Aerodynamic Optimization of Integrated Wing-Engine Geometry Using an Unstructured Vorticity Solver

by

Logan King

A thesis submitted to the Graduate Faculty of
Auburn University
in partial fulfillment of the
requirements for the Degree of
Master of Science

Auburn, Alabama
August 01, 2015

Keywords: Optimization, Engine Integration, Unstructured Mesh, Vorticity Solver

Copyright 2015 by Logan King

Approved by

Roy Hartfield, Chair, Walt and Virginia Waltosz Professor of Aerospace Engineering
Brian Thurow, Professor of Aerospace Engineering
John Burkhalter, Professor Emeritus of Aerospace Engineering

Abstract

A high fidelity surface vorticity solver in conjunction with an optimizer is used to find optimal solutions for engine integration. The case study for this paper is an aerodynamically optimized engine configuration added to the DLR-F4 from the AIAA CFD drag prediction workshop. The unstructured meshes generated during the optimization process are produced by Open Vehicle Sketch Pad. The SwarmOps particle swarm optimization algorithm is used within Phoenix Integration ModelCenter during the optimization process. A purely aerodynamically focused optimization resulted in a wing-engine geometry with the engine being moved forward and down away from the lower surface of the wing as well as the engine being placed at a maximum outboard location near the wing tip. A wing spar structural analysis tool and vertical stabilizer sizing model were added to this optimization to attain a more realistic optimal design. The result of this optimization revealed that the benefits to the ratio of lift over drag attained by moving the engines outboard were greatly overshadowed by the increased weight of the wing spar and added weight and drag of an ever increasing vertical stabilizer size.

Acknowledgments

I would like to thank Dr. Ahuja for his assistance with Flightstream, Dr. Hartfield for his direction and revisions, and Amanda for her time and support.

Table of Contents

List of Figures

List of Tables

List of Abbreviations

δ_R	Rudder Deflection Angle
$\dot{m}$	Mass Flow Rate
Γ	Dihedral
Λ	Sweep
Λ_{TE}	Trailing Edge Sweep
$\overrightarrow{g}$	Global Best Known Position
$\overrightarrow{p}$	Local Best Known Position
$\overrightarrow{v_n}$	Particle Velocity
$\overrightarrow{x_n}$	Particle Position
ϕ_g	Global Attraction Weight
ϕ_p	Local Attraction Weight
ρ	Density
σ	Bending Stress
τ	Shear Stress
τ_R	Rudder Angle of Attack Effectiveness
r_n	Random Number Weight
A	Area

b	Span
C	Chord Length
C_D	Coefficient of Drag
C_L	Coefficient of Lift
$C_{n\delta R}$	Rudder Control Derivative
D	Drag
d	Diameter
F	Thrust or Force
g	Acceleration Due to Gravity
I	Moment of Inertia
L	Lift
l_V	Vertical Stabilizer Moment Arm
LS_x	Airfoil Lower Surface Location at x
m	Mass
p	Pressure
PC	Projected Chord Length as Viewed from Above
q	Dynamic Pressure
R_x	Specific Gas Constant
S	Reference Area
T	Temperature

t	Time
US_x	Airfoil Upper Surface Location at x
V	Velocity
V_V	Tail Volume Coefficient
W	Weight
x	Position in the X Axis
y	Position in the Y Axis
z	Position in the Z Axis

Chapter 1

Introduction

Engine integration is an area of high current interest. Many variables involving aerodynamics, structural demands, weight, and cost are typical. Traditionally, after significant preliminary design work, dozens of potential design configurations are tested in wind tunnels until a satisfactory configuration is identified. However, such an approach is expensive and time intensive due to the high number of variables which makes finding a near optimal design before any construction or wind tunnel testing begins extremely important. The Navier-Stokes equations can be used to accurately describe the motion of a viscous, compressible fluid which makes these equations ideal for determining the aerodynamic forces on a aircraft configuration. However, due to the complexity of the three dimensional Navier-Stokes equations, most of these problems cannot be solved analytically.

Over the past few decades Navier-Stokes equations have been incorporated in computational fluid dynamics (CFD) codes and are solved numerically to provide flow solutions. These CFD flow solvers have the potential to provide high fidelity results for nearly any flight regime. The main two disadvantages of CFD are long computation times when compared to other methods, and the reliance of highly refined volumetric meshes to ensure accuracy and convergence. As computer computational power increases, these long CFD run times become more manageable; however, mesh generation is still often a very hands-on procedure that is difficult to automate with good results. This mesh automation is essential during the optimization process and is one of the challenges of attaining accurate solutions from any flow solver. Although computers have become increasingly more powerful with time, optimization problems can have hundreds of design variables. Run time increases with each added variable to the optimization problem which is only exasperated by longer CFD computation

times. Designers are often forced to either reduce the design space or endure optimization run times that can take months to finish. Despite these limitations, Navier-Stokes based CFD codes are nearly universally used in the aerospace industry today. Historically, when computing power was more limited or nonexistent, assumptions or simplifications were made to the Navier-Stokes equations so that flow solutions could be found.

By assuming that the flow is incompressible and inviscid, Prandtl's lifting line theory can be used to solve for the lift and induced drag for a wing given the geometry of the wing and the flight conditions. The Kutta-Joukowski theorem relates the lift on a two dimensional section of the wing to the circulation around the local airfoil. Lifting line theory extends this concept to a wing by creating a bound vortex, extending along the entire span of the wing, along with perpendicular trailing vortices to create a horseshoe vortex. The spanwise change in circulation along the wing is directly related to the spanwise change in lift. By assuming a circulation distribution along the wing and associated shed, the local lift per unit span can be found using the KuttaJoukowski theorem [1]. By integrating along the entire wing, the total lift and induced drag on a wing can be reduced to simple equations consisting of the freestream velocity, density and coefficients related to the wing geometry [1]. These quick algebraic equations would be well suited for use in an optimization problem; however, lifting line theory only gives accurate answers for unswept quarter chord lines with a moderate to high aspect ratio [1]. Unfortunately this means that lifting line theory would be unable to accurately determine lift or drag for a wide array of wing and wing-body configurations.

Methods developed to solve this problem include the lifting-surface theory and the vortex lattice numerical method. While the lifting line theory supposes only one lifting line going across the wing, lifting-surface theory models an infinite number of lifting lines which cover the entire wing to form a vortex sheet [1]. As in other methods, this total normal velocity component of the sheet must be zero [1]. The advantage of the lifting-surface theory over the lifting line theory is the ability to accurately calculate the lift and induced drag with a minimum of unknowns. The downside to the lifting-surface theory is that the formulation

of the series of equations that comprise this method are not easily found for most geometric configurations.

A similar, but often more practical method is the vortex lattice numerical method.The vortex lattice method discretizes the wing mean camber surface into a finite grid of horseshoe vortices and control points [2]. The strength and location of all the horseshoe vortices create an induced normal velocity at any given control point on the wing. As with the lifting-surface theory, the total normal velocity component at any given control point must be zero [2]. This creates a system of linear algebraic equations which can be solved using a computer. For increased resolution, a higher number of horseshoe vortices and control points must be used. With computers being able to solve such problems at higher speeds, engineers have been able to develop software that uses the vortex lattice method with considerable success to analyze finite wings with high resolution.

One of the first widely used generalized vortex lattice based softwares was VORLAX. VORLAX was developed by Lockheed-California Company for NASA in the 1970's [3]. VORLAX was able to accurately approximate the surface pressure distributions, aerodynamic forces, and moment coefficients for a very wide range of rigid aerodynamic bodies in inviscid subsonic or supersonic flows [3]. Thickness effects of the airfoils were simulated using two lattices, one for the top surface and one for the bottom surface, as long as these two surfaces were not too close to each other as is necessary at the trailing edge of the wing[3]. Fuselages were crudely simulated with either a vortex lattice planer surface or polygonal prism, with varying cross sections [3]. The flat panels which made up the wings and body of the aircraft contained a skewed-horseshoe vortex system with constant sweep for any single panel [3]. Each horseshoe vortex consisted of bound vortices along the mean camber surface with unbound trailing legs for the wake[3]. The bound legs followed the surface of the vortex lattice in the longitudinal direction whereas the unbound legs on the trailing edges of the wings went in a direction corresponding to the specified slip angle, ending at the trefftz plane [3].

While VORLAX was a great advancement at the time, it had several limitations. The accuracy of VORLAX is somewhat limited because it numerically solves the potential flow equation for linear, inviscid, and irrotational flow [3]. This equation ignores higher order terms in order to linearize the velocity potential equation and is only valid when dealing with small perturbation (thin bodies at small angles of attack) and entirely subsonic or supersonic flow [1]. This makes VORLAX unable to accurately approximate transonic flow. VORLAX also assumes singularity strength is constant across a panel. This has the advantage of decreased run time at the cost of accuracy. In many situations this does not negatively affect the accuracy of the solution to a large degree as long as the mesh is fine enough; however, this is often more problematic for supersonic problems and can lead to solver instability. Despite these limitations, versions of VORLAX are still available online and it is sometimes used today at the university level in combination with an aerodynamic superposition approach to get semi-accurate preliminary design solutions [5].

In the 1980's a more advanced potential flow solver was created called Panel Aerodynamics (PAN AIR). PAN AIR was created with the goal of solving stability problems that were common in earlier vortex lattice flow solvers for supersonic flow. One of the causes of supersonic stability issues for previous solvers was spurious line-vortex terms at the panel edges [6]. PAN AIR manages to eliminate these errors by creating a higher order panel scheme and the addition of source panels [7]. Where earlier lower order solvers kept singularity strength constant across a panel, PAN AIR allows source strengths to vary linearly over a panel and uses quadratic doublet strength distributions [7]. PAN AIR's higher order scheme solves previous supersonic stability issues by ensuring that doublet strength is continuous across all panels and that all adjacent panels have contiguous edges [7]. PAN AIR also is able to somewhat simulate boundary layer thickness by using either "exact" or "linearized" boundary conditions [7]. With exact boundary conditions the user manually adds a finite thickness to the geometry and assumes flow is tangent to the new surface to simulate the boundary layer whereas with linearized boundary conditions PAN AIR changes panel

source strengths which effectively acts as if a small amount of flow is being emitted from the panel surface to simulate the boundary layer thickness [7]. This panel source strength was assumed and thus could be altered to allow the user to match experimental data. PAN AIR still suffers from the limitations of linearized potential flow in that it can not accurately describe transonic flow [7], but it is able to provide solutions to both subsonic and supersonic flow problems. The program's higher order scheme also has the added benefit of reducing the sensitivity to changes in mesh size and arrangement [6]. The downside to this higher order scheme is a significantly increased run time and difficulties in generating the geometry [6].

Due to the longer run times inherent to higher order schemes, constant strength panel methods were still pursued. For example NASA's VSAERO, the precursor to PMARC. VSAERO and PMARC attempt to better approximate the Navier-Stokes equations by using an iterative viscous/potential flow solver [8]. These programs use potential flow panel methods with iterative wake-shape calculations coupled with a boundary layer approximation [8]. The boundary layer thickness is simulated by introducing a nonzero normal velocity on the panel surfaces is much the same way as PAN AIR [8].

Sensitivity to mesh refinement and layout is a problem inherent to earlier lower order schemes which led some subsonic only flow solvers such as MCAERO to sacrifice faster run times and use continuous quadratic doublets much like PAN AIR [6]. VSAERO and PMARC are able to get around this problem by using internal Dirichlet boundary conditions [8]. With the internal Dirichlet boundary condition, a fictitious flow within the interior of the geometry is modeled with a specified velocity potential [8]. By setting the perturbation velocity within the geometry to zero it was found that the solver was less sensitive to bad panel layouts, possibly due to smaller differences between inner and outer flow velocities [8].

Although PAN AIR, VSAERO, and PMARC are able to better mimic viscous effects than previous solvers, they are still, at their core, potential flow solvers and lack the fidelity of current Navier-Stokes CFD codes. They also have the drawback of being pressure based

solvers, as apposed to vorticity and circulation based solvers such as VORLAX, and thus require highly refined structured meshes in order to properly capture pressure gradients.

The main disadvantage of working with a structured mesh is the need of a much higher mesh size on curved surfaces compared to an unstructured mesh [11]. The U-V mapping which is typically used with structured meshes also requires the solver to define four points for each panel even if one side of the quadrilateral has been collapsed to form a triangular panel [11]. This often results in an unnecessary use of memory when defining the geometry [11]. Another advantage of unstructured meshes is the ability to more easily refine certain areas of the mesh. Due to the lack of an underlying mapping, individual panels can be deleted or refined to increase mesh fidelity [11]. The advantages of unstructured meshes pertaining to mesh size, quality, and flexibility made the use of unstructured meshes a very desirable capability for surface solvers. However, due to the need of defining panel edges as bound or trailing vortices during the calculation of induced loads on local panels, vorticity based solvers were unable to use unstructured meshes which had arbitrary panel orientation [11].

Pressure based flow solvers also suffer from poor pressure gradient resolution in the event that the mesh has any bumps or dents. In addition to mesh fidelity concerns, the attempts made by these solvers to more closely replicate viscous effects such as boundary layer thickness and flow separation caused a large increase to solver run time compared to previous potential flow solvers and thus diminished the main advantage of using them over a Navier-Stokes CFD code in an optimization problem.

For some design problems that exist only in the subsonic stable flow, such as an aircraft flying at cruise speed, viscous effects have a minuscule effect on the flow solution outside the boundary layer. Therefore it is often appropriate in these cases to ignore these viscous effects as modeled by some pressures solvers, e.g. PAN AIR, VSAERO and PMARC, in favor of decreased runtime and solver stability provided by vorticity and circulation based panel solvers. Athena Vortex Lattice (AVL) and Tornado are two such examples of vorticity

based surface solvers that are widely used today [9] [10]. Both of these surface solvers allow the user to easily define an aircraft shape and solve for aerodynamic forces and moments on the body [9] [10]. However these vorticity based flow solvers, much like previous vorticity solvers, are only compatible with structured meshes.

In 2013 Flightstream became the first vorticity based solver in the open literature able to overcome the unstructured mesh challenge for vorticity based solutions [11]. Flightstream is able to do this by using an application of vortex rings on a triangular panel. Within the solver, vortex filaments are placed on the perimeter of each facet of the mesh to form vortex loops [11]. By using an application of the Biot-Savart law, it is possible to determine the induced velocity on a point in space from the segment of the vortex loop located on one of the edges of a panel [11]. This same evaluation can then be done for the other two sides of the triangular facet to determine the total induced velocity at a point in space from the entire vortex loop located on the panel [11]. The vorticity strength can be found by applying the Von Neumann boundary condition at control points on the sub-panel surface element [11]. After the vorticity strengths are determined, the induced velocity at any point in space may be found.

Instead of using a structured wake sheet as was the case with most previous vorticity solvers, Flightstream optionally uses a relaxed wake strand model [11]. After trailing edges are marked either manually or with the solver, Flightstream creates wake strands which emanate from the nodes of the trailing edges [11]. The solver is then able to calculate the vorticity strengths of these wake strands by analyzing contributing panel edges and determining whether they have additive or diminishing effects on the vorticity strengths of the wake strands [11]. The contributing panel edges' classifications are decided by the panels' orientation relative to the node [11]. The net vorticity of the geometry is propagated downstream by the wake strands with the only decrease to vorticity strengths is due to viscous effects [11]. Flightstream numerically propagates wake strands downstream with varying vorticity strength in an attempt to simulate viscous effects in the wake. The size of

the pseudo-time steps used for the numerical propagation along each individual step of a wake strand is unique to that strand segment [11]. This allows the solver to change the local step sized based on flow conditions while still propagating all wake strands to a user designated trefftz plane [11]. The flexibility of the relaxed wake strand model allows Flightstream to more accurately describe the propagation of filaments downstream from lifting surfaces which makes it possible to capture the interaction between different segments of geometry such as a wing and the horizontal stabilizer.

Previous vorticity surface solvers used an application of the KuttaJoukowski equations to determine lift from the knowledge of vortex strengths and flow conditions; however, this is not possible with unstructured meshes due to the arbitrary orientation of the panels [11]. Flightstream is able to bypass this obstacle by using a method to calculate shed vorticity in which panel orientation is irrelevant.

Although Flightstream has the ability to generate simplistic geometry components, the program's capability and flexibility of three dimensional modeling is fairly limited compared to stand alone CAD programs. This would normally make it difficult to accurately model a geometry for use in the program's flow solver, but due to the unstructured nature of the solver, any number of outside CAD programs can be used for mesh creation. Flightstream can also be run with macros which allow the user to bypass the GUI during the flow evaluation process. The program's capacity for automation and flexibility granted by its use of unstructured meshes make Flightstream a viable tool for design optimization problems.

Typically Navier-Stokes based CFD solvers are used for the problem of engine integration; however these solvers are computationally very expensive in addition to requiring highly refined volumetric meshes. While Flightstream is unable to give accurate flow solutions for transonic or supersonic cases, engine integration problems often apply only to subsonic flow in which Flightstream can evaluate the aerodynamic loads on an aircraft with high fidelity.

The faster run time and use of less refined, unstructured, surface meshes inherent to Flightstream has the potential to allow an optimizer to explore a much wider search space in a smaller amount of time compared to a CFD solver.

