    segment.altitude_end    = 10.67    * Units.km
    segment.air_speed       = 230.0   * Units['m/s']
    segment.climb_rate      = 4.0   * Units['m/s']

    # add to mission
    mission.append_segment(segment)

#------------
    #   Combat Air Patrol (CAP) for 4 hours at 35,000 ft
#------------

    segment = Segments.Cruise.Constant_Mach_Constant_Altitude_Loiter(
      base_segment)
    segment.tag = "combat_air_patrol"

    segment.analyses.extend(analyses.cruise)

    segment.time = 7600* Units.sec
    segment.altitude = 10.67 * Units.km
    segment.mach = 0.6
```

```
    mission.append_segment(segment)

    #------------
    # #   Dash Segment (100 nm at 35,000 ft)
    #------------

    segment = Segments.Cruise.Constant_Mach_Constant_Altitude(base_segment)
    segment.tag = "dash"
    segment.analyses.extend(analyses.cruise)
    segment.altitude = 10.67 * Units.km
    segment.distance = 100. * Units.nmi
    segment.mach     = 1.6
    mission.append_segment(segment)

#------------
    #   First Descent Segment
#------------

    segment = Segments.Descent.Constant_Speed_Constant_Rate(base_segment)
    segment.tag = "descent_0"

    segment.analyses.extend( analyses.base )

    segment.altitude_end = 8.5   * Units.km
    segment.air_speed    = 305. * Units['m/s']
    segment.descent_rate = 5.0   * Units['m/s']

    # append to mission
    mission.append_segment(segment)

#------------
    #   Cruise Back to Base (400 nm)
#------------

    segment = Segments.Cruise.Constant_Speed_Constant_Altitude(base_segment)
    segment.tag = "cruise_back"
```

```
  segment.analyses.extend(analyses.cruise)
  segment.air_speed = 305 * Units["m/s"]
  segment.distance = 400. * Units.nmi
  segment.altitude = 8.5 * Units.km
  mission.append_segment(segment)

#------------
  #   Second Descent Segment: Constant Speed, Constant Rate
#------------

  segment = Segments.Descent.Constant_Speed_Constant_Rate(base_segment
   )
  segment.tag = "descent_2"

  segment.analyses.extend( analyses.landing )

  segment.altitude_end = 6.8   * Units.km
  segment.air_speed    = 195.0 * Units['m/s']
  segment.descent_rate = 5.0   * Units['m/s']

  # Add to mission
  mission.append_segment(segment)

#------------
  #   Third Descent Segment: Constant Speed, Constant Rate
#------------

  segment = Segments.Descent.Constant_Speed_Constant_Rate(base_segment
   )
  segment.tag = "descent_3"

  segment.analyses.extend( analyses.landing )

  analyses.landing.aerodynamics.settings.spoiler_drag_increment = 0.00

  segment.altitude_end = 4.0   * Units.km
  segment.air_speed    = 170.0 * Units['m/s']
  segment.descent_rate = 5.0   * Units['m/s']

  # Add to mission
```

```
    mission.append_segment(segment)

#------------
    #   Fourth Descent Segment: Constant Speed, Constant Rate
#------------

    segment = Segments.Descent.Constant_Speed_Constant_Rate(base_segment
     )
    segment.tag = "descent_4"

    segment.analyses.extend( analyses.landing )
    analyses.landing.aerodynamics.settings.spoiler_drag_increment = 0.00

    segment.altitude_end = 2.0   * Units.km
    segment.air_speed    = 150.0 * Units['m/s']
    segment.descent_rate = 5.0   * Units['m/s']

    # Add to mission
    mission.append_segment(segment)

#------------
    #   Fifth Descent Segment: Constant Speed, Constant Rate
#------------

    segment = Segments.Descent.Constant_Speed_Constant_Rate(base_segment
     )
    segment.tag = "descent_5"

    segment.analyses.extend( analyses.landing )
    analyses.landing.aerodynamics.settings.spoiler_drag_increment = 0.00

    segment.altitude_end = 0.0   * Units.km
    segment.air_speed    = 145.0 * Units['m/s']
    segment.descent_rate = 3.0   * Units['m/s']

