12-2023

Estimated Aerodynamic Forces and Moments and Optimal Orientation of the V-Bat Airframe During Vertical Landing in Gusty Conditions

Parker C. Carter
Utah State University, parker.carter@usu.edu

ESTIMATED AERODYNAMIC FORCES AND MOMENTS AND OPTIMAL ORIENTATION OF THE V-BAT AIRFRAME DURING VERTICAL LANDING IN GUSTY CONDITIONS

by

Parker C. Carter

A thesis submitted in partial fulfillment
of the requirements for the degree

of

MASTER OF SCIENCE

in

Mechanical Engineering

Approved:

Douglas Hunsaker, Ph.D. Major Professor	Som Dutta, Ph.D. Committee Member
Tianyi He, Ph.D. Committee Member	D. Richard Cutler, Ph.D. Vice Provost for Graduate Studies

UTAH STATE UNIVERSITY
Logan, Utah

2023

ABSTRACT

Estimated Aerodynamic Forces and Moments and Optimal Orientation of the V-BAT Airframe During Vertical Landing in Gusty Conditions

by

Parker C. Carter, Master of Science

Utah State University, 2023

Major Professor: Douglas Hunsaker, Ph.D.
Department: Mechanical and Aerospace Engineering

Ship based Unmanned Air Systems (UAS) face adverse conditions during flight in the form of ship airwake. The objectives of this work are to first evaluate the variation in forces and moments during the vertical landing phase of an approximated V-BAT model and second identify the vertical landing trajectory that minimizes the variation in forces and moments acting on the approximated V-BAT model. Methods for predicting the forces and moments acting on an aircraft are presented. The methods include reasonable approximations for the mass, inertia, and aerodynamics of an aircraft created from individual aircraft components and three-dimensional shapes. The V-BAT UAS model is described in detail. The model is simulated over a sweep of altitudes and bank angles. The simulation force and moment data are evaluated using two methods. The first method evaluates the average and standard deviation of the forces and moments at each altitude and bank angle. The second method creates contour plots of the bank angles that minimize the forces and moments at each altitude and time step. Using these methods produces series of bank angles that minimize the average force and force standard deviation as well as contour plots showing the angles that minimize the forces and moments at each altitude and time step. Five prescribed bank angle landing trajectories are simulated. The resulting forces and moments incurred during each simulation are piloted and compared to determine the landing trajectory that minimizes the variation in forces and moments. The results show using the minimum average and standard deviation force bank angle series while landing reduces the maximum force from 57.9 lbf to 19.25 lbf and 20.05 lbf respectively and the average force from 16.07 lbf to 6.27 lbf and 6.39 lbf respectively. Using the minimum average and standard deviation force bank angle series also reduces the maximum moment acting on the V-BAT from 20.62 lbf-ft to 14.32 lbf-ft and 18.49 lbf-ft respectively and the average moment acting on the V-BAT from 8.49 lbf-ft to 4.84 lbf-ft and 5.44 lbf-ft respectively. The results also show using the minimum force bank angle series, while landing, reduces the maximum force from 57.91 lbf to 20.05 lbf and the average force from 16.07 lbf to 4.83 lbf. Using the minimum force bank angle series also reduces the maximum moment acting on the V-BAT from 20.62 lbf-ft to 15.97 lbf-ft and the average moment acting on the V-BAT from 8.49 lbf-ft to 5.54 lbf-ft. In conclusion, the variation in forces and moments acting on the approximated V-BAT model are best minimized using a vertical landing trajectory bank angle series prescribed by the minimum force and moment contour plot data.

(142 pages)

PUBLIC ABSTRACT

Estimated Aerodynamic Forces and Moments and Optimal Orientation of the V-BAT Airframe During Vertical Landing in Gusty Conditions

Parker C. Carter

Ship based Unmanned Air Systems (UAS) are an important tool used by the United States Navy for situational awareness and short-range operations. Naval UAS are used to provide real-time intelligence, surveillance, reconnaissance, and target-acquisition while being low cost, mission flexible, and safe. Unfortunately UAS suffer disadvantages with respect to adverse environmental conditions caused by the air being displaced by the ship. The accumulation of one or more adverse conditions is known as airwake. To counteract the effects of airwake, the objectives of this work are to first evaluate the effect of forces and moments during the vertical landing phase of an aircraft model and second identify the vertical landing path that will minimize the forces and moments acting on the aircraft model.

To accomplish these objectives, an overview of the aircraft models component's mass and aerodynamic properties are given. An overview of the V-BAT UAV model and prescribed landing trajectories are discussed. The V-BAT model is simulated over a sweep of bank angles and altitudes. The simulation force and moment data are evaluated using two methods. The first method evaluates the average and standard deviation of the forces and moments at each altitude and bank angle. The second method creates contour plots of the bank angles that minimize the forces and moments acting on the V-BAT model at each altitude and time step. The results show, during landing, the forces and moments are minimised by following a series of bank angles taken from the minimum force and moment contour plot data.

To my parents, brother, and sisters.

ACKNOWLEDGMENTS

I would like to express heartfelt gratitude to my major advisor, Dr. Douglas Hunsaker, for all of his help, guidance, and encouragement in completing this research, as well as Ben Moulton for all of his assistance.

I would also like to thank my parents, grandparents, family, friends, and colleagues for all of their support and encouragement throughout my education.

Finally I would like to thank the members of the Naval Air Warfare Center Aircraft Division (NAWCAD) with the U.S. Navy Naval Air Systems Command (NAVAIR) for all their help and support.

Parker C. Carter

CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1
INTRODUCTION

1.1 Motivation

Unmanned Air Systems (UAS) in use on naval warships are becoming increasingly important for situational awareness and short-range operations. UAS such as the V-BAT 128 by Martin UAV [2] and MQ-8 Fire Scout by Northrop Grumman [3] are key tools for providing real-time intelligence, surveillance, reconnaissance, and target-acquisition. UAS have many advantages including low cost, increased flexibility, and improved safety but UAS also have some disadvantages when working within the adverse environmental conditions found around large ships.

A significant adverse condition ship based UAS face is the complex ship airwake flowing about the ship through which the UAS must travel. Ship airwake occurs when air is displaced by the body of the ship itself. As the air is forced to flow around the ship, burble is introduced to the flow causing the air to become turbulent. Complex ship airwake is induced as the ship cruses along or as winds blow over the ship. However, other adverse conditions including nearly constant unsteady wind, gusts, and thermal effects can contribute to the overall complexity of the ship airwake. Combining one, multiple, or all of these conditions creates a complex airwake that is difficult for aircraft of all shapes and sizes to navigate. Unfortunately, the effects of complex airwake are more pronounced for small UAS and rotor craft [4–9]. For this reason complex ship airwake is the main adverse condition this work seeks to counteract.

On account of these adverse conditions ship-based UAS have unique requirements in comparison to land-based UAS. These requirements result in the fact that UAS which are highly functional for land-based operations may not be suitable for ship-based operations. Currently UAS are loosely categorized as either fixed-wing configurations, which takeoff and land like a traditional aircraft, vertical-takeoff and landing (VTOL) configurations, which takeoff and land like a rotary aircraft, and hybrid VTOL UAS which can take off and land traditionally, vertically, or a combination of the two [10]. Each configuration presents unique challenges when maneuvering through the complex airwake to the deck touchdown point. The present work primarily evaluates the performance of VTOL-type maneuvers of aircraft near the ship, rather than maneuvers common to traditional fixed-wing aircraft, although the influence of fixed-wings attached to the aircraft are included in the analysis.

The ability of any aircraft to successfully navigate the takeoff and landing phases of ship-based operations depends on many factors, including the aircraft geometry, mass, inertia, control effects, control algorithms, aerodynamics, and operational conditions. Because of these factors, most aircraft follow a set of standards known as operating limitations during takeoff and landing.

The purpose of this work is to demonstrate how the variation in aerodynamic forces and moments relate to the aircraft's trajectory and show how to evaluate a trajectory with minimal variation in aerodynamic forces and moments during the aircraft's vertical landing. This is accomplished by first predicting the variation in forces and moments the aircraft experiences in order to follow a specified vertical landing trajectory then second compare a series of specified landing trajectories to determine the trajectory that minimizes the variation in forces and moments. This work uses a flight simulation tool called Simulator for Arbitrary Aircraft in Ship Airwakes (SAASHA) to accomplish this purpose.

1.2 SAASHA

SAASHA is a mass, aerodynamic, and flight simulation tool that predicts the forces and moments acting on an aircraft while flying through a predetermined trajectory and flight conditions. SAASHA can be used to simulate the forces and moments acting on any aircraft, as long as the aircraft can be approximated using a combination of simple geometric components including spheres, cylinders, cuboids, wings, and propellers.

SAASHA simulates the aircraft's flight by first accepting inputs for the atmospheric flight conditions, approximate aircraft geometry, and prescribed trajectory information.

SAASHA then calculates each of the aircraft component's weight and mass moment of inertia tensor. SAASHA then calculates the aircraft's center of gravity and uses the parallel axis theorem to calculate the aircraft's mass moment of inertia tensor about it's center of gravity. Once the moment of inertia is known, SAASHA calculates each of the component's aerodynamic properties to get the complete aerodynamic properties of the aircraft. Next SAASHA simulates the aircraft's flight through different flight conditions using the 6 degree-of-freedom (6 DoF) equations of motion in conjunction with a 4th order Runge Kutta integration method. After the simulation is complete, SAASHA returns the aircraft's time history data including the aircraft's position, orientation, velocity, and forces and moments at each time step. This data is saved as a CSV file for ease of use.

1.3 Alternative Software

Similar simulation work has been done on UAS and rotorcraft using a United States Air Force produced software called Digital DATCOM usually in conjunction with other software such as OpenVSP, [11] MATLAB, and SIMULINK [12–22]. Digital DATCOM used in conjunction with MATLAB has many advantages and could have been used to create a comparable simulator but there are several advantages to using SAASHA that DATCOM and Matlab do not have. The first advantage of SAASHA is that SAASHA does not lose accuracy at low reynolds numbers. Unfortunately, DATCOM loses accuracy at low Reynolds numbers on the order of Re = 100,000 [23, 24] which are most commonly experienced at low speed during takeoff and landing. Another advantage of SAASHA is individual components of the aircraft can be assigned a density and a weight. According to Vogeltanz [24] and Shankar, [21] when building geometries in Digital DATCOM, the DATCOM software assumes the material properties of the aircraft are homogeneous. This means DATCOM will not know the correct center of gravity location, so it will need to be calculated separately, otherwise, the simulation will produce incorrect results.

1.4 128 V-BAT

An approximate model of the Martin UAV V-BAT 128 is used in this work. The Martin UAV V-BAT 128, shown hovering in Fig. 1.1, is a VTOL UAS used by the United States Navy [25]. The V-BAT is primarily used for reconnaissance, surveillance, and target acquisition [2]. Two of the main advantages of the V-BAT are that it can be deployed from virtually anywhere and it takes up little room when disassembled. This makes the V-BAT ideal for naval use on ships where storage space is limited, and takeoff and landing conditions are not ideal.

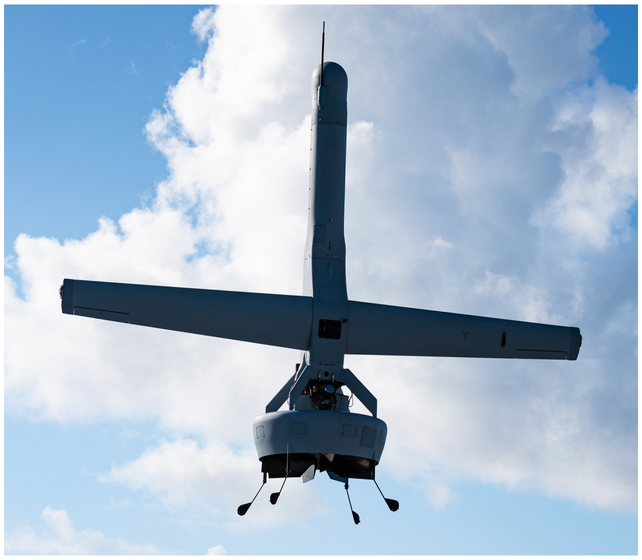

Fig. 1.1: Martin UAV V-BAT in hover position.

1.5 Airwake Database

When landing on a naval ship, the V-BAT must fly through the airwake of turbulent air being displaced by the ship. For this work, a database of complex ship airwake wind-gust velocity data has been provided by the U.S. Navy Naval Air Systems Command(NAVAIR). The airwake data describes the wind velocity in each direction within a 90.0 m x 67.0 m x 33.0 m rectangular volume as shown in Fig. 1.2. The wind velocity in each direction is known at every 1 meter junction from 0.0 m to 90.0 m in the x direction, -50.0 m to 17.0 m in the y direction and -7.0 m to 24.0 m in the z direction. The generic destroyer model used in the simulator and depicted in Fig. 1.2 was generated by the Canadian National Research Council. The airwake data used in this work was generated by NAVAIR using computational fluid dynamics(CFD). For more information on ship airwake and how CFD is used to generate airwake data see Polski [26–29] and Forsythe [30].

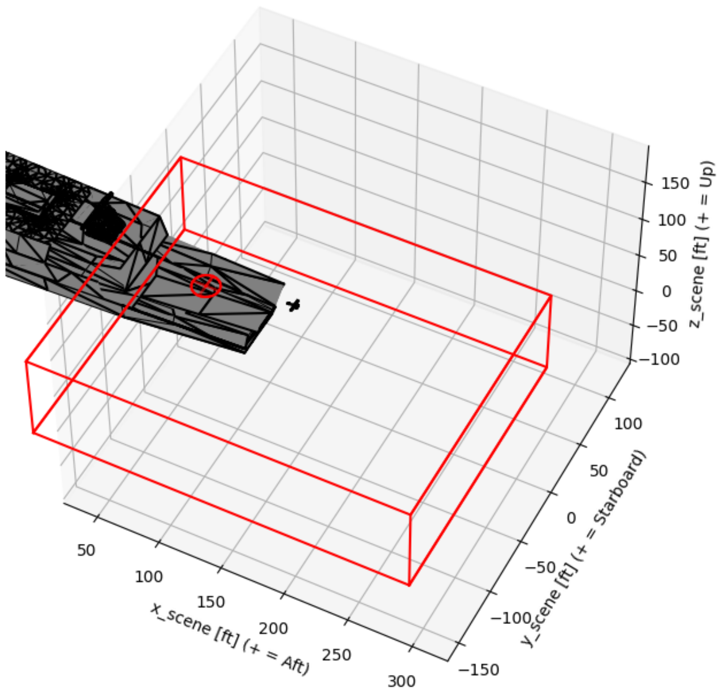

Fig. 1.2: Ship airwake rectangular volume.

1.6 Objectives

The objectives of this work are as follows:

1. Evaluate the variation in forces and moments during the vertical landing trajectory phase of an approximated V-BAT model.
2. Identify the vertical landing trajectory that minimizes the variation in forces and moments acting on the approximated V-BAT model.

To complete these objectives, an overview of the aircraft component's mass and aerodynamic computations used by SAASHA are given in Chapter 2 and Chapter 3. The 6 degree-of-freedom equations of motion are explained in Chapter 4. An overview of the V-BAT model, a prescribed landing trajectory, an example simulation, and a vertical landing trajectory simulation overview are given in Chapter 5. The landing trajectory results of this work are presented in Chapter 6. A summary and conclusion to this work is given in Chapter 7.

CHAPTER 2

A REVIEW OF AIRCRAFT MASS AND INERTIA PROPERTIES

The mass and inertia computations of each component type are presented in this chapter.

2.1 General Properties

As stated in the introduction, this work assumes the geometry of the aircraft can be approximated using a finite number of three dimensional shapes with constant density called objects or components.

2.1.1 Volume, Mass, and Weight

The volume of each object is calculated by integrating the region within the objects geometry. The volume integral is then

$$V = \iiint_V dV \tag{2.1}$$

The object's mass, m, with constant density, ρ, and volume, V, is calculated using

$$m = \rho V \tag{2.2}$$

The object's weight, W, is calculated using

$$W = mg \tag{2.3}$$

In this equation, g is the gravitational constant. Since each object is assumed to have constant density, the mass of each object can be defined by the object's volume and either the object's mass, density, or weight.

2.1.2 Center of Gravity

The location of an object's center of gravity (CG) is calculated using the mass distribution. Since each object is assumed to have a constant density, the moments about each axis are described by the volume integrals:

$$M_{yz} = \rho \iiint_V x \, dV \tag{2.4}$$

$$M_{xz} = \rho \iiint_V y \, dV \tag{2.5}$$

$$M_{xy} = \rho \iiint_V z \, dV \tag{2.6}$$

The object's CG coordinates, measured relative to the object's origin, are calculated using:

$$\bar{x} \equiv \frac{M_{yz}}{m} \tag{2.7}$$

$$\bar{y} \equiv \frac{M_{xz}}{m} \tag{2.8}$$

$$\bar{z} \equiv \frac{M_{xy}}{m} \tag{2.9}$$

2.1.3 Mass Moment of Inertia

An object's mass moment of inertia is described using the tensor

$$[\mathbf{I}] = \begin{bmatrix} I_{xx} & -I_{xy} & -I_{xz} \\ -I_{yx} & I_{yy} & -I_{yz} \\ -I_{zx} & -I_{zy} & I_{zz} \end{bmatrix} \tag{2.10}$$

Each object's moments and products of inertia are calculated using using the volume integrals [31]:

$$I_{xx} = \rho \iiint_V (y^2 + z^2)\, dV \tag{2.11}$$

$$I_{yy} = \rho \iiint_V (x^2 + z^2)\, dV \tag{2.12}$$

$$I_{zz} = \rho \iiint_V (x^2 + y^2)\, dV \tag{2.13}$$

$$I_{xy} = I_{yx} = \rho \iiint_V (xy)\, dV \tag{2.14}$$

$$I_{xz} = I_{zx} = \rho \iiint_V (xz)\, dV \tag{2.15}$$

$$I_{yz} = I_{zy} = \rho \iiint_V (yz)\, dV \tag{2.16}$$

In this work the mass moment of inertia is referred to as the *moment of inertia*.

The complete aircraft's mass and moment of inertia properties are estimated by summing each of the aircraft's individual object's mass and moment of inertia properties. The following sections describe the three dimensional shapes used to approximate the properties for a complete aircraft. The following sections include geometries for simple shapes such as cuboids and complex shapes such as wings and propellers. For more information on how each component's properties were derived, see Moulton and Hunsaker [1].

2.2 Rectangular Cuboid

A cuboid is a three dimensional six sided shape also known as a hexahedron. A rectangular cuboid has six sides that meet at right angles and are all rectangular in shape. The center of gravity for a rectangular cuboid is located at the center of the cuboid, halfway between each parallel face. The CG location also functions as the origin for the cuboid's Cartesian coordinate system, where each axis is parallel or perpendicular to the cuboid's faces. The cuboid's geometry is described by the length of the cuboid parallel to each axis, as shown in Fig. 2.1.

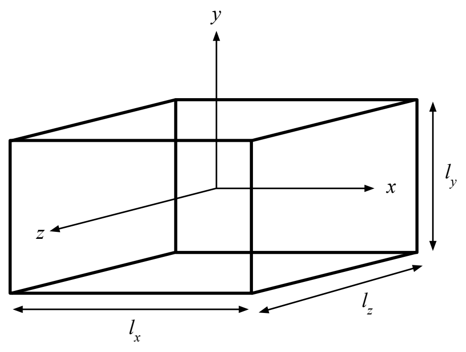

Fig. 2.1: Rectangular cuboid geometric definition and coordinate system.(Used w/permission [1])

The volume of a rectangular cuboid is calculated using the triple integral

$$V \equiv \iiint_V dV = \int_{-l_z/2}^{l_z/2}\int_{-l_y/2}^{l_y/2}\int_{-l_x/2}^{l_x/2} dxdydz = l_x l_y l_z \tag{2.17}$$

The inertia tensor of a rectangular cuboid, about it's CG, is calculated by integrating over the same limits of integration applied to Eqs. (2.11) - (2.16). The result of integration is given by

$$[\mathbf{I}] = \begin{bmatrix} \frac{m}{12}(l_y^2 + l_z^2) & 0 & 0 \\ 0 & \frac{m}{12}(l_x^2 + l_z^2) & 0 \\ 0 & 0 & \frac{m}{12}(l_x^2 + l_y^2) \end{bmatrix} \tag{2.18}$$

This equation is confirmed in Engineering Mechanics Statics and Dynamics by Hibbeler [32], Vector Mechanics for Engineers Statics and Dynamics by Beer, Johnson, and Eisenberg [33, 34], and Shigley's Mechanical Engineering Design [35].

For a hollow rectangular cuboid the inner dimensions of the cuboid are l_{x1}, l_{y1}, l_{z1} and the outer dimensions are l_{x2}, l_{y2}, l_{z2}. The volume of the cuboid is calculated as

$$V = V_2 - V_1 \tag{2.19}$$

In this equation $V_1 = l_{x1}l_{y1}l_{z1}$ and $V_2 = l_{x2}l_{y2}l_{z2}$. The inertia tensor of the hollow rectangular cuboid about it's center of gravity is then

$$[\mathbf{I}] = \begin{bmatrix} \frac{m}{12}\frac{V_2(l_{y2}^2+l_{z2}^2)-V_1(l_{y1}^2+l_{z1}^2)}{V_2-V_1} & 0 & 0 \\ 0 & \frac{m}{12}\frac{V_2(l_{x2}^2+l_{z2}^2)-V_1(l_{x1}^2+l_{z1}^2)}{V_2-V_1} & 0 \\ 0 & 0 & \frac{m}{12}\frac{V_2(l_{x2}^2+l_{y2}^2)-V_1(l_{x1}^2+l_{y1}^2)}{V_2-V_1} \end{bmatrix} \tag{2.20}$$

2.3 Hollow Cylinder

A hollow cylinder is described by an inner radius, R_1, an outer radius, R_2, and a length, h. The origin of a hollow cylinder is at the CG of the cylinder, halfway between each end. The Cartesian coordinate system is located at the cylinder's CG, with the x axis aligned with the cylinder's axis of rotation as shown in Fig. 2.2.

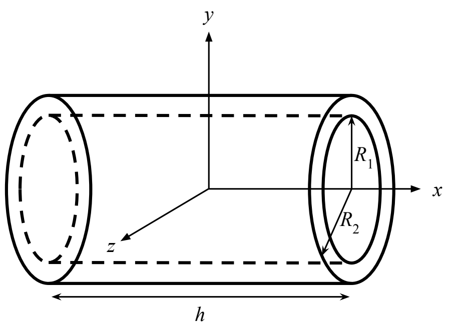

Fig. 2.2: Cylinder geometric definition and coordinate system.(Used w/permission [1])

The volume of a hollow cylinder is calculated by converting Eqs. (2.1) and (2.11)-(2.16) to cylindrical coordinates, (x, r, φ) where $r = y^2 + z^2$, and integrating gives

$$V \equiv \iiint_V dV = \int_0^{2\pi}\int_{R_1}^{R_2}\int_{-h/2}^{h/2} dxrdrd\varphi = \pi h(R_2^2 - R_1^2) \tag{2.21}$$

The inertia tensor of the cylinder about it's CG is

$$[\mathbf{I}] = \begin{bmatrix} \frac{m}{2}(R_2^2 + R_1^2) & 0 & 0 \\ 0 & \frac{m}{12}[3(R_2^2 + R_1^2) + h^2] & 0 \\ 0 & 0 & \frac{m}{12}[3(R_2^2 + R_1^2) + h^2] \end{bmatrix} \tag{2.22}$$

The process for finding the inertia of a solid cylinder is the same except $R_1 = 0$.

2.4 Hollow Sphere

A hollow sphere is described by an inner radius R_1 and an outer radius R_2. The origin of the sphere is located at the sphere's CG as shown in Fig. 2.3.

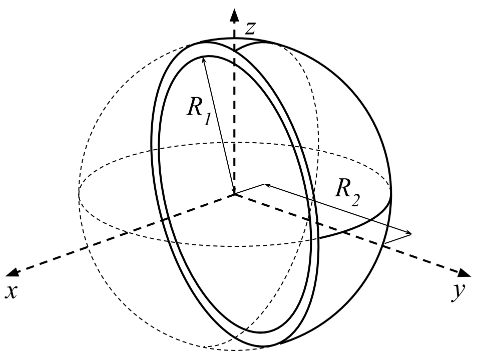

Fig. 2.3: Sphere geometric definition and coordinate system.(Used w/permission [1])

The surface area of a sphere is $4\pi r^2$ therefore, the differential volume of the sphere is $dV = 4/3\pi r^2 dr$. Applying this to the volume integral of a hollow sphere gives

$$V \equiv \iiint_V dV = \int_{R_1}^{R_2} 4\pi r^2 dr = \frac{4}{3}\pi(R_2^3 - R_1^3) \tag{2.23}$$

The inertia tensor of the sphere about it's CG is

$$[\mathbf{I}] = \begin{bmatrix} \frac{2}{5}m\frac{(R_2^5-R_1^5)}{(R_2^3-R_1^3)} & 0 & 0 \\ 0 & \frac{2}{5}m\frac{(R_2^5-R_1^5)}{(R_2^3-R_1^3)} & 0 \\ 0 & 0 & \frac{2}{5}m\frac{(R_2^5-R_1^5)}{(R_2^3-R_1^3)} \end{bmatrix} \tag{2.24}$$

The process for finding the inertia of a solid sphere is the same except $R_1 = 0$.

2.5 Wing Segment

This section outlines the process to approximate the volume, mass, and inertia properties for a single simple wing segment. This section assumes any lifting surface can be approximated by a finite number of simple wing segments. The section also assumes that simple wing geometries can be summed together to approximate more complex wing geometries.

2.5.1 Geometry and Coordinate Definitions

A single wing segment is fully defined by the span, b, root chord, c_r, tip chord, c_t, root airfoil thickness, τ_r, tip airfoil thickness, τ_t, sweep angle, Λ, dihedral angle, Γ, and airfoil thickness distribution, $\mu(x_a/c)$. A single wing segment with the wing's geometric properties is illustrated in Fig. 2.4. Note the span,b, runs parallel to the y-axis.

For a wing segment, the center of gravity is not known until the wing is analyzed. Therefore, the origin of the finite wing will not coincide with the CG of the wing. For a finite simple wing, the origin of the wing is placed at the wing-root quarter-chord as shown in Fig. 2.4. The x-axis follows the chord-line pointing out of the root leading edge, the y-axis goes in the same direction as a right wing, and the z-axis points toward the bottom of the airfoil.

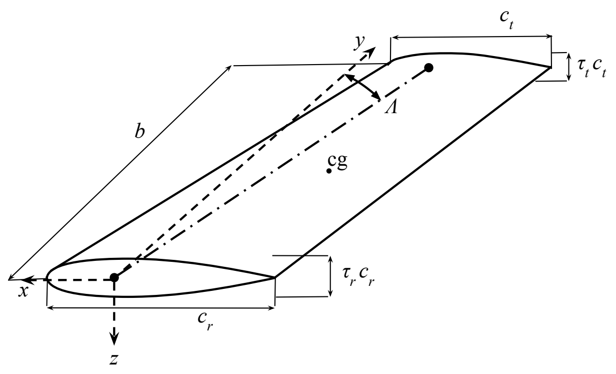

Fig. 2.4: Wing-segment geometry definitions.(Used w/permission [1])

The type of airfoil used on the wing segment has a large impact on the mass and inertia properties of the wing. This work assumes the airfoil's mass properties are dominated by the airfoil's thickness distribution. This work also assumes the wing's camber effects are negligible. The traditional airfoil coordinate system is located at the leading edge of the airfoil and has the x-axis, x_a, pointing along the chord-line toward the trailing edge and the y-axis, y_a, pointing upward out of the airfoil normal to the x-axis as shown in Fig. 2.5.

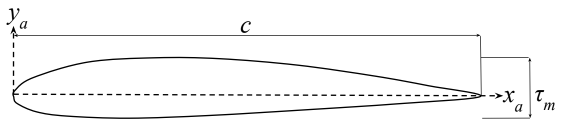

Fig. 2.5: Airfoil coordinate frame.(Used w/permission [1])

The methods presented in this work can use different types of thickness distributions but only two methods are demonstrated in this work, NACA 4-Digit-Series and diamond shaped airfoils. For more information on different thickness distributions see Moulton and Hunsaker [1].

The volume of the wing segment is calculated by integrating over these bounds in Eq. (2.1). The volume of the wing segment is then

$$V = \frac{b}{12}\kappa_a \upsilon_0 \tag{2.25}$$

The wing's center of gravity is calculated by computing the volume integrals given by Eqs. (2.4)–(2.6). The sweep angle must be accounted for in computing M_{yz} because sweep has an effect on the x coordinate of the wing's mass. The moment M_{yz} is computed by applying the shift $-y\tan\Lambda$ to the x coordinate. This shift occurs due to sweep. The equation for M_{yz} is then

$$M_{yz} = \rho\iiint_V (x - y\tan\Lambda)\, dV = \rho\iiint_V x\, dV - \rho\tan\Lambda\iiint_V y\, dV \tag{2.26}$$

The effects of sweep do not apply to the integrations for M_{xz} or M_{xy} in Eqs. (2.5) and (2.6) because there is no x terms in the integrals and sweep has no effect on the wing's y or z mass coordinates. Sweep has different effects depending on whether the wing is the right or the left wing. The coefficient δ is used in subsequent equations to account for this. The δ coefficient is defined as

$$\delta \equiv \begin{cases} 1, & \text{right wing} \\ -1, & \text{left wing} \end{cases} \tag{2.27}$$

M_{yz},M_{xz}, and M_{xy} are found using the volume integrals [1]:

$$M_{yz} = \rho[S_{1,0,0} - \tan\Lambda S_{0,1,0}] = -\rho\frac{b}{240}\left[3\kappa_b\upsilon_1 + 4b\kappa_c\upsilon_0\tan\Lambda\right] \tag{2.28}$$

$$M_{xz} = \rho\delta S_{0,1,0} = \rho\delta\frac{b^2}{60}\kappa_c\upsilon_0 \tag{2.29}$$

$$M_{xy} = \rho S_{0,0,1} = 0 \tag{2.30}$$

The location of the wing's center of gravity relative to the wing's origin is found by using Eq. (2.25) in Eq. (2.2) then applying the result with Eqs. (2.28) through (2.30) in Eqs. (2.7) through (2.9). The center of gravity location is then:

$$\bar{x} \equiv \frac{M_{yz}}{m} = -\frac{3\kappa_b\upsilon_1 + 4b\kappa_c\upsilon_0\tan\Lambda}{20\kappa_a\upsilon_0} \tag{2.31}$$

$$\bar{y} \equiv \frac{M_{xz}}{m} = \delta b\frac{\kappa_c}{5\kappa_a} \tag{2.32}$$

$$\bar{z} \equiv \frac{M_{xy}}{m} = 0 \tag{2.33}$$

Neglecting twist and camber in the wing causes the z coordinate of the CG to go to zero. Note the sweep only effects the center of gravity's x coordinate.

2.5.2 Inertia Tensor

The inertia tensor about the wing's origin is calculated by applying appropriate limits to the volume integrals given in Eqs. (2.11)–(2.16). The inertia tensor about the wing's origin is found by applying the shift due to sweep. Solving the volume integrals result in [1]

$$[\mathbf{I}]_o = \begin{bmatrix} I_{xx_o} & -I_{xy_o} & -I_{xz_o} \\ -I_{yx_o} & I_{yy_o} & -I_{yz_o} \\ -I_{zx_o} & -I_{zy_o} & I_{zz_o} \end{bmatrix} \tag{2.34}$$

where the components are

$$I_{xx_o} = \rho(S_{0,2,0} + S_{0,0,2}) = \rho\frac{b}{3360}\left(56b^2\kappa_f\upsilon_0 + \kappa_g\upsilon_3\right) \tag{2.35}$$

$$\begin{aligned} I_{yy_o} &= \rho(S_{2,0,0} + S_{0,0,2} + S_{0,2,0}\tan^2\Lambda - 2S_{1,1,0}\tan\Lambda) \\ &= \rho\frac{b}{10080}\left[84b\left(2b\kappa_f\tan^2\Lambda + \kappa_d\tan\Lambda\right)\upsilon_1 + 49\kappa_e\upsilon_2 + 3\kappa_g\upsilon_3\right] \end{aligned} \tag{2.36}$$

$$\begin{aligned} I_{zz_o} &= \rho\left[S_{2,0,0} + \left(\tan^2\Lambda + 1\right)S_{0,2,0} - 2S_{1,1,0}\tan\Lambda\right] \\ &= \rho\frac{b}{1440}\left[12b\left\{2b\left(\tan^2\Lambda + 1\right)\kappa_f\upsilon_0 + \kappa_d\upsilon_1\tan\Lambda\right\} + 7\kappa_e\upsilon_2\right] \end{aligned} \tag{2.37}$$

$$I_{xy_o} = I_{yx_o} = \rho\delta(S_{1,1,0} - S_{0,2,0}\tan\Lambda) = -\rho\delta\frac{b^2}{240}\left[4b\kappa_f\upsilon_0\tan\Lambda + \kappa_d\upsilon_1\right] \tag{2.38}$$

$$I_{xz_o} = I_{zx_o} = \rho(S_{1,0,1} - S_{0,1,1}\tan\Lambda) = 0 \tag{2.39}$$

$$I_{yz_o} = I_{zy_o} = \rho\delta S_{0,1,1} = 0 \tag{2.40}$$

and the kappa values and upsilon values are given in Moulton and Hunsaker [1].