Chapter 2

Methodology

The DLR-F4, a subsonic transport aircraft designed for the AIAA CFD drag prediction workshop, was selected to serve as the baseline for the engine integration optimization problem. The DLR-F4 was selected because, due to its original purpose as a model to test the validity of CFD solvers, the exact dimensions of the geometry are well defined and available on the NASA archives [12] [13] which allows for more accurate mesh creation. The coefficients of lift and drag were also evaluated at various angles of attack in a wind tunnel by Reference [13] which makes it possible to determine the validity of the baseline model used in Flightstream.

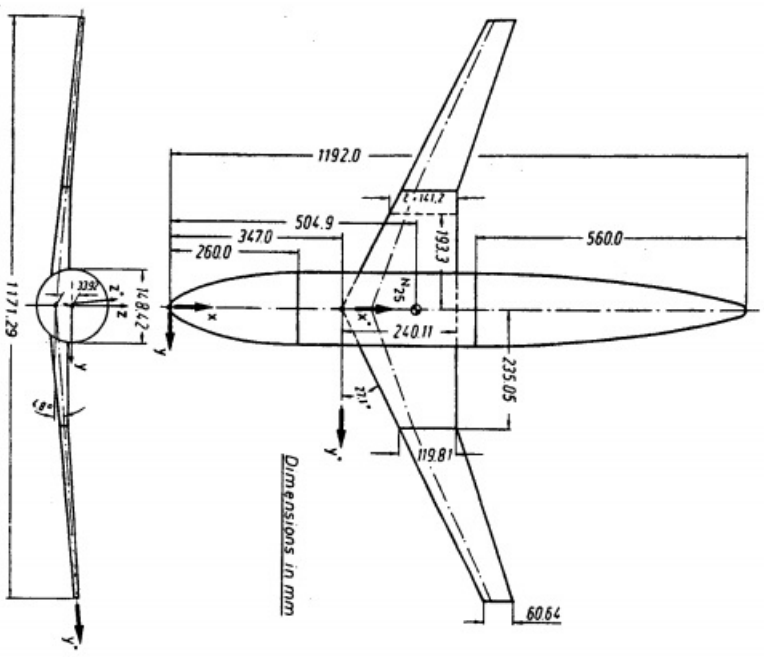


Figure 2.1: DLR-F4 Schematic [13]

2.0.1 DLR-F4 Baseline Mesh Creation

Although Flightstream has some CAD capabilities, the primary purpose of the program is that of a flow solver, and as such it is somewhat limited in its ability to produce a complex geometry. Therefore it was determined that in order to more easily create an accurate DLR-F4 geometry, an outside CAD program was necessary. The NASA open source CAD program Vehicle Sketch Pad (VSP) was selected for the process of geometry and mesh creation. VSP was chosen because it was developed with the creation of aircraft geometries in mind. Components of an aircraft, such as the wing and fuselage, are defined within VSP by a set of parameters that the user can easily change to drastically alter the shape of the aircraft. Fuselage cross sections and wing airfoils can be very accurately recreated within VSP by importing text files which define the outer mold line with a series of points. After all of the components of the are made within the program, VSP can be used to join these components and produce an unstructured mesh which is compatible with Flightstream.

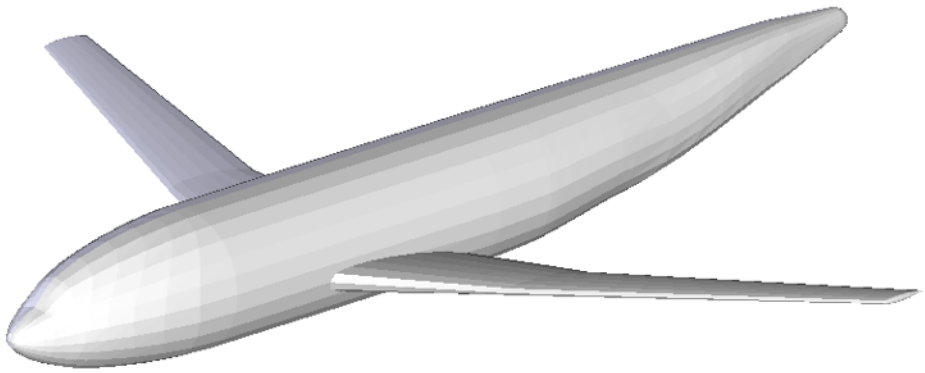


Figure 2.2: DLR-F4 Isometric View

Originally it was attempted to exactly match the fuselage cross sections and airfoils by using the profile data defined by Reference [13], but the inclusion of the entirety of the defined points within the VSP model proved to be to be too computationally demanding. Therefore it was necessary to simplify the profile data in order to increase the speed of

geometry creation. Fortunately an accurate model of the DLR-F4 with simplified fuselage cross section definition was found on the VSP Hangar [14].

The number of interpolated cross sections for the wing of the aircraft was increased to produce a finer mesh across the wing. Lower quality panels are often generated by VSP at the intersection of different components of the aircraft geometry such as the wing fuselage intersection. Flightstream can still produce accurate flow solutions despite the presence of lower quality panels but mesh quality is more critical near the trailing edge of lifting surfaces. Therefore it was necessary to manually correct panels at the intersection of the fuselage and wing within Flightstream. After the baseline model was completed, coefficients of lift and drag were obtained using Flightstream over a range of angle of attacks. The results were then compared to the wind tunnel data from Reference [13].

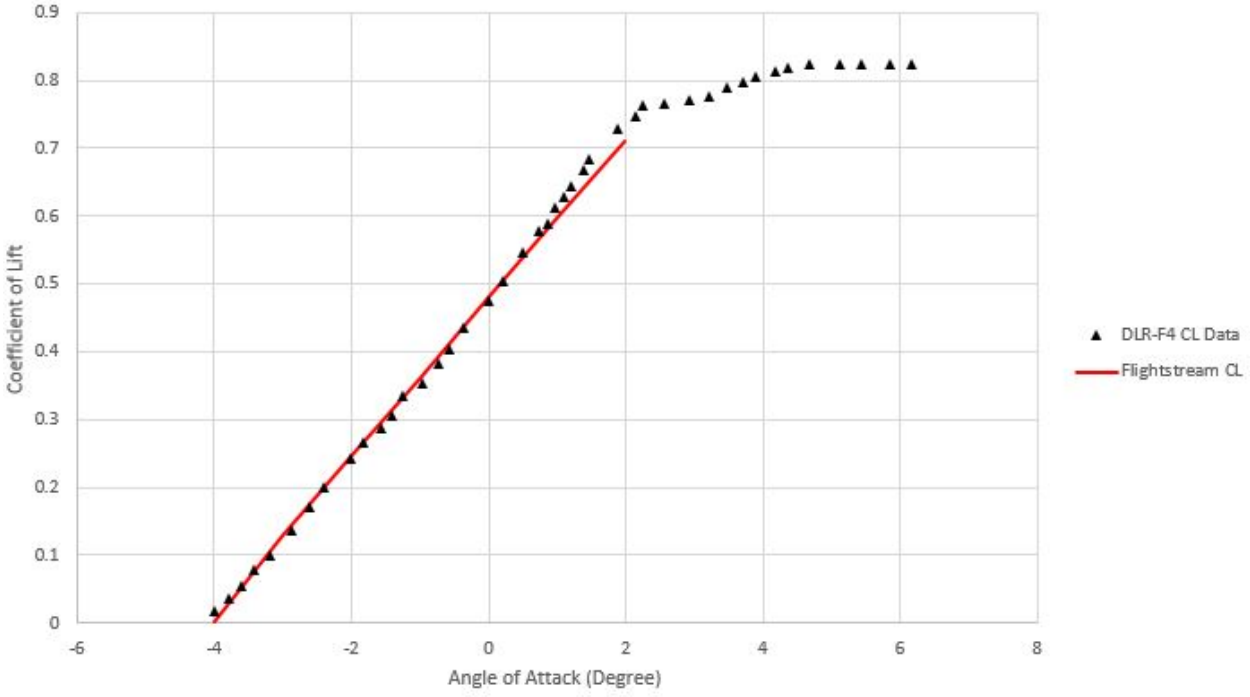


Figure 2.3: DLR-F4 CL vs Angle of Attack, Mach=.75

As can be seen in Figure 2.3, the coefficient of lift values attained with Flightstream for the VSP DLR-F4 mesh closely matched the wind tunnel data from Reference [13]. Near an angle of attack near two degrees there was flow separation on the wing. Flightstream,

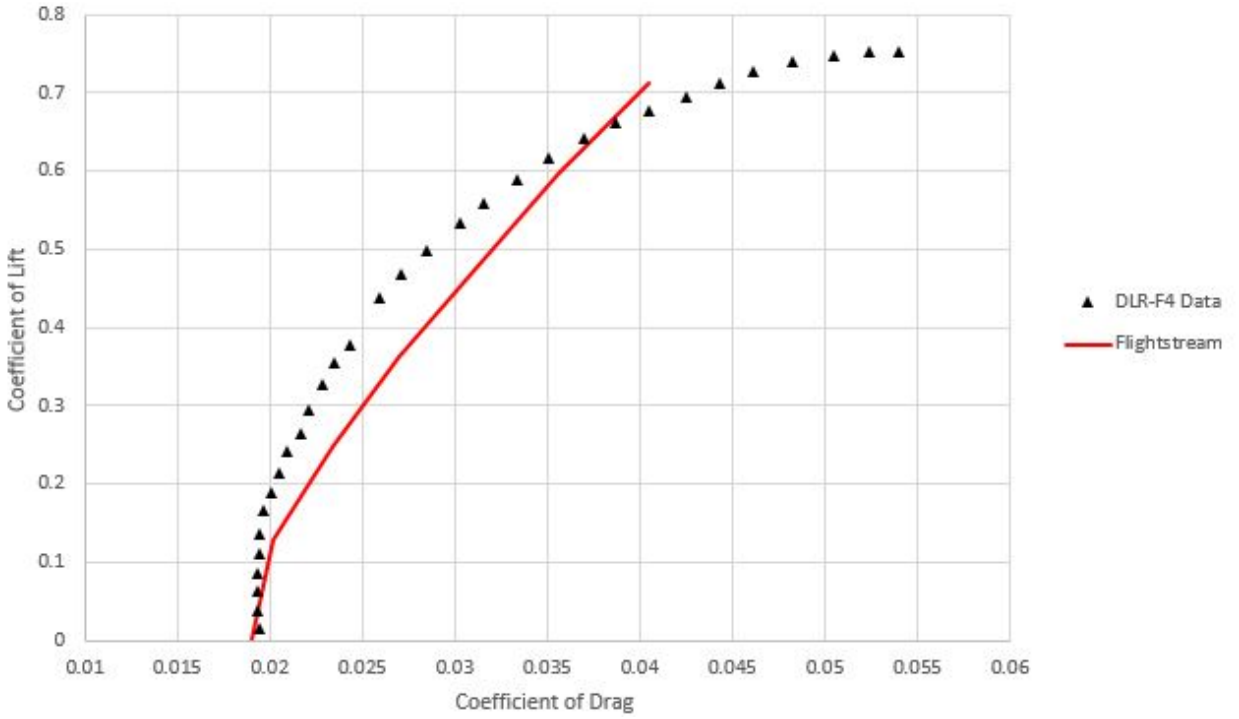


Figure 2.4: DLR-F4 Drag Polar, Mach=.75

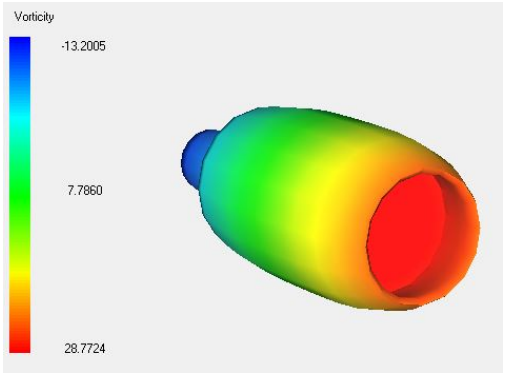
as a vorticity based solver, can not determine flow separation location, therefore any results attained over an angle of attack of two degrees were omitted from Figure 2.3.

Flightstream is able to predict drag at low Mach numbers but accuracy is decreased when compressibility effects are more prominent. Flightstream uses a semi-empirical evaluation for skin friction drag but the total drag is not accurately captured at high subsonic Mach numbers [11]. This is an area where improvement in Flightstream is desired. For the current version of Flightstream, the computed drag polar for the DLR-F4 is shown in Figure 2.4 along with experimental wind tunnel data. With the completion of a baseline geometry, it is now possible to create an engine and pylon for the engine integration.

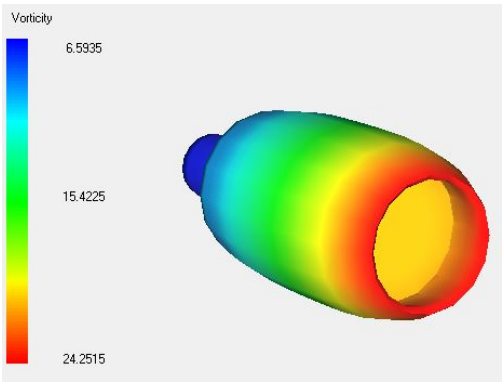
2.0.2 Engine Modeling

A high bypass turbofan engine geometry was modeled within VSP to be added to the DLR-F4 model. The engine intake was modeled down to the compressor blades which were treated as a flat, solid compressor face. The engine bypass exit and nozzle exit were also

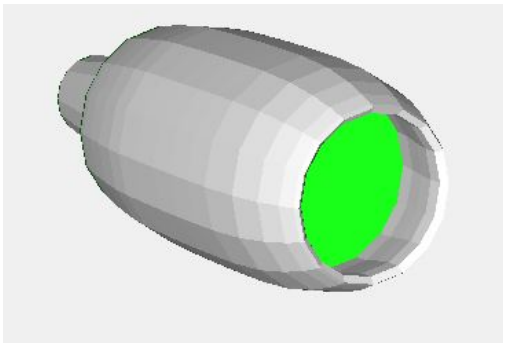
modeled as solid boundaries. Within Flightstream a Von Neumann boundary condition is applied to facets which ensures that no flow penetrates the panels. In order to model the flow through the engine, relaxed Von Neumann boundary conditions were applied to the surfaces of the compressor face, bypass exit and nozzle exit. This relaxed Von Neumann boundary specifies a flow velocity through the panels that make up these boundaries. A similar turbofan engine was analyzed using NPSS by Reference [15]. The flow conditions specified for the engine at the compressor, bypass and nozzle exit from Reference [15] were matched within the engine model in Flightstream.



(a) Engine with no Flow Modeled



(b) Engine with Flow Modeled



(c) Compressor Flow Region

Figure 2.5: Flightstream Engine Flow Solution

The thrust of this engine can be found by calculating the time rate of change of momentum from the flow entering the engine to the flow exiting the engine as well as the force

created by the pressure difference at the exit.

$$F = \frac{m_{out}V_{out} - m_{in}V_{in}}{\Delta t} + (p_e - p_a)A_e \tag{2.1}$$

This can be rewritten in terms of mass flow rate as seen below.

$$F = (\dot{m_{out}}V_{out}) - (\dot{m_{in}}V_{in}) + (p_e - p_a)A_e \tag{2.2}$$

$$\dot{m} = \rho AV \tag{2.3}$$

The mass flow rate coming into the engine is equal to the mass flow rate going through the compressor whereas the mass flow rate leaving the engine is composed of the mass flow rate exiting the nozzle and engine bypass. The velocity at these locations is known from Reference [15] and areas can be measured from the engine model in VSP. Density is not directly known at these engine locations but it can be found using the known pressure and temperature values from Reference [15] and the equation of state.

$$p = \rho R_{air} T \tag{2.4}$$

The flow properties within the engine are treated as constants for a given free stream velocity, regardless of engine location; therefore, the thrust will remain constant as long as the cross sectional area of the compressor, bypass exit, and nozzle exit remain unchanged.

2.0.3 Engine Pylon

With the baseline aircraft configuration and engine modeled, it was necessary to connect the engine to the wing with a pylon. The pylon was modeled within VSP as a symmetric vertical wing component extending from the engine to the lower surface of the DLR-F4 wing. One of the challenges with the pylon creation was making the pylon capable of changing shapes and locations as the engine was shifted relative to the wing. This was

done by defining the pylon with four coplanar points as seen in Figure 2.6 and Figure 2.7. By dictating that the bottom of the pylon had a constant attachment point to the engine, the position of the pylon could be attained from the location of the engine. Although the position of the engine and pylon could be variable, the relative position between the two components was a known constant value.

$$x_{pylon} = x_{engine} + \Delta_x \tag{2.5}$$

$$z_{pylon} = z_{engine} + \Delta_z \tag{2.6}$$

$$y_{pylon} = y_{engine} \tag{2.7}$$

These three equations give the origin location of pylon which coincides with the bottom left corner of the pylon in Figure 2.7 . With the pylon placed, it is still necessary to size and shape the pylon accordingly so that it properly interfaces with the lower surface of the wing. This geometry is developed by adjusting the leading edge sweep, spans, and chords of the three sections in VSP that make up the pylon. Before this interface can be built, it is necessary to define the shape of the pylon with the four points seen in Figure 2.6 and Figure 2.7.