    # Append to mission
    mission.append_segment(segment)

#------------
    #   Mission definition complete
```

```
#------------

#------------
  ###          Reserve mission
#------------

#------------
  #   First Climb Segment: Constant Speed, Constant Throttle
#------------

  segment = Segments.Climb.Constant_Speed_Constant_Rate()
  segment.tag = "reserve_climb"

  # connect vehicle configuration
  segment.analyses.extend(analyses.base)

  # define segment attributes
  segment.atmosphere = atmosphere
  segment.planet = planet

  segment.altitude_start = 0.0 * Units.km
  segment.altitude_end = 18000.0 * Units.ft
  segment.air_speed = 138.0 * Units["m/s"]
  segment.climb_rate = 15.3 * Units["m/s"]

  # add to misison
  mission.append_segment(segment)

#------------
  #   Loiter Segment: constant mach, constant time
#------------

  segment = Segments.Cruise.Constant_Mach_Constant_Altitude_Loiter(
    base_segment)
  segment.tag = "reserve_loiter"

  segment.analyses.extend(analyses.cruise)

  segment.mach = 0.5
  segment.time = 30.0 * Units.minutes
```

```

  mission.append_segment(segment)

#------------
  #  Final Descent Segment: consant speed, constant segment rate
#------------

  segment = Segments.Descent.Linear_Mach_Constant_Rate(base_segment)
  segment.tag = "reserve_descent_1"

  segment.analyses.extend(analyses.landing)

  segment.altitude_end = 0.0 * Units.km
  segment.descent_rate = 5.0 * Units["m/s"]
  segment.mach_end = 0.25
  segment.mach_start = 0.4

  # append to mission
  mission.append_segment(segment)

#------------
  ###         Reserve mission completed
#------------

  return mission

  def missions_setup(base_mission):

  # the mission container
  missions = SUAVE.Analyses.Mission.Mission.Container()

#------------
  #   Base Mission
#------------

  missions.base = base_mission

  # done!
  return missions

```

```python
if __name__ == '__main__':

main()
```

Appendix C: Point Defense Intercept Mission Profile Python Code

```python
#------------
  #   Define the Configurations
  #---------------

  def configs_setup(vehicle):

  #------------
  #   Initialize Configurations
  #------------

  configs = SUAVE.Components.Configs.Config.Container()

  base_config = SUAVE.Components.Configs.Config(vehicle)
  base_config.tag = 'base'
  configs.append(base_config)

  #------------
  #   Cruise Configuration
  #------------

  config = SUAVE.Components.Configs.Config(base_config)
  config.tag = 'cruise'

  configs.append(config)

  #------------
  #   Takeoff Configuration
  #------------

  config = SUAVE.Components.Configs.Config(base_config)
  config.tag = 'takeoff'
```

```

config.V2_VS_ratio = 1.21
config.maximum_lift_coefficient = 2.

configs.append(config)

#------------
#   Landing Configuration
#------------

config = SUAVE.Components.Configs.Config(base_config)
config.tag = 'landing'

config.Vref_VS_ratio = 1.23
config.maximum_lift_coefficient = 2.

configs.append(config)

return configs

#------------
#   Plot Mission
#------------

def plot_mission(results,line_style='bo-'):

# Plot Altitude, sfc, vehicle weight
plot_altitude_sfc_weight(results, line_style) #DONE

# Plot Velocities
plot_aircraft_velocities(results, line_style) #DONE

plot_fuel_use(results, line_style) #DONE

# Plot Aerodynamic Coefficients
plot_aerodynamic_coefficients(results, line_style)  #DONE

# Plot Aerodynamic Forces
plot_aerodynamic_forces(results, line_style) #DONE

```

```
# Drag Components
plot_drag_components(results, line_style) #DONE

# Plot Flight Conditions
plot_flight_conditions(results, line_style) #DONE

plot_flight_trajectory(results, line_style) #DONE

plot_stability_coefficients(results, line_style) #DONE

return

def simple_sizing(configs):

base = configs.base
base.pull_base()

# zero fuel weight
base.mass_properties.max_zero_fuel = 0.9 * base.mass_properties.
  max_takeoff

# wing areas
for wing in base.wings:
wing.areas.wetted   = 2.0 * wing.areas.reference
wing.areas.exposed  = 0.8 * wing.areas.wetted
wing.areas.affected = 0.6 * wing.areas.wetted