Using Eq. (2.2) relating the mass to the volume and Eq. (2.25) for wing volume, the tensor components can be converted to be expressed in terms of mass rather than density as given by:

$$I_{xx_o} = m\left[\frac{56b^2\kappa_f\upsilon_0 + \kappa_g\upsilon_3}{280\kappa_a\upsilon_0}\right] \tag{2.41}$$

$$I_{yy_o} = m\left[\frac{84b\left(2b\kappa_f\tan^2\mathit{\Lambda} + \kappa_d\tan\mathit{\Lambda}\right)\upsilon_1 + 49\kappa_e\upsilon_2 + 3\kappa_g\upsilon_3}{840\kappa_a\upsilon_0}\right] \tag{2.42}$$

$$I_{zz_o} = m\left[\frac{12b\left\{2b\left(\tan^2\mathit{\Lambda} + 1\right)\kappa_f\upsilon_0 + \kappa_d\upsilon_1\tan\mathit{\Lambda}\right\} + 7\kappa_e\upsilon_2}{120\kappa_a\upsilon_0}\right] \tag{2.43}$$

$$I_{xy_o} = I_{yx_o} = -\delta bm\left[\frac{4b\kappa_f\upsilon_0\tan\mathit{\Lambda} + \kappa_d\upsilon_1}{20\kappa_a\upsilon_0}\right] \tag{2.44}$$

$$I_{xz_o} = I_{zx_o} = 0 \tag{2.45}$$

$$I_{yz_o} = I_{zy_o} = 0 \tag{2.46}$$

The inertia tensor about the wing-segment coordinate frame is found by plugging the inertia components given in Eqs. (2.35)–(2.40) and (2.41)–(2.46) into Eq. (2.34). The inertia tensor of the wing segment about its CG is needed to compute the inertia tensor of the entire aircraft. The parallel axis theorem states that the inertia tensor about an arbitrary point, $[\mathbf{I}]_1$, is related to the inertia tensor about the center of gravity, $[\mathbf{I}]$, according to the relationship

$$[\mathbf{I}]_1 = [\mathbf{I}] + m\left[(\mathbf{s}\cdot\mathbf{s})[\mathbf{E}] - \mathbf{s}\mathbf{s}^T\right] \tag{2.47}$$

$$[\mathbf{I}] = [\mathbf{I}]_o - m\left[(\mathbf{s}\cdot\mathbf{s})[\mathbf{E}] - \mathbf{s}\mathbf{s}^T\right] \tag{2.48}$$

In this equation, $\mathbf{s}$ is a vector from the wing-segment CG to the wing-segment origin as given by

$$\mathbf{s} = -\begin{Bmatrix} \bar{x} \\ \bar{y} \\ \bar{z} \end{Bmatrix} \tag{2.49}$$

The term "$\mathbf{s}$" can be computed for any wing segment using Eqs. (2.31) through (2.33). The inertia tensor of the wing segment about the wing's CG is found by plugging Eq. (2.49) into Eq. (2.48) and simplifying. The CG is then

$$[\mathbf{I}] = [\mathbf{I}]_o - m \begin{bmatrix} \bar{y}^2 + \bar{z}^2 & -\bar{x}\bar{y} & -\bar{x}\bar{z} \\ -\bar{x}\bar{y} & \bar{x}^2 + \bar{z}^2 & -\bar{y}\bar{z} \\ -\bar{x}\bar{z} & -\bar{y}\bar{z} & \bar{x}^2 + \bar{y}^2 \end{bmatrix} \tag{2.50}$$

The inertia of the wing about its axis system is given by

$$[\mathbf{I}] = [\mathbf{I}]_o - m \begin{bmatrix} \bar{y}^2 + \bar{z}^2 & -\bar{x}\bar{y} & -\bar{x}\bar{z} \\ -\bar{x}\bar{y} & \bar{x}^2 + \bar{z}^2 & -\bar{y}\bar{z} \\ -\bar{x}\bar{z} & -\bar{y}\bar{z} & \bar{x}^2 + \bar{y}^2 \end{bmatrix} \tag{2.51}$$

see Moulton and Hunsaker [1].

2.6 Rotor

Aircraft use different types of rotors depending on the desired performance characteristics of the aircraft. Types of rotors can include propellers, jet turbines, helicopter blades etc. In this work a rotor is defined by the number of rotor blades, N_b, rotor diameter, d_r, hub diameter, d_h, hub height, h_h, blade root chord, c_r, blade tip chord, c_t, blade root airfoil thickness, τ_r, and blade tip airfoil thickness, τ_t. Similar to the wing twist and airfoil camber, the rotor's blade pitch and airfoil camber are assumed to be negligible. Figure 2.6 shows a 3 bladed rotor with the Cartesian coordinate system origin located at the center of the hub. Since the rotor's camber and pitch are negligible, the location of the origin at the center of the hub is also the rotor's CG.

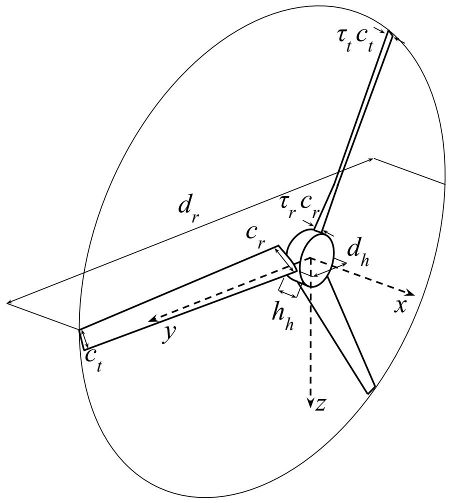

Fig. 2.6: Rotor geometry definitions and coordinate frame.(Used w/permission [1])

The term $r_r = d_h/2$ represents the blade's radius at the root while $r_t = d_r/2$ represents the blade's radius at the tip. SAASHA assumes the chord and maximum airfoil thickness ratios vary linearly as a function of the radius, r, according to the relation

$$c(r) = (c_t - c_r)\frac{(r - r_r)}{(r_t - r_r)} + c_r \tag{2.52}$$

$$\tau_m(r) = (\tau_t - \tau_r)\frac{(r - r_r)}{(r_t - r_r)} + \tau_r \tag{2.53}$$

The dimensional maximum airfoil thickness at any radius is then

$$t(r) = \tau_m(r)c(r) \tag{2.54}$$

2.6.1 Volume

The rotor's volume is approximated by summing the volume of the hub with the volume of the rotor blades. The volume of the hub, V_h, is approximated as the volume of a cylinder with the cylinder radius equal to the hub radius, r_r, and the cylinder height equal to the hub height, h_h, as shown in

$$V_h = \pi h_h r_r^2 \tag{2.55}$$

The volume of a single rotor blade is approximated using the same method as used for approximating the volume of a wing from the previous section, except the span in Eq. (2.25) is replaced with the with the difference between the rotor radius and hub radius. The total volume of the blades is then calculated by multiplying the single blade volume by the number of blades on the rotor, N_b, as

$$V_b = N_b \frac{r_t - r_r}{12} \kappa_a \upsilon_0 \tag{2.56}$$

The total rotor volume is the sum of the volume of the hub and the volume of the blades

$$V = V_h + V_b \tag{2.57}$$

Given the rotor's total weight or mass, the rotor's density is calculated by rearranging Eqs. (2.2) and (2.3) and applying the total volume estimated from Eq. (2.57). After calculating the rotor's density, the hub and blades mass and weight are individually calculated, using the volumes found in Eqs. (2.55) and (2.56). The mass and weight of the hub and blades will be used in the calculating the rotor's inertia tensor.

2.6.2 Inertia Tensor

The rotor's inertia tensor is calculated by summing the hub and blades inertia tensors. The hubs inertia tensor is approximated using Eq. (2.22) with the inner radius, R_1, equal to zero, the outer radius, R_2, equal to the hub radius, r_r, the height of the cylinder equal to the hub height, h_h and the mass of the cylinder equal to the mass of the hub, m_h. The inertia tensor is then

$$[\mathbf{I}]_h = \begin{bmatrix} \frac{m_h}{2} r_r^2 & 0 & 0 \\ 0 & \frac{m_h}{12}\left(3r_r^2 + h_h^2\right) & 0 \\ 0 & 0 & \frac{m_h}{12}\left(3r_r^2 + h_h^2\right) \end{bmatrix} \tag{2.58}$$

where m_h is the mass of the hub.

The rotor blades inertia tensor at any moment in time depends on the blades orientation about the rotor's axis. The rotor's inertia tensor is a function of time because rotors are usually spinning during flight. Because of this, an approximate static inertia tensor can be developed. To find an approximate static inertia tensor, the rotor blades mass is assumed to be distributed within a circular disk with the same outer diameter as the rotor diameter. The disks differential volume is assumed to be equal to the rotor blades differential volume at each radial distance and same radial location. This is accomplished by setting the rotor blades area equal to the area of a thin disk at any given radial distance. The area of a thin cylinder of radius, r, and height, h, is given by

$$A(r) = 2\pi r h \tag{2.59}$$

The rotor blades total area passing through a given radius is

$$A(r) = N_b \tau_m(r) c(r)^2 \upsilon_0 \tag{2.60}$$

In this equation $\tau_m \upsilon_0$ accounts for the airfoil area nondimensionalized by c^2. Setting Eq. (2.59) equal to Eq. (2.60) and solving for the disk height gives the height of the disk as a function of radial position as

$$h(r) = \frac{N_b \tau_m(r) c(r)^2 \upsilon_0}{2\pi r} \tag{2.61}$$

The system is converted to polar coordinates to simplify computing the inertia components. The axis is aligned with the rotor's axis where the x, y, and z coordinate conversions are:

$$x = x$$

$$y = r\cos\varphi$$

$$z = r\sin\varphi$$

The solution to the volume integrals result in [1]

$$[\mathbf{I}]_b = \begin{bmatrix} I_{xx} & 0 & 0 \\ 0 & I_{yy} & 0 \\ 0 & 0 & I_{zz} \end{bmatrix} \tag{2.62}$$

where the coefficients are

$$I_{xx} = m_b \frac{T_{0,2,0} + T_{0,0,2}}{V_b} = m_b \left[\frac{r_r^3 \gamma_l + r_r^2 r_t \gamma_m + r_r r_t^2 \gamma_n + r_t^3 \gamma_o}{5\left(r_t - r_r\right)\kappa_a} \right] \tag{2.63}$$

$$I_{yy} = m_b \frac{T_{2,0,0} + T_{0,0,2}}{V_b} = I_{zz} = m_b \frac{T_{2,0,0} + T_{0,2,0}}{V_b} \tag{2.64}$$

$$I_{xy} = I_{yx} = I_{xz} = I_{zx} = I_{yz} = I_{zy} = 0 \tag{2.65}$$

with

$$m_b = \rho_r V_b \tag{2.66}$$

and

$$T_{2,0,0} = N_b^3 \upsilon_0^3 \frac{\Psi}{13440\pi^2 r_r r_t (r_r - r_t)^9}$$

where the term Ψ is

$$\Psi = \gamma_a r_r^{10} + \gamma_b r_r^9 r_t + \gamma_c r_r^8 r_t^2 + \gamma_d r_r^7 r_t^3 + \gamma_e r_r^6 r_t^4 + \gamma_f r_r^5 r_t^5 + \gamma_g r_r^4 r_t^6 + \gamma_h r_r^3 r_t^7 + \gamma_i r_r^2 r_t^8 + \gamma_j r_r r_t^9 + \gamma_k r_t^{10}$$

$$T_{0,2,0} = T_{0,0,2} = \frac{1}{120} N_b \upsilon_0 \left(r_r^3 \gamma_l + r_r^2 r_t \gamma_m + r_r r_t^2 \gamma_n + r_t^3 \gamma_o \right)$$

The γ values in the equations above are quite lengthy and are included in Moulton and Hunsaker [1].

The rotor's total inertia tensor, about the rotor's origin, is calculated by summing the hub and blades inertia tensors as given by

$$[\mathbf{I}] = [\mathbf{I}]_h + [\mathbf{I}]_b \tag{2.67}$$

2.6.3 Angular Momentum

Spinning rotors create angular momentum that can effect an aircraft's dynamics whether or not the rotor generates thrust. The rotor's angular momentum about it's own axis is computed from the product of the rotor's inertia tensor and angular velocity. The rotor's inertia tensor is shown in Eq. (2.67). The rotor's angular velocity is only about the x axis due to the choice of rotor coordinate system. The rotor's angular velocity in vector form is

$$\omega = \begin{Bmatrix} \omega_x \\ 0 \\ 0 \end{Bmatrix} \tag{2.68}$$

The rotor's angular momentum vector is the product of the rotor's inertia tensor and angular velocity vector, which gives

$$\mathbf{h} = [\mathbf{I}]\omega = [\mathbf{I}] \begin{Bmatrix} \omega_x \\ 0 \\ 0 \end{Bmatrix} = \begin{Bmatrix} I_{xx}\omega_x \\ 0 \\ 0 \end{Bmatrix} \tag{2.69}$$

Note the rotor's angular momentum is in the component coordinate system. The rotor's angular momentum is used to find the angular momentum of the entire aircraft from one or more spinning rotors.

2.7 Mass and Inertial Properties of the Complete Aircraft

The entire aircraft's total mass, center-of-gravity location, and inertia tensor are found using the mass and inertia properties of each of the aircraft's individual components. Each component's influence on the entire aircraft depends on the component's position and orientation relative to the aircraft's origin and coordinate system.

2.7.1 Coordinate Frames

This work uses the traditional body-fixed coordinate frame for an aircraft. In this coordinate frame the x-axis points out of the nose of the aircraft, the y-axis points out of the right wing, and the z-axis points out of the bottom of the aircraft and is orthogonal to the x and y axes. This coordinate system is illustrated in Fig. 2.7. This coordinate system is located at the aircraft's CG. The inertia tensor of an aircraft is usually reported relative to this coordinate frame.

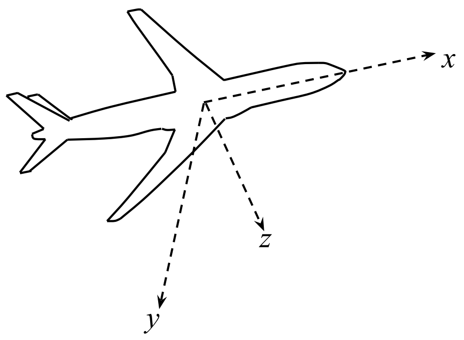

Fig. 2.7: Drawing of the aircraft coordinate frame.(Used w/permission [1])

The matrix $[\mathbf{R}]$ is a rotation matrix which is used to convert the components of a vector from the component coordinate system to the aircraft coordinate system as shown by

$$\begin{Bmatrix} v_{x_a} \\ v_{y_a} \\ v_{z_a} \end{Bmatrix} = [\mathbf{R}] \begin{Bmatrix} v_{x_c} \\ v_{y_c} \\ v_{z_c} \end{Bmatrix} \tag{2.70}$$

The matrix R in terms of Euler angles and quaternions is

$$[\mathbf{R}] = \begin{bmatrix} C_\theta C_\psi & S_\phi S_\theta C_\psi - C_\phi S_\psi & C_\phi S_\theta C_\psi + S_\phi S_\psi \\ C_\theta S_\psi & S_\phi S_\theta S_\psi + C_\phi C_\psi & C_\phi S_\theta S_\psi - S_\phi C_\psi \\ -S_\theta & S_\phi C_\theta & C_\phi C_\theta \end{bmatrix}$$

$$= \begin{bmatrix} e_x^2 + e_0^2 - e_y^2 - e_z^2 & 2(e_x e_y - e_z e_0) & 2(e_x e_z + e_y e_0) \\ 2(e_x e_y + e_z e_0) & e_y^2 + e_0^2 - e_x^2 - e_z^2 & 2(e_y e_z - e_x e_0) \\ 2(e_x e_z - e_y e_0) & 2(e_y e_z + e_x e_0) & e_z^2 + e_0^2 - e_x^2 - e_y^2 \end{bmatrix} \tag{2.71}$$

Although an aircraft can have non-zero values for Euler angles and quaternions, most aircraft do not. With most aircraft the rotation matrix $[\mathbf{R}]$ for the wing is a function of the dihedral only. Dihedral is a rotation about the x axis, this means that the Euler angles θ and ψ both go to zero. Using these Euler angles in Eq. (2.71) gives

$$[\mathbf{R}] = \begin{bmatrix} 1 & 0 & 0 \\ 0 & C_\phi & -S_\phi \\ 0 & S_\phi & C_\phi \end{bmatrix} \tag{2.72}$$

For a right wing the angle $\phi = -\mathit{\Gamma}$, and for a left wing the angle $\phi = \mathit{\Gamma}$. In this relationship a positive value for the coefficient $\mathit{\Gamma}$ represents a positive dihedral angle. A general relation for any wing dihedral is derived using Eq. (2.27), as shown in

$$\phi = -\delta \mathit{\Gamma} \tag{2.73}$$

2.7.2 Position

The position of each component is defined relative to the aircraft's coordinate system in Cartesian coordinates see Moulton and Hunsaker. [1].

2.7.3 Total Inertia Tensor

The complete system's total inertia tensor is computed by summing each component's inertia influence. The problem with this is most of the component's CG is not coincident with the entire system's CG and most of the component's ordinates are also not aligned with the entire aircraft's coordinate system. To fix this problem each component's inertia tensor must be rotated and translated before they can be summed together to get the aircraft's total inertia tensor. See Moulton and Hunsaker [1] for the rotation and translation process. The component's inertia tensor evaluated at the aircraft's CG location is found using the parallel axis theorem as given by

$$[\mathbf{I}]_c = [\mathbf{I}]_a + m\left[(\mathbf{s}\cdot\mathbf{s})[\mathbf{E}] - \mathbf{s}\mathbf{s}^{\mathrm{T}}\right] \tag{2.74}$$

where m is the component's mass, and $[\mathbf{E}]$ is again the 3×3 identity matrix. The inertia of the entire aircraft is then given by

$$[\mathbf{I}]_{\text{aircraft}} = \sum_{i=1}^{N}[\mathbf{I}]_{c_i} \tag{2.75}$$

The aircraft's inertia tensor given in Eq. (2.75) is about the aircraft's center of gravity. The aircraft's inertia tensor about it's origin is sometimes useful to know and can be calculated using the parallel axis theorem as given by

$$[\mathbf{I}]_{\text{aircraft}_O} = [\mathbf{I}]_{\text{aircraft}} + m_{\text{aircraft}}\left[(\mathbf{s}\cdot\mathbf{s})[\mathbf{E}] - \mathbf{s}\mathbf{s}^{\mathrm{T}}\right] \tag{2.76}$$

In this equation $\mathbf{s}$ is a vector connecting the aircraft's center of gravity to the aircraft's origin, $\mathbf{s} = -\mathbf{r}_{\text{aircraft}}$.

2.8 Angular Momentum of Spinning Rotors

An aircraft can experience angular momentum when the aircraft is not rotating if the aircraft is equipped with rotors. The angular momentum caused by rotating machinery such as propellers, jet engines, or other rotor blades is called *gyroscopic effects.* Each rotor's angular momentum about it's axis or rotation is calculated using Eq. (2.69). The entire aircraft's gyroscopic vector is calculated by summing the gyroscopic effects from each individual component as given by

$$\mathbf{h}_{\text{aircraft}} \equiv \begin{Bmatrix} h_{x_b} \\ h_{y_b} \\ h_{z_b} \end{Bmatrix} = \sum_{i=1}^{n} [\mathbf{R}]_i \mathbf{h}_i \tag{2.77}$$

In this summation n is the number of rotors and $[\mathbf{R}]$ is the rotation matrix of the component computed from Eq. (2.71). The components of the total gyroscopic vector, $\mathbf{h}_{\text{aircraft}}$, are usually written in tensor form when applying gyroscopic effects the to the aircraft's equations of motion as given by

$$[\mathbf{h}]_{\text{aircraft}} \equiv \begin{bmatrix} 0 & -h_{z_b} & h_{y_b} \\ h_{z_b} & 0 & -h_{x_b} \\ -h_{y_b} & h_{x_b} & 0 \end{bmatrix} \tag{2.78}$$

2.9 Additional Comments

From the equations above the mass and inertia properties of an entire aircraft can be approximated by summing a few simple components. Because of this, largely complex geometries can be broken down into simper geometries which can be evaluated. The inertia properties can also be summed to create more complex components. An example of this could be the applied to the hollow cylinder. A hollow cylinder could be used to model the cabin of an airliner and rows of cuboids could be used to model the rows of passenger seats inside the airliner. voids within an aircraft can be modeled by adding a component, such as a cylinder, to the inside of another component, such as a wing, then setting the density of the object within, the cylinder, to the negative of the external object. This method can be used to create cavities in which other structures can be added, such as actuators, internal engine turbine blades, payloads, hydraulic systems, etc. Hence, these simple shapes can be used to model much more complex internal structures and mass distributions.

Now that each component's mass and inertia computations have been presented, each component's aerodynamic force and moment computations are presented in Chapter 3.

CHAPTER 3
LOW-FIDELITY AERODYNAMIC MODELING OF BASIC GEOMETRIES

The aerodynamic force and moment computations acting on each component type are presented in this chapter.

3.1 Aerodynamic Model

The aerodynamic forces and moments acting on an aircraft depend on the aircraft's outward geometry. These forces and moments can be approximated using the 5 component types discussed in the previous section, rectangular cuboids, cylinders, spheres, wings, and rotors. Each component's aerodynamic properties depend on the velocity of the fluid surrounding the component relative to the component itself. The fluid velocity at the component's center of gravity varies depending on the surrounding air's velocity vector orientation relative to the aircraft's coordinate system $\mathbf{V}_\infty$ and the rotation rate of the aircraft about the aircraft's center of gravity as given by

$$\omega \equiv \begin{Bmatrix} p \\ q \\ r \end{Bmatrix} \tag{3.1}$$

Note the freestream velocity, $\mathbf{V}_\infty$, is the opposite sign from the aircraft's velocity vector ie. $\mathbf{V} = (u, v, w) = -\mathbf{V}_\infty$. If the aircraft's rotation rates and the freestream velocity are known relative to the aircraft coordinate system then the fluid's velocity in the component coordinate system at the component's center of gravity can be found using the equation

$$\mathbf{V}_i = [\mathbf{R}]^{-1}(\mathbf{V}_\infty + \mathbf{P}_i \times \omega) \tag{3.2}$$

In this equation, $\mathbf{P}_i \equiv \mathbf{r}_i - \mathbf{r}_{\text{aircraft}}$ is the vector connecting the component to the aircraft's center of gravity and $[\mathbf{R}]$ is the rotation matrix given in Eq. (2.71). The local velocity vector can also be expressed as

$$\mathbf{V}_i = V\mathbf{u}_i \tag{3.3}$$

In this equation, V represents the local velocity magnitude and $\mathbf{u}_i$ is a unit vector in the direction of the local velocity vector

$$\mathbf{u}_i \equiv \frac{\mathbf{V}_i}{V} \tag{3.4}$$

The local velocity vector is used to approximate each component's pseudo aerodynamic force, $\mathbf{F}_i$, and moment, $\mathbf{M}_i$, in the component coordinate system. The pseudo aerodynamic forces and moments are then rotated from the component's coordinate system to the aircraft's coordinate system about the aircraft's center of gravity and summed together to compute the entire pseudo aerodynamic force and moment vectors

$$\mathbf{F}_{\text{total}} = \sum_{i=1}^{N} [\mathbf{R}]_i \mathbf{F}_i \tag{3.5}$$

$$\mathbf{M}_{\text{total}} = \sum_{i=1}^{N} ([\mathbf{R}]_i \mathbf{M}_i + \mathbf{P}_i \times [\mathbf{R}]_i \mathbf{F}_i) \tag{3.6}$$

The pseudo aerodynamic forces and moments of each component type depends on the local fluid velocity, $\mathbf{V}$, the local fluid velocity's magnitude, V, and the fluid density, ρ.

3.1.1 Rectangular Cuboid

This section assumes the cuboid is a simple bluff body and does not create a moment about the origin of the cuboid, therefore,

$$\mathbf{M} = 0 \tag{3.7}$$

The cuboids aerodynamic force is bluff cuboid drag in the direction of the local velocity. The drag is given by

$$\mathbf{D} = \frac{1}{2}\rho V^2 S C_D \mathbf{u}_i \tag{3.8}$$

In this equation, S is the cuboid's characteristic area and C_D is a drag coefficient. This section assumes the characteristic area is the cuboid's cross-sectional area normal to the freestream velocity. the equation describing S is

$$S = \begin{Bmatrix} l_y l_z \\ l_x l_z \\ l_x l_y \end{Bmatrix} \cdot |\mathbf{u}_i| \tag{3.9}$$

To be conservative this section assumes the drag coefficient is the drag coefficient of a cube with a face perpendicular to the flow [36] as given by

$$C_D = 1.05 \tag{3.10}$$

3.1.2 Cylinder

This section assumes the pressure distribution over the cylinder does not create a moment about the cylinders origin, therefore,

$$\boldsymbol{M} = 0 \tag{3.11}$$

The aerodynamic force acting on the cylinder depends on the freestream angle relative to the cylinder's axis. In this work, the freestream angle is known as the angle of attack, α. The angle of attack is calculated by using the definition of the dot product and representing the cylinder axis with the unit vector $(1, 0, 0)$. The angle of attack is then calculated from the x-component of the local unit velocity vector u_x as given by

$$\alpha = \cos^{-1}(-u_x) \tag{3.12}$$

The total force acting on the cylinder is split into two forces. The first force is a lift force that acts perpendicular to the incoming flow. The second force is a drag force that acts in the same direction as the incoming flow. Both forces depend on the velocity magnitude normal to the cylinder's axis as given by

$$V_n = V\sqrt{u_y^2 + u_z^2} = V\sin(\alpha) \tag{3.13}$$

The surface area, S_n, of the cylinder normal to the cylinder axis is

$$S_n = 2R_2 h \tag{3.14}$$

The lift, L, and drag, D, forces are given by

$$L = \frac{1}{2}\rho V_n^2 S_n C_L \tag{3.15}$$

$$D = \frac{1}{2}\rho V_n^2 S_n C_D \tag{3.16}$$

In this equation, the lift coefficient, C_L, and drag coefficient, C_D, depend on the angle of attack. For subcritical Reynolds numbers the coefficients C_L and C_D are related to the angle of attack according the *cross-flow principle* [36]. C_L and C_D can be approximated by

$$C_L = C_{D0}\sin^2\alpha\cos\alpha \tag{3.17}$$

$$C_D = C_{D0}\sin^3\alpha + 0.02 \tag{3.18}$$

In this equation, C_{D0} is a basic drag coefficient. This section assumes the basic drag coefficient depends on the cross-flow Reynolds number, R_E, as given by

$$R_E = \frac{2\rho V_n R_2}{\mu} \tag{3.19}$$

According to many experimental and analytic cylinder aerodynamic studies, the basic drag coefficient is approximated using the piecewise function [36, 37] for a range of Reynolds numbers as given by

$$C_{D0} = \begin{cases} 1.18 + \frac{6.8}{R_E^{0.89}} + \frac{1.96}{R_E^{1/2}} - \frac{0.0004R_E}{1+3.64\text{E-}7R_E^2}, & 0 < R_E \leq 330,000 \\ 3.78\text{E-}11R_E^2 - 3.56\text{E-}5R_E + 8.7634, & 330,000 < R_E \leq 460,000 \\ -5.0\text{E-}15R_E^2 + 7.0\text{E-}08R_E + 0.346, & 460,000 < R_E \leq 10,000,000 \\ 0.55, & R_E > 10,000,000 \end{cases} \tag{3.20}$$

When $R_E < 0.01$ the code assigns a value, $C_{D0}, = 430.0$ to avoid a singularity at zero velocity. After computing the lift and drag force the total force vector is given by

$$\boldsymbol{F} = D\boldsymbol{u}_i + L\boldsymbol{u}_L \tag{3.21}$$

where

$$\boldsymbol{u}_L = \frac{\boldsymbol{v}_L}{|\boldsymbol{v}_L|} \tag{3.22}$$

$$\boldsymbol{v}_L = \boldsymbol{u}_i \times \left(\begin{Bmatrix} 1 \\ 0 \\ 0 \end{Bmatrix} \times \boldsymbol{u}_i \right) \tag{3.23}$$

3.1.3 Sphere

A flow over a sphere does not induce a moment about the origin of the sphere, therefore,

$$\boldsymbol{M} = 0 \tag{3.24}$$

The pseudo aerodynamic force acting on a sphere is equal to the sphere's drag acting in the same direction as the local velocity. The force acting on the sphere is given by

$$\boldsymbol{D} = \frac{1}{2}\rho V^2 \pi R_2^2 C_D \boldsymbol{u}_i \tag{3.25}$$

In this equation, R_2 is the sphere's outer radius, and C_D is the sphere's drag coefficient which is dependant on the Reynolds number as given by

$$R_E = \frac{2\rho V R_2}{\mu} \tag{3.26}$$

The drag coefficient is estimated from a piecewise function that has been fitted experimental data on smooth spheres [36, 37]

$$C_D = \begin{cases} \frac{24}{R_E} + \frac{6}{1+\sqrt{R_E}} + 0.4, & 0 < R_E \leq 450,000 \\ 1.0\text{E+}29 R_E^{-5.211}, & 450,000 < R_E \leq 560,000 \\ -2.0\text{E-}23 R_E^3 - 1.0\text{E-}16 R_E^2 + 9.0\text{E-}09 R_E + 0.069, & 560,000 < R_E \leq 14,000,000 \\ 0.12, & R_E > 14,000,000 \end{cases} \tag{3.27}$$

For $R_E < 0.01$, the code uses the drag coefficient $C_D = 2405.0$ to avoid a singularity at zero velocity.

3.1.4 Wing

Geometric Considerations

The wing's aerodynamics are approximated using the wing's geometric properties from the mass and inertia section along with several other aerodynamic parameters. This work assumes that the known wing aerodynamic parameters are: the wing's lift slope, $C_{L,\alpha}$, zero-lift angle of attack, α_{L0}, pitching-moment slope, $C_{m,\alpha}$, pitching moment at zero angle of attack, C_{m0}, and three drag terms C_{D0}, C_{D1}, and the Oswald efficiency e. Note the three drag terms define the drag as a quadratic function of lift. Other properties that the user is allowed to specify are the mounting angle, α_0, control-surface effectiveness, ϵ_c, and control surface deflection, δ_c. Each of these properties affect the wing's aerodynamics. The wing's mounting angle is assumed to be negligible when approximating wing's mass and inertia relative to the aircraft. This assumption is made since the effects of mounting angle are small for most aircraft. The only problem with making this assumption is that the inertia approximations will only be valid for wings with small mounting angles.

The wing's planform area is calculated with the assumption that the wing tapers linearly as given by

$$S_w = b\bar{c} \tag{3.28}$$

where the mean chord [38], $\bar{c}$, is

$$\bar{c} = \frac{c_r + c_t}{2} \tag{3.29}$$

The aspect ratio [38], R_A, is defined as

$$R_A = \frac{b^2}{S_w} = \frac{b}{\bar{c}} \tag{3.30}$$

Dominant Forces and Moments

The distributed aerodynamic loading along the wing can be represented by a force and moment vector acting at the wing's aerodynamic center. The wings aerodynamic center location depends on several factors including wing's twist and sweep. This work assumes the wing's aerodynamic center is located at the quarter-chord of the wing centroidal chord. From the geometric definition of the wing, the wing's centroidal chord is at a spanwise location defined by

$$\hat{y} = \delta \frac{b}{3} \frac{(c_r + 2c_t)}{(c_r + c_t)} \tag{3.31}$$

In this equation the value for δ is defined to be the same as in Eq. (2.27). The corresponding quarter-chord coordinate to this spanwise location is given by

$$\hat{x} = -\delta \hat{y} \tan \mathit{\Lambda} \tag{3.32}$$

From these equations the wing's aerodynamic center location in the wing coordinate system is given by

$$\begin{Bmatrix} \hat{x} \\ \hat{y} \\ \hat{z} \end{Bmatrix} = \begin{Bmatrix} -\delta \hat{y} \tan \Lambda \\ \delta \frac{b}{3} \frac{(c_r + 2c_t)}{(c_r + c_t)} \\ 0 \end{Bmatrix} \tag{3.33}$$

The wing's arodynamic center location relative to the aircraft's coordinate system is

$$\mathbf{r}_{ac} = \mathbf{r}_o + [\mathbf{R}] \begin{Bmatrix} \hat{x} \\ \hat{y} \\ \hat{z} \end{Bmatrix} \tag{3.34}$$

In this equation, $\mathbf{r}_o$ is the vector connecting the aircraft's origin to the wing's origin and $[\mathbf{R}]$ is the component's rotation matrix calculated using Eq. (2.72).

The fluid velocity relative to the wing at the aerodynamic center location depends on the freestream velocity, $\mathbf{V}_\infty$, the aircraft's rotation rate about the aircraft's center of gravity, ω, and the wing's aerodynamic center location measured relative to the aircraft's center of gravity. This velocity vector is calculated using Eq. (3.2) with the term $\mathbf{P}_i$ defined as

$$\mathbf{P}_i \equiv \mathbf{r}_{ac} - \mathbf{r}_{\text{aircraft}} \tag{3.35}$$

This work assumes the dominant forces and moments acting on the wing are only a function of the wing's longitudinal velocity component measured relative to the wing. Therefore, this work assumes that the only velocity components affecting the forces and moments acting on the wing, are the velocity components passing through the wing's aerodynamic center in the $x - z$ plane as shown by

$$\boldsymbol{V}_w = \begin{Bmatrix} V_x \\ 0 \\ V_z \end{Bmatrix} \tag{3.36}$$

The local velocity vector of interest can also be expressed as

$$\mathbf{V}_w = V_w \mathbf{u}_w \tag{3.37}$$

In this equation V_w is the local velocity's magnitude

$$V_w = \sqrt{V_x^2 + V_z^2} \tag{3.38}$$

and $\mathbf{u}_w$ is a unit vector pointing in the same direction as local velocity vector of interest

$$\mathbf{u}_w \equiv \frac{\mathbf{V}_w}{V_w} \tag{3.39}$$

This section assumes the resulting forces and moments are only longitudinal. Therefore, this section assumes the lift, drag, and pitching moment all act in an $x - z$ plane located at the wing's aerodynamic center. Furthermore, this section assumes that any sideforce acting in the y-direction and any lateral moments acting about the x or z axes at the aerodynamic center location are negligible. From this, the direction is given by

$$\boldsymbol{u}_L = \boldsymbol{u}_w \times \begin{Bmatrix} 0 \\ 1 \\ 0 \end{Bmatrix} \tag{3.40}$$

In this expression the drag acts in the direction $\boldsymbol{u}_w$, and the pitching moment acts about the wings y coordinate, therefore,

$$\boldsymbol{u}_m = \begin{Bmatrix} 0 \\ 1 \\ 0 \end{Bmatrix} \tag{3.41}$$

The total force vector in the wing coordinate system is

$$\boldsymbol{F} = D\boldsymbol{u}_w + L\boldsymbol{u}_L \tag{3.42}$$

The total moment vector in the wing coordinate system is

$$\boldsymbol{M} = m\boldsymbol{u}_m \tag{3.43}$$

The lift, drag, and pitching moment [39] are given by:

$$L = \frac{1}{2}\rho V_w^2 S_w C_L \tag{3.44}$$

$$D = \frac{1}{2}\rho V_w^2 S_w C_D \tag{3.45}$$

$$m = \frac{1}{2}\rho V_w^2 S_w \bar{c} C_m \tag{3.46}$$

In this equation the lift coefficient, C_L, drag coefficient, C_D, and pitching-moment coefficient, C_m, are all dependant on the wing's angle of attack and control surface deflections. If the mounting angle affects are included, the angle of attack is calculated using the velocity vector in the wing's coordinate system as given by

$$\alpha = \alpha_0 + \tan^{-1}\left(\frac{-u_z}{-u_x}\right) \tag{3.47}$$

Aerodynamic Model Below Stall

For angles of attack below stall, the wing's lift, drag, and pitching moment are approximated from a basic aerodynamic model given by:

$$C_{L_{\text{model}}} = C_{L,\alpha}(\alpha - \alpha_{L0} + \epsilon_c \delta_c) \tag{3.48}$$

$$C_{D_{\text{model}}} = C_{D0} + C_{D1}C_{L_{\text{model}}} + \frac{C_{L_{\text{model}}}^2}{\pi e R_A} \tag{3.49}$$

$$C_{m_{\text{model}}} = C_{m0} + C_{m,\alpha}\alpha \tag{3.50}$$

Aerodynamic Model Above Stall

At high angles of attack the wing's lift, drag, and pitching moment are approximated using the flow over a flat plate at high incidence angles. An initial approximation model gives the lift [40, 41], drag, and pitching moment as:

$$C_{L_{\text{stall}}} = 2\text{sign}(\alpha)\sin^2\alpha\cos\alpha \tag{3.51}$$

$$C_{D_{\text{stall}}} = 2\sin^{(3/2)}(|\alpha|) \tag{3.52}$$

$$C_{m_{\text{stall}}} = -0.8\sin\alpha \tag{3.53}$$

These coefficients are blended with the basic aerodynamic model below stall coefficients using a sigmoid function. Beard and McLain [40] suggest the blending function

$$\sigma = \frac{1 + e^{-M(\alpha-\alpha_b)} + e^{M(\alpha+\alpha_b)}}{[1 + e^{-M(\alpha-\alpha_b)}][1 + e^{M(\alpha+\alpha_b)}]} \tag{3.54}$$

where α_b is a transition location and M is a blending rate parameter. In this equation α_b is a transition location and M is a blending rate parameter. This function produces a value near 0 within a range of $-\alpha_b < \alpha < \alpha_b$ and a value near 1.0 outside of this range. When $\alpha = \alpha_b$, the function produces a value of $\sigma = 0.5$. Typical wings start to stall at angles of attack between $\alpha = 15 - 25$ degrees have a transition location, α_b, of 20 to 30 degrees. For most wings a blending rate between 20 and 100 produces reasonable results with soft stall occurring at $M = 20$ and abrupt stall occurring at $M = 100$.