When joining different geometry components in VSP, it is necessary to have at least part of the two components occupying the same space. This ensures that one mesh comprised of the two components is created by VSP instead of two non interlocking meshes which happen to share a surface. In the latter case its is possible to create nearly coplanar panels which can lead to solver divergence in Flightstream. Therefore section 1 and section 3 of the pylon, seen in Figure 2.7, are made to partially intersect the engine and wing respectably. This geometry intersection ensures a cohesive, interlocking mesh between the three aircraft components. Point 1 in Figure, 2.7 represents the location that the pylon intersects the outer surface of the engine and can be determined from the pylon and engine location as well as the outer diameter of the engine at that location engine. Due to the positional relationship

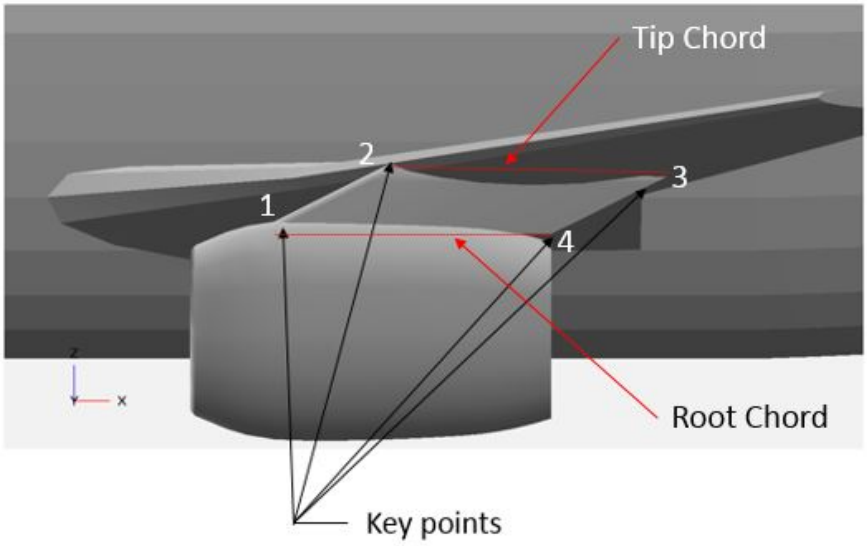


Figure 2.6: Pylon Side View

between the pylon and engine, as well as the engine shape being constant, the lower span of the pylon is also a constant value.

$$span_1 = .5D_{engine} - \Delta_z \tag{2.8}$$

$$x_1 = x_{pylon} \tag{2.9}$$

$$y_1 = y_{pylon} \tag{2.10}$$

$$z_1 = z_{pylon} + span_1 \tag{2.11}$$

The lowest most point on the trailing edge of the pylon was defined by point 4. The z coordinate of this point is already determined by the position of the pylon. The x coordinate of the pylon is determined by the user given root chord of the pylon, referred to Chord 1 in Figure 2.7.

$$x_4 = x_{pylon} + C_1 \tag{2.12}$$

$$y_4 = y_{pylon} \tag{2.13}$$

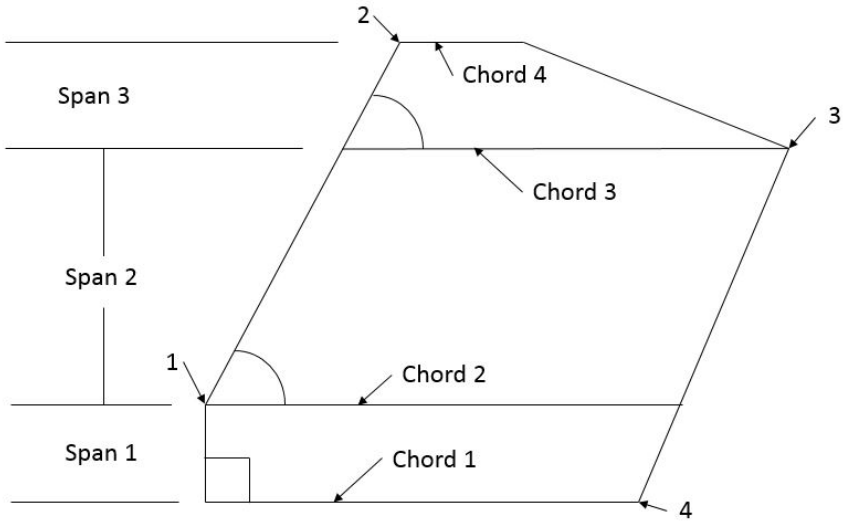


Figure 2.7: Pylon Diagram

$$z_4 = z_{pylon} \tag{2.14}$$

The upper most point on the leading edge of the pylon is defined as point 2. The x and z coordinates of point 2 are found by referencing the local airfoil of the wing that is intersected by the pylon y plane. The x coordinate of point 2 is defined as being a constant small percentage of the local airfoil projected chord aft of the leading edge of the wing. The z coordinate is defined as being the mid way point between the upper and lower surface of the airfoil at that x location. Once again, the reasoning for the location of point 2 to be within the wing as opposed to on the outer surface was to ensure that VSP created an interlocking mesh between the pylon and the wing. The length of Chord 4, as seen in Figure 2.7 is unimportant as the upper surface of the pylon is hidden within the wing and is not created during the meshing process. The only matter of importance pertaining to the length of Chord 4 is ensuring that no part of the pylon extends through the upper surface of the wing. This was achieved by making the length of Chord 4 be a small constant percentage of the local wing airfoil projected chord.

Point 3 was used to define the trailing edge sweep of the pylon and was once again determined from the local airfoil of the wing intersected by the pylon y plane. The trailing edge sweep of the pylon was a desired variable in the pylon creation process. In order to achieve this, the x coordinate of point 3 was defined by a variable percentage of the local airfoil projected chord with the z coordinate being placed at ten percent between the upper and lower surface of the local airfoil at that x location. Before point 2 or point 3 could be determined, the local shape and location of the airfoil of the wing had to be found.

The DLR-F4 wing shape is defined in Reference [12] and Reference [13] by local airfoil profiles at 4 different span-wise locations along the wing as can be seen in Figure 2.8. Reference [12] provides x and z coordinates for points on the upper and lower surface of the four defining airfoils. These profiles were used to produce the wing geometry in VSP for the baseline DLR-F4 model. VSP shapes the wing cross sections between these defined airfoils by linearly interpolating between these defined airfoils.

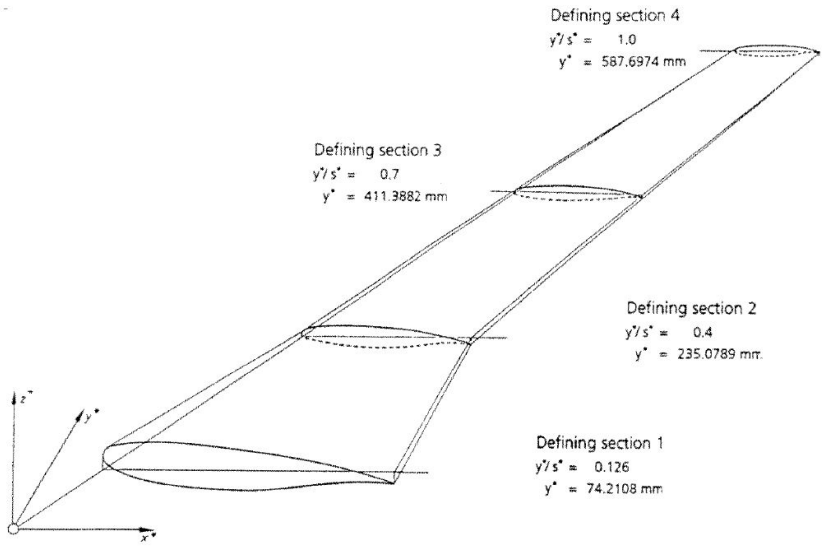


Figure 2.8: DLR-F4 Wing Definition [13]

In order to determine the shape and location for any cross section along the wing, the linear interpolation for the wing done by VSP had to be replicated externally. This was done by taking the data from each defined cross section and shifting the x coordinates so that they all had the same starting location. After this was done the x and z coordinates of

the data were normalized to ensure that each set of airfoil data had a range of 1 for the x axis. These four cross sections could then be used to define a surface which could be used with the built in Matlab function named "griddata" to calculate a linearly interpolated cross section shape at any point along the wing. This airfoil shape could then be scaled back up to normal size using the linearly interpolated projected chord length of the airfoil at that location and multiplying that value by the x and z coordinates which defined the local wing cross section shape. After scaling was complete the size and shape of the wing cross section created by any given y value along the wing was known however position information still needed to be determined.

The position information, relative to the starting location of the wing, for a cross section of the wing could be found using the wing sweep and dihedral information provided for the DLR-F4 in Reference [13]. The DLR-F4 wing has a constant wing sweep and dihedral angle which allows the following equations to define the starting location for any cross section along the wing.

$$x_{airfoil} = y_{airfoil} \tan(\Lambda_{wing}) + x_{wing} \tag{2.15}$$

$$y_{airfoil} = y_{pylon} \tag{2.16}$$

$$z_{airfoil} = y_{airfoil} \tan(\Gamma_{wing}) + z_{wing} \tag{2.17}$$

With the size shape and location of the local airfoil interfacing with the pylon, all of the necessary information is available to determine the coordinates of points 2 and 3 in Figure 2.7. K_1 is a constant value defining the starting location of point 2 as a percentage of the projected chord of the local airfoil of the wing. K_2 is a user defined variable defining the starting location of point 3 as a percentage of the projected chord of the local cross section of the wing.

$$x_2 = k_1 PC_{airfoil} + x_{airfoil} \tag{2.18}$$

$$y_2 = y_{pylon} \tag{2.19}$$

$$z_2 = .5(US_{x_2} - LS_{x_2}) + LS_{x_2} + z_{airfoil} \tag{2.20}$$

$$x_3 = k_2 PC_{airfoil} + x_{airfoil} \tag{2.21}$$

$$y_3 = y_{pylon} \tag{2.22}$$

$$z_3 = .1(US_{x_3} - LS_{x_3}) + LS_{x_3} + z_{airfoil} \tag{2.23}$$

After points 1 through 4 have been found, the location and shape of the pylon is fully defined, but to create the pylon, this information must be changed into a format that VSP can accept. Due to the pylon being created as a wing component in VSP, the shape of each section of they pylon is defined by a given span, tip chord, root chord, and leading edge sweep. The span of the lowest section of the pylon, Span 1, is already known from equation 2.8 and the root chord of the first section of the pylon, Chord 1, is a user given value and therefore is also know. The leading edge sweep of the first pylon section is zero, as can be seen from Figure 2.7. Although the trailing edge sweep is not needed to define the pylon section in VSP, this value must be determined to find the tip chord of the first pylon section, Chord 2. Since the pylon has a constant trailing edge sweep angle from point 4 to point 3, the trailing edge sweep for the first pylon section can be found by finding the trailing edge sweep angle from point 4 to point 3.

$$\Lambda_{TE} = \arctan[(x_3 - x_4)/(z_3 - z_4)] \tag{2.24}$$

$$C_2 = C_1 + span_1 \tan(\Lambda_{TE}) \tag{2.25}$$

The span of the second pylon section, Span 2 can be found using the z coordinates of points 3 and 1.

$$span_2 = z_3 - z_1 \tag{2.26}$$

Before the tip chord of the second section of the pylon can be determined, the leading edge sweep of the pylon section needs to be calculated. As seen in Figure 2.7 the leading edge sweep for the pylon is constant for the top two sections of the pylon. This allows the sweep for the second section of the pylon to be determined by calculating the sweep angle between point 1 and point 2.

$$\Lambda = \arctan[(x_2 - x_1)/(z_2 - z_1)] \tag{2.27}$$

$$C_3 = (x_3 - x_1) - span_2 \tan(\Lambda) \tag{2.28}$$

The leading edge sweep of the third section of the pylon is the same as the leading edge sweep of the second section of the pylon and thus is already known from equation 2.27. The span the of the third section of the pylon, Span 3, can be determined using the coordinates from points 2 and 3.

$$span_3 = z_2 - z_3 \tag{2.29}$$

Finally, the last necessary value needed for the pylon is the tip chord of the third section of the pylon, Chord 4. As stated previously, this value is not overly important as long as the chord is short enough so as to not protrude through the upper surface of the wing. Therefore, this chord length was made to be a small percentage of the projected chord of the local wing cross section that the pylon intersects.

$$C_4 = .2PC_{airfoil} \tag{2.30}$$

With the addition of Chord 4, all necessary vsp inputs have been determined to produce an engine pylon that can adjust shape and location to property interface with the wing and engine while allowing the engine to change location. The last step is ensuring that a

consistent pylon mesh is produced as they pylon changes shape. As with the other geometry components such as the wing and fuselage, mesh refinement can be controlled within VSP by changing the number of interpolated cross sections for each section of the component.

$$Width_{panel} = span/(NumberofInterpolatedCrossSections - 1) \tag{2.31}$$

To ensure that the pylon had consistent panel sizes between configurations with different span lengths, the average panel size on the wing of the aircraft was measured and used to determine a specified panel width along the span of the pylon. Because the engine pylon was an area of focus the panel width used on the wing was halved for the pylon. The reason for having a variable number of interpolated cross sections for the pylon was to create a near constant panel size between pylon configurations. This procedure is an attempt to eliminate mesh quality as a potential variable during an design process.

Chapter 3

Program Automation and Optimization

3.0.4 Vehicle Sketch Pad Automation

The previous chapter outlined the development of the baseline DLR-F4 geometry and engine and pylon modeling; however before any engine integration optimization could begin, an optimization pipeline would have to be developed and the process of mesh creation and fitness evaluation would have to be automated. Vehicle Sketch Pad can be used to create a specified geometry and mesh. For this engine optimization problem the geometry of the fuselage and wing of the DLR-F4 as well as the engine geometry will be constant while the engine location and pylon shape and location will be variable. For a given engine location, the equations in the previous chapter showed a method for finding the values of the necessary VSP inputs needed to create a desired aircraft configuration. After these values are found they can be entered manually into the VSP model to produce the mesh for that aircraft configuration. This process can be automated through the use of a design file within VSP.

Design files are created within VSP by selecting the variable inputs such as the ones used to control engine position on the X Y and Z axis and the inputs used to control the shape of the pylon such as span length, chord length, and leading edge sweep of the different sections of the pylon. After the variable inputs are selected, VSP writes a text file with each variable being identified by a unique identification code. After this design file is created it can be read by an outside program such as Matlab. After it is read by Matlab, the input variable values can be changed to correspond to a new aircraft configuration and then a new design file can be written to reflect the changes. Loading this new design file into the VSP baseline file will change the previous geometry to the desired new design. Finally a script can be written for VSP to create a mesh from this geometry and export it to a desired location.

After a new design file is created the new geometry mesh creation can be automated by creating a batch file which opens VSP, selects the baseline VSP file, choses the new design file, and runs the VSP script that creates and exports the new mesh.

3.0.5 Flightstream Automation

With the mesh generation fully automated it was now necessary to automate the flow solver, Flightstream. Before this can begin a baseline Flightstream file must be created which contains much of the solver settings. Within this file solver settings such as flow velocity, angle of attack, solver convergence criteria, and reference area and length are defined. After this file is created, manual interaction with the Flightstream graphical user interface can be avoided by using a macro. The macro can load the previously generated Flightstream file as well as the newly created mesh. Flightstream will then find the lift and drag produced by that geometry with the given solver and flow settings. This information is then output by Flightstream as a text file which can be read and used as part of an objective function for an optimizer. Other information and actions can be stored within the Flightstream macro such as specifying the unit used for the geometry and whether or not a symmetry plane should be used in the case of a symmetric half mesh.

The ability of Flightstream to work with half meshes can be extremely advantageous when preforming repeated runs as is the case with an optimization application. Using a half mesh with a symmetric plane cuts the mesh size in half which decreases the necessary run time dramatically without losing solver accuracy. Another feature used in Flightstream is the ability to combine boarding triangular panels to form quadrilateral panels. Not only does this process decrease the mesh size and run time, but it can serve to produce a more stable flow solution which can converge more quickly and accurately. The reason for this is the accuracy of vorticity based surface solvers can be negatively effected by having panel edges nearby the centroids of opposing panels. This situation most typically occurs near thin trailing edges of the wing in meshes produced by VSP. This is due to the upper surface

of the wing mesh being composed of triangular panels which have diagonals in a different direction than the panels which make up the lower surface of the wing. When Flightstream converts much of the triangular panel mesh into quadrilateral panels, these opposing diagonal edges between neighboring panels are removed to form unified quadrilateral panels along the trailing edge of the wing.

The Flightstream macro file can also load physics files which specify special flow regions which have boundary conditions other than the typical Von Neumann boundary condition. Two such examples are trailing edges of the aircraft wings and engine inlet and exits. Specifying a region as being a trailing edge causes Flightstream to create vortex filaments which extend from nodes along the trailing edge [11]. These trailing edge vortices are necessary in calculating the lift produced by a geometry due to the solver's integrated circulation loop method [11]. Flightstream has a built-in trailing edge identification function but it fails to ignore trailing edges which are not of interest for lift generation calculations such as the trailing edge of the engine pylon. Therefore it was necessary to create a trailing edge physics file for the DLR-F4 wing so as to avoid unwanted trailing edge vortices which would have increased solver run time.