# fuselage seats
base.fuselages['fuselage'].number_coach_seats = base.passengers

# diff the new data
base.store_diff()


# done!
return

#------------
#   Define the Mission
#------------
```

```python

def mission_setup(analyses):
#------------
#   Initialize the Mission
#------------
mission = SUAVE.Analyses.Mission.Sequential_Segments()
mission.tag = 'PDI test mission'

# atmospheric model
atmosphere = SUAVE.Attributes.Atmospheres.Earth.US_Standard_1976()
planet = SUAVE.Attributes.Planets.Earth()

# airport
airport = SUAVE.Attributes.Airports.Airport()
airport.altitude   =  0.0  * Units.ft
airport.delta_isa  =  0.0
airport.atmosphere = SUAVE.Attributes.Atmospheres.Earth.
 US_Standard_1976()

mission.airport = airport

# unpack Segments module
Segments = SUAVE.Analyses.Mission.Segments

# base segment
base_segment = Segments.Segment()

#------------
#   First Climb Segment: Constant Speed, Constant Rate
#------------

segment = Segments.Climb.Constant_Speed_Constant_Rate(base_segment)
segment.tag = "climb_1"

segment.analyses.extend( analyses.takeoff )

segment.altitude_start = 0.0   * Units.km
segment.altitude_end   = 3.0   * Units.km
segment.air_speed      = 125.0 * Units['m/s']
segment.climb_rate     = 6.0   * Units['m/s']
```

```

# Add to misison
mission.append_segment(segment)

#------------
#   Second Climb Segment: Constant Speed, Constant Rate
#------------

segment = Segments.Climb.Constant_Speed_Constant_Rate(base_segment)
segment.tag = "climb_2"

segment.analyses.extend( analyses.cruise )

segment.altitude_end   = 8.0   * Units.km
segment.air_speed      = 190.0 * Units['m/s']
segment.climb_rate     = 6.0   * Units['m/s']

# Add to mission
mission.append_segment(segment)

#------------
#   Third Climb Segment: constant Speed, Constant Rate
#------------

segment = Segments.Climb.Constant_Speed_Constant_Rate(base_segment)
segment.tag = "climb_3"

segment.analyses.extend( analyses.cruise )

segment.altitude_end = 10.67 * Units.km
segment.air_speed    = 226.0  * Units['m/s']
segment.climb_rate   = 3.0    * Units['m/s']

# Add to mission
mission.append_segment(segment)

#------------
#   Dash Segment (200 nm at 35,000 ft)
#------------

```

```
segment = Segments.Cruise.Constant_Speed_Constant_Altitude(
  base_segment)
segment.tag = "dash"
segment.analyses.extend(analyses.cruise)
segment.altitude      = 10.67 * Units.km
segment.distance      = 200. * Units.nmi
segment.air_speed     = 550 * Units["m/s"]
mission.append_segment(segment)

#------------
#   Zeroth Descent Segment
#------------

segment = Segments.Descent.Linear_Mach_Constant_Rate(base_segment)
segment.tag = "descent_0"

segment.analyses.extend( analyses.base )

segment.altitude_end = 8.5   * Units.km
segment.descent_rate = 5.0   * Units['m/s']
segment.air_speed    = 150   * Units['m/s']

# add to mission
mission.append_segment(segment)

#------------
#   Cruise Back to Base (200 nm)
#------------

segment = Segments.Cruise.Constant_Speed_Constant_Altitude(
  base_segment)
segment.tag = "cruise_back"
segment.analyses.extend(analyses.cruise)
segment.air_speed = 305 * Units["m/s"]
segment.distance = 200. * Units.nmi
segment.altitude = 8.5 * Units.km
mission.append_segment(segment)

#------------
#   Second Descent Segment: Constant Speed, Constant Rate
```

```
#------------

segment = Segments.Descent.Constant_Speed_Constant_Rate(base_segment
 )
segment.tag = "descent_1"

segment.analyses.extend( analyses.landing )

segment.altitude_end = 6.0   * Units.km
segment.air_speed    = 190.0 * Units['m/s']
segment.descent_rate = 5.0   * Units['m/s']