Total Force and Moment

Applying the sigmoid function gives an approximation for the lift, drag, and pitching moment coefficients over the entire range of angles of attack as:

$$C_L = (1-\sigma)C_{L_{\text{model}}} + \sigma C_{L_{\text{stall}}} \tag{3.55}$$

$$C_D = (1-\sigma)C_{D_{\text{model}}} + \sigma C_{D_{\text{stall}}} \tag{3.56}$$

$$C_m = (1-\sigma)C_{m_{\text{model}}} + \sigma C_{m_{\text{stall}}} \tag{3.57}$$

Note modifications to this simple model can be made for aircraft designed to maintain control effectiveness at high angles of attack. Using the results from Eqs. (3.48)–(3.53) and plugging them into Eqs. (3.55)–(3.57) gives the total lift, drag, and pitching moment of the wing.

3.1.5 Rotor

General Propulsive Properties

A rotating rotor produces aerodynamic forces and moments that a generally quantified in terms of propulsive properties. A rotor's advance ratio is a dimensionless property that relates the rotor's rotation rate to the rotor's forward velocity. This parameter is defined as

$$J \equiv \frac{2\pi V_\infty}{\omega d_r} \tag{3.58}$$

In this equation ω is the angular velocity in radians per second. The power that is transferred to the surrounding air is known as the *propulsive power* and is equal to the thrust multiplied by the freestream velocity as given by

$$P \equiv T V_\infty \tag{3.59}$$

The power the motor must provide to turn the rotor is known as the *brake power*, which is equal to the torque multiplied by the angular velocity as given by

$$P_b \equiv \ell\omega \tag{3.60}$$

The thrust produced by the rotor, T, is related to the dimensionless thrust coefficient, C_T, according to the relation

$$T = \rho(\omega/2\pi)^2 d_r^4 C_T \tag{3.61}$$

The brake power, P_b, is related to the dimensionless brake-power coefficient, C_{P_b}, according to the relation

$$P_b = \rho(\omega/2\pi)^3 d_r^5 C_{P_b} \tag{3.62}$$

In this equation, the term $(\omega/2\pi)$ has units of rotations per second. The ratio of the propulsive power to the brake power is known as the *propulsive efficiency* as given by

$$\eta_p \equiv \frac{P}{P_b} = \frac{TV_\infty}{\ell\omega} = \frac{C_T J}{C_{P_b}} \tag{3.63}$$

When a rotor is perfectly in line with the incoming freestream flow, the thrust is the dominant force and the brake power, also known as torque, is the dominant moment. If the rotor is not perfectly in line with the freastream flow, the rotor produces an additional force and moment relative to it's own coordinate system. The additional force, known as the *normal force* is caused by the asymmetric drag on the rotor blades. The additional moment, known as the *yawing moment*, is caused by the asymmetric lift on the rotor blades. The normal force produced by the rotor is directly proportional to the angle of attack, and related to the dimensionless normal-force coefficient, $C_{N,\alpha}$, according to the relation

$$N = \rho(\omega/2\pi)^2 d_r^4 C_{N,\alpha}\alpha \tag{3.64}$$

The yawing moment is related to the dimensionless yawing-moment coefficient, C_n, as given by

$$n = \rho(\omega/2\pi)^2 d_r^5 C_{n,\alpha}\alpha \tag{3.65}$$

Propulsive Parameter Inputs

Overall the rotor's propulsive properties depend on the rotor's geometry, number of blades, rotation rate, and the oncoming freestream velocity. The rotor geometry used in the previous section for approximating the mass and inertia properties are accurate for that purpose. To predict a spinning rotor's aerodynamic forces and moments or propulsive properties requires additional information. For example the rotor geometry described in the previous section did not account for any blade twist distribution (pitch) or any chord distribution other than a linearly tapered chord. A rotor's aerodynamics vary heavily based on these two parameters. In this section the rotor's propulsive properties will be approximated using inputs from the user.

The rotor's thrust, brake power, normal-force, and yawing-moment coefficients can be closely approximated as parabolic functions of the rotor's advance ratio as given by:

$$C_T = C_{T0} + C_{T1}J + C_{T2}J^2 \tag{3.66}$$

$$C_{P_b} = C_{P0} + C_{P1}J + C_{P2}J^2 \tag{3.67}$$

$$C_{N,\alpha} = C_{N1}J + C_{N2}J^2 + C_{N3}J^3 \tag{3.68}$$

$$C_{n,\alpha} = C_{n1}J + C_{n2}J^2 + C_{n3}J^3 \tag{3.69}$$

The parabolic functions for the normal force and yawing moment do not have a constant term because the normal force and yawing moment are zero when the freestream velocity is zero. As mentioned above these functions depend on the rotor's pitch, diameter, and airfoil properties. The rotor can be analyzed to find these coefficients using a method such as *blade element theory* but to do this the rotor's full properties must be known. For this work, these coefficients are given as inputs and will need to be obtained from other computational or experimental means.

Computational Method

The rotor aerodynamic model can be used two different ways in the code. The code is given values for the coefficients C_{T0}, C_{T1}, C_{T2}, C_{P0}, C_{P1}, C_{P2}, C_{N1}, C_{N2}, C_{N3}, C_{n1}, and C_{n2} as well as an operating airspeed and orientation of the rotor. This information can be used to find the rotor's thrust given the rotor's rotation rate or the rotor's rotation rate given the rotor's thrust.

Thrust given Rotation Rate: If the rotor rotation rate is given, the rotor's thrust is found by calculating the advance ratio from Eq. (3.58) then using the result in Eq. (3.66) to find the thrust coefficient. The dimensional thrust can then be found by using the thrust coefficient in Eq. (3.61).

Rotation Rate given Thrust: If the thrust is given, The rotation rate is found by using Eq. (3.58) in Eq. (3.61) to eliminate the $\omega/(2\pi)$ term, then using the result in Eq. (3.66). This gives a quadratic function of the advance ratio as

$$\left(C_{T2} - \frac{T}{\rho V_\infty^2 d_r^2}\right) J^2 + C_{T1} J + C_{T0} = 0 \tag{3.70}$$

This equation is then solved for the advance ratio, J, using the quadratic formula which gives

$$J = \frac{-C_{T1} \pm \sqrt{C_{T1}^2 - 4\left(C_{T2} - \frac{T}{\rho V_\infty^2 d_r^2}\right) C_{T0}}}{2\left(C_{T2} - \frac{T}{\rho V_\infty^2 d_r^2}\right)} \tag{3.71}$$

The positive solution is usually the correct solution.

In each case once the advance ratio is calculated, the solutions to Eqs. (3.67)–(3.69) are found. Then brake power, normal force, and yawing moment are found by plugging the solutions to Eqs. (3.67)–(3.69) into Eqs. (3.62), (3.64), and (3.65). The torque the rotor produces on the aircraft is then calculated by rearranging Eq. (3.60) to get

$$\ell = \frac{P_b}{\omega} \tag{3.72}$$

The angle of attack of the local flow relative to the rotor axis needed in Eqs. (3.64), and (3.65) is identical to that for the angle of attack of a cylinder given in Eq. (3.12).

The total force vector from the rotor's thrust and normal force are expressed in the rotor's coordinate system as

$$\boldsymbol{F} = \begin{Bmatrix} T \\ 0 \\ 0 \end{Bmatrix} + N\boldsymbol{u}_N \tag{3.73}$$

where

$$\boldsymbol{u}_N = \frac{\boldsymbol{v}_N}{|\boldsymbol{v}_N|} \tag{3.74}$$

$$\boldsymbol{v}_N = \begin{Bmatrix} 0 \\ u_y \\ u_z \end{Bmatrix} \tag{3.75}$$

The total moment vector from the rotor's torque and yawing moment are expressed in the rotor's coordinate system as

$$\boldsymbol{M} = \begin{Bmatrix} -\delta\ell \\ 0 \\ 0 \end{Bmatrix} - \delta n\boldsymbol{u}_N \tag{3.76}$$

In this equation, δ depends on the direction of the rotor as given by

$$\delta \equiv \begin{cases} 1, & \text{right-handed rotor} \\ -1, & \text{left-handed rotor} \end{cases} \tag{3.77}$$

Default Values

If the rotor information is not given, the following default values can be used to approximate the coefficients used in Eqs. (3.66) through (3.69). These coefficients are a function of pitch-to-diameter ratio, K_c, and were take from a propeller study performed by Phillips [42].

The coefficients required for the thrust coefficient, C_T, can be approximated as

$$C_{T0} = -0.119K_c^2 + 0.238K_c - 0.0194 \tag{3.78}$$

$$C_{T1} = 0.146K_c^2 - 0.0816K_c - 0.0612 \tag{3.79}$$

$$C_{T2} = 0.175K_c^3 - 0.496K_c^2 + 0.441K_c - 0.211 \tag{3.80}$$

The coefficients required for the brake power coefficient, C_{P_b}, can be approximated as

$$C_{P0} = -0.953K_c^4 + 2.5K_c^3 - 2.243K_c^2 + 0.885K_c - 0.115 \tag{3.81}$$

$$C_{P1} = 2.17K_c^4 - 5.55K_c^3 + 4.9K_c^2 - 1.75K_c + 0.225 \tag{3.82}$$

$$C_{P2} = -0.991K_c^4 + 2.34K_c^3 - 1.81K_c^2 + 0.541K_c - 0.132 \tag{3.83}$$

The coefficients required for the normal force coefficient, $C_{N,\alpha}$, can be approximated as

$$C_{N1} = 0.0147K_c^2 + 0.0116K_c - 0.0034 \tag{3.84}$$

$$C_{N2} = -0.0311K_c^2 + 0.0279K_c + 0.00984 \tag{3.85}$$

$$C_{N3} = 0.0171K_c^2 - 0.0139K_c + 0.0176 \tag{3.86}$$

The coefficients required for the yawing moment coefficient, $C_{n,\alpha}$, can be approximated as

$$C_{n1} = 0.0222K_c^2 - 0.0454K_c + 0.0034 \tag{3.87}$$

$$C_{n2} = -0.037K_c^2 + 0.0384K_c - 0.0065 \tag{3.88}$$

$$C_{n3} = 0.0206K_c^2 - 0.0266K_c + 0.0123 \tag{3.89}$$

Now that the mass and aerodynamic computations for each component type have been presented, the 6 degree-of-freedom equations of motion are explained in Chapter 4.

CHAPTER 4

6 DEGREE-OF-FREEDOM EQUATIONS OF MOTION AND SIMULATION

The 6 degree-of-freedom equations of motion and gust disturbance modeling techniques are presented in this chapter.

4.1 Equations of Motion

The generalized equations of motion for an aircraft using the flat-earth approximation are given by [43]:

$$\begin{Bmatrix} \dot{u} \\ \dot{v} \\ \dot{w} \end{Bmatrix} = \frac{g}{W} \begin{Bmatrix} F_{x_b} \\ F_{y_b} \\ F_{z_b} \end{Bmatrix} + g \begin{Bmatrix} -S_\theta \\ S_\phi C_\theta \\ C_\phi C_\theta \end{Bmatrix} + \begin{Bmatrix} rv - qw \\ pw - ru \\ qu - pv \end{Bmatrix} \tag{4.1}$$

$$\begin{Bmatrix} \dot{p} \\ \dot{q} \\ \dot{r} \end{Bmatrix} = \begin{bmatrix} I_{xx_b} & -I_{xy_b} & -I_{xz_b} \\ -I_{xy_b} & I_{yy_b} & -I_{yz_b} \\ -I_{xz_b} & -I_{yz_b} & I_{zz_b} \end{bmatrix}^{-1} \left[\begin{Bmatrix} M_{x_b} \\ M_{y_b} \\ M_{z_b} \end{Bmatrix} + \begin{bmatrix} 0 & -h_{z_b} & h_{y_b} \\ h_{z_b} & 0 & -h_{x_b} \\ -h_{y_b} & h_{x_b} & 0 \end{bmatrix} \begin{Bmatrix} p \\ q \\ r \end{Bmatrix} \right.$$
$$\left. + \begin{Bmatrix} (I_{yy_b} - I_{zz_b})qr + I_{yz_b}(q^2 - r^2) + I_{xz_b}pq - I_{xy_b}pr \\ (I_{zz_b} - I_{xx_b})pr + I_{xz_b}(r^2 - p^2) + I_{xy_b}qr - I_{yz_b}pq \\ (I_{xx_b} - I_{yy_b})pq + I_{xy_b}(p^2 - q^2) + I_{yz_b}pr - I_{xz_b}qr \end{Bmatrix} \right] \tag{4.2}$$

$$\begin{Bmatrix} \dot{x}_f \\ \dot{y}_f \\ \dot{z}_f \end{Bmatrix} = \begin{bmatrix} C_\theta C_\psi & S_\phi S_\theta C_\psi - C_\phi S_\psi & C_\phi S_\theta C_\psi + S_\phi S_\psi \\ C_\theta S_\psi & S_\phi S_\theta S_\psi + C_\phi C_\psi & C_\phi S_\theta S_\psi - S_\phi C_\psi \\ -S_\theta & S_\phi C_\theta & C_\phi C_\theta \end{bmatrix} \begin{Bmatrix} u \\ v \\ w \end{Bmatrix} + \begin{Bmatrix} V_{wx_f} \\ V_{wy_f} \\ V_{wz_f} \end{Bmatrix} \tag{4.3}$$

$$\begin{Bmatrix} \dot{\phi} \\ \dot{\theta} \\ \dot{\psi} \end{Bmatrix} = \begin{bmatrix} 1 & S_\phi S_\theta / C_\theta & C_\phi S_\theta / C_\theta \\ 0 & C_\phi & -S_\phi \\ 0 & S_\phi / C_\theta & C_\phi / C_\theta \end{bmatrix} \begin{Bmatrix} p \\ q \\ r \end{Bmatrix} \tag{4.4}$$

This formulation assumes the wind is moving at a constant velocity relative to the earth-fixed coordinate system. The aircraft specific portions of these equations are the body-fixed pseudo aerodynamic force vector, $\mathbf{F_b}$, and moment vector, $\mathbf{M_b}$, the weight, W, the body-fixed inertia tensor, $[\mathbf{I_b}]$, and the body-fixed gyroscopic tensor, $[h_b]$. The vectors $\mathbf{F_b}$ and $\mathbf{M_b}$ represent the aerodynamic forces and moments acting on the aircraft, including thrust. The weight, inertia, and gyroscopic tensors of the aircraft can vary over time without losing any fidelity to the simulation.

4.2 Gust Disturbance Modeling

4.2.1 Definitions

As stated in the introduction ship based UAS aircraft encounter adverse conditions while flying, including ship airwake, unsteady wind, gusts, and thermal effects. These adverse conditions cause disturbances in the UAS as it flies along a trajectory. In this work disturbances caused by adverse conditions are modeled as gusts acting on the UAS flying through constant wind at a given position and time. Modeling these gust disturbances in flight simulation can be useful in determining how an aircraft will react to these disturbances at varying magnitudes and duration's. The total gust disturbance vector is defined using three components in the earth-fixed coordinate system as given by

$$\begin{Bmatrix} V_{gx_f} \\ V_{gy_f} \\ V_{gz_f} \end{Bmatrix} = \begin{Bmatrix} \mathbf{V}_g \cdot \hat{\mathbf{i}}_{x_f} \\ \mathbf{V}_g \cdot \hat{\mathbf{i}}_{y_f} \\ \mathbf{V}_g \cdot \hat{\mathbf{i}}_{z_f} \end{Bmatrix} \tag{4.5}$$

Gust disturbance models can be prescribed in the earth-fixed coordinate system or the body fixed coordinate system. If the model is given in the earth-fixed coordinate system, the gust components $u_g, v_g, w_g, p_g, q_g, andr_g$ can be obtained from the general vector rotation from earth-fixed coordinates to body-fixed coordinates. If the aircraft orientation is known in terms of Euler angles, the gust model is

$$\begin{Bmatrix} V_{x_b} \\ V_{y_b} \\ V_{z_b} \end{Bmatrix} = \begin{bmatrix} C_\theta C_\psi & C_\theta S_\psi & -S_\theta \\ S_\phi S_\theta C_\psi - C_\phi S_\psi & S_\phi S_\theta S_\psi + C_\phi C_\psi & S_\phi C_\theta \\ C_\phi S_\theta C_\psi + S_\phi S_\psi & C_\phi S_\theta S_\psi - S_\phi C_\psi & C_\phi C_\theta \end{bmatrix} \begin{Bmatrix} V_{x_f} \\ V_{y_f} \\ V_{z_f} \end{Bmatrix} \tag{4.6}$$

If the aircraft orientation is known in terms of the quaternion formulation, the gust model is

$$\begin{Bmatrix} V_{x_b} \\ V_{y_b} \\ V_{z_b} \end{Bmatrix} = \begin{bmatrix} e_x^2 + e_0^2 - e_y^2 - e_z^2 & 2(e_x e_y + e_z e_0) & 2(e_x e_z - e_y e_0) \\ 2(e_x e_y - e_z e_0) & e_y^2 + e_0^2 - e_x^2 - e_z^2 & 2(e_y e_z + e_x e_0) \\ 2(e_x e_z + e_y e_0) & 2(e_y e_z - e_x e_0) & e_z^2 + e_0^2 - e_x^2 - e_y^2 \end{bmatrix} \begin{Bmatrix} V_{x_f} \\ V_{y_f} \\ V_{z_f} \end{Bmatrix} \tag{4.7}$$

The aircraft orientation expressed using quaternion algebra is

$$\begin{Bmatrix} V_{x_b} \\ V_{y_b} \\ V_{z_b} \end{Bmatrix} = \begin{Bmatrix} e_0 \\ -e_x \\ -e_y \\ -e_z \end{Bmatrix} \otimes \left(\begin{Bmatrix} 0 \\ V_{x_f} \\ V_{y_f} \\ V_{z_f} \end{Bmatrix} \otimes \begin{Bmatrix} e_0 \\ e_x \\ e_y \\ e_z \end{Bmatrix} \right) \tag{4.8}$$

For details on quaternion algebra see Chapter 11 of Mechanics of Flight by Phillips [44].

4.2.2 Atmospheric Database

As stated in the introduction, an atmospheric database of airwake wind-gust velocity data is used to simulate the adverse conditions surrounding a generic destroyer model. The atmospheric database was generated by the U.S. Navy Naval Air Systems Command(NAVAIR) using a von Karman atmospheric turbulence model and computational fluid dynamics(CFD). The generic destroyer model shown in Fig. 4.1 and Fig. 4.2 was created by the Canadian National Research Council.

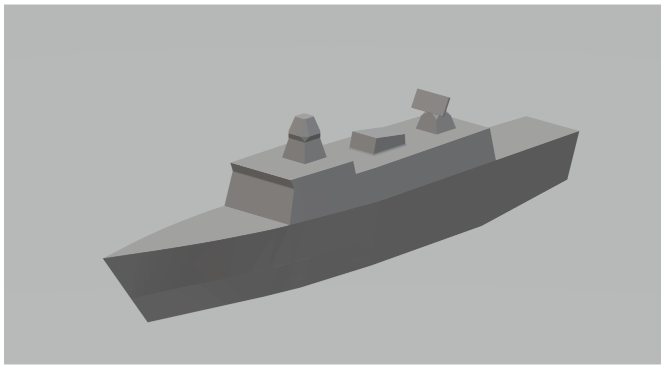

Fig. 4.1: Generic destroyer model front view.

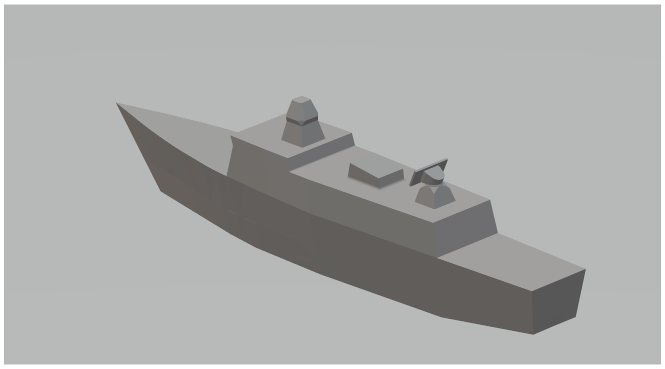

Fig. 4.2: Generic destroyer model aft view.

The atmospheric database describes the airwake wind velocity as a function of time in the x, y, and z directions over grid coordinates evenly distributed at 1 meter junctions within a 90.0 m x 67.0 m x 33.0 m rectangular volume. The rectangular volume is illustrated by the red rectangular box shown in Fig. 4.3.

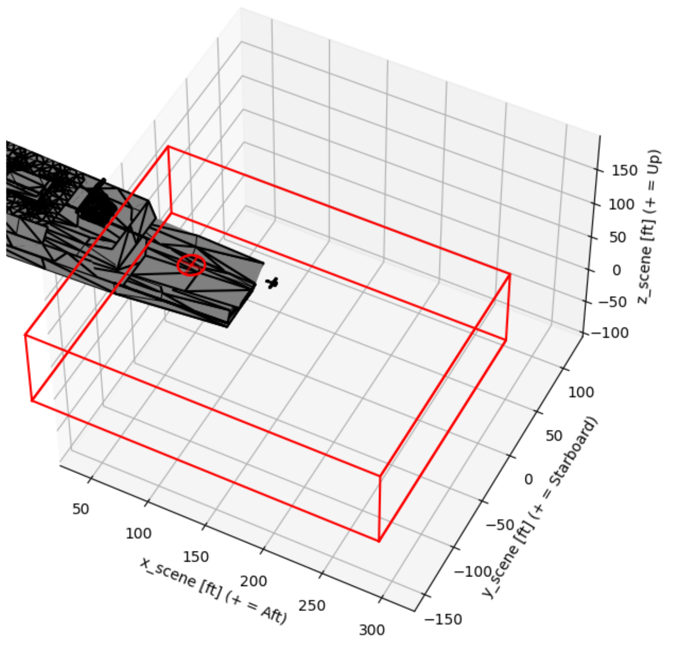

Fig. 4.3: Generic destroyer model simulation aft view with airwake box.

As seen in Fig. 4.3 and Fig. 4.4, the airwake data is not equally centered on the generic destroyer. The airwake data's origin is centered on the rear deck of the generic destroyer but the airwake data on the port side of the ship is larger than the starboard side. The measurements of the airwake box are from 0.0 m to 90.0 m in the x direction, -50.0 m to 17.0 m in the y direction and -7.0 m to 24.0 m in the z direction as shown in Fig. 4.4. The airwake data was generated in this manner because the generic destroyer is symmetric. Therefore, a solution produced while flying on one side of the destroyer would be nearly the same as if it was produced using the opposite side of the generic destroyer. For reference the landing pad for the 128 V-BAT is located at the position $y = z = 0.0$ m and $x = 15$ m (49.2 ft) as shown in 4.4.

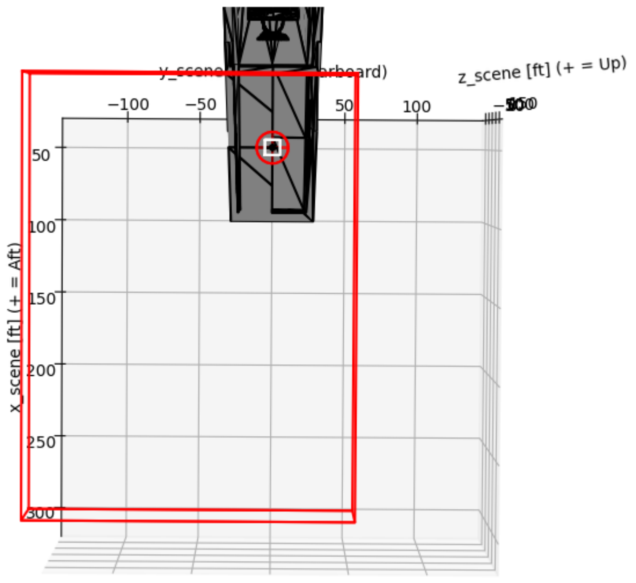

Fig. 4.4: Generic destroyer model simulation top view with airwake box.

To use the airwake wind-gust database data in simulation, SAASHA uploads the data and stores it in memory. Then SAASHA calculates the average wind velocity in the x, y, and z direction across the full database. The average velocity in the x, y, and z directions represent the V_{wx_f}, V_{wy_f}, and $V_w z_f$ terms from Eq. 4.3. Next SAASHA subtracts the average x, y, and z wind values from each location in the database. The remainder at each position represent gust disturbances or fluctuations in the flow. SAASHA then uses this data set of fluctuations when calculating the F_{x_b}, F_{y_b}, F_{z_b} terms in Eq. 4.1 and the M_{x_b}, M_{y_b}, M_{z_b} terms in Eq. 4.2.

4.3 Prescribed Trajectory Forces and Moments

For a prescribed trajectory the aircraft's orientation and Cartesian coordinates are known as a function of time. Therefore, the Aircraft's Euler angles, ϕ, θ, ψ, flat-earth coordinates x_f, y_f, z_f, and first derivatives of the orientation and coordinates with respect to time, $\dot{\phi}$, $\dot{\theta}$, $\dot{\psi}$, $(\dot{x_f}, \dot{y_f}, \dot{z_f})$ are known. This work assumes the gust velocity in earth-fixed coordinates, $(V_{gx_f}, V_{gy_f}, V_{gz_f})$, and the gust velocity's first derivative with respect to time, $(\dot{V}_{gx_f}, \dot{V}_{gy_f}, \dot{V}_{gz_f})$ are known. The aircraft's angular rotation rates are then found by rearranging Eq. (4) which gives

$$\begin{Bmatrix} p \\ q \\ r \end{Bmatrix} = \begin{bmatrix} 1 & S_\phi S_\theta / C_\theta & C_\phi S_\theta / C_\theta \\ 0 & C_\phi & -S_\phi \\ 0 & S_\phi / C_\theta & C_\phi / C_\theta \end{bmatrix}^{-1} \begin{Bmatrix} \dot{\phi} \\ \dot{\theta} \\ \dot{\psi} \end{Bmatrix} = \begin{bmatrix} 1 & 0 & -S_\theta \\ 0 & C_\phi & S_\phi C_\theta \\ 0 & -S_\phi & C_\phi C_\theta \end{bmatrix} \begin{Bmatrix} \dot{\phi} \\ \dot{\theta} \\ \dot{\psi} \end{Bmatrix} \tag{4.9}$$

The body fixed velocity components are then found by rearranging Eq. (3) which gives:

$$\begin{Bmatrix} u \\ v \\ w \end{Bmatrix} = \begin{bmatrix} C_\theta C_\psi & S_\phi S_\theta C_\psi - C_\phi S_\psi & C_\phi S_\theta C_\psi + S_\phi S_\psi \\ C_\theta S_\psi & S_\phi S_\theta S_\psi + C_\phi C_\psi & C_\phi S_\theta S_\psi - S_\phi C_\psi \\ -S_\theta & S_\phi C_\theta & C_\phi C_\theta \end{bmatrix}^{-1} \begin{Bmatrix} \dot{x}_f - V_{wx_f} \\ \dot{y}_f - V_{wy_f} \\ \dot{z}_f - V_{wz_f} \end{Bmatrix} \tag{4.10}$$

$$\begin{Bmatrix} u \\ v \\ w \end{Bmatrix} = \begin{bmatrix} C_\theta C_\psi & C_\theta S_\psi & -S_\theta \\ S_\phi S_\theta C_\psi - C_\phi S_\psi & S_\phi S_\theta S_\psi + C_\phi C_\psi & S_\phi C_\theta \\ C_\phi S_\theta C_\psi + S_\phi S_\psi & C_\phi S_\theta S_\psi - S_\phi C_\psi & C_\phi C_\theta \end{bmatrix} \begin{Bmatrix} \dot{x}_f - V_{wx_f} \\ \dot{y}_f - V_{wy_f} \\ \dot{z}_f - V_{wz_f} \end{Bmatrix} \tag{4.11}$$

Knowing the body-fixed velocities and angular rates, The pseudo aerodynamic forces and moments as a function of time are found by rearranging Eqs. 4.1 and 4.2. This gives:

$$\begin{Bmatrix} F_{x_b} \\ F_{y_b} \\ F_{z_b} \end{Bmatrix} = \frac{W}{g} \begin{Bmatrix} \dot{u} - rv + qw \\ \dot{v} - pw + ru \\ \dot{w} - qu + pv \end{Bmatrix} - W \begin{Bmatrix} -S_\theta \\ S_\phi C_\theta \\ C_\phi C_\theta \end{Bmatrix} \tag{4.12}$$

$$\begin{Bmatrix} M_{x_b} \\ M_{y_b} \\ M_{z_b} \end{Bmatrix} = \begin{bmatrix} I_{xx_b} & -I_{xy_b} & -I_{xz_b} \\ -I_{xy_b} & I_{yy_b} & -I_{yz_b} \\ -I_{xz_b} & -I_{yz_b} & I_{zz_b} \end{bmatrix} \begin{Bmatrix} \dot{p} \\ \dot{q} \\ \dot{r} \end{Bmatrix} - \begin{bmatrix} 0 & -h_{z_b} & h_{y_b} \\ h_{z_b} & 0 & -h_{x_b} \\ -h_{y_b} & h_{x_b} & 0 \end{bmatrix} \begin{Bmatrix} p \\ q \\ r \end{Bmatrix}$$
$$- \begin{Bmatrix} (I_{yy_b} - I_{zz_b})qr + I_{yz_b}(q^2 - r^2) + I_{xz_b}pq - I_{xy_b}pr \\ (I_{zz_b} - I_{xx_b})pr + I_{xz_b}(r^2 - p^2) + I_{xy_b}qr - I_{yz_b}pq \\ (I_{xx_b} - I_{yy_b})pq + I_{xy_b}(p^2 - q^2) + I_{yz_b}pr - I_{xz_b}qr \end{Bmatrix} \tag{4.13}$$

Recall that the pseudo aerodynamic forces and moments include the thrust force. Assuming there is adequate degrees of freedom to control the aircraft's forces an moments, control combinations on board the aircraft could be used to satisfy Eqs. (4.12) and (4.13) at each time step. satisfying Eqs. (4.12) and (4.13) requires total control of the aircraft's six degrees of freedom. Since this level of control is not available on most aircraft, it can be helpful to reduce the degrees of freedom required. This is done by constraining the aircraft's path. This method is known as *coordinated maneuvering.*

The V-BAT airframe model, prescribed landing trajectory, and simulation data sweep are presented in Chapter 5.

CHAPTER 5

OVERVIEW OF THE V-BAT AIRFRAME AND LANDING TRAJECTORY

The V-BAT model design, prescribed landing trajectory, and simulation data sweep are presented in this chapter.

5.1 Setup

5.1.1 V-BAT Model Design

Fig. 5.1 and 5.2 show an approximation of the Martin UAV 128 V-BAT. The V-BAT model is comprised of six components consisting of a sphere representing the nose of the aircraft, a long solid cylinder representing the fuselage, a hollow cylinder representing the base of the aircraft, a main wing, a rotor, and a rectangular cuboid representing the mass and inertia properties of the motor and accompanying hardware located in the lower region of the fuselage.

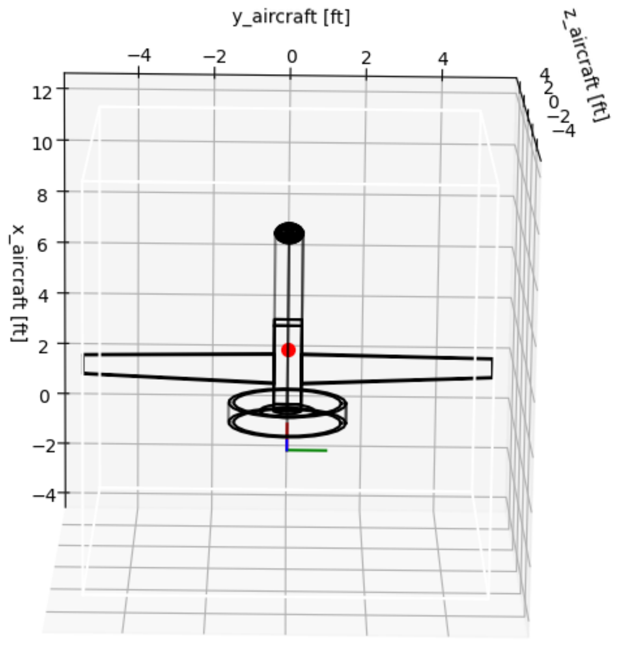

Fig. 5.1: V-BAT model in hover position.