Engine inlet and exit flow boundary conditions were discussed earlier along with engine modeling. The engine inlet boundary conditions can specify a flow velocity which penetrates the panel unlike the Von Neumann boundary condition which specifies that no flow goes through the surface panels. After these flow regions are defined within Flightstream on an engine mesh, a physics file can be produced which saves the designated flow velocity as well as the coordinates of the panels which make up the inlet boundary. However, in the case of an optimization problem where the engine location is variable, the coordinates of the panels making up the engine inlet boundaries are also variable and dependent of the location of the engine. Therefore if the engine was moved, these flow regions would have to be redefined manually within Flightstream. This process was avoided by creating an engine mesh in VSP from the turbofan geometry and moving it to the aircraft origin defined

in VSP. After it was brought into Flightstream the flow regions of the compressor, bypass exit and nozzle exit were marked and given the flow velocities outlined in Reference [15] to create a physics file. This text file can then be read in by Matlab which allows the engine boundary coordinates to be altered to reflect a new engine location. This is done by adding the X Y and Z coordinates of the new engine location defined in VSP to the preexisting panel coordinates in the physics file produced from the same engine mesh at the VSP origin. After the flow boundary panel coordinates are changed, the physics file can be rewritten with Matlab to reflect the new engine location. This new physics file is then specified within the Flightstream macro so as to avoid any manual interface with the Flightstream GUI. With VSP and Flightstream fully automated, they could now be used within an optimizer and wrapper to solve an aerodynamic optimization problem.

3.0.6 Phoenix Integration ModelCenter

The equations and discussions in the previous chapters showed how a new engine pylon configuration could be created with the DLR-F4 given inputs defining an engine location and pylon shape. It was also shown that the process of mesh creation in VSP and flow evaluations in Flightstream could be automated. However, a wrapper was still needed to properly communicate between these programs so that information and files could be passed between VSP, Matlab and Flightstream. ModelCenter, by Phoenix Integration, was chosen as the program to provide this service. Within ModelCenter a user can build an object oriented framework to solve a design problem. These components can run outside programs such as VSP and Flightstream as well as pass input and output variables between the different programs. ModelCenter can also use previously written Matlab scripts to determine the necessary inputs for other components such as VSP. Finally ModelCenter's Optimization Toolkit can be used to drive this process with one of the many optimizers included with ModelCenter. This optimization process as applied to the DLR-F4 engine integration problem can be seen in the Figure 3.1.

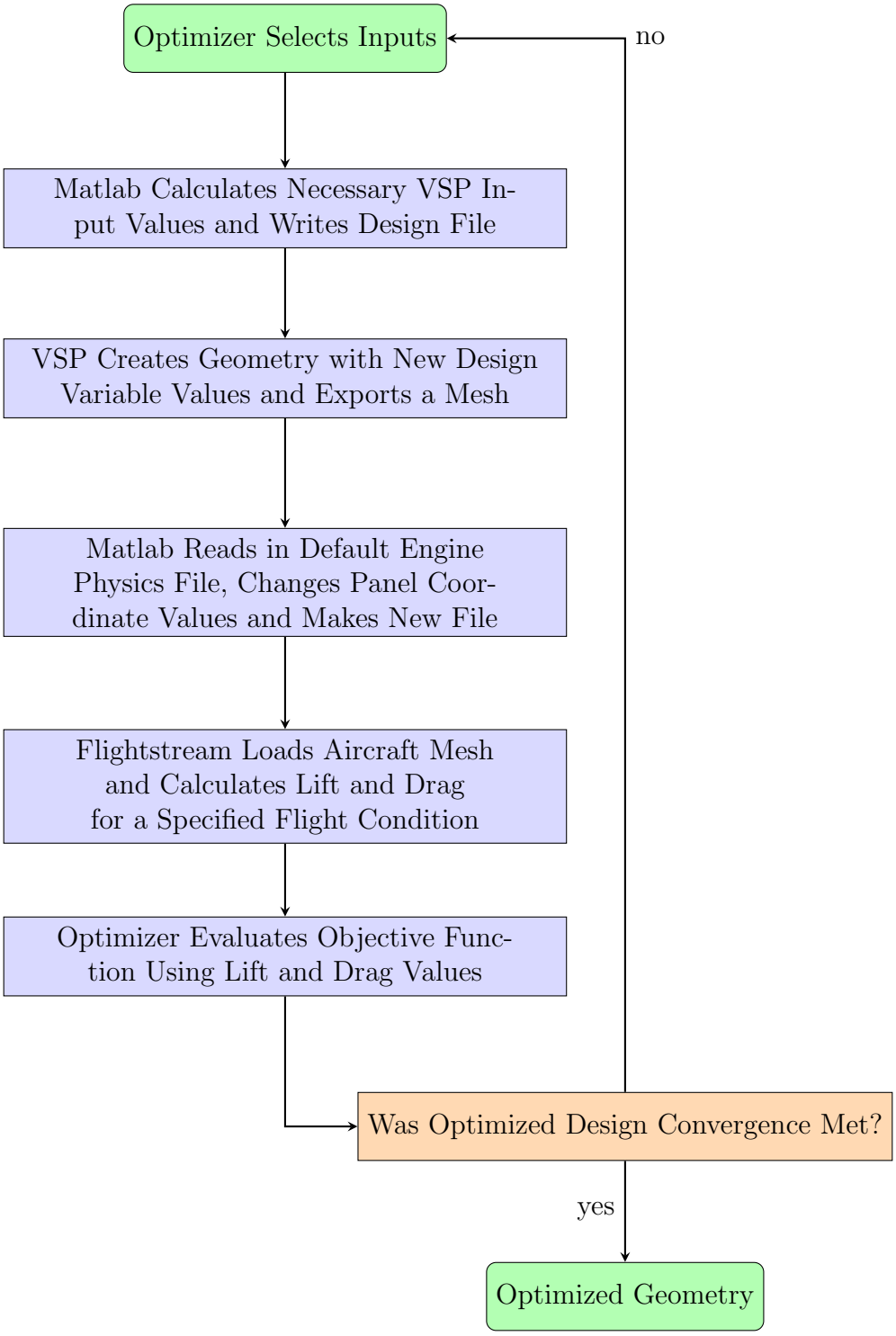


Figure 3.1: Optimization Flow Chart

This optimization process outlined in Figure 3.1 can be used to find an engine pylon configuration for the DLR-F4 which maximizes lift over drag. However, before this optimization

process could be used within ModelCenter to solve a design problem, an optimizer had to be selected. For this engine integration problem the ModelCenter included, classic particle swarm optimizer called SwarmOps (PSO) was selected. This optimizer was chosen because it does a good job of searching noisy solution spaces compared to some other optimization techniques such as the gradient based methods which can become stuck in local minimum or maximum locations within the search space. Particle swarm optimization (PSO) was first developed in 1997 by Kennedy and Eberhart as a way to create an optimization process that can mimic how information is found and shared in a society [17]. With PSO a population of particles is created which explores the search space [17]. The search of each particle is influenced by the particle's previously found local best design as determined by the fitness function as well as the best design found globally by any particle [17].

$$\overrightarrow{v_{n+1}} = \omega\overrightarrow{v_n} + \phi_p r_p(\overrightarrow{p} - \overrightarrow{x_n}) + \phi_g r_g(\overrightarrow{g} - \overrightarrow{x_n}) \tag{3.1}$$

$$\overrightarrow{x_{n+1}} = \overrightarrow{x_n} + \overrightarrow{v_{n+1}} \tag{3.2}$$

The velocity with which each particle travels through the search space is directed by user defined inputs which can create greater influence due to either the local or global best position within the domain found so far by the population of particles [16]; however, the optimizer domain is constrained to insure that no particles travel outside of the user defined search space. The optimization parameters used for the engine integration design problem are shown in the following table.

ω	ϕ_p	ϕ_g	Swarm Size
.729	1.49445	1.49445	50

Table 3.1: Optimization Parameters

Chapter 4

Aerodynamic Optimization

With the optimization method selected, ModelCenter could now be used to solve a DLR-F4 engine integration optimization problem. The design problem chosen was to maximize lift over drag generated from the DLR-F4 in a cruise situation with the addition of a turbofan engine and pylon. The aircraft velocity was set to Mach .75 with a an angle of attack of zero degrees. The fuselage and wing of the DLR-F4 was chosen to remain constant while the position of the engine beneath the wing was variable. Due to the variable engine location, the position and shape of the engine pylon was also variable. A half mesh was created in VSP along with the symmetry plane option in Flightstream to decrease solver time. The quadrilateral meshing tool was also used within Flightstream to decrease mesh size and increase solver stability. The engine location was defined by allowing the optimizer to chose the position of the pylon engine interface, pictured as Point 1 in Figure 2.7. The input variables needed for the optimizer to control the engine location and pylon shape as well as the limits for the search space can be seen in Table 4.1.

Variable	Minimum	Maximum	Starting Value
ΔX (m)	-.07	.07	-.01463
Y (m)	.18	.5	.28122
ΔZ (m)	.013	.04	.017
Percent Chord	.5	.95	.77688
Root Chord (m)	.08	.135	.13016

Table 4.1: Optimization Variables

The first three variables shown in Table 4.1 determine the engine and pylon location along the wing of the aircraft. The variable "Y" is the y coordinate of Point 1 on the engine pylon, seen in Figure 2.7. This measurement is taken as the perpendicular distance from the

plane of symmetry for the aircraft to Point 1 on the pylon. The variable ΔX corresponds to the distance along the x axis that Point 1 of the pylon is from the leading edge of the aircraft wing for a given Y value. A positive ΔX value corresponds to an engine configuration where Point 1 is in front of the leading edge of the wing while a negative ΔX value corresponds to an engine configuration where Point 1 is aft of the leading edge. The variable ΔZ determines the distance that Point 1 is below the leading edge of the wing along the z axis of the aircraft. Finally the variables "Root Chord" and Percent Chord help to define the rest of the shape of the pylon. Root Chord determines the length of Chord 1 of the pylon, as seen in Figure 2.7, and Percent Chord determines what percentage of the chord length of the local wing airfoil for a given Y location that the trailing edge of the pylon will interface with. The starting location for each variable was chosen at random by the SwarmOps PSO in ModelCenter. The coefficients of lift and drag obtained by Flightstream during the optimization process were used to determine the lift to drag ratio of each configuration. This ratio was then used as the optimizers objective function to determine the fitness of a given aircraft configuration. The results of this optimization run can be seen in the figures below.

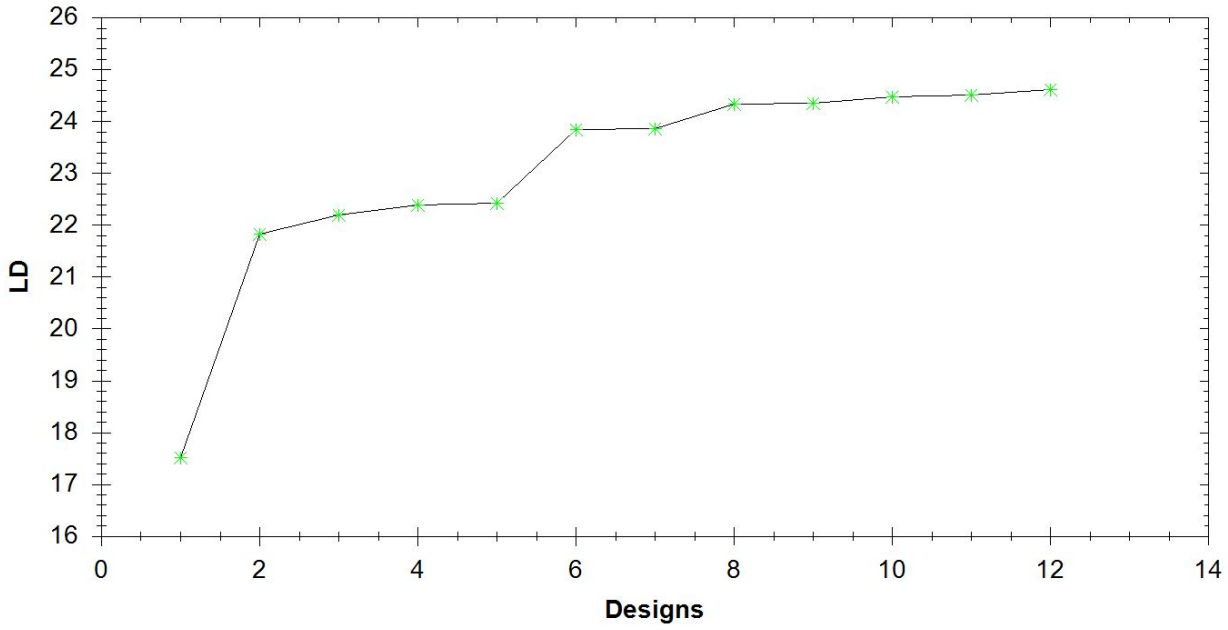


Figure 4.1: Optimization Convergence History

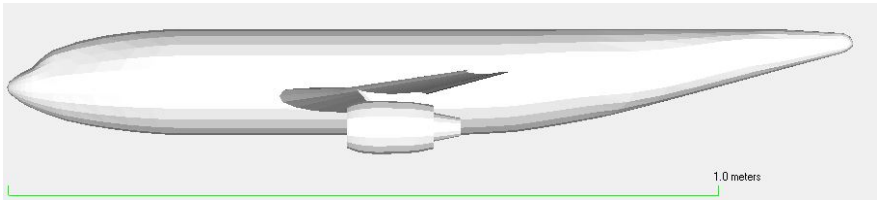
Variable	Starting Design	Best Design
ΔX (m)	-.01463	.06202
Y (m)	.28122	.5
ΔZ (m)	.017	.02474
Percent Chord	.77688	.95
Root Chord (m)	.13016	.12297
Lift over Drag	17.5049	24.608

Table 4.2: Optimization Design Values

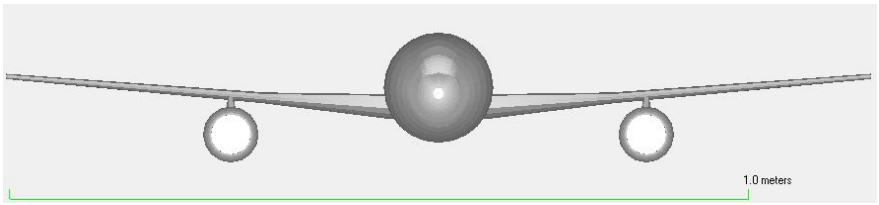
Figure 4.1 shows the lift to drag ratios found during the optimization process by Flightstream. Each point in the figure represents a new best design point found by the optimizer. As can be seen from Table 4.2, the lift to drag ratio was improved by around 40 percent of the starting value. The optimization run took less than seven "clock" hours to complete[1] on a laptop with an Intel(R) Core (TM) i7 CPU 1.73 GHz processor and 485 engine and pylon configurations were tested.

Figure 4.2 and Figure 4.3 show that the pylon was moved outboard on the wing to the maximum allowed value. This occurs because the pylon is acting like a winglet or fence to block wingtip vortex formation. The engine was also moved forward and lowered compared to the wing connection point. This was an expected outcome as it was theorized that higher lift to drag values would be attained by having the engine interfere as little as possible with the wing by moving the engine to the area of the wing with the smallest chord and moving the engine away from the lower surface. This decrease in flow interference is very noticeable in Figure 4.5 and 4.4. An interesting outcome of the optimization is that the percent chord variable was maximized in the best design. This increase to the upper chord length of the pylon may cause the pylon to act in a similar manner to a winglet and reduce the effect of the vortex formed from a differences in pressures to the left and right of the pylon. This vortex formation can be seen in Figure 4.6.

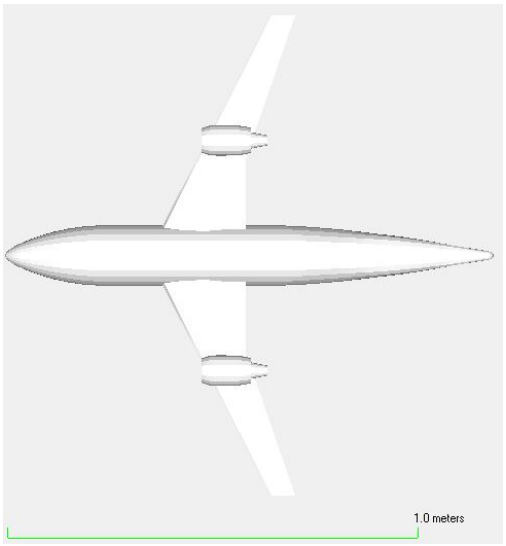
[1]The optimization was recorded as taking 13 hours and 15 minutes but the optimization was paused for up to 7 hours over the night.