# Add to mission
mission.append_segment(segment)

#------------
#   Third Descent Segment: Constant Speed, Constant Rate
#------------

segment = Segments.Descent.Constant_Speed_Constant_Rate(base_segment
 )
segment.tag = "descent_2"

segment.analyses.extend( analyses.landing )

analyses.landing.aerodynamics.settings.spoiler_drag_increment = 0.00

segment.altitude_end = 4.0   * Units.km
segment.air_speed    = 150.0 * Units['m/s']
segment.descent_rate = 5.0   * Units['m/s']

# Add to mission
mission.append_segment(segment)

#------------
#   Fourth Descent Segment: Constant Speed, Constant Rate
#------------

segment = Segments.Descent.Constant_Speed_Constant_Rate(base_segment
 )
```

```
segment.tag = "descent_3"

segment.analyses.extend( analyses.landing )
analyses.landing.aerodynamics.settings.spoiler_drag_increment = 0.00

segment.altitude_end = 2.0   * Units.km
segment.air_speed    = 130.0 * Units['m/s']
segment.descent_rate = 5.0   * Units['m/s']

# Add to mission
mission.append_segment(segment)

#------------
#   Fifth Descent Segment: Constant Speed, Constant Rate
#------------

segment = Segments.Descent.Constant_Speed_Constant_Rate(base_segment
 )
segment.tag = "descent_4"

segment.analyses.extend( analyses.landing )
analyses.landing.aerodynamics.settings.spoiler_drag_increment = 0.00

segment.altitude_end = 0.0   * Units.km
segment.air_speed    = 120.0 * Units['m/s']
segment.descent_rate = 3.0   * Units['m/s']

# Append to mission
mission.append_segment(segment)

#------------
#   Mission definition complete
#------------

#------------
###         Reserve mission
#------------

#------------
#   First Climb Segment: Constant Speed, Constant Throttle
```

```
#------------

segment = Segments.Climb.Constant_Speed_Constant_Rate()
segment.tag = "reserve_climb"

# connect vehicle configuration
segment.analyses.extend(analyses.base)

# define segment attributes
segment.atmosphere = atmosphere
segment.planet = planet

segment.altitude_start = 0.0 * Units.km
segment.altitude_end = 18000.0 * Units.ft
segment.air_speed = 138.0 * Units["m/s"]
segment.climb_rate = 15.3 * Units["m/s"]

# add to misison
mission.append_segment(segment)

#------------
#   Loiter Segment: constant mach, constant time
#------------

segment = Segments.Cruise.Constant_Mach_Constant_Altitude_Loiter(
 base_segment)
segment.tag = "reserve_loiter"

segment.analyses.extend(analyses.cruise)

segment.mach = 0.5
segment.time = 30.0 * Units.minutes

mission.append_segment(segment)

#------------
#  Final Descent Segment: consant speed, constant segment rate
#------------

segment = Segments.Descent.Linear_Mach_Constant_Rate(base_segment)
```

```
segment.tag = "reserve_descent_1"

segment.analyses.extend(analyses.landing)

segment.altitude_end = 0.0 * Units.km
segment.descent_rate = 5.0 * Units["m/s"]
segment.mach_end = 0.25
segment.mach_start = 0.4

# append to mission
mission.append_segment(segment)

#------------
###         Reserve mission completed
#------------

return mission

def missions_setup(base_mission):

# the mission container
missions = SUAVE.Analyses.Mission.Mission.Container()

#------------
#   Base Mission
#------------

missions.base = base_mission

# done!
return missions

if __name__ == '__main__':

main()
```

Appendix D: Intercept/Escort Mission Profile Python Code

```
#------------
#   Define the Configurations
#---------------

def configs_setup(vehicle):

#------------
#   Initialize Configurations
#------------

configs = SUAVE.Components.Configs.Config.Container()

base_config = SUAVE.Components.Configs.Config(vehicle)
base_config.tag = 'base'
configs.append(base_config)

#------------
#   Cruise Configuration
#------------

config = SUAVE.Components.Configs.Config(base_config)
config.tag = 'cruise'

configs.append(config)