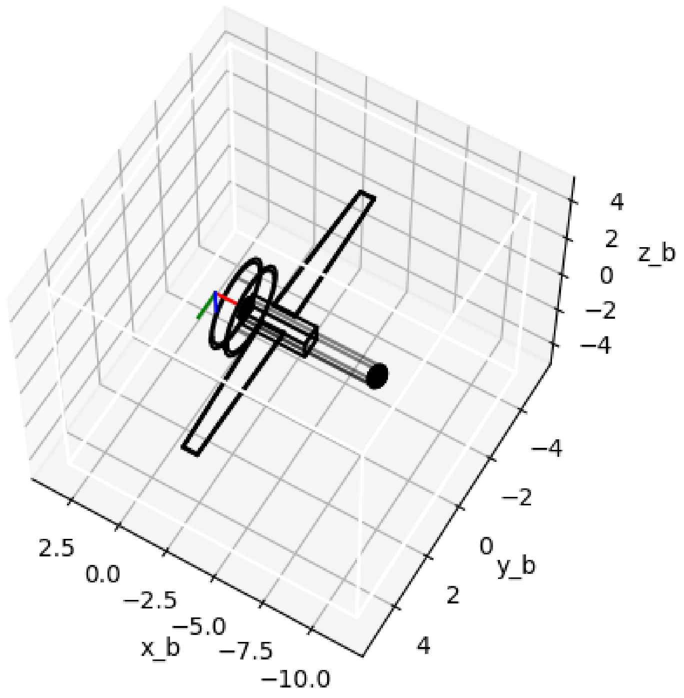

Fig. 5.2: V-BAT model in horizontal flight position.

The geometric dimensions and weight of each of the V-BAT model's components were estimated from online sources [45]. The other dimensions were estimated from photographs of the V-BAT. The following tables show the geometric dimensions and mass properties of each component.

The geometric and mass properties for the nose are shown in Table 5.1.

Table 5.1: V-BAT nose geometric properties.

Property	Nose
Radius[ft]	0.125
Weight[lbf]	5.00
Location[ft]	[8.20, 0.00, 0.00]

The geometric and mass properties for the base cylinder and fuselage are shown in Table 5.2.

Table 5.2: V-BAT cylindrical fuselage geometric properties.

Property	Base Cylinder	Fuselage
Inner Radius[ft]	1.375	0.0
Outer Radius[ft]	1.50	0.35
Length[ft]	0.75	6.70
Weight[lbf]	4.00	28.0
Location[ft]	[4.87, 0.00, 0.00]	[3.24, 0.00, 0.00]

The geometric and mass properties for the right main wing are shown in Table 5.3. The properties for the left main wing are identical except the location is at -0.35 in the y direction.

Table 5.3: V-BAT wing geometric properties.

Property	Main Wing
Span [ft]	4.85
Chord [ft]	[1.105, 0.71]
Thickness [%]	[12, 12]
Sweep [deg]	0.0
Dihedral [deg]	0.0
Airfoil	NACA 2412
Weight[lbf]	2.00
Location[ft]	[3.50, 0.35, -0.35]

The geometric and mass properties for the rotor are shown in Table 5.4.

Table 5.4: V-BAT rotor geometric properties.

Property	Rotor
Span [ft]	0.225
Chord [ft]	[0.20, 0.20]
Thickness [%]	[12, 12]
Sweep [deg]	0.0
Dihedral [deg]	0.0
Hub Radius[ft]	0.125
Hub Height[ft]	0.20
Airfoil	NACA 2412
Weight[lbf]	3.00
Location[ft]	[1.50, 0.00, 0.00]

The geometric and mass properties for the motor and hardware in the lower half of the fuselage are shown in Table 5.5.

Table 5.5: V-BAT lower fuselage geometric properties.

Property	Lower Fuselage
Height[ft]	0.70
Width[ft]	0.70
Length[ft]	3.22
Weight[lbf]	60.0
Location[ft]	[3.24, 0.00, 0.00]

To simulate the V-BAT model, the geometric and mass and inertia properties are compiled into a JSON file. The V-BAT JSON file is loaded into SAASHA and the data is used to calculate the V-BAT model's mass, inertia, and aerodynamic properties. Once these properties have been calculated, the V-BAT can be used in the 6 DoF simulation to simulate flight through specified trajectories which will be explained in the next section.

5.1.2 Landing Trajectory Overview

When the 128 V-BAT is in the air, it flies in a horizontal orientation like a conventional fixed wing aircraft but when the V-BAT takes off or lands, it hovers in a vertical orientation then transitions to or from the horizontal position. The 128 V-BAT uses thrust vectoring to accomplish these maneuvers.

A typical landing sequence for the 128 V-BAT is shown in Fig. 5.3. When landing, the V-BAT model flies horizontally toward the landing pad. As the V-BAT gets closer to the pad it decreases velocity and starts to pitch up. As the V-BAT pitches upward it gains altitude then hovers in place before descending vertically to the landing pad.

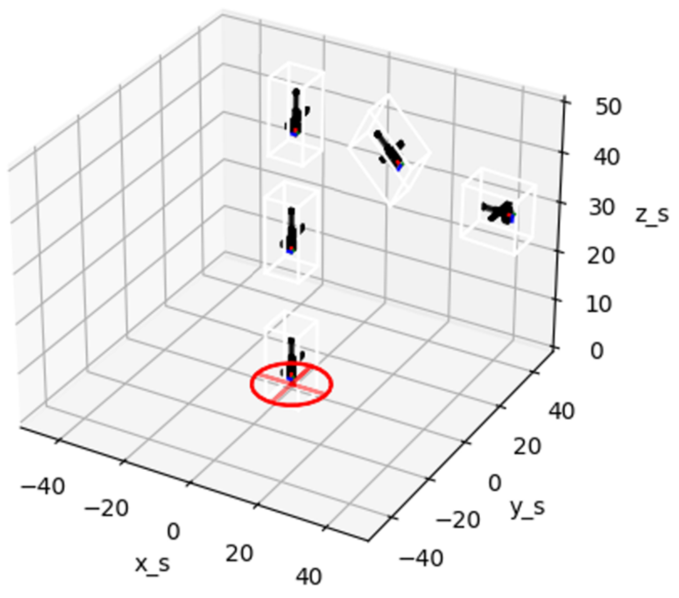

Fig. 5.3: V-BAT landing trajectory simulation.

Fig. 5.4 shows the V-BAT model landing trajectory altitude versus range with a time step of 0.1 seconds. The total landing sequence lasts for 10 seconds. During this landing sequence the V-BAT starts at an altitude of 46 ft above the ship's deck. Then as the V-BAT approaches the landing zone, it pitches up until it reaches an altitude of 52 ft then descends to the ship's deck.

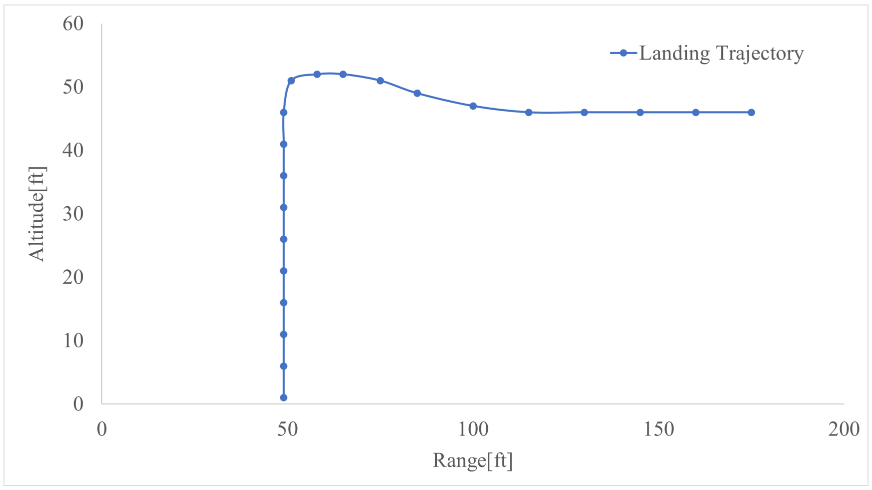

Fig. 5.4: V-BAT landing trajectory plot of altitude versus range.

To simulate the landing trajectory shown in Fig. 5.4, the SAASHA program requires a series of data points along the aircraft's trajectory. Specifically, the SAASHA program requires a CSV file with the aircraft's x, y, and z locations, bank, elevation, and heading angles, and the time step at which the aircraft is in the specified location and orientation. Table 5.6 shows the trajectory data depicted by the V-BAT model in Fig. 5.3 and the landing trajectory in Fig. 5.4.

Table 5.6: UAS trajectory data used to simulate the 128 V-BAT model through a prescribed landing trajectory.

Time[s]	x-position[ft]	y-position[ft]	z-position[ft]	ϕ[deg]	θ[deg]	ψ[deg]
0.0	175.0	0.0	46.0	0.0	0.0	0.0
0.5	160.0	0.0	46.0	0.0	0.0	0.0
1.0	145.0	0.0	46.0	0.0	0.0	0.0
1.5	130.0	0.0	46.0	0.0	0.0	0.0
2.0	115.0	0.0	46.0	0.0	0.0	0.0
2.5	100.0	0.0	47.0	0.0	15.0	0.0
3.0	85.0	0.0	49.0	0.0	30.0	0.0
3.5	75.0	0.0	51.0	0.0	45.0	0.0
4.0	65.0	0.0	52.0	0.0	60.0	0.0
4.5	58.0	0.0	52.0	0.0	75.0	0.0
5.0	51.0	0.0	51.0	0.0	90.0	0.0
5.5	49.0	0.0	46.0	0.0	90.0	0.0
6.0	49.0	0.0	41.0	0.0	90.0	0.0
6.5	49.0	0.0	36.0	0.0	90.0	0.0
7.0	49.0	0.0	31.0	0.0	90.0	0.0
7.5	49.0	0.0	26.0	0.0	90.0	0.0
8.0	49.0	0.0	21.0	0.0	90.0	0.0
8.5	49.0	0.0	16.0	0.0	90.0	0.0
9.0	49.0	0.0	11.0	0.0	90.0	0.0
9.5	49.0	0.0	6.0	0.0	90.0	0.0
10.0	49.0	0.0	1.0	0.0	90.0	0.0

A simulation of this trajectory using SAASHA is discussed in the next section.

5.1.3 Prescribed Landing Trajectory SAASHA Simulation Example

Using the V-BAT model and prescribed trajectory, SAASHA simulates the forces and moments acting on the V-BAT model in x, y, and z directions at each time step. SAASHA starts by accepting several different input files including the V-BAT JSON file and the trajectory model. Once the aircraft and trajectory data has been uploaded, the program calculates the mass, inertia, and aerodynamic properties of the V-BAT then the program uses cubic interpolation to calculate the remainder of the aircraft's trajectory between each data point. The SAASHA program then simulates the aircraft's flight over the trajectory and records the aircraft's state at each timestep. The recorded state of the aircraft includes the aircraft's velocity, position, orientation, forces and moments required to complete the landing, and forces and moments the aircraft experiences during the simulation. Several of the resulting plots for this trajectory are shown.

Fig. 5.5 shows the V-BAT's landing position versus time. As seen in the figure the V-BAT flies in the x-direction toward the landing pad located at $x = 49$ ft. As the V-BAT gets closer to the landing pad it starts to slow down and pitch up. As the V-BAT slows down in the x-direction the velocity starts to increase in the z-direction. As shown in the Fig 5.5 the V-BAT's position converges toward an x position of 49 ft. The figure also shows the V-BAT's z location remains relatively constant until the V-BAT pitches up and starts to descend at the constant rate. At this point the z position decreases till the V-BAT reaches the deck.

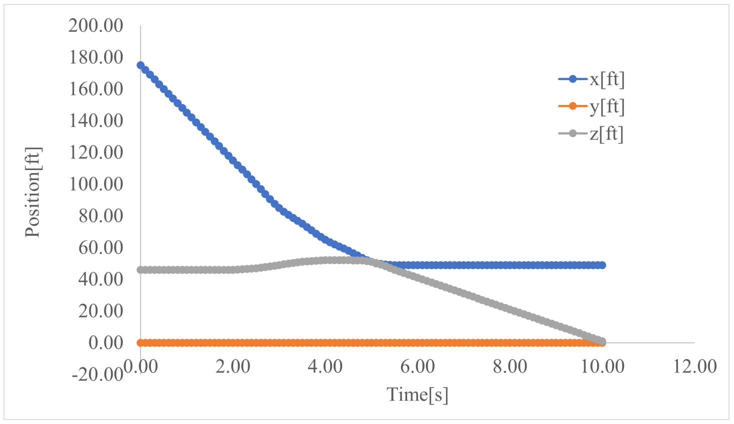

Fig. 5.5: V-BAT landing trajectory position versus time plot.

Fig. 5.6 shows the velocity of the V-BAT relative to the flow field. As shown in the figure the V-BAT flies into the wind in the x-direction. Approximately 3 seconds into later the V-BAT starts to pitch up. At this point the V-BAT's velocity relative to the wind in the x-direction decreases while the velocity in the z-direction increases due to the coordinate system rotation. Eventually the velocity reaches an equilibrium point where the velocity relative to the z-direction is approximately 59 ft/s and the velocity relative to the x-direction is about -10 ft/s. This result is reasonable considering the air comes over the front and top of the generic destroyer model then flows turbulently down to the landing pad and off the back of the ship. Most of the air in this case pushes against the belly of the of the aircraft in the z-direction but a portion of the remainder of the air pushes down on the V-BAT in the $-x$-direction.

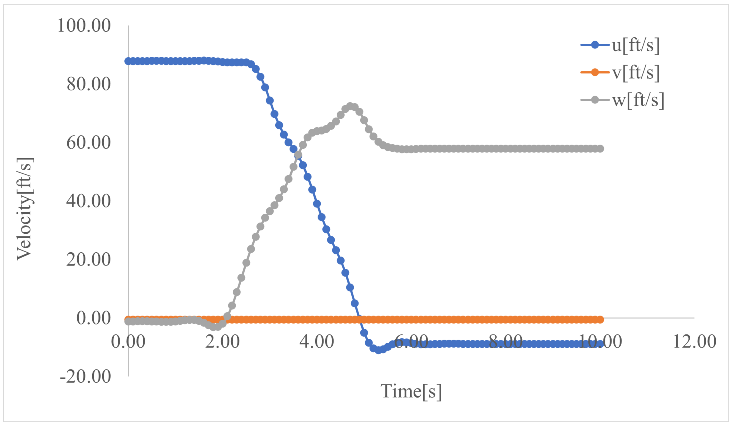

Fig. 5.6: V-BAT landing trajectory velocity versus time plot.

Fig. 5.7 shows the forces acting on the V-BAT versus time. As the V-BAT flies into the turbulent airwake, gusts of wind push on the aircraft in each direction which explains the small sudden spikes in force in the z-direction. Then as the V-BAT pitches upward, more of the V-BAT's body and main wing become exposed to the turbulent flow. This causes the force in the negative z-direction to increase very rapidly. The force in each direction then decreases and converges toward zero as the V-BAT descends into the wind break caused by the generic destroyer.

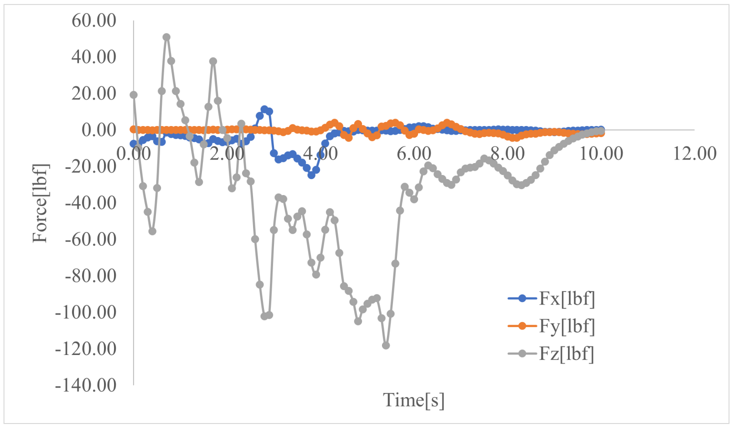

Fig. 5.7: V-BAT landing trajectory force versus time plot.

Fig. 5.8 shows the moments acting on the V-BAT versus time. As the V-BAT starts to fly throught the turbulent air the largest moment is positive about the V-BAT's x axis. This is probably due to the turbulent air coming over the top of the generic destroyer pushing down ward on the V-BAT. As the V-BAT pitches up between 2 and 3 seconds into the landing the moment about the x axis becomes a large negative number this is most likely due to the V-BAT lifting into the wind in combination with a strong gusts. As the V-BAT model lands and moves into the wind break the moments converge toward zero.

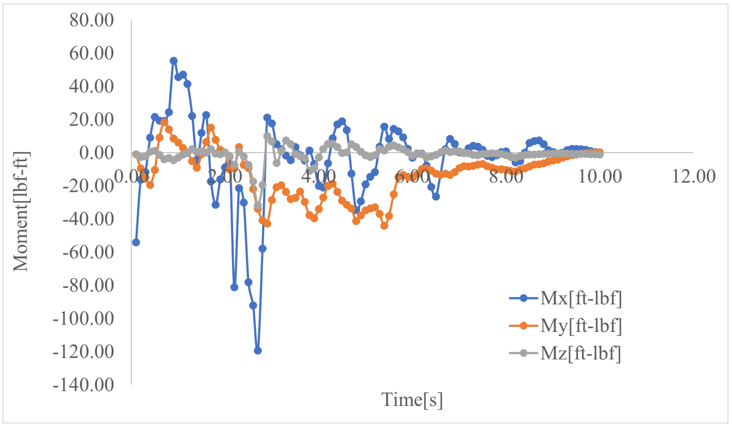

Fig. 5.8: V-BAT landing trajectory moment versus time plot.

In this example SAASHA was given a predetermined trajectory to use in the simulation but the SAASHA program is also able to generate flight paths using a starting and an ending points and interpolating the rest of the flight trajectory. For this work the later method will be used to simulate the V-BAT's landing trajectory as discussed in the next section.

5.2 Landing Trajectory Simulation

5.2.1 Landing Approach and Descent

The V-BAT's landing trajectory can be split into two independent actions, the horizontal landing pad approach, and the vertical decent to the deck. The V-BAT can approach the ships landing pad from many directions and altitudes as illustrated in Fig. 5.9 however, due to safety concerns UAS are generally not flown directly toward the front of the ship. In most cases UAS pilots approach the ship from either side or aft of the ship.

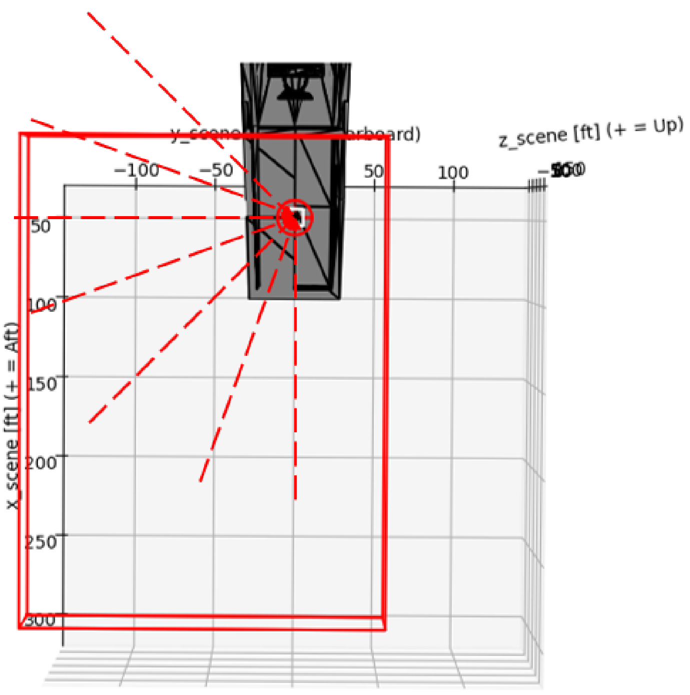

Fig. 5.9: Landing approach trajectories

As the V-BAT approaches the landing pad, the V-BAT experiences turbulence caused by the ship's airwake. As the V-BAT pitches up to start it's vertical descent, the full body and wings of the aircraft become exposed to the turbulent airwake. To minimize the forces and moments acting on the V-BAT, the UAS pilot can change the V-BAT's bank angle relative to the oncoming airwake as the V-BAT descends to the ship's landing pad. This decent is illustrated in Fig. 5.10. This work seeks to simulate the V-BAT's vertical decent to identify a series of bank angle orientations that will minimize the variation in forces and moments acting on the V-BAT model as it lands.

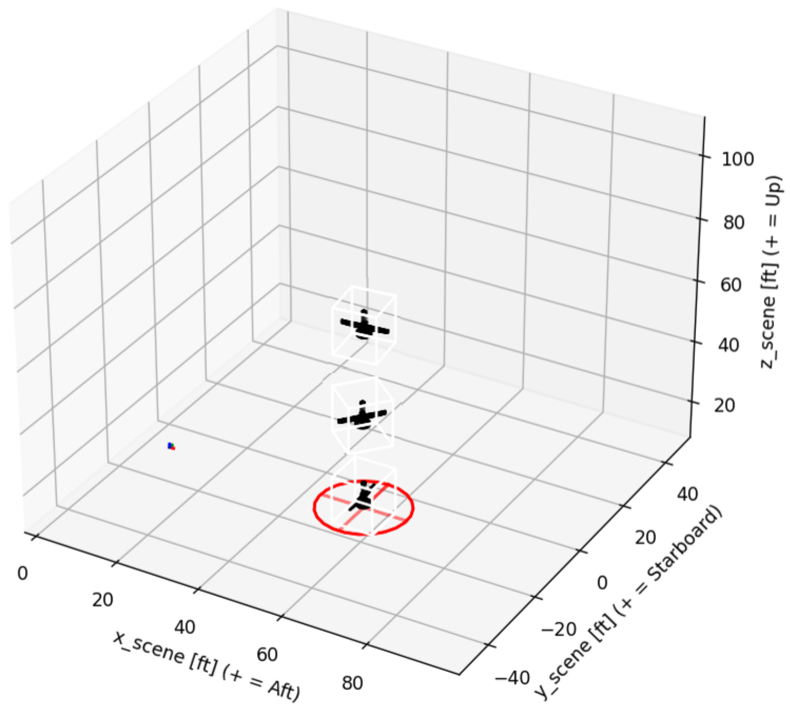

Fig. 5.10: V-BAT landing descent

5.2.2 Descent Testing

To find the best bank angle orientation for the V-BAT at any point in time, the V-BAT is simulated in place at a prescribed altitude and bank angle for 59 seconds, from 0.5 to 59.5 seconds in the gust database, with a time step of 0.5 seconds. After the simulation is complete, the V-BAT's bank angle is incremented and the simulation is run again. This process is repeated over a sweep of angles and altitudes. During each simulation, SAASHA calculates the forces and moments acting on the V-BAT at each altitude, bank angle orientation, and time step. SAASHA records this information as well as the V-BAT's state to a CSV file. Once the full sweep of simulations is complete, the data is analyzed to find the orientation that minimizes the variation in forces and moments acting on the V-BAT.

There are several methods to minimize the forces and moments acting on the V-BAT. One method is to compute the average and standard deviation of the total forces and moments at each altitude and orientation angle. Then plot the results on an error bar chart and contour plot to compare the results and see what angles minimize the average, and standard deviations.

Another method is to compare the forces and moments acting on the V-BAT at each time step and altitude to find the minimum force or moment, then take the orientation angle corresponding to that force or moment as the optimum angle for the time step. Once the optimum angle has been found at each time step the data can be mapped on a 2D contour plot to determine a series of orientation angles and that will minimize the force and moment variation as the V-BAT descends.

5.2.3 Minimum Average and Standard Deviation Decent

To find the bank angle orientations that minimize the average and standard deviation of the forces and moments, the force and moment maximum, minimum, average, and standard deviations are calculated at each altitude and orientation angle. Next the average and standard deviations are plotted on the x axis with the altitude on the y axis. Using this data, a set of hover ladder plots are created showing the series of optimum orientation angles that occur at the lowest average forces and moments.

The advantage of using a hover ladder plot is it shows the series of bank angle orientations that minimize the forces and moments for most of the descent trajectory, regardless of the time step. The drawback of using a hover ladder plot is there are better series of landing orientations if the time of landing is taken into account.

5.2.4 Contour Plots Overview

To create contour plots of the altitude versus time versus the minimum force, the first step is to compare the forces at each time step and altitude to determine the lowest force and associated orientation angle at each time step. Once this is known, the minimum forces, moments, force angles, and moment angles can be plotted and observed. The advantage of using a contour plot over the minimum average and standard deviation plots is the contour plot takes time into consideration when determining the best bank angle orientation to operate at during landing. That is the contour plot shows the bank angle orientation that minimizes the forces and moments at each altitude and time step.

The one drawback of the contour plot is it is specific to each time step. The best orientation angle is the best for that time step but not necessarily the best for the entire data set so each trajectory must be planned accordingly.

5.2.5 Altitude and Angle Sweep

The V-BAT is simulated through a sweep of altitudes and angles at each altitude. For this work the V-BAT starts at an altitude of 50 ft above the landing pad then descends by 1 foot increments until it reaches 1 ft above the landing pad. A landing altitude of 1 ft above the landing pad is used because the actual 128 V-BAT UAS has thin wire legs to land on which are not included in the V-BAT model.

Since the V-BAT model is nearly symmetric across the y axis, top to bottom, the forces and moments felt by the V-BAT at any bank angle will be nearly the same as if the V-BAT was rotated 180 degrees about the x axis as shown in Fig. 5.11 through Fig. 5.14. Since the forces and moments across the y axis are nearly equal, the sweep of bank angles the V-BAT passes through at each altitude start at 0 degrees, then rotate every 5 degrees until the bank angle reaches 180 degrees. Once the angle sweep is complete, the V-BAT's altitude decrements by 1 foot and the angle sweep repeats.

The data used in Fig. 5.11 through Fig. 5.14 is found using a wrapper code that simulates SAASHA through two sweeps of altitudes and bank angles as described. The two sets of data are generated using an altitude sweep from 50 ft to 1 ft with a decrement of 1 ft. At each altitude, the first set of data uses a bank angle sweep from 0 to 180 degrees with an increment of 5 degrees. Meanwhile, the second set of data uses a bank angle sweep from 0 to 360 degrees with an increment of 5 degrees. Both data sets show the lowest forces and moments and associated bank angles at each altitude and time step. The 360 degree force, moment, and bank angle, data is subtracted from the 180 degree data and the results are plotted.

Fig. 5.11 shows the scatter and contour plots of the minimum force bank angle difference between the 0 to 180 degrees data set and the 0 to 360 degrees data set. As expected, most of the bank angle difference data is clustered about 0 degrees and -180 degrees. The points where the data difference is approximately 0 degrees corresponds to the lowest force occurring within 0 and 180 degrees at that altitude and time step. The points where the data difference is approximately -180 degrees corresponds to the minimum force occurring within 0 to 360 degrees at that altitude and time step.

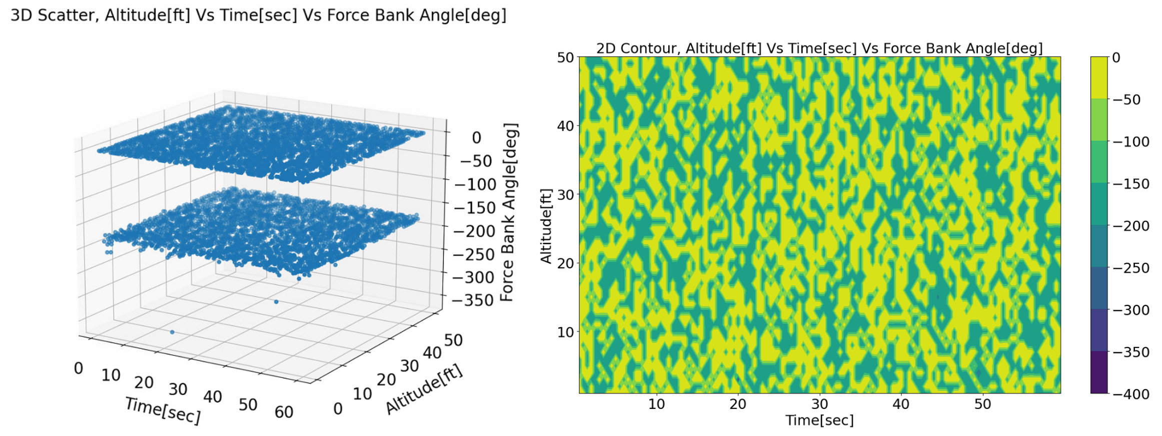

Fig. 5.11: Minimum force bank angle difference scatter(left) and contour(right) plots showing the bank angle difference centered around 0 and - 180 degrees.

Fig. 5.12 shows the scatter and contour plots of the minimum force difference between the 0 to 180 degrees data set and the 0 to 360 degrees data set. As expected, the difference in forces is clustered around 0 lbf with the greatest difference equal to 0.48 lbf.

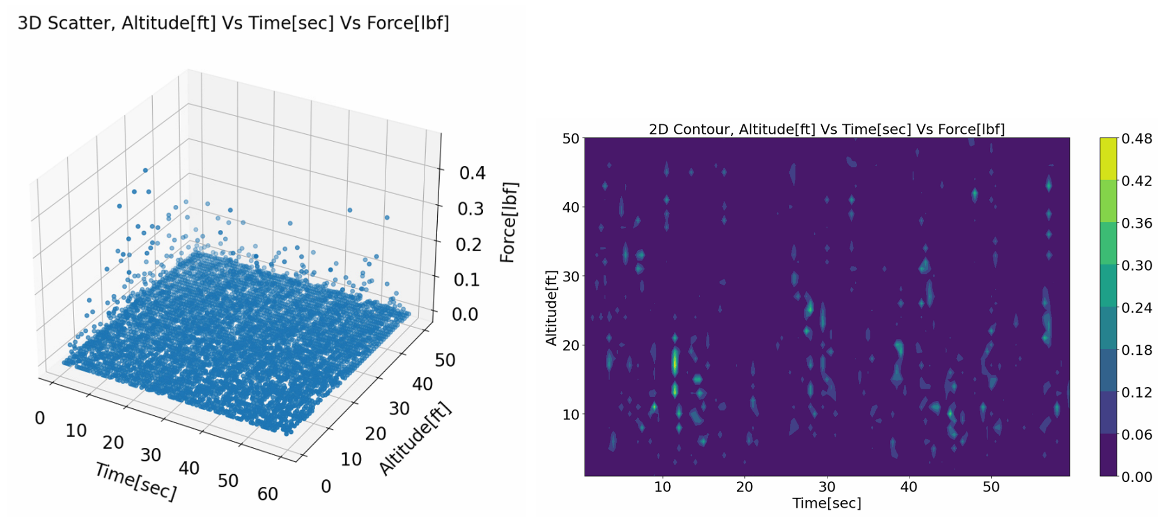

Fig. 5.12: Minimum force difference scatter(left) and contour(right) plots.

Fig. 5.13 shows the scatter and contour plots of the minimum moment bank angle difference between the 0 to 180 degrees data set and the 0 to 360 degrees data set. similar to the force bank angle difference, most of the moment bank angle difference data is clustered about 0 degrees and -180 degrees, however, there are noticeably more outliers. As with the force bank angle plots, the points where the data difference is approximately 0 degrees corresponds to the lowest moment occurring within 0 and 180 degrees at that altitude and time step. The points where the data difference is approximately -180 degrees corresponds to the minimum moment occurring within 0 to 360 degrees at that altitude and time step.

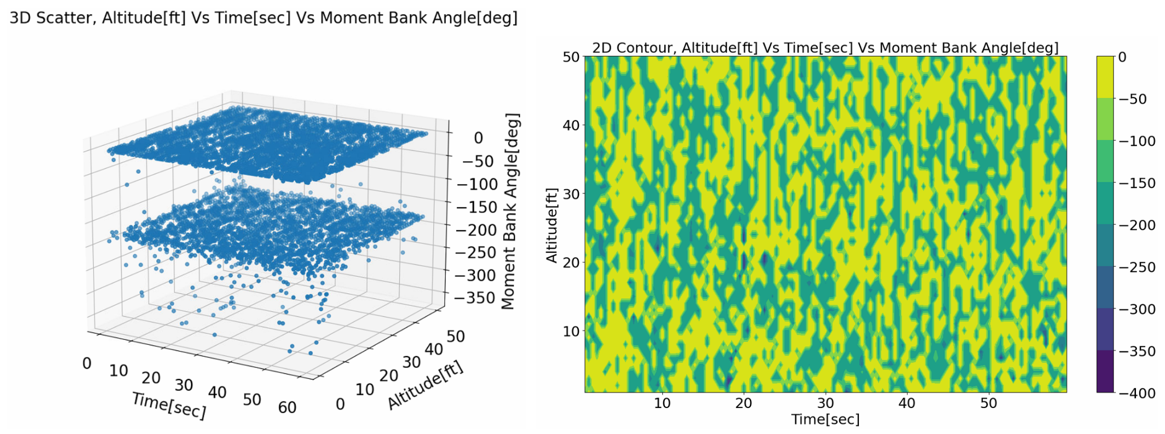

Fig. 5.13: Minimum moment bank angle difference scatter(left) and contour(right) plots showing the bank angle difference centered around 0 and - 180 degrees.

Fig. 5.14 shows the scatter and contour plots of the minimum moment difference between the 0 to 180 degrees data set and the 0 to 360 degrees data set. Similar to the force difference, the difference in moments is clustered around 0 lbf-ft with the greatest difference equal to 3.2 lbf-ft. A moment difference of 3.2 lbf-ft seems large but this value is reasonable considering the V-BAT model has a near 10 ft wing span and is almost 9 ft long.

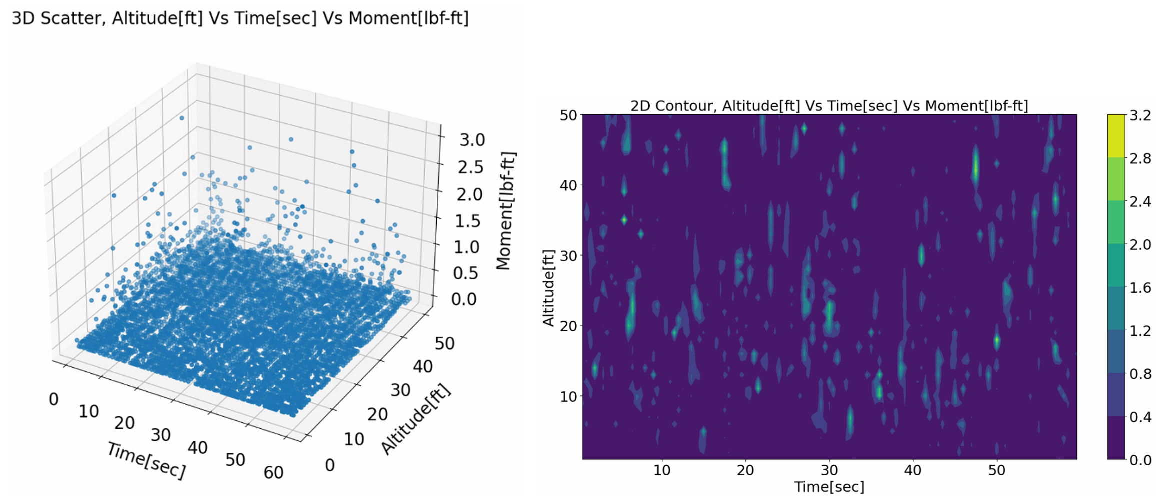

Fig. 5.14: Minimum moment difference scatter(left) and contour(right) plots.