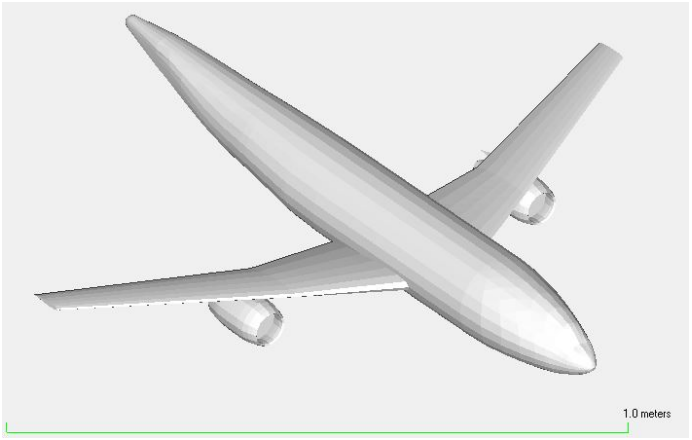
(a) Side View



(b) Front View

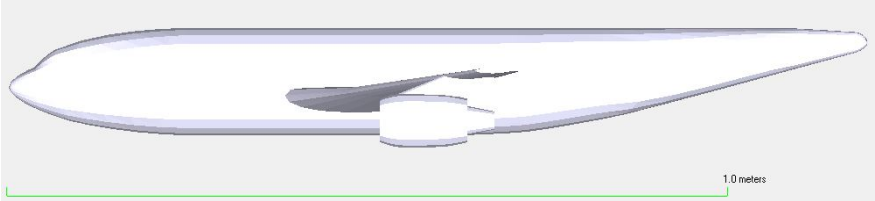


(c) Bottom View

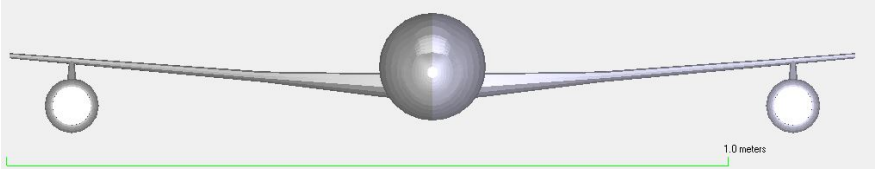


(d) Isometric View

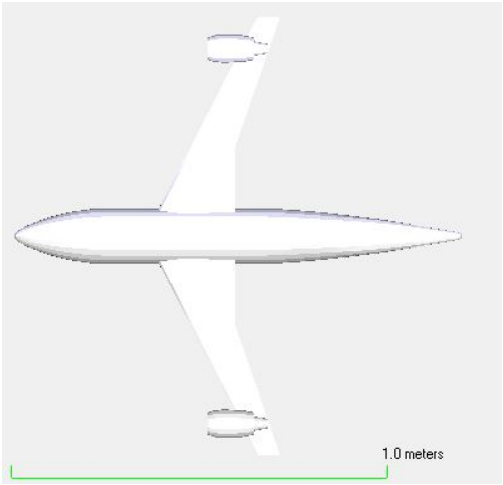
Figure 4.2: Starting Design



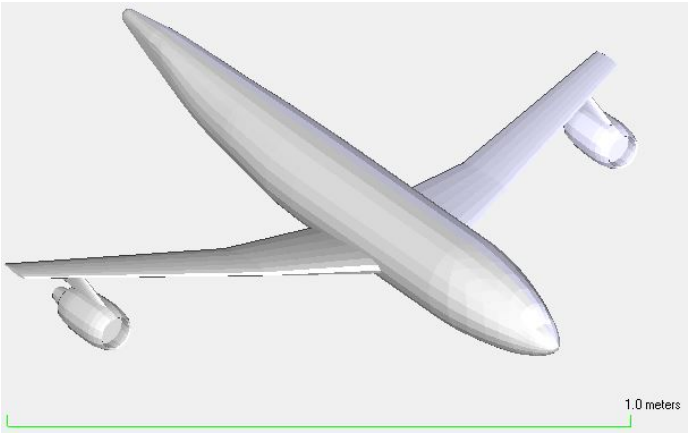
(a) Side View



(b) Front View

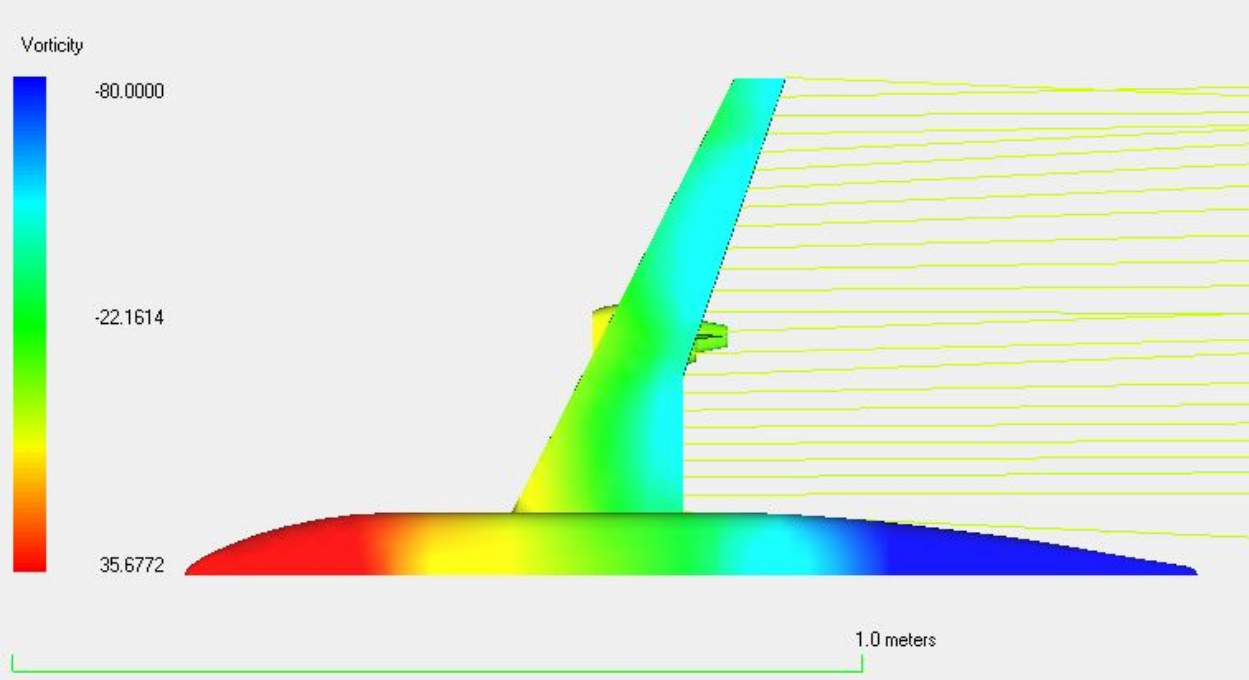


(c) Bottom View

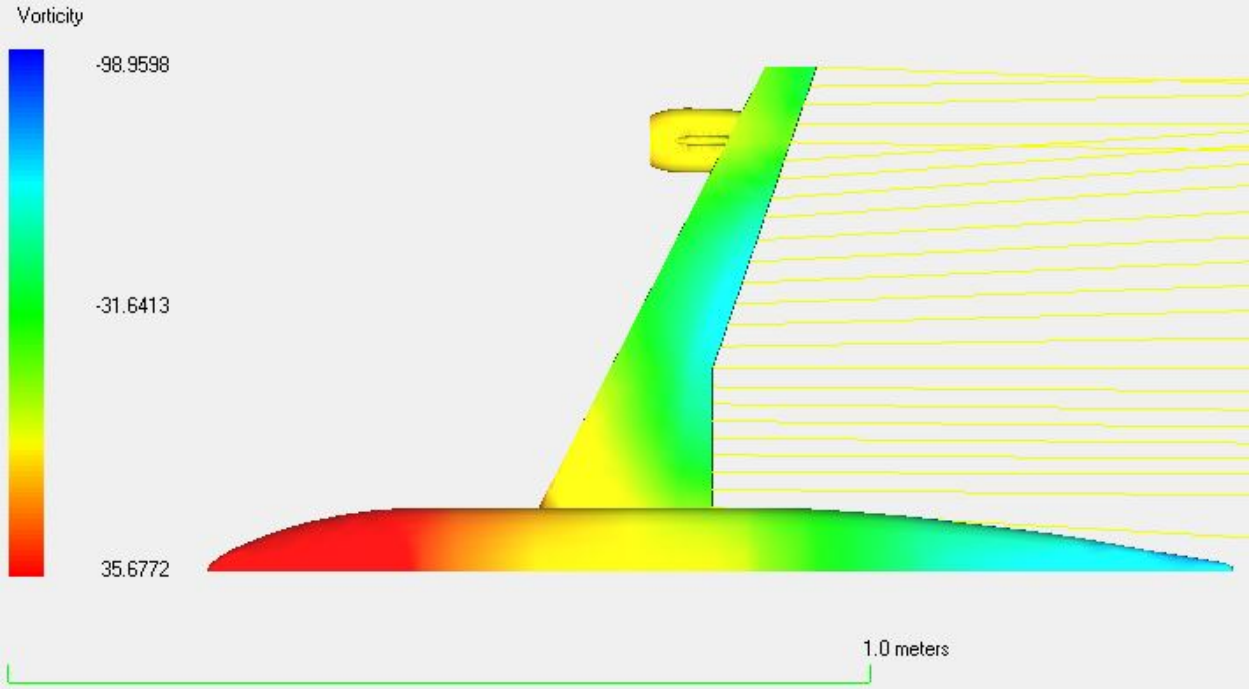


(d) Isometric View

Figure 4.3: Best Design

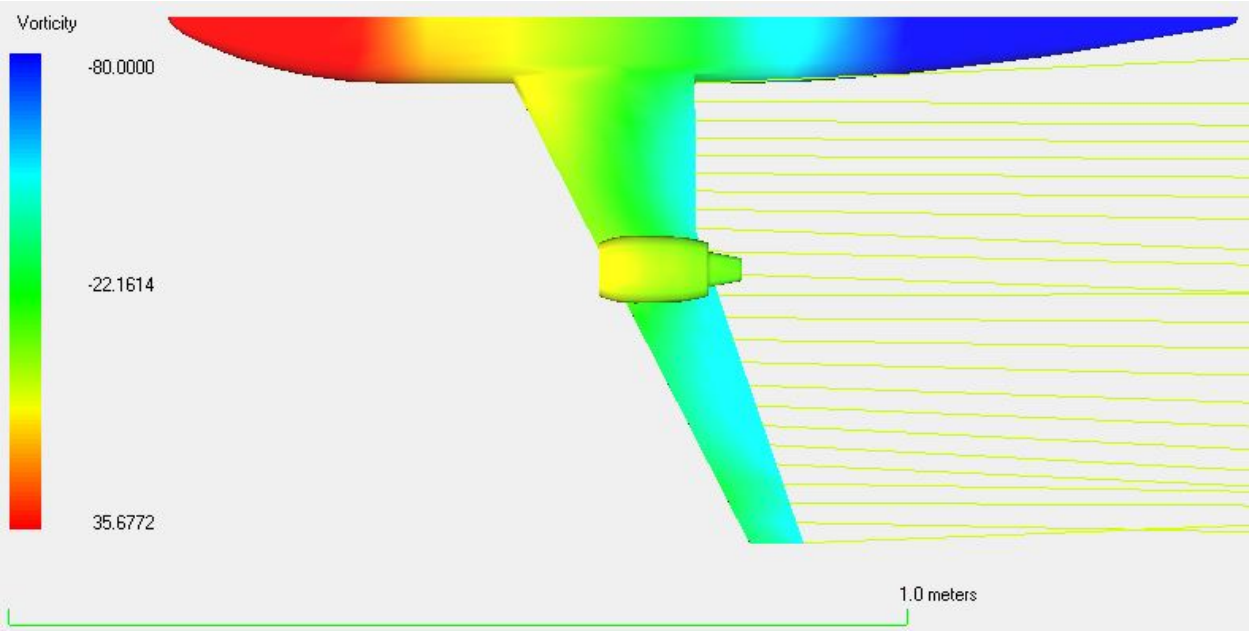


(a) Starting Design

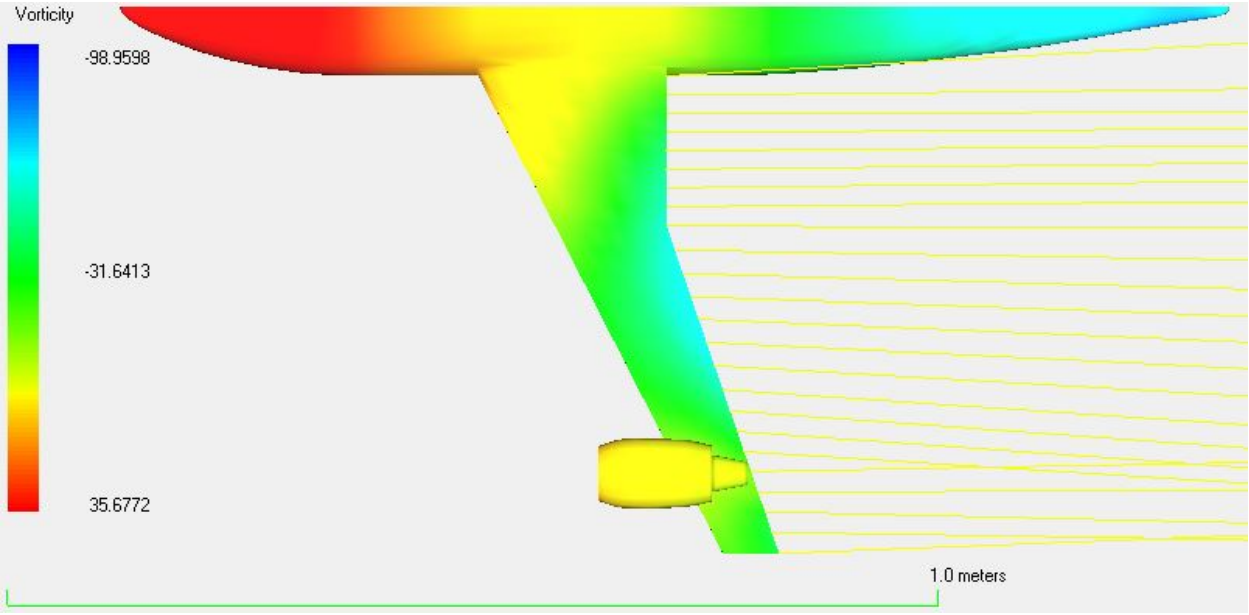


(b) Best Design

Figure 4.4: Vorticity Contour Plot: Top View

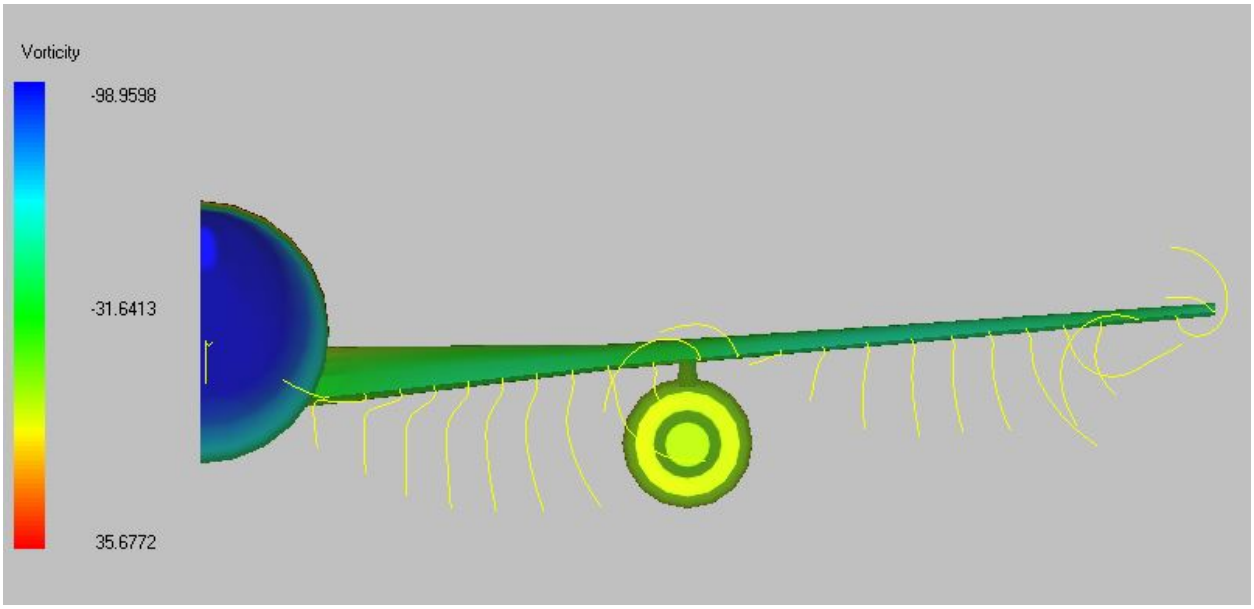


(a) Starting Design

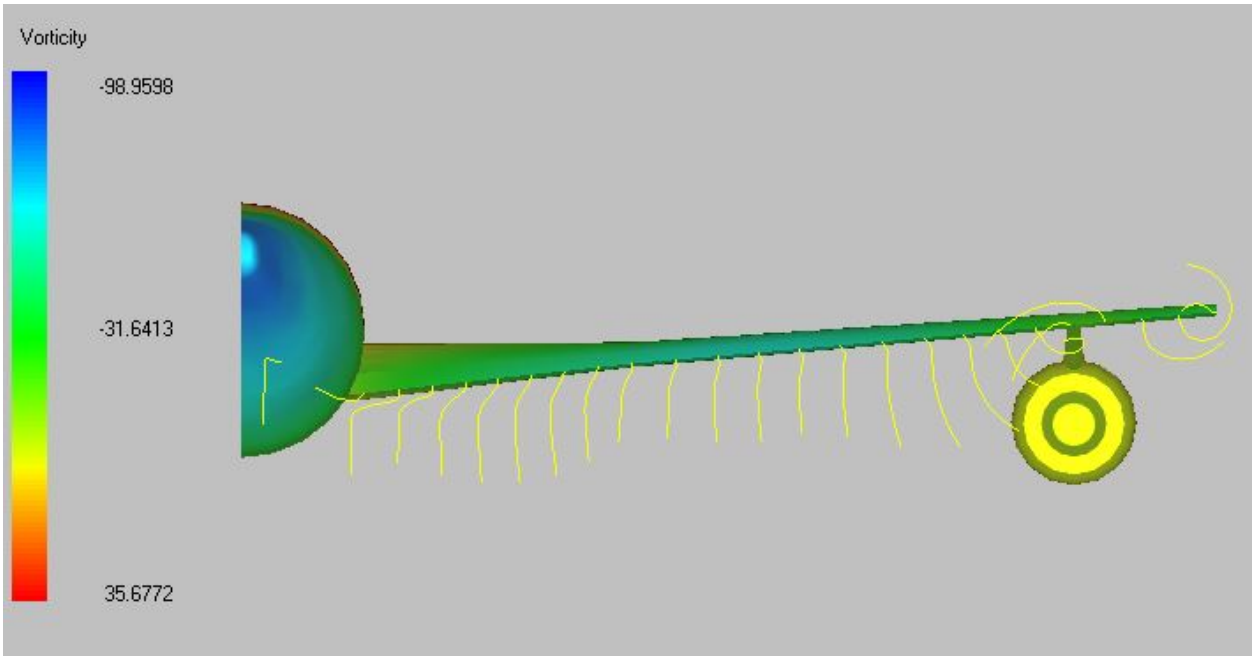


(b) Best Design

Figure 4.5: Vorticity Contour Plot: Bottom View



(a) Starting Design



(b) Best Design

Figure 4.6: Vorticity Contour Plot: Rear View

Chapter 5

Aerodynamic Optimization with Structural Penalty

The aerodynamic optimization from the previous chapter pushed the engine as far outboard as possible. In order to get a more realistic engine integration solution, structural and aerodynamic moment constraints must be considered. With an engine placement further from the plane of symmetry of the aircraft an increased moment is generated on the structure of the wing by the thrust and weight of the engine. Changes in lift and drag from the wing will also change the moment experienced by the wing. For the wing to be capable of sustaining an increased moment, it is necessary to increase the strength of the wing structure. This will inevitably lead to an increase to weight of the aircraft. The presence of two thrust producing engines also necessitate the presence of a vertical stabilizer large enough to overcome the yawing moment created in an engine out scenario. The further outboard the engine is placed on the wing, the larger a vertical stabilizer must be to counteract any potential yawing moments. By calculating the added weight of the wing structure and vertical stabilizer needed for an engine and pylon configuration, a structural penalty can be placed on the engine optimization problem.