#------------
#   Takeoff Configuration
#------------

config = SUAVE.Components.Configs.Config(base_config)
config.tag = 'takeoff'
```

```

config.V2_VS_ratio = 1.21
config.maximum_lift_coefficient = 2.

configs.append(config)

#------------
#   Landing Configuration
#------------

config = SUAVE.Components.Configs.Config(base_config)
config.tag = 'landing'

config.Vref_VS_ratio = 1.23
config.maximum_lift_coefficient = 2.

configs.append(config)

return configs

#------------
#   Plot Mission
#------------

def plot_mission(results,line_style='bo-'):

# Plot Altitude, sfc, vehicle weight
plot_altitude_sfc_weight(results, line_style) #DONE

# Plot Velocities
plot_aircraft_velocities(results, line_style) #DONE

plot_fuel_use(results, line_style) #DONE

# Plot Aerodynamic Coefficients
plot_aerodynamic_coefficients(results, line_style)  #DONE

# Plot Aerodynamic Forces
plot_aerodynamic_forces(results, line_style) #DONE

```

```
# Drag Components
plot_drag_components(results, line_style) #DONE

# Plot Flight Conditions
plot_flight_conditions(results, line_style) #DONE

plot_flight_trajectory(results, line_style) #DONE

plot_stability_coefficients(results, line_style) #DONE

return

def simple_sizing(configs):

base = configs.base
base.pull_base()

# zero fuel weight
base.mass_properties.max_zero_fuel = 0.9 * base.mass_properties.
 max_takeoff

# wing areas
for wing in base.wings:
wing.areas.wetted   = 2.0 * wing.areas.reference
wing.areas.exposed  = 0.8 * wing.areas.wetted
wing.areas.affected = 0.6 * wing.areas.wetted

# fuselage seats
base.fuselages['fuselage'].number_coach_seats = base.passengers

# diff the new data
base.store_diff()


# done!
return

#------------
#   Define the Mission
#------------
```

```

def mission_setup(analyses):
#------------
#   Initialize the Mission
#------------
mission = SUAVE.Analyses.Mission.Sequential_Segments()
mission.tag = 'I/E test mission'

# atmospheric model
atmosphere = SUAVE.Attributes.Atmospheres.Earth.US_Standard_1976()
planet = SUAVE.Attributes.Planets.Earth()

# airport
airport = SUAVE.Attributes.Airports.Airport()
airport.altitude   =  0.0  * Units.ft
airport.delta_isa  =  0.0
airport.atmosphere = SUAVE.Attributes.Atmospheres.Earth.
 US_Standard_1976()

mission.airport = airport

# unpack Segments module
Segments = SUAVE.Analyses.Mission.Segments

# base segment
base_segment = Segments.Segment()

#------------
#   First Climb Segment: Constant Speed, Constant Rate
#------------

segment = Segments.Climb.Constant_Speed_Constant_Rate(base_segment)
segment.tag = "climb_1"

segment.analyses.extend( analyses.takeoff )

segment.altitude_start = 0.0   * Units.km
segment.altitude_end   = 3.0   * Units.km
segment.air_speed      = 125.0 * Units['m/s']
segment.climb_rate     = 6.0   * Units['m/s']
```

```

# Add to misison
mission.append_segment(segment)

#------------
#   Second Climb Segment: Constant Speed, Constant Rate
#------------

segment = Segments.Climb.Constant_Speed_Constant_Rate(base_segment)
segment.tag = "climb_2"

segment.analyses.extend( analyses.cruise )

segment.altitude_end   = 8.0   * Units.km
segment.air_speed      = 190.0 * Units['m/s']
segment.climb_rate     = 6.0   * Units['m/s']

# Add to mission
mission.append_segment(segment)

#------------
#   Third Climb Segment: constant Speed, Constant Rate
#------------

segment = Segments.Climb.Constant_Speed_Constant_Rate(base_segment)
segment.tag = "climb_3"

segment.analyses.extend( analyses.cruise )

segment.altitude_end = 10.67 * Units.km
segment.air_speed    = 226.0  * Units['m/s']
segment.climb_rate   = 3.0    * Units['m/s']