The landing trajectory sweep results are presented in Chapter 6.

CHAPTER 6
SIMULATION RESULTS

The landing trajectory bank angle sweep results are presented in this chapter.

6.1 Minimum Average and Standard Deviation Results

Fig. 6.1 shows the altitude versus average force of the V-BAT at each orientation angle. As shown in the figure, the average forces acting on the V-BAT decrease as the altitude decreases. This is most likely due to the top of the destroyer model blocking the main stream of wind as the V-BAT lands. The left edge of the data shown in red represents the series of minimum average forces and bank angles. Following this series of angles during vertical landing is one method to minimize the variation in the average forces over the landing trajectory.

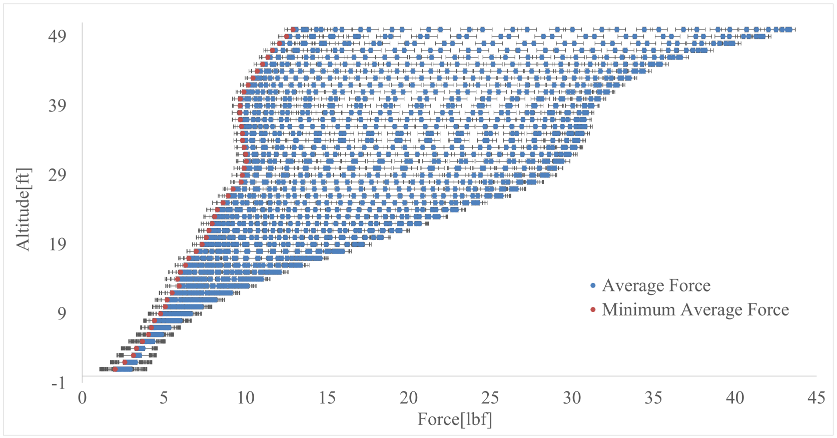

Fig. 6.1: Altitude versus minimum average force plot.

Converting Fig 6.1 to a 3D scatter plot using the bank angle as the third dimension gives Fig. 6.2. As expected, the lowest average force follows a series of bank angles clustered about a bank angle of 90 degrees where the wing is mostly parallel to the wind as denoted by the red data points.

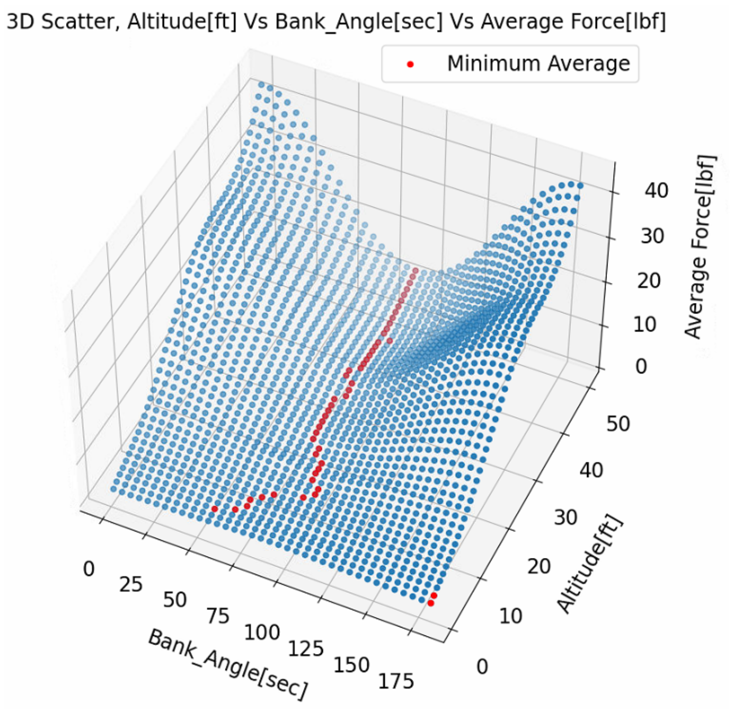

Fig. 6.2: Altitude versus bank angle versus minimum average force.

Fig. 6.3 shows the altitude versus average moment of the V-BAT at each orientation angle. Similar to the average force plot, the average moments acting on the V-BAT decrease as the altitude decreases. As before, this is most likely due to the top of the destroyer model blocking the main stream of wind as the V-BAT lands. The left edge of the data shown in red represents the series of minimum average moments and corresponding bank angles. Following this series of angles during vertical landing is one method to minimize the variation in the average moments over the landing trajectory.

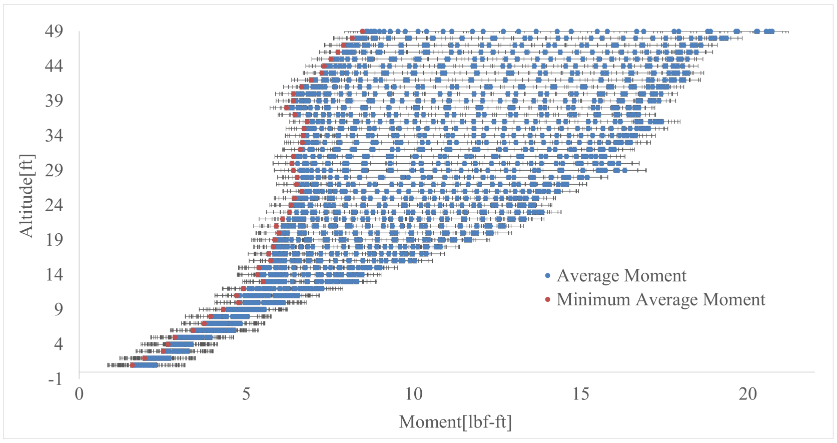

Fig. 6.3: Altitude versus minimum average moment plot.

Converting Fig. 6.3 to a 3D scatter plot using the bank angle as the third dimension gives Fig. 6.4. Similar to the average force plot, Fig 6.4 shows the bank angles corresponding to the minimum average moment are clustered about a bank angle of 90 degrees but the bank angle series seems to diverge away from 90 degrees more than the minimum average force series. Note similar plots can be generated for the force and moment standard deviations.

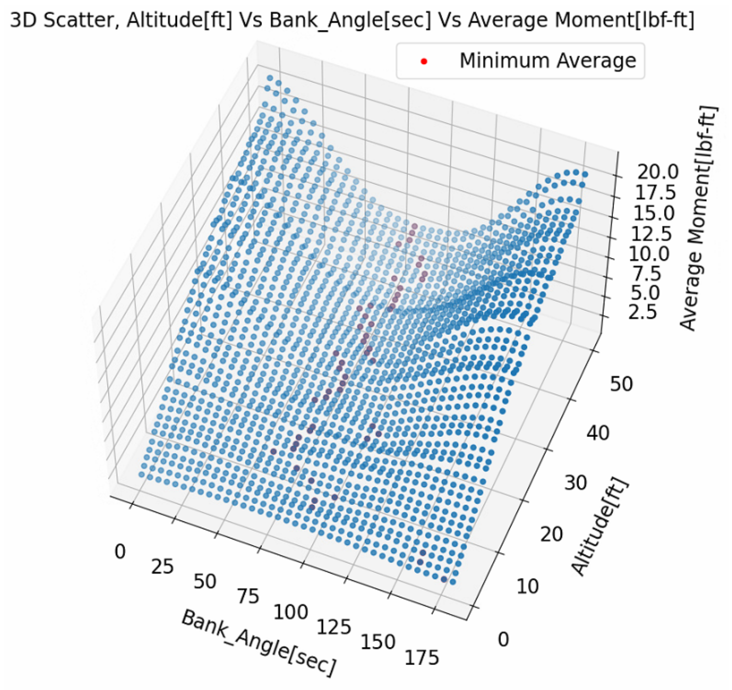

Fig. 6.4: Altitude versus bank angle versus minimum average moment.

Fig. 6.5 shows the 3D and 2D contour plots for the altitude versus angle versus average force. The two lines depicted in the plot are the minimum average force solution in red and the minimum force standard deviation solution in blue. Note the average force solution follows the lowest path across the contour plot while the standard deviation solution deviates from the lowest path from time to time.

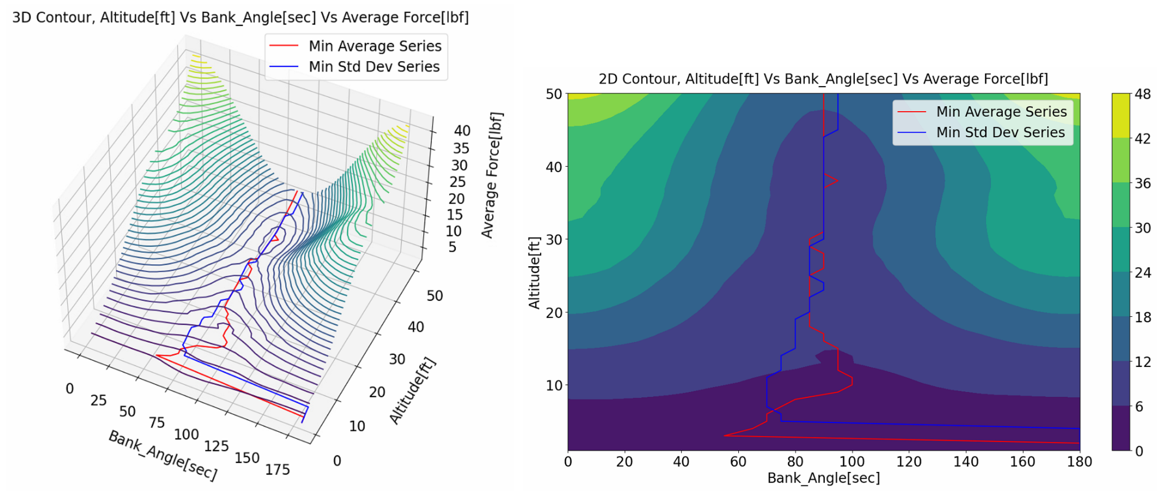

Fig. 6.5: Minimum average force 3D and 2D contour plots with bank angle series solutions shown.

Fig. 6.6 shows the 3D and 2D contour plots for the altitude versus angle versus standard deviation. The two lines depicted in the plot are the minimum average force solution in red and the minimum force standard deviation solution in blue. Note in this plot as expected, the minimum force standard deviation follows the lowest path while the minimum average force deviates. Note the minimum average solution and the minimum standard deviation solution are the same between Fig. 6.5 and Fig. 6.6. This is because the minimum average force bank angle series and minimum standard deviation bank angle series solutions are only dependent on the altitude.

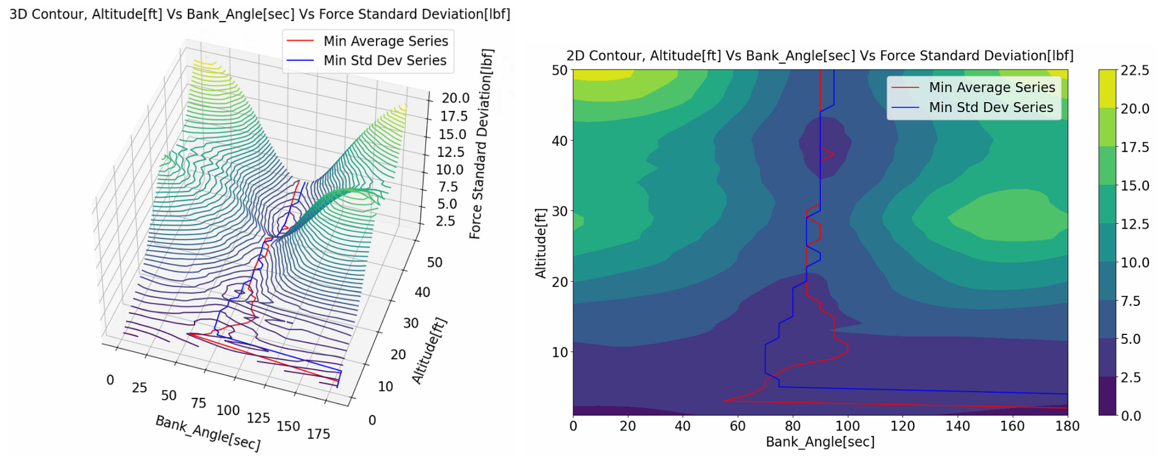

Fig. 6.6: Minimum force standard deviation 3D and 2D contour plots with bank angle series solutions shown.

Similar to the minimum average force plot, Fig. 6.7 shows the 3D and 2D contour plots for the altitude versus angle versus average moment. The two lines depicted in the plot are the minimum average moment solution in red and the minimum moment standard deviation solution in blue. Similar to the previous plots, the minimum average moment solution follows the lowest path across the contour plot while the minimum standard deviation solution deviates from the lowest path occasionally.

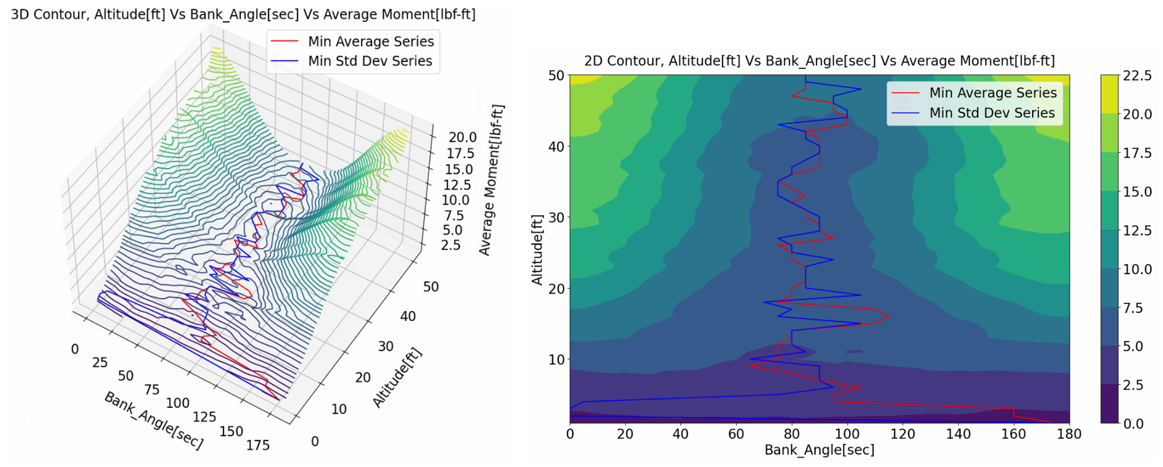

Fig. 6.7: Minimum average moment contour plots with bank angle series solutions.

Fig. 6.8 shows the 3D and 2D contour plots for the altitude versus angle versus moment standard deviation. As with the other plots, the two lines depicted in the plot are the minimum average moment solution in red and the minimum moment standard deviation solution in blue. As expected, the minimum moment standard deviation follows the lowest path while the minimum average moment deviates. The minimum average moment solution and the minimum moment standard deviation solutions are the same between Fig. 6.7 and Fig. 6.8. This is because the minimum average moment bank angle series and minimum moment standard deviation bank angle series solutions are only dependant on the altitude.

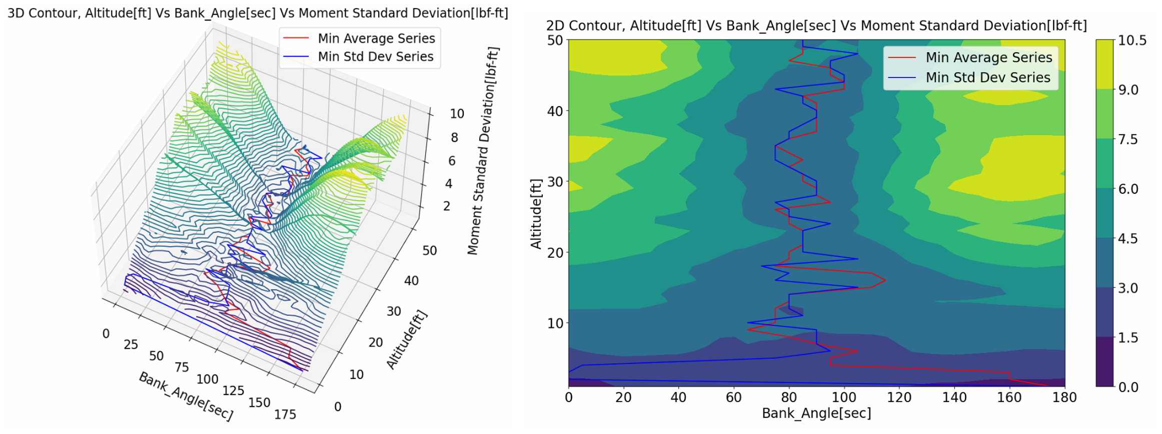

Fig. 6.8: Minimum moment standard deviation 3D and 2D contour plots with bank angle series solutions shown.

6.2 Force and Moment Orientation Contour Plots

Fig. 6.9 shows the 3D and 2D contour plots of the altitude versus time versus minimum force. The plots show that the minimum force decreases significantly with decrease in altitude. This is likely due to wind protection from the generic destroyer.

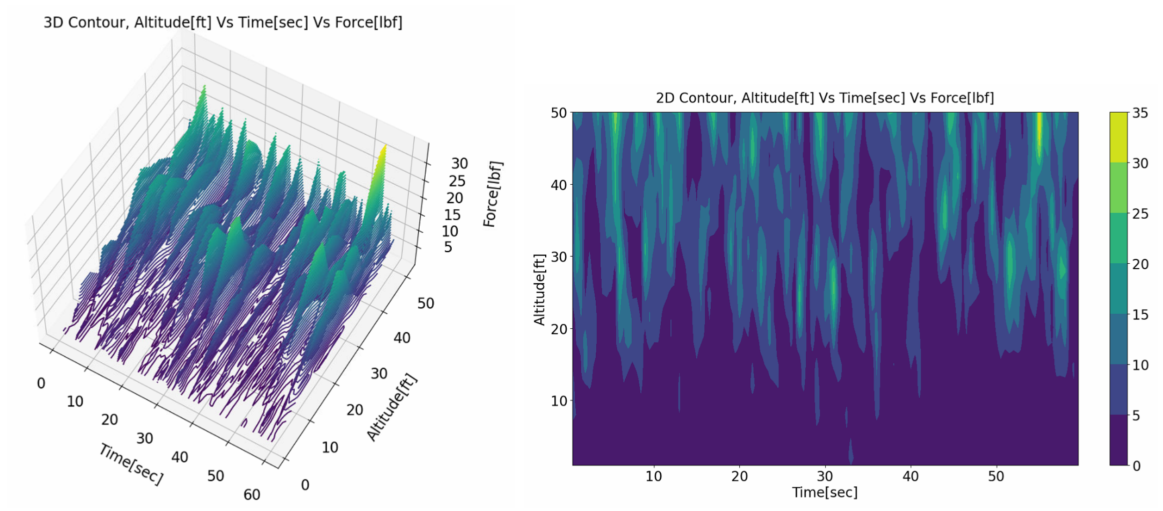

Fig. 6.9: Altitude versus time versus minimum force 3D and 2D contour plots.

Fig. 6.10 shows the altitude versus time versus minimum force bank angle scatter and 2D contour plots. The plots show the bank angles that minimize the forces acting on the V-BAT are clustered around 90 degrees for most altitudes except near the deck of the ship where the bank angle is more sporadic. These plots, in conjunction with Fig. 6.9, are significant because they show the bank angle that corresponds to the exact minimum force at each time step and altitude. A UAS pilot or computer could look at the minimum force contour plot in conjunction with the minimum force bank angle and plan any number of landing trajectories that always meet the path that follows lowest possible force.

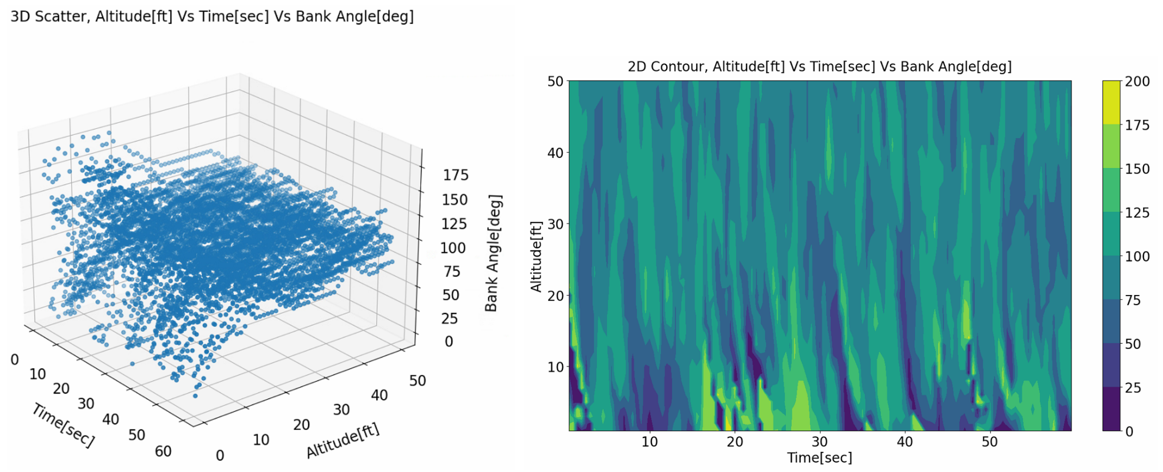

Fig. 6.10: Altitude versus time versus minimum force bank angle scatter and 2D contour plots.

Similarly Fig. 6.11 shows the 3D and 2D contour plots of the altitude versus time versus minimum moment. Like Fig. 6.9, the plots show that the minimum moment decreases with decrease in altitude due to the generic destroyer.

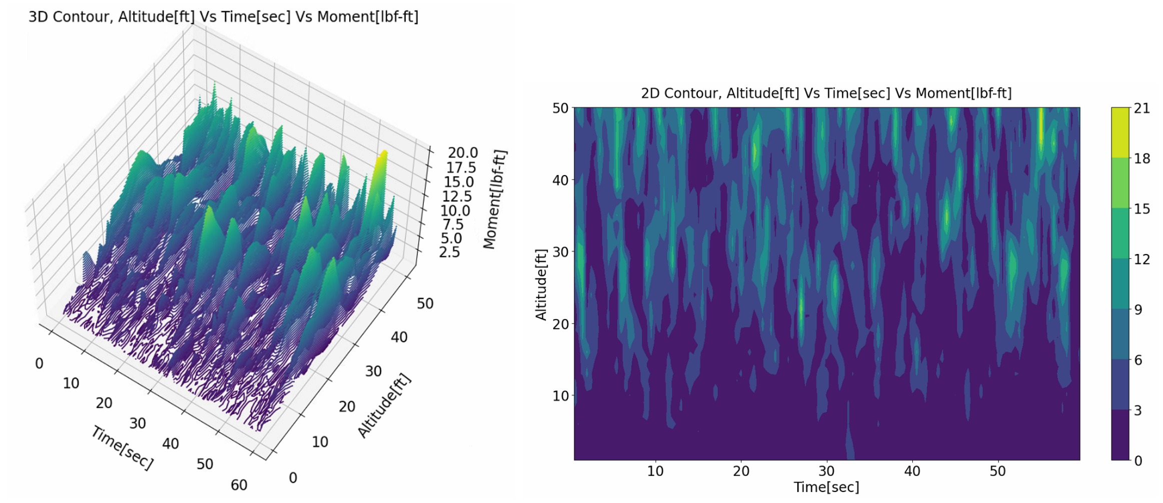

Fig. 6.11: Altitude versus time versus minimum moment 3D and 2D contour plots.

Fig. 6.12 shows the altitude versus time versus minimum moment bank angle scatter and 2D contour plots. The plots show the bank angles that minimize the moments acting on the V-BAT are clustered around 90 degrees for most altitudes, however, the data is spread out more than the minimum force angle data. The minimum moment bank angle data near the deck of the ship is becomes more random and deviates farther from a bank angle of 90 degrees. As before, these plots, Fig. 6.12 in conjunction with Fig. 6.11, are significant because they show the bank angle that corresponds to the exact minimum moment at each time step and altitude. A UAS pilot or computer could look at the minimum moment contour plot in conjunction with the minimum moment bank angle and plan trajectories and that will always meet the path that follows lowest possible moment.

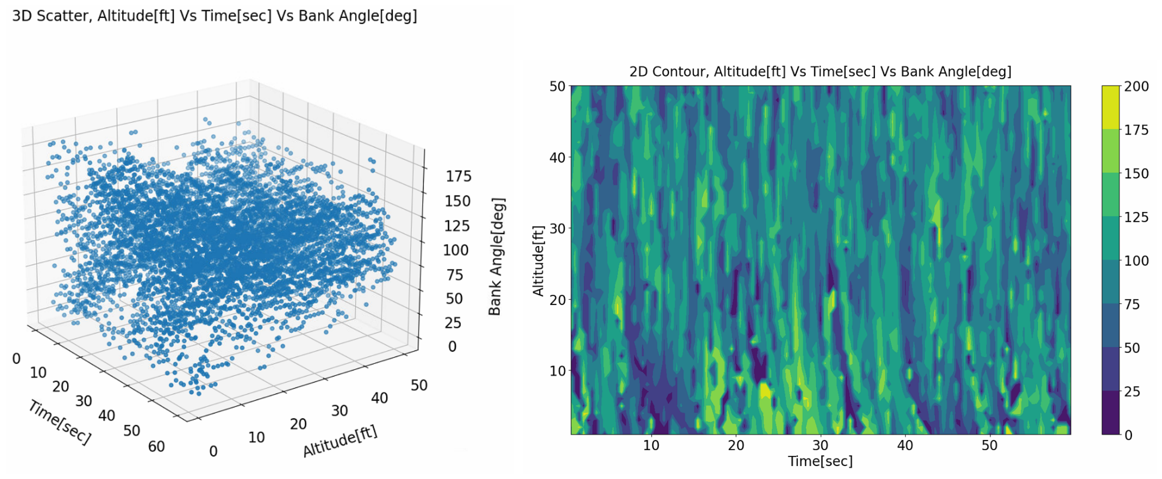

Fig. 6.12: Altitude versus time versus minimum moment bank angle scatter and 2D contour plots.

6.3 Vertical Descent Trajectory Simulation

Table 6.1 shows a prescribed descent trajectory with the V-BAT starting at an altitude of 50 ft and descending to 1 ft above the deck in 10 seconds. This data is used to simulate a set of bank angle series shown in Table 6.2.

Table 6.1: Trajectory data used to simulate V-BAT model over five different vertical trajectories.

Time[s]	x-position[ft]	y-position[ft]	z-position[ft]	θ[deg]	ψ[deg]
0.0	49.0	0.0	50.0	90.0	0.0
0.5	49.0	0.0	47.0	90.0	0.0
1.0	49.0	0.0	44.0	90.0	0.0
1.5	49.0	0.0	41.0	90.0	0.0
2.0	49.0	0.0	38.0	90.0	0.0
2.5	49.0	0.0	35.0	90.0	0.0
3.0	49.0	0.0	32.0	90.0	0.0
3.5	49.0	0.0	29.0	90.0	0.0
4.0	49.0	0.0	26.0	90.0	0.0
4.5	49.0	0.0	23.0	90.0	0.0
5.0	49.0	0.0	21.0	90.0	0.0
5.5	49.0	0.0	19.0	90.0	0.0
6.0	49.0	0.0	17.0	90.0	0.0
6.5	49.0	0.0	15.0	90.0	0.0
7.0	49.0	0.0	13.0	90.0	0.0
7.5	49.0	0.0	11.0	90.0	0.0
8.0	49.0	0.0	9.0	90.0	0.0
8.5	49.0	0.0	7.0	90.0	0.0
9.0	49.0	0.0	5.0	90.0	0.0
9.5	49.0	0.0	3.0	90.0	0.0
10.0	49.0	0.0	1.0	90.0	0.0

Table 6.2 shows five different series of bank angles corresponding to the prescribed decent trajectory described in Table 6.1. Note each of the bank angles in Table 6.1 correspond to each of the times and positions in Table 6.2 respectively. The first series describes the

V-Bat landing at 0 degrees bank angle for the whole trajectory. The second series describes the V-BAT landing with it's wings turned adjacent to the wind at 90 degrees bank angle with no variation. The third series describes the V-BAT following the minimum average force bank angle series. The fourth series describes the V-BAT following the minimum force standard deviation bank angle series. The fifth series describes the V-BAT following the minimum force bank angles obtained from the minimum force bank angle contour plot. Note the first four series do not depend on time but the minimum force angle contour series does.

Table 6.2: Bank angle series used to simulate the V-BAT model over five different vertical landing trajectories.

0 deg ϕ[deg]	90 deg ϕ[deg]	Min Avg ϕ[deg]	Min Std Dev ϕ[deg]	Min F ϕ[deg]
0.0	90.0	90.0	95.0	105.0
0.0	90.0	90.0	95.0	110.0
0.0	90.0	90.0	90.0	105.0
0.0	90.0	90.0	90.0	110.0
0.0	90.0	95.0	90.0	100.0
0.0	90.0	90.0	90.0	55.0
0.0	90.0	90.0	90.0	55.0
0.0	90.0	85.0	85.0	100.0
0.0	90.0	90.0	85.0	75.0
0.0	90.0	85.0	90.0	95.0
0.0	90.0	85.0	85.0	90.0
0.0	90.0	85.0	80.0	115.0
0.0	90.0	90.0	80.0	120.0
0.0	90.0	95.0	80.0	125.0
0.0	90.0	95.0	75.0	100.0
0.0	90.0	100.0	70.0	70.0
0.0	90.0	95.0	70.0	35.0
0.0	90.0	75.0	70.0	50.0
0.0	90.0	70.0	75.0	60.0
0.0	90.0	55.0	0.0	30.0
0.0	90.0	0.0	0.0	15.0

Simulating the prescribed descent trajectory given in Table 6.1 with each of the five corresponding bank angle series given in Table 6.2 gives the five force versus time curves in Fig 6.13. As shown in the figure, the trajectory from the 0 degree angle series incurs the highest forces. The 90 degree angle series follows a low force path at higher altitudes but incurs higher forces closer to the landing pad where the forces become more sporadic. The minimum average force and minimum standard deviation cases follow just above the minimum force contour case at higher altitudes but at lower altitudes they incur more force similar to the 90 degree case. As shown in the figure, the minimum force contour case follows the minimum force path for the duration of the trajectory.

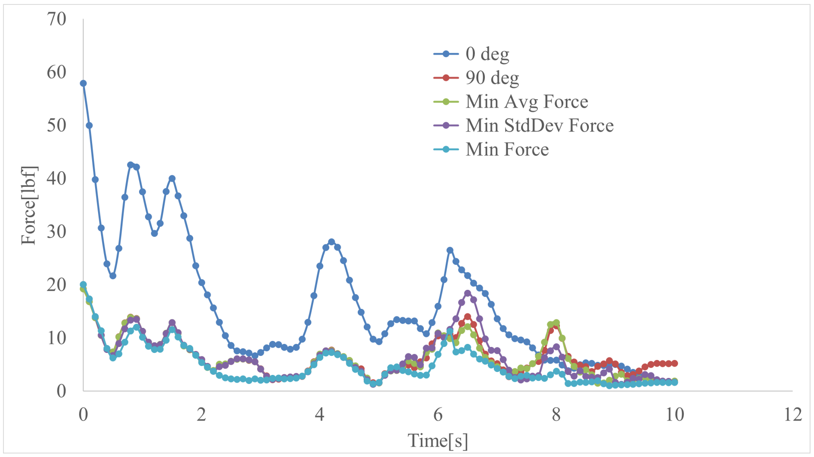

Fig. 6.13: Force versus time vertical landing trajectories.

Table 6.3 shows the maximum, minimum, average, and standard deviation of the forces acting on the V-BAT using each bank angle series over the prescribed trajectory. The 0 degree series incurred the greatest maximum force at 57.91 lbf, almost three times higher than the other bank angle series. The 0 degree series also incurred the highest average force at 16.07 lbf and the largest standard deviation at 11.99 lbf. The minimum force series incurred the lowest average force of 4.83 lbf, about 1.5 lbf less than the next lowest average. The minimum force series also incurred the second lowest standard deviation after the 90 degree series.

Table 6.3: Maximum, minimum, average, and standard deviation of the forces acting on the V-BAT over the five vertical landing trajectories.

Bank Angle Series	0 deg ϕ	90 deg ϕ	Min Avg ϕ	Min Std Dev ϕ	Min F ϕ
Max Force[lbf]	57.91	19.25	19.25	20.05	20.05
Min Force[lbf]	1.67	1.59	1.35	1.30	1.04
Avg Force[lbf]	16.07	6.67	6.27	6.39	4.83
SD Force[lbf]	11.99	3.57	3.79	4.25	3.65

The corresponding moment plot using the minimum force bank angle series is given in Fig. 6.14. As shown in the figure, the 0 degree series incurs the highest moment for most of the trajectory until almost 8 seconds into the trajectory where it mixes in with the other trajectories. The 90 degree, minimum average, minimum standard deviation, and minimum force series follow similar paths for the first 6.5 seconds of the trajectory then each series incurs different moments. For the last 3 seconds of the trajectory the 0 degree, 90 degree and minimum average series stay relatively close together, meanwhile, the minimum force and minimum standard deviation series incur higher moments than the other 3 series.

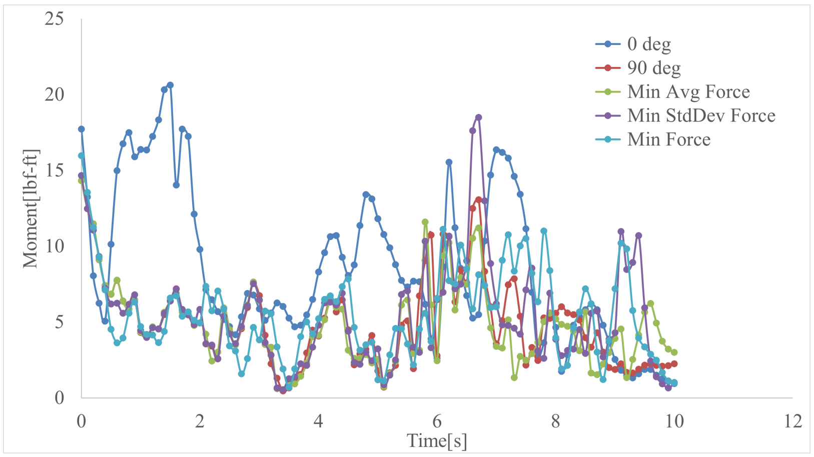

Fig. 6.14: Moment versus time vertical landing trajectories.