5.0.7 Wing Weight Estimation

The structural support of the DLR-F4 wing was assumed to be composed of a single I beam made of aircraft steel, 5 Cr-Mo-V. This wing spar was said to have a fixed support where the wing attached at the fuselage. The three failure conditions examined for this wing spar are failure due to shear stress and failure due to compression or tension caused by a bending moment on the wing. The shear and bending stresses are created by the lift, drag, and weight forces generated by the wing and engine, as well as the thrust produced by the

engine. Before these calculations can be done, the lift and drag distribution along the wing must be found.

After an engine-pylon configuration is analyzed in Flightstream, span-wise lift and drag coefficients are written to an output file. These span-wise points are randomly placed along the span by Flightstream but the sum of these coefficients is equal to the total lift and drag coefficient for the aircraft. These data points can be used to form piece-wise cubic functions for the coefficients of lift and drag. These functions can then be used to estimate the coefficients of lift and drag along the wing with more even spacing which is more useful during wing loading analysis. However, it is important to remember that these values must be normalized by the ratio of data points to the number of interpolated points in order to ensure that they still produce the same amount of total lift and drag on the wing.

After the lift and drag distributions have been found for the wing, shear and moment calculations can begin. The lift and drag distributions can be approximated by a series of point loads on the wing spar. After the wing is split into a set number of elements of equal spacing, the lift and drag coefficients found from the span-wise lift and drag functions can be applied at individual points along the wing. The lift and drag coefficients at these span-wise locations can then be converted to lift and drag forces with the following equations where S is the reference area of the wing.

$$L = .5\rho V^2 S C_L \tag{5.1}$$

$$D = .5\rho V^2 S C_D \tag{5.2}$$

After lift and drag point forces are applied on the wing the thrust and weight of the engine can be added to the load distribution. This is done by splitting the thrust and weight of the engine between the two wing elements nearest to the span-wise engine location. The distribution of the thrust and weight of the engine between the two wing elements is determined by how close the span-wise engine location of the engine is to the two elements.

The closer the engine is to one of the two wing elements, the greater percentage of the engine forces it bears.

$$F_{element_n} = [1 - |y_{element_n} - y_{engine}|/span_{element}]F_{engine} \tag{5.3}$$

$$W_{element_n} = [1 - |y_{element_n} - y_{engine}|/span_{element}]W_{engine} \tag{5.4}$$

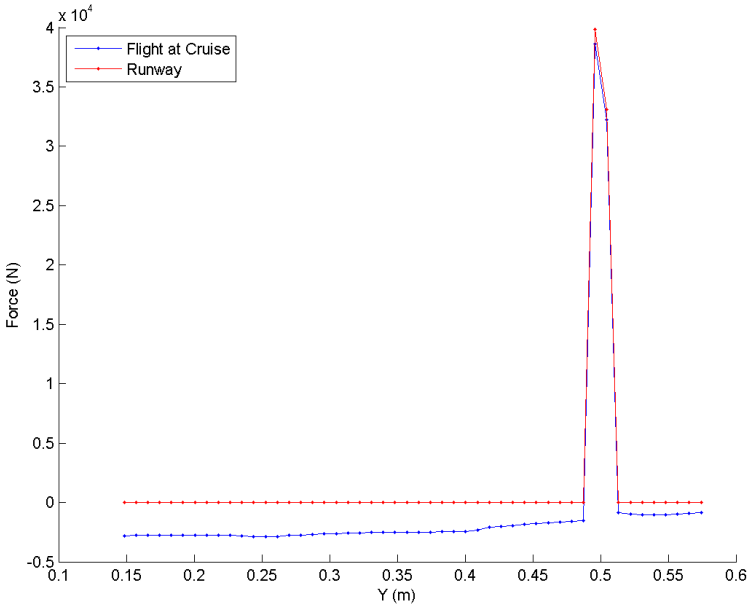
With the addition of the engine forces to the wing, all forces affecting the wing have been accounted for and converted to point loads with the exception of the support forces. Because the DLR-F4 has a cantilever wing, the sum of the forces felt and moments felt by the wing must be counteracted by the fixed support where the wing spar attaches to the fuselage. This supporting moment and force can be said to be acting at the most inboard location of the exposed wing. The load distribution for the optimal design from chapter 4 can be seen in the Figure 5.1.

After the total force distribution for the wing has been found, the shear and moment distribution for the wing can be determined. The forces of lift and weight act in a perpendicular direction to drag and thrust; therefore, they will produce perpendicular shear forces and bending moments for the wing spar. The shear force distribution for the wing spar can be found by integrating the force distribution across the wing. Due to the nature of cantilever beams, it is often easier to start calculations at the wing tip where the shear forces and bending moments are known to be zero.

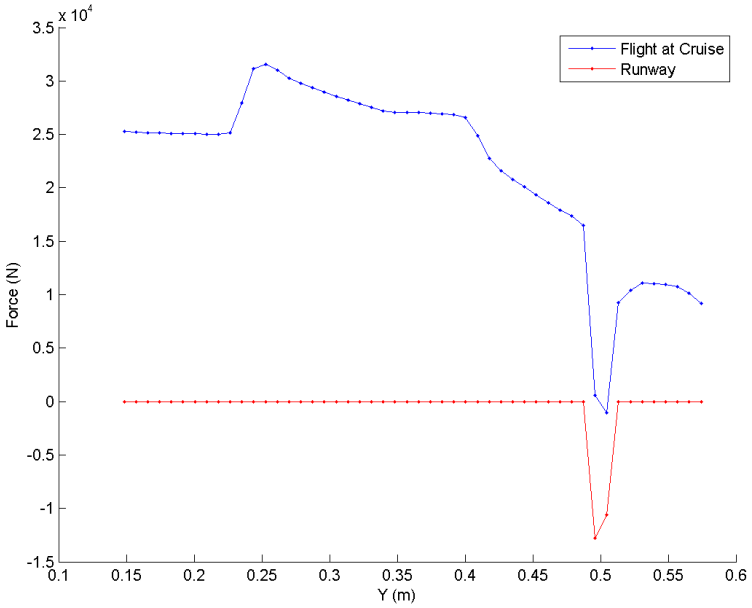
$$F_{shear}(y) = \int_0^y F(y)dy \tag{5.5}$$

The bending moment felt by the wing spar is the integral of the shear force distribution.

$$M(y) = \int_0^y F_{shear}(y)dy \tag{5.6}$$



(a) Span-Wise Forces in the X Direction

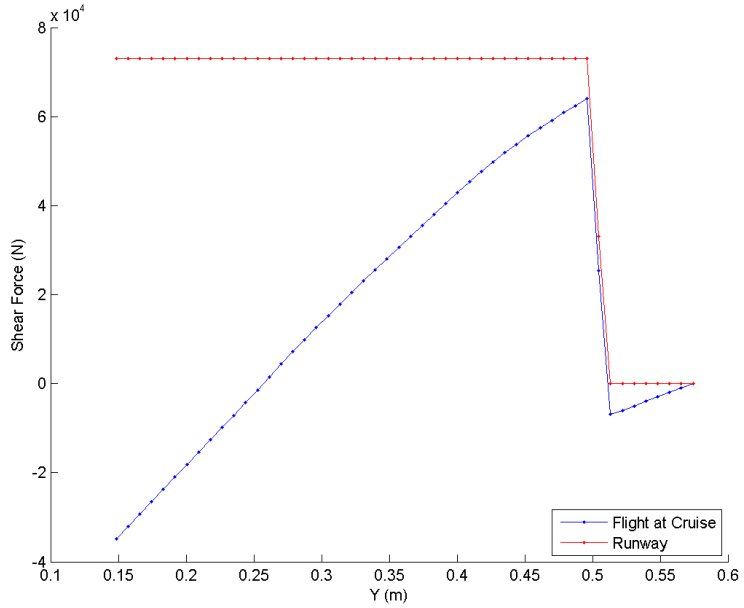


(b) Span-Wise Forces in the Z Direction

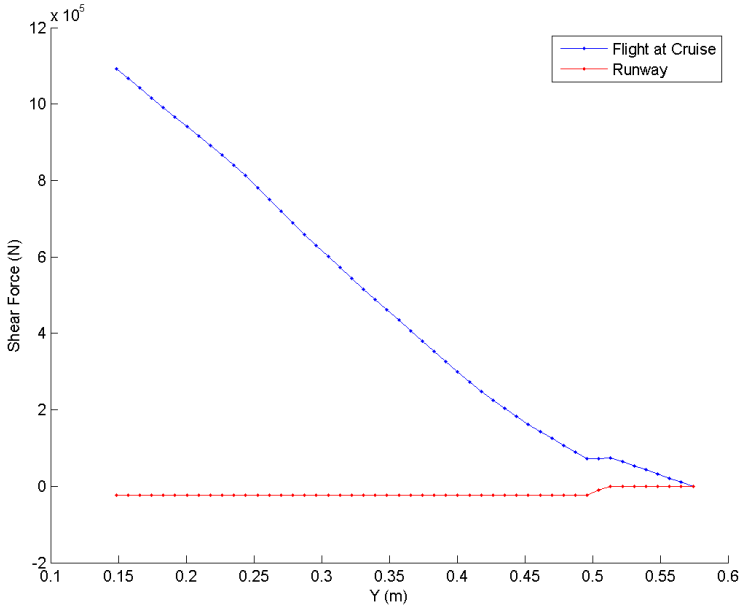
Figure 5.1: Span-Wise Force Distribution

These shear and moment calculations should be done for both the design case as well as the case where the aircraft is stationary with full thrust. The later case is important to consider during the design phase to ensure that wing can support the weight and thrust

of the engine on the runway without the forces of lift and drag to counteract them. The span-wise shear and bending moments for the optimal design from chapter 4 can be seen in Figures 5.2 and 5.3.

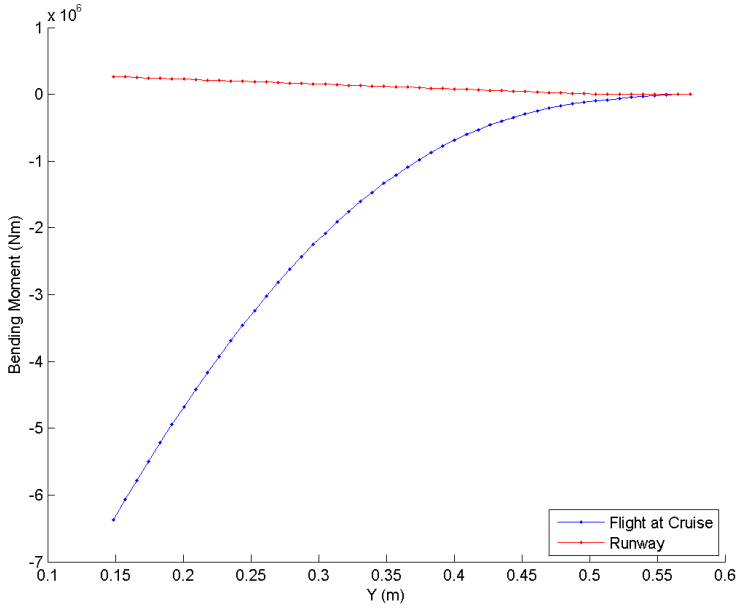


(a) Span-Wise Shear in the X Direction

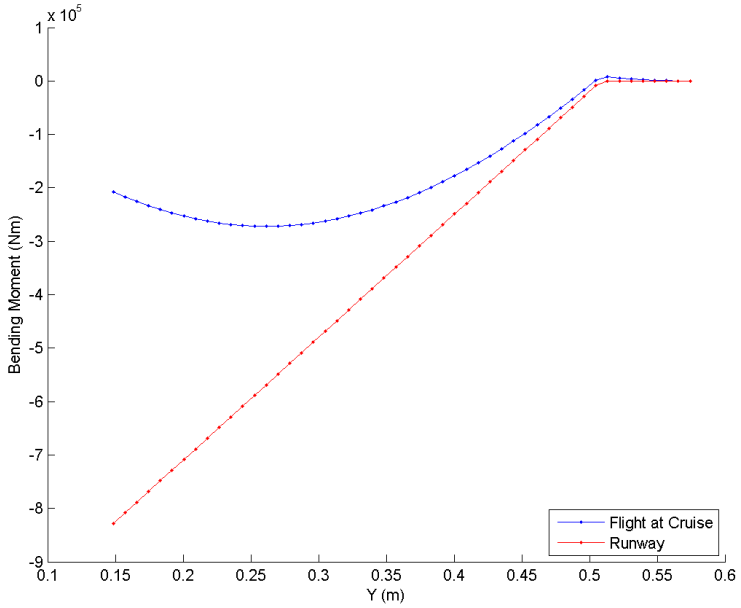


(b) Span-Wise Shear in the Z Direction

Figure 5.2: Span-Wise Shear Forces



(a) Span-Wise Bending Moment about the X Axis



(b) Span-Wise Bending Moment about the Z Axis

Figure 5.3: Span-Wise Bending Moments

The shear force and bending moment applied at any given span-wise location of the wing will produce a shear and bending stress felt by the wing spar cross section. These stresses are largely dependent on the dimensions of the cross section selected for the wing

spar. For this estimation it is assumed that the wing spar is an I beam with the following dimensions seen in Figure 5.4. The scale of this cross section will be variable in order to adjust to the local shear and bending stresses.

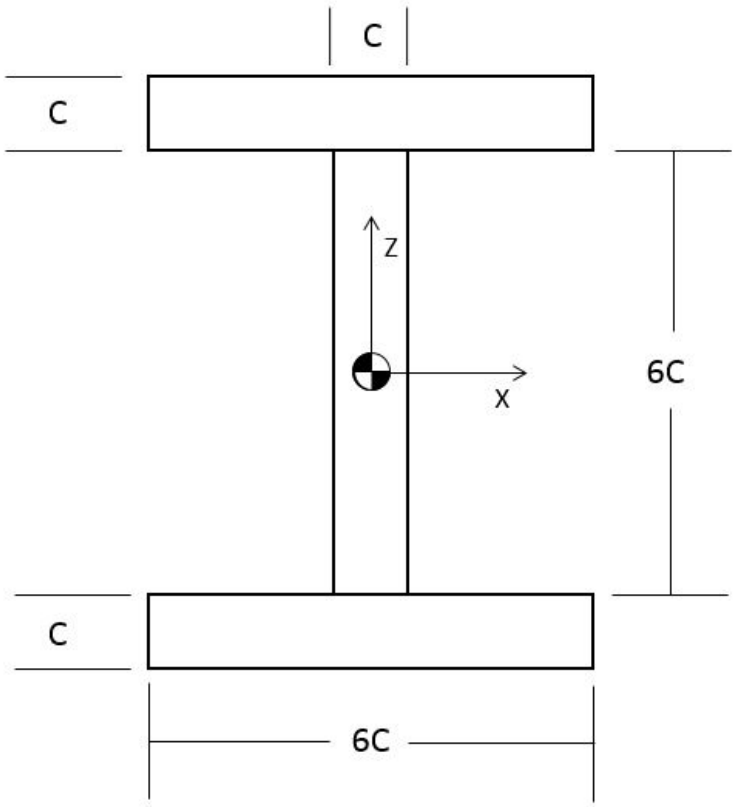


Figure 5.4: Wing Spar Cross Section

This cross section will have the following cross sectional area and moments of inertia about its centroid.

$$A = 18c^2 \tag{5.7}$$

$$I_x = 166c^4 \tag{5.8}$$

$$I_z = 36.5c^4 \tag{5.9}$$

The shear stress for a cross section of the wing spar is determined by the shear force applied at the location of the cross section and the cross sectional area of the beam.

$$F_{shear} = \sqrt{{F_{shear_x}}^2 + {F_{shear_z}}^2} \tag{5.10}$$

$$\tau = F_{shear}/A \tag{5.11}$$

The maximum bending stress felt by the cross section of the beam can be found using the bending moment, the perpendicular distance measured from the neutral axis of the cross section to the outer surface of the beam, and the moment of inertia about the neutral axis.

$$\sigma_x = M_x z/I_x \tag{5.12}$$

$$\sigma_z = M_z x/I_z \tag{5.13}$$

The bending stress created from lift and weight will be greatest at the top and bottom surfaces of the I beam whereas the bending stress created from thrust and drag will be greatest at farthest sides of the spar cross section. Each bending stress will cause compression on half of the cross section and tension on the other half of the cross section. These bending stresses felt by the cross section can be superimposed onto each other to determine the net bending stress created by the sum of the two bending moments. The end result is two opposite corners of the cross section will be compressed or stretched by both bending moments to create a maximum bending stress on the cross section of the spar.

$$\sigma_{max} = \sigma_x + \sigma_z \tag{5.14}$$

In order to determine the necessary cross section size needed by the beam to withstand the stresses calculated with equation 5.11 and equation 5.14, some information about the

material of the wing spar is needed. A wing spar made of the aircraft steel 5 Cr-Mo-V will have the following properties shown in Table 5.1.