# Add to mission
mission.append_segment(segment)

#------------
#   Dash Segment (300 nm at 35,000 ft)
#------------

```

```
segment = Segments.Cruise.Constant_Speed_Constant_Altitude(
  base_segment)
segment.tag = "dash"
segment.analyses.extend(analyses.cruise)
segment.altitude      = 10.67 * Units.km
segment.distance      = 300. * Units.nmi
segment.air_speed     = 550 * Units["m/s"]
mission.append_segment(segment)

#------------
#   Zeroth Descent Segment
#------------

segment = Segments.Descent.Linear_Mach_Constant_Rate(base_segment)
segment.tag = "descent_0"

segment.analyses.extend( analyses.base )

segment.altitude_end = 6.1   * Units.km
segment.descent_rate = 5.0   * Units['m/s']
segment.air_speed    = 150   * Units['m/s']

# add to mission
mission.append_segment(segment)

#------------
#   escort (300 nm)
#------------

segment = Segments.Cruise.Constant_Speed_Constant_Altitude(
  base_segment)
segment.tag = "escort"
segment.analyses.extend(analyses.cruise)
segment.air_speed = 171.5 * Units["m/s"]
segment.distance = 300 * Units.nmi
segment.altitude = 6.1 * Units.km
mission.append_segment(segment)

#------------
#   Fourth Climb Segment: linear Mach
```

```
#------------

segment = Segments.Climb.Constant_Speed_Constant_Rate(base_segment)
segment.tag = "climb_4"

segment.analyses.extend( analyses.base )

segment.altitude_end    = 8.5    * Units.km
segment.air_speed       = 125   * Units['m/s']
segment.climb_rate      = 4.0   * Units['m/s']

# add to mission
mission.append_segment(segment)


#------------
#   Cruise Back to Base (300 nm)
#------------

segment = Segments.Cruise.Constant_Speed_Constant_Altitude(
  base_segment)
segment.tag = "cruise_back"
segment.analyses.extend(analyses.cruise)
segment.air_speed = 305 * Units["m/s"]
segment.distance = 300. * Units.nmi
segment.altitude = 8.5 * Units.km
mission.append_segment(segment)

#------------
#   First Descent Segment: Constant Speed, Constant Rate
#------------

segment = Segments.Descent.Constant_Speed_Constant_Rate(base_segment
  )
segment.tag = "descent_1"

segment.analyses.extend( analyses.cruise )

segment.altitude_end = 8.5    * Units.km
segment.air_speed     = 200.0 * Units['m/s']
```

```
segment.descent_rate = 4.5   * Units['m/s']

# Add to mission
mission.append_segment(segment)

#------------
#   Second Descent Segment: Constant Speed, Constant Rate
#------------

segment = Segments.Descent.Constant_Speed_Constant_Rate(base_segment
 )
segment.tag = "descent_2"

segment.analyses.extend( analyses.landing )

segment.altitude_end = 6.0   * Units.km
segment.air_speed    = 190.0 * Units['m/s']
segment.descent_rate = 5.0   * Units['m/s']

# Add to mission
mission.append_segment(segment)

#------------
#   Third Descent Segment: Constant Speed, Constant Rate
#------------

segment = Segments.Descent.Constant_Speed_Constant_Rate(base_segment
 )
segment.tag = "descent_3"

segment.analyses.extend( analyses.landing )

analyses.landing.aerodynamics.settings.spoiler_drag_increment = 0.00

segment.altitude_end = 4.0   * Units.km
segment.air_speed    = 150.0 * Units['m/s']
segment.descent_rate = 5.0   * Units['m/s']

# Add to mission
mission.append_segment(segment)
```

```

#------------
#   Fourth Descent Segment: Constant Speed, Constant Rate
#------------

segment = Segments.Descent.Constant_Speed_Constant_Rate(base_segment
 )
segment.tag = "descent_4"

segment.analyses.extend( analyses.landing )
analyses.landing.aerodynamics.settings.spoiler_drag_increment = 0.00

segment.altitude_end = 2.0   * Units.km
segment.air_speed    = 130.0 * Units['m/s']
segment.descent_rate = 5.0   * Units['m/s']