Table 6.4 shows the maximum, minimum, average, and standard deviation of the moments acting on the V-BAT using each bank angle series over the prescribed trajectory. The 0 degree series incurred the greatest maximum moment at 20.62 lbf-ft, over 2 lbf-ft higher than the next highest maximum moment. The 0 degree series also incurred the highest average moment at 8.49 lbf-ft and the largest standard deviation at 5.08 lbf-ft. The minimum average series incurred the lowest average moment at 4.84 lbf-ft and the lowest standard deviation of 2.69 lbf-ft. The minimum force series incurred the second highest average moment of 5.54 lbf-ft and the second lowest standard deviation of 2.88 lbf-ft.

Table 6.4: Maximum, minimum, average, and standard deviation of the moments acting on the V-BAT over the five vertical landing trajectories.

Bank Angle Series	0 deg ϕ	90 deg ϕ	Min Avg ϕ	Min Std Dev ϕ	Min F ϕ
Max Moment[lbf-ft]	20.62	14.32	14.32	18.49	15.97
Min Moment[lbf-ft]	1.01	0.46	0.56	0.53	0.69
Avg Moment[lbf-ft]	8.49	4.99	4.84	5.44	5.54
SD Moment[lbf-ft]	5.08	2.91	2.69	3.38	2.88

Optimizing for minimum forces over the trajectory, this data shows the minimum force bank angle series achieved the best results. The minimum force series achieved average or better force statistics compared to the other series in terms of minimizing the forces and achieved average moment statistics compared to the other series in terms of minimizing the moments while optimizing for minimum forces. The minimum average force bank angle series performed second best. The minimum average series achieved the good results for minimizing the forces and achieved the best results for minimizing the moments. Overall the 0 degree bank angle series performed the worst in all categories.

A summary and conclusions of this work are presented in Chapter 7.

CHAPTER 7
SUMMARY AND CONCLUSIONS

A summary of this work and research conclusions are presented in this chapter.

7.1 Work Summary

The methods used to calculate the required forces and moments acting on an aircraft with an arbitrary geometry composed of simple geometric shapes have been presented. These methods can be applied to any arbitrary geometry and produce reasonable approximate results for the forces and moments acting on the aircraft. The objectives of this research are to evaluate the forces and moments during the vertical landing phase of an approximated V-BAT model and identify the vertical landing trajectory that minimizes the variation in forces and moments acting on the V-BAT model.

The methods used include an approximation for calculating the aircraft's mass, moment of inertia, and aerodynamic properties. The mass and inertia are calculated based on the summation of the mass and moment of inertia of individual geometric components including cuboids, spheres, cylinders, wings, and rotors. Together, these components can approximate any number of complex aircraft.

The methods used to calculate the required aerodynamic forces and moments the approximated aircraft must produce during a simulation based on the 6 degree-of-freedom equations of motion have been presented. The effects of adverse conditions such as complex airwake are included in the simulation of the aircraft using gust disturbance modeling.

An overview of an approximate V-BAT model configuration and a prescribed landing trajectory mimicking the actual 128 V-BAT has been shown. Simulation results produced by SAASHA using the approximate V-BAT model, prescribed landing trajectory, and atmospheric database are given. The landing simulation results show the V-BAT model's position, velocity, and required forces and moments at each time step along the trajectory.

The V-BAT's landing is comprised of two parts, the landing approach and the landing descent. This work focuses on mapping the forces and moments acting on the V-BAT in different orientations during landing descent to the deck of the destroyer. To map the V-Bat's decent, the V-Bat model is simulated staying in place over a series of altitudes and orientation bank angles. Force and moment data is recorded at each altitude, orientation, bank angle and time step.

The data gathered in the data sweep is analyzed in two ways. The first way is to compute the average and standard deviation of the forces and moments at each altitude and bank angle. Next the data is used to generate a contour plot of the altitudes versus bank angles versus forces or moments showing the orientations that minimize the average and standard deviation of the forces and moments. An optimum series of orientation bank angles is found by searching each contour plot and finding the orientation angles that minimizes the average and standard deviation of the forces and moments.

The second way to analyze the data is to compare the forces and moments at each altitude and time step to generate a contour plot of the altitudes versus time steps versus orientation bank angles that minimize the forces and moments at each time step. Using these contour plots a UAS Pilot or computer can chart a landing course that will minimize the variation in forces and moments acting on the V-BAT.

The simulation results show the series of bank angles that minimize the average forces and moments as well as the standard deviation of the forces and moments at each altitude in the altitude-bank angle sweep. The results also show the bank angles that produce the minimum forces and moments at each altitude and time step in the altitude-bank angle sweep.

The results also show the forces and moments acting on the V-BAT during five prescribed vertical descent landing trajectory simulations. The simulations show using the 90 degree bank angle series reduces the maximum force the V-BAT experienced from 57.91 lbf, using the 0 degree bank angle series, to 19.25 lbf. Using the 90 degree bank angle series also reduces the average force acting on the V-BAT from 16.07 lbf with a standard deviation of

11.99 lbf, using the 0 degree bank angle series, to 6.67 lbf with a standard deviation of 3.57 lbf. The simulations also show using the 90 degree bank angle series reduces the maximum moment from 20.62 lbf-ft, using the 0 degree bank angle series, to 14.32 lbf-ft. Finally, using the 90 degree bank angle series also reduces the average moment acting on the V-BAT from 8.49 lbf-ft with a standard deviation of 5.08 lbf-ft, using the 0 degree bank angle series, to 4.99 lbf-ft with a standard deviation of 2.91 lbf-ft.

The simulations show using the minimum average and standard deviation force bank angle series reduces the maximum force from 57.9 lbf, using the 0 degree bank angle series, to 19.25 lbf and 20.05 lbf respectively. Using the minimum average and standard deviation force bank angle series also reduces the average force acting on the V-BAT from 16.07 lbf with a standard deviation of 11.99 lbf, using the 0 degree bank angle series, to 6.27 lbf with a standard deviation of 3.79 lbf and 6.39 lbf with a standard deviation of 4.25 lbf respectively. The simulations also show using the minimum average and standard deviation force bank angle series reduces the maximum moment from 20.62 lbf-ft, using the 0 degree bank angle series, to 14.32 lbf-ft and 18.49 lbf-ft respectively. Finally, using the minimum average and standard deviation force bank angle series also reduces the average moment acting on the V-BAT from 8.49 lbf-ft with a standard deviation of 5.08 lbf-ft, using the 0 degree bank angle series, to 4.84 lbf-ft with a standard deviation of 2.69 lbf-ft and 5.44 lbf-ft with a standard deviation of 3.38 lbf-ft respectively.

The simulations show using the minimum force bank angle series reduces the maximum force the V-BAT experienced from 57.91 lbf, using the 0 degree bank angle series, to 20.05 lbf which is over 37 lbf less than the 0 degree bank angle series. Using the minimum force bank angle series also reduces the average force acting on the V-BAT from 16.07 lbf with a standard deviation of 11.99 lbf, in the 0 degree bank angle series, to 4.83 lbf with a standard deviation of 3.65 lbf. The simulations also show, using the minimum force bank angle series reduces the maximum moment from 20.62 lbf-ft, using the 0 degree bank angle series, to 15.97 lbf-ft. Finally, using the minimum force bank angle series also reduces the average moment acting on the V-BAT from 8.49 lbf-ft with a standard deviation of 5.08 lbf-ft, using the 0 degree bank angle series, to 5.54 lbf-ft with a standard deviation of 2.88 lbf-ft.

7.2 Conclusion

These simulation results show the minimum force bank angle series performed best in minimizing the variation in forces and moments acting on the V-BAT. The four bank angle series compared to the 0 degree bank angle series, produced similar results in reducing the maximum force and moment experienced by the V-BAT during the trajectory. The minimum force bank angle series achieved the best performance in minimizing the average force and standard deviation acting on the V-BAT over the full landing trajectory. The minimum force bank angle series achieved an average force of 4.83 lbf with a standard deviation of 3.65 lbf. The 90 degree, minimum average, and minimum standard deviation bank angle series achieved similar standard deviations but experienced between 1.4 and 1.9 lbf higher average force over the course of the landing trajectory.

Using the 0 degree bank angle series, the average force experienced by the V-BAT was 16.07 lbf with a standard deviation of 11.99 lbf. Compared to the 0 degree bank angle series, the minimum force bank angle series averaged 11.24 lbf less over the course of the trajectory, which equates to less than a third of the average force experienced using the 0 degree bank angle series. Note the average force experienced by the 0 degree bank angle series is large enough that the average force experienced using the minimum force bank angle series is almost a full standard deviation outside of the average force experienced using the 0 degree bank angle series.

In conclusion, reorienting the V-BAT so the wings are adjacent to the wind or using the minimum average and standard deviation force and moment bank angle series produces good results in minimizing the variation in forces and moments. However, the variation in forces and moments acting on the approximated V-BAT model are best minimized using a vertical landing trajectory bank angle series prescribed by the minimum force and moment contour plot data.

REFERENCES

[1] Moulton, B. C. and Hunsaker, D. F., "Simplified Mass and Inertial Estimates for Aircraft with Components of Constant Density," *AIAA SciTech 2023 Forum*, 2023, p. 2432.

[2] naval technology.com, "Martin UAV's V-BAT UAS flies on 11th MEU missions on board USS Portland," `https://www.navy.mil/Resources/Photo-Gallery/igphoto/2003057171/`, 2021.

[3] northropgrumman.com, "Fire Scout," `https://www.northropgrumman.com/what-we-do/air/fire-scout`, 2023.

[4] Figueira, J. M. P., *The use of offline simulation tools to estimate Ship-Helicopter Operating Limitations*, Ph.D. thesis, UNIVERSITE AIX-MARSEILLE, 2017.

[5] Snyder, M. R., Kang, H. S., Brownell, C. J., and Burks, J. S., "Validation of ship air wake simulations and investigation of ship air wake impact on rotary wing aircraft," *Naval Engineers Journal*, Vol. 125, No. 1, 2013, pp. 49–50.

[6] Reddy, K., Toffoletto, R., and Jones, K., "Numerical simulation of ship airwake," *Computers & fluids*, Vol. 29, No. 4, 2000, pp. 451–465.

[7] Cao, Y. and Qin, Y., "Insight into the factors affecting the safety of take-off and landing of the ship-borne helicopter," *Proceedings of the Institution of Mechanical Engineers, Part G: Journal of Aerospace Engineering*, Vol. 236, No. 10, 2022, pp. 1925–1946.

[8] Yoo, C.-S., Park, B.-J., Cho, A., Kang, Y.-s., et al., "Wake modeling and simulation of tilt rotor UAV shipboard landing," *2014 14th International Conference on Control, Automation and Systems (ICCAS 2014)*, IEEE, 2014, pp. 1635–1639.

[9] Misra, G., Gao, T., and Bai, X., "Modeling and simulation of UAV carrier landings," *AIAA Scitech 2019 Forum*, 2019, p. 1981.

[10] Rehan, M., Akram, F., Shahzad, A., Shams, T., and Ali, Q., "Vertical take-off and landing hybrid unmanned aerial vehicles: An overview," *The Aeronautical Journal*, Vol. 126, No. 1306, 2022, pp. 2017–2057.

[11] Segui, M., Mantilla, M., and Botez, R. M., "Design and validation of an aerodynamic model of the cessna citation x horizontal stabilizer using both OpenVSP and digital Datcom," *International Journal of Aerospace and Mechanical Engineering*, Vol. 12, No. 1, 2018, pp. 45–53.

[12] Bagheri, S., *Modeling, simulation and control system design for civil unmanned aerial vehicle (UAV)*, Master's thesis, UMEA University, 2014.

[13] Elbedewy, A. A., Mohamed, S. S., Hamed, H. O., Kapeel, I., Safwat, E., Abozeid, M., and Kamel, A. M., "Modeling and Simulation of Small UAV Flight Dynamics," *The International Undergraduate Research Conference*, Vol. 5, The Military Technical College, 2021, pp. 541–547.

[14] Enomoto, M. and Yamamoto, Y., "Modelling, simulation and navigation experiments of Unmanned Aerial Vehicle," *2015 IEEE International Conference on Mechatronics and Automation (ICMA)*, 2015, pp. 482–487.

[15] Huang, J., Wang, B., Deng, Y., Ning, K., and Zhang, Y., "Aerodynamic Design of Fixed-wing Mode for A Ducted-fan Tiltrotor UAV by Digital DATCOM," *2022 International Conference on Unmanned Aircraft Systems (ICUAS)*, 2022, pp. 859–864.

[16] Jodeh, N. M., *Development of autonomous unmanned aerial vehicle platform: modeling, simulating, and flight testing*, Ph.D. thesis, Air Force Institute of Technology, 2006.

[17] Kapeel, E. H., Safwat, E., Kamel, A. M., Khalil, M. K., Elhalwagy, Y. Z., and Hendy, H., "Physical Modeling, Simulation and Validation of Small Fixed-Wing UAV," *Unmanned Systems*, Vol. 11, No. 04, 2023, pp. 327–350.

[18] Movahhed, S. R. and Hamed, M. A., "Calculating Aerodynamic Coefficients of Fixed Wing Aircrafts Using DATCOM Software with Special Focus on Rudderless Flying-wing UAVs," *The 21st International Conference of Iranian Aerospace Society*, 2023.

[19] Ou, Q., Chen, X., Park, D., Marburg, A., and Pinchin, J., "Integrated Flight Dynamics Modelling for Unmanned Aerial Vehicles," *2008 IEEE/ASME International Conference on Mechtronic and Embedded Systems and Applications*, 2008, pp. 570–575.

[20] Rauf, A., Zafar, M. A., Ashraf, Z., and Akhtar, H., "Aerodynamic modeling and state-space model extraction of a UAV using DATCOM and Simulink," *2011 3rd International Conference on Computer Research and Development*, Vol. 4, 2011, pp. 88–92.

[21] Shankar, P., Chung, W.-T., Husman, J., and Wells, V., "A novel software framework for teaching aircraft dynamics and control," *Computer Applications in Engineering Education*, Vol. 23, No. 1, 2015, pp. 63–71.

[22] Wong, D. R., Ou, Q., Sinclair, M., Li, Y. J., Chen, X. Q., and Marburg, A., "Unmanned Aerial Vehicle Flight Model Validation Using On-Board Sensing and Instrumentation," *2008 15th International Conference on Mechatronics and Machine Vision in Practice*, 2008, pp. 109–114.

[23] www.pdas.com, "Description of Digital Datcom," `https://www.pdas.com/datcomDescription.html`, 2022.

[24] Vogeltanz, T. and Jašek, R., "Free software for the modelling and simulation of a mini-UAV," *Proceedings of the Mathematics and Computers in Science and Industry, Varna, Bulgaria*, 2014, pp. 13–15.

[25] navy.mil, "220809-M-WN068-1008," `https://www.naval-technology.com/news/martin-uavs-v-bat-uas-flies-aboard-uss-portland/?cf-view`, 2022.

[26] Polsky, S. A., Wilkinson, C., Nichols, J., Ayers, D., Mercado-Perez, J., and Davis, T. S., “Development and application of the SAFEDI tool for virtual dynamic interface ship airwake analysis,” *54th AIAA aerospace sciences meeting*, 2016, p. 1771.

[27] Polsky, S., Imber, R., Czerwiec, R., and Ghee, T., “A computational and experimental determination of the air flow around the landing deck of a US Navy destroyer (DDG): Part II,” *37th AIAA fluid dynamics conference and exhibit*, 2007, p. 4484.

[28] Polsky, S. and Naylor, S., “CVN Airwake Modeling and Integration: Initial Steps in the Creation and Implementation of a Virtual Burble for F-18 Carrier Landing Simulations,” *AIAA Modeling and Simulation Technologies Conference and Exhibit*, 2005, p. 6298.

[29] Polsky, S., “A computational study of unsteady ship airwake,” *40th AIAA Aerospace Sciences Meeting & Exhibit*, 2002, p. 1022.

[30] Forsythe, J. R., Lynch, E., Polsky, S., and Spalart, P., “Coupled flight simulator and CFD calculations of ship airwake using kestrel,” *53rd AIAA Aerospace Sciences Meeting*, 2015, p. 0556.

[31] Phillips, W. F., “Aircraft Equations of Motion : Newton’s Second Law for Rigid-Body Dynamics,” *Mechanics of Flight*, chap. 7, John Wiley & Sons, Inc., 2nd ed., 2010, pp. 725–735.

[32] Hibbeler, R. C., *Engineering Mechanics Statics and Dynamics*, Pearson, 14th ed., 2016.

[33] Beer, F. P., Johnson, E. R. J., and Eisenberg, E. R., *Vector Mechanics for Engineers Statics*, MCGraw Hill, 8th ed., 2007.

[34] Beer, F. P., Johnson, E. R. J., and Eisenberg, E. R., *Vector Mechanics for Engineers Dynamics*, MCGraw Hill, 8th ed., 2007.

[35] Budynas, R. G. and Nisbett, J. K., *Shigley’s Mechanical Engineering Design*, McGraw Hill, 10th ed., 2014.

[36] Hoerner, S. F., *Fluid-Dynamic Drag*, Author, Midland Park, New Jersey, 1958.

[37] White, F. M., *Viscous Fluid Flow*, McGraw Hill, 2006.

[38] Phillips, W. F., *Mechanics of Flight*, chap. 1.8, John Wiley & Sons, Inc., Hoboken, 2nd ed., 2010, pp. 46–93.

[39] Phillips, W. F., *Mechanics of Flight*, chap. 1, John Wiley & Sons, Inc., New Jersey, 2nd ed., 2010, pp. 1–134.

[40] Beard, R. W. and McLain, T. W., *Small Unmanned Aircraft: Theory and Practice*, Princeton university press, 2012.

[41] Stengel, R. F., *Flight Dynamics*, Princeton University Press, 2004.

[42] Phillips, W. F., “Overview of Propulsion : Propeller Blade Theory,” *Mechanics of Flight*, chap. 2, John Wiley & Sons, Inc., 2nd ed., 2010, pp. 173–202.

[43] Phillips, W. F., "Aircraft Equations of Motion : Inertial and Gyroscopic Coupling," *Mechanics of Flight*, chap. 7, John Wiley & Sons, Inc., 2nd ed., 2010, pp. 805–806.

[44] Phillips, W. F., "Aircraft Flight Simulation," *Mechanics of Flight*, chap. 11, John Wiley & Sons, Inc., 2nd ed., 2010, pp. 989–1068.

[45] shield.ai, "Features and Specs," `https://shield.ai/v-bat/`, 2022.

APPENDICES

APPENDIX A
INPUT FILES

A.1 V-BAT Input File

```
{
   "simulation" : {
        "begin_time[sec]" : 0.0,
        "end_time[sec]" : 15.0,
        "time_step[sec]" : 0.01,
        "ground_plane[ft]" : 0.0
   },
   "atmosphere" : {
        "properties" : "standard",
        "constant_wind[ft/s]": [57.89,0.0,0.0],
        "gusts" : {
            "database" : {
                "numpy_binary" : "../databases/Case1_FS_V2_10s.npz",
                "units" : "meters",
                "original_wind_over_deck[ft/s]" : 57.89,
                "shift_origin[ft]" : [0.0, 0.0, 0.0],
                "show_bounding_box" : "red"
            },
            "ramp_in[sec]" : 0.0
        }
   },
    "aircraft" : {
        "filepath" : "vbat2.json"
    },
    "scene_components" : {
        "filepath" : "NATO_scene.json"
    },
    "trajectory" : {
        "xideal_touchdown_point[ft]" : [49.2, 0.0, 0.0],
        "specified" : true,
        "path" : {
            "interpolation" : "linear",
            "filepath" : "trajectory_1.csv",
            "x[ft]" : [100.0, 49.2],
            "y[ft]" : [0.0, 0.0],
            "z[ft]" : [100.0, 0.0],
            "bank_angle[deg]" : [0.0, 0.0],
            "elevation_angle[deg]" : [0.0, 90.0],
            "heading[deg]" : [0.0, 0.0]
        },
        "initial" : {
            "velocity[ft/s]" : 0.0,
            "alpha[deg]" : 0.0,
            "beta[deg]" : 0.0,
            "p[deg/s]" : 0.0,
            "q[deg/s]" : 0.0,
            "r[deg/s]" : 0.0,
            "x[ft]" : 0.0,
            "y[ft]" : 0.0,
            "z[ft]" : 200.0,
            "bank_angle[deg]" : 0.0,
            "elevation_angle[deg]" : 0.0,
            "heading[deg]" : 0.0
        }
    },
    "view" : {
        "aircraft_geometry" : true,
        "live_scene" : false,
        "time_history" : true
    }
}
```

A.2 V-BAT Geometry and Mass File

```
{
    "airfoils" : {
        "filepath" : "../databases/db_airfoils.json"
    },
    "propulsion" : {
        "filepath" : "../databases/db_propulsion.json"
    },
    "components" : {
        "main_wing" : {
            "type" : "wing",
            "include_aero" : true,
            "location[ft]" : [3.5, 0.35, -0.35],
            "weight[lbf]" : 2.0,
            "side" : "both",
            "span[ft]" : 4.85,
            "chord[ft]" : [1.105, 0.71],
            "thickness[%]" : [12.0,12.0],
            "sweep[deg]" : 0.0,
            "dihedral[deg]" : 0.0,
            "airfoil" : "NACA_4",
            "aerodynamics" : {
                "mounting_angle[deg]" : 0.0,
                "alpha_L0[deg]" : 0.0,
                "CL,alpha[1/rad]" : 5.5,
                "CD0" : 0.1,
                "CD1" : 0.0,
                "e" : 0.8,
                "Cm0" : 0.0,
                "Cm,alpha[1/rad]" : 0.0
            },
            "color" : "black"
        },
        "prop" : {
            "type" : "rotor",
            "include_aero" : true,
            "location[ft]" : [1.5, 0.0, 0.0],
            "orientation[deg]" : [0.0, 0.0, 0.0],
            "weight[lbf]" : 3.0,
            "rotation" : "RH",
            "blade_count" : 10,
            "rotor_diameter[ft]" : 1.4,
            "blade_chord[ft]" : [0.2, 0.2],
            "blade_thickness[%]" : [12.0,12.0],
            "airfoil" : "NACA_4",
            "hub_diameter[ft]" : 0.25,
            "hub_height[ft]" : 0.2,
            "propulsion" : "prop2",
            "color" : "black",
            "rpm" : 50,
            "thrust[lbf]" : 0.5
        },
        "nose" : {
            "type" : "sphere",
            "include_aero" : true,
            "location[ft]" : [8.2, 0.0, 0.0],
            "orientation[deg]" : [0.0, 0.0, 0.0],
            "weight[lbf]" : 5.0,
            "radius[ft]" : 0.35,
            "color" : "black"
        },
        "fuselage" : {
            "type" : "cylinder",
            "include_aero" : true,
            "location[ft]" : [4.87, 0.0, 0.0],
            "orientation[deg]" : [0.0, 0.0, 0.0],
            "weight[lbf]" : 28.0,
            "radius[ft]" : 0.35,
            "length[ft]" : 6.70,
            "color" : "black"
        },
        "ring" : {
            "type" : "cylinder",
            "include_aero" : true,
            "location[ft]" : [1.43, 0.0, 0.0],
            "orientation[deg]" : [0.0, 0.0, 0.0],
            "weight[lbf]" : 4.0,
            "radius[ft]" : 1.5,
            "radius_inner[ft]" : 1.375,
            "length[ft]" : 0.75,
            "color" : "black"
        },
        "fuselage_2" : {
            "type" : "cuboid",
            "include_aero" : false,
            "location[ft]" : [3.24, 0.0, 0.0],
            "orientation[deg]" : [0.0, 0.0, 0.0],
            "weight[lbf]" : 60.0,
            "length[ft]" : [3.22, 0.70, 0.70],
            "color" : "black"
        },
        "bounding_box" : {
            "type" : "cuboid",
            "include_aero" : false,
            "location[ft]" : [4.0, 0.0, 0.0],
            "orientation[deg]" : [0.0, 0.0, 0.0],
            "weight[lbf]" : 0.0,
            "length[ft]" : [15.0, 10.0, 10.0],
            "color" : "white"
        }
    }
}
```

APPENDIX B
DATA TABLES

B.1 Bank Angle Tables

Bank angles corresponding to the lowest average forces and moments and lowest standard deviation in forces and moments are given in Table B.1 and Table B.2.

Table B.1: Bank angles corresponding to the minimum average forces, force standard deviations, average moments, and moment standard deviations at altitudes from 50 ft to 26 ft.

Altitude[ft]	Min Avg Force ϕ	Min SD Force ϕ	Min Avg Moment ϕ	Min SD Moment ϕ
50.0	90.0	95.0	85	85
49.0	90.0	95.0	85	85
48.0	90.0	95.0	85	105
47.0	90.0	95.0	80	95
46.0	90.0	95.0	95	95
45.0	90.0	95.0	95	100
44.0	90.0	90.0	100	100
43.0	90.0	90.0	100	75
42.0	90.0	90.0	85	85
41.0	90.0	90.0	90	85
40.0	90.0	90.0	90	90
39.0	90.0	90.0	90	90
38.0	95.0	90.0	90	85
37.0	90.0	90.0	90	80
36.0	90.0	90.0	80	80
35.0	90.0	90.0	75	75
34.0	90.0	90.0	80	75
33.0	90.0	90.0	85	75
32.0	90.0	90.0	80	80
31.0	90.0	90.0	85	85
30.0	85.0	90.0	85	90
29.0	85.0	85.0	90	90
28.0	90.0	85.0	90	90
27.0	90.0	85.0	95	75
26.0	90.0	85.0	75	80

Table B.2: Bank angles corresponding to the minimum average forces, force standard deviations, average moments, and moment standard deviations at altitudes from 25 ft to 1 ft.

Altitude[ft]	Min Avg Force ϕ	Min SD Force ϕ	Min Avg Moment ϕ	Min SD Moment ϕ
25.0	85.0	85.0	80	80
24.0	85.0	90.0	80	95
23.0	85.0	90.0	85	85
22.0	85.0	85.0	85	85
21.0	85.0	85.0	85	85
20.0	85.0	85.0	80	85
19.0	85.0	80.0	80	105
18.0	85.0	80.0	75	70
17.0	90.0	80.0	110	80
16.0	90.0	80.0	115	75
15.0	95.0	80.0	110	105
14.0	95.0	75.0	80	80
13.0	95.0	75.0	80	80
12.0	95.0	75.0	75	80
11.0	100.0	70.0	75	85
10.0	100.0	70.0	75	65
9.0	95.0	70.0	65	90
8.0	80.0	70.0	80	90
7.0	75.0	70.0	90	90
6.0	70.0	75.0	105	95
5.0	70.0	75.0	95	75
4.0	65.0	180.0	95	5
3.0	55.0	180.0	160	0
2.0	180.0	180.0	160	0
1.0	180.0	180.0	175	175