Property	Value
ρ	.281(lb/in^3)
τ_{max}	260 kpsi
σ_{cmax}	220 kpsi
σ_{tmax}	240 kpsi

Table 5.1: Aircraft Steel (5 Cr-Mo-V) Material Properties [18]

The material properties in Table 5.1 can be used along with eqs. (5.7) to (5.14) to determine the cross section scaling necessary to prevent failure due to shear or bending. Because the aircraft steel is stronger in compression, the maximum tensile stress value in Table 5.1 will be used when determining the cross section scaling needed to prevent failure due to bending. After the cross section size needed to prevent failure due to shear or bending is found, the larger of the two cross sections is selected as the local cross section of the wing spar. The weight of the spar can be found by assuming a linear change in cross section area between each point. The following equation shows how the weight of one section between two points is found. This can be done for all the sections to find the total weight of the spar.

$$W_{y_1,y_2} = \rho g.5(A_1 + A_2)(y_2 - y_1) \tag{5.15}$$

If the weight of a beam section were to be converted to a point load, it would be applied at some location between the two endpoints of the beam section; however, all previously known load values are applied at the endpoints of these beam sections. Therefor; it would be more convenient to split the weight of each beam section into two smaller loads which are applied at the previously established wing element locations. This can be done by integrating equation 5.15 to find the moment produced by the weight of the spar section about one of

its endpoints.

$$M_{y_1} = (1/6)(A_2 - A_1)(y_2 - y_1)^2 + .5A_1(y_2 - y_1)^2 \tag{5.16}$$

This total moment produced by the weight can then be used to determine the force applied at one of the end points of the beam section to produce that moment.

$$F_2 = M_{y_1}/(y_2 - y_1) \tag{5.17}$$

This ensures that point load at the end of the spar section replicates the moment produced by the weight; however it is still necessary to apply another point load at the opposing end of the section to ensure that the two point loads sum to the total weight of the section.

$$F_1 = W_{y_1,y_2} - F_2 \tag{5.18}$$

This can be done for the total length of the wing span to convert the weight of the wing spar to point loads applied at the previously defined span-wise locations. The forces applied at the end points of adjacent spar locations can be summed to find the total point load applied at that spar cross section.

The weight distribution in the form of point loads can now be added to the wing forces in the Z direction so that shear, moment and area calculations can be redone. This is necessary because the previously calculated spar cross section areas did not take the weight of the wing into account. This process is then repeated until the weight of the wing is within .1 Newton of the previously found wing weight. This process can be seen in the following figure.

After the wing spar weight is found, the weight of the wing can be found by assuming that the wing spar accounts for 80 percent of the weight of the wing.

$$W_{wing} = W_{spar}/.8 \tag{5.19}$$

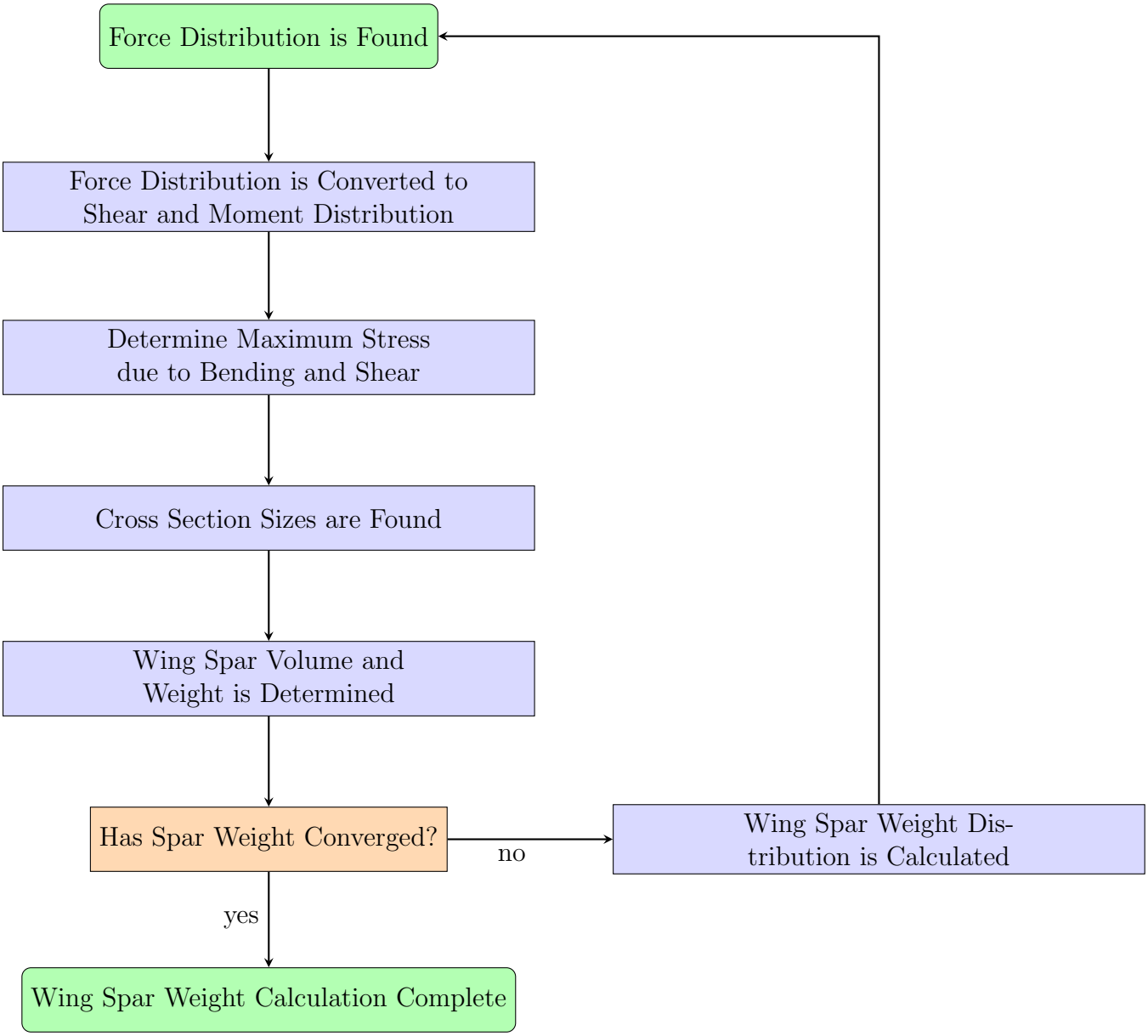


Figure 5.5: Wing Spar Weight Calculation

5.0.8 Vertical Stabilizer Sizing

Although the DLR-F4 does not have a vertical stabilizer, one would be needed to offset the yawing moment generated by the thrust produced by one of the added engines in the event that the other engine failed. The yawing moment produced by one engine is directly proportional to the magnitude of the thrust and the engine's distance from the plane of

symmetry.

$$M_z = Fy_{engine} \tag{5.20}$$

If a constant thrust is assumed, the further outboard the engine is placed on the wing, the larger the yawing moment will be. To counteract this yawing moment the rudder will have to be capable of producing a yawing moment that is equal and opposite to the yawing moment produced by the engine. The process of determining the yawing moment produced by a rudder deflection is well documented in Reference [19].

$$M_z = -\delta_R q S b C_{n\delta R} \tag{5.21}$$

$$q = .5\rho V_{MC}^2 \tag{5.22}$$

The design velocity for the one engine out condition is 80 percent of the stall speed for the aircraft and the rudder control derivative can be found with the following equations [19].

$$C_{n\delta R} = -C_{L_{\alpha V}} V_V \eta_v \tau_R \frac{b_R}{b_V} \tag{5.23}$$

$$V_V = \frac{l_v S_v}{bS} \tag{5.24}$$

For a given vertical stabilizer configuration and a prescribed rudder deflection angle for the one engine out case, the reference area of the vertical stabilizer can be found by rearranging eqs. 5.20 to 5.24.

$$S_V = \frac{Fy_{engine}}{\delta_R q C_{L_{\alpha V}} \eta_R \tau_R \frac{b_R}{b_V} l_V} \tag{5.25}$$

From equation 5.25 it can be seen that given a constant vertical stabilizer configuration and deflection angle, the required vertical stabilizer size to counteract the yawing moment due to thrust will scale linearly with the outboard location of the engine. Reference [18] outlines how this vertical stabilizer area can then be used to find the weight of the vertical stabilizer

in newtons using empirical relationships.

$$W_V = 27 S_V g \tag{5.26}$$

The drag created by a given vertical stabilizer configuration can be found by first modeling it in VSP and analyzing the mesh in Flightstream. Flightstream can give the coefficient of drag of the vertical stabilizer using the reference area of the baseline vertical stabilizer. After this coefficient of drag is found for the cruise condition of the aircraft, the drag produced by a similar vertical stabilizers of varying sizes can be found with the following equation.

$$D_V = q S_V C_{D_V} \tag{5.27}$$

5.0.9 Optimization with Wing Spar and Vertical Stabilizer Sizing Penalty

The previous engine integration optimization focused exclusively on maximizing the ratio of lift to drag for the DLR-F4. The result was engines being placed as far outboard as the optimizer would allow. This is not a very realistic result when other factors such as structural weight and tail sizing are considered. To address these issues, the objective function must be altered to reflect the disadvantages of increased weight and drag due to increases in wing spar and vertical stabilizer sizes.

$$Fitness = \frac{L - W_{fuselage} - 2W_{wing} - W_V}{D + D_V} \tag{5.28}$$

Much like the weight of the vertical stabilizer, the fuselage weight can be estimated with an empirical equation [18].

$$W_{fuselage} = 24 S_{fuselage} g \tag{5.29}$$

Equations 5.26 and 5.29 seem to be more accurate for more realistic transport aircraft sizes, so for the purpose of this optimization the DLR-F4 geometry will be scaled up to have

the same length as an Airbus A320 while using the same flow conditions as the previous optimization case.

The vertical stabilizer added to the DLR-F4, pictured in Figure 5.6, will scale with its root chord length in order to produce a large enough moment to off set the yawing moment created by a one engine out scenario. The vertical stabilizer will be required to counteracting the moment produced by the thrust of one engine with a 20 degree rudder deflection.

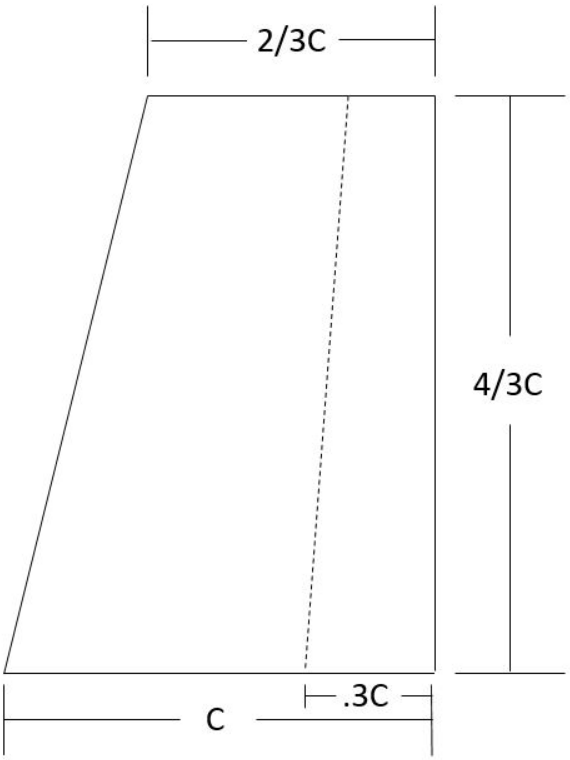


Figure 5.6: Vertical Stabilizer Side View

The DLR-F4 will have an assumed stall speed of 67 m/s and finally the vertical stabilizer will be assumed to have a constant moment arm relative to the center of gravity of the aircraft, regardless of vertical stabilizer sizing. The moment arm of the vertical stabilizer along with other constants needed for the tail sizing calculation can be seen in Table 5.3a.

The coefficient of drag for the vertical stabilizer was found by creating the geometry within VSP and analyzing it within Flightstream using the same flow conditions as the DLR-F4 cruise. The reference area used for the drag coefficient evaluation was that of the vertical

Constant	Value
V_s	67 m/s
ρ	1.22000 kg/m^3
δ_R	20 deg
S	.1454 m^2
b	1.17129 m
η_V	.97
$\frac{b_R}{b_V}$	1
l_V	.6 m
$C_{L_{\alpha V}}$	4.5 $/rad$
τ_R	.52 [19]
C_{D_V}	.0063807

Table 5.2: Tail Sizing Constant Values

(a) All Lengths and Areas Listed are Pre Scale Up

stabilizer tested in Flightstream. This coefficient of drag was assumed to be constant for all vertical stabilizers regardless of size which meant that the drag produced by the vertical stabilizer scaled linearly with vertical stabilizer area.

With the vertical stabilizer scaling fully defined, ModelCenter can once again be used to find an optimal engine-pylon configuration for the DLR-F4. The flow conditions and search space, and starting geometry of this optimization are the same as the previous case; however, the geometry is scaled by a factor of 31.518 in order to have a similar length to an A320. The engine thrust is found in the same way as outlined in 2.0.2, but the weight of the engine is assumed to be 5,250 pounds which is equal to the weight of the engine used on an Airbus A320, CFM56-5B [20]. Finally the factor of safety used for wing spar cross section sizing was set to 1.5. The results of this optimization can be seen in the figure and table below.

Unlike the previous optimization which attempted to maximize the ratio of lift to drag, the fitness function of this optimization drove the engines to the maximum allowable inboard location. Although the lift to drag ratio was higher at the wing tips, the added weight of the wing spar and vertical stabilizer more than outweighed the aerodynamic benefits. As can be seen by equations 5.25 and 5.26, the area and weight of the vertical stabilizer will scale

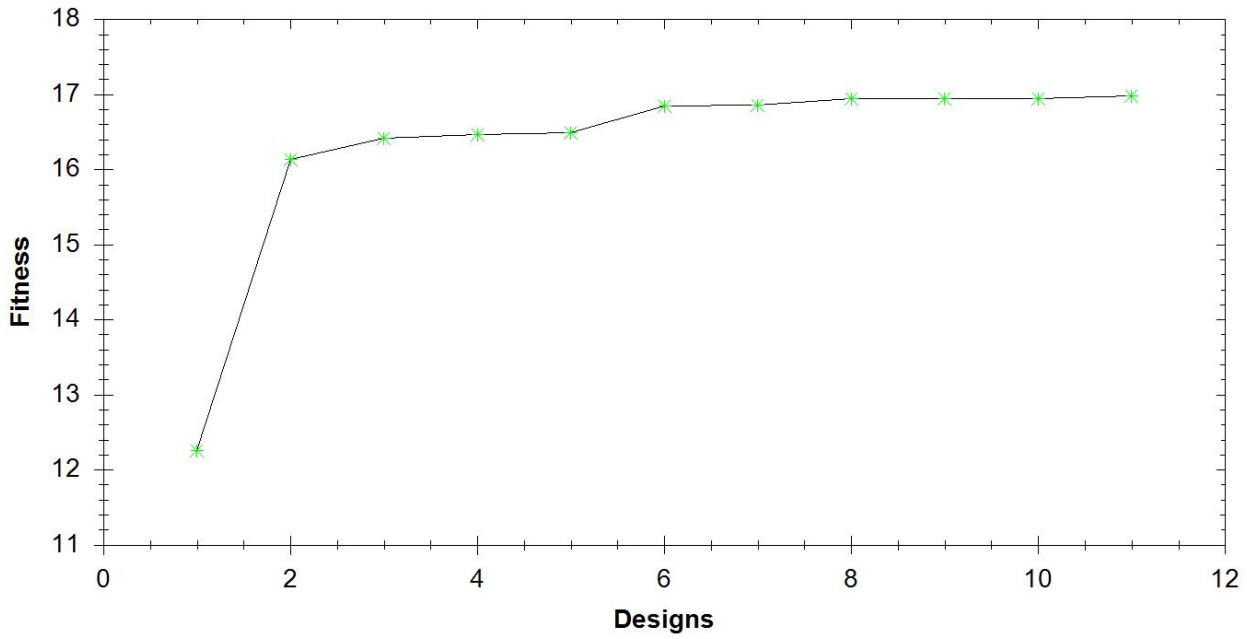


Figure 5.7: Optimization Convergence History

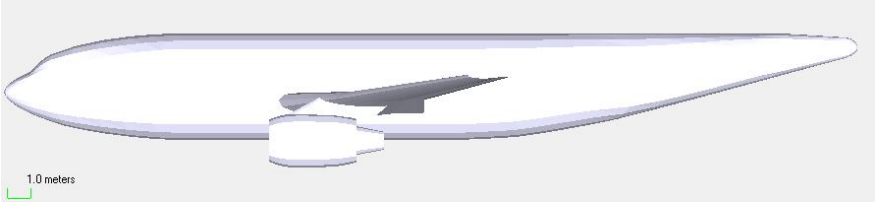
Variable	Starting Design	Best Design
ΔX (m)	-.01463	.0429
Y (m)	.28122	.18
ΔZ (m)	.017	.0228
Percent Chord	.77688	.64943
Root Chord (m)	.13016	.12066
Fitness	12.2564	16.978

Table 5.4: Optimization Design Values

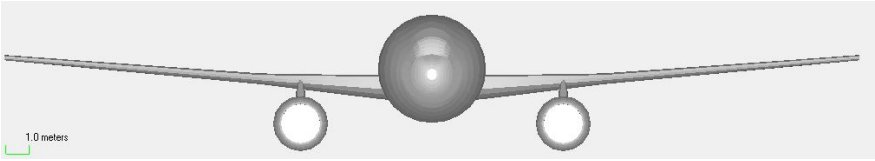
(a) Input Distances are not Scaled

linearly with the span-wise distance. The full range of vertical stabilizers produced by the optimizer can be seen in the figure below.