# Add to mission
mission.append_segment(segment)

#------------
#   Fifth Descent Segment: Constant Speed, Constant Rate
#------------

segment = Segments.Descent.Constant_Speed_Constant_Rate(base_segment
 )
segment.tag = "descent_5"

segment.analyses.extend( analyses.landing )
analyses.landing.aerodynamics.settings.spoiler_drag_increment = 0.00

segment.altitude_end = 0.0   * Units.km
segment.air_speed    = 120.0 * Units['m/s']
segment.descent_rate = 3.0   * Units['m/s']

# Append to mission
mission.append_segment(segment)

#------------
#   Mission definition complete
#------------
```

```

#------------
###         Reserve mission
#------------

#------------
#   First Climb Segment: Constant Speed, Constant Throttle
#------------

segment = Segments.Climb.Constant_Speed_Constant_Rate()
segment.tag = "reserve_climb"

# connect vehicle configuration
segment.analyses.extend(analyses.base)

# define segment attributes
segment.atmosphere = atmosphere
segment.planet = planet

segment.altitude_start = 0.0 * Units.km
segment.altitude_end = 18000.0 * Units.ft
segment.air_speed = 138.0 * Units["m/s"]
segment.climb_rate = 15.3 * Units["m/s"]

# add to misison
mission.append_segment(segment)

segment = Segments.Cruise.Constant_Mach_Constant_Altitude_Loiter(
  base_segment)
segment.tag = "reserve_loiter"

segment.analyses.extend(analyses.cruise)

segment.mach = 0.5
segment.time = 30.0 * Units.minutes

mission.append_segment(segment)

#------------
#  Final Descent Segment: consant speed, constant segment rate
```

```
#------------

segment = Segments.Descent.Linear_Mach_Constant_Rate(base_segment)
segment.tag = "reserve_descent_1"

segment.analyses.extend(analyses.landing)

segment.altitude_end = 0.0 * Units.km
segment.descent_rate = 5.0 * Units["m/s"]
segment.mach_end = 0.25
segment.mach_start = 0.4

# append to mission
mission.append_segment(segment)

#------------
###          Reserve mission completed
#------------

return mission

def missions_setup(base_mission):

# the mission container
missions = SUAVE.Analyses.Mission.Mission.Container()

#------------
#   Base Mission
#------------

missions.base = base_mission

# done!
return missions

if __name__ == '__main__':