APPENDIX C
CODE

C.1 SAASHA Wrapper Code

```
# test file to run saasha and compute the difference between the forces and moments

# change states
import numpy as np
from saasha.sim import *
import pandas as pd
import csv
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
# elementwise root sum square of three vectors of equal length
plot_everything = False
Record_FMA_Results_to_1_File = False
# Functions

# takes the root sum square of each component and returns an array of magnitudes
def rt_sum_sqr(x,y,z):
    f = np.zeros(len(x))
    for i in range(len(x)):
        f[i] = (x[i]**2 + y[i]**2 + z[i]**2)**0.5
    f=np.array(f)
    return f

# Searches for the minimum value and its corresponding angle
def min_finder(A,B):

    Values = []
    Angles = []

    for i in range(len(A[0,:])):
        vec = A[:,i]
        Val = np.min(vec)

        vec2 = []
        for j in range(len(vec)):
            vec2.append(vec[j])

        Ang = B[vec2.index(Val)]

        Values.append(Val)
        Angles.append(Ang)

    Values = np.array(Values)
    Angles = np.array(Angles)

    return Values, Angles

# creates a 3D scatter plot
def plot_3D_scatter(X_vals,Y_vals,Z_vals,X_label,Y_label,Z_label,Color):

    x_array = X_vals
    y_array = Y_vals
    z_array = Z_vals
    fig = plt.figure()
    ax = fig.add_subplot(111, projection='3d')

    # Create a 3D scatter plot
    ax.scatter(x_array, y_array, z_array, c=Color, marker='o')

    # Set labels for the axes
    ax.set_xlabel(X_label)
    ax.set_ylabel(Y_label)
    ax.set_zlabel(Z_label)

    # Set a title for the plot
    ax.set_title('3D Scatter ' + Y_label + " Vs " + X_label + " Vs " + Z_label)

    # Show the plot
    plt.show()
    return

# creates a 3D contour plot
def plot_3D_contour(X_vals,Y_vals,Z_vals,X_label,Y_label,Z_label,Color):

    fig = plt.figure()

    ax = plt.axes(projection='3d')
    ax.contour3D(X_vals,Y_vals , Z_vals, 50, cmap='viridis')
    # Set labels for the axes
    ax.set_xlabel(X_label)
    ax.set_ylabel(Y_label)
    ax.set_zlabel(Z_label)

    # Set a title for the plot
    ax.set_title('3D Contour ' + Y_label + " Vs " + X_label + " Vs " + Z_label)

    # Show the plot
    plt.show()
    return

# creates a 2D contour plot
def plot_2D_contour(X_vals,Y_vals,Z_vals,X_label,Y_label,Z_label,Color):

    #fig = plt.figure()

    h = plt.contourf(X_vals, Y_vals, Z_vals, cmap='viridis')
    plt.axis('scaled')
    plt.colorbar()

    plt.xlabel(X_label)
    plt.ylabel(Y_label)
    plt.title('2D Contour' + Y_label + " Vs " + X_label + " Vs " + Z_label)

    plt.show()
    return

# creates a 3D scatter plot, 3D contour plot, and a 2D contour plot
def plot_all(X_vals,Y_vals,Z_vals,X_label,Y_label,Z_label,Color):

    x_array = X_vals
    y_array = Y_vals
    z_array = Z_vals

    # 2D Contour Plot
    fig3 = plt.figure()
    h = plt.contourf(X_vals, Y_vals, Z_vals, cmap='viridis')
    plt.axis('scaled')
    plt.colorbar()
    plt.xlabel(X_label)
    plt.ylabel(Y_label)
    plt.title('2D Contour ' + Y_label + " Vs " + X_label + " Vs " + Z_label)

    # 3D Contour Plot
    fig2 = plt.figure()

    ax2 = plt.axes(projection='3d')
    ax2.contour3D(X_vals, Y_vals, Z_vals, 50, cmap='viridis')
    # Set labels for the axes
    ax2.set_xlabel(X_label)
    ax2.set_ylabel(Y_label)
    ax2.set_zlabel(Z_label)

    # Set a title for the plot
    ax2.set_title('3D Contour ' +Y_label + " Vs " + X_label + " Vs " + Z_label)


    # 3D Scatter Plot
    fig1 = plt.figure()
    ax1 = fig1.add_subplot(111, projection='3d')

    # Create a 3D scatter plot
    #ax1.scatter(x_array, y_array, z_array, c=Color, marker='o')

    ax1.scatter(x_array, y_array, z_array, cmap='viridis', marker='o')

    # Set labels for the axes
    ax1.set_xlabel(X_label)
    ax1.set_ylabel(Y_label)
    ax1.set_zlabel(Z_label)

    # Set a title for the plot
    ax1.set_title('3D Scatter ' +Y_label + " Vs " + X_label + " Vs " + Z_label)

    # Show the plots
    plt.show()
    return

def minimum_data(Ang_Mat, Max_Mat, Min_Mat, Avg_Mat, Sdv_Mat):


    Max_Sol = []
    Min_Sol = []
    Avg_Sol = []
    Sdv_Sol = []

    for i in range(len(Ang_Mat[:,1])):

        Bnk_Vals = []
        Max_Vals = []
        Min_Vals = []
        Avg_Vals = []
        Sdv_Vals = []

        for j in range(len(Ang_Mat[i,:])):
            Bnk_Vals.append(Ang_Mat[i,j])
            Max_Vals.append(Max_Mat[i,j])
            Min_Vals.append(Min_Mat[i,j])
            Avg_Vals.append(Avg_Mat[i,j])
            Sdv_Vals.append(Sdv_Mat[i,j])
        # Forces calculate the minimum of the max min mean and sd of each force
        Min_Max_Val = np.min(Max_Mat[i,:])
        Min_Min_Val = np.min(Min_Mat[i,:])
        Min_Avg_Val = np.min(Avg_Mat[i,:])
        Min_Sdv_Val = np.min(Sdv_Mat[i,:])


        # Max
        Min_Max_angle = Ang_Mat[i, Max_Vals.index(Min_Max_Val)]
        Min_Max_Min = Min_Mat[i, Max_Vals.index(Min_Max_Val)]
        Min_Max_Avg = Avg_Mat[i, Max_Vals.index(Min_Max_Val)]
        Min_Max_Sdv = Sdv_Mat[i, Max_Vals.index(Min_Max_Val)]


        # Min
        Min_Min_angle = Ang_Mat[i, Min_Vals.index(Min_Min_Val)]
        Min_Min_Max = Max_Mat[i, Min_Vals.index(Min_Min_Val)]
        Min_Min_Avg = Avg_Mat[i, Min_Vals.index(Min_Min_Val)]
        Min_Min_Sdv = Sdv_Mat[i, Min_Vals.index(Min_Min_Val)]


        # Avg
        Min_Avg_angle = Ang_Mat[i, Avg_Vals.index(Min_Avg_Val)]
        Min_Avg_Max = Max_Mat[i, Avg_Vals.index(Min_Avg_Val)]
        Min_Avg_Min = Min_Mat[i, Avg_Vals.index(Min_Avg_Val)]
        Min_Avg_Sdv = Sdv_Mat[i, Avg_Vals.index(Min_Avg_Val)]


        # Sdv
        Min_Sdv_angle = Ang_Mat[i, Sdv_Vals.index(Min_Sdv_Val)]
        Min_Sdv_Max = Max_Mat[i, Sdv_Vals.index(Min_Sdv_Val)]
        Min_Sdv_Min = Min_Mat[i, Sdv_Vals.index(Min_Sdv_Val)]
        Min_Sdv_Avg = Avg_Mat[i, Sdv_Vals.index(Min_Sdv_Val)]


        # append each force and moment to the list of values to be used to generate the csv files

        Max_Sol.append([Min_Max_angle, Min_Max_Val, Min_Max_Min, Min_Max_Avg, Min_Max_Sdv])
        Min_Sol.append([Min_Min_angle, Min_Min_Max, Min_Min_Val, Min_Min_Avg, Min_Min_Sdv])
        Avg_Sol.append([Min_Avg_angle, Min_Avg_Max, Min_Avg_Min, Min_Avg_Val, Min_Avg_Sdv])
        Sdv_Sol.append([Min_Sdv_angle, Min_Sdv_Max, Min_Sdv_Min, Min_Sdv_Avg, Min_Sdv_Val])


    Max_Sol = np.array(Max_Sol)
    Min_Sol = np.array(Min_Sol)
    Avg_Sol = np.array(Avg_Sol)
    Sdv_Sol = np.array(Sdv_Sol)

    return Max_Sol, Min_Sol, Avg_Sol, Sdv_Sol


# Writes a matrix of any size to a csv file
def write_csv_file( Property_Name, File_Name, Data_Matrix):

    property = "," + Property_Name

    with open(File_Name, 'w') as Write_File:

        for j in range(len(Data_Matrix[1, :])):
            if j == 0:
                Write_File.write(Property_Name)
            else:
                Write_File.write(property)
        Write_File.write("\n")
        for i in range(len(Data_Matrix[:, 1])):
            for j in range(len(Data_Matrix[1, :])):
                if j == 0:
                    Write_File.write("{:>20.12E}".format(Data_Matrix[i, j])) # altitude
                else:
                    Write_File.write(",{:>20.12E}".format(Data_Matrix[i, j])) # altitude
            Write_File.write("\n")
    return

if __name__=="__main__":

    # Run landing Trajectories from 0 to 90

    # Specify the output file name
    time_frame = "0_60_20_"
    output_file_x = time_frame + "state_hist_"


    output_file = output_file_x + "combined.xlsx"
    # Base file we are starting from
    filename = "input_2.json"

    Force_Results_File = time_frame + "Force_Results.csv"
    Moment_Results_File = time_frame + "Moment_Results.csv"
    Force_Devation_Results_File = time_frame + "Force_Dev_Results.csv"
    Moment_Devation_Results_File = time_frame + "Moment_Dev_Results.csv"
    Min_Force_Devation_Results_File = time_frame + "Min_F_Dev_Results.csv"
    Min_Moment_Devation_Results_File = time_frame + "Min_M_Dev_Results.csv"


    FDA_Altitude_File = time_frame + "FDA_Altitude.csv"
    FDA_Angles_File = time_frame + "FDA_Angles.csv"

    FDA_Max_Force_File = time_frame + "FDA_Max_Force.csv"
    FDA_Min_Force_File = time_frame + "FDA_Min_Force.csv"
    FDA_AVG_Force_File = time_frame + "FDA_AVG_Force.csv"
    FDA_StdDev_Force_File = time_frame + "FDA_StdDev_F.csv"

    FDA_Max_Moment_File = time_frame + "FDA_Max_Moment.csv"
    FDA_Min_Moment_File = time_frame + "FDA_Min_Moment.csv"
    FDA_AVG_Moment_File = time_frame + "FDA_AVG_Moment.csv"
    FDA_StdDev_Moment_File = time_frame + "FDA_StdDev_M.csv"


    FMA_Altitude_File = time_frame + "FMA_Altitude.csv"
    FMA_Time_File = time_frame + "FMA_Time.csv"
    FMA_Force_Angle_File = time_frame + "FMA_F_Angle.csv"
    FMA_Force_File = time_frame + "FMA_Force.csv"
    FMA_Moment_Angle_File = time_frame + "FMA_M_Angle.csv"
    FMA_Moments_File = time_frame + "FMA_Moment.csv"


    # Final Plot file mapping out the values
    FMA_Results_File = time_frame + "FMA_Results.csv"

    csv_files = []
    altitudes = []
    angles = []

    # Forces
    Max_Force = []
    Min_Force = []
    Mean_Force = []
    Force_std_dev = []
```

```
# Moments
Max_Moment = []
Min_Moment = []
Mean_Moment = []
Moment_std_dev = []

Total_Force_Matrix = []
Total_Moment_Matrix = []

# min Values arrays

Min_Max_Force = []
Min_Min_Force = []
Min_Mean_Force = []
Min_StdDev_Force = []

Min_Max_Moment = []
Min_Min_Moment = []
Min_Mean_Moment = []
Min_StdDev_Moment = []

Min_Max_Force_angle = []
Min_Min_Force_angle = []
Min_Mean_Force_angle = []
Min_StdDev_Force_angle = []

Min_Max_Moment_angle = []
Min_Min_Moment_angle = []
Min_Mean_Moment_angle = []
Min_StdDev_Moment_angle = []

# FDA Arrays

FDA_Altitude = []
FDA_Angles = []

FDA_Max_Force = []
FDA_Min_Force = []
FDA_AVG_Force = []
FDA_StdDev_Force = []

FDA_Max_Moment = []
FDA_Min_Moment = []
FDA_AVG_Moment = []
FDA_StdDev_Moment = []


# FMA arrays
Min_FMA_Min_Force = []
Min_FMA_Min_Moment = []

Min_FMA_Time = []
Min_FMA_Altitude = []

Min_FMA_Min_Force_Angle = []
Min_FMA_Min_Moment_Angle = []

Min_FMA_Min_Force_2 = []
Min_FMA_Min_Moment_2 = []

Min_FMA_Time_2 = []
Min_FMA_Altitude_2 = []

Min_FMA_Min_Force_Angle_2 = []
```

```
Min_FMA_Min_Moment_Angle_2 = []

Time_2 = []

if Record_FMA_Results_to_1_File:
    csv_files.append(FMA_Results_File)

csv_files.append(FMA_Altitude_File)
csv_files.append(FMA_Time_File)
csv_files.append(FMA_Force_Angle_File)
csv_files.append(FMA_Force_File)
csv_files.append(FMA_Moment_Angle_File)
csv_files.append(FMA_Moments_File)

# FDA Result files
csv_files.append(Min_Force_Devation_Results_File)
csv_files.append(Min_Moment_Devation_Results_File)
csv_files.append(Force_Devation_Results_File)
csv_files.append(Moment_Devation_Results_File)

# FDA Files
csv_files.append(FDA_Altitude_File)
csv_files.append(FDA_Angles_File)

csv_files.append(FDA_Max_Force_File)
csv_files.append(FDA_Min_Force_File)
csv_files.append(FDA_AVG_Force_File)
csv_files.append(FDA_StdDev_Force_File)

csv_files.append(FDA_Max_Moment_File)
csv_files.append(FDA_Min_Moment_File)
csv_files.append(FDA_AVG_Moment_File)
csv_files.append(FDA_StdDev_Moment_File)



#csv_files.append(Force_Results_File)
#csv_files.append(Moment_Results_File)

# Load the json file
json_string = open(filename).read()
input_dict = json.loads(json_string)
Min_Alt = []
rows_counter = 0
# loop for updating the values then running saasha
for j in range(50,0,-1):
    #j = 50
    columns_counter = 0
    input_dict["trajectory"]["path"]["z[ft]"] = [j,j]


    Min_angle = []
    FDA_Alt = []

    Alt_Max_Force = []
    Alt_Min_Force = []
    Alt_Mean_Force = []
    Alt_StdDev_Force = []

    Alt_Max_Moment = []
    Alt_Min_Moment = []
    Alt_Mean_Moment = []
    Alt_StdDev_Moment = []

    FMA_Min_F = []
```

```
FMA_Min_M = []

alt = []

for i in range(0,185,5):

# Specify the file path where you want to save the JSON data
    #filename_new = "landing_input_" + str(i) + ".json"
    input_filename = "landing_input_x.json"

    # csv output file name
    states_filename = "0_60_20_state_hist_" + str(j) + "_" + str(i) + ".csv"
    # Update the output file name
    input_dict["simulation"]["states_filename"] = states_filename
    # Update the landing bank angle
    input_dict["trajectory"]["path"]["bank[deg]"] = [i,i]


    # Open the file to save the json data
    with open(input_filename, "w") as json_file:
        # Use the json.dump() function to write the data to the file
        json.dump(input_dict, json_file)


    # Start of saasha simulation
    print("sim " + str(j) + "_" + str(i) + " Started")
    # Initialize sim
    print("\nReading input file...")
    mysim = sim(input_filename)
    print("Done")
    # Initialize State
    y = mysim.initialize_state()
    # Run simulation
    mysim.run(y)
    print("sim " + str(j) + "_" + str(i) + " Finished")

    # List of file paths to your CSV files
    csv_files.append(states_filename)

    # Altitudes and Angles for CSV Files
    altitudes.append(j)
    angles.append(i)

    alt.append(j)
    # file to open and analyze
    fn = states_filename

    print("Reading forces from file ", fn)
    # get the data from the file
    State_Data = np.genfromtxt(fn, delimiter=",", skip_header=1)

    # print(State_Data)
    # print(State_Data[1,20])
    # print(State_Data[1,21])
    # print(State_Data[1,22])
    Time_Values = State_Data[:, 0]

    Total_Force = rt_sum_sqr(State_Data[:, 20], State_Data[:, 21], State_Data[:, 22]) #
         calculate the total forces
    Total_Moment = rt_sum_sqr(State_Data[:, 23], State_Data[:, 24], State_Data[:, 25]) #
         calculate the total moments

    # print("##########################################################################")
    # print(Total_Force)
    # print("##########################################################################")
    FMA_Time_Values = State_Data[:,0]
    FMA_Min_F.append(Total_Force)
    FMA_Min_M.append(Total_Moment)

    # print('xxxxxxxxxxxxxxxxxxxx')
    # print(FMA_Min_F)
    # print('xxxxxxxxxxxxxxxxxxxx')

    # if i == 0:
    #     FMA_Min_F = Total_Force
    #     FMA_Min_M = Total_Moment
    # else:
    #     FMA_Min_F = np.vstack((FMA_Min_F, Total_Force))
    #
    #     FMA_Min_M = np.vstack((FMA_Min_M, Total_Moment))



    # Calculate max, min, mean, and std dev of the forces

    # Force values
    F_Max = np.max(Total_Force)
    F_Min = np.min(Total_Force)
    F_Mean = np.mean(Total_Force)
    F_Std_Dev = np.std(Total_Force)

    # Moment values
    M_Max = np.max(Total_Moment)
    M_Min = np.min(Total_Moment)
    M_Mean = np.mean(Total_Moment)
    M_Std_Dev = np.std(Total_Moment)


    # append minimum values of each set of values for this altitude
    Min_angle.append(i)
    FDA_Alt.append(j)

    Alt_Max_Force.append(F_Max)
    Alt_Min_Force.append(F_Min)
    Alt_Mean_Force.append(F_Mean)
    Alt_StdDev_Force.append(F_Std_Dev)

    Alt_Max_Moment.append(M_Max)
    Alt_Min_Moment.append(M_Min)
    Alt_Mean_Moment.append(M_Mean)
    Alt_StdDev_Moment.append(M_Std_Dev)
    # end of minimum values

    # Forces list used in csv 1 by x
    Max_Force.append(F_Max)
    Min_Force.append(F_Min)
    Mean_Force.append(F_Mean)
    Force_std_dev.append(F_Std_Dev)

    # Moments list used in csv 1 by x
    Max_Moment.append(M_Max)
    Min_Moment.append(M_Min)
    Mean_Moment.append(M_Mean)
    Moment_std_dev.append(M_Std_Dev)

    columns_counter += 1

FMA_Min_F_Mat = np.array(FMA_Min_F)
```

```
FMA_Min_M_Mat = np.array(FMA_Min_M)

Alt_FMA_Min_Force, Alt_FMA_Min_Force_Angle = min_finder(FMA_Min_F_Mat, Min_angle)
Alt_FMA_Min_Moment, Alt_FMA_Min_Moment_Angle = min_finder(FMA_Min_M_Mat, Min_angle)
alt = []
for k in range(len(Alt_FMA_Min_Force)):
    Min_FMA_Time_2.append(FMA_Time_Values[k])
    Min_FMA_Altitude_2.append(j)

    Min_FMA_Min_Force_2.append(Alt_FMA_Min_Force[k])
    Min_FMA_Min_Moment_2.append(Alt_FMA_Min_Moment[k])

    Min_FMA_Min_Force_Angle_2.append(Alt_FMA_Min_Force_Angle[k])
    Min_FMA_Min_Moment_Angle_2.append(Alt_FMA_Min_Moment_Angle[k])

    alt.append(j)


# FDA Values
FDA_Altitude.append(FDA_Alt)
FDA_Angles.append(Min_angle)

FDA_Max_Force.append(Alt_Max_Force)
FDA_Min_Force.append(Alt_Min_Force)
FDA_AVG_Force.append(Alt_Mean_Force)
FDA_StdDev_Force.append(Alt_StdDev_Force)

FDA_Max_Moment.append(Alt_Max_Moment)
FDA_Min_Moment.append(Alt_Min_Moment)
FDA_AVG_Moment.append(Alt_Mean_Moment)
FDA_StdDev_Moment.append(Alt_StdDev_Moment)

# FMA Values
Min_FMA_Time.append(FMA_Time_Values)
Min_FMA_Altitude.append(alt)

Min_FMA_Min_Force.append(Alt_FMA_Min_Force)
Min_FMA_Min_Moment.append(Alt_FMA_Min_Moment)

Min_FMA_Min_Force_Angle.append(Alt_FMA_Min_Force_Angle)
Min_FMA_Min_Moment_Angle.append(Alt_FMA_Min_Moment_Angle)
Time_2.append(FMA_Time_Values)


# print("%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%")
# print((FMA_Min_F[:,0]))
# print((FMA_Min_F[0,:]))
# print("%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%")


# Forces calculate the minimum of the max min mean and sd of each force
Min_F_Max = np.min(Alt_Max_Force)
Min_F_Min = np.min(Alt_Min_Force)
Min_F_Mean = np.min(Alt_Mean_Force)
Min_F_Std_Dev = np.min(Alt_StdDev_Force)

# Moments calculate the minimum of the max min mean and sd of each Moment
Min_M_Max = np.min(Alt_Max_Moment)
Min_M_Min = np.min(Alt_Min_Moment)
Min_M_Mean = np.min(Alt_Mean_Moment)
Min_M_Std_Dev = np.min(Alt_StdDev_Moment)


# Angle of minimum forces
Min_F_Max_angle = Min_angle[Alt_Max_Force.index(Min_F_Max)]
```

```
    Min_F_Min_angle = Min_angle[Alt_Min_Force.index(Min_F_Min)]
    Min_F_Mean_angle = Min_angle[Alt_Mean_Force.index(Min_F_Mean)]
    Min_F_Std_Dev_angle = Min_angle[Alt_StdDev_Force.index(Min_F_Std_Dev)]

    # Angle of minimum moments
    Min_M_Max_angle = Min_angle[Alt_Max_Moment.index(Min_M_Max)]
    Min_M_Min_angle = Min_angle[Alt_Min_Moment.index(Min_M_Min)]
    Min_M_Mean_angle = Min_angle[Alt_Mean_Moment.index(Min_M_Mean)]
    Min_M_Std_Dev_angle = Min_angle[Alt_StdDev_Moment.index(Min_M_Std_Dev)]


    # append each force and moment to the list of values to be used
    # to generate the csv files
    Min_Alt.append(j)

    Min_Max_Force.append(Min_F_Max)
    Min_Min_Force.append(Min_F_Min)
    Min_Mean_Force.append(Min_F_Mean)
    Min_StdDev_Force.append(Min_F_Std_Dev)

    Min_Max_Moment.append(Min_M_Max)
    Min_Min_Moment.append(Min_M_Min)
    Min_Mean_Moment.append(Min_M_Mean)
    Min_StdDev_Moment.append(Min_M_Std_Dev)

    # Find the angle for each minimum value
    Min_Max_Force_angle.append(Min_F_Max_angle)
    Min_Min_Force_angle.append(Min_F_Min_angle)
    Min_Mean_Force_angle.append(Min_F_Mean_angle)
    Min_StdDev_Force_angle.append(Min_F_Std_Dev_angle)

    Min_Max_Moment_angle.append(Min_M_Max_angle)
    Min_Min_Moment_angle.append(Min_M_Min_angle)
    Min_Mean_Moment_angle.append(Min_M_Mean_angle)
    Min_StdDev_Moment_angle.append(Min_M_Std_Dev_angle)

    rows_counter += 1
# end loop
# test1 = np.array([1,2,3])
# test2 = np.array([4,5,6])
# test3 = np.array([7,8,9])
#
# test4 = np.array([test1, test2, test3])

# print(test4)
# print(test4[:,0])
# print(test4[0,:])

# FDA Arrays
FDA_Altitude = np.array(FDA_Altitude)
FDA_Angles = np.array(FDA_Angles)

FDA_Max_Force = np.array(FDA_Max_Force)
FDA_Min_Force = np.array(FDA_Min_Force)
FDA_AVG_Force = np.array(FDA_AVG_Force)
FDA_StdDev_Force = np.array(FDA_StdDev_Force)


FDA_Max_Moment = np.array(FDA_Max_Moment)
FDA_Min_Moment = np.array(FDA_Min_Moment)
FDA_AVG_Moment = np.array(FDA_AVG_Moment)
FDA_StdDev_Moment = np.array(FDA_StdDev_Moment)
```

```
FDA_Max_F, FDA_Min_F, FDA_Avg_F, FDA_Sdv_F = minimum_data(FDA_Angles, FDA_Max_Force,
     FDA_Min_Force, FDA_AVG_Force, FDA_StdDev_Force)
FDA_Max_M, FDA_Min_M, FDA_Avg_M, FDA_Sdv_M = minimum_data(FDA_Angles, FDA_Max_Moment,
     FDA_Min_Moment, FDA_AVG_Moment, FDA_StdDev_Moment)

# FMA Arrays
Min_FMA_Time = np.array(Min_FMA_Time)
Time_2 = np.array(Time_2)
Min_FMA_Altitude = np.array(Min_FMA_Altitude)

Min_FMA_Min_Force = np.array(Min_FMA_Min_Force)
Min_FMA_Min_Moment = np.array(Min_FMA_Min_Moment)

Min_FMA_Min_Force_Angle = np.array(Min_FMA_Min_Force_Angle)
Min_FMA_Min_Moment_Angle = np.array(Min_FMA_Min_Moment_Angle)

'''
print("time",Min_FMA_Time)
print(" ")
print("time 2",Time_2)
print(" ")
print("altitude",Min_FMA_Altitude)
print(" ")
print("angle",Min_FMA_Min_Force_Angle)
'''

if plot_everything:
    # Plot everything
    plot_all(Min_FMA_Time,Min_FMA_Altitude,Min_FMA_Min_Force_Angle,"Time[sec]","Altitude[ft]",/n
    "Force_Angle[deg]","b")

    plot_all(Min_FMA_Time,Min_FMA_Altitude,Min_FMA_Min_Force,"Time[sec]","Altitude[ft]",\n
    "Force[lbf]","b")

    plot_all(Min_FMA_Time,Min_FMA_Altitude,Min_FMA_Min_Moment_Angle,"Time[sec]","Altitude[ft]",\n
    "Moment_Angle[deg]","b")

    plot_all(Min_FMA_Time,Min_FMA_Altitude,Min_FMA_Min_Moment,"Time[sec]","Altitude[ft]",\n
    "Moment[lbf-ft]","b")


write_csv_file("Altitude[ft]", FMA_Altitude_File, Min_FMA_Altitude)
write_csv_file("Time[sec]", FMA_Time_File, Min_FMA_Time)
write_csv_file("Min_F_Angle[deg]", FMA_Force_Angle_File, Min_FMA_Min_Force_Angle)
write_csv_file("Min_Force[lbf]", FMA_Force_File, Min_FMA_Min_Force)
write_csv_file("Min_M_Angle[deg]", FMA_Moment_Angle_File, Min_FMA_Min_Moment_Angle)
write_csv_file("Min_Moment[lbf-ft]", FMA_Moments_File, Min_FMA_Min_Moment)


# Record max min avg and std dev values for contour plots
write_csv_file("Altitude[ft]", FDA_Altitude_File, FDA_Altitude)
write_csv_file("Bank_Angle[deg]", FDA_Angles_File, FDA_Angles)

write_csv_file("Max_F[lbf]", FDA_Max_Force_File, FDA_Max_Force)
write_csv_file("Min_F[lbf]", FDA_Min_Force_File, FDA_Min_Force)
write_csv_file("Mean_Force[lbf]", FDA_AVG_Force_File, FDA_AVG_Force)
write_csv_file("Force_Std_Dev[lbf]", FDA_StdDev_Force_File, FDA_StdDev_Force)

write_csv_file("Max_M[lbf]", FDA_Max_Moment_File, FDA_Max_Moment)
write_csv_file("Min_M[lbf]", FDA_Min_Moment_File, FDA_Min_Moment)
write_csv_file("Mean_Moment[lbf]", FDA_AVG_Moment_File, FDA_AVG_Moment)
write_csv_file("Moment_Std_Dev[lbf]", FDA_StdDev_Moment_File, FDA_StdDev_Moment)

# Record the FMA forces and moments to one sheet
if Record_FMA_Results_to_1_File:
```

```
# Record FMA Results
with open(FMA_Results_File, 'w') as FMA_Res_File:

    #FMA_Res_File.write("Altitude[ft],Time[sec],Min_F_Angle[deg],Min_F[lbf],\n
    Min_M_Angle[deg],Min_M[lbf],Altitude[ft]")
    # Titles
    for j in range(len(Min_FMA_Altitude[1, :])):
        FMA_Res_File.write("Altitude[ft],")
    FMA_Res_File.write(" ,")

    for j in range(len(Min_FMA_Altitude[1, :])):
        FMA_Res_File.write("Time[sec],")
    FMA_Res_File.write(" ,")

    for j in range(len(Min_FMA_Altitude[1, :])):
        FMA_Res_File.write("Min_F_Angle[deg],")
    FMA_Res_File.write(" ,")

    for j in range(len(Min_FMA_Altitude[1, :])):
        FMA_Res_File.write("Min_F[lbf],")
    FMA_Res_File.write(" ,")

    for j in range(len(Min_FMA_Altitude[1, :])):
        FMA_Res_File.write("Min_M_Angle[deg],")
    FMA_Res_File.write(" ,")

    for j in range(len(Min_FMA_Altitude[1, :])):
        FMA_Res_File.write("Min_M[lbf],")
    FMA_Res_File.write(" ,")

    for j in range(len(Min_FMA_Altitude[1, :])):
        FMA_Res_File.write("Altitude[ft],")

    FMA_Res_File.write("\n")

    # Results
    for i in range(len(Min_FMA_Altitude[:,1])):
        for j in range(len(Min_FMA_Altitude[1,:])):
            FMA_Res_File.write("{:>20.12E},".format(Min_FMA_Altitude[i,j]))# altitude
        FMA_Res_File.write(",")

        for j in range(len(Min_FMA_Altitude[1, :])):
            FMA_Res_File.write("{:>20.12E},".format(Min_FMA_Time[i,j])) # Time
        FMA_Res_File.write(",")

        for j in range(len(Min_FMA_Altitude[1, :])):
            FMA_Res_File.write("{:>20.12E},".format(Min_FMA_Min_Force_Angle[i,j])) # angle min
                force
        FMA_Res_File.write(",")

        for j in range(len(Min_FMA_Altitude[1, :])):
            FMA_Res_File.write("{:>20.12E},".format(Min_FMA_Min_Force[i,j])) # min force
        FMA_Res_File.write(",")

        for j in range(len(Min_FMA_Altitude[1, :])):
            FMA_Res_File.write("{:>20.12E},".format(Min_FMA_Min_Moment_Angle[i,j])) # angle min
                moment
        FMA_Res_File.write(",")

        for j in range(len(Min_FMA_Altitude[1, :])):
            FMA_Res_File.write("{:>20.12E},".format(Min_FMA_Min_Moment[i,j])) # min moment
        FMA_Res_File.write(",")

            # Error Calculations
            # Min_Force_Dev_Results_File.write(",{:>20.12E}".format((Min_Max_Force[i] -
                Min_Mean_Force[i])/Min_Mean_Force[i])) # min max force error
            # Min_Force_Dev_Results_File.write(",{:>20.12E}".format((Min_Mean_Force[i] -
                Min_Min_Force[i])/Min_Mean_Force[i])) # min min force error
            # Min_Force_Dev_Results_File.write(",{:>20.12E}".format(Min_StdDev_Force[i] /
                Min_Mean_Force[i])) # min std dev error

        for j in range(len(Min_FMA_Altitude[1, :])):
            FMA_Res_File.write("{:>20.12E},".format(Min_FMA_Altitude[i,j])) # altitude

        FMA_Res_File.write("\n")

# Record Min Force Results
with open(Min_Force_Devation_Results_File, 'w') as Min_Force_Dev_Results_File:

    Min_Force_Dev_Results_File.write(
        "Altitude[ft],\
        Min_F_Max_Angle[deg],Min_F_Max_Max[lbf],Min_F_Max_Min[lbf],Min_F_Max_Avg[lbf],\n
        Min_F_Max_Sdv[lbf],\
        Min_F_Min_Angle[deg],Min_F_Min_Max[lbf],Min_F_Min_Min[lbf],Min_F_Min_Avg[lbf],\n
        Min_F_Min_Sdv[lbf],\
        Min_F_Avg_Angle[deg],Min_F_Avg_Max[lbf],Min_F_Avg_Min[lbf],Min_F_Avg_Avg[lbf],\n
        Min_F_Avg_Sdv[lbf],\
        Min_F_Sdv_Angle[deg],Min_F_Sdv_Max[lbf],Min_F_Sdv_Min[lbf],Min_F_Sdv_Avg[lbf],\n
        Min_F_Sdv_Sdv[lbf],\
        Min_F_Max_Error,Min_F_Min_Error,Min_F_Std_Dev_Error,Altitude[ft]")
        #Min_F_Min_Angle[deg],Min_F_Min[lbf],\
        #Min_F_Mean_angle[deg],Min_F_Mean{lbf],\
        #Min_F_Std_Dev_Angle[deg],Min_F_Std_Dev[lbf],\
        #Min_F_Max_Error,Min_F_Min_Error,Min_F_Std_Dev_Error,Altitude[ft]")

    Min_Force_Dev_Results_File.write("\n")

    for i in range(len(Min_Alt)):
        Min_Force_Dev_Results_File.write("{:>20.12E}".format(Min_Alt[i]))# altitude

        Min_Force_Dev_Results_File.write(",{:>20.12E}".format(Min_Max_Force_angle[i])) # angle min
             max force
        Min_Force_Dev_Results_File.write(",{:>20.12E}".format(Min_Max_Force[i])) # min max force
        Min_Force_Dev_Results_File.write(",{:>20.12E}".format(FDA_Max_F[i,2])) # min max min force
        Min_Force_Dev_Results_File.write(",{:>20.12E}".format(FDA_Max_F[i,3])) # min max avg force
        Min_Force_Dev_Results_File.write(",{:>20.12E}".format(FDA_Max_F[i,4])) # min max sdv force

        Min_Force_Dev_Results_File.write(",{:>20.12E}".format(Min_Min_Force_angle[i])) # angle min
             min force
        Min_Force_Dev_Results_File.write(",{:>20.12E}".format(FDA_Min_F[i,1])) # min max sdv force
        Min_Force_Dev_Results_File.write(",{:>20.12E}".format(Min_Min_Force[i])) # min min force
        Min_Force_Dev_Results_File.write(",{:>20.12E}".format(FDA_Min_F[i,3])) # min max sdv force
        Min_Force_Dev_Results_File.write(",{:>20.12E}".format(FDA_Min_F[i,4])) # min max sdv force

        Min_Force_Dev_Results_File.write(",{:>20.12E}".format(Min_Mean_Force_angle[i])) # angle
             min avg force
        Min_Force_Dev_Results_File.write(",{:>20.12E}".format(FDA_Avg_F[i,1]))
        Min_Force_Dev_Results_File.write(",{:>20.12E}".format(FDA_Avg_F[i,2]))
        Min_Force_Dev_Results_File.write(",{:>20.12E}".format(Min_Mean_Force[i])) # min avg force
        Min_Force_Dev_Results_File.write(",{:>20.12E}".format(FDA_Avg_F[i,4]))

        Min_Force_Dev_Results_File.write(",{:>20.12E}".format(Min_StdDev_Force_angle[i])) # angle
             min std dev
        Min_Force_Dev_Results_File.write(",{:>20.12E}".format(FDA_Sdv_F[i,1]))
        Min_Force_Dev_Results_File.write(",{:>20.12E}".format(FDA_Sdv_F[i,2]))
        Min_Force_Dev_Results_File.write(",{:>20.12E}".format(FDA_Sdv_F[i,3]))
        Min_Force_Dev_Results_File.write(",{:>20.12E}".format(Min_StdDev_Force[i])) # min std dev

        # Error Calculations
        Min_Force_Dev_Results_File.write(",{:>20.12E}".format((Min_Max_Force[i] -
             Min_Mean_Force[i])/Min_Mean_Force[i])) # min max force error
        Min_Force_Dev_Results_File.write(",{:>20.12E}".format((Min_Mean_Force[i] -
             Min_Min_Force[i])/Min_Mean_Force[i])) # min min force error
        Min_Force_Dev_Results_File.write(",{:>20.12E}".format(Min_StdDev_Force[i] /
             Min_Mean_Force[i])) # min std dev error

        Min_Force_Dev_Results_File.write(",{:>20.12E}".format(Min_Alt[i])) # altitude
        Min_Force_Dev_Results_File.write("\n")

# Record Min Moment Results
with open(Min_Moment_Devation_Results_File, 'w') as Min_Moment_Dev_Results_File:

    Min_Moment_Dev_Results_File.write(
        "Altitude[ft],\
        Min_M_Max_Angle[deg],Min_M_Max_Max[lbf],Min_M_Max_Min[lbf],Min_M_Max_Avg[lbf],\n
        Min_M_Max_Sdv[lbf],\
        Min_M_Min_Angle[deg],Min_M_Min_Max[lbf],Min_M_Min_Min[lbf],Min_M_Min_Avg[lbf],\n
        Min_M_Min_Sdv[lbf],\
        Min_M_Avg_Angle[deg],Min_M_Avg_Max[lbf],Min_M_Avg_Min[lbf],Min_M_Avg_Avg[lbf],\n
        Min_M_Avg_Sdv[lbf],\
        Min_M_Sdv_Angle[deg],Min_M_Sdv_Max[lbf],Min_M_Sdv_Min[lbf],Min_M_Sdv_Avg[lbf],\n
        Min_M_Sdv_Sdv[lbf],\
        Min_M_Max_Error,Min_M_Min_Error,Min_M_Std_Dev_Error,Altitude[ft]")

    Min_Moment_Dev_Results_File.write("\n")

    for i in range(len(Min_Alt)):
        Min_Moment_Dev_Results_File.write("{:>20.12E}".format(Min_Alt[i])) # altitude

        Min_Moment_Dev_Results_File.write(",{:>20.12E}".format(Min_Max_Moment_angle[i])) # angle
             min max force
        Min_Moment_Dev_Results_File.write(",{:>20.12E}".format(Min_Max_Moment[i])) # min max force
        Min_Moment_Dev_Results_File.write(",{:>20.12E}".format(FDA_Max_M[i,2])) # min max min force
        Min_Moment_Dev_Results_File.write(",{:>20.12E}".format(FDA_Max_M[i,3])) # min max avg force
        Min_Moment_Dev_Results_File.write(",{:>20.12E}".format(FDA_Max_M[i,4])) # min max sdv force



        Min_Moment_Dev_Results_File.write(",{:>20.12E}".format(Min_Min_Moment_angle[i])) # angle
             min min force
        Min_Moment_Dev_Results_File.write(",{:>20.12E}".format(FDA_Min_M[i,1])) # min max sdv force
        Min_Moment_Dev_Results_File.write(",{:>20.12E}".format(Min_Min_Moment[i])) # min min force
        Min_Moment_Dev_Results_File.write(",{:>20.12E}".format(FDA_Min_M[i,3])) # min max sdv force
        Min_Moment_Dev_Results_File.write(",{:>20.12E}".format(FDA_Min_M[i,4])) # min max sdv force



        Min_Moment_Dev_Results_File.write(",{:>20.12E}".format(Min_Mean_Moment_angle[i])) # angle
             min avg force
        Min_Moment_Dev_Results_File.write(",{:>20.12E}".format(FDA_Avg_M[i,1]))
        Min_Moment_Dev_Results_File.write(",{:>20.12E}".format(FDA_Avg_M[i,2]))
        Min_Moment_Dev_Results_File.write(",{:>20.12E}".format(Min_Mean_Moment[i])) # min avg force
        Min_Moment_Dev_Results_File.write(",{:>20.12E}".format(FDA_Avg_M[i,4]))

        Min_Moment_Dev_Results_File.write(",{:>20.12E}".format(Min_StdDev_Moment_angle[i])) #
             angle min std dev
        Min_Moment_Dev_Results_File.write(",{:>20.12E}".format(FDA_Sdv_M[i,1]))
        Min_Moment_Dev_Results_File.write(",{:>20.12E}".format(FDA_Sdv_M[i,2]))
        Min_Moment_Dev_Results_File.write(",{:>20.12E}".format(FDA_Sdv_M[i,3]))
        Min_Moment_Dev_Results_File.write(",{:>20.12E}".format(Min_StdDev_Moment[i])) # min std dev

        # Error Calculations
        Min_Moment_Dev_Results_File.write(",{:>20.12E}".format((Min_Max_Moment[i] -
             Min_Mean_Moment[i]) / Min_Mean_Moment[i])) # min max moment error
        Min_Moment_Dev_Results_File.write(",{:>20.12E}".format((Min_Mean_Moment[i] -
             Min_Min_Moment[i]) / Min_Mean_Moment[i])) # min min moment error
        Min_Moment_Dev_Results_File.write(",{:>20.12E}".format(Min_StdDev_Moment[i] /
             Min_Mean_Moment[i])) # min std dev error

        Min_Moment_Dev_Results_File.write(",{:>20.12E}".format(Min_Alt[i])) # altitude
        Min_Moment_Dev_Results_File.write("\n")

# Record Force Results csv
with open(Force_Devation_Results_File, 'w') as Force_Dev_Results_File:

    Force_Dev_Results_File.write("Altitude[ft],Angle[deg],Max_Force[lbf],Min_Force[lbf],\n
    Mean_Force[lbf],Force_Standard_Deviation[lbf],Mean_F-StdDev[lbf],Mean_F+StdDev,Max_Percent,\n
    Min_Percent,Mean+Max_P,Mean-Min_P,Mean+StdDev,Mean-StdDev,Altitude[ft],Angle[deg]")
    Force_Dev_Results_File.write("\n")

    for i in range(len(altitudes)):
        Force_Dev_Results_File.write("{:>20.12E}".format(altitudes[i])) # altitude
        Force_Dev_Results_File.write(",{:>20.12E}".format(angles[i])) # angle
        Force_Dev_Results_File.write(",{:>20.12E}".format(Max_Force[i])) # max force
        Force_Dev_Results_File.write(",{:>20.12E}".format(Min_Force[i])) # min force
        Force_Dev_Results_File.write(",{:>20.12E}".format(Mean_Force[i])) # avg force
        Force_Dev_Results_File.write(",{:>20.12E}".format(Force_std_dev[i])) # force std dev
        Force_Dev_Results_File.write(",{:>20.12E}".format(Mean_Force[i] - Force_std_dev[i])) #
             Mean - std dev
        Force_Dev_Results_File.write(",{:>20.12E}".format(Mean_Force[i] + Force_std_dev[i])) #
             Mean + std dev

        Force_Dev_Results_File.write(",{:>20.12E}".format((Max_Force[i] - Mean_Force[i]) /
             Mean_Force[i])) # max percent
        Force_Dev_Results_File.write(",{:>20.12E}".format((Mean_Force[i] - Min_Force[i]) /
             Mean_Force[i])) # min percent

        Force_Dev_Results_File.write(",{:>20.12E}".format(Mean_Force[i] + (Max_Force[i] -
             Mean_Force[i])/Mean_Force[i])) # Mean + max percent
        Force_Dev_Results_File.write(",{:>20.12E}".format(Mean_Force[i] -
             (Mean_Force[i]-Min_Force[i])/Mean_Force[i])) # Mean - min percent
        Force_Dev_Results_File.write(",{:>20.12E}".format(Mean_Force[i] +
             (Force_std_dev[i]/Mean_Force[i]))) # standard dev + percent
        Force_Dev_Results_File.write(",{:>20.12E}".format(Mean_Force[i] -
             (Force_std_dev[i]/Mean_Force[i]))) # standard dev - percent

        Force_Dev_Results_File.write(",{:>20.12E}".format(altitudes[i])) # altitude
        Force_Dev_Results_File.write(",{:>20.12E}".format(angles[i])) # angle
        Force_Dev_Results_File.write("\n")

# record moment results csv
with open(Moment_Devation_Results_File, 'w') as Moment_Dev_Results_File:

    Moment_Dev_Results_File.write("Altitude[ft],Angle[deg],Max_Moment[lbf-ft],Min_Moment[lbf-ft],\n
    Mean_Moment[lbf-ft],Moment_Std_Dev[lbf-ft],Mean_M-StdDev[lbf],Mean_M+StdDev,Max_Percent,\n
    Min_Percent,Mean+Max_P,Mean-Min_P,Mean+StdDev,Mean-StdDev,Altitude[ft],Angle[deg]")
    Moment_Dev_Results_File.write("\n")
    for i in range(len(altitudes)):
        Moment_Dev_Results_File.write("{:>20.12E}".format(altitudes[i])) # altitude
        Moment_Dev_Results_File.write(",{:>20.12E}".format(angles[i])) # angle
        Moment_Dev_Results_File.write(",{:>20.12E}".format(Max_Moment[i])) # max force
        Moment_Dev_Results_File.write(",{:>20.12E}".format(Min_Moment[i])) # min force
        Moment_Dev_Results_File.write(",{:>20.12E}".format(Mean_Moment[i])) # avg force
        Moment_Dev_Results_File.write(",{:>20.12E}".format(Moment_std_dev[i])) # force std dev
        Moment_Dev_Results_File.write(",{:>20.12E}".format(Mean_Moment[i] - Moment_std_dev[i])) #
             Mean - std dev
        Moment_Dev_Results_File.write(",{:>20.12E}".format(Mean_Moment[i] + Moment_std_dev[i])) #
             Mean + std dev
        Moment_Dev_Results_File.write(",{:>20.12E}".format((Max_Moment[i] -
             Mean_Moment[i])/Mean_Moment[i])) # max percent
        Moment_Dev_Results_File.write(",{:>20.12E}".format((Mean_Moment[i] -
             Min_Moment[i])/Mean_Moment[i])) # min percent

        Moment_Dev_Results_File.write(",{:>20.12E}".format(Mean_Moment[i] + (Max_Moment[i] -
             Mean_Moment[i])/Mean_Moment[i])) # Mean + max percent
        Moment_Dev_Results_File.write(",{:>20.12E}".format(Mean_Moment[i] - (Mean_Moment[i] -
             Min_Moment[i])/Mean_Moment[i])) # Mean - min percent
        Moment_Dev_Results_File.write(",{:>20.12E}".format(Mean_Moment[i] +
             (Moment_std_dev[i]/Mean_Moment[i]))) # standard dev + percent
        Moment_Dev_Results_File.write(",{:>20.12E}".format(Mean_Moment[i] -
             (Moment_std_dev[i]/Mean_Moment[i]))) # standard dev - percent

        Moment_Dev_Results_File.write(",{:>20.12E}".format(altitudes[i])) # altitude
        Moment_Dev_Results_File.write(",{:>20.12E}".format(angles[i])) # angle
        Moment_Dev_Results_File.write("\n")

    # total force file

########################################
print("Moments and Forces Found")



# Combining Files into one csv from Chat GPT
# Create an empty dictionary to store DataFrames
dataframes = {}

# Load each CSV file into a DataFrame and store it in the dictionary
for file in csv_files:
    # Use a unique key for each DataFrame, such as the file name without extension
    key = file.split('.')[0]
    dataframes[key] = pd.read_csv(file)

# Create a Pandas Excel writer
with pd.ExcelWriter(output_file, engine='xlsxwriter') as writer:
    # Loop through the DataFrames and write each one to a separate tab
    for sheet_name, dataframe in dataframes.items():
        dataframe.to_excel(writer, sheet_name=sheet_name, index=False)

print("csv files ready")
```