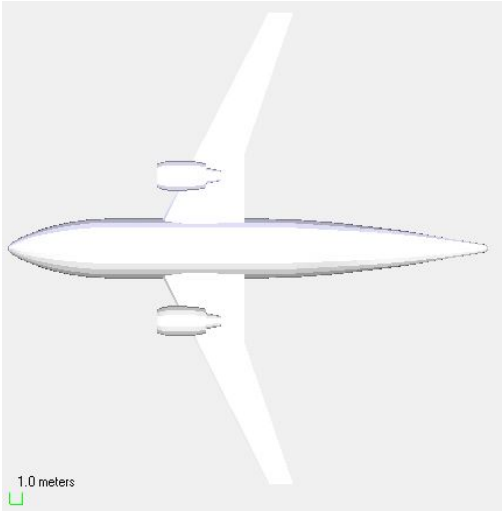
The optimizer also once again moved the engine forward and to a farther distance below the wing to maximize the lift to drag ratio generated at the span-wise engine location. Unlike the previous optimization, the uppermost pylon chord length was not driven to the maximum allowable value. This may be due to the lack of a nearby wing tip vortex. When the engine is placed near the wing tip it can act as a winglet to reduce the wing tip vortex but when the engine is placed more inboard a increase to the upper pylon chord length only serves to further disrupt the flow. With the engine configuration of the optimal design, the



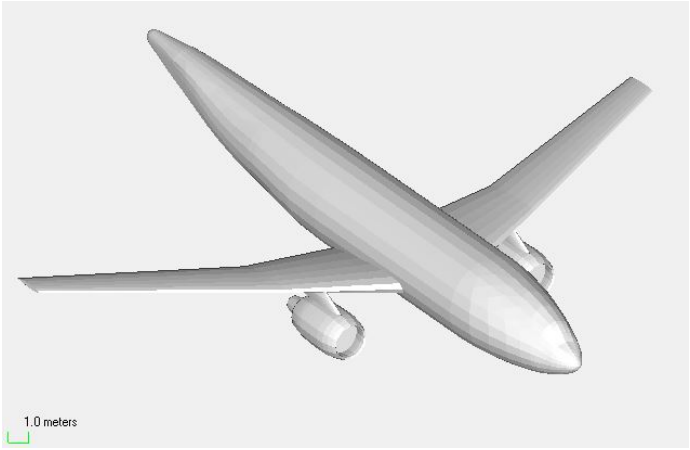
(a) Side View



(b) Front View



(c) Bottom View



(d) Isometric View

Figure 5.8: Best Design

flow appears to be minimally affected, producing a similar wing vorticity distribution as the previous optimal design as can be seen in Figures 4.5b,4.4b and 5.11.

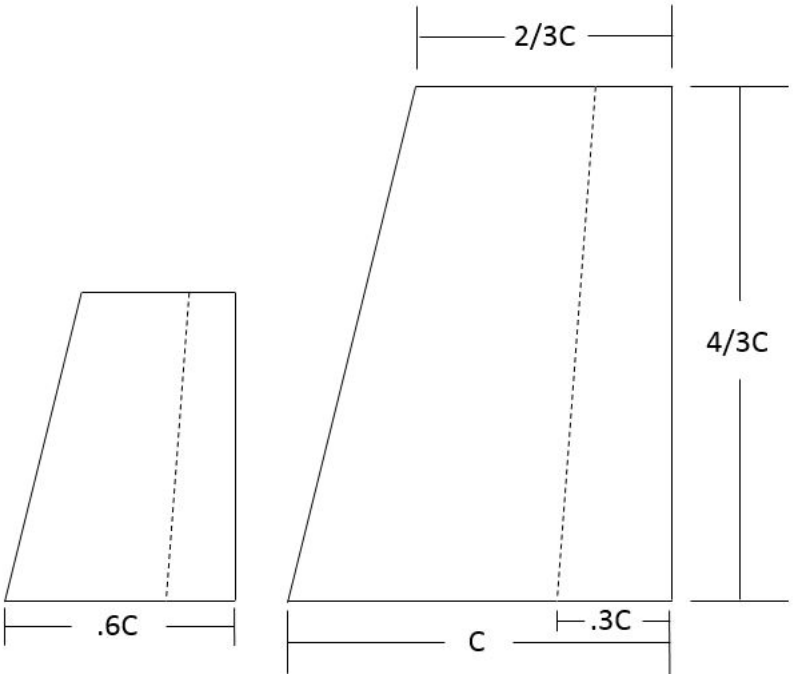
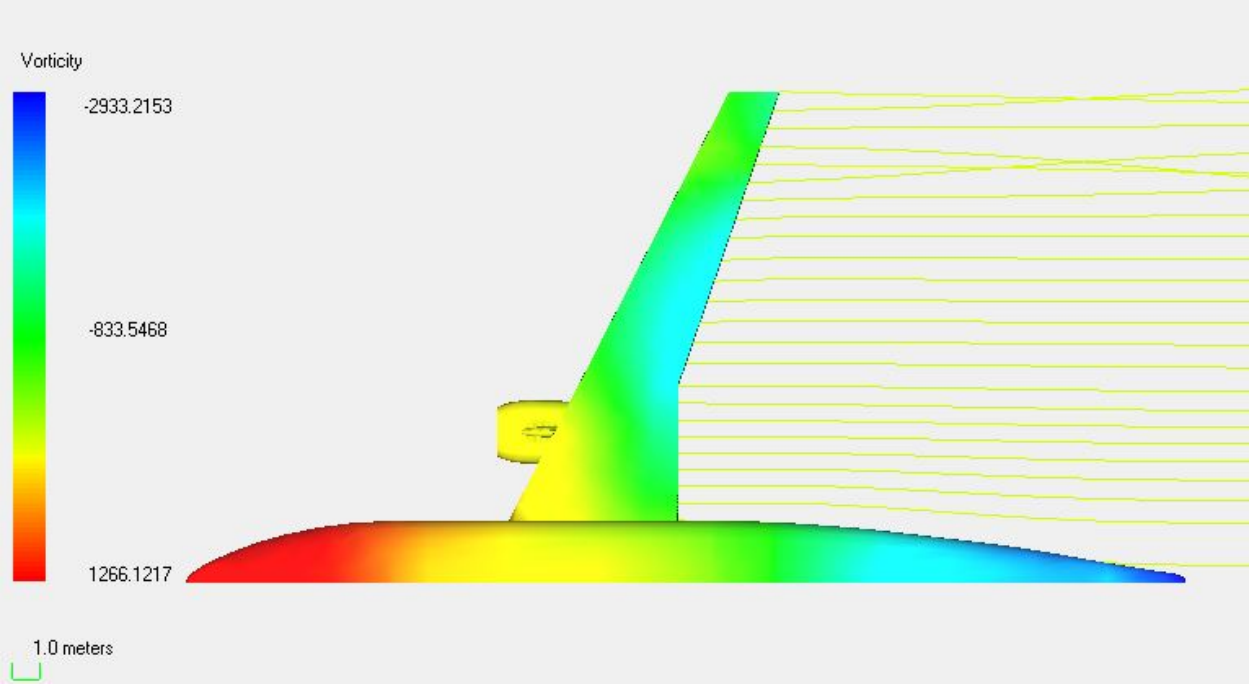
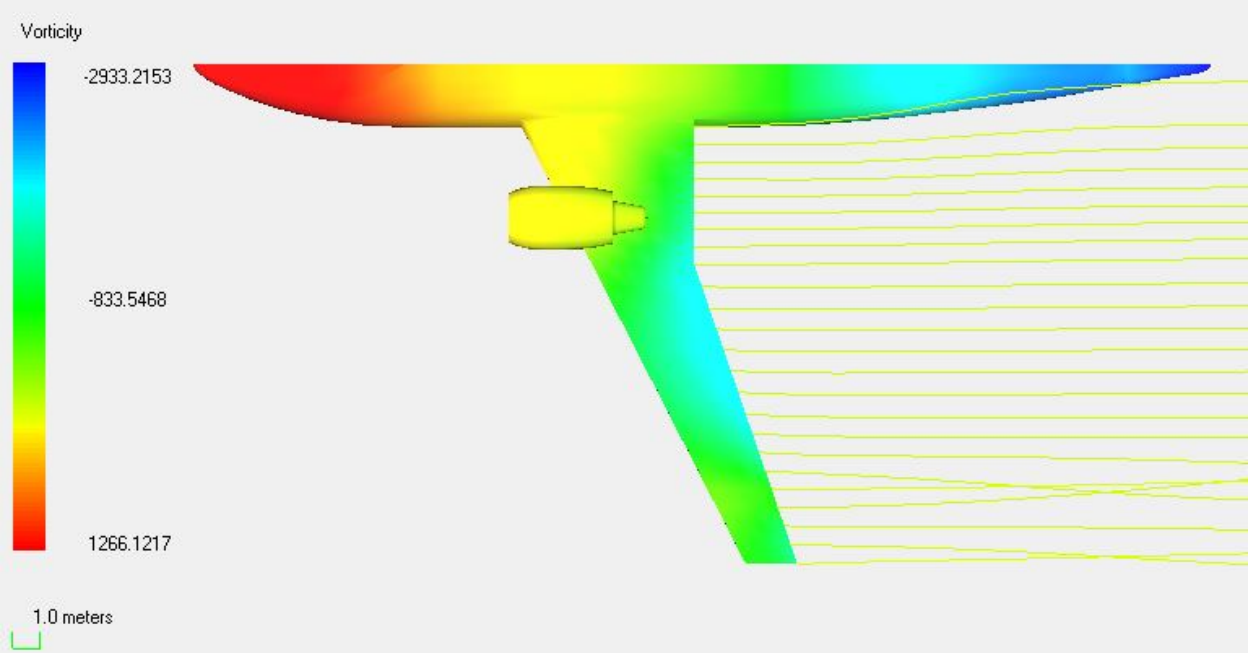


Figure 5.9: Range of Pylon Sizes for Optimization

(a) c=13.3169 m



(a) Top View



(b) Bottom View

Figure 5.11: Vorticity Contour Plots of Best Design

Chapter 6

Conclusions and Future Work

The process and results of two engine integration optimizations of the DLR-F4 geometry using an unstructured mesh vorticity based panel solver were presented. Flightstream and ModelCenter were able to identify the optimal engine-pylon configuration for both a purely aerodynamic optimization and an aerodynamic optimization with more realistic structural constraints. For the purely aerodynamic optimization, the optimal engine placement was found to be near the wingtip which reduced the effect of wing tip vortices. However, this benefit to the ratio of lift over drag was found to be overshadowed by the added weight and drag produced by an ever increasingly large vertical stabilizer needed to offset potential yawing moments produced by the engines. These optimizations showed that Flightstream can provide an adequate alternative to Navier-Stokes based CFD solvers in finding an optimized preliminary design solution. The decrease to run time resulting from the use of unstructured surface meshes and a vorticity based solver is of immense value to the optimization process. An engine optimization run that may have taken weeks or longer using CFD was preformed in a few hours.

In the future an effort should be made to parallelize this optimization process to further decrease the optimization run time. In addition to improving the optimizer, the fidelity of the vertical stabilizer model could be improved by including the vertical stabilizer geometry to the aircraft model in VSP to measure the yawing moment produced by a rudder deflection as opposed to using empirical relationships. Also due to the modular nature of optimization with ModelCenter, greater complexities could be added to the model with the addition of more outside programs. The addition of an engine analysis program such as NPSS would allow for the ability to alter the internal engine configuration instead of simply the location

of the engine. The estimation of weight could also be more accurately measured through the integration of a finite element structural tool such as Nastran. Nastran could allow for the optimization of more elaborate wing spar cross sections and the evaluation of more failure conditions. This program could also be used to improve the pylon model by changing the shape and weight of the pylon to account differing structural demands. Although the addition of these propulsion and structural analysis tools would increase optimization run time, the presence of an unstructured mesh vorticity flow solver can reduce the run time to a manageable level.

Bibliography

[1] Anderson, J. D., *Fundamentals of Aerodynamics 5th ed.*, McGraw/Hill, NY, 2011.

[2] Bertin, J. J., and Cummings, R. M., *Aerodynamics for Engineers 5th ed.*, Pearson Prentice-Hall, NJ, 2009.

[3] Baker, W. M., Elliott, R. D., and Miranda, L. R., "A Generalized Vortex Lattice Method for Subsonic and Supersonic Flow Applications," NASA CR-2865, Dec. 1977.

[4] Martin, G. L., "Paneling Techniques for use with the VORLAX Computer Program," NASA CR-145364, April 1978.

[5] Ting, E., Reynolds, K., Nguyen, N. and Totah, J., "Aerodynamic Analysis of the Truss-Braced Wing Aircraft Using Vortex-Lattice Superposition Approach," *32nd AIAA Applied Aerodynamics Conference,* Atlanta, GA, 2014.

[6] Erickson, L. L., "Panel Methods - An Introduction," NASA Technical Paper 2995, 1990.

[7] Derbyshire, T., and Sidwell, K., W., "PAN AIR Summary Document (Version 1.0)," NASA CR-3250, April 1982.

[8] Maskew, B., "Program VSAERO Theory Document, A Computer Program for Calculating Nonlinear Aerodynamic Characteristics of Arbitrary Configurations," NASA CR-11945, Sep. 1987.

[9] Drela, M., and Youngren, H., "AVL 3.30 User Primer," MIT, August 2010. `http://web.mit.edu/drela/Public/web/avl/avl_doc.txt`. Accessed 2/15/15.

[10] Melin, T., "A Vortex Lattice MATLAB Implementaion for Linear Aerodynamic Wing Applications," Masters Thesis, Department of Aeronautics, Royal Institue of Technology, 2000.

[11] Ahuja, V., "Aerodynamic Loads over Arbitrary Bodies by Method of Integrated Circulation," Ph. D. Dissertation, Department of Aerospace Engineering, Auburn University, Auburn, AL, 2013.

[12] Frink, N. T., "Notes on Preparing the Geometry," NASA, November 2002. `http://aaac.larc.nasa.gov/tsab/cfdlarc/aiaa-dpw/Workshop1/files/geometry.html`

[13] Redeker, G., "A Selection of Experimental Test Cases for the Validation of CFD Codes," AGARD-AR-303, 1994.

[14] Meyer, P., "DLR-F4," VSP Hangar. http://hangar.openvsp.org/vspfiles/196

[15] Andrus, I., "Comparative Analysis of a High Bypass Turbofan Using a Pulsed Detonation Combustor," Thesis, Department of the Airforce Air University, Air Force Institute of Technology, 2007, pp. 54.

[16] Pendersen, M., "SwarmOps for C♯, Numeric and Heuristic Optimization Source-Code Library for C♯ The Manual," 2011.

[17] Kennedy, J. "The Particle Swarm: Social Adaptation of Knowledge," *IEEE International Conference on Evolutionary Computation*, Indianapolis, IN 1997.

[18] Raymer,D. *Aircraft Design: A Conceptual Approach 4th ed.* American Institute of Aeronautics and Astronautics, Inc., VA, 2006.

[19] Sadraey, M. *Aircraft Design: A Systems Engineering Approach* Wiley Publications 2012

[20] "CFM56-5B Turbofan Engine," CFM Aero Engines. http://www.cfmaeroengines.com/engines/cfm56-5b