main()
```

Turnitin Report

rizky

ORIGINALITY REPORT

9%	7%	3%	5%
SIMILARITY INDEX	INTERNET SOURCES	PUBLICATIONS	STUDENT PAPERS

PRIMARY SOURCES

1	vdocuments.mx Internet Source	1%
2	Submitted to Embry Riddle Aeronautical University Student Paper	1%
3	archive.aoe.vt.edu Internet Source	1%
4	suave.stanford.edu Internet Source	1%
5	Submitted to Nazarbayev University Student Paper	1%
6	nasticgrana.blogspot.com Internet Source	<1%
7	www.slideshare.net Internet Source	<1%
8	Submitted to RMIT University Student Paper	<1%
9	hdl.handle.net Internet Source	<1%

10	Submitted to Virginia Polytechnic Institute and State University Student Paper	<1 %
11	Submitted to Middle East Technical University Student Paper	<1 %
12	www.aiaa.org Internet Source	<1 %
13	Submitted to Imperial College of Science, Technology and Medicine Student Paper	<1 %
14	Melgueira, Pedro Miguel Lúcio. "Educational Data Mining Applied to Moodle Data From the University of ÉVora", Universidade de Evora (Portugal), 2024 Publication	<1 %
15	Karim Abu Salem, Giuseppe Palaia, Alessandro A. Quarta. "Impact of Figures of Merit Selection on Hybrid-Electric Regional Aircraft Design and Performance Analysis", Energies, 2023 Publication	<1 %
16	Rohan S. Sharma, Serhat Hosder. "Investigation of Aircraft Design Space Exploration with Machine Learning", AIAA Scitech 2021 Forum, 2021 Publication	<1 %

17	Muhammad Ali Baig. "US-China Strategic Competition - Military Strategy and Contemporary Doctrine", Routledge, 2025 Publication	<1 %
18	assets-eu.researchsquare.com Internet Source	<1 %
19	docslib.org Internet Source	<1 %
20	Submitted to Liberty University Student Paper	<1 %
21	Trent W. Lukaczyk, Andrew D. Wendorff, Michael Colonno, Thomas D. Economon, Juan J. Alonso, Tarik H. Orra, Carlos Ilario. "SUAVE: An Open-Source Environment for Multi-Fidelity Conceptual Vehicle Design", 16th AIAA/ISSMO Multidisciplinary Analysis and Optimization Conference, 2015 Publication	<1 %
22	interestingengineering.com Internet Source	<1 %
23	Ahmed F. El-Sayed. "History and Evolution of Aircraft - Technological Advancements in Size, Speed, Armaments, and Engines", CRC Press, 2024 Publication	<1 %

24	Manuel A. Rendón, Patrícia H. Hallak, Yipsy Roque Benito, Marcelo Assato et al. "Aircraft hybrid-electric environment for analysis and design (AHEAD) applied on a fixed-wing drone", Results in Engineering, 2025 Publication	<1 %
25	www.airforce-technology.com Internet Source	<1 %
26	Pasandideh, Shahryar. "In Search of a Prime Mover: An Investigation into the Relationship Between Technological Change and Shifts in the Fielded Military Capabilities of States", The George Washington University, 2024 Publication	<1 %
27	ceur-ws.org Internet Source	<1 %
28	escholarship.org Internet Source	<1 %
29	www.coursehero.com Internet Source	<1 %
30	Thu Aung Han. "Algorithm for Optimizing the Design Parameters of a Light Transport Aircraft at the Stage of Preliminary Design", E3S Web of Conferences, 2023 Publication	<1 %
	www.science.gov	

31	Internet Source	<1 %
32	Amit Kumar Tyagi, Shrikant Tiwari. "AI and Blockchain in Smart Grids - Fundamentals, Methods, and Applications", CRC Press, 2025 Publication	<1 %
33	Emilio M. Botero, Juan J. Alonso. "Conceptual Design and Optimization of Small Transitioning UAVs using SUAVE", 18th AIAA/ISSMO Multidisciplinary Analysis and Optimization Conference, 2017 Publication	<1 %
34	Stanislav Karpuk, Ali Elham. "Influence of Novel Airframe Technologies on the Feasibility of Fully-Electric Regional Aviation", Aerospace, 2021 Publication	<1 %
35	www.fliphtml5.com Internet Source	<1 %
36	Justin Bronk. "The Future of NATO Airpower - How are Future Capability Plans Within the Alliance Diverging and How can Interoperability be Maintained?", Whitehall Papers, 2020 Publication	<1 %
37	upcommons.upc.edu Internet Source	<1 %

38	Egeli, Sıtkı. "Greek Air Power as a National Security Instrument", Bilkent Universitesi (Turkey), 2022 Publication	<1 %
39	Submitted to University of Southampton Student Paper	<1 %
40	Matthew Clarke, Juan J. Alonso. "Lithium–Ion Battery Modeling for Aerospace Applications", Journal of Aircraft, 2021 Publication	<1 %
41	Mohammad H. Sadraey. "Aircraft Performance - An Engineering Approach", CRC Press, 2017 Publication	<1 %

Exclude quotes	On	Exclude matches	Off
Exclude bibliography	On		

Curriculum Vitae

	Basic Information
Name	Muhammad Rizky Permana
Place of Birth	Jakarta, Indonesia
Date of Birth	13, 06, 2003
Address	Jl. Gading Serpong Boulevard, Kabupaten Tangerang, Banten, Indonesia
Year	**Education**
2021 - present	International University Liaison Indonesia (IULI)
2017 - 2021	Emirates Private School
2016 - 2017	International Jubilee Private School
Year	**Courses**
2024	Non-Destructive Test Ultrasonic Testing Level II
2024	Non-Destructive Test Penetrant Testing Level II
2024	Non-Destructive Test Magnetic Testing Level II
2024	ISO 9001
2023	Pilot License for Small Drone
Year	**Seminars & Workshops**
2024	AIAA Design Competition (4th Place)
2023	Basic Aircraft Maintenance at UNSURYA