C.2 Plotting Code

```
# Code for plotting SAASHA data

# import statements
from mpl_toolkits import mplot3d
import numpy as np
import matplotlib.pyplot as plt
from matplotlib import cm

from saasha.sim import *
#import pandas as pd
#import csv
from mpl_toolkits.mplot3d import Axes3D

save = False
```

```
title = True
# max_min_avg_sdv = True

# creates a 3D scatter plot
def plot_3D_scatter(X_vals,Y_vals,Z_vals,X_label,Y_label,Z_label,Color):

    x_array = X_vals
    y_array = Y_vals
    z_array = Z_vals

    # Plot setup
    fig = plt.figure()
    ax = fig.add_subplot(111, projection='3d')

    # Create a 3D scatter plot
    ax.scatter(x_array, y_array, z_array, c=Color, marker='o')

    # Set labels for the axes
    ax.set_xlabel(X_label)
    ax.set_ylabel(Y_label)
    ax.set_zlabel(Z_label)

    # Set a title for the plot
    ax.set_title('3D Scatter ' + Y_label + " Vs " + X_label + " Vs " + Z_label)

    # Show the plot
    plt.show()
    return

# creates a 3D contour plot
def plot_3D_contour(X_vals,Y_vals,Z_vals,X_label,Y_label,Z_label,Color):

    fig = plt.figure()

    # Plot set up
    ax = plt.axes(projection='3d')
    ax.contour3D(X_vals,Y_vals , Z_vals, 50, cmap='viridis')
    # Set labels for the axes
    ax.set_xlabel(X_label)
    ax.set_ylabel(Y_label)
    ax.set_zlabel(Z_label)

    # Set a title for the plot
    ax.set_title('3D Contour ' + Y_label + " Vs " + X_label + " Vs " + Z_label)

    # Show the plot
    plt.show()
    return

# creates a 2D contour plot
def plot_2D_contour(X_vals,Y_vals,Z_vals,X_label,Y_label,Z_label,Color):


    #fig = plt.figure()

    h = plt.contourf(X_vals, Y_vals, Z_vals, cmap='viridis')
    plt.axis('scaled')
    plt.colorbar()

    #labels
    plt.xlabel(X_label)
    plt.ylabel(Y_label)
    plt.title('2D Contour' + Y_label + " Vs " + X_label + " Vs " + Z_label)

    plt.show()
    return

# creates a 3D scatter plot, 3D contour plot, and a 2D contour plot
def plot_all(X_vals, Y_vals, Z_vals, X_label, Y_label, Z_label, Color, time_frame, min_path = False,
     max_min_avg_sdv = False, x_min=[], y_min=[], z_min=[]):

    title_f = 17
    tick_f = 17
    space_f = 30
    tick_s = 15
    contour_s = 20

    # 2D Contour Plot
    fig3 = plt.figure()
    h = plt.contourf(X_vals, Y_vals, Z_vals, cmap='viridis')
    #plt.axis('scaled')
    plt.colorbar().ax.tick_params(axis='y', labelsize=contour_s)

    if max_min_avg_sdv:
        #plt.plot(x_min[0], y_min, c="k", label="Min Maximum series",)
        #plt.plot(x_min[1], y_min, c="darkorange", label="Min Minimum series")
        plt.plot(x_min[2], y_min, c="r", label="Min Average Series")
        plt.plot(x_min[3], y_min, c="b", label="Min Std Dev Series")
        plt.legend(fontsize=contour_s)

    plt.xticks(fontsize=contour_s)
    plt.yticks(fontsize=contour_s)

    plt.xlabel(X_label, fontsize=contour_s)
    plt.ylabel(Y_label, fontsize=contour_s)
    if title:
        plt.title('2D Contour, ' + Y_label + " Vs " + X_label + " Vs " + Z_label, fontsize=contour_s,
             pad=15)

    if save:
        plt.savefig(time_frame + '2D Contour ' + Y_label + " Vs " + X_label + " Vs " + Z_label)



    # 3D Contour Plot
    fig2 = plt.figure()


    ax2 = plt.axes(projection='3d')
    ax2.contour3D(X_vals, Y_vals, Z_vals, 50, cmap='viridis')
    # Set labels for the axes

    ax2.tick_params(axis='x', which='both', labelsize=tick_f, pad=tick_s)
    ax2.tick_params(axis='y', which='both', labelsize=tick_f, pad=tick_s)
    ax2.tick_params(axis='z', which='both', labelsize=tick_f, pad=tick_s)

    ax2.set_xlabel(X_label, fontsize=title_f,labelpad=space_f)
    ax2.set_ylabel(Y_label, fontsize=title_f,labelpad=space_f)
    ax2.set_zlabel(Z_label, fontsize=title_f,labelpad=space_f)

    if max_min_avg_sdv:
        #ax2.plot3D(x_min[0], y_min, z_min[0], c="k", label="Min Maximum Series",)
        #ax2.plot3D(x_min[1], y_min, z_min[1], c="darkorange", label="Min Minimum Series")
        ax2.plot3D(x_min[2], y_min, z_min[2], c="r", label="Min Average Series")
        ax2.plot3D(x_min[3], y_min, z_min[3], c="b", label="Min Std Dev Series")
        ax2.legend(fontsize=title_f)

    # Set a title for the plot
    if title:
        ax2.set_title('3D Contour, ' + Y_label + " Vs " + X_label + " Vs " + Z_label,
            fontsize=title_f)

    if save:
        plt.savefig(time_frame + '3D Contour ' + Y_label + " Vs " + X_label + " Vs " + Z_label)



    # 3D Scatter Plot
    fig1 = plt.figure()
    ax1 = fig1.add_subplot(111, projection='3d')

    # Create a 3D scatter plot
    #ax1.scatter(x_array, y_array, z_array, c=Color, marker='o')

    ax1.scatter(X_vals, Y_vals, Z_vals, cmap='viridis', marker='o')

    #if min_path:
        #ax1.scatter(x_min, y_min, z_min, c=Color, label="Min Average Moment Series", marker='o')
        #ax1.legend()

    #if max_min_avg_sdv:
        #ax1.scatter(x_min[0], y_min, z_min[0], c="k", label="Minimum Average", marker='o')
        #ax1.scatter(x_min[1], y_min, z_min[1], c="g", label="Minimum Average", marker='o')
    #ax1.scatter(x_min[2], y_min, z_min[2], c="r", label="Minimum Average", marker='o')
        #ax1.scatter(x_min[3], y_min, z_min[3], c="m", label="Minimum Average", marker='o')
    if max_min_avg_sdv:
        #ax1.plot3D(x_min[0], y_min, z_min[0], c="k", label="Min Maximum Series",)
        #ax1.plot3D(x_min[1], y_min, z_min[1], c="darkorange", label="Min Minimum Series")
        ax1.plot3D(x_min[2], y_min, z_min[2], c="r", label="Min Average Series")
        ax1.plot3D(x_min[3], y_min, z_min[3], c="b", label="Min Std Dev Series")
        ax1.legend(fontsize=title_f)


    ax1.tick_params(axis='x', which='both', labelsize=tick_f, pad=tick_s)
    ax1.tick_params(axis='y', which='both', labelsize=tick_f, pad=tick_s)
    ax1.tick_params(axis='z', which='both', labelsize=tick_f, pad=tick_s)

    # Set labels for the axes
    ax1.set_xlabel(X_label, fontsize=title_f,labelpad=space_f)
    ax1.set_ylabel(Y_label, fontsize=title_f,labelpad=space_f)
    ax1.set_zlabel(Z_label, fontsize=title_f,labelpad=space_f)

    # Set a title for the plot
    if title:
        ax1.set_title('3D Scatter, ' +Y_label + " Vs " + X_label + " Vs " + Z_label, fontsize=title_f)

    if save:
        plt.savefig(time_frame + '3D Scatter ' + Y_label + " Vs " + X_label + " Vs " + Z_label)


    # Show the plots
    plt.show()
    return

    return

# Creates a csv file of a 2x2 matrix
def CSV_File_Data(File_Name, Print_Data=False):
    Data = np.genfromtxt(File_Name, delimiter=",", skip_header=1)
    if Print_Data:
        print(Data)
    return Data

# Plots all minimum force and moment values
```

```
def All_Plot(Time_Stamp_Version):

    time_frame = Time_Stamp_Version

    # file names to import data from
    FMA_Altitude_File = time_frame + "FMA_Altitude.csv"
    FMA_Time_File = time_frame + "FMA_Time.csv"
    FMA_Force_Angles_File = time_frame + "FMA_F_Angle.csv"
    FMA_Forces_File = time_frame + "FMA_Force.csv"
    FMA_Moment_Angles_File = time_frame + "FMA_M_Angle.csv"
    FMA_Moments_File = time_frame + "FMA_Moment.csv"

    # Imported/extracted data
    Min_FMA_Altitude = CSV_File_Data(FMA_Altitude_File)
    Min_FMA_Time = CSV_File_Data(FMA_Time_File)
    Min_FMA_Min_Force_Angle = CSV_File_Data(FMA_Force_Angles_File)
    Min_FMA_Min_Force = CSV_File_Data(FMA_Forces_File)
    Min_FMA_Min_Moment_Angle = CSV_File_Data(FMA_Moment_Angles_File)
    Min_FMA_Min_Moment = CSV_File_Data(FMA_Moments_File)

    #if plot_everything:
    # Plot everything
    plot_all(Min_FMA_Time, Min_FMA_Altitude, Min_FMA_Min_Force_Angle, "Time[sec]", "Altitude[ft]",
         "Min F Bank Angle[deg]", "b",time_frame)
    plot_all(Min_FMA_Time, Min_FMA_Altitude, Min_FMA_Min_Force, "Time[sec]", "Altitude[ft]",
         "Force[lbf]", "b",time_frame)
    plot_all(Min_FMA_Time, Min_FMA_Altitude, Min_FMA_Min_Moment_Angle, "Time[sec]", "Altitude[ft]",
         "Min M Bank Angle[deg]", "b",time_frame)
    plot_all(Min_FMA_Time, Min_FMA_Altitude, Min_FMA_Min_Moment, "Time[sec]", "Altitude[ft]",
         "Moment[lbf-ft]", "b",time_frame)

# Plots difference between the data
def All_Plot_Difference(Time_Stamp_Version, Time_Stamp_Version_2):

    time_frame = Time_Stamp_Version
    time_frame_2 = Time_Stamp_Version_2

    # Data 1 filenames
    FMA_Altitude_File = time_frame + "FMA_Altitude.csv"
    FMA_Time_File = time_frame + "FMA_Time.csv"
    FMA_Force_Angles_File = time_frame + "FMA_F_Angle.csv"
    FMA_Forces_File = time_frame + "FMA_Force.csv"
    FMA_Moment_Angles_File = time_frame + "FMA_M_Angle.csv"
    FMA_Moments_File = time_frame + "FMA_Moment.csv"

    # Lists of imported file data from Data 1
    Min_FMA_Altitude = CSV_File_Data(FMA_Altitude_File)
    Min_FMA_Time = CSV_File_Data(FMA_Time_File)
    Min_FMA_Min_Force_Angle = CSV_File_Data(FMA_Force_Angles_File)
    Min_FMA_Min_Force = CSV_File_Data(FMA_Forces_File)
    Min_FMA_Min_Moment_Angle = CSV_File_Data(FMA_Moment_Angles_File)
    Min_FMA_Min_Moment = CSV_File_Data(FMA_Moments_File)

    # Data 2

    # Data 2 filenames
    FMA_Altitude_File_2 = time_frame_2 + "FMA_Altitude.csv"
    FMA_Time_File_2 = time_frame_2 + "FMA_Time.csv"
    FMA_Force_Angles_File_2 = time_frame_2 + "FMA_F_Angle.csv"
    FMA_Forces_File_2 = time_frame_2 + "FMA_Force.csv"
    FMA_Moment_Angles_File_2 = time_frame_2 + "FMA_M_Angle.csv"
    FMA_Moments_File_2 = time_frame_2 + "FMA_Moment.csv"

    # Lists of imported file data from Data 2
    Min_FMA_Altitude_2 = CSV_File_Data(FMA_Altitude_File_2)
    Min_FMA_Time_2 = CSV_File_Data(FMA_Time_File_2)
    Min_FMA_Min_Force_Angle_2 = CSV_File_Data(FMA_Force_Angles_File_2)
    Min_FMA_Min_Force_2 = CSV_File_Data(FMA_Forces_File_2)
    Min_FMA_Min_Moment_Angle_2 = CSV_File_Data(FMA_Moment_Angles_File_2)
    Min_FMA_Min_Moment_2 = CSV_File_Data(FMA_Moments_File_2)


    # difference in forces and angles
    Min_FMA_Min_Force_Angle_3 = Min_FMA_Min_Force_Angle_2 - Min_FMA_Min_Force_Angle
    Min_FMA_Min_Force_3 = Min_FMA_Min_Force_2 - Min_FMA_Min_Force
    Min_FMA_Min_Moment_Angle_3 = Min_FMA_Min_Moment_Angle_2 - Min_FMA_Min_Moment_Angle
    Min_FMA_Min_Moment_3 = Min_FMA_Min_Moment_2 - Min_FMA_Min_Moment

    # Check the difference in data
    print(Min_FMA_Min_Force_Angle_3)
    print(Min_FMA_Min_Force_3)
    print(Min_FMA_Min_Moment_Angle_3)
    print(Min_FMA_Min_Moment_3)

    #if plot_everything:
    # Plot all the data accordingly
    plot_all(Min_FMA_Time, Min_FMA_Altitude, Min_FMA_Min_Force_Angle_3, "Time[sec]", "Altitude[ft]",
         "Force Bank Angle[deg]", "b","360-180 case")
    plot_all(Min_FMA_Time, Min_FMA_Altitude, Min_FMA_Min_Force_3, "Time[sec]", "Altitude[ft]",
         "Force[lbf]", "b", "b","360-180 case")
    plot_all(Min_FMA_Time, Min_FMA_Altitude, Min_FMA_Min_Moment_Angle_3, "Time[sec]", "Altitude[ft]",
         "Moment Bank Angle[deg]", "b","360-180 case")
    plot_all(Min_FMA_Time, Min_FMA_Altitude, Min_FMA_Min_Moment_3, "Time[sec]", "Altitude[ft]",
         "Moment[lbf-ft]", "b","360-180 case")

# Plots all Maximum, Minimum, Average, and Standard deviation values
def All_Plot_2(Time_Stamp_Version, Plot_Max_Min_F=False, Plot_Max_Min_M=False, min_path=False):

    time_frame = Time_Stamp_Version

    # Filenames for the altitude
    FDA_Altitude_File = time_frame + "FDA_Altitude.csv"
    FDA_Angles_File = time_frame + "FDA_Angles.csv"

    # Filenames for the forces
    FDA_Max_Force_File = time_frame + "FDA_Max_Force.csv"
    FDA_Min_Force_File = time_frame + "FDA_Min_Force.csv"
    FDA_AVG_Force_File = time_frame + "FDA_AVG_Force.csv"
    FDA_StdDev_Force_File = time_frame + "FDA_StdDev_F.csv"

    # Filenames for the moments
    FDA_Max_Moment_File = time_frame + "FDA_Max_Moment.csv"
    FDA_Min_Moment_File = time_frame + "FDA_Min_Moment.csv"
    FDA_AVG_Moment_File = time_frame + "FDA_AVG_Moment.csv"
    FDA_StdDev_Moment_File = time_frame + "FDA_StdDev_M.csv"

    # Imported file data
    FDA_Altitude = CSV_File_Data(FDA_Altitude_File)
    FDA_Angle = CSV_File_Data(FDA_Angles_File)

    # Imported force data
    FDA_Max_Force = CSV_File_Data(FDA_Max_Force_File)
    FDA_Min_Force = CSV_File_Data(FDA_Min_Force_File)
    FDA_AVG_Force = CSV_File_Data(FDA_AVG_Force_File)
    FDA_StdDev_Force = CSV_File_Data(FDA_StdDev_Force_File)

    # Imported moment datd
    FDA_Max_Moment = CSV_File_Data(FDA_Max_Moment_File)
    FDA_Min_Moment = CSV_File_Data(FDA_Min_Moment_File)
    FDA_AVG_Moment = CSV_File_Data(FDA_AVG_Moment_File)
```

```
FDA_StdDev_Moment = CSV_File_Data(FDA_StdDev_Moment_File)

# Results filename
F_Results_File = time_frame + "Min_F_Dev_Results.csv"
M_Results_File = time_frame + "Min_M_Dev_Results.csv"

# Results to CSV
F_Results = CSV_File_Data(F_Results_File)
M_Results = CSV_File_Data(M_Results_File)

# controls to the code
min_path = True
Plot_Max_Min_F=False
Plot_Max_Min_M=False
max_min_avg_sdv = True

# Forces
min_f_alt = F_Results[:,0]

# Max Force angles and values
min_f_max_angle = F_Results[:,1]
min_f_max = F_Results[:,2]

# Min Force angles and values
min_f_min_angle = F_Results[:,3]
min_f_min = F_Results[:,4]

# Avg Force angles and values
min_f_avg_angle = F_Results[:,5]
min_f_avg = F_Results[:,6]

# Std Dev Force angles and values
min_f_dev_angle = F_Results[:,7]
min_f_dev = F_Results[:,8]


# Moments
min_m_alt = M_Results[:,0]

# Max Moment angles and values
min_m_max_angle = M_Results[:,1]
min_m_max = M_Results[:,2]

# Min Moment angles and values
min_m_min_angle = M_Results[:,3]
min_m_min = M_Results[:,4]

# Avg Moment angles and values
min_m_avg_angle = M_Results[:,5]
min_m_avg = M_Results[:,6]

# Std Dev angles and values
min_m_dev_angle = M_Results[:,7]
min_m_dev = M_Results[:,8]
#if plot_everything:
# Plot everything


# Optimum paths for the minimum max, min, avg, and std dev
if max_min_avg_sdv:

    # Forces

    # minimum avg force angles, altitudes, and average force
    min_f_avg_angle = [F_Results[:,1], F_Results[:,6], F_Results[:,11], F_Results[:,16]]
```

```
        min_f_alt = F_Results[:,0]
        min_f_avg = [F_Results[:,4], F_Results[:,9], F_Results[:,14], F_Results[:,19]]

        # minimum force standard devation
        min_f_dev_angle = [F_Results[:,1], F_Results[:,6], F_Results[:,11], F_Results[:,16]]
        min_f_alt = F_Results[:,0]
        min_f_dev = [F_Results[:,5], F_Results[:,10], F_Results[:,15], F_Results[:,20]]

        # Moments

        # minimum avg moment angles, altitudes, and average force
        min_m_avg_angle = [M_Results[:,1], M_Results[:,6], M_Results[:,11], M_Results[:,16]]
        min_m_alt = M_Results[:,0]
        min_m_avg =[M_Results[:,4], M_Results[:,9], M_Results[:,14], M_Results[:,19]]

        # minimum moment standard devation
        min_m_dev_angle = [M_Results[:,1], M_Results[:,6], M_Results[:,11], M_Results[:,16]]
        min_m_alt = M_Results[:,0]
        min_m_dev = [M_Results[:,5], M_Results[:,10], M_Results[:,15], M_Results[:,20]]

    # Plots Max and min forces
    if Plot_Max_Min_F:
        plot_all(FDA_Angle, FDA_Altitude, FDA_Max_Force, "Bank_Angle[sec]", "Altitude[ft]", "Maximum
             Force[lbf]", "r",time_frame, min_path, max_min_avg_sdv, min_f_max_angle, min_f_alt ,
             min_f_max)

        plot_all(FDA_Angle, FDA_Altitude, FDA_Min_Force, "Bank_Angle[sec]", "Altitude[ft]", "Minimum
             Force[lbf]", "r",time_frame, min_path, max_min_avg_sdv, min_f_min_angle, min_f_alt,
             min_f_min)

    # Plots average and standard dev of the force
    plot_all(FDA_Angle, FDA_Altitude, FDA_AVG_Force, "Bank_Angle[sec]", "Altitude[ft]", "Average
         Force[lbf]", "r",time_frame, min_path, max_min_avg_sdv, min_f_avg_angle, min_f_alt,
         min_f_avg)

    plot_all(FDA_Angle, FDA_Altitude, FDA_StdDev_Force, "Bank_Angle[sec]", "Altitude[ft]", "Force
         Standard Deviation[lbf]", "r",time_frame, min_path, max_min_avg_sdv, min_f_dev_angle,
         min_f_alt, min_f_dev)


    # Plots max and min moment
    if Plot_Max_Min_M:
        plot_all(FDA_Angle, FDA_Altitude, FDA_Max_Moment, "Bank_Angle[sec]", "Altitude[ft]", "Maximum
             Moment[lbf-ft]", "r",time_frame, min_path, max_min_avg_sdv, min_m_max_angle, min_m_alt,
             min_m_max)

        plot_all(FDA_Angle, FDA_Altitude, FDA_Min_Moment, "Bank_Angle[sec]", "Altitude[ft]", "Minimum
             Moment[lbf-ft]", "r",time_frame, min_path, max_min_avg_sdv, min_m_min_angle, min_m_alt,
             min_m_min)

    # Plots average and standard dev of the moment
    plot_all(FDA_Angle, FDA_Altitude, FDA_AVG_Moment, "Bank_Angle[sec]", "Altitude[ft]", "Average
         Moment[lbf-ft]", "r",time_frame, min_path, max_min_avg_sdv, min_m_avg_angle, min_m_alt,
         min_m_avg)

    plot_all(FDA_Angle, FDA_Altitude, FDA_StdDev_Moment, "Bank_Angle[sec]", "Altitude[ft]", "Moment
         Standard Deviation[lbf-ft]","r",time_frame, min_path, max_min_avg_sdv, min_m_dev_angle,
         min_m_alt, min_m_dev)




if __name__=="__main__":

    # 360 degree data
    time_frame = "0_60_19_"
    #All_Plot(time_frame)

    # 180 degree data
    time_frame_2 = "0_60_18_"
    All_Plot(time_frame_2)

    # 180 degree data updated
    time_frame_3 = "0_60_20_"

    #All_Plot_Difference(time_frame, time_frame_2)

    # Plots the altitude vs angle vs max, min, avg, and sdv
    #All_Plot_2(time_frame_2)

    #All_Plot_2(time_frame_3)
    '''
    plot_everything = False

    FMA_Altitude_File = time_frame + "FMA_Altitude.csv"
    FMA_Time_File = time_frame + "FMA_Time.csv"
    FMA_Force_Angles_File = time_frame + "FMA_F_Angle.csv"
    FMA_Forces_File = time_frame + "FMA_Force.csv"
    FMA_Moment_Angles_File = time_frame + "FMA_M_Angle.csv"
    FMA_Moments_File = time_frame + "FMA_Moment.csv"

    Min_FMA_Altitude = CSV_File_Data(FMA_Altitude_File)
    Min_FMA_Time = CSV_File_Data(FMA_Time_File)
    Min_FMA_Min_Force_Angle = CSV_File_Data(FMA_Force_Angles_File)
    Min_FMA_Min_Force = CSV_File_Data(FMA_Forces_File)
    Min_FMA_Min_Moment_Angle = CSV_File_Data(FMA_Moment_Angles_File)
    Min_FMA_Min_Moment = CSV_File_Data(FMA_Moments_File)

    if plot_everything:
        # Plot everything
        plot_all(Min_FMA_Time,Min_FMA_Altitude,Min_FMA_Min_Force_Angle,"Time[sec]","Altitude[ft]",\n
                "Force_Angle[deg]","b")
        plot_all(Min_FMA_Time,Min_FMA_Altitude,Min_FMA_Min_Force,"Time[sec]","Altitude[ft]",\n
                "Force[lbf]","b")
        plot_all(Min_FMA_Time,Min_FMA_Altitude,Min_FMA_Min_Moment_Angle,"Time[sec]","Altitude[ft]",\n
                "Moment_Angle[deg]","b")
        plot_all(Min_FMA_Time,Min_FMA_Altitude,Min_FMA_Min_Moment,"Time[sec]","Altitude[ft]",\n
                "Moment[lbf-ft]","b")
    '''
```

C.3 Final Plot Code

```
import numpy as np
from saasha.sim import *
import pandas as pd
import csv
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
# elementwise root sum square of three vectors of equal length
def rt_sum_sqr(x,y,z):
    f = np.zeros(len(x))
    for i in range(len(x)):
        f[i] = (x[i]**2 + y[i]**2 + z[i]**2)**0.5
    f=np.array(f)
    return f
```

```
if __name__=="__main__":

    # Base file we are starting from
    filename = "input_2.json"

    time_span = "10_0_10_"

    output_file_x = time_span + "land_final_state_hist_"
    output_file = output_file_x + "combined.xlsx"

    Force_Results_File = time_span + "land_F_Results_f.csv"
    Moment_Results_File = time_span + "land_M_Results_f.csv"

    Forces = []
    Moments = []
    Time_Values = []
    csv_files = []
    csv_files.append(Force_Results_File)
    csv_files.append(Moment_Results_File)

    json_string = open(filename).read()
    input_dict = json.loads(json_string)

    for i in range(1, 6, 1):
        # Specify the file path where you want to save the JSON data
        # filename_new = "landing_input_" + str(i) + ".json"
        input_filename = "landing_input_x.json"

        # csv output file name
        states_filename = time_span + "land_state_hist_" + str(i) + ".csv"

        # Update the output file name
        input_dict["simulation"]["states_filename"] = states_filename

        # Update the landing input
        input_dict["trajectory"]["path"]["filepath"] = "trajectory_final_" + str(i) +".csv"

        # Open the file to save the json data
        with open(input_filename, "w") as json_file:
            # Use the json.dump() function to write the data to the file
            json.dump(input_dict, json_file)

        # Start of saasha simulation
        print("sim " + str(i) + " Started")
        # Initialize sim
        print("\nReading input file...")
        mysim = sim(input_filename)
        print("Done")
        # Initialize State
        y = mysim.initialize_state()
        # Run simulation
        mysim.run(y)
        print("sim " + str(i) + " Finished")

        # List of file paths to your CSV files
        csv_files.append(states_filename)

        # file to open and analyze
        fn = states_filename

        print("Reading forces from file ", fn)
        State_Data = np.genfromtxt(fn, delimiter=",", skip_header=1) # get the data from the file
        # print(State_Data)
        # print(State_Data[1,20])
        # print(State_Data[1,21])
        # print(State_Data[1,22])
        Time_Values_x = State_Data[:, 0]

        Total_Force = rt_sum_sqr(State_Data[:, 20], State_Data[:, 21], State_Data[:, 22]) # calculate
             the total forces
        Total_Moment = rt_sum_sqr(State_Data[:, 23], State_Data[:, 24],
                                  State_Data[:, 25]) # calculate the total moments

        # print("################################################################################")
        # print(Total_Force)
        # print("################################################################################")

        Time_Values.append(State_Data[:, 0])
        Forces.append(Total_Force)
        Moments.append(Total_Moment)

    Time_Values = np.array(Time_Values)
    Forces = np.array(Forces)

    print(Forces)

    Moments = np.array(Moments)

    # Record Force Results csv
    with open(Force_Results_File, 'w') as Force_Dev_Results_File:

        Force_Dev_Results_File.write("Time[sec], 0 deg Force[lbf], 90 deg Force[lbf], Avg Force[lbf],
             StdDev Force[lbf], C Map Force[lbf]")
        Force_Dev_Results_File.write("\n")

        for i in range(len(Time_Values_x)):
            Force_Dev_Results_File.write("{:>20.12E}".format(Time_Values_x[i])) # altitude
            Force_Dev_Results_File.write(",{:>20.12E}".format(Forces[0][i])) # angle
            Force_Dev_Results_File.write(",{:>20.12E}".format(Forces[1][i])) # max force
            Force_Dev_Results_File.write(",{:>20.12E}".format(Forces[2][i])) # min force
            Force_Dev_Results_File.write(",{:>20.12E}".format(Forces[3][i])) # avg force
            Force_Dev_Results_File.write(",{:>20.12E}".format(Forces[4][i])) # force std dev

            Force_Dev_Results_File.write("\n")

    # Record Force Results csv
    with open(Moment_Results_File, 'w') as Moment_Dev_Results_File:

        Moment_Dev_Results_File.write("Time[sec], 0 deg Moments[lbf-ft], 90 deg Moments[lbf-ft], Avg
             Moments[lbf-ft], StdDev Moments[lbf-ft], C Map Moments[lbf-ft]")
        Moment_Dev_Results_File.write("\n")

        for i in range(len(Time_Values_x)):
            Moment_Dev_Results_File.write("{:>20.12E}".format(Time_Values_x[i])) # altitude
            Moment_Dev_Results_File.write(",{:>20.12E}".format(Moments[0][i])) # angle
            Moment_Dev_Results_File.write(",{:>20.12E}".format(Moments[1][i])) # max force
            Moment_Dev_Results_File.write(",{:>20.12E}".format(Moments[2][i])) # min force
            Moment_Dev_Results_File.write(",{:>20.12E}".format(Moments[3][i])) # avg force
            Moment_Dev_Results_File.write(",{:>20.12E}".format(Moments[4][i])) # force std dev

            Moment_Dev_Results_File.write("\n")




    # Combining Files into one csv from Chat GPT
    # Create an empty dictionary to store DataFrames
    dataframes = {}

    # Load each CSV file into a DataFrame and store it in the dictionary
    for file in csv_files:
        # Use a unique key for each DataFrame, such as the file name without extension
        key = file.split('.')[0]
        dataframes[key] = pd.read_csv(file)

    # Create a Pandas Excel writer
    with pd.ExcelWriter(output_file, engine='xlsxwriter') as writer:
        # Loop through the DataFrames and write each one to a separate tab
        for sheet_name, dataframe in dataframes.items():
            dataframe.to_excel(writer, sheet_name=sheet_name, index=False)

    print("csv files ready")
```